



UNIVERSIDADE FEDERAL DE SANTA CATARINA
CURSO DE SISTEMAS DE INFORMAÇÃO
DISCIPLINA: INE 5632 – PROJETOS II

Persistência de Dados em Java: Um Estudo Aplicado ao Sistema Informatizado para Gerenciamento de Ouvidorias

RONY REINEHR BRAND
FLORIANÓPOLIS, FEVEREIRO DE 2006.

UNIVERSIDADE FEDERAL DE SANTA CATARINA
CURSO DE SISTEMAS DE INFORMAÇÃO
DISCIPLINA: INE 5632 – PROJETOS II

Persistência de Dados em Java: Um Estudo Aplicado ao Sistema Informatizado para Gerenciamento de Ouvidorias

Orientador: Leandro José Komosinski

Banca Examinadora:

Maria Marta Leite

Ronaldo dos Santos Mello

RONY REINEHR BRAND
FLORIANÓPOLIS, FEVEREIRO DE 2006.

SUMÁRIO

1	Introdução	7
1.1	Ouvidoria	8
1.2	Tema	10
1.3	Definição do Problema	10
1.3.1	Camada de Persistência	11
1.4	Escopo de Estudo	11
1.5	Objetivo Geral	12
1.6	Objetivos Específicos	12
1.7	Motivação	12
1.8	A Empresa: OMD Soluções para Ouvidorias	13
1.8.1	A pré-incubadora: GeNESS	13
2	Sistema Informatizado para Gerenciamento de Ouvidorias	14
2.1	Outras Soluções Existentes em Ouvidoria	15
3	JDO	17
3.1	Um breve histórico	17
3.2	A especificação JDO	18
3.3	Mapeamentos em JDO	18
3.3.1	Primeiro Exemplo	19
3.3.2	Segundo Exemplo	20
3.3.3	Terceiro Exemplo	21
3.3.4	Quarto Exemplo	22
3.4	Enhancer (Enxertador) JDO	23
3.5	JDOQL – JDO Query Language	23
3.6	JPOX - Java Persistent Objects JDO	24
3.7	Spring	24
3.8	JUnit	25
3.9	Exemplo Prático: JPOX, Spring e JUnit	25
3.9.1	DAOJdo	25
3.9.2	BaseDAOJdo	28
3.9.3	IdentidadeNomeDAOJdo	28
3.9.4	AssuntoDAOJdo	29
3.9.5	ApplicationContext.xml	29
3.9.6	BaseDAOTestCase	31
3.9.7	AssuntoDAOJdoTest	32

4	Desenvolvimento da Camada de Persistência	34
4.1	Levantamento dos Requisitos de Persistência.....	34
4.2	Diagrama de Classes de Persistência.....	35
4.2.1	Diagrama de classes resumido.....	36
4.2.2	Classes detalhadas: Domínio do problema	36
4.2.3	Classes detalhadas: Outros domínios	45
5	Comparação das Versões do Software	47
6	Considerações Finais	48
6.1	Conclusão	48
6.2	Trabalhos Futuros	48
	Referências Bibliográficas	50
	Anexos:	52

RESUMO

O Sistema Informatizado para Gerenciamento de Ouvidorias possui três premissas para flexibilizar sua implantação: independência de *Browser*, independência do Sistema Operacional e independência do Banco de Dados. Entretanto, a última premissa ainda não estava resolvida. Para resolvê-la foi construída uma camada de persistência afim de isolar e desacoplar o código de persistência do restante da aplicação.

Para o desenvolvimento da camada de persistência foi estudada a versão 2.0 da especificação JDO e aplicada através de sua implementação de referência, o JPOX. Foram utilizados também os *frameworks* Spring - por fornecer diversas facilidades para construção de persistência e suporte a transações declarativas - e JUnit para criar testes automatizados das classes que implementam o padrão de projeto DAO.

A utilização dos *frameworks* supramencionados, os conceitos de herança e polimorfismo e também o isolamento da persistência em DAOs, proporcionaram altos níveis de reusabilidade, além de código fonte conciso, claro e testado. Desta forma, ocorre a redução dos custos para manutenção do código e para adição de novas funcionalidades, propiciando, ainda, maior flexibilidade para implantação do software.

ABSTRACT

The *Sistema Informatizado para Gerenciamento de Ouvidorias*, a information system for Ombudsman management, that possess three premises to flexible your implantation: independence of Browser, independence of the Operational System and independence of the Data Base. However, the last premise was still not decided. To decide a persistence layer was constructed to isolate and to separate the code from persistence of remain of the application.

For the development of the persistence layer, the version 2.0 of specification JDO was studied and applied using its reference implementation, the JPOX. They had also been used framework Spring - by supplying diverse easiness for persistence construction and support for declarative transactions - and framework JUnit to create automatically tests of the class that implement the design pattern DAO.

The use above mentioned about frameworks, the concepts of inheritance and polymorphism and also the isolation of the persistence in DAOs, had provided high levels of reusability beyond code source concise, clearly and tested. Therefore, happen the reduction of the costs for maintenance of the code and to addition of new functionalities, propitiating still greater flexibility for implantation of software.

1 Introdução

Dada a atual conjuntura política e social que demanda das organizações processos de mudanças e adaptação cada vez mais rápidos, possuir apenas dados armazenados não é mais suficiente e pode se tornar muito caro. Logo, há uma necessidade crescente de transformar esses dados em informação, auxiliando o processo de tomada de decisão e possibilitando as organizações mudanças mais rápidas. É neste cenário que estamos vivendo hoje, também conhecido como a Era da Informação.

Por outro lado, de acordo com Mello e Cunha (citado por Leite, 2004) os clientes estão cada vez mais exigentes, intolerantes à falhas e atrasos, motivados por uma concorrência cada vez mais acirrada e, também, por terem informações para uma melhor avaliação antes de uma aquisição. Para Leite (2004) isto exige que o cliente seja encantado, conquistado para que se torne fiel e também um parceiro, demonstrando sua satisfação. Os produtos e/ou serviços destas organizações passam a ter um valor diferenciado frente aos seus concorrentes e também seu preço passa a ter peso menor na aquisição do produto e/ou serviço. Do contrário, em sua primeira oportunidade de menor preço, o cliente se deslocará ao concorrente e poderá até mesmo denegrir a imagem da organização onde se encontrava anteriormente. Kotler (citado por Leite, 2004) indica que os clientes satisfeitos tendem a indicar o produto e/ou serviços a potenciais clientes. Dessa forma, o custo de seu atendimento tende a cair no decorrer do tempo, além do fato que estes podem comprar mais e serem menos sensíveis a mudanças de preços quando satisfeitos.

Com o intuito de fidelizar e satisfazer seus clientes, muitas organizações estão mudando seu foco, antes voltado apenas ao lucro a qualquer custo. Agora estão caminhando para um novo foco, o foco no cliente. Dada essa necessidade, o gerenciamento do relacionamento com o cliente ou apenas CRM (*Customer Relationship Management*) está em plena evidência. Shani e Chalasani (citado por Leite, 2004) defeniram CRM como:

um esforço integrado para identificar, construir e manter uma rede de relacionamentos com consumidores individuais e fortalecer continuamente esta rede com benefícios mútuos para ambos os lados através de contatos interativos, individualizados e com agregação de valor em um longo período de tempo.

Segundo Leite (2004) as empresas estão percebendo a importância de prover um atendimento personalizado. Dessa forma, elas podem aumentar sua lucratividade e obter mais sucesso caso suas atenções sejam dirigidas para seus atuais clientes. A autora também ressalta a importância do envolvimento de todos os colaboradores da organização, não somente aqueles que possuem contato direto com os clientes.

Para que haja um bom relacionamento é necessário o suporte das mais diversas áreas da organização, possibilitando que esta seja vista de forma única e bem integrada sob a ótica dos clientes. Logo, é dever das organizações se adaptarem aos seus clientes, dando um bom atendimento, satisfazê-los em suas necessidades, para que haja um relacionamento duradouro, vantajoso e lucrativo para os dois lados. Este relacionamento também pode ser chamado de “ganha-ganha”. Farias e Oliveira (2005) afirmam que estratégias de CRM só terão resultado com o comprometimento dos colaboradores, pois eles que representam as organizações e também concretizam os relacionamentos se estiverem motivados e capacitados para realizá-los e mantê-los.

Para que a mudança do foco (nos clientes) da organização proposta pelo CRM obtenha efetividade, é necessário possuir um processo de implantação muito bem planejado. Leite (2004) sugere que este processo seja visto como um projeto e que seja gerenciado como tal. É importante que esta implantação seja apoiada pela alta administração e que alguém do alto escalão e respeito dentro da organização acompanhe esse processo e cobre as mudanças necessárias, pois essa implantação exigirá mudanças de processos e também maior integração entre os setores para que eles trabalhem em sintonia objetivando a satisfação e a fidelização do cliente.

Sem os aspectos citados anteriormente tal processo mais cedo ou mais tarde tenderá ao fracasso. Porém, não é objetivo deste trabalho aprofundar este tema, tratado de forma detalhada pelos autores Leite (2004) e Farias e Oliveira (2005).

Complementarmente ao CRM, segundo Alves Jr. (2002), outra tendência que apresenta grande crescimento é o instituto das Ouvidorias (em inglês Ombudsman). Originalmente criado na Suécia por volta de 1809, chegou ao Brasil na segunda metade da década de 80. Entretanto, começou a ganhar expressividade a partir de 1995 com a criação da Associação Brasileira de Ouvidores/Ombudsman e também pela realização de Encontros Nacionais discutindo o tema.

1.1 Ouvidoria

Alves Jr. (2002) cita que, devido ao grande volume de serviços prestados, principalmente por organizações de médio e grande porte, alguns erros acabam passando despercebidos e se perpetuam devido à dificuldade da identificação destes problemas, causando insatisfação ou até mesmo fuga de seus clientes-cidadãos¹. Mas como corrigir estes erros e evitar que eles se repitam? É este questionamento que a Ouvidoria visa responder apoiado em processos organizacionais pré-estabelecidos e com prazos bem definidos.

A Ouvidoria visa à aproximação da relação entre a organização e seus clientes-cidadãos fornecendo um canal de comunicação objetivo, personalizado, a fim de ouvir quais as insatisfações, as sugestões e até mesmo os elogios apresentados pelos seus clientes. Dessa forma, o Ouvidor, funcionário de origem interna ou externa, que gerencia o funcionamento da Ouvidoria, passa ser a "voz" dos clientes-cidadãos na organização, apontando aos gestores da organização quais problemas estão ocorrendo e

¹ Cliente-Cidadão é um termo utilizado para abranger tanto o cliente de uma organização da iniciativa privada, bem como, o cidadão que usufrui um serviço da administração pública.

apresentando sugestões de melhoria [ALVES JR. 2002].

Para que isto seja possível, uma Ouvidoria recebe manifestações dos clientes-cidadãos da organização. Dessa forma, uma Manifestação é a expressão de uma situação vivida pelo cliente-cidadão que pode revelar tanto uma situação de agrado por um bom atendimento ou até a denúncia de algum ato ilícito praticado por funcionário da organização.

Assim, segundo ALVES JR. (2002), essas manifestações² apresentadas pelos clientes-cidadãos podem ser classificadas da seguinte forma:

- Reclamação: são queixas, manifestações de desagrado ou protesto.
- Denúncia: é mais grave do que uma reclamação; é delatar um fato de caráter ético, ilícito, sigiloso ou de risco coletivo.
- Sugestão: proposta de mudanças, alterações de procedimentos submetidos à apreciação da instituição.
- Elogio: reconhecimento ou demonstração de satisfação para com o serviço recebido.
- Informação: fornecimento de dados sobre um serviço ou prestador de serviço.

É importante ressaltar que quando um cliente-cidadão recorre à Ouvidoria, ele está normalmente com uma carga emocional muito grande, normalmente devido a atendimentos insatisfatórios em outras instâncias da organização, vislumbrando na Ouvidoria sua última possibilidade de resolução de seu problema. Logo, a Ouvidoria não pode ser um processo burocrático. OMD 2005 (3) diz que a Ouvidoria pretende humanizar algumas soluções burocráticas que tendem a impessoalidade, sugerindo a adoção de novos procedimentos.

As organizações devem ver nas manifestações de seus clientes-cidadãos uma ótima possibilidade de conquistar e satisfazer seus clientes, dado que estes somente manifestam seus sentimentos caso estejam interessados em manter esse relacionamento, melhorando os produtos ou serviços prestados pela organização.

Conforme o número destas manifestações cresce, a organização passa a ter um banco de dados extremamente importante sobre o seu funcionamento, evidenciando os

² Em sua Dissertação Alves Jr. (2002) utilizou o termo “solicitação”. Entretanto, para as classificações de elogio e sugestão este termo é muito restrito. Por isso, este termo foi substituído por “manifestação”, que abrange todas as classificações.

problemas vividos pelos seus clientes, sugestões apresentadas, entre outros. Entretanto, possuir este banco de dados não é suficiente. Neste sentido, OMD 2004 alerta para a necessidade de a Ouvidoria possuir uma infra-estrutura adequada para atender a demanda de manifestações, bem como, possuir um Sistema de Informação eficiente que gerencie as manifestações, controle os prazos e encaminhamentos dentro de uma manifestação. Além disso, é necessário que tal sistema possibilite a elaboração de relatórios e gráficos gerenciais. No entanto, nada disto adianta se não houverem profissionais bem capacitados para interpretar estes relatórios e gráficos transformando-os em informações valiosas sobre o funcionamento da organização. Por exemplo, em quais unidades, departamentos, setores da organização que existe maior frequência de reclamações, quais os assuntos mais frequentes de manifestações, entre outros.

OMD 2004 ressalta o compromisso do Ouvidor de repassar aos gestores da organização informações sobre as maiores demandas identificadas através das manifestações de seus clientes-cidadãos. Por isso, a Ouvidoria deve ser um órgão de apoio, conhecido também pelo termo "*staff*" e também o Ouvidor deve ter livre acesso às mais diversas áreas da organização, questionando, propondo soluções baseadas nas manifestações recebidas, bem como é necessário que as unidades contribuam para que haja bom funcionamento do canal de comunicação entre a organização e seus clientes. Caso não esteja neste nível, pode acontecer dos colaboradores não darem a relevância necessária e o que deveria ser um benefício pode se tornar algo que denigra a imagem e credibilidade da organização.

Portanto, as organizações que possuem uma Ouvidoria bem implantada poderão ter benefícios tais como: obtenção de um processo contínuo de melhoria organizacional, consolidado pela opinião de seus clientes-cidadãos e ter uma melhoria da sua imagem organizacional, melhoria da satisfação e da fidelização dos seus clientes-cidadãos, justificando, assim os custos envolvidos na sua implantação.

É possível observar uma tendência de que o CRM normalmente está voltado ao comportamento do cliente ou de um segmento de clientes. Já a Ouvidoria normalmente está mais focada na opinião do cliente. Assim é perfeitamente possível integrar estas duas estratégias, visando obter resultados ainda mais expressivos em relação a satisfação e fidelização dos clientes, pois através da opinião do Cliente, suportada pelos processos que integram a Ouvidoria possibilitará a comprovação ou correção de falhas relacionadas ao comportamento dos clientes definidos pelo CRM, proporcionando relacionamentos ainda mais duradouros e vantajosos para ambos os lados.

1.2 Tema

O tema deste trabalho é o estudo e aplicação de persistência de dados, utilizando JDO, para obtenção de independência do banco de dados para o Sistema Informatizado para Gerenciamento de Ouvidorias.

1.3 Definição do Problema

Este Trabalho de Conclusão de Curso surgiu através de um problema encontrado na Empresa OMD Soluções para Ouvidorias, que desenvolveu o Sistema Informatizado para Gerenciamento de Ouvidorias, parte integrante de sua solução. O objetivo do software é ser o mais flexível possível quanto à infra-estrutura existente na organização onde será implantado. Para isto, o software possui três bases de sustentação: independência do navegador (*browser*), independência do sistema operacional e independência do banco

de dados. As duas primeiras bases já estão resolvidas, uma através de testes entre diferentes navegadores e outra através do contêiner *Servlet TomCat*. No entanto, a última premissa ainda não estava resolvida.

Logo, o problema a ser resolvido compreende em tornar independente o banco de dados do Sistema Informatizado para Gerenciamento de Ouvidorias. Assim, haverá uma diminuição nos custos de implantação, pois o Sistema possuirá uma camada de persistência que flexibilizará sua implantação utilizando, na maioria dos casos, a infraestrutura existente na organização. A tecnologia é importante e grande facilitadora. Entretanto, se forem agregados custos de aquisição de Hardware e Software, isto pode inviabilizar sua implantação principalmente em organizações de menor porte.

1.3.1 Camada de Persistência

Camada de Persistência é uma aplicação que visa diminuir a complexidade e tornar mais transparente o armazenamento de dados. Isto torna-se muito interessante quando existem dois mundos diferentes a exemplo da Orientação a Objetos, onde cada entidade do mundo real é um objeto, tendo de conviver com Bancos de Dados Relacionais, onde cada entidade do mundo real se torna uma linha de uma tabela. Além disso, nestes bancos de dados existem diversas diferenças nos dialetos de SQL (*Structured Query Language*), que podem tornar árdua a troca de um Banco de Dados Relacional para outro [MARAFON 2005].

Por isso, constrói-se uma camada de persistência, normalmente baseada em alguma API ou Framework de Persistência, a exemplo de JDO, Hibernate, iBatis SQLMaps, EJB, entre outros. Estas tecnologias auxiliam bastante o desenvolvedor desta camada de persistência, pois diminui substancialmente a quantidade de código a ser escrito, pois já existem diversas funcionalidades comuns de armazenamento implementadas e outras funcionalidades são obtidas através da especificação de meta dados em XML (*eXtensible Markup Language*) ou em arquivos de texto, não necessitando recompilação de classes.

1.4 Escopo de Estudo

Neste trabalho é estudada a especificação para persistência de dados JDO e, implementada uma camada de persistência ao Sistema Informatizado para Gerenciamento de Ouvidorias, para que ele se torne independente do banco de dados, facilitando seu processo de implantação, podendo ser instalado em diferentes bancos de dados relacionais, necessitando de poucas alterações. Após a construção da camada de persistência é realizada uma análise comparativa entre a camada atual utilizando JDBC e a camada construída com JDO abordando aspectos de desempenho, portabilidade, facilidade de manutenção e adição de novas funcionalidades.

A especificação para persistência de dados JDO, foi escolhida por possuir diversos fornecedores o que não ocorre com outras tecnologias de persistência como é o caso de Hibernate e, iBatis SQLMaps. Utilizando sua infra-estrutura padrão é possível trocar de fornecedor. Além disso, vários fornecedores disponibilizam extensões que trazem mais facilidades e também diversas otimizações para o mapeamento objeto-relacional. A versão 2.0 da especificação JDO, que até o final deste trabalho não estava finalizada, mas encontrava-se bem próxima deste estágio, proporcionando grandes vantagens sobre sua versão 1.0, bem como, se comparado ao framework de persistência Hibernate, que atualmente é um padrão de mercado nesta área de persistência, logo a utilização de uma implementação JDO mostra até como um desafio, para demonstrar seu valor.

1.5 Objetivo Geral

Estudar a especificação para persistência de dados JDO e aplicá-la através de sua implementação referência, o JPOX, ao Sistema Informatizado para Gerenciamento de Ouvidorias, tornando-o independente do banco de dados relacional.

1.6 Objetivos Específicos

- Descrever o Sistema Informatizado para Gerenciamento de Ouvidorias, para esclarecer sua complexidade e justificar a necessidade da existência de uma camada de persistência independente do banco de dados;
- Estudar a especificação para persistência de dados JDO e aplicá-la com sua implementação de referência, o JPOX, além de fornecer um exemplo prático de implementação para consolidar os conceitos envolvidos;
- Desenvolver a camada de persistência para obtenção da independência do banco de dados;
- Realizar uma análise comparativa entre camada atual (JDBC) versus camada de persistência (JDO).

1.7 Motivação

A motivação surgiu através de um problema encontrado pela empresa OMD Soluções para Ouvidorias, que está incubada no Centro GeNESS (Centro de Geração de Novos Empreendimentos em Software e Serviços), desde maio de 2003. O autor deste trabalho é sócio desta empresa desde janeiro de 2004, em conjunto com o Sr. Mário Nelson Alves Jr. Este último tem o título de mestre em Administração com ênfase na área de Ouvidorias.

A nova versão do software, desenvolvida neste trabalho, torna o software independente do banco de dados relacional trazendo maior flexibilidade e facilidade ao processo de implantação, demonstrando assim a importância de integrar o departamento comercial e de tecnologia. Dessa forma, será possível utilizar a infra-estrutura disponível na organização, reduzindo os custos com aquisição de softwares e equipamentos de hardware. Isto se torna mais relevante, dado que muitas organizações não aceitam possuir dois ou mais bancos de dados, devido as suas particularidades tornando o processo de gerenciamento de banco de dados ainda mais complexo. Além disso, como demonstrado na introdução do trabalho, a organização deve se adaptar a seus clientes e não os clientes serem obrigados a adaptar-se à organização.

O software já possui diversos diferenciais frente a seus concorrentes. Este trabalho traz um diferencial ainda maior à empresa, principalmente frente a soluções “caseiras” desenvolvidas pelo próprio departamento de informática ou de desenvolvimento de software. Tais soluções, normalmente, devido a competição interna nas organizações para o desenvolvimento de softwares, não permitem ao software construído atingir um nível de maturidade. Ocasionalmente falta de tempo para realizar melhoria no software, ou mesmo pelo Ouvidor não conseguir especificar ou ainda por não saber as necessidades envolvidas para que haja bom gerenciamento da Ouvidoria. Logo, o resultado geralmente é bastante incompleto e propenso à falhas.

Assim, o trabalho realiza uma ponte entre a Universidade Federal de Santa Catarina e a empresa OMD Soluções para Ouvidorias, dando um caráter bastante prático, que tem como produto final o Sistema Informatizado para Gerenciamento de Ouvidorias independente do banco de dados. Isto é possível utilizando a especificação para persistência de dados JDO (*Java Data Object*) para a construção de uma camada de persistência.

1.8 A Empresa: OMD Soluções para Ouvidorias

A OMD Soluções para Ouvidorias nasceu a partir de uma dissertação de mestrado que demonstrou a contribuição estratégica das ouvidorias para a melhoria dos serviços prestados pelas organizações. A partir da identificação de uma oportunidade de mercado nesta área, foi elaborado e apresentado um Plano de Negócios ao GeNESS, uma pré-incubadora para empresas de base tecnológica e vinculado ao Departamento de Informática e Estatística(INE) da Universidade Federal de Santa Catarina - UFSC.

O projeto foi então selecionado e, desde abril de 2003, a OMD Soluções para Ouvidorias está instalada nesta pré-incubadora, recebendo capacitação técnica e gerencial para o desenvolvimento de suas atividades.

Durante estes três anos a empresa já se destaca na capacitação de profissionais de Ouvidoria/Ombudsman e também por sua solução inovadora nesta área, compreendendo um processo de implantação muito bem planejado e o Sistema Informatizado para Gerenciamento de Ouvidorias que possui controle total de todo o processo da Ouvidoria além de excelentes relatórios e gráficos gerenciais, possibilitando a melhoria contínua da organização.

1.8.1 A pré-incubadora: GeNESS

O Centro de Geração de Novos Empreendimentos em Software e Serviços – GeNESS é uma pré-incubadora vinculada ao Departamento de Informática e Estatística (INE) da Universidade Federal de Santa Catarina – UFSC. Sua missão é incentivar e dar suporte à criação de empresas de base tecnológica, especialmente as de software e serviços, sendo uma referência de apoio à criação de empresas qualificadas para alavancar empreendimentos na área de informática, agindo como uma ponte integradora entre os ambientes acadêmico, científico, industrial e de mercado de forma decisiva para a realização do potencial tecnológico no Estado de Santa Catarina [GENESS].

2 Sistema Informatizado para Gerenciamento de Ouvidorias

O sistema informatizado para gerenciamento de Ouvidorias é inovador na sua área, por se tratar de um software totalmente web para gerenciamento da Ouvidoria, tornando possível ao Ouvidor possuir total controle das manifestações apresentadas à organização, desde seu recebimento, passando por seu encaminhamento para as áreas responsáveis, setores ou departamentos dentro da organização, resposta ao cliente-cidadão, avaliação da resposta e encerramento da manifestação.

Entretanto, como citado por Alves Jr (2004) no artigo “Garantindo a efetividade da Ouvidoria”, de nada adianta a organização possuir uma Ouvidoria para receber as manifestações de seus clientes-cidadãos se ela não estiver disposta a mudar seus processos e corrigir suas disfunções. Para auxiliar esta mudança é necessário saber quais as unidades e assuntos que merecem maior atenção dos gestores, sob a ótica do principal interessado: O CLIENTE-CIDADÃO. Novamente, percebe-se a necessidade de um sistema informatizado que emita relatórios e gráficos gerenciais, permitindo uma melhor identificação destes problemas principalmente a medida que o número de manifestações aumenta. O software deve ainda permitir o refinamento das informações, compreendendo desde uma visão sistêmica da organização, podendo detalhar até uma manifestação específica (visão micro).

A seguir é apresentado um resumo das principais características do Sistema Informatizado para Gerenciamento de Ouvidorias:

- Distribuído na forma de “software livre”.
- Fácil adaptação aos equipamentos e softwares já existentes na Organização.
- Pode ser acessado pela Internet de qualquer Sistema Operacional (Windows, Linux) e também de qualquer Navegador (Internet Explorer, Mozilla, Firefox, Opera).
- Emprega a Tecnologia JAVA/WEB (JSP/SERVLET), não sendo necessária à instalação do software para cada usuário.
- Registra e controla o recebimento de manifestações originadas por qualquer forma de atendimento: call center, atendimento pessoal, internet, carta, fax, caixa de sugestões, etc.
- Gerencia o encaminhamento das manifestações para as áreas responsáveis pela resposta na Organização, acompanhando os prazos estabelecidos.
- Encaminha respostas para o cidadão, disponibilizando-as para call center, Internet, carta, fax, etc.
- Permite a emissão de gráficos e relatórios gerenciais dinâmicos drill-down (recurso que permite ligação entre relatórios, possibilitando gerar relatórios gerenciais de alto nível, vislumbrando toda a organização até o detalhamento de uma manifestação específica).
- Possibilita efetuar diversas configurações no Sistema, mesmo quando está ativo (configuração on-line).

Porém, tal sistema não funciona sozinho, “apertando apenas um botão”. Para que

um sistema de informação apresente um bom funcionamento, é necessário que haja um suporte de processos e principalmente de pessoas. Na Ouvidoria não poderia ser diferente já que o sistema informatizado é uma ferramenta, não uma “solução final”. Portanto, para que a implantação de uma Ouvidoria obtenha sucesso, é necessário muito mais do que um software.

De forma resumida, a implantação de uma Ouvidoria exige um amplo conhecimento da organização e também de seus produtos e/ou serviços, bem como a identificação dos potenciais usuários da Ouvidoria. Após isto, é necessário definir os possíveis assuntos, prazos internos e externos para que o cliente-cidadão, á no momento de sua manifestação, obtenha um prazo máximo para recebimento de sua resposta.

Como este prazo depende das mais diversas áreas da organização, é necessário que haja uma sinergia entre elas e cada uma possua um representante da Ouvidoria, pois em médias e grandes organizações apenas o Ouvidor não conseguiria gerenciar todo o processo, encaminhar respostas ao cliente-cidadão e também propor sugestões de melhorias. Deve-se ter o cuidado de não confundir a Ouvidoria com um órgão que resolve todos os problemas da Organização, uma “resolvedoria”. Sua função é gerenciar o encaminhamento da manifestação e solicitar a resposta para a área responsável nos prazos estipulados, avaliando a resposta obtida e retornando ao cliente-cidadão.

Da mesma forma, por trazer profundas mudanças na organização, é fundamental que todas as pessoas da organização estejam sensibilizadas sobre a importância da Ouvidoria, bem como do Cliente-Cidadão para a organização, evitando um ponto de vista que a rotula como apenas mais um órgão de fiscalização dentro da organização.

Conclui-se, então, que somente com um software muito pouco pode ser feito. Porém um processo de implantação bem planejado e um sistema informatizado podem auxiliar em muito no gerenciamento das manifestações e na identificação de áreas que estejam merecendo maior atenção. Entretanto, não é objetivo do trabalho se aprofundar no processo de implantação. Mais detalhes podem ser obtidos no site da empresa OMD Soluções para Ouvidorias e também na dissertação de Alves Jr (2002).

2.1 Outras Soluções Existentes em Ouvidoria

No mercado onde está inserida a empresa OMD Soluções para Ouvidorias, a maior parte dos concorrentes têm seu foco voltado para a “terceirização” do serviço de ouvidoria, ou seja, as organizações interessadas contratam a prestação deste serviço por um período certo ou indeterminado. Tal situação descaracteriza um dos princípios básicos da Ouvidoria: o estabelecimento de um canal de comunicação entre o cliente-cidadão e a organização.

A empresa Infoexpert, com sede em Recife-PE, oferece um sistema chamado PROVIDORIA que se propõe a manter a organização informada das falhas ou acertos dos procedimentos adotados, que foram observados pelo usuário desde o início do seu atendimento, inclusive durante o período de pós-atendimento. A partir dessas informações, uma equipe de assessores e especialistas da própria organização interage com o sistema para corrigir e ajustar os mecanismos e processos, visando garantir uma maior eficiência dos serviços ofertados. Esta empresa está no mercado desde 1999 e a maior parte de seus clientes localiza-se na região Nordeste. O software é alugado para seus clientes através de contratos anuais, sendo que seu preço varia entre R\$ 250,00 a 900,00/mês por ponto instalado, conforme a complexidade e número de usuários.

Um concorrente que se aproxima do produto oferecido pela OMD Soluções para Ouvidorias atua sob o nome Ombudsman Associados, com sede em São Paulo-SP. Ela

oerece serviço de consultoria voltada para administração pública, incluindo a implantação do serviço de Ouvidoria/Ombudsman, desde a fase de orientação e convencimento da organização (estratégia), diagnóstico/prognóstico, formatação, comunicação, posicionamento, até seu pleno funcionamento de maneira pro-ativa. Entretanto, com relação ao sistema de informações, não há um produto pronto.

Outro grupo de concorrentes é formado por empresas de software não especializadas no desenvolvimento de soluções para ouvidorias. Normalmente, o sistema para gerenciamento da demanda da ouvidoria é agregado como mais uma parte do pacote de informatização da organização, como é o caso do produto oferecido pela Cetil Informática, com sede em Blumenau-SC; Soft Micro, com sede em Araçatuba-SP; Softsul – Sociedade Sul-Riograndense de Apoio ao Desenvolvimento de Software e Betha Sistemas Ltda., com sede em Criciúma-SC.

Ainda há aqueles que prestam consultoria além do fornecimento do software. É o caso da Ombudsm@n Service, Target Consultoria, com sede em São Paulo-SP e Ouvidoria IPM - IPM Automação e Consultoria Ltda., com sede em Porto Alegre-RS.

A própria Associação Brasileira de Ouvidores/Ombudsman – ABO, com sede em São Paulo–SP, presta consultoria para implantação de ouvidorias e a realização de cursos na administração pública e privada.

Verifica-se também que algumas empresas de software desenvolvem apenas o sistema informatizado para gerenciamento da demanda da ouvidoria, normalmente agregando este pacote a um produto maior já instalado na organização.

Em comparação com essas empresas, a OMD Soluções para Ouvidorias destaca-se pela consultoria prévia realizada nas dependências da organização, com o objetivo de estabelecer os fluxos, responsabilidades e prazos para encaminhamento das manifestações. Aliado a este fator, o sistema informatizado permite a emissão de gráficos e relatórios gerenciais com informações sobre as ocorrências mais freqüentes, transformando a Ouvidoria numa poderosa ferramenta gerencial para identificar áreas da organização que estejam merecendo maior atenção dos gestores, sob a ótica do principal interessado: O CLIENTE-CIDADÃO; e assim servir de suporte ao processo de tomada de decisões.

Outro aspecto importante diz respeito à localização geográfica, tendo em vista que as empresas concorrentes estão situadas em outros estados, dessa forma, os custos relativos à implantação de ouvidorias em Santa Catarina seriam menores que os praticados pela concorrência.

Especialmente no Estado de Santa Catarina, onde não há empresas já consolidadas exclusivamente nesta área, a entrada de uma nova empresa no mercado é mais acessível. Além disso, a empresa pioneira tende a se tornar um referencial para o mercado e é exatamente o que está ocorrendo com a empresa OMD Soluções para Ouvidorias dentro do estado, iniciando a colheita dos frutos plantados a partir de 2003, consolidados ainda mais por uma forte parceria com a Associação Brasileira de Ouvidores – Secção Santa Catarina (ABO-SC).

3 JDO

3.1 Um breve histórico

Java Data Object, ou apenas JDO como é mais conhecido, é a especificação da comunidade (JCP) para implementação de persistência de objetos em Java. Em 1999, foi iniciado o desenvolvimento da API JDO tendo sua primeira versão lançada em 2002. O objetivo da API JDO é facilitar a construção de uma camada de persistência, fazendo com que a camada lógica (ou camada de negócio ou ainda camada de regra de negócio) independa se os dados são persistidos em arquivos, bancos de dados relacionais ou mesmo bancos de dados orientados a objetos [JDOCENTRAL].

Apesar de fornecer diversas facilidades para a construção de persistência de objetos em Java, a versão 1.0 da especificação JDO, possuía uma série de deficiências fazendo que as implementações JDO, como JPOX, Kodo, e outras, necessitassem criar inúmeras extensões em sua implementação. Dessa forma, perdia-se uma das maiores vantagens da API JDO 1.0 a independência de fornecedor, se comparada ao Hibernate e ao EJB (Enterprise Java Beans). Através desta independência, as classes de persistência e seus metadados podem ser mantidos na troca entre uma implementação JDO para outra, como por exemplo, trocar a implementação JPOX pela implementação Kodo.

Por essas deficiências apresentadas na versão 1.0 da especificação JDO e também devido a Sun financiar a especificação JDO e também a especificação EJB. Por isso, a API JDO esteve prestes a ser descontinuada. No entanto, isto não aconteceu, pois se analisarmos mais a fundo EJB e JDO possuem objetivos e aplicações diferentes.

A especificação EJB, devido a sua complexidade, deveria ser utilizada em projetos grandes que são críticos, seja pelo número de acessos, número de transações, entre outros. Ele também exige que a aplicação esteja em algum servidor de aplicação, como o JBoss, Weblogic, entre outros. Já com JDO não existe tal exigência. É possível construir tanto aplicações locais como aplicações *web*. No caso de aplicações *web* basta realizar o *deploy* em um contêiner *Servlet* como o *TomCat*. O aprendizado para o desenvolvimento de persistência utilizando JDO é muito mais suave, pois através de JDO realizamos a persistência de POJOs (*Plain Old Java Objects*), que não nada mais são do que *JavaBeans*, com construtor padrão sem parâmetros e métodos públicos *get* e *set* para acessar suas propriedades. Além disso, JDO é uma solução muito mais leve, podendo ser utilizado em pequenos e médios projetos.

Por esses motivos no final de 2004 em uma reunião dos fornecedores que compõem a linguagem JAVA, observaram que as duas tecnologias possuíam cada um seu espaço. Como resultado desta reunião definiram que a especificação JDO ganharia uma nova versão afim de corrigir os problemas existentes na versão 1.0 e que a implementação de referência da especificação 2.0 ficaria com o JPOX (*Java Persistent Objects JDO*). JPOX é uma implementação open-source da especificação JDO totalmente compatível com a versão 1.0 e está auxiliando na finalização da especificação JDO 2.0. Esta versão resolve grande parte dos problemas existentes na versão 1.0, além de disponibilizar uma série de melhorias propiciando uma persistência de objetos ainda mais facilitada. Assim a especificação JDO vem conquistando grande expressão e mostrando-se em diversas ocasiões mais adequada se comparada com Hibernate, o padrão do mercado em termos de persistência de objetos Java, e também a persistência utilizando EJB. No decorrer desse capítulo as vantagens da utilização da API JDO 2.0 ficarão mais evidentes.

3.2 A especificação JDO

Na documentação do Site do JPOX, o JDO é descrito como uma *interface* (ou API) para persistir POJOs em um local de persistência (*datastore*). Este local pode ser um Banco de Dados Relacional, um Banco de Dados Orientado a Objetos, um arquivo XML ou qualquer outra forma de armazenamento. Embora sua utilização mais comum é no mapeamento objeto-relacional, onde há um Modelo de Orientado a Objetos que precisa ser persistido e retornado de um Modelo de Entidade-Relacional materializado em um Banco de Dados Relacional como MySQL, PostgreSQL, SQL Server, Oracle, entre outros [JPOX].

Para que isto aconteça, os objetos persistidos são gerenciados por instâncias de *PersistenceManager*, obtidos através de um *PersistenceManagerFactory*. Para persistir um objeto (seja novo ou já existente) utilizaremos o método *makePersistent* sendo executado em uma instância de *PersistenceManager*. A chamada deste método é “traduzida” em um INSERT ou UPDATE, entendido pelo banco de dados relacional escolhido. No entanto, cabe ressaltar que em informática e tecnologia nada é “mágico”. Antes de chegarmos neste ponto, precisamos definir os metadados que representarão o mapeamento objeto-relacional, ou seja, como uma classe com seus atributos, associações e heranças são persistidas em apenas uma tabela ou em diversas tabelas. Também em tempo de compilação, deve-se fazer um enxerto (*enhancer*) no bytecode, fazendo com que as classes que representam o domínio do problema estendam *PersistenceCapable* e implementem alguns métodos, tornando possível sua persistência [JPOX]. Existem pessoas que vêem nisto uma desvantagem pois é um processo intrusivo que altera o *bytecode*. Entretanto, se olharmos com mais calma, esse processo de enxerto possui vantagens como veremos mais adiante neste capítulo.

3.3 Mapeamentos em JDO

Como comentado anteriormente, para que uma classe seja persistida, ela deve ser um POJO e também é necessário que haja mapeamento explicando como esta será persistida. Em JDO esta definição ocorre em arquivos com extensão *.jdo*, que obedecem a sintaxe de um arquivo XML. Existem diversas opções para colocar o(s) arquivo(s) *.jdo* em uma aplicação. As mais comuns são a definição de um único arquivo *.jdo* onde é realizado o mapeamento de todas as classes de domínio (normalmente *package.jdo*). Alternativamente é possível definir um arquivo *.jdo* para cada uma das classes de domínio (*nomeClasse.jdo*) [JPOX].

O mapeamento de cada classe diz em qual tabela ela será persistida, e também por exemplo: qual coluna é chave primária (PK); para cada atributo qual é sua coluna e qual o tipo desta coluna; como associações entre classes serão mapeadas: haverá uma coluna com chave estrangeira (FK), haverá uma tabela de relacionamento, ou ainda no banco de dados o relacionamento só será materializado no outro lado do relacionamento [JPOX]. Também é possível definir superclasses abstratas, sendo possível assim herdar atributos desde que em tabelas diferentes os atributos tenham a mesma coluna mapeada [JPOX]. Isto é muito importante sob o ponto de vista do reuso de código e algo que em EJB 2.1 não é possível.

Afim de tornar estes mapeamentos mais claros são demonstrados quatro exemplos, não sendo necessário entender o significado das classes. O objetivo neste momento é apenas apresentar alguns exemplos de mapeamento objeto-relacional em JDO. As tabelas mapeadas são criadas com MySQL. Porém, se tomado certo cuidado com os tipos de coluna utilizados, pode-se facilmente trocar o banco de dados. Ainda neste

capitulo são apresentados exemplos de como criar classes para gerenciar a persistência dos objetos.

3.3.1 Primeiro Exemplo

O primeiro exemplo contempla o mapeamento de herança, definição de PK, atributos Long, String e também um Date. Isto já serve para a maioria dos casos onde os atributos não são associados a outras classes de domínio do problema:

Classes BaseDomain, IdentidadeNome, Assunto e Feriado

```
public abstract class BaseDomain {
    protected Long id;
    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
}

public abstract class IdentidadeNome extends BaseDomain {
    protected String nome;
    public String getNome() {
        return nome;
    }
    public void setNome(String nome) {
        this.nome = nome;
    }
}

public class Assunto extends IdentidadeNome {
}

public class Feriado extends BaseDomain {
    private Date data;
    public Date getData() {
        return data;
    }
    public void setData(Date data) {
        this.data = data;
    }
}
```

A classe abstrata BaseDomain possui o atributo id e, por ser uma classe abstrata não é materializada em uma tabela, sabem-se apenas que em algum nível de subclasse haverá uma tabela mapeada com uma coluna de chave primária id.

Mapeamento da classe BaseDomain

```
<class name="BaseDomain" detachable="true" identity-type="application">
    <inheritance strategy="subclass-table"/>
    <field name="id" primary-key="true" value-strategy="autoassign"/>
</class>
```

Como a classe abstrata IdentidadeNome herda o atributo id e, este já foi mapeado em BaseDomain, define-se somente o mapeamento para o atributo nome. A propriedade *detachable* definida como true, a exemplo de BaseDomain, indica que ao obter-se uma de suas subclasses, esta já virá com os atributos id e nome definidos:

Mapeamento da classe IdentidadeNome

```
<class name="IdentidadeNome" detachable="true">
  <inheritance strategy="subclass-table"/>
  <field name="nome">
    <column length="100"/>
  </field>
</class>
```

A classe Assunto apenas herda os atributos id e nome. Dessa forma, é necessário apenas dizer que ela materializa a herança na tabela de assunto. A classe Feriado herda o atributo id e define um atributo data, mas também materializa a herança do atributo id na tabela de feriado.

Mapeamento das classes Assunto e Feriado

```
<class name="Assunto" table="assunto">
  <inheritance strategy="new-table"/>
</class>

<class name="Feriado" table="feriado">
  <inheritance strategy="new-table"/>
  <field name="data"/>
</class>
```

Tabelas assunto e feriado (MySQL)

```
assunto (id int(11) auto-increment primary key, nome varchar(100) not null);

feriado (id int(11) auto-increment primary key, data date not null);
```

Como observado nos mapeamentos demonstrados, caso o atributo possua o mesmo nome da coluna não é necessário defini-lo, isto se mostra ainda mais interessante em projeto com grande quantidade de Classes e atributos, evitando assim que os mapeamentos cresçam demasiadamente.

3.3.2 Segundo Exemplo

Como pode ser mapeado um atributo do tipo primitivo boolean ou do Wrapper Boolean, tendo em mente que os bancos de dados normalmente não possuem este tipo de coluna? A classe Modo que possui um atributo dizendo se o modo é retornável ou não:

Classe Modo

```
public class Modo extends IdentidadeNome {
  private boolean retornavel;
  public boolean isRetornavel() {
    return retornavel;
  }
  public void setRetornavel(boolean retornavel) {
    this.retornavel = retornavel;
  }
}
```

Para persistir o atributo *retornavel* é utilizado uma coluna do tipo char de tamanho 1

e no banco de dados, a coluna automaticamente recebe 'Y' para o valor verdadeiro e 'N' para o valor falso.

Mapeamento da classe Modo

```
<class name="Modo" table="modo">
  <inheritance strategy="new-table"/>
  <field name="retornavel">
    <column length="1" jdbc-type="char"/>
  </field>
</class>
```

Tabela modo (MySQL)

```
modo (id int(11) auto-increment primary key, nome varchar(100) not null,
retornavel char(1) not null);
```

3.3.3 Terceiro Exemplo

O terceiro exemplo demonstra como mapear uma associação: Como é mapeada uma classe Cidade com um atributo estado que é da classe de domínio Estado?

Classes Cidade e Estado

```
public class Cidade extends IdentidadeNome {
    private Estado estado;
    public Estado getEstado() {
        return estado;
    }

    public void setEstado(Estado estado) {
        this.estado = estado;
    }
}

public class Estado extends IdentidadeNome {
    private String sigla;
    public String getSigla() {
        return sigla;
    }
    public void setSigla(String sigla) {
        this.sigla = sigla;
    }
}
```

A classe Cidade possui um atributo estado, caso haja alguma alteração neste atributo esta alteração será persistida. Outra vantagem que pode ser observada em JDO é a definição de *fetch-group*, principalmente se comparada a Hibernate e EJB. Isto significa que pode-se, *em tempo de execução*, definir quais objetos e coleções associados são retornados em uma consulta. Este tipo de definição traz muita flexibilidade, pois definem-se os grupos de retorno no mapeamento das classes, mas são chamados da forma escolhida em tempo de execução.

Imagine um objeto com diversos objetos e coleções associados. Geralmente ao executar uma consulta sobre esse objeto não são necessários todos os objetos e coleções associadas. Com JDO retorna-se apenas o necessário. Assim tem-se uma execução mais rápida e evita-se tráfego desnecessário de dados na rede.

Mapeamento das classes Cidade e Estado

```
<class name="Estado" table="estado">
  <inheritance strategy="new-table"/>
  <field name="sigla">
    <column length="2"/>
  </field>
</class>

<class name="Cidade" table="cidade">
  <inheritance strategy="new-table"/>
  <field name="estado" persistence-modifier="persistent">
    <column name="estado"/>
  </field>
  <fetch-group name="detach_estado">
    <field name="estado"/>
  </fetch-group>
</class>
```

Tabelas cidade e estado (MySQL)

```
cidade (id int(11) auto-increment primary key, nome varchar(100) not null,
estado int(11) not null foreign key);
```

```
estado (id int(11) auto-increment primary key, sigla char(2) not null, nome
varchar(100) not null);
```

3.3.4 Quarto Exemplo

Para terminar os exemplos de mapeamentos, este exemplo demonstra um mapeamento bidirecional ao nível de classes, mas apenas unidirecional no banco de dados. Esta escolha foi feita para facilitar a consulta de objetos. Pois em um mapeamento modelagem conceitual de dados não é necessário armazenar o relacionamento nos dois lados pois um dos lados é opcional e no outro lado a chave estrangeira é única e obrigatória (foi neste lado que foi materializado o mapeamento). Vejamos um trecho das classes:

Classe OutroAssunto e Trecho da classe Manifestação

```
public class OutroAssunto extends IdentidadeNome {
    private Manifestacao manifestacao;
    public Manifestacao getManifestacao() {
        return manifestacao;
    }
    public void setManifestacao(Manifestacao manifestacao) {
        this.manifestacao = manifestacao;
    }
}

public class Manifestacao extends BaseDomain {
    private OutroAssunto outroAssunto;
    public OutroAssunto getOutroAssunto() {
        return outroAssunto;
    }
    public void setOutroAssunto(OutroAssunto outroAssunto) {
        this.outroAssunto = outroAssunto;
    }
}
```

A tabela “outro_assunto” possuirá uma coluna “manifestacao” com FK para o id da tabela de “manifestacao”. Entretanto a tabela “manifestação” não possui nenhum

relacionamento para a tabela “outro_assunto”. Abaixo, observam-se os mapeamentos que permitem o relacionamento bidirecional ao nível de classes:

Trecho do mapeamento da classe Manifestacao e mapeamento da classe OutroAssunto

```
<class name="Manifestacao" table="manifestacao">
  <inheritance strategy="new-table"/>
  <field name="outroAssunto" persistence-modifier="persistent"
    mapped-by="manifestacao"/>
</class>
<class name="OutroAssunto" table="outro_assunto">
  <inheritance strategy="new-table"/>
  <field name="manifestacao" persistence-modifier="persistent">
    <column name="manifestacao"/>
  </field>
</class>
```

Tabelas manifestacao e outro_assunto (MySQL)

```
manifestacao (id int(11) auto-increment primary key);

outro_assunto (id int(11) auto-increment primary key, nome varchar(100)
not null, manifestacao int(11) not null foreign key);
```

3.4 Enhancer (Enxertador) JDO

O Enhancer JDO é a ferramenta responsável por alterar o *bytecode* das classes de domínio, para que em algum nível de superclasse haja a classe *PersistenceCapable*, possibilitando assim seu gerenciamento pelo JDO [JPOX]. O processo de enxerto de código ocorre após a compilação das classes de domínio. Para obter maior produtividade, é recomendável automatizar este processo, configurando um processo de pós-compilação. este pode ser configurado por meio de Ant ou Maven [JPOX]. Apesar de alterar o *bytecode* existe um compromisso nas diversas implementações JDO para que o enxerto de código não afete a execução de “debug” [MARAFON 2005].

O fato das classes de domínio serem enxertadas em tempo de compilação proporciona economia em tempo de execução, pois não é necessário usar A API Reflection para acessar os atributos a exemplo de Hibernate. O *bytecode* já estará pronto para ser persistido em implementação JDO. Para mais detalhes sobre o processo de *enhancer* podem ser consultados na documentação do JPOX e também o site JDOCentral.

3.5 JDOQL – JDO Query Language

JDOQL é a linguagem de consulta de objetos utilizada por todas as implementações JDO. Trata-se de uma linguagem muito poderosa permitindo realizar consultas bastante complexas, sem a necessidade de utilizar diretamente JDBC com SQL. Isto permite desenvolver código muito mais conciso e reusável, evitando repetições de códigos semelhantes que geram grandes impactos na manutenção de uma aplicação. Ainda neste capítulo retomaremos alguns destes itens.

Consultas JDOQL podem ser definidas em uma *String* única ou utilizando criando a consulta através da classe *Query*, definindo cada um das partes da consulta, a classe de domínio que será consulta, os resultados de uma consulta, as condições e os parâmetros. Mais uma de suas vantagens pode ser encontrada com relação aos

resultados uma consulta, ao invés de serem retornados num *array* de objetos (`Object[]`), é possível informar qual a classe de retorno, bastando que haja correspondência do tipo e do nome de um atributo, diminuindo em muito o trabalho necessário principalmente em relatórios ou consultas que utilizam funções de agregação como *min*, *max* e *count*. Ao utilizar estas funções normalmente não o resultado da consulta não corresponde a objetos de uma classe de domínio. Dessa forma define-se uma classe de retorno, evitando o trabalho incomodo com `Object[]` a cada item de uma coleção de objetos [JPOX].

Analisando de forma mais detalhada a JDOQL, percebe-se que ela é mais semelhante a sintaxe Java se comparado ao HQL (Hibernate Query Language). Por exemplo, comparações são realizadas com o operador `==`. Com JDOQL, quando trabalha-se com apenas uma classe, pode-se acessar diretamente seus atributos. Sem a necessidade de utilizar um *alias* para acessá-lo o que não é possível em HQL. Em JDOQL também podemos acessar os atributos em uma consulta de uma única classe através de `this`, em Hibernate `this` se comporta como um nome renomeado qualquer.

3.6 JPOX - Java Persistent Objects JDO

O JPOX (Java Persistent Objects JDO), atualmente na versão 1.1.0.06-beta, é a implementação de referência da especificação JDO 2.0 realizando esforços expressivos para finalizá-la. Além disso, JPOX é uma implementação de código aberto da especificação JDO, que é totalmente compatível com a versão 1.0. Graças aos esforços da implementação JPOX, a implementação da SolarMetric, o Kodo, entre outros, a especificação JDO está conquistando maior expressividade devido a qualidade de suas implementações, na versão JDO 2.0, além de demonstrar vantagens se comparado ao Hibernate e também ao EJB [JPOX].

Em sua documentação existe uma explicação bastante extensa sobre a persistência e consulta de objetos em JDO, além de exemplos integrando JPOX com tarefas automatizadas de Ant e Maven para realizar automaticamente o processo de *enhancer*. Também há exemplos de integração com ambientes de desenvolvimento Eclipse e Netbeans. Mas o que se mostra mais interessante é sua integração com outros *frameworks* como Tapestry. Porém, para este trabalho foi utilizada somente a integração JPOX e Spring que se mostrou extremamente útil [JPOX].

Caso o trabalho tivesse utilizado somente JPOX, seria necessário possuir uma classe, seguindo o padrão de projeto *Factory*, afim de gerenciar o `PersistenceManagerFactory` e também as instâncias de `PersistenceManager`. Em cada método seria necessário obter uma instância de `PersistenceManager` e controlá-la através de transações programáticas, ou seja, devia-se realizar o controle de transações ao nível de código [JPOX].

3.7 Spring

O framework Spring é um “container leve”, com uma série de serviços fornecidos. Apenas aqueles necessários a aplicação foram utilizados. Um de seus serviços mais conhecidos é a Inversão de Controle (*Inversion of Control - IoC*) ou Injeção de Dependências, possibilitando realizar configuração que são carregadas quando buscamos um `bean` definido no `applicationContext.xml`. Sua integração com persistência proporciona maior reuso de código e maior facilidade de uso em classes que seguem o

padrão de projeto DAO (Data Access Object). Por exemplo, para JDBC, o Spring fornece um `JdbcTemplate`; para Hibernate fornece `HibernateTemplate`; e para JDO o `JdoTemplate` [SPRING].

Além disso, utilizando Spring é possível definir transações declarativamente, configurando-as de maneira genérica permitindo herdar essas configurações, reduzindo em muito à quantidade linhas necessária para definir um DAO e uma transação para este DAO. Dessa forma, não é mais necessário nenhum código nos DAOs para controlar transações [SPRING]. Mais informações podem ser obtidas através do site do Framework Spring.

3.8 JUnit

Se um software não tem um teste automatizado, assume-se que ele não funciona - cabe ao desenvolvedor provar que ele funciona através de testes automatizado [CHAVES 2002]. Por esse motivo e até pelo fato deste trabalho não desenvolver uma aplicação completa, mas sim desenvolver uma camada de persistência, foi utilizado JUnit para criar testes automatizados para cada um dos DAOs desenvolvidos. As metas do *framework* para testes JUnit são os seguintes: ajudar a escrever testes úteis; ajudar a criar testes que retenham seus valores com o tempo; e deve ajudar a baixar o custo de escrever testes reutilizando código [MASSOL&HUSTED 2005].

Pensando nisso, JUnit mostrou-se extremamente útil, principalmente pois durante o desenvolvimento da camada de persistência houve grande empenho em desenvolver o código seguindo o padrão de projeto DAO utilizando exaustivamente herança, reuso de código e polimorfismo. Isto proporcionou a criação de testes muito semelhantes e que requerem baixo esforço para escrevê-los. Mais adiante são vistos exemplos de código de teste escrito com JUnit.

3.9 Exemplo Prático: JPOX, Spring e JUnit

Neste exemplo observa-se a persistência e a pesquisa de objetos utilizando JPOX integrado ao Spring e também apresenta-se testes automatizados utilizando JUnit para provar que o código realmente funciona. A integração JPOX e Spring proporciona uma melhor integração entre a camada de persistência e a camada lógica, além de tornar mais organizados os diversos meta dados e configurações de um sistema.

3.9.1 DAOJdo

Este exemplo inicia como a criação de uma superclasse abstrata `DAOJdo` que implementa o padrão de projeto DAO. Em `DAOJdo` define-se métodos para retornar todos os objetos de uma classe, também retornar um objeto de uma classe a partir de sua chave primária e também salvar um objeto - por salvar um objeto entende-se realizar tanto *insert* como *update* ao nível de banco de dados.

Classe Abstrata DAOJdo

```
public abstract class DAOJdo extends JdoDaoSupport implements DAO {  
  
    protected static final String ID_NAO_ENCONTRADO = "Registro não  
    encontrado.";  
    protected static final String NENHUM_REGISTRO_ENCONTRADO = "Nenhum  
    registro não encontrado.";
```

```

public Object save(BaseDomain domain) throws Exception {
    return getPersistenceManager().makePersistent(domain);
}

public BaseDomain getDomain(Long id) throws Exception {
    return this.getDomain(getReferenceClass(), id);
}

public BaseDomain getDomain(Class classe, Long id) throws Exception {
    try{
        BaseDomain domain = (BaseDomain)
            getJdoTemplate().getObjectById(classe, id);
        return (BaseDomain)
            getPersistenceManager().detachCopy(domain);
    } catch (DataAccessException e) {
        throw new Exception(ID_NAO_ENCONTRADO);
    }
}

public Collection getAll(String ordering) throws Exception {
    return this.getAll(getReferenceClass(), ordering);
}

public Collection getAll(Class classe, String ordering) throws
    Exception {
    Collection collection;
    if (ordering == null) {
        collection = getJdoTemplate().find(classe);
    } else {
        collection = getJdoTemplate().find(classe, null, ordering);
    }
    if (collection.size() > 0) {
        collection=getPersistenceManager().detachCopyAll(collection);
        return collection;
    } else {
        throw new Exception(NENHUM_REGISTRO_ENCONTRADO);
    }
}

protected Collection pesquisar(Class classe, String filtros, String
parametros, Object[] valores, String ordenacao) throws Exception {
    Collection collection;
    if (valores != null) {
        if (ordenacao == null){
            collection = getJdoTemplate().find(classe, filtros,
                parametros, valores);
        } else {
            collection = getJdoTemplate().find(classe, filtros,
                parametros, valores, ordenacao);
        }
        if (collection.size() > 0) {
            collection =
                getPersistenceManager().detachCopyAll(collection);
        } else {
            throw new Exception(NENHUM_REGISTRO_ENCONTRADO);
        }
    } else {
        collection = this.getAll(ordenacao);
    }
    return collection;
}

```

```

protected boolean isResult(Class classe, String filtros, String
    parametros, Object[] valores) throws Exception {
    Collection collection = getJdoTemplate().find(classe, filtros,
        parametros, valores);

    if (collection.size() > 0) {
        return true;
    } else {
        return false;
    }
}

protected abstract Class getReferenceClass();
}

```

A classe `DAOJdo` estende a classe `JdoDaoSupport`, que é uma classe fornecida pelo *framework* Spring e possui um atributo de `JdoTemplate`. Isto permite configurar o objeto `PersistenceManagerFactory` apenas para esta superclasse abstrata e todas as suas subclasses possuirão este objeto configurado, proporcionando menos configurações para as subclasses de `DAOJdo`. A cada chamada ao método `getJdoTemplate()` ou `getPersistenceManager()` é obtida uma nova instância de `PersistenceManager`. Esse objeto é responsável por executar operações para persistir, excluir ou consultar objetos com JDO [JOHNSON&HOELLER 2004].

O método `save` proporciona comandos de banco de dados tanto de *insert* quanto de *update* de qualquer objeto de domínio definido no(s) arquivo(s) de mapeamento `.jdo` da aplicação. Isto já proporciona grande redução na quantidade de linhas necessárias para realizar *insert* e *update* bem como grande reuso de código se comparado ao uso de JDBC, mesmo se este estiver integrado ao Spring, pois é necessário reimplementar este método em cada um dos domínios, desprezando ainda o fato que o método `makePersistent` retorna a chave primária do objeto inserido no Banco de Dados. JDO, a exemplo do Hibernate, também viabiliza a persistência por alcance, em inglês *persistence-by-reachability*, ou seja, se houver um grafo de objetos a serem persistidos não é necessário chamar o método `makePersistent` para todos os objetos. Caso haja apenas um objeto que encontre todos os outros objetos, será realizada apenas uma chamada para persisti-los [JPOX].

O método `getDomain` retorna um objeto do domínio da aplicação a partir de sua chave primária. Caso não encontre objeto com a chave informada será lançada uma exceção identificando essa situação adversa. O objeto retornado virá com todos os objetos e coleções contidos no plano de retorno atual (`FetchPlan`). Para demonstrar um exemplo disto vamos utilizar a classe `Cidade` que possui um atributo de Estado - definidos no item de mapeamentos deste capítulo - estando no plano de retorno a cidade retornada virá o objeto estado associado. A chamada ao método `detachCopy` retorna uma cópia do objeto que não pertence mais ao grafo de persistência. Se alterada alguma de suas propriedades será necessário chamar o método `makePersistent` para persistir suas alterações fazendo que o objeto retorne ao grafo de persistência [JPOX].

O método `getAll` retorna todos os objetos de uma classe. A chamada ao método `detachCopyAll` fará uma cópia de todos os objetos contidos na coleção que não pertence mais ao grafo de persistência. Caso não encontre nenhum resultado será disparada uma exceção [JPOX].

O método `isResult` auxilia no tratamento das exceções no caso de existir alguma

coluna com chave única. Por fim, o método abstrato `getReferenceClass` serve para informar a qual classe de domínio uma classe DAO pertence.

3.9.2 BaseDAOJdo

Como existem classes no domínio que seus objetos não podem ser simplesmente excluídos, ou seja, a exclusão faz parte de um processo atômico - o processo será realizado por completo ou nada será alterado -, este é o caso da classe `OutroAssunto` e de outras classes de `Outros`. Logo, foi criada a classe `BaseDAOJdo` para as classes em que a exclusão pode ser realizada diretamente.

Classe Abstrata `IdentidadeNomeDAOJdo`

```
public abstract class BaseDAOJdo extends DAOJdo implements BaseDAO {

    public void delete(BaseDomain domain) throws Exception {
        this.delete(domain.getId());
    }

    public void delete(Long id) throws Exception {
        getPersistenceManager().deletePersistent(
            getJdoTemplate().getObjectById(getReferenceClass(), id));
    }

}
```

3.9.3 IdentidadeNomeDAOJdo

Como no domínio existem muitas classes com atributo `nome` foi criada uma superclasse `IdentidadeNomeDAOJdo` que estende `BaseDAOJdo`, nesta classe são disponibilizados métodos comuns relacionados ao atributo `nome` como: pesquisa por nome, saber se existe um objeto com o atributo `nome`, gerar exceção caso exista um determinado nome já armazenado. Esta classe também servirá para consulta objetos das classes de `Classificacao`, `Identificacao` que praticamente não são alteradas, por isso não foi criado DAO para elas.

Classe `IdentidadeNomeDAOJdo`

```
public class IdentidadeNomeDAOJdo extends BaseDAOJdo {

    public Collection pesquisarPorNome(String nome, String ordenacao)
        throws Exception {

        Collection collection;
        if (nome != null && (!nome.equals(""))) {
            collection = super.pesquisar(getReferenceClass(),
                "nome == vNome", "String vNome", new Object[] {nome}, ordenacao);
        } else {
            collection = super.getAll(ordenacao);
        }
        return collection;
    }

    public boolean isResult(String nome) throws Exception {
        return super.isResult(getReferenceClass(), "nome == vNome",
            "String vNome", new Object[] {nome});
    }

    protected Class getReferenceClass() {
        return null;
    }

}
```

```

protected void gereExcecaoSeExistir(String nome, String
                                     mensagemExcecao) throws Exception {
    if (isResult(nome))
        throw new Exception(mensagemExcecao);
}
}

```

3.9.4 AssuntoDAOJdo

Esta classe representa o DAO de Assunto, conforme visualizado a seguir, seu tamanho é bem reduzido, pois grande parte dos métodos já estão implementados. Foi necessário apenas reimplementar o método `save` para tratar de maneira mais personalizada a exceção caso o Assunto já exista e implementar o método `getReferenceClass` informando que este DAO realiza a persistência da classe de domínio Assunto.

Classe AssuntoDAOJdo

```

public class AssuntoDAOJdo extends IdentidadeNomeDAOJdo {

    protected static final String REGISTRO_JA_EXISTE = "Assunto já
existe.";

    protected Class getReferenceClass() {
        return Assunto.class;
    }

    public Object save(BaseDomain domain) throws Exception {
        Assunto assunto = (Assunto) domain;
        try {
            if (assunto.getId() != null) {
                Assunto assuntoBD = (Assunto) getDomain(assunto.getId());
                if (! (assunto.getNome().equals(assuntoBD.getNome())))
                    super.gereExcecaoSeExistir(assunto.getNome(),
                                                REGISTRO_JA_EXISTE);
            }
            return super.save(domain);
        } catch (Exception e) {
            super.gereExcecaoSeExistir(assunto.getNome(),
                                        REGISTRO_JA_EXISTE);
            throw e;
        }
    }
}

```

3.9.5 ApplicationContext.xml

Neste arquivo do Spring configura-se a criação do objeto `PersistenceManagerFactory` informando que a implementação JDO é o JPOX, este objeto também conterá as informações de conexão que serão utilizadas pelas instâncias de `PersistenceManager`. Na classe `DAOJdo` injetaremos este objeto e assim todas as suas classes herdarão essa propriedade.

Como observado no código fonte da classe `DAOJdo`, não foi escrito nenhum código de controle de transação, isto será realizado utilizando o serviço de transações

declarativas fornecido pelo *framework* Spring. Dessa forma, configura-se o `transactionManager` informando que as transações serão controladas pela classe `JdoTransactionManager`, bem como configura-se uma transação abstrata que será utilizada como base para chamar os objetos DAO. Esta transação abstrata contém quais os métodos que devem ser controlados por transação e qual o nível deste controle.

Com estas configurações bem planejadas e também classes seguindo fielmente os conceitos de herança, polimorfismo e reuso de código, reduziu drasticamente as configurações necessárias as classes de DAOs concretas, principalmente se compararmos com EJB. Ao refletirmos um pouco sobre configurações utilizadas numa aplicação muitas delas são comuns e poderiam ser herdadas. Infelizmente são poucos os *frameworks* e tecnologias que permitem herdar configurações, como os *frameworks* Spring e Tiles fornecem. As configurações realizadas para `IdentidadeNomeDAOJdo` e `AssuntoDAOJdo`, bem como para seus respectivos controles de transação `somenteLeitura` e `assunto` possibilitam trechos de configuração ainda menores.

Trecho do ApplicationContext.xml

```
<bean id="pmf"
class="org.springframework.orm.jdo.LocalPersistenceManagerFactoryBean">
  <property name="jdoProperties">
    <props>
      <prop key="javax.jdo.PersistenceManagerFactoryClass">
        org.jpox.PersistenceManagerFactoryImpl
      </prop>
      <prop key="javax.jdo.option.ConnectionURL">
        jdbc:mysql://localhost:3306/ouvidoria
      </prop>
      <prop key="javax.jdo.option.ConnectionUserName">
        nomeUsuario
      </prop>
      <prop key="javax.jdo.option.ConnectionPassword">
        senha
      </prop>
      <prop key="javax.jdo.option.ConnectionDriverName">
        com.mysql.jdbc.Driver
      </prop>
    </props>
  </property>
</bean>

<bean id="DAO" class="br.com.omd.ouvidoria.persistencia.geral.dao.jdo.DAOJdo"
abstract="true">
  <property name="persistenceManagerFactory" ref="pmf"/>
</bean>

<bean id="transactionManager"
class="org.springframework.orm.jdo.JdoTransactionManager">
  <property name="persistenceManagerFactory" ref="pmf"/>
</bean>

<bean id="abstractTxDefinition"
class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean"
abstract="true">
  <property name="transactionManager" ref="transactionManager"/>
  <property name="transactionAttributes">
```

```

        <props>
            <prop key="get*">PROPAGATION_REQUIRED,readOnly</prop>
            <prop key="pesquisar*">PROPAGATION_REQUIRED,readOnly</prop>
            <prop key="save*">PROPAGATION_REQUIRED</prop>
            <prop key="delete*">PROPAGATION_REQUIRED</prop>
            <prop key="*">PROPAGATION_REQUIRED,readOnly</prop>
        </props>
    </property>
</bean>

<bean id="baseDAO"
class="br.com.ond.ouvidoria.persistencia.geral.dao.jdo.BaseDAOJdo"
abstract="true" parent="DAO"/>

<bean id="identidadeNomeDAO"
class="br.com.ond.ouvidoria.persistencia.geral.dao.jdo.IdentidadeNomeDAOJdo"
parent="baseDAO"/>
<bean id="somenteLeitura" parent="abstractTxDefinition">
    <property name="target" ref="identidadeNomeDAO"/>
</bean>

<bean id="assuntoDAO"
class="br.com.ond.ouvidoria.persistencia.geral.dao.jdo.AssuntoDAOJdo"
parent="identidadeNomeDAO"/>
<bean id="assunto" parent="abstractTxDefinition">
    <property name="target" ref="assuntoDAO"/>
</bean>

```

3.9.6 BaseDAOTestCase

Para realização dos testes automatizados foi utilizado o *framework* JUnit sendo criada a superclasse abstrata de teste BaseTestCase, que cria o contexto para a chamada ao *framework* Spring que serão herdados por todos os testes de DAO, além de um método para exibir coleção e um método abstrato para exibir um objeto de domínio.

Classe Abstrata BaseDAOTestCase

```

public abstract class BaseDAOTestCase extends TestCase {
    protected final static ApplicationContext ctx;
    private static final String configLocation = "applicationContext-
test.xml";
    protected Iterator iterator;
    protected static final String ordenacao = "id ASC";

    static {
        ctx = new ClassPathXmlApplicationContext(configLocation);
    }

    public void exibaColecao(Collection colecao) throws Exception {
        iterator = colecao.iterator();
        while(iterator.hasNext()){
            this.exibaDomain(iterator.next());
        }
    }
    public abstract void exibaDomain(Object parametro);
}

```

3.9.7 AssuntoDAOJdoTest

Para que um teste mantenha seu valor durante o tempo é necessário planejar bem sua construção. Dessa forma, após construir a primeira classe para testar DAO, o esforço para construir novos testes é muito baixo. Para que a criação e execução de testes em JUnit, é necessário que os métodos de teste seguissem o padrão `testXXX()`.

Classe AssuntoDAOJdoTest

```
public class AssuntoDAOJdoTest extends BaseDAOTestCase {
    private AssuntoDAOJdo dao;
    private Assunto domain;
    private Long id = null;

    protected void setUp() throws Exception {
        dao = (AssuntoDAOJdo) ctx.getBean("assunto");
    }
    public void testSave() throws Exception {
        System.out.println("SAVE");

        this.inserir("incluido");
        this.inserir("incluido");
        this.inserir("incluido2");

        this.exibaColecao(dao.getAll(ordenacao));
        System.out.println("");
    }

    public void testUpdate() throws Exception {
        System.out.println("UPDATE");

        this.atualizar("incluido", "atualizado");
        this.atualizar("incluido2", "atualizado");

        this.exibaColecao(dao.getAll(ordenacao));
        System.out.println("");
    }

    public void testDelete() throws Exception {
        System.out.println("DELETE");

        dao.delete((BaseDomain) dao.pesquisarPorNome("atualizado",
                                                    null).toArray()[0]);
        dao.delete((BaseDomain) dao.pesquisarPorNome("incluido2",
                                                    null).toArray()[0]);

        this.exibaColecao(dao.getAll(ordenacao));
        System.out.println("");
    }

    public void inserir(String nome) throws Exception {
        domain = new Assunto();
        domain.setNome(nome);
        this.salvar();
    }

    public void atualizar(String antigo, String novo) throws Exception {
        domain = (Assunto)dao.pesquisarPorNome(antigo,null).toArray()[0];
        domain.setNome(novo);
        this.salvar();
    }
}
```

```

public void salvar() throws Exception {
    try {
        dao.save(domain);
    } catch (Exception e){
        assertEquals(AssuntoDAOJdo.REGISTRO_JA_EXISTE,
                    e.getMessage());
    }
}

public void exibaDomain(Object parametro) {
    domain = (Assunto) parametro;
    System.out.println(domain.getId().longValue());
    System.out.println(domain.getNome());
}
}

```

O método `setUp` serve para criar o DAO que será testado. Esta é a única forma de compartilhar objetos entre os diversos métodos de teste. Neste caso o DAO obtido do Spring já está encapsulado em uma transação.

O método `testSave` realiza o teste do método `save` de `AssuntoDAOJdo`, tentando realizar três *inserts* no banco de dados, pelo menos um deles lançará exceção. Se a exceção for `AssuntoDAOJdo.REGISTRO_JA_EXISTE` o teste passará pois há um `assertEquals` que compara a mensagem de exceção a esta constante, caso contrário se a exceção for diferente será lançada a exceção de `AssertionFailureException`. O método `testUpdate` é bastante semelhante e também pelo menos um lançará uma exceção que será verificada por `assert`.

O `testDelete` fará o teste do método `delete`, para que ao final da execução deste teste o Banco de Dados retorne ao estado inicial antes da execução dos testes. Paralelamente estão sendo testados métodos de pesquisa `getAll`, `pesquisarPorNome` e dentro do método `delete` está sendo testado o método `getDomain`.

Como foi possível observar durante este capítulo é bastante fácil utilizar a implementação de referência JDO, JPOX, além proporcionar grande reuso de código e vantagens não encontradas em outros *frameworks*. No próximo capítulo são esclarecidas as classes de domínio que são parte do Sistema Informatizado para Gerenciamento de Ouvidorias.

4 Desenvolvimento da Camada de Persistência

Após a explanação sobre tecnologias utilizadas neste trabalho, neste capítulo são explicadas as classes envolvidas no desenvolvimento da camada de persistência, iniciando pelo levantamento dos requisitos. Logo em seguida, demonstra-se cada uma das classes de domínio passíveis de persistência.

4.1 Levantamento dos Requisitos de Persistência

Para o desenvolvimento da camada de persistência foram levantados os seguintes requisitos:

- Inserir, atualizar, excluir e consultar a entidade de assunto de manifestação que contém um nome que não pode ser repetido.
- Inserir, atualizar, excluir e consultar a entidade cargo de um funcionário que contém um nome que não pode ser repetido;
- Inserir, atualizar, excluir e consultar a entidade cidade, que contém um nome e um estado. Não podendo existir duas ou mais cidades com mesmo nome no mesmo estado.
- Inserir, atualizar, excluir e consultar a entidade modo. O modo terá um nome, que não se repetir e identifica como o manifestante chegou a Ouvidoria para se manifestar, bem como a forma desejada para envio da resposta. Há modos pode ser retornados e outros que não podem, deve haver uma consulta também por este campo.
- Inserir, atualizar, excluir e consultar datas de feriados e dias não úteis, estas datas devem ser únicas. Os feriados serão utilizados para calcular o prazo da manifestação, dessa forma deve haver a possibilidade de consultar entre um período tanto retornando todas as datas de feriado, bem como contando a quantidade de feriados daquele período, para a consulta pode ou não ser informada uma data inicial e/ou uma data final.
- Consultar classificações das manifestações, possíveis forma de identificação do Manifestante e também perfis de acesso ao sistema.
- Inserir, atualizar, excluir e consultar os prazos do sistema contendo: o nível, situação e a quantidade de dias úteis do prazo. Não podendo existir dois ou mais prazos com o mesmo nível e situação.
- Inserir, atualizar, excluir e consultar usuários do sistema: nome, apelido, *login*(único), senha, unidade, cargo, perfil de acesso, telefone e email.
- Inserir e consultar manifestações que contém data, hora, unidade, envolvidos e descrição da ocorrência do fato, caso o usuário não encontre a unidade é registrada uma outra unidade, neste cadastro também deve ser armazenado a forma de identificação do Manifestante, o modo de resposta desejado por ele, a classificação da manifestação e um assunto, da mesma forma caso o usuário não encontre o assunto desejado é registrado um outro assunto. Ao finalizar o cadastro é gravado um código de segurança para a manifestação e data prevista de conclusão da manifestação.
- O cadastro do Manifestante contém o nome, endereço, cidade, estado, telefones

e email, caso não encontre a cidade é registrada uma outra cidade.

- Registrar o recebimento da manifestação: data e hora do recebimento, modo de entrada, e ainda o usuário e seu cargo.
- Registrar o encaminhamento das manifestações registrando a data, data prevista para resposta, unidade de origem e destino e ainda o usuário e seu cargo.
- Registrar respostas às manifestações, contendo: data, origem, reposta, quantidade de dias úteis utilizados na resposta, unidade de origem e destino e ainda o usuário e seu cargo.
- Registrar comentários realizados pela Ouvidoria: data, comentário, data prevista para resposta, unidade de origem e destino e ainda o usuário e seu cargo.
- Registrar a conclusão de uma manifestação: data e hora da conclusão, unidade de origem e destino usuário, cargo e unidade responsável pela resposta, bem como a quantidade de dias úteis utilizados na manifestação;
- Fornecer uma consulta para informar a quantidade de cada uma das outras unidades, bem como fornecer um mecanismo para substituir estas outras unidades por unidades cadastradas, ao final do processo são excluídas as outras unidades.
- Fornecer uma consulta para informar a quantidade de cada um dos outros assuntos, bem como fornecer um mecanismo para substituir estes outros assuntos por assuntos cadastrados, ao final do processo são excluídos os outros assuntos.
- Fornecer uma consulta para informar a quantidade de cada uma das outras cidades, bem como fornecer um mecanismo para substituir estas outras cidades por cidades cadastradas, ao final do processo são excluídas as outras cidades.

4.2 Diagrama de Classes de Persistência

4.2.1 Diagrama de classes resumido

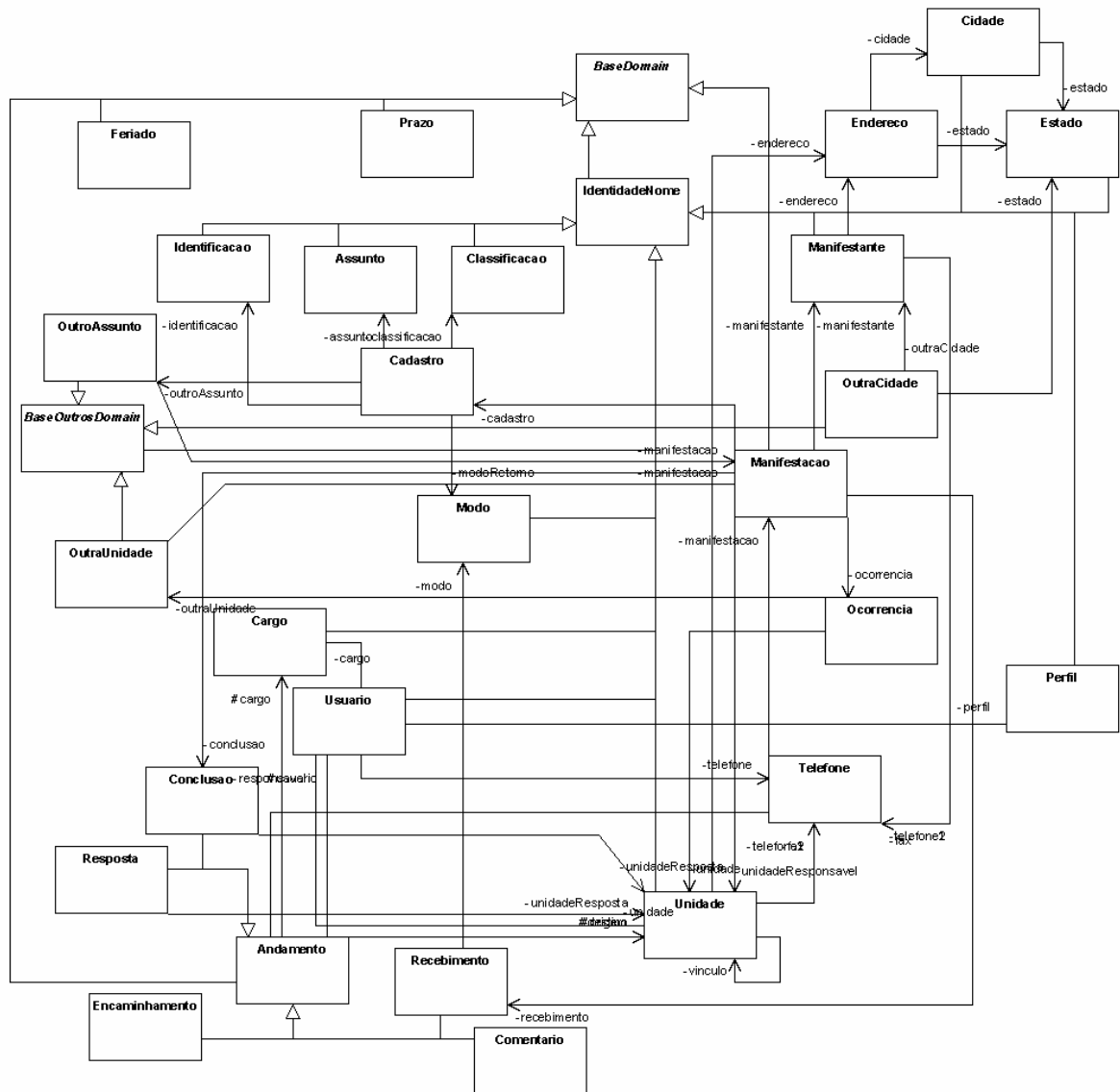


Figura 1 – Diagrama de Classes de Persistência Resumido

Como observado no item anterior, há grande quantidade de requisitos e também pelo grande número de associações entre os objetos do domínio do problema. Portanto, não será possível demonstrar os atributos, métodos e associações existentes num único diagrama de classes de domínio. Primeiro acima é apresentado um diagrama de classes resumido com as classes e suas associações. Logo em seguida é demonstrada cada uma das classes separadamente, com o detalhamento de seus atributos e métodos.

4.2.2 Classes detalhadas: Domínio do problema

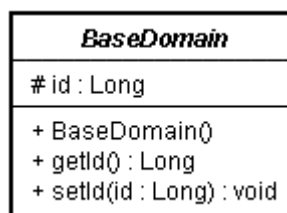


Figura 2 – BaseDomain

Esta superclasse abstrata foi construída pois grande parte das classes do domínio necessita de um atributo `id`, com o objetivo de representar as chaves primárias nas tabelas do banco de dados.

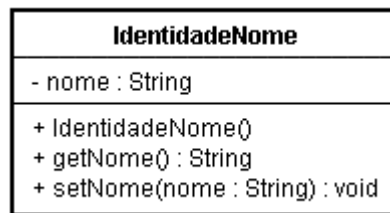


Figura 3 - IdentidadeNome

Diversas classes também do domínio necessitam de um atributo `nome`, assim, foi criada a classe `IdentidadeNome`. Com estas duas superclasses fornecidas neste diagrama de classe, já há grande reuso de código, pois sempre que uma classe precisar somente do `id` ou necessitar de um `id` e de um `nome`, ela obtenha estes atributos através de herança, seja de `BaseDomain` ou de `IdentidadeNome`.



Figura 4 - Andamento

Esta classe representa o que existe em comum entre os diversos tipos de andamentos. Um andamento sempre está associado a uma manifestação, possuindo também os seguintes atributos: data e hora, usuário, cargo, unidade de origem e destino.

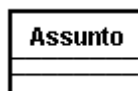


Figura 5 - Assunto

Esta classe representa o assunto de uma manifestação, como seus atributos já foram herdados de `IdentidadeNome`, não houve a necessidade de atributos e métodos adicionais.

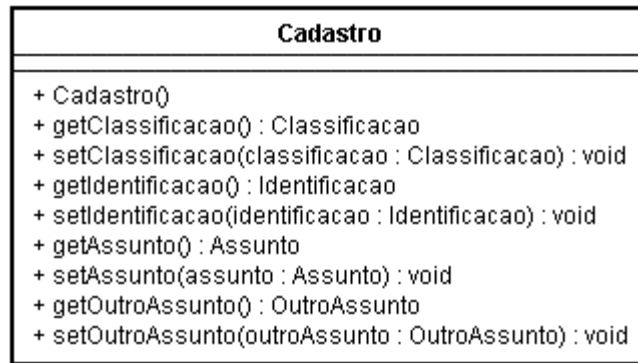


Figura 6 - Cadastro

Esta classe representa os dados do cadastro de uma manifestação, são eles: a classificação, a identificação escolhida pelo manifestante, o assunto e também o outro assunto que nem sempre será preenchido.

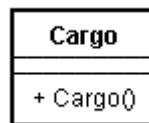


Figura 7 - Cargo

Esta classe representa o cargo de um funcionário. Seus atributos foram herdados de `IdentidadeNome`.

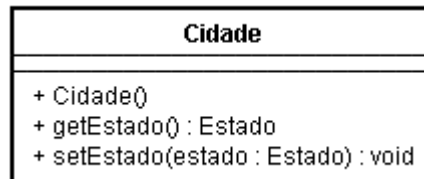


Figura 8 - Cidade

Esta classe representa uma cidade e possui um atributo da classe `Estado` e os outros atributos foram herdados de `IdentidadeNome`.

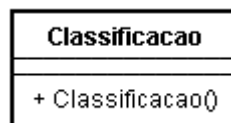


Figura 9 - Classificacao

Esta classe representa a classificação de uma manifestação, seus atributos foram herdados de `IdentidadeNome`.

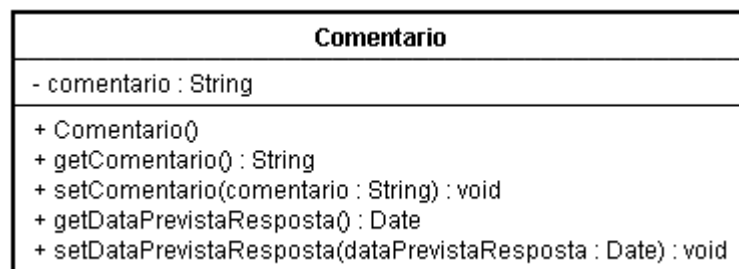


Figura 10 - Comentario

Esta classe herda `Andamento` e possui mais dois atributos comentário e a data prevista de resposta.

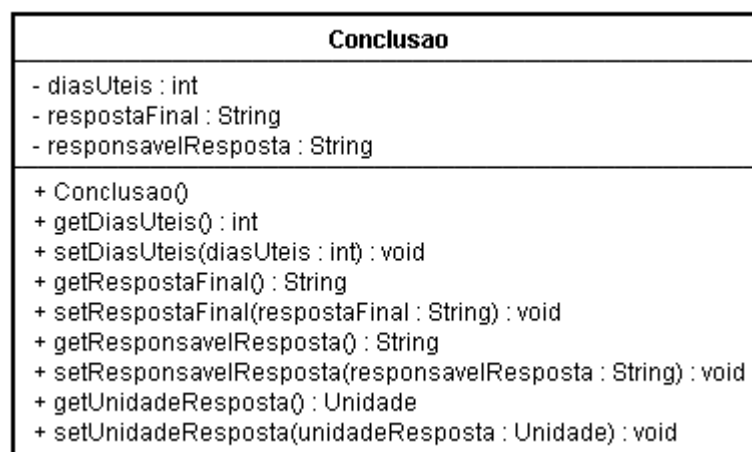


Figura 11 - Conclusao

Esta classe representa a finalização da manifestação. Ao ser finalizada a manifestação armazena a quantidade de dias úteis utilizada pela manifestação, a resposta final, o responsável e a unidade da resposta final além dos atributos herdados de `Andamento`.

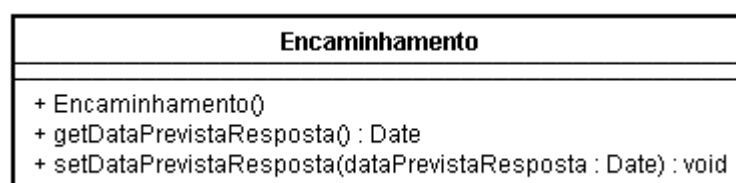


Figura 12 - Encaminhamento

Esta classe representa um encaminhamento de uma manifestação, além dos atributos de herdados da classe `Andamento`, também possui um atributo contendo a data prevista de resposta.

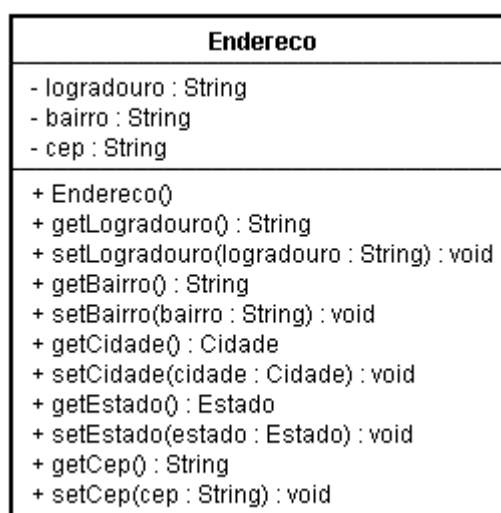


Figura 13 - Endereco

Esta classe representa um endereço contendo os atributos logradouro, bairro,

cidade, estado e cep.

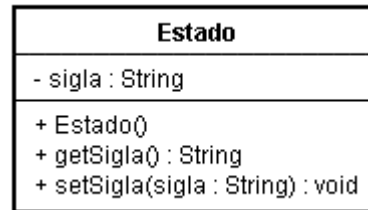


Figura 14 - Estado

Esta classe representa um estado, possuindo um atributo que sua sigla e os outros atributos foram herdados de `IdentidadeNome`.



Figura 15 - Feriado

Esta classe representa datas não-úteis e feriados, fazendo que o controle de prazos do Sistema passe sobre estas datas aos realizar algum cálculo, esta classe herda o atributo `id` de `BaseDomain`.

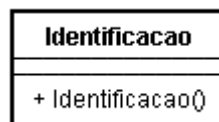


Figura 16 - Identificacao

Esta classe representa a forma com que um manifestante se identifica em uma manifestação, seus atributos foram herdados de `IdentidadeNome`.

Manifestacao
- nova : boolean - codigoSeguranca : Long
+ Manifestacao() + getOcorrencia() : Ocorrencia + setOcorrencia(ocorrencia : Ocorrencia) : void + getCadastro() : Cadastro + setCadastro(cadastro : Cadastro) : void + getRecebimento() : Recebimento + setRecebimento(recebimento : Recebimento) : void + getManifestante() : Manifestante + setManifestante(manifestante : Manifestante) : void + isNova() : boolean + setNova(nova : boolean) : void + getDataUltimaModificacao() : Date + setDataUltimaModificacao(dataUltimaModificacao : Date) : void + getUnidadeResponsavel() : Unidade + setUnidadeResponsavel(unidadeResponsavel : Unidade) : void + getEncaminhamentos() : Collection + setEncaminhamentos(encaminhamentos : Collection) : void + getRespostas() : Collection + setRespostas(respostas : Collection) : void + getComentarios() : Collection + setComentarios(comentarios : Collection) : void + getConclusao() : Conclusao + setConclusao(conclusao : Conclusao) : void + getCodigoSeguranca() : Long + setCodigoSeguranca(codigoSeguranca : Long) : void

Figura 17 - Manifestacao

Esta classe representa uma manifestação, possui os dados da ocorrência do fato, dados de cadastro, dados do recebimento, manifestante se identificado, um atributo do tipo `boolean` informando se é uma manifestação nova ou não, uma data informando sua ultima modificação, bem como, a unidade que detêm no momento a manifestação. Além disso, a manifestação possui coleções de encaminhamentos, respostas e comentários e uma associação para a conclusão da manifestação. A classe de manifestação ainda herda o `id` de `BaseDomain` e possui um código de segurança para que não seja possível realizar consultas sequenciais nas manifestações.

Manifestante
- email : String
+ Manifestante() + getEndereco() : Endereco + setEndereco(endereco : Endereco) : void + getTelefone1() : Telefone + setTelefone1(telefone1 : Telefone) : void + getTelefone2() : Telefone + setTelefone2(telefone2 : Telefone) : void + getFax() : Telefone + setFax(fax : Telefone) : void + getEmail() : String + setEmail(email : String) : void + getOutraCidade() : OutraCidade + setOutraCidade(outraCidade : OutraCidade) : void

Figura 18 - Manifestante

Representa o cliente-cidadão que expressa manifestações sobre fatos ocorridos, podendo informar seu endereço ou seu(s) telefone(s) ou e-mail. Manifestante herda os atributos `id` e `nome` de `IdentidadeNome`.

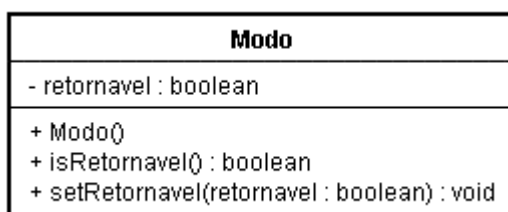


Figura 19 - Modo

Representa o modo de entrada e/ou de resposta de uma manifestação, quem diz um modo é retornável ou não é o atributo do tipo boolean. Modo também herda os atributos `id` e `nome` de `IdentidadeNome`.

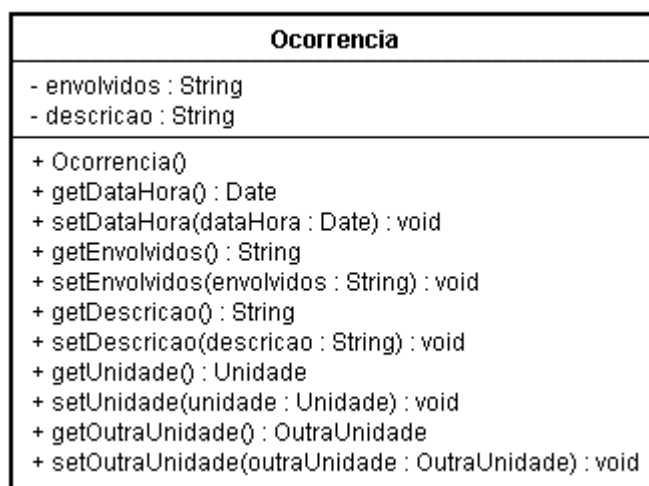


Figura 20 - Ocorrencia

Esta classe representa os dados da ocorrência do fato de uma manifestação são eles: a data e hora, os envolvidos, a descrição, a unidade e a outra unidade.

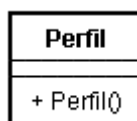


Figura 21 - Perfil

Representa os perfis de acesso disponíveis no sistema, esta classe herda `id` e `nome` de `IdentidadeNome`.



Figura 22 - Prazo

Esta classe representa um prazo existente no sistema, o prazo é formado pelo id herdado de `BaseDomain` e possui um atributo nível que corresponde ao nível hierárquico deste prazo dentro da estrutura da Ouvidoria, a situação é para qual condição de manifestação aquele prazo se refere e por ultimo a quantidade de dias úteis deste prazo.

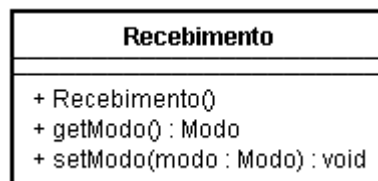


Figura 23 - Recebimento

Recebimento representa o momento em que a manifestação é recebida, sendo atribuídos os atributos herdados de `Andamento` e possui um atributo modo que representa o modo entrada da manifestação.

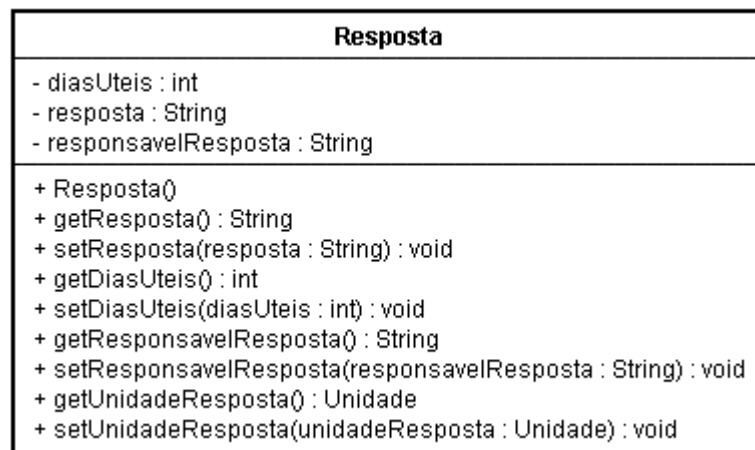


Figura 24 - Resposta

Esta classe representa o cadastro de resposta interna a manifestação, além de herdar atributos de `Andamento` possui a quantidade de dias úteis desde o recebimento até a resposta, a resposta e a unidade responsável pela resposta.

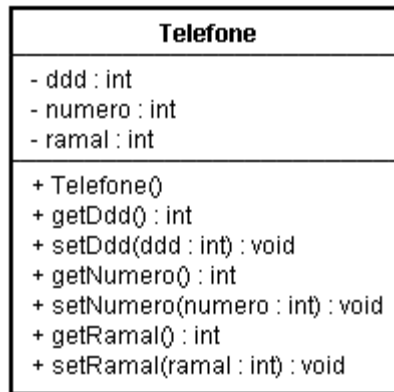


Figura 25 – Telefone

Representa um telefone possuindo um código de DDD, o número de telefone propriamente dito e um ramal.

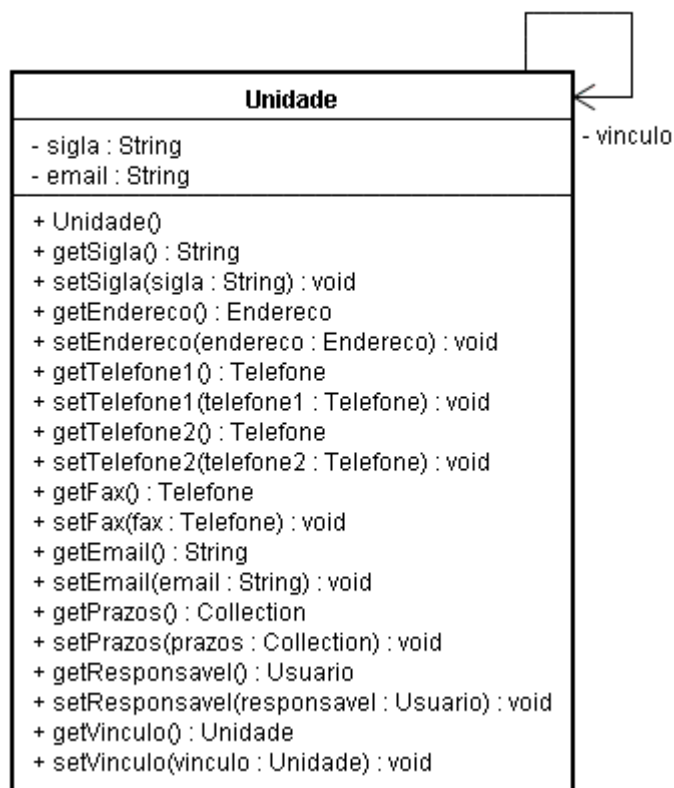


Figura 26 – Unidade

Representa uma unidade, setor, departamento da organização para o nível hierárquico de encaminhamentos das manifestações. A Unidade herda os atributos `id` e `nome` de `IdentidadeNome` e também possui uma sigla, um endereço, telefones, email, um usuário responsável, um vínculo, ou seja, ela pode ser vinculada à unidade superior para compor a estrutura hierárquica da Ouvidoria e ainda possui uma coleção de prazos.

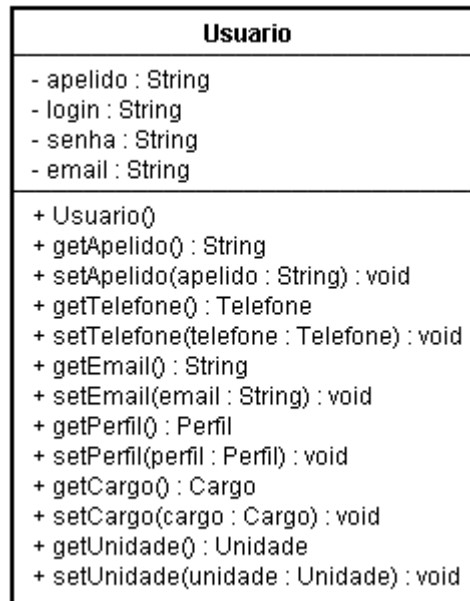


Figura 27 – Usuário

Representa um colaborador, funcionário que utiliza o Sistema da Ouvidoria, a classe herda o id e nome de *IdentidadeNome*. Possuindo ainda um apelido, um telefone, um email, perfil de acesso, cargo e unidade.

4.2.3 Classes detalhadas: Outros domínios

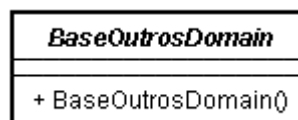


Figura 28 – BaseOutrosDomain

Esta classe representa os outros cadastros, proporcionando uma maior segurança na utilização dos métodos de substituição, já que a exclusão só pode ocorrer durante o processo de substituição. Esta classe abstrata herda o id e nome de *IdentidadeNome*.

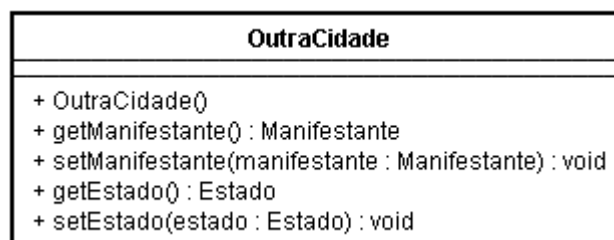


Figura 29 – OutraCidade

Esta classe representa o cadastro de outra cidade herdando os atributos id e nome de *BaseOutrosDomain*, além de possuir o atributo estado e manifestante.

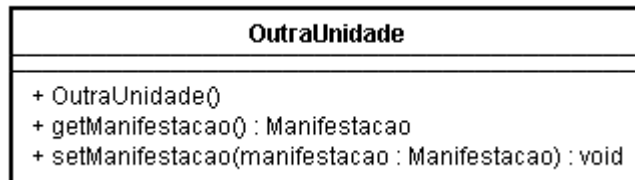


Figura 30 – OutraUnidade

Esta classe representa o cadastro de outra unidade herdando os atributos `id` e `nome` de `BaseOutrosDomain`, e também possui o atributo `manifestação`.

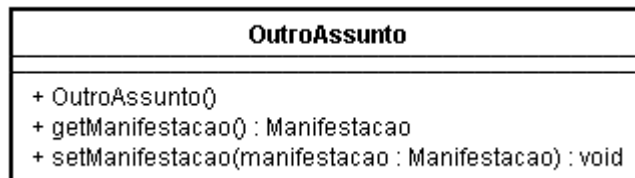


Figura 31 – OutroAssunto

Esta classe representa o cadastro de outro assunto herdando os atributos `id` e `nome` de `BaseOutrosDomain`, e também a qual `manifestação` pertence este outro assunto.

5 Comparação das Versões do Software

Depois de tudo o que foi visualizado neste trabalho, cabe neste instante a realização de uma análise comparativa entre as versões do Sistema Informatizado para Gerenciamento de Ouvidoria. Lembrando que a primeira versão do software foi desenvolvida utilizando JSP/Servlet e acessando diretamente MySQL com JDBC. Tal implementação, apesar de muito performática, possui grandes problemas de manutenção e também dificulta a adição de novas funcionalidades, pois há muitos códigos repetidos, bem como scripts de SQL “espalhados” pelo código fonte, ou seja, eles não estavam agrupados em classes que implementam o padrão de projeto DAO. Portanto, atingir o objetivo de Independência do Banco de Dados mostra-se uma tarefa bastante árdua, pois para trabalhar em um Banco de Dados diferente de MySQL, seria necessário “caçar” os scripts de SQL e com grande probabilidade criar *bugs* por descuido. Isto pode trazer custos muito altos de manutenção e ainda denegrir a imagem da empresa OMD que está se consolidando no mercado de Ouvidorias.

Por isso, partir para um *framework* de persistência como JPOX, que foi utilizado durante o trabalho, mostra-se muito atrativo, principalmente aliando-o aos conceitos de Orientação a Objetos e reuso de código. Obteve-se como resultado uma quantidade de código menor e também minimizou-se repetições, tornando a manutenção em seu código muito mais facilitada e muito mais segura, pois o JPOX não foi utilizado isoladamente, ele foi utilizado de forma integrada ao Spring, complementado ainda pela execução de testes automatizados utilizando JUnit.

A camada de persistência desenvolvida neste Trabalho de Conclusão de Curso é realmente independente do Banco de Dados, pois basta trocar as configurações de conexão ou *datasource* que a camada de persistência rodará em outro banco de dados relacional, conforme a última das premissas do Sistema Informatizado para Gerenciamento de Ouvidorias.

A adição de novas funcionalidades de persistência também será muito facilitada, pois bastará estender a estrutura atual, reimplementar apenas alguns métodos e reutilizar os métodos já implementados. Portanto, os custos e tempo de desenvolvimento serão muito menores, principalmente considerando-se o trabalho necessário para implementar métodos de *insert* e *update* utilizando JDBC. Ao analisar a implementação de métodos de *insert* e *update* verifica-se que ela mostra-se muito dispendiosa e de grande trabalho manual e, ainda, é necessário ser cuidadoso com a ordem das colunas nas inserções e nas atualizações, além de ser necessário “destrinchar” todo objeto antes de executar estas operações. Já utilizando um *framework* de persistência como JPOX, realiza-se a persistência de objetos de forma transparente, ou seja, ele cuida desta questão, bem como reduz em muito a probabilidade de erros cometidos pelo desenvolvedor, principalmente se utilizados métodos definidos em superclasses.

6 Considerações Finais

6.1 Conclusão

Ouvir o cliente-cidadão é muito importante, pois conhecendo suas necessidades, suas sugestões, suas reclamações e também seus elogios as organizações terão instrumentos e medidas para entrar em um ciclo de melhora contínua. É nesta área que a Ouvidoria atua, buscando a melhoria contínua das organizações e da satisfação de seu cliente-cidadão.

A tecnologia dá apoio necessário para a tabulação das informações, identificando áreas que precisem de maior atenção, através de relatórios e gráficos gerenciais. É neste contexto que está inserido o Sistema Informatizado para Gerenciamento para Ouvidorias e este Trabalho de Conclusão de Curso onde foi construída a camada de persistência que trouxe muito mais flexibilidade ao software podendo ser instalado em qualquer Banco de Dados Relacional. Conforme já citado na introdução deste trabalho: "É a organização que deve adaptar-se ao cliente, não o cliente necessita adaptar-se a organização".

Focando um pouco mais no aspecto técnico deste trabalho, a versão 2.0 da especificação JDO, que se encontra em estágio avançado para sua finalização, demonstrou grandes avanços se comparada à versão 1.0 da especificação. Se comparado às outras tecnologias e *frameworks* de persistência como Hibernate e EJB, a API JDO possui vantagens até mesmo em termos de performance, por causa do processo de *enhancer* (tempo de compilação). Este processo proporciona economia em tempo de execução se comparado a métodos de persistência que utilizam a API Reflection, a exemplo de Hibernate.

A definição dos mapeamentos mostrou-se muito simples, com diversas facilidades como a possibilidade do nome de tabelas, nomes e tipos de colunas serem encontradas automaticamente, caso haja correspondência de nome entre a classe e a tabela, bem como se houver correspondência de nome entre o atributo e a coluna. Reduzindo o tempo gasto com mapeamento objeto-relacional, e também reduzindo o tamanho do(s) arquivo(s) de mapeamento.

JPOX também mostrou-se muito eficiente no uso de funções de agregação e facilidades na obtenção de consultas para relatórios, sendo possível definir uma classe sem nenhum vínculo com os objetos de domínios e persistência.

E naturalmente, os conceitos de herança e polimorfismo oriundos da Orientação a Objetos, aliado a excelentes frameworks como o JPOX, Spring e JUnit, propiciaram uma camada de persistência com código fonte conciso, claro e testado. Dessa forma, ocorre redução nos custos envolvidos para manutenção do código e para adição de novas funcionalidades, além de atingir altos níveis de reusabilidade.

Portanto, na criação de novos projetos ou de novas versões de um software, utilize-se dos grandes benefícios da utilização deste trio de aliados, principalmente se utilizados de forma integrada, conforme demonstrado durante este trabalho.

6.2 Trabalhos Futuros

Como o Trabalho de Conclusão de Curso compreendeu a construção de uma camada de persistência, ainda é necessário adequar o restante do código de JSP/Servlets e dividi-lo em duas camadas: uma delas contendo a lógica e regras de

negócio e outra camada de apresentação *web*.

A camada lógica poderia ser construída utilizando os serviços disponibilizados pelo *framework* Spring, estudando e aplicando de forma mais detalhada o leque de recursos fornecidos por este excelente contêiner “leve”.

Já a camada de apresentação *web* poderia ser melhor estruturada utilizando algum dos *frameworks* existentes no mercado, baseados no padrão de projeto MVC. Struts em princípio, seria a primeira opção. No entanto, ele vem perdendo força e espaço. Outra alternativa é o SpringMVC, até mesmo porque as duas outras camadas estarão utilizando o *framework* Spring facilitando dessa forma sua integração. Essa camada também poderia ser implementada com JSF (Java Server Faces), tornando a aplicação mais iterativa, já que possui grande suporte a eventos, assemelhando-se a iteratividade de aplicações *desktop*. Seja qual for o *framework* escolhido na camada *web*, o estudo e aplicação de AJAX (Asynchronous JavaScript Technology and XML) também trará excelentes resultados.

Durante a finalização deste trabalho houve contatos com o autor deste trabalho, de interessados em dar continuidade a esta separação em camadas, pois, como resultado deste trabalho existe uma camada de persistência construída e testada. Assim, um novo trabalho poderia apenas utilizá-la e utilizar este tempo para estudar outros *frameworks* com maior profundidade, por exemplo.

Referências Bibliográficas

[ALVES JR 2002] ALVES JR., Mário Nelson. A contribuição estratégica das ouvidorias para a melhoria dos serviços prestados pelas organizações: Um estudo de caso na Secretaria de Estado da Saúde de Santa Catarina. Dissertação de Mestrado. UDESC, Florianópolis, 2002. Disponível em <www.omd.com.br/docs.htm>. Acesso em 05 de Dezembro de 2004.

[ALVES JR 2004] ALVES JR., Mário Nelson. Garantindo a efetividade da Ouvidoria. UDESC, Florianópolis, 2004. Disponível em <www.omd.com.br/docs.htm>. Acesso em 10 de Maio de 2005.

[CHAVES 2002] CHAVES, Rafael. Testando aplicações JAVA com JUnit. Abril de 2002.

[FARIAS&OLIVEIRA 2005] FARIAS, Henrique de Britto; OLIVEIRA, Roberta Lingnau de. Uma Ferramenta Introdutória à Formação de Pessoal na Implantação de CRM Baseado em Gerenciamento de Projetos Ágeis. UFSC, Florianópolis, 2005.

[GENESS] Centro GeNESS - Centro de Geração de Novos Empreendimentos em Software e Serviços. Disponível em: <<http://www.geness.ufsc.br>>. Acesso em 05 de Dezembro de 2004.

[JDOCENTRAL] JDOCentral. Disponível em: <<http://www.jdocentral.org>>. Acesso em 20 de Junho de 2005.

[JOHNSON 2005] JOHNSON, Rod. Expert Professional Java Development with the Spring Framework. EUA, Wrox, 2005.

[JOHNSON&HOELLER 2004] JOHNSON, Rod; HOELLER, Juergen. Expert one-on-one J2EE development without EJB. EUA, Wrox, 2004.

[JPOX] JPOX Java Persistence Objects. Disponível em: <<http://www.jpox.org>>. Acesso em 23 de Janeiro de 2006.

[LEITE 2004] LEITE, Maria Marta. Pressupostos para Implantação de Estratégias de Relacionamento com os Clientes em Pequenas e Médias Organizações: uma Abordagem baseada em Gerenciamento de Projetos. Tese (Doutorado em Engenharia de Produção). Curso de Pós-graduação em Engenharia de Produção. Universidade Federal de Santa Catarina, Florianópolis, 2004.

[MARAFON 2005] MARAFON, Tiago Antônio. Camada de Persistência em Java Estudo Comparativo entre EJB, JDO e Hibernate. Trabalho de Conclusão de Curso. UFSC,

Florianópolis, 2005.

[MASSOL&HUSTED 2005] MASSOL, Vincent; HUSTED, Ted. JUnit em Ação. Rio de Janeiro, Ed. Ciência Moderna, 2005.

[OMD 2004] OMD Soluções para Ouvidorias. Apostila do Curso de Capacitação de Ouvidores – Módulo I. Florianópolis. Novembro de 2004.

[OMD 2005 (1)] OMD Soluções para Ouvidorias. Disponível em <www.ond.com.br>. Acesso em 01 de Junho de 2005.

[OMD 2005 (2)] OMD Soluções para Ouvidorias. Plano de Negócios OMD Soluções para Ouvidorias. Florianópolis. Junho de 2005.

[OMD 2005 (3)] OMD Soluções para Ouvidorias. Apostila do Curso de Capacitação de Ouvidores – Módulo II. Florianópolis. Julho de 2005.

[RAIBLE 2004] RAIBLE, Matt. Spring Live. EUA, SourceBeat, 2004

[SPRING] Spring Framework. Disponível em: <<http://www.springframework.org>>. Acesso em 17 de Novembro de 2005.

[WALLS&BREIDENBACH 2005] WALLS, Craig; BREIDENBACH, Ryan. Spring in Action. EUA, Manning, 2005.

Anexos:

Persistência de Dados em Java: Um Estudo Aplicado ao Sistema Informatizado para Gerenciamento de Ouvidorias

Rony Reinehr Brand

Universidade Federal de Santa Catarina

Sistemas de Informação

rony@inf.ufsc.br

Resumo

O Sistema Informatizado para Gerenciamento de Ouvidorias possui três premissas para flexibilizar sua implantação: independência de Browser, independência do Sistema Operacional e independência do Banco de Dados. Entretanto, a última premissa ainda não estava resolvida. Para resolvê-la foi construída uma camada de persistência afim de isolar e desacoplar o código de persistência do restante da aplicação.

Para o desenvolvimento da camada de persistência foi estudada a versão 2.0 da especificação JDO e aplicada através de sua implementação de referência, o JPOX. Foram utilizados também os frameworks Spring - por fornecer diversas facilidades para construção de persistência e suporte a transações declarativas - e JUnit para criar testes automatizados das classes que implementam o padrão de projeto DAO.

A utilização dos frameworks supramencionados, os conceitos de herança e polimorfismo e também o isolamento da persistência em DAOs, proporcionaram altos níveis de reusabilidade, além de código fonte conciso, claro e testado. Desta forma, ocorre a redução dos custos para manutenção do código e para adição de novas funcionalidades, propiciando, ainda, maior flexibilidade para implantação do software.

Abstract

The Sistema Informatizado para Gerenciamento de Ouvidorias, a information system for Ombudsman management, that possess three premises to flexible your implantation: independence of Browser, independence of the Operational System and independence of the Data Base. However, the last premise was still not decided. To decide a persistence layer was constructed to isolate and to separate the code from persistence of remain of the application.

For the development of the persistence layer, the version 2.0 of specification JDO was studied and applied using its reference implementation, the JPOX. They had also

been used framework Spring - by supplying diverse easiness for persistence construction and support for declarative transactions - and framework JUnit to create automatically tests of the class that implement the design pattern DAO.

The use above mentioned about frameworks, the concepts of inheritance and polymorphism and also the isolation of the persistence in DAOs, had provided high levels of reusability beyond code source concise, clearly and tested. Therefore, happen the reduction of the costs for maintenance of the code and to addition of new functionalities, propitiating still greater flexibility for implantation of software.

1. Introdução

Dada a atual conjuntura política e social que demanda das organizações processos de mudanças e adaptação cada vez mais rápidos, possuir apenas dados armazenados não é mais suficiente e pode se tornar muito caro. Logo, há uma necessidade crescente de transformar esses dados em informação, auxiliando o processo de tomada de decisão e possibilitando as organizações mudanças mais rápidas. É neste cenário que estamos vivendo hoje, também conhecido como a Era da Informação.

Por outro lado, de acordo com Mello e Cunha (citado por Leite, 2004) os clientes estão cada vez mais exigentes, intolerantes à falhas e atrasos, motivados por uma concorrência cada vez mais acirrada e, também, por terem informações para uma melhor avaliação antes de uma aquisição. Para Leite (2004) isto exige que o cliente seja encantado, conquistado para que se torne fiel e também um parceiro, demonstrando sua satisfação. Os produtos e/ou serviços destas organizações passam a ter um valor diferenciado frente aos seus concorrentes e também seu preço passa a ter peso menor na aquisição do produto e/ou serviço.

Com o intuito de fidelizar e satisfazer seus clientes, muitas organizações estão mudando seu foco, antes voltado apenas ao lucro a qualquer custo. Agora estão

caminhando para um novo foco, o foco no cliente. Dada essa necessidade, o gerenciamento do relacionamento com o cliente ou apenas CRM (*Customer Relationship Management*) está em plena evidência.

Shani e Chalasani (citado por Leite, 2004) definiram CRM como: “um esforço integrado para identificar, construir e manter uma rede de relacionamentos com consumidores individuais e fortalecer continuamente esta rede com benefícios mútuos para ambos os lados através de contatos interativos, individualizados e com agregação de valor em um longo período de tempo”.

Complementarmente ao CRM, segundo Alves Jr. (2002), outra tendência que apresenta grande crescimento é o instituto das Ouvidorias (em inglês *Ombudsman*). Originalmente criado na Suécia por volta de 1809, chegou ao Brasil na segunda metade da década de 80. Entretanto, começou a ganhar expressividade a partir de 1995 com a criação da Associação Brasileira de Ouvidores/Ombudsman e também pela realização de Encontros Nacionais discutindo o tema.

1.1 Ouvidoria

Alves Jr. (2002) cita que, devido ao grande volume de serviços prestados, principalmente por organizações de médio e grande porte, alguns erros acabam passando despercebidos e se perpetuam devido à dificuldade da identificação destes problemas, causando insatisfação ou até mesmo fuga de seus clientes-cidadãos. Mas como corrigir estes erros e evitar que eles se repitam? É este questionamento que a Ouvidoria visa responder apoiado em processos organizacionais pré-estabelecidos e com prazos bem definidos.

A Ouvidoria visa à aproximação da relação entre a organização e seus clientes-cidadãos fornecendo um canal de comunicação objetivo, personalizado, a fim de ouvir quais as insatisfações, as sugestões e até mesmo os elogios apresentados pelos seus clientes. Dessa forma, o Ouvidor, funcionário de origem interna ou externa, que gerencia o funcionamento da Ouvidoria, passa ser a "voz" dos clientes-cidadãos na organização, apontando aos gestores da organização quais problemas estão ocorrendo e apresentando sugestões de melhoria [ALVES JR. 2002].

Conforme o número destas manifestações cresce, a organização passa a ter um banco de dados extremamente importante sobre o seu funcionamento, evidenciando os problemas vividos pelos seus clientes, sugestões apresentadas, entre outros. Entretanto, possuir este banco de dados não é suficiente.

Neste sentido, OMD 2004 alerta para a necessidade de a Ouvidoria possuir uma infra-estrutura adequada para atender a demanda de manifestações, bem como,

possuir um Sistema de Informação eficiente que gerencie as manifestações, controle os prazos e encaminhamentos dentro de uma manifestação. Além disso, é necessário que tal sistema possibilite a elaboração de relatórios e gráficos gerenciais. No entanto, nada disto adianta se não houverem profissionais bem capacitados para interpretar estes relatórios e gráficos transformando-os em informações valiosas sobre o funcionamento da organização. Por exemplo, em quais unidades, departamentos, setores da organização que existe maior frequência de reclamações, quais os assuntos mais frequentes de manifestações, entre outros.

1.2 Motivação

A partir dessa necessidade nasceu em 2003 a empresa OMD Soluções para Ouvidorias, com o objetivo de prestar consultoria e construir um sistema de apoio à área de Ouvidoria. O Sistema Informatizado para Gerenciamento de Ouvidoria, trata-se de uma excelente ferramenta gerencial que apoia as decisões do Ouvidor, afim de que este possa propor a melhorias necessárias para a melhoria contínua da organização.

O objetivo do software é ser o mais flexível possível quanto à infra-estrutura existente na organização onde será implantado. Para isto, o software possui três bases de sustentação: independência do navegador (*browser*), independência do sistema operacional e independência do banco de dados. As duas primeiras bases já estão resolvidas, uma através de testes entre diferentes navegadores e outra através do contêiner *Servlet TomCat*. No entanto, a última premissa ainda não estava resolvida e muitas organizações não aceitam trabalhar com mais de um banco de dados.

Por isso, este trabalho realizou a construção de camada de persistência utilizando a especificação JDO versão 2.0 e sua implementação de referência, o JPOX.

2. JDO

Java Data Object, ou apenas JDO como é mais conhecido, é a especificação da comunidade (JCP) para implementação de persistência de objetos em Java. Em 1999, foi iniciado o desenvolvimento da API JDO tendo sua primeira versão lançada em 2002. O objetivo da API JDO é facilitar a construção de uma camada de persistência, fazendo com que a camada lógica independa se os dados são persistidos em arquivos, bancos de dados relacionais ou mesmo bancos de dados orientados a objetos [JDOCENTRAL].

Para que isto aconteça, os objetos persistidos são gerenciados por instâncias de *PersistenceManager*,

obtidos através de um `PersistenceManagerFactory`. Para persistir um objeto (seja novo ou já existente) utilizaremos o método `makePersistent` sendo executado em uma instância de `PersistenceManager`. A chamada deste método é “traduzida” em um `INSERT` ou `UPDATE`, entendido pelo banco de dados relacional escolhido.

No entanto, cabe ressaltar que em informática e tecnologia nada é “mágico”. Antes de chegarmos neste ponto, precisamos definir os metadados que representarão o mapeamento objeto-relacional, ou seja, como uma classe com seus atributos, associações e heranças são persistidas em apenas uma tabela ou em diversas tabelas. Também em tempo de compilação, deve-se fazer um enxerto (*enhancer*) no *bytecode*, fazendo com que as classes que representam o domínio do problema estendam `PersistenceCapable` e implementem alguns métodos, tornando possível sua persistência [JPOX].

Além disso, JDO possui a `JDOQL` (*JDO Query Language*), uma linguagem de consulta de objetos utilizada por todas as implementações JDO. Trata-se de uma linguagem muito poderosa permitindo realizar consultas bastante complexas, sem a necessidade de utilizar diretamente JDBC com SQL. Isto permite desenvolver código muito mais conciso e reusável, evitando repetições de códigos semelhantes que geram grandes impactos na manutenção de uma aplicação.

Uma de suas vantagens pode ser encontrada com relação aos resultados uma consulta, ao invés de serem retornados num array de objetos (`Object[]`), é possível informar qual a classe de retorno, bastando que haja correspondência do tipo e do nome de um atributo, diminuindo em muito o trabalho necessário principalmente em relatórios ou consultas que utilizam funções de agregação como `min`, `max` e `count`. Ao utilizar estas funções normalmente não o resultado da consulta não corresponde a objetos de uma classe de domínio. Dessa forma define-se uma classe de retorno, evitando o trabalho incomodo com `Object[]` a cada item de uma coleção de objetos [JPOX].

Para mais detalhes sobre a especificação JDO versão 2.0 e também para visualizar exemplos das vantagens comentadas durante este artigo, podem ser obtidos na documentação do JPOX e também o site `JDOCentral`.

2.1 Outros Frameworks: Spring e JUnit

Afim de proporcionar maior reuso de código, um código fonte mais conciso e também testado, foram utilizados também os *frameworks* Spring - por fornecer diversas facilidades para construção de persistência e

suporte a transações declarativas - e JUnit para criar testes automatizados das classes que implementam o padrão de projeto DAO (*Data Access Object*).

O framework Spring é um “container leve”, com uma série de serviços fornecidos. Apenas aqueles necessários a aplicação foram utilizados. Um de seus serviços mais conhecidos é a Inversão de Controle (*Inversion of Control - IoC*) ou Injeção de Dependências, possibilitando realizar configuração que são carregadas quando buscamos um bean definido no `applicationContext.xml`. Sua integração com persistência proporciona maior reuso de código e maior facilidade de uso em classes que seguem o padrão de projeto DAO [SPRING].

Para JDO ele fornece a classe `JdoTemplate` que encapsula o controle da classe `PersistenceManagerFactory` e também das instâncias de `PersistenceManager`, eliminando a necessidade de controlar as transações de forma programática..

Com objetivo de testar a camada de persistência utilizando melhores práticas de desenvolvimento, foi utilizado o *framework* JUnit para construção de testes automatizados, que mantenha seu valor ao longo do tempo, bem como proporciona a redução dos custos, utilizando princípios de reusabilidade [MASSOL&HUSTED 2005].

Chaves (2002), também resalta a importância de realizar testes automatizados: “Se um software não tem um teste automatizado, assume-se que ele não funciona - cabe ao desenvolvedor provar que ele funciona através de testes automatizado”.

3.1. Sistema Informatizado para Gerenciamento de Ouvidorias

O sistema informatizado para gerenciamento de Ouvidorias é inovador na sua área, por se tratar de um software totalmente web para gerenciamento da Ouvidoria, tornando possível ao Ouvidor possuir total controle das manifestações apresentadas à organização, desde seu recebimento, passando por seu encaminhamento para as áreas responsáveis, setores ou departamentos dentro da organização, resposta ao cliente-cidadão, avaliação da resposta e encerramento da manifestação.

Abaixo são apresentadas duas figuras para esclarecer a dimensão e complexidade do sistema, demonstrando a necessidade da utilização de framework de persistência como o JPOX. A Figura 1 apresenta de forma resumida as funcionalidades existentes no software:

5. Referências

[ALVES JR 2002] ALVES JR., Mário Nelson. **A contribuição estratégica das ouvidorias para a melhoria dos serviços prestados pelas organizações: Um estudo de caso na Secretaria de Estado da Saúde de santa catarina.** Dissertação de Mestrado. UDESC, Florianópolis, 2002. Disponível em <www.ond.com.br/docs.htm>. Acesso em 05 de Dezembro de 2004.

[CHAVES 2002] CHAVES, Rafael. **Testando aplicações JAVA com JUnit.** Abril de 2002.

[JDOCENTRAL] **JDOCentral.** Disponível em: <<http://www.jdocentral.org>>. Acesso em 20 de Junho de 2005.

[JOHNSON&HOELLER 2004] JOHNSON, Rod; HOELLER, Juergen. **Expert one-on-one J2EE development without EJB.** EUA, Wrox, 2004.

[JPOX] **JPOX Java Persistence Objects.** Disponível em: <<http://www.jpox.org>>. Acesso em 23 de Janeiro de 2006.

[LEITE 2004] LEITE, Maria Marta. **Pressupostos para Implantação de Estratégias de Relacionamento com os Clientes em Pequenas e Médias Organizações: uma Abordagem baseada em Gerenciamento de Projetos.** Tese (Doutorado em Engenharia de Produção). Curso de Pós-graduação em Engenharia de Produção. UFSC, Florianópolis, 2004.

[MASSOL&HUSTED 2005] MASSOL, Vincent; HUSTED, Ted. **JUnit em Ação.** Rio de Janeiro, Ed. Ciência Moderna, 2005.

[OMD 2004] OMD Soluções para Ouvidorias. **Apostila do Curso de Capacitação de Ouvidores – Módulo I.** Florianópolis. Novembro de 2004.

[SPRING] **Spring Framework.** Disponível em: <<http://www.springframework.org>>. Acesso em 17 de Novembro de 2005.

Código Fonte:

```
applicationContext-test.xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
'http://www.springframework.org/dtd/spring-beans.dtd'>
<beans>
  <!-- JPOX PersistenceManagerFactory -->
  <bean id="pmf"
class="org.springframework.orm.jdo.LocalPersistenceManagerFactoryBean">
    <!-- TODO Move this database spec from here to a DataSource -->
    <property name="jdoProperties">
      <props>
        <prop
key="javax.jdo.PersistenceManagerFactoryClass">org.jpox.PersistenceManagerFactor
yImpl</prop>
        <prop
key="javax.jdo.option.ConnectionURL">jdbc:mysql://localhost:3306/ouvidoria</prop
>
          <prop key="javax.jdo.option.ConnectionUserName">root</prop>
          <prop key="javax.jdo.option.ConnectionPassword">miseravel</prop>
          <prop
key="javax.jdo.option.ConnectionDriverName">com.mysql.jdbc.Driver</prop>
          <prop key="javax.jdo.option.NontransactionalRead">true</prop>
        </props>
      </property>
    </bean>

    <!-- Transaction manager for a single JPOX PMF (alternative to JTA) -->
    <bean id="transactionManager"
class="org.springframework.orm.jdo.JdoTransactionManager">
      <property name="persistenceManagerFactory" ref="pmf"/>
    </bean>

    <!-- Transaction manager that delegates to JTA (for a transactional JNDI
DataSource) -->
    <!--
    <bean id="transactionManager"
class="org.springframework.transaction.jta.JtaTransactionManager"/>
    -->

    <!--Definição dos Beans Abstratos-->
    <bean id="DAO"
class="br.com.ond.ouvidoria.persistencia.geral.dao.jdo.DAOJdo" abstract="true">
      <property name="persistenceManagerFactory" ref="pmf"/>
    </bean>

    <bean id="baseDAO"
class="br.com.ond.ouvidoria.persistencia.geral.dao.jdo.BaseDAOJdo"
abstract="true" parent="DAO"/>
    <bean id="identidadeNomeDAO"
class="br.com.ond.ouvidoria.persistencia.geral.dao.jdo.IdentidadeNomeDAOJdo"
parent="baseDAO"/>

    <bean id="baseOutrosDAO"
class="br.com.ond.ouvidoria.persistencia.geral.outros.dao.jdo.BaseOutrosDAOJdo"
abstract="true"
parent="DAO"/>
    <!--Definição dos Beans Abstratos-->

    <!--Definição da Transação Abstrata-->
```

```

    <bean id="abstractTxDefinition"
class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean"
abstract="true">
    <property name="transactionManager" ref="transactionManager"/>
    <property name="transactionAttributes">
        <props>
            <prop key="get*">PROPAGATION_REQUIRED,readOnly</prop>
            <prop key="pesquisar*">PROPAGATION_REQUIRED,readOnly</prop>
            <prop key="save*">PROPAGATION_REQUIRED</prop>
            <prop key="delete*">PROPAGATION_REQUIRED</prop>
            <prop key="*">PROPAGATION_REQUIRED,readOnly</prop>
        </props>
    </property>
</bean>
<!-- Definição da Transação Abstrata -->

<!-- Somente Consulta -->
<bean id="somenteLeitura" parent="abstractTxDefinition">
    <property name="target" ref="identidadeNomeDAO"/>
</bean>
<!-- Somente Consulta -->

<!-- Assunto -->
<bean id="assuntoDAO"
class="br.com.ond.ouvidoria.persistencia.geral.dao.jdo.AssuntoDAOJdo"
parent="identidadeNomeDAO"/>
    <bean id="assunto" parent="abstractTxDefinition">
        <property name="target" ref="assuntoDAO"/>
    </bean>
<!-- Assunto -->

<!-- Cargo -->
<bean id="cargoDAO"
class="br.com.ond.ouvidoria.persistencia.geral.dao.jdo.CargoDAOJdo"
parent="identidadeNomeDAO"/>
    <bean id="cargo" parent="abstractTxDefinition">
        <property name="target" ref="cargoDAO"/>
    </bean>
<!-- Cargo -->

<!-- Cidade -->
<bean id="cidadeDAO"
class="br.com.ond.ouvidoria.persistencia.geral.dao.jdo.CidadeDAOJdo"
parent="identidadeNomeDAO"/>
    <bean id="cidade" parent="abstractTxDefinition">
        <property name="target" ref="cidadeDAO"/>
    </bean>
<!-- Cidade -->

<!-- Feriado -->
<bean id="feriadoDAO"
class="br.com.ond.ouvidoria.persistencia.geral.dao.jdo.FeriadoDAOJdo"
parent="baseDAO"/>
    <bean id="feriado" parent="abstractTxDefinition">
        <property name="target" ref="feriadoDAO"/>
    </bean>
<!-- Feriado -->

<!-- Modo -->
<bean id="modoDAO"
class="br.com.ond.ouvidoria.persistencia.geral.dao.jdo.ModoDAOJdo"
parent="baseDAO"/>

```

```

<bean id="modo" parent="abstractTxDefinition">
  <property name="target" ref="modoDAO"/>
</bean>
<!-- Modo -->

<!-- OutraUnidade -->
<bean id="outraUnidadeDAO"
class="br.com.omd.ouvidoria.persistencia.geral.outros.dao.jdo.OutraUnidadeDAOJdo
"
  parent="baseOutrosDAO"/>
<bean id="outraUnidade" parent="abstractTxDefinition">
  <property name="target" ref="outraUnidadeDAO"/>
</bean>
<!-- OutraUnidade -->

<!-- OutroAssunto -->
<bean id="outroAssuntoDAO"
class="br.com.omd.ouvidoria.persistencia.geral.outros.dao.jdo.OutroAssuntoDAOJdo
"
  parent="baseOutrosDAO"/>
<bean id="outroAssunto" parent="abstractTxDefinition">
  <property name="target" ref="outroAssuntoDAO"/>
</bean>
<!-- OutroAssunto -->
</beans>

```

Andamento.java

```

/*
 * Andamento.java
 *
 * Created on 8 de Janeiro de 2006, 16:15
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

```

```
package br.com.omd.ouvidoria.logica.domain;
```

```
import java.util.Date;
```

```
/**
```

```
*
```

```
* @author Rony Reinehr Brand
```

```
*/
```

```
public class Andamento extends BaseDomain {
```

```

  private Date dataHora;
  protected Usuario usuario;
  protected Cargo cargo;
  protected Unidade origem;
  protected Unidade destino;
  private Manifestacao manifestacao;

```

```
/** Creates a new instance of Andamento */
```

```
public Andamento() {
}
```

```
public Date getDataHora() {
  return dataHora;
}
```

```

public void setDataHora(Date dataHora) {
    this.dataHora = dataHora;
}

public Usuario getUsuario() {
    return usuario;
}

public void setUsuario(Usuario usuario) {
    this.usuario = usuario;
}

public Cargo getCargo() {
    return cargo;
}

public void setCargo(Cargo cargo) {
    this.cargo = cargo;
}

public Unidade getOrigem() {
    return origem;
}

public void setOrigem(Unidade origem) {
    this.origem = origem;
}

public Unidade getDestino() {
    return destino;
}

public void setDestino(Unidade destino) {
    this.destino = destino;
}

public Manifestacao getManifestacao() {
    return manifestacao;
}

public void setManifestacao(Manifestacao manifestacao) {
    this.manifestacao = manifestacao;
}
}

```

Assunto.java

```

/*
 * Assunto.java
 *
 * Created on 6 de Julho de 2005, 21:05
 */

package br.com.ond.ouvidoria.logica.domain;

/**
 *
 * @author Rony Reinehr Brand
 */
public class Assunto extends IdentidadeNome {

```

```
}
```

```
BaseDomain.java
```

```
/*  
 * BaseDomain.java  
 *  
 * Created on 20 de Setembro de 2005, 23:57  
 */
```

```
package br.com.ond.ouvidoria.logica.domain;
```

```
/**  
 *  
 * @author Desenvolvimento  
 */  
public class BaseDomain {  
  
    protected Long id;  
  
    /** Creates a new instance of Identitidade */  
    public BaseDomain() {  
    }  
  
    public Long getId() {  
        return id;  
    }  
  
    public void setId(Long id) {  
        this.id = id;  
    }  
  
}
```

```
Cadastro.java
```

```
/*  
 * Cadastro.java  
 *  
 * Created on 8 de Janeiro de 2006, 17:05  
 *  
 * To change this template, choose Tools | Template Manager  
 * and open the template in the editor.  
 */
```

```
package br.com.ond.ouvidoria.logica.domain;
```

```
import br.com.ond.ouvidoria.logica.domain.outros.OutroAssunto;
```

```
/**  
 *  
 * @author Rony Reinehr Brand  
 */  
public class Cadastro {  
  
    private Identificacao identificacao;  
    private Modo modoRetorno;  
    private Classificacao classificacao;  
    private Assunto assunto;  
    private OutroAssunto outroAssunto;
```

```

/**
 * Creates a new instance of Cadastro
 */
public Cadastro() {
}

public Classificacao getClassificacao() {
    return classificacao;
}

public void setClassificacao(Classificacao classificacao) {
    this.classificacao = classificacao;
}

public Identificacao getIdentificacao() {
    return identificacao;
}

public void setIdentificacao(Identificacao identificacao) {
    this.identificacao = identificacao;
}

public Assunto getAssunto() {
    return assunto;
}

public void setAssunto(Assunto assunto) {
    this.assunto = assunto;
}

public OutroAssunto getOutroAssunto() {
    return outroAssunto;
}

public void setOutroAssunto(OutroAssunto outroAssunto) {
    this.outroAssunto = outroAssunto;
}
}

```

Cargo.java

```

/*
 * Cargo.java
 *
 * Created on 8 de Janeiro de 2006, 14:51
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

```

```

package br.com.umd.ouvidoria.logica.domain;

```

```

/**
 *
 * @author Rony Reinehr Brand
 */
public class Cargo extends IdentidadeNome {

    /** Creates a new instance of Cargo */
    public Cargo() {

```

```
    }  
}
```

Cidade.java

```
/*  
 * Cargo.java  
 *  
 * Created on 8 de Janeiro de 2006, 14:51  
 *  
 * To change this template, choose Tools | Template Manager  
 * and open the template in the editor.  
 */
```

```
package br.com.omd.ouvidoria.logica.domain;
```

```
/**  
 *  
 * @author Rony Reinehr Brand  
 */  
public class Cargo extends IdentidadeNome {  
  
    /** Creates a new instance of Cargo */  
    public Cargo() {  
    }  
  
}
```

Classificacao.java

```
/*  
 * Classificacao.java  
 *  
 * Created on 6 de Julho de 2005, 21:06  
 */
```

```
package br.com.omd.ouvidoria.logica.domain;
```

```
/**  
 *  
 * @author Rony  
 */  
public class Classificacao extends IdentidadeNome{  
  
    /** Creates a new instance of Classificacao */  
    public Classificacao() {  
    }  
  
}
```

Comentário.java

```
/*  
 * Comentario.java  
 *  
 * Created on 8 de Janeiro de 2006, 16:33  
 *  
 * To change this template, choose Tools | Template Manager  
 * and open the template in the editor.  
 */
```

```

package br.com.ond.ouvidoria.logica.domain;

import br.com.ond.ouvidoria.util.Data;
import java.util.Date;

/**
 *
 * @author Rony Reinehr Brand
 */
public class Comentario extends Andamento {

    private Date dataPrevistaResposta;
    private String comentario;

    /** Creates a new instance of Comentario */
    public Comentario() {
    }

    public String getComentario() {
        return comentario;
    }

    public void setComentario(String comentario) {
        this.comentario = comentario;
    }

    public Date getDataPrevistaResposta() {
        return dataPrevistaResposta;
    }

    public void setDataPrevistaResposta(Date dataPrevistaResposta) {
        this.dataPrevistaResposta = dataPrevistaResposta;
    }

}

```

Conclusao.java

```

/*
 * Conclusao.java
 *
 * Created on 8 de Janeiro de 2006, 17:26
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

```

```

package br.com.ond.ouvidoria.logica.domain;

import br.com.ond.ouvidoria.util.Hora;

/**
 *
 * @author Rony Reinehr Brand
 */
public class Conclusao extends Andamento {

    private int diasUteis;
    private String respostaFinal;
    private String responsavelResposta;
    private Unidade unidadeResposta;

```

```

/** Creates a new instance of Conclusao */
public Conclusao() {
}

public int getDiasUteis() {
    return diasUteis;
}

public void setDiasUteis(int diasUteis) {
    this.diasUteis = diasUteis;
}

public String getRespostaFinal() {
    return respostaFinal;
}

public void setRespostaFinal(String respostaFinal) {
    this.respostaFinal = respostaFinal;
}

public String getResponsavelResposta() {
    return responsavelResposta;
}

public void setResponsavelResposta(String responsavelResposta) {
    this.responsavelResposta = responsavelResposta;
}

public Unidade getUnidadeResposta() {
    return unidadeResposta;
}

public void setUnidadeResposta(Unidade unidadeResposta) {
    this.unidadeResposta = unidadeResposta;
}
}

```

Encaminhamento.java

```

/*
 * Encaminhamento.java
 *
 * Created on 8 de Janeiro de 2006, 16:30
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

```

```

package br.com.omd.ouvidoria.logica.domain;

```

```

import java.util.Date;

```

```

/**
 *
 * @author Rony Reinehr Brand
 */

```

```

public class Encaminhamento extends Andamento {

```

```

    private Date dataPrevistaResposta;

```

```

    /** Creates a new instance of Encaminhamento */

```

```

public Encaminhamento() {
}

public Date getDataPrevistaResposta() {
    return dataPrevistaResposta;
}

public void setDataPrevistaResposta(Date dataPrevistaResposta) {
    this.dataPrevistaResposta = dataPrevistaResposta;
}
}

```

Endereco.java

```

/*
 * Endereco.java
 *
 * Created on 8 de Janeiro de 2006, 15:16
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

```

```

package br.com.ond.ouvidoria.logica.domain;

```

```

/**
 *
 * @author Rony Reinehr Brand
 */
public class Endereco {

    private String logradouro;
    private String bairro;
    private Cidade cidade;
    private Estado estado;
    private String cep;

    /** Creates a new instance of Endereco */
    public Endereco() {
    }

    public String getLogradouro() {
        return logradouro;
    }

    public void setLogradouro(String logradouro) {
        this.logradouro = logradouro;
    }

    public String getBairro() {
        return bairro;
    }

    public void setBairro(String bairro) {
        this.bairro = bairro;
    }

    public Cidade getCidade() {
        return cidade;
    }
}

```

```

public void setCidade(Cidade cidade) {
    this.cidade = cidade;
}

public Estado getEstado() {
    return estado;
}

public void setEstado(Estado estado) {
    this.estado = estado;
}

public String getCep() {
    return cep;
}

public void setCep(String cep) {
    this.cep = cep;
}
}

```

Estado.java

```

/*
 * Estado.java
 *
 * Created on 6 de Julho de 2005, 20:30
 */

package br.com.ond.ouvidoria.logica.domain;

/**
 *
 * @author Rony
 */
public class Estado extends IdentidadeNome {

    private String sigla;
    /** Creates a new instance of Estado */
    public Estado() {
    }

    public String getSigla() {
        return sigla;
    }

    public void setSigla(String sigla) {
        this.sigla = sigla;
    }

}

```

Feriado.java

```

/*
 * Estado.java
 *
 * Created on 6 de Julho de 2005, 20:30
 */

package br.com.ond.ouvidoria.logica.domain;

```

```

/**
 *
 * @author Rony
 */
public class Estado extends IdentidadeNome {

    private String sigla;
    /** Creates a new instance of Estado */
    public Estado() {
    }

    public String getSigla() {
        return sigla;
    }

    public void setSigla(String sigla) {
        this.sigla = sigla;
    }

}

```

IdentidadeNome.java

```

/*
 * IdentidadeNome.java
 *
 * Created on 20 de Setembro de 2005, 23:58
 */

```

```

package br.com.omd.ouvidoria.logica.domain;

```

```

/**
 *
 * @author Desenvolvimento
 */
public class IdentidadeNome extends BaseDomain {

    private String nome;

    /** Creates a new instance of IdentidadeNome */
    public IdentidadeNome() {
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

}

```

Identificacao.java

```

/*
 * Identificacao.java
 *
 * Created on 6 de Julho de 2005, 21:07
 */

```

```

package br.com.umd.ouvidoria.logica.domain;

/**
 *
 * @author Rony
 */
public class Identificacao extends IdentidadeNome {

    /** Creates a new instance of Identificacao */
    public Identificacao() {
    }

}

```

Manifestacao.java

```

package br.com.umd.ouvidoria.logica.domain;

import br.com.umd.ouvidoria.util.Data;
import java.util.Collection;
import java.util.Date;

public class Manifestacao extends BaseDomain {

    private Ocorrencia ocorrencia;
    private Cadastro cadastro;
    private Recebimento recebimento;
    private Manifestante manifestante;
    private boolean nova;
    private Date dataUltimaModificacao;
    private Unidade unidadeResponsavel;
    private Collection encaminhamentos;
    private Collection respostas;
    private Collection comentarios;
    private Conclusao conclusao;
    private Long codigoSeguranca;

    public Manifestacao() {
    }

    public Ocorrencia getOcorrencia() {
        return ocorrencia;
    }

    public void setOcorrencia(Ocorrencia ocorrencia) {
        this.occurencia = ocorrencia;
    }

    public Cadastro getCadastro() {
        return cadastro;
    }

    public void setCadastro(Cadastro cadastro) {
        this.cadastro = cadastro;
    }

    public Recebimento getRecebimento() {
        return recebimento;
    }

```

```

}

public void setRecebimento(Recebimento recebimento) {
    this.recebimento = recebimento;
}

public Manifestante getManifestante() {
    return manifestante;
}

public void setManifestante(Manifestante manifestante) {
    this.manifestante = manifestante;
}

public boolean isNova() {
    return nova;
}

public void setNova(boolean nova) {
    this.nova = nova;
}

public Date getDataUltimaModificacao() {
    return dataUltimaModificacao;
}

public void setDataUltimaModificacao(Date dataUltimaModificacao) {
    this.dataUltimaModificacao = dataUltimaModificacao;
}

public Unidade getUnidadeResponsavel() {
    return unidadeResponsavel;
}

public void setUnidadeResponsavel(Unidade unidadeResponsavel) {
    this.unidadeResponsavel = unidadeResponsavel;
}

public Collection getEncaminhamentos() {
    return encaminhamentos;
}

public void setEncaminhamentos(Collection encaminhamentos) {
    this.encaminhamentos = encaminhamentos;
}

public Collection getRespostas() {
    return respostas;
}

public void setRespostas(Collection respostas) {
    this.respostas = respostas;
}

public Collection getComentarios() {
    return comentarios;
}

public void setComentarios(Collection comentarios) {
    this.comentarios = comentarios;
}

```

```

public Conclusao getConclusao() {
    return conclusao;
}

public void setConclusao(Conclusao conclusao) {
    this.conclusao = conclusao;
}

public Long getCodigoSeguranca() {
    return codigoSeguranca;
}

public void setCodigoSeguranca(Long codigoSeguranca) {
    this.codigoSeguranca = codigoSeguranca;
}
}

```

Manifestante.java

```
package br.com.ond.ouvidoria.logica.domain;
```

```
import br.com.ond.ouvidoria.util.Data;
import java.util.Collection;
import java.util.Date;
```

```
public class Manifestacao extends BaseDomain {

    private Ocorrencia ocorrencia;
    private Cadastro cadastro;
    private Recebimento recebimento;
    private Manifestante manifestante;
    private boolean nova;
    private Date dataUltimaModificacao;
    private Unidade unidadeResponsavel;
    private Collection encaminhamentos;
    private Collection respostas;
    private Collection comentarios;
    private Conclusao conclusao;
    private Long codigoSeguranca;

    public Manifestacao() {
    }

    public Ocorrencia getOcorrencia() {
        return ocorrencia;
    }

    public void setOcorrencia(Ocorrencia ocorrencia) {
        this.ocorrencia = ocorrencia;
    }

    public Cadastro getCadastro() {
        return cadastro;
    }

    public void setCadastro(Cadastro cadastro) {
        this.cadastro = cadastro;
    }
}

```

```

}

public Recebimento getRecebimento() {
    return recebimento;
}

public void setRecebimento(Recebimento recebimento) {
    this.recebimento = recebimento;
}

public Manifestante getManifestante() {
    return manifestante;
}

public void setManifestante(Manifestante manifestante) {
    this.manifestante = manifestante;
}

public boolean isNova() {
    return nova;
}

public void setNova(boolean nova) {
    this.nova = nova;
}

public Date getDataUltimaModificacao() {
    return dataUltimaModificacao;
}

public void setDataUltimaModificacao(Date dataUltimaModificacao) {
    this.dataUltimaModificacao = dataUltimaModificacao;
}

public Unidade getUnidadeResponsavel() {
    return unidadeResponsavel;
}

public void setUnidadeResponsavel(Unidade unidadeResponsavel) {
    this.unidadeResponsavel = unidadeResponsavel;
}

public Collection getEncaminhamentos() {
    return encaminhamentos;
}

public void setEncaminhamentos(Collection encaminhamentos) {
    this.encaminhamentos = encaminhamentos;
}

public Collection getRespostas() {
    return respostas;
}

public void setRespostas(Collection respostas) {
    this.respostas = respostas;
}

public Collection getComentarios() {
    return comentarios;
}

```

```

public void setComentarios(Collection comentarios) {
    this.comentarios = comentarios;
}

public Conclusao getConclusao() {
    return conclusao;
}

public void setConclusao(Conclusao conclusao) {
    this.conclusao = conclusao;
}

public Long getCodigoSeguranca() {
    return codigoSeguranca;
}

public void setCodigoSeguranca(Long codigoSeguranca) {
    this.codigoSeguranca = codigoSeguranca;
}
}

```

Modo.java

```

/*
 * Modo.java
 *
 * Created on 6 de Julho de 2005, 20:57
 */

package br.com.umd.ouvidoria.logica.domain;

/**
 *
 * @author Rony
 */
public class Modo extends IdentidadeNome {

    private boolean retornavel;

    /** Creates a new instance of Modo */
    public Modo() {
    }

    public boolean isRetornavel() {
        return retornavel;
    }

    public void setRetornavel(boolean retornavel) {
        this.retornavel = retornavel;
    }

}

```

Ocorrencia.java

```

package br.com.umd.ouvidoria.logica.domain;

import br.com.umd.ouvidoria.logica.domain.outros.OutraUnidade;
import br.com.umd.ouvidoria.util.Data;

```

```

import br.com.ond.ouvidoria.util.Hora;
import java.util.Date;

/**
 * Write a description of class Manifestacao here.
 *
 * @author (Rony)
 * @version (a version number or a date)
 */

public class Ocorrencia {

    private Date dataHora;
    private String envolvidos;
    private String descricao;
    private Unidade unidade;
    private OutraUnidade outraUnidade;

    public Ocorrencia() {
    }

    public Date getDataHora() {
        return dataHora;
    }

    public void setDataHora(Date dataHora) {
        this.dataHora = dataHora;
    }

    public String getEnvolvidos() {
        return envolvidos;
    }

    public void setEnvolvidos(String envolvidos) {
        this.envolvidos = envolvidos;
    }

    public String getDescricao() {
        return descricao;
    }
}

```

```

public void setDescricao(String descricao) {
    this.descricao = descricao;
}

public Unidade getUnidade() {
    return unidade;
}

public void setUnidade(Unidade unidade) {
    this.unidade = unidade;
}

public OutraUnidade getOutraUnidade() {
    return outraUnidade;
}

public void setOutraUnidade(OutraUnidade outraUnidade) {
    this.outraUnidade = outraUnidade;
}
}

```

Perfil.java

```

/*
 * Perfil.java
 *
 * Created on 6 de Julho de 2005, 19:55
 */

package br.com.ond.ouvidoria.logica.domain;

/**
 *
 * @author Rony
 */
public class Perfil extends IdentidadeNome{

    /** Creates a new instance of Perfil */
    public Perfil() {

```

```
}
```

```
}
```

Prazo.java

```
/*
```

```
 * Prazo.java
```

```
 *
```

```
 * Created on 6 de Julho de 2005, 20:21
```

```
*/
```

```
package br.com.ond.ouvidoria.logica.domain;
```

```
/**
```

```
 *
```

```
 * @author Rony
```

```
*/
```

```
public class Prazo extends BaseDomain {
```

```
    private String nivel;
```

```
    private String situacao;
```

```
    private int prazo;
```

```
    /** Creates a new instance of Prazo */
```

```
    public Prazo() {  
    }  
}
```

```
    public String getNivel() {  
        return nivel;  
    }  
}
```

```
    public void setNivel(String nivel) {  
        this.nivel = nivel;  
    }  
}
```

```
    public String getSituacao() {  
        return situacao;  
    }  
}
```

```
public void setSituacao(String situacao) {
    this.situacao = situacao;
}

public int getPrazo() {
    return prazo;
}

public void setPrazo(int prazo) {
    this.prazo = prazo;
}
}
```

Recebimento.java

```
package br.com.ond.ouvidoria.logica.domain;

import br.com.ond.ouvidoria.util.Data;
import br.com.ond.ouvidoria.util.Hora;

public class Recebimento extends Andamento {

    private Modo modo;

    public Recebimento() {
    }

    public Modo getModo() {
        return modo;
    }

    public void setModo(Modo modo) {
        this.modo = modo;
    }
}
```

```

Resposta.java
/*
 * Resposta.java
 *
 * Created on 6 de Julho de 2005, 21:35
 */

package br.com.ond.ouvidoria.logica.domain;

import br.com.ond.ouvidoria.util.Data;

/**
 *
 * @author Rony
 */
public class Resposta extends Andamento {

    private int diasUteis;
    private String resposta;
    private String responsavelResposta;
    private Unidade unidadeResposta;

    /** Creates a new instance of Resposta */
    public Resposta() {
    }

    public String getResposta() {
        return resposta;
    }

    public void setResposta(String resposta) {
        this.resposta = resposta;
    }

    public int getDiasUteis() {
        return diasUteis;
    }

    public void setDiasUteis(int diasUteis) {
        this.diasUteis = diasUteis;
    }
}

```

```

public String getResponsavelResposta() {
    return responsavelResposta;
}

public void setResponsavelResposta(String responsavelResposta) {
    this.responsavelResposta = responsavelResposta;
}

public Unidade getUnidadeResposta() {
    return unidadeResposta;
}

public void setUnidadeResposta(Unidade unidadeResposta) {
    this.unidadeResposta = unidadeResposta;
}
}

```

Telefone.java

```

/*
 * Telefone.java
 *
 * Created on 8 de Janeiro de 2006, 15:34
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

```

```

package br.com.omd.ouvidoria.logica.domain;

```

```

/**
 *
 * @author Rony Reinehr Brand
 */

```

```

public class Telefone {

    private int ddd;
    private int numero;
    private int ramal;

```

```

    /** Creates a new instance of Telefone */
    public Telefone() {
    }

    public int getDdd() {
        return ddd;
    }

    public void setDdd(int ddd) {
        this.ddd = ddd;
    }

    public int getNumero() {
        return numero;
    }

    public void setNumero(int numero) {
        this.numero = numero;
    }

    public int getRamal() {
        return ramal;
    }

    public void setRamal(int ramal) {
        this.ramal = ramal;
    }
}

```

Unidade.java

```

/*
 * Unidade.java
 *
 * Created on 6 de Julho de 2005, 19:34
 */

package br.com.omd.ouvidoria.logica.domain;

import java.util.Collection;

```

```

/**
 *
 * @author Rony
 */
public class Unidade extends IdentidadeNome {

    private String sigla;
    private Endereco endereco;
    private Telefone telefonel;
    private Telefone telefone2;
    private Telefone fax;
    private String email;
    private Collection prazos;
    private Usuario responsavel;
    private Unidade vinculo;

    /** Creates a new instance of Unidade */
    public Unidade() {
    }

    public String getSigla() {
        return sigla;
    }

    public void setSigla(String sigla) {
        this.sigla = sigla;
    }

    public Endereco getEndereco() {
        return endereco;
    }

    public void setEndereco(Endereco endereco) {
        this.endereco = endereco;
    }

    public Telefone getTelefonel() {
        return telefonel;
    }
}

```

```
public void setTelefonel(Telefone telefonel) {
    this.telefonel = telefonel;
}

public Telefone getTelefone2() {
    return telefone2;
}

public void setTelefone2(Telefone telefone2) {
    this.telefone2 = telefone2;
}

public Telefone getFax() {
    return fax;
}

public void setFax(Telefone fax) {
    this.fax = fax;
}

public String getEmail() {
    return email;
}

public void setEmail(String email) {
    this.email = email;
}

public Collection getPrazos() {
    return prazos;
}

public void setPrazos(Collection prazos) {
    this.prazos = prazos;
}

public Usuario getResponsavel() {
    return responsavel;
}
```

```

public void setResponsavel(Usuario responsavel) {
    this.responsavel = responsavel;
}

public Unidade getVinculo() {
    return vinculo;
}

public void setVinculo(Unidade vinculo) {
    this.vinculo = vinculo;
}
}

```

Usuario.java

```

package br.com.ond.ouvidoria.logica.domain;

```

```

/**
 * Write a description of class Usuario here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Usuario extends IdentidadeNome {

    private String apelido;
    private String login;
    private String senha;
    private Telefone telefone;
    private String email;
    private Perfil perfil;
    private Cargo cargo;
    private Unidade unidade;

    /**
     * Constructor for objects of class Usuario
     */
    public Usuario() {
    }
}

```

```
public String getApellido() {
    return apellido;
}

public void setApellido(String apellido) {
    this.apellido = apellido;
}

public Telefone getTelefone() {
    return telefone;
}

public void setTelefone(Telefone telefone) {
    this.telefone = telefone;
}

public String getEmail() {
    return email;
}

public void setEmail(String email) {
    this.email = email;
}

public Perfil getPerfil() {
    return perfil;
}

public void setPerfil(Perfil perfil) {
    this.perfil = perfil;
}

public Cargo getCargo() {
    return cargo;
}

public void setCargo(Cargo cargo) {
    this.cargo = cargo;
}
```

```

public Unidade getUnidade() {
    return unidade;
}

public void setUnidade(Unidade unidade) {
    this.unidade = unidade;
}
}

```

package.jdo

```

<?xml version="1.0"?>
<!DOCTYPE jdo PUBLIC "-//Sun Microsystems, Inc.//DTD Java Data Objects Metadata
1.0//EN"
    "http://java.sun.com/dtd/jdo_1_0.dtd">
<jdo>
  <package name="br.com.ond.ouvidoria.logica.domain">
    <class name="BaseDomain" detachable="true" identity-type="application">
      <inheritance strategy="subclass-table"/>
      <field name="id" primary-key="true" value-strategy="autoassign"/>
    </class>
    <class name="IdentidadeNome" detachable="true">
      <inheritance strategy="subclass-table"/>
      <field name="nome">
        <column length="100"/>
      </field>
    </class>
    <class name="Assunto" table="assunto">
      <inheritance strategy="new-table"/>
    </class>
    <class name="Cargo" table="cargo">
      <inheritance strategy="new-table"/>
    </class>
    <class name="Classificacao" table="classificacao">
      <inheritance strategy="new-table"/>
    </class>
    <class name="Identificacao" table="identificacao">
      <inheritance strategy="new-table"/>
    </class>
    <class name="Modo" table="modo">

```

```

    <inheritance strategy="new-table"/>
    <field name="retornavel" >
        <column length="1" jdbc-type="char"/>
    </field>
</class>
<class name="Estado" table="estado">
    <inheritance strategy="new-table"/>
    <field name="sigla">
        <column length="2"/>
    </field>
</class>
<class name="Cidade" table="cidade">
    <inheritance strategy="new-table"/>
    <field name="estado" persistence-modifier="persistent">
        <column name="estado"/>
    </field>
    <fetch-group name="detach_estado">
        <field name="estado"/>
    </fetch-group>
</class>
<class name="Feriado" table="feriado">
    <inheritance strategy="new-table"/>
    <field name="data"/>
</class>
<class name="Unidade" table="unidade">
    <inheritance strategy="new-table"/>
</class>
<class name="Manifestacao" table="manifestacao">
    <inheritance strategy="new-table"/>
    <field name="nova" >
        <column length="1" jdbc-type="char"/>
    </field>
    <field name="unidadeOcorrencia" persistence-modifier="persistent">
        <column name="unidade_ocorrencia"/>
    </field>
    <field name="outraUnidade" persistence-modifier="persistent" mapped-
by="manifestacao"/>
    <field name="assunto" persistence-modifier="persistent">
        <column name="assunto"/>
    </field>
    <field name="outroAssunto" persistence-modifier="persistent" mapped-
by="manifestacao"/>

```

```

    </class>
</package>
<package name="br.com.omd.ouvidoria.logica.domain.outros">
    <class name="BaseOutrosDomain">
        <inheritance strategy="subclass-table"/>
    </class>
    <class name="OutraUnidade" table="outra_unidade">
        <inheritance strategy="new-table"/>
        <field name="manifestacao" persistence-modifier="persistent">
            <column name="manifestacao"/>
        </field>
    </class>
    <class name="OutroAssunto" table="outro_assunto">
        <inheritance strategy="new-table"/>
        <field name="manifestacao" persistence-modifier="persistent">
            <column name="manifestacao"/>
        </field>
    </class>
</package>
</jdo>

```

BaseOutrosDomain.java

```

/*
 * BaseOutrosDomain.java
 *
 * Created on 22 de Janeiro de 2006, 14:54
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

```

```

package br.com.omd.ouvidoria.logica.domain.outros;

import br.com.omd.ouvidoria.logica.domain.IdentidadeNome;

/**
 *
 * @author Desenvolvimento
 */
public abstract class BaseOutrosDomain extends IdentidadeNome {

```

```

    /**
     * Creates a new instance of BaseOutrosDomain
     */
    public BaseOutrosDomain() {
    }

}

OutraCidade.java
/*
 * Assunto.java
 *
 * Created on 6 de Julho de 2005, 21:05
 */

package br.com.ond.ouvidoria.logica.domain.outros;

import br.com.ond.ouvidoria.logica.domain.Estado;
import br.com.ond.ouvidoria.logica.domain.Manifestante;

/**
 *
 * @author Rony
 */
public class OutraCidade extends BaseOutrosDomain {

    private Manifestante manifestante;
    private Estado estado;

    /** Creates a new instance of Assunto */
    public OutraCidade() {
    }

    public Manifestante getManifestante() {
        return manifestante;
    }

    public void setManifestante(Manifestante manifestante) {
        this.manifestante = manifestante;
    }
}

```

```

    }

    public Estado getEstado() {
        return estado;
    }

    public void setEstado(Estado estado) {
        this.estado = estado;
    }
}

```

OutraUnidade.java

```

/*
 * OutraUnidade.java
 *
 * Created on 6 de Julho de 2005, 21:05
 */

package br.com.omd.ouvidoria.logica.domain.outros;

import br.com.omd.ouvidoria.logica.domain.Manifestacao;

/**
 *
 * @author Rony
 */
public class OutraUnidade extends BaseOutrosDomain {

    private Manifestacao manifestacao;
    /** Creates a new instance of Assunto */
    public OutraUnidade() {
    }

    public Manifestacao getManifestacao() {
        return manifestacao;
    }

    public void setManifestacao(Manifestacao manifestacao) {
        this.manifestacao = manifestacao;
    }
}

```

```
}
```

OutroAssunto.java

```
/*
```

```
 * OutroAssunto.java
```

```
 *
```

```
 * Created on 6 de Julho de 2005, 21:05
```

```
 */
```

```
package br.com.ond.ouvidoria.logica.domain.outros;
```

```
import br.com.ond.ouvidoria.logica.domain.Manifestacao;
```

```
/**
```

```
 *
```

```
 * @author Rony
```

```
 */
```

```
public class OutroAssunto extends BaseOutrosDomain {
```

```
    private Manifestacao manifestacao;
```

```
    /** Creates a new instance of Assunto */
```

```
    public OutroAssunto() {
```

```
    }
```

```
    public Manifestacao getManifestacao() {
```

```
        return manifestacao;
```

```
    }
```

```
    public void setManifestacao(Manifestacao manifestacao) {
```

```
        this.manifestacao = manifestacao;
```

```
    }
```

```
}
```

AssuntoDAO.java

```
/*
```

```
 * AssuntoDAO.java
```

```
 *
```

```
 * Created on 9 de Janeiro de 2006, 00:07
```

```
*
* To change this template, choose Tools | Template Manager
* and open the template in the editor.
*/
```

```
package br.com.ond.ouvidoria.persistencia.geral.dao;
```

```
/**
 *
 * @author Rony Reinehr Brand
 */
public interface AssuntoDAO extends IdentidadeNomeDAO {

}
```

BaseDAO.java

```
/*
 * BaseDAO.java
 *
 * Created on 9 de Janeiro de 2006, 00:04
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */
```

```
package br.com.ond.ouvidoria.persistencia.geral.dao;
```

```
import br.com.ond.ouvidoria.logica.domain.BaseDomain;
import java.util.Collection;
import org.springframework.dao.DataAccessException;
```

```
/**
 *
 * @author Rony Reinehr Brand
 */
public interface BaseDAO extends DAO {

    public void delete(BaseDomain domain) throws Exception;

    public void delete(Long id) throws Exception;
```

```
}
```

CargoDAO.java

```
/*
```

```
 * CargoDAO.java
```

```
 *
```

```
 * Created on 15 de Janeiro de 2006, 01:28
```

```
 *
```

```
 * To change this template, choose Tools | Template Manager
```

```
 * and open the template in the editor.
```

```
*/
```

```
package br.com.ond.ouvidoria.persistencia.geral.dao;
```

```
/**
```

```
 *
```

```
 * @author Rony Reinehr Brand
```

```
*/
```

```
public interface CargoDAO extends IdentidadeNomeDAO {
```

```
}
```

CidadeDAO.java

```
/*
```

```
 * CidadeDAO.java
```

```
 *
```

```
 * Created on 15 de Janeiro de 2006, 12:14
```

```
 *
```

```
 * To change this template, choose Tools | Template Manager
```

```
 * and open the template in the editor.
```

```
*/
```

```
package br.com.ond.ouvidoria.persistencia.geral.dao;
```

```
import br.com.ond.ouvidoria.logica.domain.BaseDomain;
```

```
import java.util.Collection;
```

```
/**
```

```

*
* @author Rony Reinehr Brand
*/
public interface CidadeDAO extends IdentidadeNomeDAO {

    public BaseDomain getDomain(Long id, boolean detachEstado) throws Exception;

    public Collection pesquisarPorEstado(Long idEstado) throws Exception;

    public Collection pesquisarPorNome(String nome, String ordenacao, boolean
detachEstado) throws Exception;

    public boolean isResult(String nomeCidade, Long idEstado) throws Exception;

    public Collection getAll(String ordenacao, boolean detachEstado) throws
Exception;

}

```

DAO.java

```

/*
* DAO.java
*
* Created on 23 de Janeiro de 2006, 21:10
*
* To change this template, choose Tools | Template Manager
* and open the template in the editor.
*/

package br.com.ond.ouvidoria.persistencia.geral.dao;

import br.com.ond.ouvidoria.logica.domain.BaseDomain;
import java.util.Collection;

/**
*
* @author Desenvolvimento
*/
public interface DAO {

    public Collection getAll(String ordering) throws Exception;

```

```

    public Collection getAll(Class classe, String ordering) throws Exception;

    public BaseDomain getDomain(Long id) throws Exception;

    public BaseDomain getDomain(Class classe, Long id) throws Exception;

    public Object save(BaseDomain domain) throws Exception;

}

```

FeriadoDAO.java

```

/*
 * FeriadoDAO.java
 *
 * Created on 15 de Janeiro de 2006, 16:27
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

package br.com.ond.ouvidoria.persistencia.geral.dao;

import java.util.Collection;
import java.util.Date;

/**
 *
 * @author Rony Reinehr Brand
 */
public interface FeriadoDAO extends BaseDAO {

    public Collection pesquisarPorData(Date data, String ordenacao) throws
Exception;

    public Collection pesquisarPorPeriodo(Date dataInicial, Date dataFinal,
String ordenacao) throws Exception;

    public Long getQuantidadePorPeriodo(Date dataInicial, Date dataFinal) throws
Exception;

```

```
        public boolean isResult(Date data) throws Exception;
    }
}
```

IdentidadeNomeDAO.java

```
/*
 * IdentidadeNomeDAO.java
 */

package br.com.omd.ouvidoria.persistencia.geral.dao;

import br.com.omd.ouvidoria.persistencia.geral.dao.BaseDAO;
import java.util.Collection;
import org.springframework.dao.DataAccessException;

/**
 *
 * @author Rony Reinehr Brand
 */
public interface IdentidadeNomeDAO extends BaseDAO {

    public Collection pesquisarPorNome(String nome, String ordenacao) throws
Exception;

    public boolean isResult(String nome) throws Exception;
}
}
```

ModoDAO.java

```
/*
 * ModoDAO.java
 *
 * Created on 16 de Janeiro de 2006, 19:14
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

package br.com.omd.ouvidoria.persistencia.geral.dao;
```

```

import java.util.Collection;

/**
 *
 * @author Rony Reinehr Brand
 */
public interface ModoDAO extends IdentidadeNomeDAO{

    public Collection pesquisarPorRetornavel(boolean retornavel, String
ordenacao) throws Exception;

}

```

AssuntoDAOJdo.java

```

/*
 * AssuntoDAOJdo.java
 *
 * Created on 8 de Janeiro de 2006, 21:00
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 *
 */

```

```

package br.com.ond.ouvidoria.persistencia.geral.dao.jdo;

```

```

import br.com.ond.ouvidoria.logica.domain.Assunto;
import br.com.ond.ouvidoria.logica.domain.BaseDomain;
import br.com.ond.ouvidoria.persistencia.geral.dao.AssuntoDAO;
import java.util.Collection;
import org.springframework.dao.DataAccessException;
import org.springframework.orm.jdo.support.JdoDaoSupport;

```

```

/**
 *
 * @author Rony Reinehr Brand
 */
public class AssuntoDAOJdo extends IdentidadeNomeDAOJdo implements AssuntoDAO {

    protected static final String REGISTRO_JA_EXISTE = "Assunto já existe.";

```

```

/** Creates a new instance of AssuntoDAOJdo */
public AssuntoDAOJdo() {
}

protected Class getReferenceClass() {
    return Assunto.class;
}

public Object save(BaseDomain domain) throws Exception {
    Assunto assunto = (Assunto) domain;
    try {
        if (assunto.getId() != null) {
            Assunto assuntoBD = (Assunto) getDomain(assunto.getId());
            if (! (assunto.getNome().equals(assuntoBD.getNome())))
                super.gereExcecaoSeExistir(assunto.getNome(),
REGISTRO_JA_EXISTE);
        }
        return super.save(domain);
    } catch (Exception e) {
        super.gereExcecaoSeExistir(assunto.getNome(), REGISTRO_JA_EXISTE);
        throw e;
    }
}
}

```

BaseDAOJdo.java

```

/*
 * BaseDAOJdo.java
 *
 * Created on 8 de Janeiro de 2006, 18:07
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

```

```

package br.com.ond.ouvidoria.persistencia.geral.dao.jdo;

import br.com.ond.ouvidoria.logica.domain.BaseDomain;
import br.com.ond.ouvidoria.persistencia.geral.dao.BaseDAO;

```

```

import java.util.Collection;

/**
 *
 * @author Rony Reinehr Brand
 */
public abstract class BaseDAOJdo extends DAOJdo implements BaseDAO {

    /** Creates a new instance of BaseDAOJdo */
    public BaseDAOJdo() {
    }

    public void delete(BaseDomain domain) throws Exception {
        this.delete(domain.getId());
    }

    public void delete(Long id) throws Exception {

getPersistenceManager().deletePersistent(getJdoTemplate().getObjectById(getReferenceClass(),id));
    }

}

```

CargoDAOJdo.java

```

/*
 * Cargo.java
 *
 * Created on 15 de Janeiro de 2006, 01:18
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

package br.com.ond.ouvidoria.persistencia.geral.dao.jdo;

import br.com.ond.ouvidoria.logica.domain.BaseDomain;
import br.com.ond.ouvidoria.logica.domain.Cargo;
import br.com.ond.ouvidoria.persistencia.geral.dao.CargoDAO;
import org.springframework.dao.DataAccessException;

```

```

/**
 *
 * @author Rony Reinehr Brand
 */
public class CargoDAOJdo extends IdentidadeNomeDAOJdo implements CargoDAO {
    protected static final String REGISTRO_JA_EXISTE = "Cargo já existe.";
    /** Creates a new instance of Cargo */
    public CargoDAOJdo() {
    }

    protected Class getReferenceClass() {
        return Cargo.class;
    }

    public Object save(BaseDomain domain) throws Exception {
        Cargo cargo = (Cargo) domain;
        try {
            if (cargo.getId() != null) {
                Cargo cargoBD = (Cargo) getDomain(cargo.getId());
                if (! (cargo.getNome().equals(cargoBD.getNome())))
                    super.gereExcecaoSeExistir(cargo.getNome(),
REGISTRO_JA_EXISTE);
            }
            return super.save(domain);
        } catch (Exception e) {
            super.gereExcecaoSeExistir(cargo.getNome(), REGISTRO_JA_EXISTE);
            throw e;
        }
    }

    public void delete(Long id) throws Exception {
        //Criar FKs para impedir exclusão
        super.delete(id);
    }
}

```

CidadeDAOJdo.java

```

/*
 * CidadeDAOJdo.java
 *
 * Created on 15 de Janeiro de 2006, 12:04
 *

```

```

* To change this template, choose Tools | Template Manager
* and open the template in the editor.
*/

package br.com.ond.ouvidoria.persistencia.geral.dao.jdo;

import br.com.ond.ouvidoria.logica.domain.BaseDomain;
import br.com.ond.ouvidoria.logica.domain.Cidade;
import br.com.ond.ouvidoria.logica.domain.Estado;
import br.com.ond.ouvidoria.persistencia.geral.dao.CidadeDAO;
import java.util.Collection;

/**
 *
 * @author Rony Reinehr Brand
 */
public class CidadeDAOJdo extends IdentidadeNomeDAOJdo implements CidadeDAO{

    protected static final String REGISTRO_JA_EXISTE = "Cidade já existe.";

    /** Creates a new instance of CidadeDAOJdo */
    public CidadeDAOJdo() {
    }

    protected Class getReferenceClass() {
        return Cidade.class;
    }

    public Object save(BaseDomain domain) throws Exception {
        Cidade cidade = (Cidade)domain;
        try {
            if (cidade.getId() !=null) {
                Cidade cidadeBD = (Cidade)getDomain(cidade.getId(), true);
                if (! (cidade.getNome().equals(cidadeBD.getNome())
                cidade.getEstado().getId().equals(cidadeBD.getEstado().getId())))
                    this.gereExcecaoSeExistir(cidade.getNome(),
                cidade.getEstado().getId());
            }
            return super.save(cidade);
        } catch (Exception e){
            this.gereExcecaoSeExistir(cidade.getNome(),
                cidade.getEstado().getId());
        }
    }
}

```

```

        throw e;
    }
}

protected void gereExcecaoSeExistir(String nome, Long idEstado) throws
Exception {
    if (isResult(nome, idEstado))
        throw new Exception(REGISTRO_JA_EXISTE);
}

public void delete(Long id) throws Exception {
    //Criar FKs para impedir exclusão
    super.delete(id);
}

public BaseDomain getDomain(Long id, boolean detachEstado) throws Exception
{
    if (detachEstado)
        getPersistenceManager().getFetchPlan().addGroup("detach_estado");
    return super.getDomain(id);
}

public Collection pesquisarPorNome(String nome, String ordenacao, boolean
detachEstado) throws Exception {
    if (detachEstado)
        getPersistenceManager().getFetchPlan().addGroup("detach_estado");
    return super.pesquisarPorNome(nome, ordenacao);
}

public Collection pesquisarPorEstado(Long idEstado) throws Exception {
    getPersistenceManager().getFetchPlan().addGroup("detach_estado");
    return super.pesquisar(getReferenceClass(), "estado.id == idEstado",
"Long idEstado", new Object[] {idEstado}, null);
}

public boolean isResult(String nomeCidade, Long idEstado) throws Exception {
    return super.isResult(getReferenceClass(), "nome == nomeCidade &&
estado.id == idEstado",
        "String nomeCidade, Long idEstado", new Object[] {nomeCidade,
idEstado});
}

public Collection getAll(String ordenacao, boolean detachEstado) throws
Exception {

```

```

        if (detachEstado)
            getPersistenceManager().getFetchPlan().addGroup("detach_estado");
        return super.getAll(ordemacao);
    }
}

```

DAOJdo.java

```

/*
 * DAOJdo.java
 *
 * Created on 23 de Janeiro de 2006, 21:07
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

package br.com.ond.ouvidoria.persistencia.geral.dao.jdo;

import br.com.ond.ouvidoria.logica.domain.BaseDomain;
import br.com.ond.ouvidoria.persistencia.geral.dao.DAO;
import java.util.Collection;
import org.springframework.dao.DataAccessException;
import org.springframework.orm.jdo.support.JdoDaoSupport;

/**
 *
 * @author Desenvolvimento
 */
public abstract class DAOJdo extends JdoDaoSupport implements DAO {

    protected static final String ID_NAO_ENCONTRADO = "Registro não encontrado.";

    protected static final String NENHUM_REGISTRO_ENCONTRADO = "Nenhum registro não encontrado.";

    /** Creates a new instance of DAOJdo */
    public DAOJdo() {
    }
}

```

```

public Object save(BaseDomain domain) throws Exception {
    return getPersistenceManager().makePersistent(domain);
}

public BaseDomain getDomain(Long id) throws Exception {
    return this.getDomain(getReferenceClass(), id);
}

public BaseDomain getDomain(Class classe, Long id) throws Exception {
    try{
        BaseDomain domain = (BaseDomain)
getJdoTemplate().getObjectById(classe, id);
        return (BaseDomain) getPersistenceManager().detachCopy(domain);
    } catch (DataAccessException e) {
        throw new Exception(ID_NAO_ENCONTRADO);
    }
}

public Collection getAll(String ordering) throws Exception {
    return this.getAll(getReferenceClass(), ordering);
}

public Collection getAll(Class classe, String ordering) throws Exception {
    Collection collection;
    if (ordering == null) {
        collection = getJdoTemplate().find(classe);
    } else {
        collection = getJdoTemplate().find(classe, null, ordering);
    }
    if (collection.size() > 0) {
        collection = getPersistenceManager().detachCopyAll(collection);
        return collection;
    } else {
        throw new Exception(NENHUM_REGISTRO_ENCONTRADO);
    }
}

protected Collection pesquisar(Class classe, String filtros, String
parametros, Object[] valores, String ordenacao) throws Exception {
    Collection collection;
    if (valores != null) {
        if (ordenacao == null){

```

```

        collection = getJdoTemplate().find(classe, filtros, parametros,
valores);
    } else {
        collection = getJdoTemplate().find(classe, filtros, parametros,
valores, ordenacao);
    }
    if (collection.size() > 0) {
        collection = getPersistenceManager().detachCopyAll(collection);
    } else {
        throw new Exception(NENHUM_REGISTRO_ENCONTRADO);
    }
} else {
    collection = this.getAll(ordenacao);
}
return collection;
}

protected boolean isResult(Class classe, String filtros, String parametros,
Object[] valores) throws Exception {
    Collection collection = getJdoTemplate().find(classe, filtros,
parametros, valores);
    if (collection.size() > 0) {
        return true;
    } else {
        return false;
    }
}
protected abstract Class getReferenceClass();
}

```

FeriadoDAOJdo.java

```

/*
 * FeriadoDAOJdo.java
 *
 * Created on 15 de Janeiro de 2006, 15:52
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

```

```

package br.com.ond.ouvidoria.persistencia.geral.dao.jdo;

import br.com.ond.ouvidoria.logica.domain.BaseDomain;
import br.com.ond.ouvidoria.logica.domain.Feriado;
import br.com.ond.ouvidoria.persistencia.geral.dao.FeriadoDAO;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Date;
import javax.jdo.Query;

/**
 *
 * @author Rony Reinehr Brand
 */
public class FeriadoDAOJdo extends BaseDAOJdo implements FeriadoDAO {

    protected static final String REGISTRO_JA_EXISTE = "Feriado já existe.";
    /** Creates a new instance of FeriadoDAOJdo */
    public FeriadoDAOJdo() {
    }

    protected Class getReferenceClass() {
        return Feriado.class;
    }

    public Object save(BaseDomain domain) throws Exception {
        Feriado feriado = (Feriado) domain;
        try {
            if (feriado.getId() != null) {
                Feriado feriadoBD = (Feriado) getDomain(feriado.getId());
                if (! (feriado.getData().equals(feriadoBD.getData())))
                    this.gereExcecaoSeExistir(feriado.getData(),
REGISTRO_JA_EXISTE);
            }
            return super.save(domain);
        } catch (Exception e) {
            this.gereExcecaoSeExistir(feriado.getData(), REGISTRO_JA_EXISTE);
            throw e;
        }
    }
}

```

```

    public Collection pesquisarPorData(Date data, String ordenacao) throws
Exception {
    Collection collection;
    if (data != null) {
        collection = super.pesquisar(getReferenceClass(), "data == vData",
"java.util.Date vData", new Object[] {data}, ordenacao);
    } else {
        collection = super.getAll(ordenacao);
    }
    return collection;
}

```

```

    public Collection pesquisarPorPeriodo(Date dataInicial, Date dataFinal,
String ordenacao) throws Exception {
    Collection collection;
    if (dataInicial != null || dataFinal != null) {
        String filtro = "";
        String parametros = "";
        ArrayList valores = new ArrayList();
        if (dataInicial != null) {
            filtro = "data >= dataInicial";
            parametros = "java.util.Date dataInicial";
            valores.add(dataInicial);
        }
        if (dataFinal != null) {
            if (!filtro.equals("")) {
                filtro += " && ";
                parametros += ", ";
            }
            filtro += "data <= dataFinal";
            parametros += "java.util.Date dataFinal";
            valores.add(dataFinal);
        }
        collection = super.pesquisar(getReferenceClass(), filtro,
parametros, valores.toArray(), ordenacao);
    } else {
        collection = super.getAll(ordenacao);
    }
    return collection;
}

```

```

    public Long getQuantidadePorPeriodo(Date dataInicial, Date dataFinal) throws
Exception {

```

```

Query query = getPersistenceManager().newQuery(getReferenceClass());
query.setResult("count(this.data)");
ArrayList valores = new ArrayList();
if (dataInicial != null || dataFinal != null) {
    String filtro = "";
    String parametros = "";
    if (dataInicial != null) {
        filtro = "data >= :dataInicial";
        parametros = "java.util.Date dataInicial";
        valores.add(dataInicial);
    }
    if (dataFinal != null) {
        if (!filtro.equals("")) {
            filtro += " && ";
            parametros += ", ";
        }
        filtro += "data <= :dataFinal";
        parametros += "java.util.Date dataFinal";
        valores.add(dataFinal);
    }
    query.setFilter(filtro);
    query.declareParameters(parametros);
}
Collection collection =
(Collection)query.executeWithArray(valores.toArray());
return (Long)collection.iterator().next();
}

public boolean isResult(Date data) throws Exception {
    return super.isResult(getReferenceClass(), "data == vData",
"java.util.Date vData", new Object[] {data});
}

protected void gereExcecaoSeExistir(Date data, String mensagemExcecao)
throws Exception {
    if (isResult(data))
        throw new Exception(mensagemExcecao);
}
}

```

IdentidadeNomeDAOJdo.java

```
/*
 * IdentidadeNomeDaoJdo.java
 *
 * Created on 8 de Janeiro de 2006, 19:06
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

package br.com.omd.ouvidoria.persistencia.geral.dao.jdo;

import br.com.omd.ouvidoria.logica.domain.BaseDomain;
import br.com.omd.ouvidoria.logica.domain.IdentidadeNome;
import br.com.omd.ouvidoria.persistencia.geral.dao.IdentidadeNomeDAO;
import java.util.Collection;
import org.springframework.dao.DataAccessException;

/**
 *
 * @author Rony Reinehr Brand
 */
public class IdentidadeNomeDAOJdo extends BaseDAOJdo implements
IdentidadeNomeDAO {

    /** Creates a new instance of IdentidadeNomeDaoJdo */
    public IdentidadeNomeDAOJdo() {
    }

    public Collection pesquisarPorNome(String nome, String ordenacao) throws
Exception {
        Collection collection;
        if (nome != null && (!nome.equals(""))) {
            collection = super.pesquisar(getReferenceClass(), "nome == vNome",
"String vNome", new Object[] {nome}, ordenacao);
        } else {
            collection = super.getAll(ordenacao);
        }
        return collection;
    }
}
```

```

        public boolean isResult(String nome) throws Exception {
            return super.isResult(getReferenceClass(), "nome == vNome", "String
vNome", new Object[] {nome});
        }

        protected Class getReferenceClass() {
            return IdentidadeNome.class;
        }

        protected void gereExcecaoSeExistir(String nome, String mensagemExcecao)
throws Exception {
            if (isResult(nome))
                throw new Exception(mensagemExcecao);
        }
    }
}

```

ModoDAOJdo.java

```

/*
 * ModoDAOJdo.java
 *
 * Created on 16 de Janeiro de 2006, 19:21
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

```

```

package br.com.umd.ouvidoria.persistencia.geral.dao.jdo;

```

```

import br.com.umd.ouvidoria.logica.domain.BaseDomain;
import br.com.umd.ouvidoria.logica.domain.Modos;
import br.com.umd.ouvidoria.persistencia.geral.dao.ModosDAO;
import java.util.Collection;

```

```

/**
 *
 * @author Rony Reinehr Brand
 */

```

```

public class ModosDAOJdo extends IdentidadeNomeDAOJdo implements ModosDAO {

```

```

    protected static final String REGISTRO_JA_EXISTE = "Modos já existe.";

```

```

/** Creates a new instance of ModoDAOJdo */
public ModoDAOJdo() {
}

public Object save(BaseDomain domain) throws Exception {
    Modo modo = (Modo) domain;
    try {
        if (modo.getId() != null) {
            Modo modoBD = (Modo) getDomain(modo.getId());
            if (! (modo.getNome().equals(modoBD.getNome())))
                super.gereExcecaoSeExistir(modo.getNome(),
REGISTRO_JA_EXISTE);
        }
        return super.save(domain);
    } catch (Exception e) {
        super.gereExcecaoSeExistir(modo.getNome(), REGISTRO_JA_EXISTE);
        throw e;
    }
}

public Collection pesquisarPorRetornavel(boolean retornavel, String
ordenacao) throws Exception {
    return super.pesquisar(getReferenceClass(), "retornavel ==
isRetornavel", "Boolean isRetornavel", new Object[] {new Boolean(retornavel)},
ordenacao);
}

protected Class getReferenceClass() {
    return Modo.class;
}
}

```

BaseOutrosDAO.java

```

/*
 * BaseOutrosDAO.java
 *
 * Created on 22 de Janeiro de 2006, 10:42
 *
 * To change this template, choose Tools | Template Manager

```

```

* and open the template in the editor.
*/

package br.com.omd.ouvidoria.persistencia.geral.outros.dao;

import br.com.omd.ouvidoria.logica.domain.BaseDomain;
import br.com.omd.ouvidoria.logica.domain.outros.BaseOutrosDomain;
import br.com.omd.ouvidoria.persistencia.geral.dao.DAO;
import java.util.Collection;

/**
 *
 * @author Desenvolvimento
 */
public interface BaseOutrosDAO extends DAO {

    public Collection getQuantidadePorOutro() throws Exception;

    public Collection pesquisarPorNome(String nome) throws Exception;

    public void substituirOutrosPorDomain(String nomeOutro, BaseDomain domain)
throws Exception;

}

```

OutraCidadeDAO.java

```

/*
 * OutraCidadeDAO.java
 *
 * Created on 18 de Janeiro de 2006, 20:23
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

package br.com.omd.ouvidoria.persistencia.geral.outros.dao;

import br.com.omd.ouvidoria.logica.domain.Cidade;
import br.com.omd.ouvidoria.logica.domain.outros.OutraCidade;

```

```

/**
 *
 * @author Rony Reinehr Brand
 */
public interface OutraCidadeDAO extends BaseOutrosDAO {

}

```

OutraUnidadeDAO.java

```

/*
 * OutraUnidadeDAO.java
 *
 * Created on 18 de Janeiro de 2006, 20:23
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

```

```

package br.com.umd.ouvidoria.persistencia.geral.outros.dao;

```

```

/**
 *
 * @author Rony Reinehr Brand
 */
public interface OutraUnidadeDAO extends BaseOutrosDAO {

    public Object pesquisarOutraUnidadePorManifestacao(Long idManifestacao)
    throws Exception;

}

```

OutroAssuntoDAO.java

```

/*
 * OutroAssuntoDAO.java
 *
 * Created on 18 de Janeiro de 2006, 20:23
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.

```

```

*/

package br.com.ond.ouvidoria.persistencia.geral.outros.dao;

/**
 *
 * @author Rony Reinehr Brand
 */
public interface OutroAssuntoDAO extends BaseOutrosDAO {

    public Object pesquisarOutroAssuntoPorManifestacao(Long idManifestacao)
    throws Exception;

}

```

BaseOutrosDAOJdo.java

```

/*
 * BaseOutrosDAOJdo.java
 *
 * Created on 22 de Janeiro de 2006, 10:41
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

package br.com.ond.ouvidoria.persistencia.geral.outros.dao.jdo;

import br.com.ond.ouvidoria.logica.domain.BaseDomain;
import br.com.ond.ouvidoria.logica.domain.outros.BaseOutrosDomain;
import br.com.ond.ouvidoria.logica.dto.OutrosDTO;
import br.com.ond.ouvidoria.persistencia.geral.dao.jdo.DAOJdo;
import br.com.ond.ouvidoria.persistencia.geral.outros.dao.BaseOutrosDAO;
import java.util.ArrayList;
import java.util.Collection;
import javax.jdo.Query;

/**
 *
 * @author Desenvolvimento
 */

```

```

public abstract class BaseOutrosDAOJdo extends DAOJdo implements BaseOutrosDAO {

    /** Creates a new instance of BaseOutrosDAOJdo */
    public BaseOutrosDAOJdo() {
    }

    // public BaseDomain getDomain(Long id) throws Exception {
    //     return this.getDomain(getReferenceClass(), id);
    // }
    //
    // public BaseDomain getDomain(Class classe, Long id) throws Exception {
    //     try{
    //         BaseDomain domain = (BaseDomain)
getJdoTemplate().getObjectById(classe, id);
    //         return (BaseDomain) getPersistenceManager().detachCopy(domain);
    //     } catch (DataAccessException e) {
    //         throw new Exception(ID_NAO_ENCONTRADO);
    //     }
    // }

    public Collection pesquisarPorNome(String nomeOutro) throws Exception {
        Collection collection;
        if (nomeOutro != null && (!nomeOutro.equals(""))) {
            collection = super.pesquisar(getReferenceClass(), "nome == vNome",
"String vNome", new Object[]{nomeOutro}, null);
        } else {
            collection = super.getAll(null);
        }
        return collection;
    }

    public Collection getQuantidadePorOutro() throws Exception {
        Query query = getPersistenceManager().newQuery(getReferenceClass());
        query.setResultClass(OutrosDTO.class);
        query.setResult("count(nome) as quantidade, nome");
        query.setGrouping("nome");
        //query.setOrdering("count(nome) DESC");
        //System.out.println(query.toString());
        return new ArrayList((Collection)query.execute());
    }

    protected void delete(String nomeOutro) throws Exception {

```

```

        Collection collection = getJdoTemplate().find(getReferenceClass(), "nome
== vNome", "String vNome", new Object[]{nomeOutro});
        getPersistenceManager().deletePersistentAll(collection);
    }

    public abstract void substituirOutrosPorDomain(String nomeOutro, BaseDomain
domain) throws Exception;

}

```

OutraCidadeDAOJdo.java

```

/*
 * OutraCidadeDAOJdo.java
 *
 * Created on 18 de Janeiro de 2006, 20:25
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

package br.com.omd.ouvidoria.persistencia.geral.outros.dao.jdo;

import br.com.omd.ouvidoria.logica.domain.Cidade;
import br.com.omd.ouvidoria.logica.domain.outros.OutraCidade;
import br.com.omd.ouvidoria.persistencia.geral.outros.dao.OutraCidadeDAO;

/**
 *
 * @author Rony Reinehr Brand
 */
public class OutraCidadeDAOJdo extends BaseOutrosDAOJdo implements
OutraCidadeDAO {

    /**
     * Creates a new instance of OutraCidadeDAOJdo
     */
    public OutraCidadeDAOJdo(){

    }

    public void substituirOutroPorDomain(OutraCidade outraCidade, Cidade cidade)
throws Exception {

```

```

    }

    protected Class getReferenceClass() {
        return OutraCidade.class;
    }
}

OutraUnidadeDAOJdo.java
/*
 * OutraCidadeDAOJdo.java
 *
 * Created on 18 de Janeiro de 2006, 20:25
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

package br.com.omd.ouvidoria.persistencia.geral.outros.dao.jdo;

import br.com.omd.ouvidoria.logica.domain.BaseDomain;
import br.com.omd.ouvidoria.logica.domain.Manifestacao;
import br.com.omd.ouvidoria.logica.domain.Unidade;
import br.com.omd.ouvidoria.logica.domain.outros.BaseOutrosDomain;
import br.com.omd.ouvidoria.logica.domain.outros.OutraUnidade;
import br.com.omd.ouvidoria.persistencia.geral.outros.dao.OutraUnidadeDAO;
import java.util.Collection;
import java.util.Iterator;

/**
 *
 * @author Rony Reinehr Brand
 */
public class OutraUnidadeDAOJdo extends BaseOutrosDAOJdo implements
OutraUnidadeDAO {

    /**
     * Creates a new instance of OutraCidadeDAOJdo
     */
    public OutraUnidadeDAOJdo(){

```

```

    }

    protected Class getReferenceClass() {
        return OutraUnidade.class;
    }

    public void substituirOutrosPorDomain(String nomeOutro, BaseDomain domain)
    throws Exception {
        Unidade unidade = (Unidade)getDomain(Unidade.class, domain.getId());
        Collection collection = super.pesquisar(Manifestacao.class,
        "outraUnidade.nome == nomeOutro", "String nomeOutro",
            new Object[]{nomeOutro}, null);
        Iterator iterator = collection.iterator();
        while (iterator.hasNext()){
            Manifestacao manifestacao = (Manifestacao)iterator.next();
            manifestacao.setUnidadeOcorrencia(unidade);
            super.save(manifestacao);
        }
        super.delete(nomeOutro);
    }

    public Object pesquisarOutraUnidadePorManifestacao(Long idManifestacao)
    throws Exception {
        return super.pesquisar(getReferenceClass(), "manifestacao.id ==
        idManifestacao", "Long idManifestacao",
            new Object[]{idManifestacao}, null).toArray()[0];
    }
}

```

OutroAssuntoDAOJdo.java

```

/*
 * OutraCidadeDAOJdo.java
 *
 * Created on 18 de Janeiro de 2006, 20:25
 *
 * To change this template, choose Tools | Template Manager

```

```

* and open the template in the editor.
*/

package br.com.ond.ouvidoria.persistencia.geral.outros.dao.jdo;

import br.com.ond.ouvidoria.logica.domain.Assunto;
import br.com.ond.ouvidoria.logica.domain.BaseDomain;
import br.com.ond.ouvidoria.logica.domain.Manifestacao;
import br.com.ond.ouvidoria.logica.domain.outros.OutroAssunto;
import br.com.ond.ouvidoria.persistencia.geral.outros.dao.OutroAssuntoDAO;
import java.util.Collection;
import java.util.Iterator;

/**
 *
 * @author Rony Reinehr Brand
 */
public class OutroAssuntoDAOJdo extends BaseOutrosDAOJdo implements
OutroAssuntoDAO {

    /**
     * Creates a new instance of OutroAssuntoDAO
     */
    public OutroAssuntoDAOJdo(){

    }

    protected Class getReferenceClass() {
        return OutroAssunto.class;
    }

    public void substituirOutrosPorDomain(String nomeOutro, BaseDomain domain)
throws Exception {
        Assunto assunto = (Assunto)getDomain(Assunto.class, domain.getId());
        Collection collection = super.pesquisar(Manifestacao.class,
"outroAssunto.nome == nomeOutro", "String nomeOutro",
                new Object[]{nomeOutro}, null);
        Iterator iterator = collection.iterator();
        while (iterator.hasNext()){
            Manifestacao manifestacao = (Manifestacao)iterator.next();
            manifestacao.setAssunto(assunto);
        }
    }
}

```

```

        super.save(manifestacao);
    }
    super.delete(nomeOutro);
}

    public Object pesquisarOutroAssuntoPorManifestacao(Long idManifestacao)
throws Exception {
        return super.pesquisar(getReferenceClass(), "manifestacao.id ==
idManifestacao", "Long idManifestacao",
            new Object[]{idManifestacao}, null).toArray()[0];
    }
}

```

BaseDAOTestCase.java

```

package br.com.ond.ouvidoria.persistencia.geral.dao;
import java.util.Collection;
import java.util.Iterator;
import junit.framework.TestCase;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
/*
 * BaseDAOTestCase.java
 *
 * Created on 15 de Janeiro de 2006, 10:28
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */
/**
 *
 * @author Rony Reinehr Brand
 */
public abstract class BaseDAOTestCase extends TestCase {

    /** Creates a new instance of BaseDAOTestCase */
    public BaseDAOTestCase() {
    }
    protected final static ApplicationContext ctx;

    private static final String configLocation = "applicationContext-test.xml";

```

```

protected Iterator iterator;

protected static final String ordenacao = "id ASC";

static {
    ctx = new ClassPathXmlApplicationContext(configLocation);
}

public void exibaColecao(Collection colecao) throws Exception {
    iterator = colecao.iterator();
    while(iterator.hasNext()){
        this.exibaDomain(iterator.next());
    }
}

public abstract void exibaDomain(Object parametro);
}

```

AssuntoDAOJdoTest.java

```

package br.com.omd.ouvidoria.persistencia.geral.dao.jdo;
import br.com.omd.ouvidoria.logica.domain.Assunto;
import br.com.omd.ouvidoria.logica.domain.BaseDomain;
import br.com.omd.ouvidoria.persistencia.geral.dao.AssuntoDAO;
import br.com.omd.ouvidoria.persistencia.geral.dao.BaseDAOTestCase;
import java.util.Collection;
import java.util.Iterator;

/*
 * AssuntoDAOJdoTest.java
 * JUnit based test
 *
 * Created on 15 de Janeiro de 2006, 10:13
 */

/**
 *
 * @author Rony Reinehr Brand
 */
public class AssuntoDAOJdoTest extends BaseDAOTestCase {

```

```

private AssuntoDAO dao;

private Assunto domain;

private Long id = null;

protected void setUp() throws Exception {
    dao = (AssuntoDAO) ctx.getBean("assunto");
}

/**
 * Test of save method, of class
br.com.ond.ouvidoria.persistencia.geral.dao.jdo.AssuntoDAOJdo.
 */
public void testSave() throws Exception {
    System.out.println("SAVE");

    this.inserir("incluido");
    this.inserir("incluido");
    this.inserir("incluido2");

    this.exibaColecao(dao.getAll(ordenacao));
    System.out.println("");
}

/**
 * Test of update method, of class AssuntoDAOJdo.
 */
public void testUpdate() throws Exception {
    System.out.println("UPDATE");

    this.atualizar("incluido", "atualizado");
    this.atualizar("incluido2", "atualizado");

    this.exibaColecao(dao.getAll(ordenacao));
    System.out.println("");
}

/**
 * Test of delete method, of class AssuntoDAOJdo.
 */

```

```

public void testDelete() throws Exception {
    System.out.println("DELETE");

    dao.delete((BaseDomain)
null).toArray()[0]);
    dao.pesquisarPorNome("atualizado",
null).toArray()[0]);
    dao.delete((BaseDomain)
null).toArray()[0]);
    dao.pesquisarPorNome("incluido2",
null).toArray()[0]);

    this.exibaColecao(dao.getAll(ordenação));
    System.out.println("");
}

public void inserir(String nome) throws Exception {
    domain = new Assunto();
    domain.setNome(nome);
    this.salvar();
}

public void atualizar(String antigo, String novo) throws Exception {
    domain = (Assunto)dao.pesquisarPorNome(antigo, null).toArray()[0];
    domain.setNome(novo);
    this.salvar();
}

public void salvar() throws Exception {
    try {
        dao.save(domain);
    } catch (Exception e){
        assertEquals(AssuntoDAOJdo.REGISTRO_JA_EXISTE, e.getMessage());
    }
}

public void exibaDomain(Object parametro) {
    domain = (Assunto) parametro;
    System.out.println(domain.getId().longValue());
    System.out.println(domain.getNome());
}
}

```

```

/*
 * CargoDAOJdoTest.java
 * JUnit based test
 *
 * Created on 20 de Janeiro de 2006, 20:01
 */

package br.com.ond.ouvidoria.persistencia.geral.dao.jdo;

import br.com.ond.ouvidoria.logica.domain.BaseDomain;
import br.com.ond.ouvidoria.logica.domain.Cargo;
import br.com.ond.ouvidoria.persistencia.geral.dao.BaseDAOTestCase;
import br.com.ond.ouvidoria.persistencia.geral.dao.CargoDAO;
import java.util.Collection;
import java.util.Iterator;

/**
 *
 * @author Desenvolvimento
 */
public class CargoDAOJdoTest extends BaseDAOTestCase {

    private CargoDAO dao;

    private Cargo domain;

    private Long id = null;

    protected void setUp() throws Exception {
        dao = (CargoDAO) ctx.getBean("cargo");
    }

    /**
     * Test of save method, of class
     br.com.ond.ouvidoria.persistencia.geral.dao.jdo.CargoDAOJdo.
     */
    public void testSave() throws Exception {
        System.out.println("SAVE");

        this.inserir("incluido");
        this.inserir("incluido");
    }
}

```

```

        this.inserir("incluido2");

        this.exibaColecao(dao.getAll(ordenacao));
        System.out.println("");
    }

    /**
     * Test of update method, of class CargoDAOJdo.
     */
    public void testUpdate() throws Exception {
        System.out.println("UPDATE");

        this.atualizar("incluido", "atualizado");
        this.atualizar("incluido2", "atualizado");

        this.exibaColecao(dao.getAll(ordenacao));
        System.out.println("");
    }

    /**
     * Test of delete method, of class CargoDAOJdo.
     */
    public void testDelete() throws Exception {
        System.out.println("DELETE");

        dao.delete((BaseDomain) null).toArray()[0];
        dao.delete((BaseDomain) null).toArray()[0];
        dao.pesquisarPorNome("atualizado",
        dao.pesquisarPorNome("incluido2",

        this.exibaColecao(dao.getAll(ordenacao));
        System.out.println("");
    }

    public void salvar() throws Exception {
        try {
            dao.save(domain);
        } catch (Exception e){
            assertEquals(CargoDAOJdo.REGISTRO_JA_EXISTE, e.getMessage());
        }
    }
}

```

```

public void exibaDomain(Object parametro) {
    domain = (Cargo) parametro;
    System.out.println(domain.getId().longValue());
    System.out.println(domain.getNome());
}

public void inserir(String nome) throws Exception {
    domain = new Cargo();
    domain.setNome(nome);
    this.salvar();
}

public void atualizar(String antigo, String novo) throws Exception {
    domain = (Cargo)dao.pesquisarPorNome(antigo, null).toArray()[0];
    domain.setNome(novo);
    this.salvar();
}
}

```

CidadeDAOJdoTest.java

```

/*
 * CidadeDAOJdo.java
 * JUnit based test
 *
 * Created on 20 de Janeiro de 2006, 20:06
 */

package br.com.ond.ouvidoria.persistencia.geral.dao.jdo;

import br.com.ond.ouvidoria.logica.domain.BaseDomain;
import br.com.ond.ouvidoria.logica.domain.Cidade;
import br.com.ond.ouvidoria.logica.domain.Estado;
import br.com.ond.ouvidoria.persistencia.geral.dao.BaseDAOTestCase;
import br.com.ond.ouvidoria.persistencia.geral.dao.CidadeDAO;
import java.util.Collection;
import java.util.Iterator;

/**
 *
 */

```

```

* @author Desenvolvimento
*/
public class CidadeDAOJdoTest extends BaseDAOTestCase {

    private CidadeDAO dao;

    private Cidade domain;

    private Estado estado;

    private Long id = null;

    private Long idEstado = null;

    protected void setUp() throws Exception {
        dao = (CidadeDAO) ctx.getBean("cidade");
    }

    /**
     * Test of save method, of class
     br.com.omd.ouvidoria.persistencia.geral.dao.jdo.CidadeDAOJdo.
     */
    public void testSave() throws Exception {
        System.out.println("SAVE");

        this.inserir("incluido", 1);
        this.inserir("incluido", 1);
        this.inserir("incluido2", 2);

        this.exibaColecao(dao.getAll(ordemacao, true));
        System.out.println("");
        this.exibaColecao(dao.pesquisarPorEstado(new Long(1)));
        System.out.println("");
        System.out.println("");
    }

    /**
     * Test of update method, of class CidadeDAOJdo.
     */
    public void testUpdate() throws Exception {
        System.out.println("UPDATE");
    }
}

```

```

        this.atualizar("incluido", "atualizado", 2);
        this.atualizar("incluido2", "atualizado", 2);

        this.exibaColecao(dao.getAll(ordenacao, true));
        System.out.println("");
        this.exibaColecao(dao.pesquisarPorEstado(new Long(2)));
        System.out.println("");
        System.out.println("");
    }

    /**
     * Test of delete method, of class CidadeDAOJdo.
     */
    public void testDelete() throws Exception {
        System.out.println("DELETE");

        dao.delete((BaseDomain) null).toArray()[0];
        dao.pesquisarPorNome("atualizado",
        dao.delete((BaseDomain) null).toArray()[0];
        dao.pesquisarPorNome("incluido2",

        this.exibaColecao(dao.getAll(ordenacao, true));
        System.out.println("");
    }

    public void salvar() throws Exception {
        try {
            dao.save(domain);
        } catch (Exception e){
            //e.printStackTrace();
            assertEquals(CidadeDAOJdo.REGISTRO_JA_EXISTE, e.getMessage());
        }
    }

    public void exibaDomain(Object parametro) {
        domain = (Cidade) parametro;
        System.out.println(domain.getId().longValue());
        System.out.println(domain.getNome());
        estado = domain.getEstado();
        if (estado != null) {
            System.out.println(estado.getId());

```

```

        System.out.println(estado.getSigla());
        System.out.println(estado.getNome());
    }
}

public void inserir(String nome, long idEstado) throws Exception {
    domain = new Cidade();
    domain.setNome(nome);
    Long id_estado = new Long(idEstado);
    estado = (Estado) dao.getDomain(Estado.class, id_estado);
    domain.setEstado(estado);
    this.salvar();
}

public void atualizar(String antigo, String novo, long idEstado) throws
Exception {
    domain = (Cidade)dao.pesquisarPorNome(antigo, null).toArray()[0];
    domain.setNome(novo);
    domain.setEstado((Estado) dao.getDomain(Estado.class, new
Long(idEstado)));
    this.salvar();
}
}

```

FeriadoDAOJdoTest.java

```

/*
 * CidadeDAOJdo.java
 * JUnit based test
 *
 * Created on 20 de Janeiro de 2006, 20:06
 */

package br.com.ond.ouvidoria.persistencia.geral.dao.jdo;

import br.com.ond.ouvidoria.logica.domain.BaseDomain;
import br.com.ond.ouvidoria.logica.domain.Cidade;
import br.com.ond.ouvidoria.logica.domain.Estado;
import br.com.ond.ouvidoria.persistencia.geral.dao.BaseDAOTestCase;
import br.com.ond.ouvidoria.persistencia.geral.dao.CidadeDAO;

```

```

import java.util.Collection;
import java.util.Iterator;

/**
 *
 * @author Desenvolvimento
 */
public class CidadeDAOJdoTest extends BaseDAOTestCase {

    private CidadeDAO dao;

    private Cidade domain;

    private Estado estado;

    private Long id = null;

    private Long idEstado = null;

    protected void setUp() throws Exception {
        dao = (CidadeDAO) ctx.getBean("cidade");
    }

    /**
     * Test of save method, of class
     br.com.ond.ouvidoria.persistencia.geral.dao.jdo.CidadeDAOJdo.
     */
    public void testSave() throws Exception {
        System.out.println("SAVE");

        this.inserir("incluido", 1);
        this.inserir("incluido", 1);
        this.inserir("incluido2", 2);

        this.exibaColecao(dao.getAll(ordenacao, true));
        System.out.println("");
        this.exibaColecao(dao.pesquisarPorEstado(new Long(1)));
        System.out.println("");
        System.out.println("");
    }
}

```

```

/**
 * Test of update method, of class CidadeDAOJdo.
 */
public void testUpdate() throws Exception {
    System.out.println("UPDATE");

    this.atualizar("incluido", "atualizado", 2);
    this.atualizar("incluido2", "atualizado", 2);

    this.exibaColecao(dao.getAll(ordenacao, true));
    System.out.println("");
    this.exibaColecao(dao.pesquisarPorEstado(new Long(2)));
    System.out.println("");
    System.out.println("");
}

/**
 * Test of delete method, of class CidadeDAOJdo.
 */
public void testDelete() throws Exception {
    System.out.println("DELETE");

    dao.delete((BaseDomain) null).toArray()[0];
    dao.pesquisarPorNome("atualizado",
    dao.delete((BaseDomain) null).toArray()[0];
    dao.pesquisarPorNome("incluido2",

    this.exibaColecao(dao.getAll(ordenacao, true));
    System.out.println("");
}

public void salvar() throws Exception {
    try {
        dao.save(domain);
    } catch (Exception e){
        //e.printStackTrace();
        assertEquals(CidadeDAOJdo.REGISTRO_JA_EXISTE, e.getMessage());
    }
}

public void exibaDomain(Object parametro) {
    domain = (Cidade) parametro;
}

```

```

System.out.println(domain.getId().longValue());
System.out.println(domain.getNome());
estado = domain.getEstado();
if (estado != null) {
    System.out.println(estado.getId());
    System.out.println(estado.getSigla());
    System.out.println(estado.getNome());
}
}

```

```

public void inserir(String nome, long idEstado) throws Exception {
    domain = new Cidade();
    domain.setNome(nome);
    Long id_estado = new Long(idEstado);
    estado = (Estado) dao.getDomain(Estado.class, id_estado);
    domain.setEstado(estado);
    this.salvar();
}

```

```

public void atualizar(String antigo, String novo, long idEstado) throws
Exception {
    domain = (Cidade)dao.pesquisarPorNome(antigo, null).toArray()[0];
    domain.setNome(novo);
    domain.setEstado((Estado) dao.getDomain(Estado.class, new
Long(idEstado)));
    this.salvar();
}
}

```

ModoDAOJdoTest.java

```

/*
 * ModoDAOJdoTest.java
 * JUnit based test
 *
 * Created on 21 de Janeiro de 2006, 18:34
 */

```

```

package br.com.ond.ouvidoria.persistencia.geral.dao.jdo;

```

```

import br.com.ond.ouvidoria.logica.domain.BaseDomain;
import br.com.ond.ouvidoria.logica.domain.Mod0;
import br.com.ond.ouvidoria.persistencia.geral.dao.BaseDAOTestCase;
import br.com.ond.ouvidoria.persistencia.geral.dao.Mod0DAO;
import java.util.Collection;
import java.util.Iterator;

/**
 *
 * @author Desenvolvimento
 */
public class Mod0DAOJdoTest extends BaseDAOTestCase {

    private Mod0DAO dao;

    private Mod0 domain;

    private Long id = null;

    private Collection collection;

    private Iterator iterator;

    protected void setUp() throws Exception {
        dao = (Mod0DAO) ctx.getBean("modo");
    }

    /**
     * Test of save method, of class
     br.com.ond.ouvidoria.persistencia.geral.dao.jdo.Mod0DAOJdo.
     */
    public void testSave() throws Exception {
        System.out.println("SAVE");

        this.inserir("incluido", true);
        this.inserir("incluido", false);
        this.inserir("incluido2", true);

        this.exibaColecao(dao.getAll(ordemacao));
        System.out.println("");
    }
}

```

```

/**
 * Test of update method, of class ModoDAOJdo.
 */
public void testUpdate() throws Exception {
    System.out.println("UPDATE");

    this.atualizar("incluido", "atualizado", false);
    this.atualizar("incluido2", "atualizado", true);

    this.exibaColecao(dao.getAll(ordemacao));
    System.out.println("");
}

/**
 * Test of delete method, of class ModoDAOJdo.
 */
public void testDelete() throws Exception {
    System.out.println("DELETE");

    dao.delete((BaseDomain) null).toArray()[0];
    dao.pesquisarPorNome("atualizado",
    dao.delete((BaseDomain) null).toArray()[0];
    dao.pesquisarPorNome("incluido2",

    this.exibaColecao(dao.getAll(ordemacao));
    System.out.println("");
}

public void salvar() throws Exception {
    try {
        dao.save(domain);
    } catch (Exception e){
        assertEquals(ModoDAOJdo.REGISTRO_JA_EXISTE, e.getMessage());
    }
}

public void exibaDomain(Object parametro) {
    domain = (Modo) parametro;
    System.out.println(domain.getId().longValue());
    System.out.println(domain.getNome());
    System.out.println(domain.isRetornavel());
}

```

```

    }

    public void inserir(String nome, boolean retornavel) throws Exception {
        domain = new Modo();
        domain.setNome(nome);
        domain.setRetornavel(retornavel);
        this.salvar();
    }

    public void atualizar(String antigo, String novo, boolean retornavel) throws
Exception {
        domain = (Modo)dao.pesquisarPorNome(antigo, null).toArray()[0];
        domain.setNome(novo);
        domain.setRetornavel(retornavel);
        this.salvar();
    }
}

```

OutraUnidadeDAOJdo.java

```

package br.com.omd.ouvidoria.persistencia.geral.outros.dao.jdo;
import br.com.omd.ouvidoria.logica.domain.Manifestacao;
import br.com.omd.ouvidoria.logica.domain.Unidade;
import br.com.omd.ouvidoria.logica.domain.outros.OutraUnidade;
import br.com.omd.ouvidoria.logica.dto.OutrosDTO;
import br.com.omd.ouvidoria.persistencia.geral.dao.BaseDAOTestCase;
import br.com.omd.ouvidoria.persistencia.geral.outros.dao.OutraUnidadeDAO;

/*
 * OutraUnidadeDAOJdoTest.java
 * JUnit based test
 *
 * Created on 15 de Janeiro de 2006, 10:13
 */

/**
 *
 * @author Rony Reinehr Brand
 */
public class OutraUnidadeDAOJdoTest extends BaseDAOTestCase {

```

```

private OutraUnidadeDAO dao;

private OutraUnidade outroDomain;

private Long id = null;

private Manifestacao manifestacao = null;

private OutrosDTO outroDTO;

protected void setUp() throws Exception {
    dao = (OutraUnidadeDAO) ctx.getBean("outraUnidade");
}

/**
 * Test of save method, of class
br.com.ond.ouvidoria.persistencia.geral.dao.jdo.OutraUnidadeDAOJdo.
 */
public void testSave() throws Exception {
    System.out.println("SAVE");

    this.inserir("outroIncluido", 1);
    this.inserir("outroIncluido", 2);

    this.exibaColecao(dao.getQuantidadePorOutro());
    System.out.println("");
}

/**
 * Test of update method, of class OutraUnidadeDAOJdo.
 */
public void testUpdate() throws Exception {
    System.out.println("UPDATE");

    this.atualizar(1, "outroAtualizado");
    this.atualizar(2, "outroAtualizado");

    this.exibaColecao(dao.getQuantidadePorOutro());
    System.out.println("");
}

```

```

/**
 * Test of substituir method, of class OutraUnidadeDAOJdo.
 */
public void testSubstituirOutrosPorDomain() throws Exception {
    System.out.println("SUBSTITUIR");

    this.substituir("outroAtualizado", 2);

    this.exibaColecao(dao.getQuantidadePorOutro());
    System.out.println("");
}

public void inserir(String nome, int idManifestacao) throws Exception {
    manifestacao = (Manifestacao)dao.getDomain(Manifestacao.class, new
Long(idManifestacao));
    outroDomain = new OutraUnidade();
    outroDomain.setNome(nome);
    outroDomain.setManifestacao(manifestacao);
    dao.save(outroDomain);
}

public void atualizar(int idManifestacao, String novo) throws Exception {
    outroDomain = (OutraUnidade)dao.pesquisarOutraUnidadePorManifestacao(new
Long(idManifestacao));
    outroDomain.setNome(novo);
    dao.save(outroDomain);
}

public void substituir(String nomeOutro, int idDomain) throws Exception {
    Unidade unidade = new Unidade();
    unidade.setId(new Long(idDomain));
    dao.substituirOutrosPorDomain(nomeOutro, unidade);
}

public void exibaDomain(Object parametro) {
    outroDTO = (OutrosDTO) parametro;
    System.out.println(outroDTO.getQuantidade());
    System.out.println(outroDTO.getNome());
}

```

```
}
```

```
OutroAssuntoDAOJdoTest.java
```

```
package br.com.ond.ouvidoria.persistencia.geral.outros.dao.jdo;  
import br.com.ond.ouvidoria.logica.domain.Assunto;  
import br.com.ond.ouvidoria.logica.domain.Manifestacao;  
import br.com.ond.ouvidoria.logica.domain.outros.OutroAssunto;  
import br.com.ond.ouvidoria.logica.dto.OutrosDTO;  
import br.com.ond.ouvidoria.persistencia.geral.dao.BaseDAOTestCase;  
import br.com.ond.ouvidoria.persistencia.geral.outros.dao.OutroAssuntoDAO;
```

```
/*
```

```
 * OutroAssuntoDAOJdoTest.java  
 * JUnit based test  
 *  
 * Created on 15 de Janeiro de 2006, 10:13  
 */
```

```
/**
```

```
 *  
 * @author Rony Reinehr Brand  
 */
```

```
public class OutroAssuntoDAOJdoTest extends BaseDAOTestCase {
```

```
    private OutroAssuntoDAO dao;
```

```
    private OutroAssunto outroDomain;
```

```
    private Long id = null;
```

```
    private Manifestacao manifestacao = null;
```

```
    private OutrosDTO outroDTO;
```

```
    protected void setUp() throws Exception {
```

```
        dao = (OutroAssuntoDAO) ctx.getBean("outroAssunto");
```

```
    }
```

```
/**
```

```
 * Test of save method, of class
```

```

br.com.ond.ouvidoria.persistencia.geral.dao.jdo.OutroAssuntoDAOJdo.
    */
    public void testSave() throws Exception {
        System.out.println("SAVE");

        this.inserir("outroIncluido", 1);
        this.inserir("outroIncluido", 2);

        this.exibaColecao(dao.getQuantidadePorOutro());
        System.out.println("");
    }

    /**
     * Test of update method, of class OutroAssuntoDAOJdo.
     */
    public void testUpdate() throws Exception {
        System.out.println("UPDATE");

        this.atualizar(1, "outroAtualizado");
        this.atualizar(2, "outroAtualizado");

        this.exibaColecao(dao.getQuantidadePorOutro());
        System.out.println("");
    }

    /**
     * Test of substituir method, of class OutroAssuntoDAOJdo.
     */
    public void testSubstituirOutrosPorDomain() throws Exception {
        System.out.println("SUBSTITUIR");

        this.substituir("outroAtualizado", 2);

        this.exibaColecao(dao.getQuantidadePorOutro());
        System.out.println("");
    }

    public void inserir(String nome, int idManifestacao) throws Exception {
        manifestacao = (Manifestacao)dao.getDomain(Manifestacao.class, new
Long(idManifestacao));
        outroDomain = new OutroAssunto();

```

```

        outroDomain.setNome(nome);
        outroDomain.setManifestacao(manifestacao);
        dao.save(outroDomain);
    }

    public void atualizar(int idManifestacao, String novo) throws Exception {
        outroDomain = (OutroAssunto)dao.pesquisarOutroAssuntoPorManifestacao(new
Long(idManifestacao));
        outroDomain.setNome(novo);
        dao.save(outroDomain);
    }

    public void substituir(String nomeOutro, int idDomain) throws Exception {
        Assunto assunto = new Assunto();
        assunto.setId(new Long(idDomain));
        dao.substituirOutrosPorDomain(nomeOutro, assunto);
    }

    public void exibaDomain(Object parametro) {
        outroDTO = (OutrosDTO) parametro;
        System.out.println(outroDTO.getQuantidade());
        System.out.println(outroDTO.getNome());
    }
}

```

Script de criação das tabelas em MySQL, correspondentes ao domínio do problema:

```
CREATE TABLE `assunto` (  
  `ID` int(11) unsigned NOT NULL auto_increment,  
  `Nome` varchar(100) NOT NULL default '',  
  PRIMARY KEY (`ID`),  
  UNIQUE KEY `Nome` (`Nome`)  
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
```

```
CREATE TABLE `cargo` (  
  `ID` int(11) unsigned NOT NULL auto_increment,  
  `Nome` varchar(100) NOT NULL default '',  
  PRIMARY KEY (`ID`),  
  UNIQUE KEY `Nome` (`Nome`)  
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
```

```
CREATE TABLE `cidade` (  
  `ID` int(11) unsigned NOT NULL auto_increment,  
  `Nome` varchar(100) NOT NULL default '',  
  `Estado` int(11) default '0',  
  PRIMARY KEY (`ID`),  
  UNIQUE KEY `Cidade_Estado` (`Nome`,`Estado`),  
  KEY `Nome` (`Nome`)  
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
```

```
CREATE TABLE `classificacao` (  
  `ID` int(11) NOT NULL auto_increment,  
  `Nome` varchar(100) NOT NULL default '',  
  PRIMARY KEY (`ID`),  
  UNIQUE KEY `Nome` (`Nome`)  
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
```

```
CREATE TABLE `comentarios` (  
  `ID` bigint(20) unsigned NOT NULL auto_increment,  
  `Data` date NOT NULL default '0000-00-00',  
  `Comentario` longtext NOT NULL,  
  `Usuario` int(11) NOT NULL default '0',  
  `dataPrevistaResposta` date NOT NULL default '0000-00-00',  
  `Origem` int(11) NOT NULL default '0',  
  `Destino` int(11) NOT NULL default '0',  
  `manifestacao` int(10) unsigned NOT NULL default '0',
```

```
PRIMARY KEY (`ID`)  
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
```

```
CREATE TABLE `encaminhamentos` (  
  `ID` bigint(20) unsigned NOT NULL auto_increment,  
  `Data` date NOT NULL default '0000-00-00',  
  `Usuario` int(11) NOT NULL default '0',  
  `dataPrevistaResposta` date NOT NULL default '0000-00-00',  
  `Origem` int(11) NOT NULL default '0',  
  `Destino` int(11) NOT NULL default '0',  
  `manifestacao` int(11) unsigned NOT NULL default '0',  
  PRIMARY KEY (`ID`)  
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
```

```
CREATE TABLE `envios` (  
  `ID` bigint(20) unsigned NOT NULL auto_increment,  
  `Data` date NOT NULL default '0000-00-00',  
  `Hora` time NOT NULL default '00:00:00',  
  `DiasUteis` int(11) NOT NULL default '0',  
  `RespostaFinal` longtext NOT NULL,  
  `ResponsavelResposta` varchar(100) NOT NULL default '0',  
  `UnidadeResposta` int(11) NOT NULL default '0',  
  `Usuario` int(11) NOT NULL default '0',  
  `Unidade` int(11) NOT NULL default '0',  
  `manifestacao` int(10) unsigned NOT NULL default '0',  
  PRIMARY KEY (`ID`)  
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
```

```
CREATE TABLE `estado` (  
  `ID` int(11) unsigned NOT NULL auto_increment,  
  `Sigla` char(2) NOT NULL default '',  
  `Nome` varchar(100) NOT NULL default '',  
  PRIMARY KEY (`ID`),  
  UNIQUE KEY `Nome` (`Nome`),  
  UNIQUE KEY `Sigla` (`Sigla`)  
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
```

```
CREATE TABLE `feriado` (  
  `ID` int(11) unsigned NOT NULL auto_increment,  
  `Data` date NOT NULL default '0000-00-00',  
  PRIMARY KEY (`ID`),
```

```
UNIQUE KEY `Data` (`Data`)  
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
```

```
CREATE TABLE `identificacao` (  
  `ID` int(11) NOT NULL auto_increment,  
  `Nome` varchar(100) NOT NULL default '',  
  PRIMARY KEY (`ID`),  
  UNIQUE KEY `Nome` (`Nome`)  
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
```

```
CREATE TABLE `manifestacao` (  
  `id` int(11) unsigned NOT NULL auto_increment,  
  `codigo_Manifestacao` bigint(20) unsigned NOT NULL default '0',  
  `Descricao` longtext NOT NULL,  
  `Data_Ocorrencia` datetime NOT NULL default '0000-00-00 00:00:00',  
  `Recebimento` bigint(20) unsigned NOT NULL default '0',  
  `Envio` bigint(20) unsigned default '0',  
  `Envolvidos` longtext,  
  `manifestante` bigint(20) unsigned NOT NULL default '0',  
  `Unidade_Ocorrencia` int(11) NOT NULL default '0',  
  `modo_entrada` int(11) NOT NULL default '0',  
  `Modo_Retorno` int(11) NOT NULL default '0',  
  `Identificacao` int(11) NOT NULL default '0',  
  `Assunto` int(11) NOT NULL default '0',  
  `UnidadeResponsavel` int(11) NOT NULL default '0',  
  `Nova` char(1) NOT NULL default 'Y',  
  `ultimaModificacao` date NOT NULL default '0000-00-00',  
  `dataPrevistaResposta` date NOT NULL default '0000-00-00',  
  `dataPrevistaConclusao` date NOT NULL default '0000-00-00',  
  `Classificacao` int(11) NOT NULL default '0',  
  PRIMARY KEY (`id`)  
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
```

```
CREATE TABLE `manifestantes` (  
  `ID` bigint(20) unsigned NOT NULL auto_increment,  
  `nome` varchar(100) default '',  
  `Endereco` varchar(100) default '',  
  `Bairro` int(11) unsigned default '0',  
  `Cidade` int(11) unsigned default '0',  
  `Estado` int(11) unsigned default '0',  
  `Telefone1` varchar(25) default '',
```

```

`Telefone2` varchar(25) default '',
`cep` varchar(10) default '0',
`Fax` varchar(25) default '',
`Email` varchar(100) default '',
PRIMARY KEY (`ID`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

CREATE TABLE `modo` (
  `ID` int(11) NOT NULL auto_increment,
  `Nome` varchar(100) NOT NULL default '',
  `Retornavel` char(1) NOT NULL default 'N',
  PRIMARY KEY (`ID`),
  UNIQUE KEY `Nome` (`Nome`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

CREATE TABLE `outra_unidade` (
  `ID` int(11) NOT NULL auto_increment,
  `Nome` varchar(100) NOT NULL default '',
  `Manifestacao` bigint(20) unsigned NOT NULL default '0',
  PRIMARY KEY (`ID`),
  UNIQUE KEY `Manifestacao` (`Manifestacao`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

CREATE TABLE `outracidade` (
  `ID` int(11) NOT NULL auto_increment,
  `Descricao` varchar(100) NOT NULL default '',
  `Manifestante` bigint(20) unsigned default '0',
  PRIMARY KEY (`ID`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

CREATE TABLE `outro_assunto` (
  `ID` int(11) NOT NULL auto_increment,
  `Nome` varchar(100) NOT NULL default '',
  `Manifestacao` bigint(20) unsigned NOT NULL default '0',
  PRIMARY KEY (`ID`),
  UNIQUE KEY `Manifestacao` (`Manifestacao`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

DROP TABLE IF EXISTS `perfil`;
CREATE TABLE `perfil` (
  `ID` int(11) NOT NULL auto_increment,

```

```

`Nome` varchar(100) NOT NULL default '',
PRIMARY KEY (`ID`),
UNIQUE KEY `Nome` TYPE BTREE (`Nome`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

```

```

CREATE TABLE `prazos` (
  `id` int(11) NOT NULL auto_increment,
  `Nivel` varchar(100) NOT NULL default '0',
  `Situacao` varchar(50) NOT NULL default '',
  `prazo` int(11) NOT NULL default '0',
  PRIMARY KEY (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

```

```

CREATE TABLE `recebimentos` (
  `ID` bigint(20) unsigned NOT NULL auto_increment,
  `Data` date NOT NULL default '0000-00-00',
  `Hora` time NOT NULL default '00:00:00',
  `Origem` varchar(30) NOT NULL default '',
  `Destino` int(11) NOT NULL default '0',
  `manifestacao` int(10) unsigned NOT NULL default '0',
  PRIMARY KEY (`ID`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

```

```

CREATE TABLE `respostas` (
  `ID` bigint(20) unsigned NOT NULL auto_increment,
  `Data` date NOT NULL default '0000-00-00',
  `Resposta` longtext NOT NULL,
  `Responsavel` varchar(100) NOT NULL default '',
  `Cargo` int(11) NOT NULL default '0',
  `Origem` int(11) NOT NULL default '0',
  `Destino` int(11) NOT NULL default '0',
  `Manifestacao` int(11) unsigned NOT NULL default '0',
  PRIMARY KEY (`ID`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

```

```

CREATE TABLE `unidade` (
  `ID` int(11) NOT NULL auto_increment,
  `Sigla` varchar(5) NOT NULL default '',
  `Nome` varchar(100) NOT NULL default '',

```

```

`Endereco` varchar(100) NOT NULL default '',
`Bairro` int(11) unsigned NOT NULL default '0',
`Cidade` int(11) unsigned NOT NULL default '0',
`Estado` int(11) unsigned NOT NULL default '0',
`Telefone1` varchar(25) NOT NULL default '',
`Telefone2` varchar(25) NOT NULL default '',
`Fax` varchar(25) NOT NULL default '',
`Email` varchar(100) NOT NULL default '',
`CEP` varchar(9) NOT NULL default '0',
`prazoRespostal` int(11) NOT NULL default '0',
`prazoResposta2` int(11) NOT NULL default '0',
`responsavel` int(11) NOT NULL default '0',
`Vinculo` int(11) NOT NULL default '0',
PRIMARY KEY (`ID`),
UNIQUE KEY `Nome` (`Nome`),
UNIQUE KEY `Sigla` (`Sigla`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

```

```

CREATE TABLE `usuarios` (
  `ID` int(11) NOT NULL auto_increment,
  `Nome` varchar(100) NOT NULL default '',
  `User` varchar(100) NOT NULL default '',
  `Senha` varchar(32) NOT NULL default '',
  `Perfil` int(11) default '0',
  `Cargo` int(11) default '0',
  `email` varchar(100) default '',
  `telefone` varchar(20) default '',
  `Unidade` int(11) default NULL,
  PRIMARY KEY (`ID`),
  UNIQUE KEY `User` (`User`),
  KEY `Nome` (`Nome`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

```