UNIVERSIDADE FEDERAL DO CEARÁ - UFC
DEPARTAMENTO DE COMPUTAÇÃO - DC
ESPECIALIZAÇÃO EM TECNOLOGIAS DA INFORMAÇÃO - ETI

Reúso em Desenvolvimento de *Software* para a *Web* Através de *Frameworks*

por

Diego Silveira Costa Nascimento

Fortaleza – Ceará,
Abril de 2007.

Diego Silveira Costa Nascimento

Reúso em Desenvolvimento de *Software* para a *Web* Através de *Frameworks*

Trabalho realizado como requisito parcial para a obtenção do título de Especialista em Tecnologias da Informação com Ênfase no Desenvolvimento Web.

Orientador: M.Sc. José Maria da Silva Monteiro Filho

Fortaleza - Ceará,
Abril de 2007.

Reúso em Desenvolvimento de *Software* para a *Web* Através de *Frameworks*

Diego Silveira Costa Nascimento

PARECER: _____

DATA: ___/___/___

BANCA EXAMINADORA:

_____

M.Sc. José Maria da Silva Monteiro Filho
(Orientador)

_____

Dra. Vânia Maria Ponte Vidal
(Coordenadora)

_____

M.Sc. Fernando Cordeiro de Lemos
(Professor)

Agradecimentos

A Deus, por iluminar meus caminhos, e pela força para superar os obstáculos.

Aos meus pais, Petrúcio e Arlete, que como ninguém, sempre estiveram comigo na busca da realização dos meus intermináveis sonhos. Pela a vida que me deram, pelos valores que sempre procuraram me passar, e por cada momento que vivemos juntos.

A minha irmã Andreza, que sem dúvidas, mesmo da maneira só dela, estará do meu lado sempre que eu precisar.

As tias, Alice e Araci, pela ajuda nos livros e presença em Fortaleza.

Aos meus familiares, avós, avôs, tias, tios e primos, por sempre me incentivarem nos meus ideais.

A pessoa com quem tive o primeiro contato quando cheguei a belíssima cidade de Fortaleza, não só me por me receber e mostrar os diversos e novos lugares, mas também, por fazer parte de cada dia dessa inesquecível experiência em minha vida, Rosana, como várias vezes pronunciei, “- Minha secretária preferida!”.

Aos professores que contribuíram com seus conhecimentos e experiências para nossa formação. Lembrando, a professora, Dra. Vânia Maria Ponte Vidal, pela realização do excelente curso, e em especial, ao professor, amigo, orientador e genuíno educador, José Maria da Silva Monteiro Filho, por sua dedicação, indescritível, para a realização deste trabalho, e aos colaboradores da Universidade Federal do Ceará pela dedicação.

Aos amigos, Angelina Costa, Dênis Roberto, Lorena Barros e Orleans Mota, pelas várias horas, juntos, dedicadas aos diversos trabalhos da especialização. E por todos que participaram de forma direta ou indiretamenta de mais essa etapa.

E a todos vocês, meu muito OBRIGADO!!!!!

"Nós somos o que fazemos repetidamente, a excelência não é um feito, e sim, um hábito."

(Aristóteles)

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Quadros

Lista de Abreviações

API – Application Programming Interface
CSS – Cascading Style Sheets
CSV – Comma-Separated Values
DTD – Document Type Definition
EJB – Enterprise JavaBeans
HTML – HyperText Markup Language
HTTP – HyperText Transfer Protocol
IDE – Integrated Development Environment
JCP – Java Community Process
JDBC – Java Database Connectivity
JDK – Java Development Kit
JEE – Java Enterprise Edition
JSF – JavaServer Faces
JSP – Java Server Pages
JTA – Java Transaction API
MVC – Model, View e Controller
ORB – Object Request Broker
PDF – Portable Document Format
POJO – Plain Old Java Objects
SQL – Structured Query Language
UML – Unified Modeling Language
UI – User Interface
WTP – Web Tools Platform
XLS – Excel Files
XML – eXtensible Markup Language
XSD – XML Schema Definition

1 Introdução

A necessidade de se desenvolver sistemas de forma cada vez mais rápida aliada à exigência de se gerar produtos de *software* de extrema qualidade, vem obrigando as empresas a optarem por tecnologias auxiliares para cumprir suas metas de produção.

Desta forma, um dos principais objetivos da Engenharia de *Software* consiste em conceber técnicas e tecnologias que possibilitem o reúso de *software*, através do qual obtém-se a redução do esforço de desenvolvimento e o aumento da qualidade do produto de *software* gerado [16].

Reúso de *software* é o processo de criar sistemas a partir de artefatos ou conhecimentos de *software* já existentes. Esta simples e poderosa idéia foi introduzida em 1968, durante a conferência de engenharia de *software* da *NATO*. Durante este período, o reúso tem sido visto como a solução para a crise de *software* e técnica crucial na melhoria da produtividade de *software* [16]

No contexto da Engenharia de Software, diferentes abordagens buscam melhorar a qualidade dos artefatos de software, bem como diminuir o tempo e o esforço necessários para produzi-los. *Frameworks* são estruturas de classes que constituem implementações incompletas que, estendidas, permitem produzir diferentes artefatos de *software*. A grande vantagem desta abordagem é a promoção de reúso de código e projeto, que pode diminuir o tempo e o esforço exigidos na produção de *software*. Em contrapartida, é complexo desenvolver *frameworks*, bem como aprender a usá-los. A abordagem de *frameworks* pode se valer de padrões para a obtenção de estruturas de classes bem organizadas e mais aptas a modificações e extensões [15].

Frameworks encapsulam detalhes de implementação voláteis através de seus pontos de extensão, interfaces estáveis e bem definidas, aumentando a modularidade da aplicação. Os locais de mudanças de projeto e implementação da aplicação construída usando o *framework* são localizados, diminuindo o esforço para entender e manter a aplicação [15].

Entretanto, estamos vivenciando atualmente uma explosão de diversos frameworks disponíveis. Muitos deles se propõem a solucionar problemas semelhantes. Além disso, a integração entre eles nem sempre é fácil de se perceber, e muito menos de implementar. Tudo isso pode dificultar a adoção de *frameworks*, principalmente pelos desenvolvedores mais jovens.

Este trabalho se propõe a discutir, de forma didática, um pouco destas tecnologias, destacando seus benefícios e limitações, através da implementação evolutiva de uma aplicação *Web* (estudo de caso). Assim, buscamos apresentar os principais *frameworks* que fornecem suporte ao desenvolvimento de *software* para a *Web*. A cada etapa da construção (*refactoring*) do estudo de caso, um novo *framework* é introduzido, sua integração com as tecnologias anteriormente utilizadas é apresentada e as vantagens obtidas são discutidas.

O restante deste trabalho está organizado da seguinte forma: O capítulo 2 aborda os conceitos teóricos de frameworks, descrevendo suas definições, tipos, classificações, pontos positivos, pontos negativos e exemplos disponíveis no mercado. O capítulo 3 descreve o estudo de caso (problema prático do mundo real) a ser implementado, um sistema bancário, a partir da utilização de *frameworks*. O capítulo 4 apresenta a evolução (*refactoring*) do estudo de caso, mostrando, de forma evolutiva, o amadurecimento do desenvolvimento da aplicações exemplo. Para isso serão discutidas nove implementações: Banco com *Servlet/JDBC*, *JSP/JDBC*, *JSP/Servlet/MVC/JDBC*, *Struts/MVC/JDBC*, *JavaServer Faces/MVC/JDBC*, *JavaServer Faces/MVC/DAO*, *JavaServerFaces/MVC/DAO*, *JavaServerFaces/MVC/DAO/Validação* e *JavaServerFaces/MVC/DAO/Validação/Hibernate*. O capítulo 5 apresenta as conclusões obtidas e aponta direções para trabalhos futuros.

2 Framework

Framework é um conjunto de classes que colaboram entre si para fornecer um serviço a uma parte invariante de um subsistema lógico [1].

Com o crescimento da demanda de desenvolvimento para serviços computacionais, cada vez mais, empresas da área de criação de sistemas, estão explorando os benefícios dos *frameworks* em seus projetos, buscando reusabilidade, customização, fácil manutenção e, principalmente, agilidade no processo de fabricação, garantindo a qualidade final por meio de elementos que já foram exaustivamente testados e que acoplam uma série de funcionalidades importantes: como segurança, persistência, conectividade, entre outras.

Existe uma quantidade suficiente de *frameworks* disponível no mercado, entre pagos e gratuitos, que atendem a diversas necessidades e soluções. A reusabilidade obtida a partir da utilização destes *frameworks*, quando bem trabalhada, proporciona vários benefícios para todo o processo de desenvolvimento.

2.1 Outras Definições

“Aplicação semi-completa, reutilizável que, quando especializada, produz aplicações personalizadas”.

(Johnson & Foote, 1988)

“Coleção de classes abstratas e concretas e a interface entre elas, representando o projeto de um sub-sistema”.

(Pree, 1995)

“Uma arquitetura desenvolvida com o objetivo de se obter a máxima reutilização, representada como um conjunto de classes abstratas e concretas, com grande potencial de especialização”.

(Mattsson – 1996)

2.2 Características dos Frameworks

Os *frameworks* podem assumir várias características, mas as mais importantes são[2]:

- Um *framework* deve ser reusável;
- Deve ser extensível;
- Deve ser de uso seguro;
- Deve ser eficiente;
- Deve ser completo.

2.3 Pontos Positivos

A seguir, destacamos as principais vantagens que podem ser obtidas através da utilização de *frameworks*[3,4,5].

- Reusabilidade de código;
- Agiliza o processo de desenvolvimento;
- Facilidade na manutenção do código;
- Mantém comportamento padrão nas aplicações;
- Permite unificar trabalhos que são de domínios de áreas distintas, como desenvolvimento, negócio ou projetos de interfaces;
- Em sua maioria, implementam Padrões de Projetos que têm uma forte consagração, tanto no mercado, quanto na área acadêmica;
- São conhecidas por funcionarem bem em outras aplicações;
- Estão prontos para serem usados com o próximo projeto;
- Pode ser usado por outras equipes na organização;
- Não se aplica apenas às aplicações, mas também, aos componentes da mesma;
- Menor quantidade de erros no desenvolvimento;
- Produto com melhor qualidade no produto final;
- Padronização.

2.4 Pontos Negativos

A seguir, destacamos as principais desvantagens que podem advir da utilização de *frameworks* [3,4,5].

- Dificuldade para se desenvolver bons *frameworks*, pois é necessário ter uma boa experiência no domínio;
- A documentação é essencial para o usuário (desenvolvedor) poder utilizar o *framework*;
- Dificuldade para se manter compatibilidade com versões anteriores, já que *frameworks* se tornam mais maduros com o passar do tempo e as aplicações devem evoluir em paralelo;
- O processo de depuração pode se tornar complicado;
- A flexibilidade e a generalização do *framework* podem trabalhar contra sua eficiência em algumas aplicações.

2.5 Diferenças entre Framework e Bibliotecas de Classes

Nesta seção, destacamos as principais diferenças entre *frameworks* e bibliotecas de classe conforme apresentado na tabela 2.1.

Frameworks	Bibliotecas de Classes
• Customização com subclasse ou composição; • Chama funções de aplicação; • Controla o fluxo de execução; • Define interação entre os objetos; • Prover comportamento padrão.	• Classes instanciadas pelo cliente; • Cliente chama função; • Não tem fluxo de controle pré-definido; • Não tem interação pré-definida; • Não tem comportamento padrão.

Tabela 2.1 – Comparação entre *frameworks* e bibliotecas de classes[2].

A figura 2.1 mostra graficamente, a diferença entre *frameworks* e bibliotecas de classes.

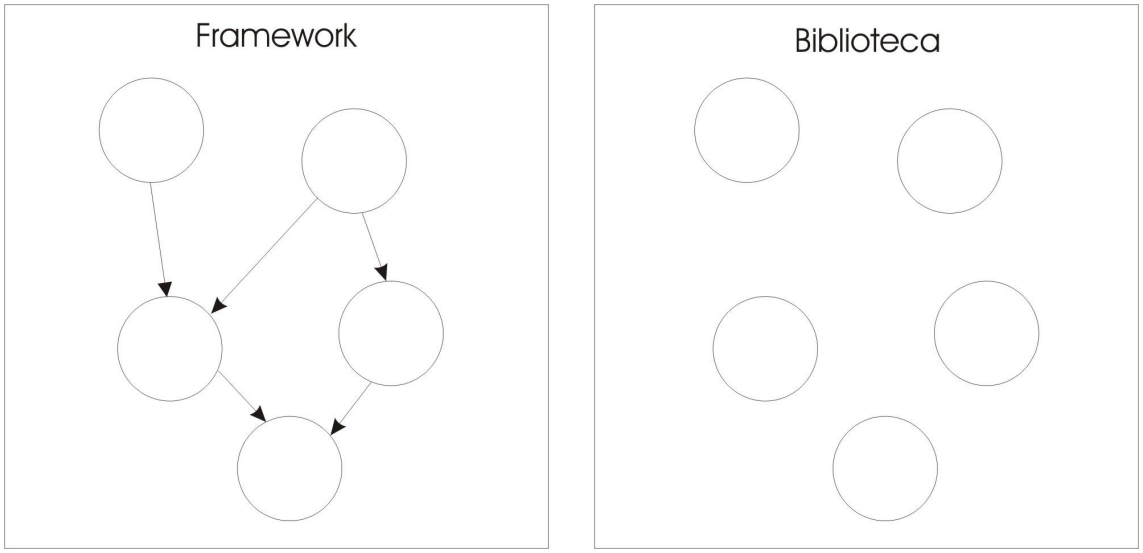

Figura 2.1 – Diferença entre *framework* e bibliotecas de classes[2].

2.6 Tipos de Frameworks

Nesta seção, descrevemos os diferentes tipos de *frameworks*.

- *Frameworks* Caixa Branca: Reúso provido por Herança;
- *Frameworks* Caixa Preta: Reúso provido por composição;
- *Frameworks* Caixa Cinza: Reúso provido da mistura das duas tecnologias.

O uso de cada tipo de *framework* proporciona outras facilidades, como é mostrado na figura 2.2. No fator relacionado ao desenvolvimento, o *framework* do tipo caixa branca é o que apresenta maior facilidade de desenvolvimento dessa tecnologia, seguido por caixa cinza e por último, caixa preta. Já no fator relacionado á usabilidade por parte do programador, o *framework* do tipo caixa preta é o que melhor apresenta melhores resultados, seguido por caixa cinza e por último, caixa branca.

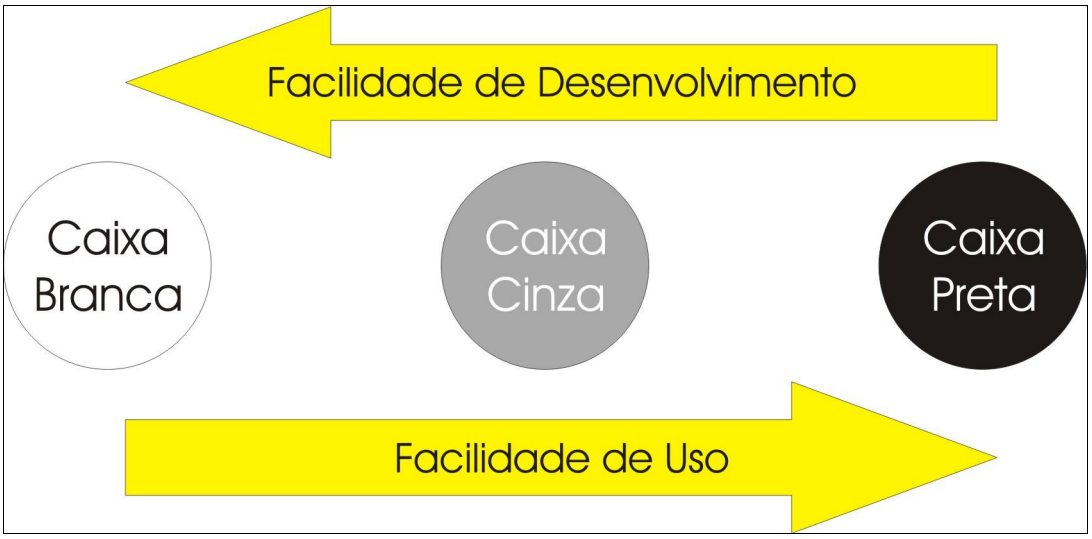

Figura 2.2 – Tipos de *frameworks* quando sua facilidade de desenvolvimento e facilidade de uso [3].

2.7 Classificação de Frameworks de Aplicação

Nesta sessão, são mostrados os *frameworks* de aplicação. Esses frameworks podem ser classificados como: *framework de infra-estrutura do sistema*, *framework de integração de middleware* e *framework de aplicação empresarial*.

2.7.1 Framework de Infra-estrutura do Sistema

São *frameworks* que simplificam o desenvolvimento da infra-estrutura de sistemas portáveis e eficientes. Em geral são usados internamente em uma organização de software e não são vendidos a clientes diretamente. Como exemplos deste tipo de *framework*, podemos citar: comunicação, interfaces com o usuário e ferramentas de processamento de linguagem.

2.7.2 Framework de Integração de Middleware

São *frameworks* usados, em geral, para integrar aplicações e componentes distribuídos. Projetados para melhorar a habilidade dos desenvolvedores em

modularizar, reutilizar e estender sua infra-estrutura de software para funcionar “sem costuras” em um ambiente distribuído. Como exemplos deste tipo de *framework*, podemos citar: *Object Request Broker (ORB)*, *middleware* orientado a mensagens e bases de dados transacionais.

2.7.3 Framework de Aplicação Empresarial

São *frameworks* voltados a domínios de aplicação mais amplos e a peça fundamental para atividades de negócios das empresas. São mais caros para desenvolver ou comprar, mas podem dar um retorno substancial do investimento, já que permitem o desenvolvimento de aplicações e produtos diretamente. Como exemplo desses tipos de *frameworks*, podemos citar: telecomunicação, aviação, manufatura e engenharia financeira.

Esses são exemplos de *frameworks* que já são consagrados no mercado de desenvolvimento de software:

- *Apache Cocoon*
- *Apache Struts*
- *Apache Ant*
- *Hibernate*
- *iReport / JasperReport*
- *JavaServer Faces*
- *Tiles*
- *Spring*
- *Velocity*

2.8.1 Apache Cocoon

O *Apache Cocoon*, muitas vezes chamado apenas de *Cocoon*, é um *framework* de desenvolvimento para *web* baseado em componentes e no conceito de separação de interesses. O *framework* é focado na publicação de *XML* e é construído na linguagem de programação *Java*. A flexibilidade proporcionada pelo uso do *XML* permite a publicação de conteúdo de maneira rápida e numa grande variedade de formatos, incluindo *HTML*, *PDF*, e *WML*. *Cocoon* é também muito usado como uma ferramenta de *data warehousing* ou como um *middleware* para transporte de dados entre sistemas.[6]

2.8.*2 Apache Struts*

Struts é um *framework* de desenvolvimento da camada controladora, numa estrutura seguindo o padrão Model 2 (uma variante do *MVC* oficializada pela *Sun*), de aplicações *web* construído em Java, para ser utilizado em um *container web* em um servidor *J2EE*.[7]

2.8.3 *Hibernate*

O *Hibernate* é um *framework* de acesso a banco de dados escrito em Java. Ele é um *software* livre e de código aberto. O objetivo do *Hibernate* é facilitar a construção de aplicações Java que utilizam de bases de dados relacionais. O uso de ferramentas de mapeamento objeto relacional, como o *Hibernate*, diminui a complexidade resultante da convivência de modelos diferentes; o modelo orientado a objetos (da linguagem Java) e o relacional (da maioria dos *SGBDs*). Ele é responsável apenas pelo mapeamento das tabelas do modelo relacional para classes da linguagem Java. As questões relacionadas ao gerenciamento de transações e a tecnologia de acesso à base de dados são de responsabilidade de outros elementos da infra-estrutura da aplicação. Apesar da *API* do

Hibernate possuir operações para o controle transacional, por exemplo, ele simplesmente delega estas tarefas para infra-estrutura no qual foi instalada[8].

2.8.4 *JasperReport*

JasperReports é um *framework* para geração de relatórios com uma rica habilidade para organizar e apresentar o conteúdo gerado. Gera impressão em *PDF*, *HTML*, *XLS*, *XML* e *CSV*. Esse *framework* pode ser usado em qualquer aplicação Java. Seu principal objetivo é auxiliar na criação e impressão de documentos.

2.8.5 *JavaServer Faces*

JavaServer Faces é um *framework MVC* para o desenvolvimento de aplicações *web*, que permite o desenvolvimento de aplicações para a *Internet* tal como fazíamos com *Delphi* ou *Visual Basic*, ou seja, arrastando e soltando os componentes na tela *JSP*, definindo propriedades dos mesmos, etc.

O *JSF* é atualmente considerado por muitos como a última palavra em termos de desenvolvimento de aplicações *web*, resultado da experiência e maturidade adquiridas com o *JSP*, *Servlet*, Modelo *MVC* e *Struts*.

Características:

- Permite que o desenvolvedor crie *UIs* através de um conjunto de componentes *UIs* pré-definidos;
- Fornece um conjunto de *tags JSP* para acessar os componentes;
- Reúsa componentes da página;
- Associam os eventos do lado do cliente com os manipuladores dos eventos do lado do servidor (os componentes de entrada possuem um valor local representando o estado no lado servidor);
- Fornece separação de funções que envolvem a construção de aplicações *web*[10].

2.8.6 *Tiles*

O *Tiles* é um *framework* que permite a criação de *templates* de apresentação, ou seja, é uma forma mais avançada de ação de inclusão de *JSP*. Em uma aplicação *Tiles*, o *template* do segundo plano ou *layout* geralmente são incluídos para preencher cada uma dessas posições. Se o cabeçalho mudar, então apenas esse arquivo de *template* precisará ser alterado. A alteração é refletida automaticamente em todas as páginas que incluem esse *template*. Os componentes *HTML* padrões, como as *CSS*, também funcionam bem como *templates* dinâmicos. Uma folha de estilo pode ajudar a manter os *templates* consistentes internamente e minimizadamente mais os efeitos da alteração[11].

2.8.7 *Spring*

O *Spring* é um *framework* de código aberto criado para endereçar a complexidade de desenvolvimento de aplicativos corporativos. O *Spring* torna possível usar *POJOs* para alcançar coisas que previamente só eram possíveis com *EJBs*. Porém, a utilidade do *Spring* não é limitada ao desenvolvimento de aplicações *server-side*. Qualquer aplicativo escrito na linguagem Java pode se beneficiar do *Spring* em termos de simplicidade, testabilidade e agrupamento. Ele também consegue utilizar de recuros de programação orientada a aspectos[12].

2.8.8 *Velocity* [13]

Velocity é um versátil *framework* do grupo *Jakarta*, feito para interpretar e renderizar *templates*, fazendo com que a relação entre programador e *webdesigner* seja mais agradável e produtiva, permitindo que *designers* possam trabalhar sem ter de interferir em algo mais complexo, como *JSP*.

3 Sistema Desenvolvido

Este trabalho se propõe a desenvolver uma aplicação real na qual se possa demonstrar, evolutivamente, as práticas na utilização de *frameworks* na produção de *software*. Neste sentido, desenvolvemos a clássica aplicação bancária, onde um colaborador bancário pode executar diversas atividades diariamente, na manutenção das contas de seus clientes conforme mostrado na Figura 3.1.

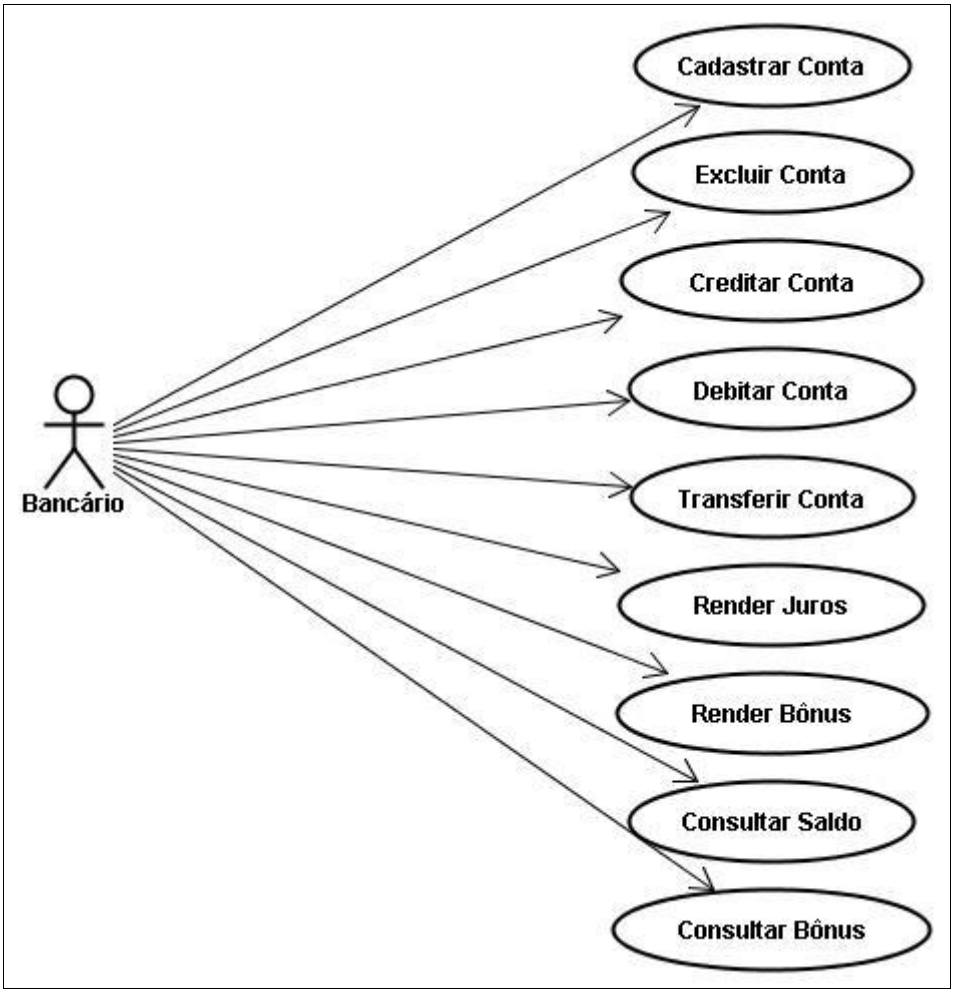

Figura 3.1 – Diagrama de caso de uso.

3.1 Frameworks Utilizados

Neste trabalho utilizamos os seguintes *frameworks*:

- *Hibernate*
- *JavaServer Faces*
- *Struts*

3.2 Ferramentas Utilizadas

Utilizamos várias ferramentas disponíveis no mercado, de livre licença, do ambiente de desenvolvimento, ferramentas para modelagem, banco de dados, *container web* e máquina virtual.

3.2.1 *IDE*

Eclipse WTP é um ferramenta para desenvolvimento de aplicações *J2EE*. Essa ferramenta já inclui *plugins* para trabalhar com diversas tecnologias já bem difundidas para *web* como: editor *HTML*, *javascript*, *CSS*, *JSP*, *SQL*, *XML*, *DTD* e *XSD*.

3.2.2 Modelagem

Jude é uma ferramenta de modelagem *UML* de sistemas orientado a objeto. Ferramenta gráfica que permite criar diversos diagramas: caso de uso, classe, seqüência, estado, atividade, colaboração, componente e desenvolvimento.

3.2.3 Persistência de Dados

Oracle 10g Express Editon é um banco de dados que foi baseado no *10g release 2*, com livre licença para desenvolvimento e distribuição. É de simples instalação e administração.

3.2.4 Container *Web*

O *Tomcat* é um servidor de aplicações *Java* para *web*. É um *software* livre e de código aberto surgido dentro do conceituado projeto *Apache Jakarta* e que foi oficialmente endossado pela *Sun* como Implementação de Referência (RI) para as tecnologias Java *Servlet* e *JavaServer Pages*. O *Tomcat* é robusto e eficiente o suficiente para ser utilizado mesmo em um ambiente de produção. Tecnicamente, é um *Container web*, parte da plataforma corporativa *J2EE* que abrange as tecnologias *Servlet* e *JSP*. Ele tem capacidade de atuar também como servidor *web/HTTP* autônomo, ou pode funcionar integrado a um servidor *web* dedicado, como o *Apache HTTPD* ou o *Microsoft IIS*, provendo a parte dinâmica de Java *Servlet* e *JSP*.

3.2.5 *JDK*

O *JDK* é um *kit* de desenvolvimento *Java* fornecido livremente pela *Sun*. Constitui um conjunto de programas que engloba compilador, interpretador e utilitários, fornecendo um pacote de ferramentas básicas para o desenvolvimento de aplicações *Java*.

4 Estudo de Caso

A qual estamos acostumados, um banco é uma instituição financeira que se dedica a atividades relacionadas ao recebimento de depósitos, aplicações de dinheiro, fazer empréstimos de curto e longo prazo, receber contas e impostos, entre outras.

Utilizando dessa idéia, simulamos um sistema que registra e controla as contas dos seus clientes. O sistema trabalha com três tipos específicos de contas: uma conta do tipo simples, uma conta do tipo poupança e ou uma outra do tipo bônus. Cada conta tem suas características de serviços comuns e distintos. A conta do tipo simples, como o próprio nome diz, é o tipo de conta mais simples, e que permite apenas os serviços de operações de crédito e débito. A conta do tipo poupança é uma conta, que também permite operações de uma conta do tipo simples, mas oferece o serviço a mais de rendimento acumulativo do valor do saldo a partir de um valor de juros. Já a conta do tipo bônus, que também permite as operações de uma conta simples, não permite o serviço extra que possui a conta do tipo poupança, mas possui sua particularidade, onde permite uma bonificação a cada operação de crédito. A figura 4.1. mostra o diagrama de classe que representa a estrutura dessas contas.

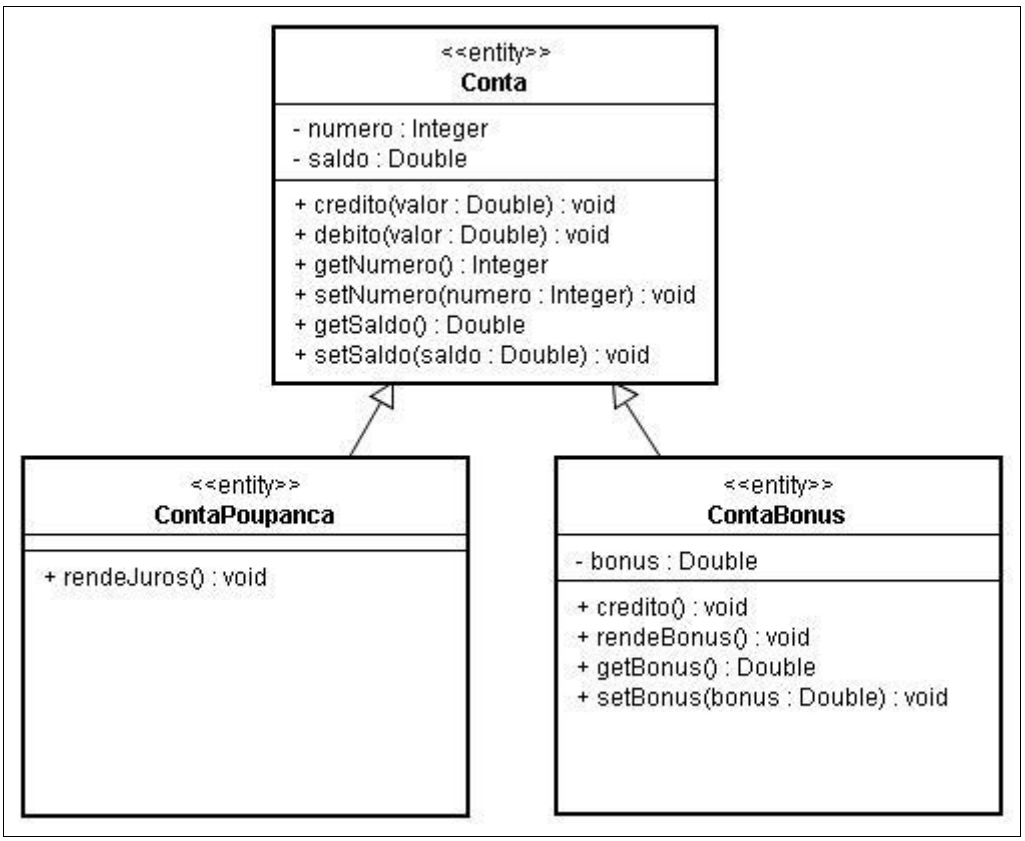

Figura 4.1 – Diagrama de modelagem de contas.

A idéia por trás da herança é poder reutilizar ou alterar os métodos de classes existente, bem como adicionar novos campos de instâncias e novos métodos a fim de adaptá-los a novas situações[20].

O sistema também tem que permitir que operações sejam realizadas para que, de fato, o mesmo funcione. No banco, os colaboradores, em especial os bancários, cumprem com suas atribuições, na qual chamamos de negócio: abertura de contas, operação de crédito e débito, transferências de valores entre contas, operação de rendimento de juros e bônus, e consultas de saldo e bônus.

A figura 4.2 mostra o diagrama de classe que representa o negócio de um banco para as operações de conta.

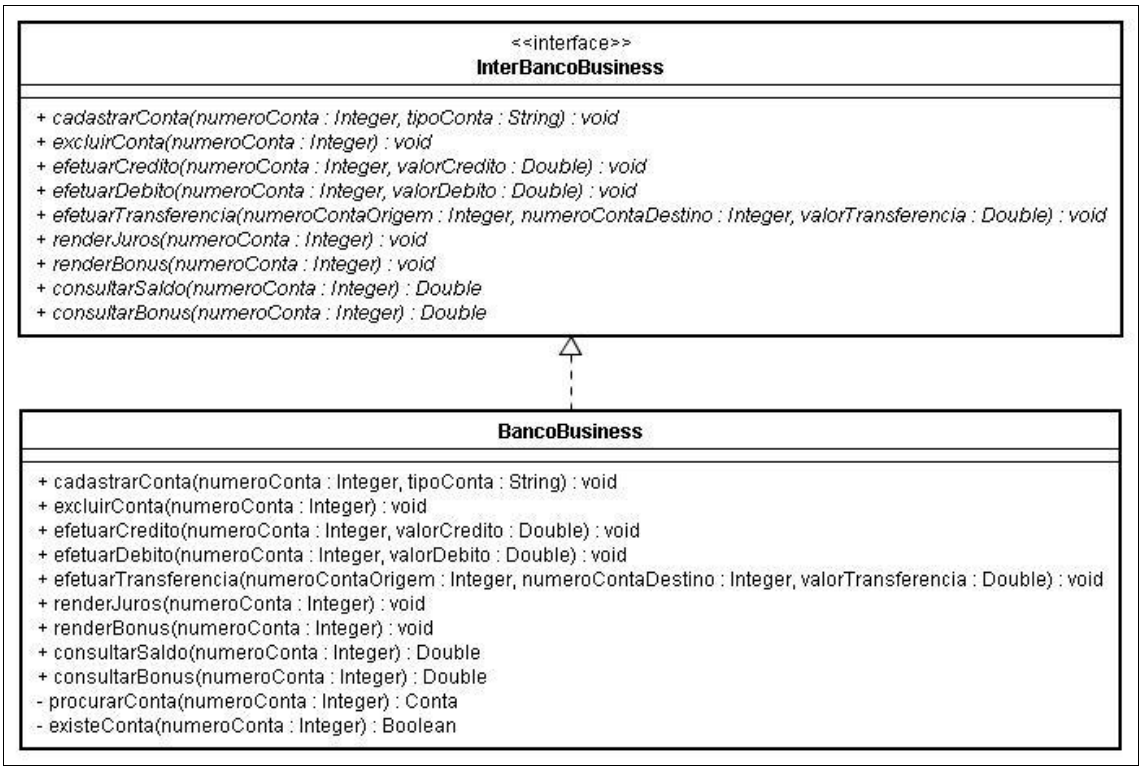

Figura 4.2 – Classe de negócio para operações de conta.

A utilização de uma implementação a partir de uma interface é uma promessa de que uma classe vai implementar certos métodos com certas características. A maneira com que esses métodos são implementados depende da própria classe, evidentimente[20].

Neste trabalho, foram realizadas oito implementações diferentes testadas em plataforma *web* com tecnologia Java *(J2EE)*. Implementações, essas, mais artesanais, que não proviam da utilização de algum *framework*, ou outras que exploram dos benefícios e facilidades oferecidos por cada *framework* específico.

4.1 Banco com Servlet e JDBC

Neste tópico iniciamos o primeiro exemplo de uma aplicação bancária utilizando apenas *Servlet* e *JDBC*.

Servlet é uma *API Java* disponíveis aos programadores, para gerar conteúdo dinâmico, geralmente em *HTML*, baseado no modelo *request/response*. Os *servlets* normalmete utilizam o protocolo *HTTP*, a pesar de não serem restritos a ele.

JDBC é um conjunto de classes e interfaces *(API)* escritas em *Java* que faz o envio de instruções *SQL* para qualquer banco de dados relacional[14];

Neste teste, desenvolvemos um conjunto de formulários de entrada de dados que representam a parte de interação do bancário, e esses, são responsáveis em chamar os elementos de controle específicos para execução da ação solicitada. Os elementos de controles quando acionados, executam as ações de acordo com as operações dos objetos de negócio*(business)* e em seguida, gera os elementos de resposta. A figura 4.3, mostra o modelo de interação.

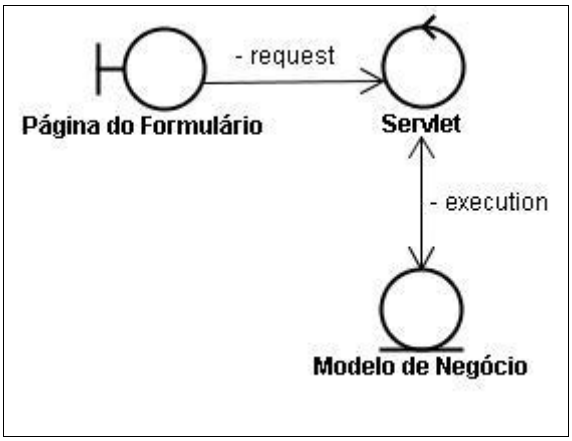

Figura 4.3 – Modelo da interação entre fronteira e controle em *servlet*.

Para o desenvolvimento prático da aplicação bancária com *servlet*, criamos a estrutura organizada em pasta conforme mostrado na figura 4.4.

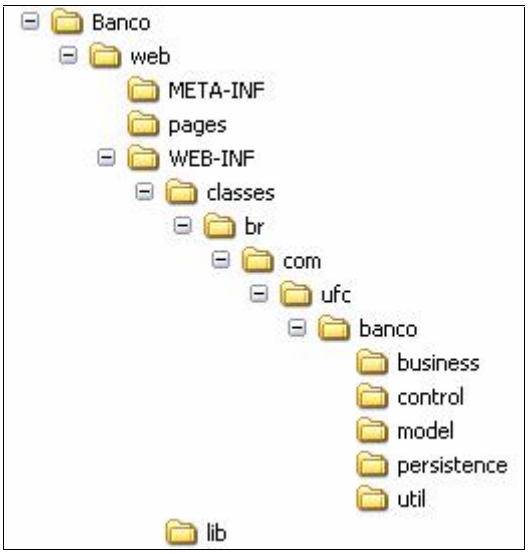

Figura 4.4 – Estrutura do diretório para a aplicação em *servlet*.

O diretório raiz *Banco* é também chamado de *contexto*, é a partir dele que é criada a estrutura da aplicação *web*.

Dentro do diretório *web* é inserido o arquivo da página inicial da interação do bancário. Nesse nosso caso, é um arquivo *HTML* que representa o menu com as possíveis opções das operações do sistema. O nome físico do arquivo, conforme padrão, é chamado de *index.html* e possui conteúdo conforme quadro 4.1.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
        <title>Banco  -  Servlet/JDBC </title>
    </head>
    <body>
      ETI - Especialização em Tecnologia da Informação<br/>
      UFC - Universidade Federal do Ceará

      <h1>Menu</h1>
        <a href="/banco_1/pages/cadastrarConta.html">Cadastrar
Conta</a><br/>
            <a href="/banco_1/pages/excluirConta.html">Excluir
Conta</a><br/>
            <a
href="/banco_1/pages/creditarDebitarConta.html">Creditar/Debitar
Conta</a><br/>
            <a
href="/banco_1/pages/transferirConta.html">Transferência</a><br/>
            <a href="/banco_1/pages/renderJurosConta.html">Render
Juros</a><br/>
            <a href="/banco_1/pages/renderBonusConta.html">Render
Bônus</a><br/>
            <a href="/banco_1/pages/consultarSaldoConta.html">Consultar
Saldo</a><br/>
            <a href="/banco_1/pages/consultarBonusConta.html">Consultar
Bônus</a>
    </body>
</html>
```

Quadro 4.1 – Conteúdo do arquivo *index.html* que representa a tela de menu da aplicação.

A figura 4.5, mostra a interpretação do arquivo *index.html* pelo navegador.

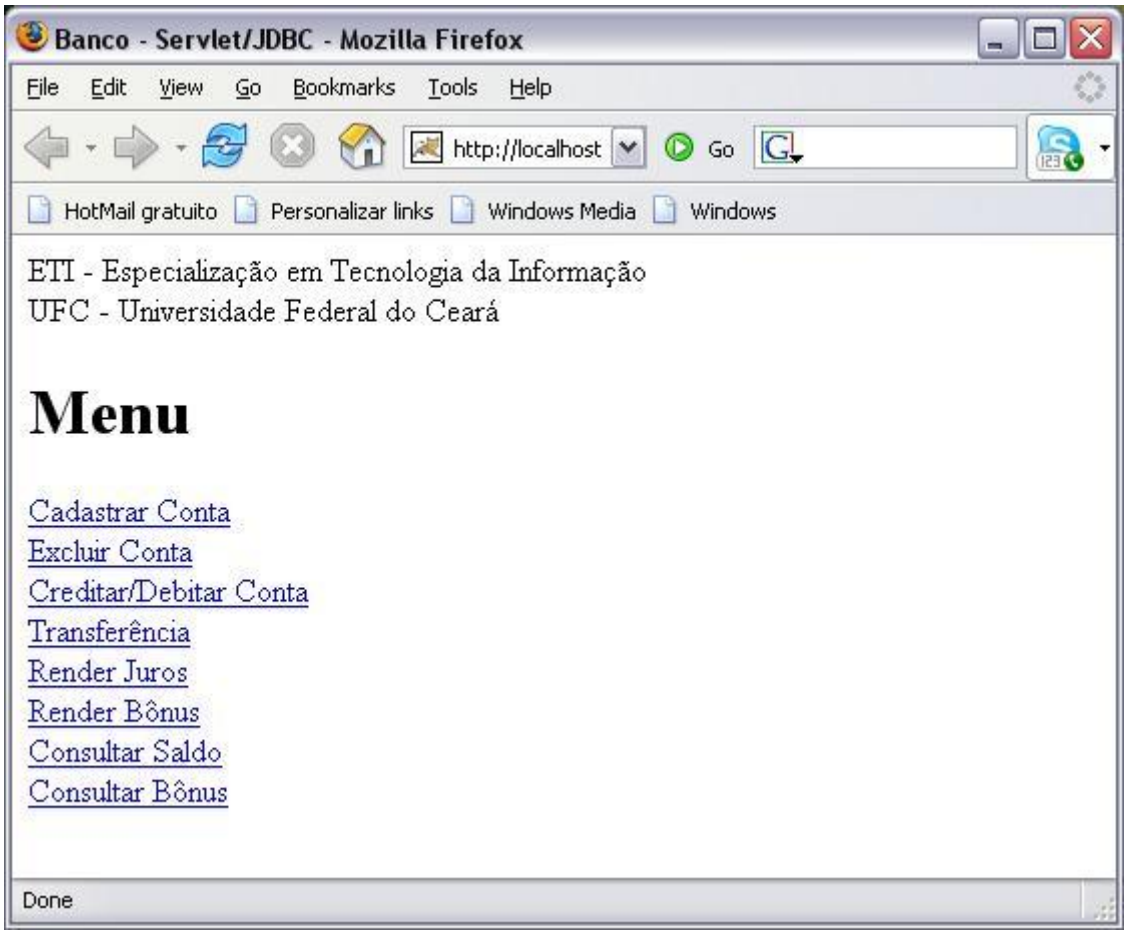

Figura 4.5 – Menu da aplicação.

Para manter os outros formulários de entrada organizados, criamos o diretório *pages*, no qual contém apenas arquivos *HTML* que são chamados pelo menu, para cadastrar conta, excluir conta, creditar e debitar conta, transferência entre contas, render juros, render bônus, consultar saldo e consultar bônus.

O formulário para cadastrar conta é fisicamente chamado de *cadastrarConta.html* e possui o conteúdo mostrado no quadro 4.2.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
        <title>Banco  -  Servlet/JDBC</title>
    </head>
    <body>
```

```
            ETI - Especialização em Tecnologia da Informação<br/>
      UFC - Universidade Federal do Ceará

      <h1>Cadastrar Conta</h1>
      <hr/>
      <form name="conta"
action="/banco_1/servlet/CadastrarContaServlet" method="post">
            <table>
                  <tr>
                        <td>Conta</td>
                        <td>
                              <input type="text" name="numeroConta"/>
                        </td>
                  </tr>
                  <tr>
                        <td>Tipo</td>
                        <td>
                              <select name="tipoConta">
                                    <option value=""></option>
                                    <option value="C">Conta
Simples</option>
                                    <option value="P">Conta
Poupança</option>
                                    <option value="B">Conta
Bônus</option>
                              </select>
                        </td>
                  </tr>
            </table>
            <input type="button" name="btnCadastrar" value="Cadastrar"
onclick="submit();"/>
            <input type="button" name="btnLimpar" value="Limpar"
onclick="limpar();"/>
                  <br/>
                  <br/>
            <a href="/banco_1/index.html">Menu Principal</a>
      </form>
    </body>
    <script type="text/javascript" language="javascript">
      function limpar(){
            document.forms[0].elements["numeroConta"].value = "";
            document.forms[0].elements["tipoConta"].value = "";
      }
    </script>
</html>
```

Quadro 4.2 – Conteúdo do arquivo *cadastrarConta.html* que representa o formulário de cadastrar conta.

A figura 4.6, mostra a interpretação do arquivo *cadastrarConta.html* pelo navegador.

Figura 4.6 – Formulário para cadastrar contas.

O formulário para excluir conta é fisicamente chamado de *excluirConta.html* e possui o conteúdo mostrado no quadro 4.3.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
        <title>Banco  -  Servlet/JDBC</title>
    </head>
    <body>
            ETI - Especialização em Tecnologia da Informação<br/>
      UFC - Universidade Federal do Ceará

      <h1>Excluir Conta</h1>
      <hr/>
      <form name="conta" action="/banco_1/servlet/ExcluirContaServlet"
method="post">
            <table>
```

```
                    <tr>
                          <td>Conta</td>
                          <td>
                                <input type="text" name="numeroConta"/>
                          </td>
                    </tr>
              </table>
              <input type="button" name="btnExcluir" value="Excluir"
onclick="submit();"/>
              <input type="button" name="btnLimpar" value="Limpar"
onclick="limpar();"/>
              <br/>
                    <br/>
              <a href="/banco_1/index.html">Menu Principal</a>
        </form>
    </body>
    <script type="text/javascript" language="javascript">
        function limpar(){
            document.forms[0].elements["numeroConta"].value = "";
        }
    </script>
</html>
```

Quadro 4.3 – Conteúdo do arquivo *excluirConta.html* que representa formulário de excluir conta.

A figura 4.7, mostra a interpretação do arquivo *excluirConta.html* pelo navegador.

Figura 4.7 – Formulário para excluir contas.

O formulário para creditar e debitar conta é fisicamente chamado de *creditarDebitarConta.html* e possui o conteúdo mostrado no quadro 4.4.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
        <title>Banco  -  Servlet/JDBC</title>
    </head>
    <body>
            ETI - Especialização em Tecnologia da Informação<br/>
      UFC - Universidade Federal do Ceará

      <h1>Creditar/Debitar Conta</h1>
      <hr/>
      <form name="conta"
action="/banco_1/servlet/CreditarDebitarContaServlet" method="post">
            <input type="hidden" name="method"/>
            <table>
                  <tr>
                        <td>Conta</td>
                        <td>
                              <input type="text" name="numeroConta"/>
                        </td>
                  </tr>
                  <tr>
                        <td>Valor</td>
                        <td>
                              <input type="text" name="valor"/>
                        </td>
                  </tr>
            </table>
            <input type="button" name="btnCreditar" value="Creditar"
onclick="creditar();"/>
            <input type="button" name="btnDebitar" value="Debitar"
onclick="debitar();"/>
            <input type="button" name="btnLimpar" value="Limpar"
onclick="limpar();"/>
            <br/>
                  <br/>
            <a href="/banco_1/index.html">Menu Principal</a>
      </form>
    </body>
    <script type="text/javascript" language="javascript">
      function creditar(){
            document.forms[0].elements["method"].value = "creditar";
            document.forms[0].submit();
      }
      function debitar(){
            document.forms[0].elements["method"].value = "debitar";
            document.forms[0].submit();
      }
      function limpar(){
            document.forms[0].elements["numeroConta"].value = "";
```

```
            document.forms[0].elements["valor"].value = "";
        }
    </script>
</html>
```

Quadro 4.4 – Conteúdo do arquivo *creditarDebitarConta.html* que representa formulário de creditar e debitar conta.

A figura 4.8, mostra a interpretação do arquivo *creditarDebitarConta.html* pelo navegador.

Figura 4.8 – Formulário para creditar/debitar de contas.

O formulário para transferir valores entre conta é fisicamente chamado de *transferirConta.html* e possui o conteúdo mostrado no quadro 4.5.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
```

```
        <title>Banco  -  Servlet/JDBC</title>
    </head>
    <body>
            ETI - Especialização em Tecnologia da Informação<br/>
      UFC - Universidade Federal do Ceará

      <h1>Transferência Conta</h1>
      <hr/>
      <form name="conta"
action="/banco_1/servlet/TransferirContaServlet" method="post">
            <table>
                  <tr>
                        <td>Conta Origem</td>
                        <td>
                              <input type="text"
name="numeroContaOrigem"/>
                        </td>
                  </tr>
                  <tr>
                        <td>Conta Destino</td>
                        <td>
                              <input type="text"
name="numeroContaDestino"/>
                        </td>
                  </tr>
                  <tr>
                        <td>Valor</td>
                        <td>
                              <input type="text" name="valor"/>
                        </td>
                  </tr>
            </table>
            <input type="button" name="btnTransferir"
value="Transferir" onclick="submit();"/>
            <input type="button" name="btnLimpar" value="Limpar"
onclick="limpar();"/>
            <br/>
                  <br/>
            <a href="/banco_1/index.html">Menu Principal</a>
      </form>
    </body>
    <script type="text/javascript" language="javascript">
        function limpar(){
            document.forms[0].elements["numeroContaOrigem"].value = "";
            document.forms[0].elements["numeroContaDestino"].value =
"";
            document.forms[0].elements["valor"].value = "";
      }
    </script>

</html>
```

Quadro 4.5 – Conteúdo do arquivo *transferirConta.html* que representa formulário de transferir valores entre contas.

A figura 4.9, mostra a interpretação do arquivo *transferirConta.html* pelo navegador.

Figura 4.9 – Formulário para transferir valores entre contas.

O formulário para render juros conta é fisicamente chamado de *renderJurosConta.html* e possui o conteúdo mostrado no quadro 4.6.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
        <title>Banco  -  Servlet/JDBC</title>
    </head>
    <body>
            ETI - Especialização em Tecnologia da Informação<br/>
      UFC - Universidade Federal do Ceará

      <h1>Render Juros Conta</h1>
      <hr/>
      <form name="conta"
action="/banco_1/servlet/RenderJurosContaServlet" method="post">
```

```
            <table>
                  <tr>
                        <td>Conta</td>
                        <td>
                              <input type="text" name="numeroConta"/>
                        </td>
                  </tr>
                  <tr>
                        <td>Juros</td>
                        <td>
                              <input type="text" name="valorJuros"/>
                        </td>
                  </tr>
            </table>
            <input type="button" name="btnRenderJuros" value="Render
Juros" onclick="submit();"/>
            <input type="button" name="btnLimpar" value="Limpar"
onclick="limpar();"/>
            <br/>
                  <br/>
            <a href="/banco_1/index.html">Menu Principal</a>
      </form>
    </body>
    <script type="text/javascript" language="javascript">
        function limpar(){
            document.forms[0].elements["numeroConta"].value = "";
            document.forms[0].elements["valorJuros"].value = "";
      }
    </script>
</html>
```

Quadro 4.6 – Conteúdo do arquivo *renderJurosConta.html* que representa formulário de reder juros contas.

A figura 4.10, mostra a interpretação do arquivo *renderJurosConta.html* pelo navegador.

Figura 4.10 – Formulário para render juros de conta.

O formulário para render juros de conta é fisicamente chamado de *renderBonusConta.html* e possui o conteúdo mostrado no quadro 4.7.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
        <title>Banco  -  Servlet/JDBC</title>
    </head>
    <body>
            ETI - Especialização em Tecnologia da Informação<br/>
      UFC - Universidade Federal do Ceará

      <h1>Render Bônus Conta</h1>
      <hr/>
      <form name="conta"
action="/banco_1/servlet/RenderBonusContaServlet" method="post">
           <table>
```

```
                  <tr>
                        <td>Conta</td>
                        <td>
                              <input type="text" name="numeroConta"/>
                        </td>
                  </tr>
            </table>
            <input type="button" name="btnRenderBonus" value="Render
Bônus" onclick="submit();"/>
            <input type="button" name="btnLimpar" value="Limpar"
onclick="limpar();"/>
            <br/>
                  <br/>
            <a href="/banco_1/index.html">Menu Principal</a>
      </form>
    </body>
    <script type="text/javascript" language="javascript">
        function limpar(){
            document.forms[0].elements["numeroConta"].value = "";
      }
    </script>

</html>
```

Quadro 4.7 – Conteúdo do arquivo *renderBonusConta.html* que representa formulário de reder juros contas.

A figura 4.11, mostra a interpretação do arquivo *renderBonusConta.html* pelo navegador.

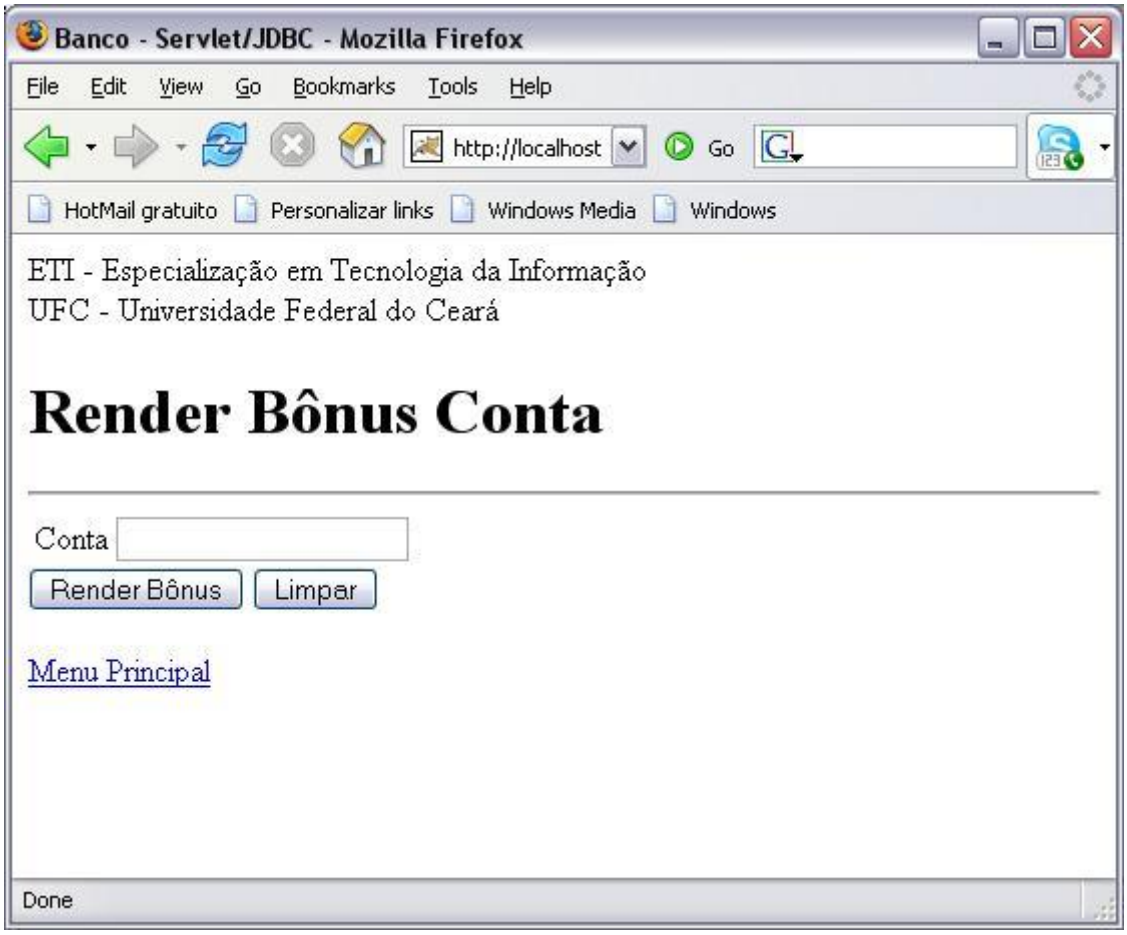

Figura 4.11 – Formulário para render bônus de conta.

O formulário para consultar saldo conta é fisicamente chamado de *consultarSaldoConta.html* e possui o conteúdo mostrado no quadro 4.8.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
        <title>Banco  -  Servlet/JDBC</title>
    </head>
    <body>
            ETI - Especialização em Tecnologia da Informação<br/>
      UFC - Universidade Federal do Ceará

      <h1>Consultar Saldo Conta</h1>
      <hr/>
      <form name="conta"
action="/banco_1/servlet/ConsultarSaldoContaServlet" method="post">
            <table>
                  <tr>
                        <td>Conta</td>
                        <td>
                              <input type="text" name="numeroConta"/>
```

```
                        </td>
                    </tr>
                </table>
                <input type="button" name="btnConsultar" value="Consultar
Saldo" onclick="submit();"/>
                <input type="button" name="btnLimpar" value="Limpar"
onclick="limpar();"/>
                <br/>
                    <br/>
                <a href="/banco_1/index.html">Menu Principal</a>
        </form>
    </body>
    <script type="text/javascript" language="javascript">
        function limpar(){
            document.forms[0].elements["numeroConta"].value = "";
        }
    </script>
</html>
```

Quadro 4.8 – Conteúdo do arquivo *consultarSaldoConta.html* que representa formulário de consultar saldo contas.

A figura 4.12 mostra a interpretação do arquivo *consultarSaldoConta.html* pelo navegador.

Figura 4.12 – Formulário para consultar saldo da conta.

O formulário para consultar saldo conta é fisicamente chamado de *consultarBonusConta.html* e possui o conteúdo mostrado no quadro 4.9.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
        <title>Banco  -  Servlet/JDBC</title>
    </head>
    <body>
            ETI - Especialização em Tecnologia da Informação<br/>
      UFC - Universidade Federal do Ceará

      <h1>Consultar Bônus Conta</h1>
      <hr/>
      <form name="conta"
action="/banco_1/servlet/ConsultarBonusContaServlet" method="post">
            <table>
                  <tr>
                        <td>Conta</td>
                        <td>
                              <input type="text" name="numeroConta"/>
                        </td>
                  </tr>
            </table>
            <input type="button" name="btnConsultar" value="Consultar
Bônus" onclick="submit();"/>
            <input type="button" name="btnLimpar" value="Limpar"
onclick="limpar();"/>
            <br/>
                  <br/>
            <a href="/banco_1/index.html">Menu Principal</a>
      </form>
    </body>
    <script type="text/javascript" language="javascript">
        function limpar(){
            document.forms[0].elements["numeroConta"].value = "";
      }
    </script>
</html>
```

Quadro 4.9 – Conteúdo do arquivo *consultarBonusConta.html* que representa formulário de consultar bônus contas.

A figura 4.13, mostra a interpretação do arquivo *consultarBonusConta.html* pelo navegador.

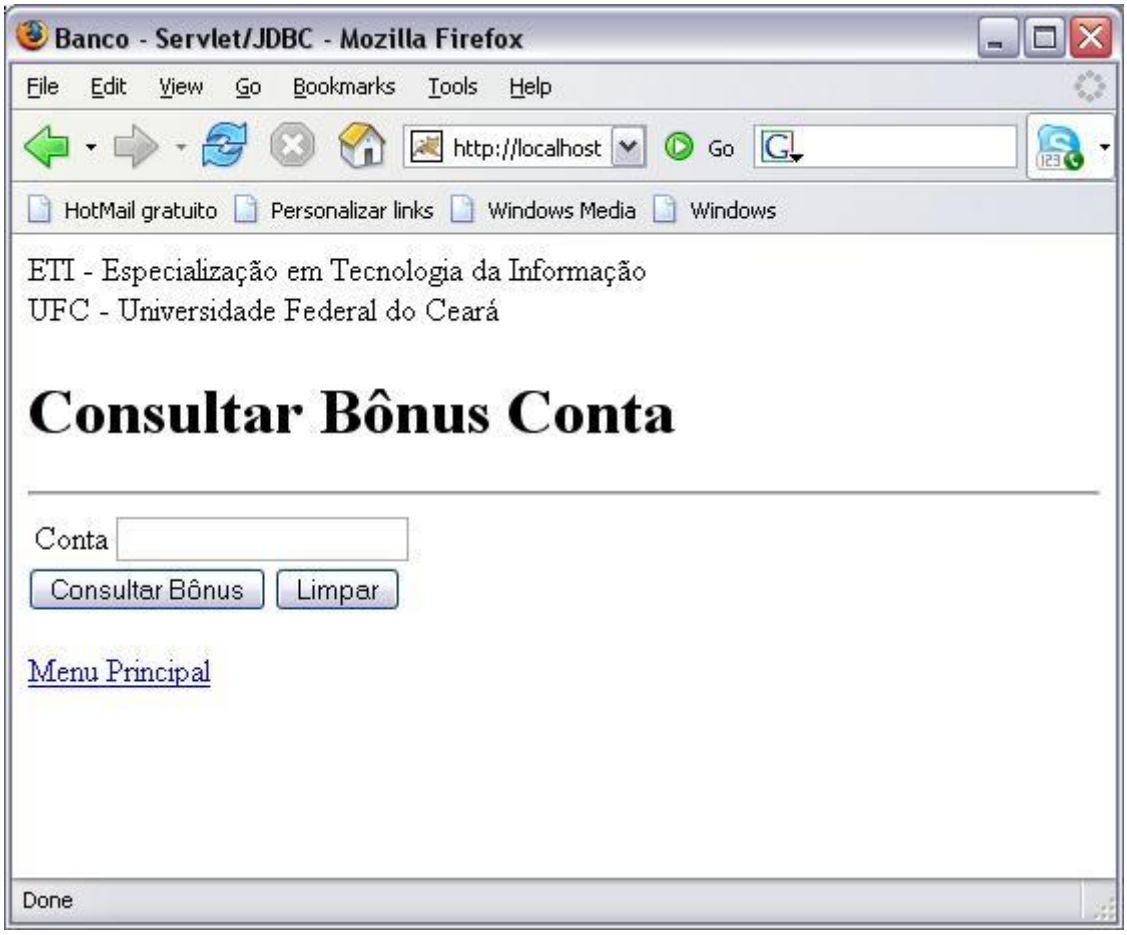

Figura 4.13 – Formulário para consultar bônus da conta.

A figura 4.14 mostra a interpretação do arquivo de resultado pelo navegador.

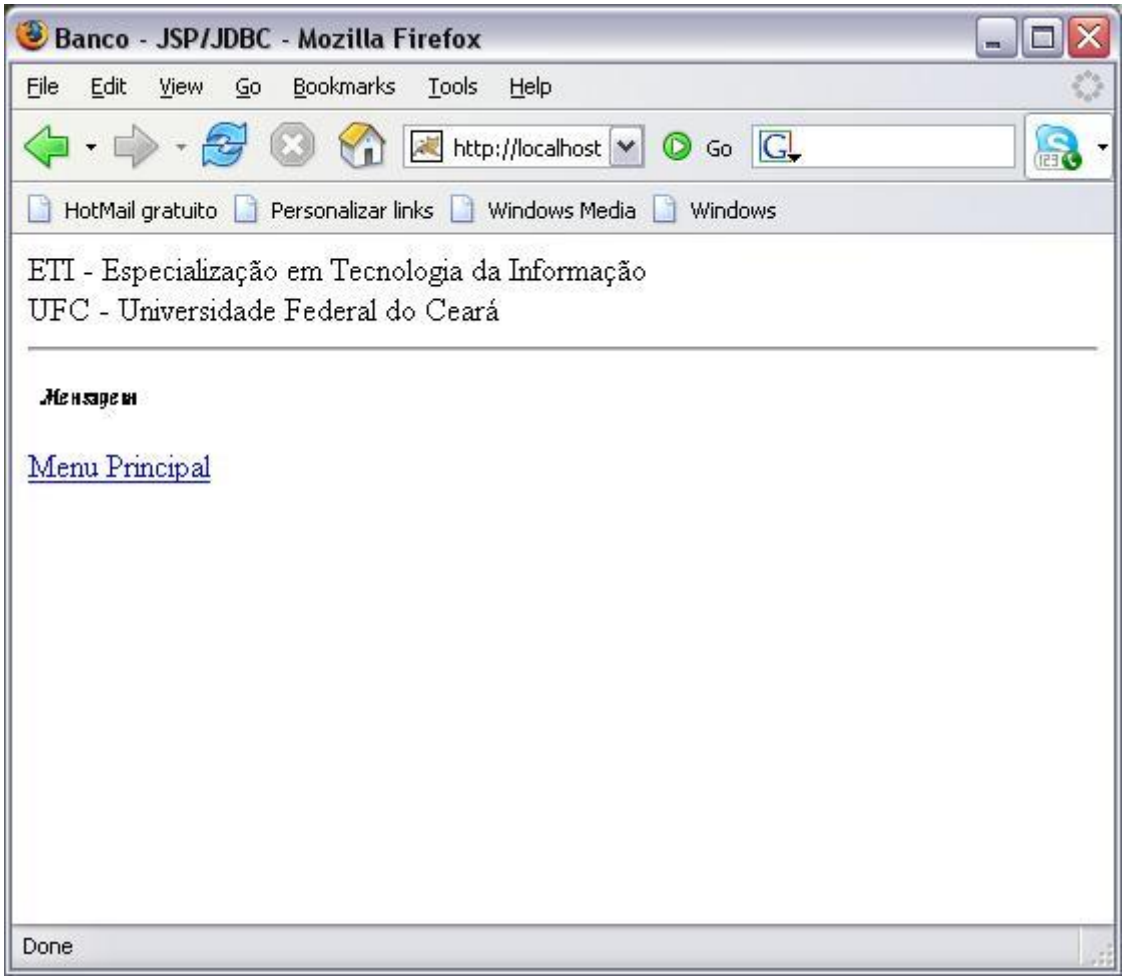

Figura 4.14 – Tela de resultado das operações.

No diretório *WEB-INF* é onde está guardado tudo que se refere a *Java*. Nele deverá conter, obrigatoriamente, o arquivo de nome físico *web.xml*, que é responsável basicamente pelos mapeamentos das classes *servlet*, *tags* customizadas e página de inicialização.

A finalidade e o formato do descritor de desenvolvimento da aplicação *web* são tratados pelas *Sun Servlet Specification*, e é basicamente usado para informar ao *contêiner* do *Servlet* sobre como configurar os *Servlets* e outros objetos de alto nível de que uma aplicação precisa[11].

O quadro 4.10 mostra o conteúdo do arquivo *web.xml*, e pode-se observar que nele são mapeadas as classes *servlets* da nossa aplicação.

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
      version="2.4">

      <servlet>
      <servlet-name>CadasdrarContaServlet</servlet-name>
      <servlet-
class>br.com.ufc.banco.control.CadastrarContaServlet</servlet-class>
      </servlet>
      <servlet-mapping>
      <servlet-name>CadasdrarContaServlet</servlet-name>
      <url-pattern>/servlet/CadastrarContaServlet</url-pattern>
      </servlet-mapping>
      <servlet>
      <servlet-name>ExcluirContaServlet</servlet-name>
      <servlet-
class>br.com.ufc.banco.control.ExcluirContaServlet</servlet-class>
      </servlet>
      <servlet-mapping>
      <servlet-name>ExcluirContaServlet</servlet-name>
      <url-pattern>/servlet/ExcluirContaServlet</url-pattern>
      </servlet-mapping>
      <servlet>
      <servlet-name>CreditarDebitarContaServlet</servlet-name>
      <servlet-
class>br.com.ufc.banco.control.CreditarDebitarContaServlet</servlet-
class>
      </servlet>
      <servlet-mapping>
      <servlet-name>CreditarDebitarContaServlet</servlet-name>
      <url-pattern>/servlet/CreditarDebitarContaServlet</url-pattern>
      </servlet-mapping>
      <servlet>
      <servlet-name>TransferirContaServlet</servlet-name>
      <servlet-
class>br.com.ufc.banco.control.TransferirContaServlet</servlet-class>
      </servlet>
      <servlet-mapping>
      <servlet-name>TransferirContaServlet</servlet-name>
      <url-pattern>/servlet/TransferirContaServlet</url-pattern>
      </servlet-mapping>
      <servlet>
      <servlet-name>RenderJurosContaServlet</servlet-name>
      <servlet-
class>br.com.ufc.banco.control.RenderJurosContaServlet</servlet-class>
      </servlet>
      <servlet-mapping>
      <servlet-name>RenderJurosContaServlet</servlet-name>
      <url-pattern>/servlet/RenderJurosContaServlet</url-pattern>
      </servlet-mapping>
      <servlet>
      <servlet-name>RenderBonusContaServlet</servlet-name>
      <servlet-
class>br.com.ufc.banco.control.RenderBonusContaServlet</servlet-class>
      </servlet>
      <servlet-mapping>
      <servlet-name>RenderBonusContaServlet</servlet-name>
      <url-pattern>/servlet/RenderBonusContaServlet</url-pattern>
      </servlet-mapping>
```

```
      <servlet>
      <servlet-name>ConsultarSaldoContaServlet</servlet-name>
      <servlet-
class>br.com.ufc.banco.control.ConsultarSaldoContaServlet</servlet-
class>
      </servlet>
      <servlet-mapping>
      <servlet-name>ConsultarSaldoContaServlet</servlet-name>
      <url-pattern>/servlet/ConsultarSaldoContaServlet</url-pattern>
      </servlet-mapping>
      <servlet>
      <servlet-name>ConsultarBonusContaServlet</servlet-name>
      <servlet-
class>br.com.ufc.banco.control.ConsultarBonusContaServlet</servlet-
class>
      </servlet>
      <servlet-mapping>
      <servlet-name>ConsultarBonusContaServlet</servlet-name>
      <url-pattern>/servlet/ConsultarBonusContaServlet</url-pattern>
      </servlet-mapping>

      <session-config>
            <session-timeout>30</session-timeout>
      </session-config>
      <welcome-file-list>
            <welcome-file>index.html</welcome-file>
      </welcome-file-list>
</web-app>
```

Quadro 4.10 – Conteúdo do arquivo *web.xm*.

Ainda no diretório *WEB-INF*, existe dois sub-diretórios: *lib* e *classes*. O *lib* guarda as bibliotecas de classes utilizadas no projeto. Já o diretório *classes*, serve para guardar as classes que foram desenvolvidas conforme seus pacotes. Os pacotes, *packages*, que também são estruturas de pastas, foram organizados de acordo com a especialidade das classes que são: modelo, negócio, controle, persistência e utilidades comuns. A figura 4.15, mostra a estrutura que as *packages* estão organizadas.

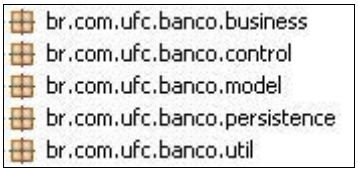

Figura 4.15 – Estrutura das *packages*.

Um motivo para se aninhar *packages* é garantir a exclusividade dos nomes de cada *package*. Caso fossem fornecidas mais classes utilitárias junto com os nossas *packages*, poderíamos colocá-las dentro de um pacote chamado *br.com.ufc.banco.external*, o que é melhor que simplesmente dar um nome a *package* como *external*; alguém mais poderia ter a mesma idéia, causando um conflito. Pode-se

ter quantos níveis de aninhamento se queira. Realmente, para garantir com certeza um nome único de *package*, a própria *Sun* recomenda que se use o próprio nome de domínio da *Internet* da empresa(o qual, presumivelmente, é único) escrito em ordem invertida como prefixo de um pacote[20].

A *package* de modelo, composta pela estrutura *br.com.ufc.banco.model*, guarda apenas classes referentes a modelo. As classes nela presente são: *Conta*, *ContaBonus* e *ContaPoupanca*.

A classe *Conta* representa o tipo de conta simples, sua implementação é mostrada no quadro 4.11.

```
package br.com.ufc.banco.model;

public class Conta {

	private Integer numero;

	private Double saldo;

	public Conta() {

	}

	public Conta(Integer numero) {
		this.numero = numero;
		saldo = 0d;
	}

	public void credito(Double valor) {
		saldo = saldo + valor;
	}

	public void debito(Double valor) {
		saldo = saldo - valor;
	}

	public Integer getNumero() {
		return this.numero;
	}

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

```
	public double getSaldo() {
		return this.saldo;
	}

	public void setSaldo(Double saldo) {
		this.saldo = saldo;
	}
}
```

Quadro 4.11 – Implementação da classe conta do tipo simples.

A classe *ContaPoupanca* representa o tipo de conta poupança, sua implementação é mostrada no quadro 4.12.

```
package br.com.ufc.banco.model;

public class ContaPoupanca extends Conta {

	public ContaPoupanca() {

	}

	public ContaPoupanca(Integer numero) {
		super(numero);
	}

	public void rendeJuros(Double taxa) {
		super.credito(super.getSaldo() * taxa);
	}
}
```

Quadro 4.12 – Implementação da classe conta do tipo poupança.

A classe *ContaBonus* representa o tipo de conta bônus, sua implementação é mostrada no quadro 4.13.

```
package br.com.ufc.banco.model;

public class ContaBonus extends Conta {

	private Double bonus;

	public ContaBonus() {

	}

	public ContaBonus(Integer numero) {
		super(numero);
		this.bonus = 0d;
```

```
	}

	public void credito(Double valor) {
		this.bonus = this.bonus + (valor * 0.01);
		super.credito(valor);
	}

	public void rendeBonus() {
		super.credito(this.bonus);
		this.bonus = 0d;
	}

	public double getBonus() {
		return this.bonus;
	}

	public void setBonus(Double bonus) {
		this.bonus = bonus;
	}
}
```

Quadro 4.13 – Implementação da classe conta do tipo bônus.

A *package* de negócio, composta pela estrutura *br.com.ufc.banco.business*, guarda apenas classes referentes a negócio e tratamento de exceção. As classes nela presentes são: *InterBancoBusiness, BancoBusiness, ContaExistenteException, ContaInexistenteException, ContaPoupancaException, ContaBonusException e LimiteContaException*.

A interface *InterBancoBusiness* representa as assinaturas das operações que podem ser realizadas em uma conta, sua implementação é mostrada no quadro 4.14.

```
package br.com.ufc.banco.business;

public interface InterBancoBusiness {

	public void cadastrarConta(Integer numeroConta, String tipoConta)
			throws ContaExistenteException;

	public void excluirConta(Integer numeroConta)
			throws ContaInexistenteException;

	public void efetuarCredito(Integer numeroConta, Double valorCredito)
			throws ContaInexistenteException;

	public void efetuarDebito(Integer numConta, Double valorDebito)
```

```
                        throws ContaInexistenteException;

        public void renderBonus(Integer numeroConta)
                        throws ContaInexistenteException, ContaBonusException;

        public void renderJuros(Integer numeroConta, Double valorJuros)
                        throws ContaInexistenteException, ContaPoupancaException;

        public void efetuarTransferencia(Integer numeroContaOrigem,
                        Integer numeroContaDestino, Double valorTransferencia)
                        throws ContaInexistenteException, LimiteContaException;

        public Double consultarBonus(Integer numeroConta)
                        throws ContaInexistenteException, ContaBonusException;

        public Double consultarSaldo(Integer numeroConta)
                        throws ContaInexistenteException;
}
```

Quadro 4.14 – Implementação da interface de negócio do banco.

A classe *BancoBusiness* implementa as funcionalidades da interface *InterBancoBusiness*, sua implementação é mostrada no quadro 4.15.

```
package br.com.ufc.banco.business;

import br.com.ufc.banco.model.Conta;
import br.com.ufc.banco.model.ContaBonus;
import br.com.ufc.banco.model.ContaPoupanca;
import br.com.ufc.banco.persistence.BancoSQL;

public class BancoBusiness implements InterBancoBusiness {

        private BancoSQL bancoSQL;

        public BancoBusiness() {
                bancoSQL = new BancoSQL();
        }

        public void cadastrarConta(Integer numeroConta, String tipoConta)
                        throws ContaExistenteException {

                Conta conta = null;

                if (this.existeConta(numeroConta)) {
                        throw new ContaExistenteException(numeroConta.toString());
                }

                if ("C".equals(tipoConta)) {
                        conta = new Conta(numeroConta);
```

```
		}

		if ("P".equals(tipoConta)) {
			conta = new ContaPoupanca(numeroConta);
		}

		if ("B".equals(tipoConta)) {
			conta = new ContaBonus(numeroConta);
		}

		bancoSQL.inserir(conta);
	}

	public void excluirConta(Integer numeroConta)
			throws ContaInexistenteException {

		Conta conta = procurarConta(numeroConta);
		bancoSQL.excluir(conta);
	}

	public void efetuarCredito(Integer numeroConta, Double valorCredito)
			throws ContaInexistenteException {

		Conta conta = procurarConta(numeroConta);
		conta.credito(valorCredito);
		bancoSQL.atualizar(conta);
	}

	public void efetuarDebito(Integer numeroConta, Double valorDebito)
			throws ContaInexistenteException {

		Conta conta = procurarConta(numeroConta);
		conta.debito(valorDebito);
		bancoSQL.atualizar(conta);
	}

	public void renderBonus(Integer numeroConta)
			throws ContaInexistenteException, ContaBonusException {

		Conta conta = procurarConta(numeroConta);

		if (conta instanceof ContaBonus) {
			((ContaBonus) conta).rendeBonus();
			bancoSQL.atualizar(conta);
		} else {
			throw new ContaBonusException(numeroConta.toString());
		}

	}
```

```
	public void renderJuros(Integer numeroConta, Double valorJuros)
			throws ContaInexistenteException, ContaPoupancaException {

		Conta conta = this.procurarConta(numeroConta);

		if (conta instanceof ContaPoupanca) {
			((ContaPoupanca) conta).rendeJuros(valorJuros);
			bancoSQL.atualizar(conta);
		} else {
			throw new ContaPoupancaException(numeroConta.toString());
		}
	}

	public void efetuarTransferencia(Integer numeroContaOrigem,
			Integer numeroContaDestino, Double valorTransferencia)
			throws ContaInexistenteException, LimiteContaException {

		Conta contaOrigem = procurarConta(numeroContaOrigem);
		Conta contaDestino = procurarConta(numeroContaDestino);

		if (contaOrigem.getSaldo() >= valorTransferencia) {
			contaOrigem.debito(valorTransferencia);
			contaDestino.credito(valorTransferencia);

			bancoSQL.atualizar(contaOrigem);
			bancoSQL.atualizar(contaDestino);
		} else {
			throw new LimiteContaException();
		}

	}

	public Double consultarBonus(Integer numeroConta)
			throws ContaInexistenteException, ContaBonusException {

		Conta conta = this.procurarConta(numeroConta);

		if (conta instanceof ContaBonus) {
			return ((ContaBonus) conta).getBonus();
		} else {
			throw new ContaBonusException(numeroConta.toString());
		}
	}

	public Double consultarSaldo(Integer numeroConta)
			throws ContaInexistenteException {

		Conta conta;
		conta = this.procurarConta(numeroConta);
		return conta.getSaldo();
```

```
	}

	private Conta procurarConta(Integer numeroConta)
			throws ContaInexistenteException {

		Conta conta = (Conta) bancoSQL.procurar(new Conta(numeroConta));

		if (null == conta) {
			throw new ContaInexistenteException(numeroConta.toString());
		}
		return conta;
	}

	private Boolean existeConta(Integer numeroConta) {

		Conta c = bancoSQL.procurar(new Conta(numeroConta));
		if (c == null) {
			return new Boolean(Boolean.FALSE);
		} else {
			return new Boolean(Boolean.TRUE);
		}
	}
}
```

Quadro 4.15 – Implementação da classe de negócio do banco.

Na execução das operações, podem ocorrer erros específicos em tempo de execução, que deverão ser tratados na chamada dessas operações. Esses erros são modelados a partir de um erro geral, conforme mostrado na figura 4.16, e são representados pelas classes: *ContaExistenteException, ContaInexistenteException, ContaPoupancaException, ContaBonusException e LimiteContaException*.

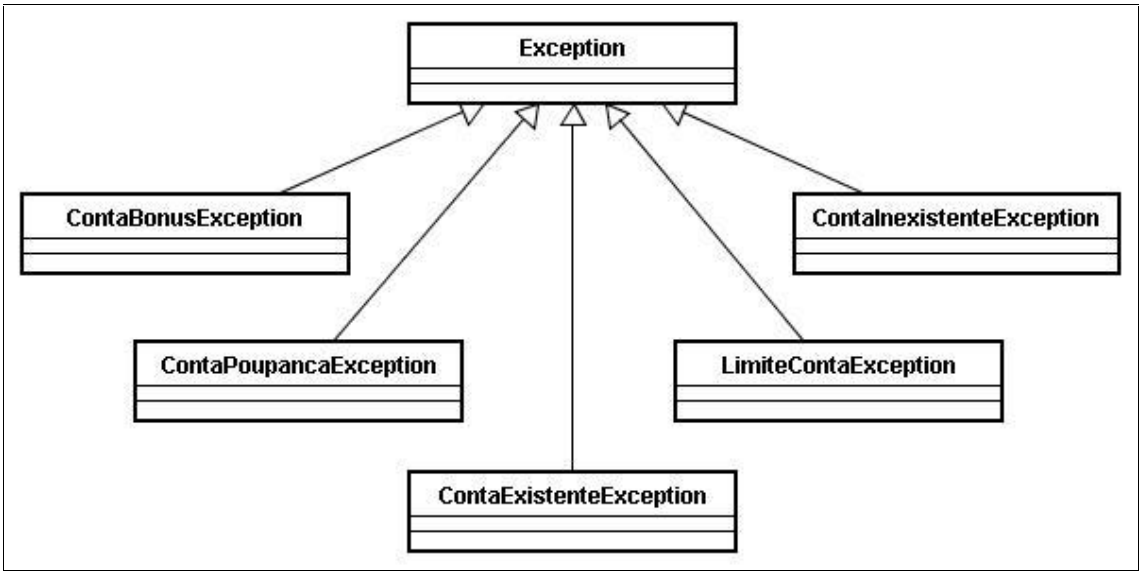

Figura 4.16 – Modelagem dos erros específicos que são possíveis em uma chamada de operação.

A utilização de técnicas de exceção permite que usuários não percam todo o seu trabalho durante um sessão de um programa por uma falha do sistema ou alguma circunstância externa, devendo basicamente[20]:

- Notificar o usuário de um erro;
- Salvar todo trabalho;
- Permitir sair do programa de forma adequada.

A classe *ContaExistenteException* representa uma exceção que ocorrerá quando tentar criar uma conta já existente, sua implementação é mostrada no quadro 4.16.

```
package br.com.ufc.banco.business;

public class ContaExistenteException extends Exception {

	public ContaExistenteException(String numeroConta) {
		super("Conta \"" + numeroConta + "\" já cadastrada!");
	}
}
```

Quadro 4.16 – Implementação da classe de exceção para conta já cadastrada.

A classe *ContaInexistenteException* representa uma exceção que ocorrerá quando tentar utilizar uma conta não criada, sua implementação é mostrada no quadro 4.17.

```
package br.com.ufc.banco.business;

public class ContaInexistenteException extends Exception {

	public ContaInexistenteException(String numeroConta) {
		super("Conta \"" + numeroConta + "\" Inexistente!");
	}
}
```

Quadro 4.17 – Implementação da classe de exceção para conta não cadastrada.

A classe *ContaPoupancaException* representa uma exceção que ocorrerá quando tentar utilizar alguma operação de uma conta que não for do tipo poupança. Sua implementação é mostrada no quadro 4.18.

```
package br.com.ufc.banco.business;

public class ContaPoupancaException extends Exception {

	public ContaPoupancaException(String numeroConta) {
		super("Conta \"" + numeroConta + "\" não é tipo poupança!");
	}
}
```

Quadro 4.18 – Implementação da classe de exceção para operações de conta que não for do tipo poupança.

A classe *ContaBonusException* representa uma exceção que ocorrerá quando tentar utilizar alguma operação de uma conta que não for do tipo bônus. Sua implementação é mostrada no quadro 4.19.

```
package br.com.ufc.banco.business;

public class ContaBonusException extends Exception {

	public ContaBonusException(String numeroConta) {
		super("Conta \"" + numeroConta + "\" não é tipo bônus!");
	}
}
```

Quadro 4.19 – Implementação da classe de exceção para operações de conta que não for do tipo bônus.

A classe *LimiteContaException* representa uma exceção que ocorrerá quando tentar utilizar alguma operação em que o saldo não for suficiente. Sua implementação é mostrada no quadro 4.20.

```
package br.com.ufc.banco.business;

public class LimiteContaException extends Exception {

	public LimiteContaException() {
		super("Saldo insuficiente!");
	}
}
```

Quadro 4.20 – Implementação da classe de exceção para operações de saldo insuficiente.

A *package* de controle, composta pela estrutura *br.com.ufc.banco.control*, guarda apenas classes referentes ao controle das operações. As classes nela presente são: *CadastrarContaServlet, ExcluirContaServlet, CreditarDebitarContaServlet, TransferirContaServlet, RenderJurosContaServlet, RenderBonusContaServlet, ConsultarSaldoContaServlet* e *ConsultarBonusContaServlet*.

Os formulários submetem seus dados diretamente para os servlets, por sua vez, os mesmo recuperam os valores dos campos e executam a classe de negócio conforme a ação desejada. Toda classe *Servlet* tem que estender de *HttpServlet* e implementar os métodos *doGet* e *doPost*. Esses são basicamente os métodos responsáveis pela recuperação dos dados do formulário, diferindo apenas como os dados são enviados para o próprio *Servlet.* O método *doGet* os dados são passados diretamente pela *URL*, já o método *doPost*, os dados são passados por arquivo, *InputStream*, tendo apenas que abrir e ler o mesmo.

O *servlet CadastrarContaServlet*, representa o controle para cadastrar uma conta e gerar o resultado da ação, sua implementação é mostrada no quadro 4.21.

```
package br.com.ufc.banco.control;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import br.com.ufc.banco.business.BancoBusiness;
import br.com.ufc.banco.business.ContaExistenteException;
import br.com.ufc.banco.business.InterBancoBusiness;

public class CadastrarContaServlet extends HttpServlet {

        protected void processRequest(HttpServletRequest request,
                        HttpServletResponse response) throws ServletException,
IOException {

                PrintWriter out = response.getWriter();

                String numero = request.getParameter("numeroConta");
                String tipo = request.getParameter("tipoConta");
```

```
                out
                        .println("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD
HTML 4.01 Transitional//EN\"");
                out.println("\"http://www.w3.org/TR/html4/loose.dtd\">");
                out.println("<html>");
                out.println("    <head>");
                out.println("            <meta http-equiv=\"Content-Type\"
content=\"text/html; charset=UTF-8\">");
                out.println("            <title>Banco  -  JSP/JDBC</title>");
                out.println("    </head>");
                out.println("    <body>");
                out.println("            ETI - Especialização em Tecnologia da
Informação<br/>");
                out.println("            UFC - Universidade Federal do Ceará");
                out.println("            <hr/>");
                try {
                        InterBancoBusiness bancoBusiness = new BancoBusiness();
                        bancoBusiness.cadastrarConta(new Integer(numero), tipo);

                        out.println("Conta cadastrada com sucesso!");
                } catch (ContaExistenteException e) {
                        out.println(e.getMessage());
                }
                out.println("            <br/>");
                out.println("            <br/>");
                out.println("            <a href=\"/banco_1/index.html\">Menu
Principal</a>");
                out.println("    </body>");
                out.println("</html>");
        }

        protected void doGet(HttpServletRequest request,
                        HttpServletResponse response) throws ServletException,
IOException {
                processRequest(request, response);
        }

        protected void doPost(HttpServletRequest request,
                        HttpServletResponse response) throws ServletException,
IOException {
                processRequest(request, response);
        }
}
```

Quadro 4.21 – Implementação do *servlet* de controle para cadastrar conta.

O *servlet ExcluirContaServlet* representa o controle para excluir uma conta e gerar o resultado da ação, sua implementação é mostrada no quadro 4.22.

```
package br.com.ufc.banco.control;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import br.com.ufc.banco.business.BancoBusiness;
import br.com.ufc.banco.business.ContaInexistenteException;
import br.com.ufc.banco.business.InterBancoBusiness;

public class ExcluirContaServlet extends HttpServlet {

	protected void processRequest(HttpServletRequest request,
			HttpServletResponse response) throws ServletException,
IOException {

		PrintWriter out = response.getWriter();

		String numero = request.getParameter("numeroConta");

		out
				.println("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD
HTML 4.01 Transitional//EN\"");
		out.println("\"http://www.w3.org/TR/html4/loose.dtd\">");
		out.println("<html>");
		out.println("    <head>");
		out
				.println("                <meta http-equiv=\"Content-Type\"
content=\"text/html; charset=UTF-8\">");
		out.println("            <title>Banco  -  JSP/JDBC</title>");
		out.println("    </head>");
		out.println("    <body>");
		out.println("            ETI - Especialização em Tecnologia da
Informação<br/>");
		out.println("            UFC - Universidade Federal do Ceará");
		out.println("            <hr/>");
		try {
			InterBancoBusiness bancoBusiness = new BancoBusiness();
			bancoBusiness.excluirConta(new Integer(numero));

			out.println("Conta excluída com sucesso!");
		} catch (ContaInexistenteException e) {
			out.println(e.getMessage());
		}
		out.println("            <br/>");
```

```
                out.println("            <br/>");
                out.println("            <a href=\"/banco_1/index.html\">Menu
Principal</a>");
                out.println("    </body>");
                out.println("</html>");
        }

        protected void doGet(HttpServletRequest request,
                        HttpServletResponse response) throws ServletException,
IOException {
                processRequest(request, response);
        }

        protected void doPost(HttpServletRequest request,
                        HttpServletResponse response) throws ServletException,
IOException {
                processRequest(request, response);
        }
}
```

Quadro 4.22 – Implementação do *servlet* de controle para excluir conta.

O *servlet CreditarDebitarContaServlet*, representa o controle para creditar e debitar uma conta e gerar o resultado da ação, sua implementação é mostrada no quadro 4.23.

```
package br.com.ufc.banco.control;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import br.com.ufc.banco.business.BancoBusiness;
import br.com.ufc.banco.business.ContaInexistenteException;
import br.com.ufc.banco.business.InterBancoBusiness;

public class CreditarDebitarContaServlet extends HttpServlet {

        protected void processRequest(HttpServletRequest request,
                        HttpServletResponse response) throws ServletException,
IOException {

                PrintWriter out = response.getWriter();
```

```
                String numero = request.getParameter("numeroConta");
                String valor = request.getParameter("valor");
                String method = request.getParameter("method");

                out
                                .println("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD
HTML 4.01 Transitional//EN\"");
                out.println("\"http://www.w3.org/TR/html4/loose.dtd\">");
                out.println("<html>");
                out.println("    <head>");
                out
                                .println("                <meta http-equiv=\"Content-Type\"
content=\"text/html; charset=UTF-8\">");
                out.println("            <title>Banco  -  JSP/JDBC</title>");
                out.println("    </head>");
                out.println("    <body>");
                out.println("            ETI - Especialização em Tecnologia da
Informação<br/>");
                out.println("            UFC - Universidade Federal do Ceará");
                out.println("            <hr/>");
                try {
                        InterBancoBusiness bancoBusiness = new BancoBusiness();

                        if ("debitar".equals(method)) {
                                bancoBusiness.efetuarDebito(new Integer(numero), new
Double(
                                                valor));
                                out.println("Débito realizado com sucesso!");
                        } else {
                                bancoBusiness.efetuarCredito(new Integer(numero), new
Double(
                                                valor));
                                out.println("Crédito realizado com sucesso!!");
                        }

                } catch (ContaInexistenteException e) {
                        out.println(e.getMessage());
                }
                out.println("            <br/>");
                out.println("            <br/>");
                out.println("            <a href=\"/banco_1/index.html\">Menu
Principal</a>");
                out.println("    </body>");
                out.println("</html>");
        }

        protected void doGet(HttpServletRequest request,
                        HttpServletResponse response) throws ServletException,
IOException {
                processRequest(request, response);
```

```
	}

	protected void doPost(HttpServletRequest request,
			HttpServletResponse response) throws ServletException,
IOException {
		processRequest(request, response);
	}
}
```

Quadro 4.23 – Implementação do *servlet* de controle para creditar/debitar conta.

O *servlet TransferirContaServlet*, representa o controle para transferir valores entre contas e gerar o resultado da ação, sua implementação é mostrada no quadro 4.24.

```
package br.com.ufc.banco.control;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import br.com.ufc.banco.business.BancoBusiness;
import br.com.ufc.banco.business.ContaInexistenteException;
import br.com.ufc.banco.business.InterBancoBusiness;
import br.com.ufc.banco.business.LimiteContaException;

public class TransferirContaServlet extends HttpServlet {

	protected void processRequest(HttpServletRequest request,
			HttpServletResponse response) throws ServletException,
IOException {

		PrintWriter out = response.getWriter();

		String numeroOrigem = request.getParameter("numeroContaOrigem");
		String numeroDestino = request.getParameter("numeroContaDestino");
		String valor = request.getParameter("valor");

		out
				.println("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD
HTML 4.01 Transitional//EN\"");
		out.println("\"http://www.w3.org/TR/html4/loose.dtd\">");
		out.println("<html>");
		out.println("    <head>");
		out
```

```
                                .println("                <meta http-equiv=\"Content-Type\"
content=\"text/html; charset=UTF-8\">");
                out.println("            <title>Banco  -  JSP/JDBC</title>");
                out.println("    </head>");
                out.println("    <body>");
                out.println("            ETI - Especialização em Tecnologia da
Informação<br/>");
                out.println("            UFC - Universidade Federal do Ceará");
                out.println("            <hr/>");
                try {
                        InterBancoBusiness bancoBusiness = new BancoBusiness();
                        bancoBusiness.efetuarTransferencia(new
Integer(numeroOrigem), new Integer(
                                        numeroDestino), new Double(valor));

                        out.println("Transferência realizada com sucesso!");
                } catch (ContaInexistenteException e) {
                        out.println(e.getMessage());
                } catch (LimiteContaException e) {
                        out.println(e.getMessage());
                }
                out.println("            <br/>");
                out.println("            <br/>");
                out.println("            <a href=\"/banco_1/index.html\">Menu
Principal</a>");
                out.println("    </body>");
                out.println("</html>");
        }

        protected void doGet(HttpServletRequest request,
                        HttpServletResponse response) throws ServletException,
IOException {
                processRequest(request, response);
        }

        protected void doPost(HttpServletRequest request,
                        HttpServletResponse response) throws ServletException,
IOException {
                processRequest(request, response);
        }
}
```

Quadro 4.24 – Implementação do *servlet* de controle para transferir valores entre contas.

O *servlet RenderJurosContaServlet* representa o controle para render juros de uma conta e gerar o resultado da ação, sua implementação é mostrada no quadro 4.25.

```
package br.com.ufc.banco.control;
```

```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import br.com.ufc.banco.business.BancoBusiness;
import br.com.ufc.banco.business.ContaInexistenteException;
import br.com.ufc.banco.business.ContaPoupancaException;
import br.com.ufc.banco.business.InterBancoBusiness;

public class RenderJurosContaServlet extends HttpServlet {

	protected void processRequest(HttpServletRequest request,
			HttpServletResponse response) throws ServletException,
IOException {

		PrintWriter out = response.getWriter();

		String numero = request.getParameter("numeroConta");
		String valor = request.getParameter("valorJuros");

		out
				.println("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD
HTML 4.01 Transitional//EN\"");
		out.println("\"http://www.w3.org/TR/html4/loose.dtd\">");
		out.println("<html>");
		out.println("    <head>");
		out
				.println("                <meta http-equiv=\"Content-Type\"
content=\"text/html; charset=UTF-8\">");
		out.println("            <title>Banco  -  JSP/JDBC</title>");
		out.println("    </head>");
		out.println("    <body>");
		out.println("            ETI - Especialização em Tecnologia da
Informação<br/>");
		out.println("            UFC - Universidade Federal do Ceará");
		out.println("            <hr/>");
		try {
			InterBancoBusiness bancoBusiness = new BancoBusiness();
			bancoBusiness.renderJuros(new Integer(numero), new
Double(valor));

			out.println("Render Juros realizado com sucesso!");
		} catch (ContaInexistenteException e) {
			out.println(e.getMessage());
		} catch (ContaPoupancaException e) {
			out.println(e.getMessage());
```

```
                }
                out.println("            <br/>");
                out.println("            <br/>");
                out.println("            <a href=\"/banco_1/index.html\">Menu
Principal</a>");
                out.println("    </body>");
                out.println("</html>");
        }

        protected void doGet(HttpServletRequest request,
                        HttpServletResponse response) throws ServletException,
IOException {
                processRequest(request, response);
        }

        protected void doPost(HttpServletRequest request,
                        HttpServletResponse response) throws ServletException,
IOException {
                processRequest(request, response);
        }
}
```

Quadro 4.25 – Implementação do *servlet* de controle para render juros conta.

O *servlet RenderBonusContaServlet* representa o controle para render bônus conta e gerar o resultado da ação, sua implementação é mostrada no quadro 4.26.

```
package br.com.ufc.banco.control;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import br.com.ufc.banco.business.BancoBusiness;
import br.com.ufc.banco.business.ContaBonusException;
import br.com.ufc.banco.business.ContaInexistenteException;
import br.com.ufc.banco.business.InterBancoBusiness;

public class RenderBonusContaServlet extends HttpServlet {

        protected void processRequest(HttpServletRequest request,
                        HttpServletResponse response) throws ServletException,
IOException {

                PrintWriter out = response.getWriter();
```

```
            String numero = request.getParameter("numeroConta");

            out
                        .println("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD
HTML 4.01 Transitional//EN\"");
            out.println("\"http://www.w3.org/TR/html4/loose.dtd\">");
            out.println("<html>");
            out.println("    <head>");
            out
                        .println("                <meta http-equiv=\"Content-Type\"
content=\"text/html; charset=UTF-8\">");
            out.println("            <title>Banco  -  JSP/JDBC</title>");
            out.println("    </head>");
            out.println("    <body>");
            out.println("            ETI - Especialização em Tecnologia da
Informação<br/>");
            out.println("            UFC - Universidade Federal do Ceará");
            out.println("            <hr/>");
            try {
                  InterBancoBusiness bancoBusiness = new BancoBusiness();
                  bancoBusiness.renderBonus(new Integer(numero));

                  out.println("Render Bônus realizado com sucesso!");
            } catch (ContaInexistenteException e) {
                  out.println(e.getMessage());
            } catch (ContaBonusException e) {
                  out.println(e.getMessage());
            }
            out.println("            <br/>");
            out.println("            <br/>");
            out.println("            <a href=\"/banco_1/index.html\">Menu
Principal</a>");
            out.println("    </body>");
            out.println("</html>");
      }

      protected void doGet(HttpServletRequest request,
                  HttpServletResponse response) throws ServletException,
IOException {
            processRequest(request, response);
      }

      protected void doPost(HttpServletRequest request,
                  HttpServletResponse response) throws ServletException,
IOException {
            processRequest(request, response);
      }
}
```

Quadro 4.26 – Implementação do *servlet* de controle para render bônus conta.

O *servlet ConsultarSaldoContaServlet* representa o controle para consultar saldo de uma conta e gerar o resultado da ação, sua implementação é mostrada no quadro 4.27.

```
package br.com.ufc.banco.control;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import br.com.ufc.banco.business.BancoBusiness;
import br.com.ufc.banco.business.ContaInexistenteException;
import br.com.ufc.banco.business.InterBancoBusiness;

public class ConsultarSaldoContaServlet extends HttpServlet {

	protected void processRequest(HttpServletRequest request,
			HttpServletResponse response) throws ServletException,
IOException {

		PrintWriter out = response.getWriter();

		String numero = request.getParameter("numeroConta");

		out
				.println("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD
HTML 4.01 Transitional//EN\"");
		out.println("\"http://www.w3.org/TR/html4/loose.dtd\">");
		out.println("<html>");
		out.println("    <head>");
		out
				.println("                <meta http-equiv=\"Content-Type\"
content=\"text/html; charset=UTF-8\">");
		out.println("            <title>Banco  -  JSP/JDBC</title>");
		out.println("    </head>");
		out.println("    <body>");
		out.println("            ETI - Especialização em Tecnologia da
Informação<br/>");
		out.println("            UFC - Universidade Federal do Ceará");
		out.println("            <hr/>");
		try {
			InterBancoBusiness bancoBusiness = new BancoBusiness();
```

```
                        Double saldo = bancoBusiness.consultarSaldo(new
Integer(numero));

                        out.println("Saldo conta " + numero + " R$:" + saldo);
                } catch (ContaInexistenteException e) {
                        out.println(e.getMessage());
                }
                out.println("           <br/>");
                out.println("           <br/>");
                out.println("           <a href=\"/banco_1/index.html\">Menu
Principal</a>");
                out.println("    </body>");
                out.println("</html>");
        }

        protected void doGet(HttpServletRequest request,
                        HttpServletResponse response) throws ServletException,
IOException {
                processRequest(request, response);
        }

        protected void doPost(HttpServletRequest request,
                        HttpServletResponse response) throws ServletException,
IOException {
                processRequest(request, response);
        }
}
```

Quadro 4.27 – Implementação do *servlet* de controle para consultar saldo conta.

O *servlet ConsultarBonusContaServlet* representa o controle para consultar bônus de uma conta e gerar o resultado da ação, sua implementação é mostrada no quadro 4.28.

```
package br.com.ufc.banco.control;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import br.com.ufc.banco.business.BancoBusiness;
import br.com.ufc.banco.business.ContaBonusException;
import br.com.ufc.banco.business.ContaInexistenteException;
import br.com.ufc.banco.business.InterBancoBusiness;
```

```
public class ConsultarBonusContaServlet extends HttpServlet {

	protected void processRequest(HttpServletRequest request,
			HttpServletResponse response) throws ServletException,
IOException {

		PrintWriter out = response.getWriter();

		String numero = request.getParameter("numeroConta");

		out
				.println("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD
HTML 4.01 Transitional//EN\"");
		out.println("\"http://www.w3.org/TR/html4/loose.dtd\">");
		out.println("<html>");
		out.println("    <head>");
		out
				.println("                <meta http-equiv=\"Content-Type\"
content=\"text/html; charset=UTF-8\">");
		out.println("            <title>Banco  -  JSP/JDBC</title>");
		out.println("    </head>");
		out.println("    <body>");
		out.println("            ETI - Especialização em Tecnologia da
Informação<br/>");
		out.println("            UFC - Universidade Federal do Ceará");
		out.println("            <hr/>");
		try {
			InterBancoBusiness bancoBusiness = new BancoBusiness();
			Double saldo = bancoBusiness.consultarBonus(new
Integer(numero));

			out.println("Bônus conta " + numero + " R$:" + saldo);
		} catch (ContaInexistenteException e) {
			out.println(e.getMessage());
		} catch (ContaBonusException e) {
			out.println(e.getMessage());
		}
		out.println("            <br/>");
		out.println("            <br/>");
		out.println("            <a href=\"/banco_1/index.html\">Menu
Principal</a>");
		out.println("    </body>");
		out.println("</html>");
	}

	protected void doGet(HttpServletRequest request,
			HttpServletResponse response) throws ServletException,
IOException {
		processRequest(request, response);
```

```
	}

	protected void doPost(HttpServletRequest request,
			HttpServletResponse response) throws ServletException,
IOException {
		processRequest(request, response);
	}
}
```

Quadro 4.28 – Implementação do *servlet* de controle para consultar bônus conta.

A *package* de persistência, composta pela estrutura *br.com.ufc.banco.persistence*, guarda apenas classes e arquivos de propriedades referentes à persistência de dados. A classe e arquivo nela presente são: *BancoSQL* e *database.properties*.

A classe *BancoSQL*, responsável pelo operações no bando de dados, sua implementação é mostrada no quadro 4.29.

```
package br.com.ufc.banco.persistence;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import br.com.ufc.banco.model.Conta;
import br.com.ufc.banco.model.ContaBonus;
import br.com.ufc.banco.model.ContaPoupanca;
import br.com.ufc.banco.util.UtilProperties;

public class BancoSQL {

	private String url;

	private Connection con;

	private Statement stmt;

	private ResultSet rs;

	public BancoSQL() {

		try {
			Class.forName(UtilProperties.getAcessoMapCode("DRIVER"));
			url = UtilProperties.getAcessoMapCode("URL");
			con = DriverManager.getConnection(url, UtilProperties
```

```
                                        .getAcessoMapCode("USERNAME"),
UtilProperties
                                        .getAcessoMapCode("PASSWORD"));
                        stmt = con.createStatement();
                } catch (ClassNotFoundException e) {
                        System.out.println(e.getMessage());
                } catch (SQLException e) {
                        System.out.println(e.getMessage());
                }
        }

        public void inserir(Conta conta) {

                String tipoConta;

                if (conta instanceof ContaBonus) {
                        tipoConta = "B";
                } else if (conta instanceof ContaPoupanca) {
                        tipoConta = "P";
                } else {
                        tipoConta = "C";
                }

                String clausula;
                if ("B".equals(tipoConta)) {
                        clausula = "insert into conta (numero, tipo,saldo, bonus) values("
                                        + conta.getNumero() + ",\'" + tipoConta + "\',0,0)";
                } else {
                        clausula = "insert into conta (numero, tipo,saldo) values("
                                        + conta.getNumero() + ",\'" + tipoConta + "\',0)";
                }

                try {
                        stmt.executeUpdate(clausula);
                } catch (SQLException e1) {
                        System.out.println(e1.getMessage());
                } catch (Exception e1) {
                        System.out.println(e1.getMessage());
                }
        }

        public void excluir(Conta conta) {

                String clausula = "delete from conta where numero ="
                                + conta.getNumero();

                try {
                        stmt.executeUpdate(clausula);
                } catch (SQLException e1) {
                        System.out.println(e1.getMessage());
```

```
		}
	}

	public void atualizar(Conta conta) {

		String clausula;

		if (conta instanceof ContaBonus) {
			clausula = "update conta set saldo=" + conta.getSaldo()
					+ ", bonus= " + ((ContaBonus) conta).getBonus()
					+ "where numero = " + conta.getNumero();
		} else {
			clausula = "update conta set saldo=" + conta.getSaldo()
					+ "where numero = " + conta.getNumero();
		}

		try {
			stmt.executeUpdate(clausula);
		} catch (SQLException e1) {
			System.out.println(e1.getMessage());
		}
	}

	public Conta procurar(Conta conta) {

		Conta c = null;

		String clausula = "select * from conta where numero = "
				+ conta.getNumero();
		try {
			rs = stmt.executeQuery(clausula);

			if (rs.next()) {

				switch (rs.getString("tipo").charAt(0)) {

				case 'C':
					c = new Conta(new
Integer(rs.getString("numero")));
					c.setSaldo(new Double(rs.getString("saldo")));
					break;

				case 'P':
					c = new ContaPoupanca(new
Integer(rs.getString("numero")));
					c.setSaldo(new Double(rs.getString("saldo")));
					break;

				case 'B':
```

```
                                        c = new ContaBonus(new
Integer(rs.getString("numero")));
                                        c.setSaldo(new Double(rs.getString("saldo")));
                                        ((ContaBonus) c)
                                                        .setBonus(new
Double(rs.getString("bonus")));

                                        break;

                                }
                        }
                } catch (SQLException e1) {
                        System.out.println(e1.getMessage());
                }

                return c;
        }
}
```

Quadro 4.29 – Implementação da classe de operação no banco de dados.

Arquivo *database.properties*, responsável por guardar os parâmetros de acesso do banco de dados, sua implementação é mostrada no quadro 4.30.

```
DRIVER=oracle.jdbc.driver.OracleDriver
URL=jdbc:oracle:thin:@localhost:1521:xe
USERNAME=system
PASSWORD=123456
```

Quadro 4.30 – Conteúdo do arquivo de configuração do banco de dados.

A figura 4.17 mostra a estrutura da tabela.

Object Type **TABLE** Object **CONTA**

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
CONTA	NUMERO	Number	-	-	-	1	-	-	-
	TIPO	Varchar2	1	-	-	-	-	-	-
	SALDO	Float	22	63	-	-	-	-	-
	BONUS	Number	-	-	-	-	✓	-	-

1 - 4

Figura 4.17 – Estrutrua física da tabela conta.

A *package* de utilidade comuns, composta pela estrutura *br.com.ufc.banco.util*, guarda apenas classes que podem ser comuns a outras classes. A classe nela presente é: *UtilProperties*.

A classe *UtilProperties*, responsável pela leitura do arquivo *database.properties*, sua implementação é mostrada no quadro 4.31.

```
package br.com.ufc.banco.util;

import java.io.IOException;
import java.io.InputStream;
import java.util.Properties;

import br.com.ufc.banco.persistence.BancoSQL;

public class UtilProperties {

	private static Properties properties;

	static {
		properties = new Properties();
		try {
			InputStream is =
BancoSQL.class.getResource("database.properties")
					.openStream();
			properties.load(is);
		} catch (IOException e) {
			e.printStackTrace();
		}
	}

	public static String getAcessoMapCode(String key) {

		String code = properties.getProperty(key);
		return code;
	}
}
```

Quadro 4.31 – Implementação da classe de leitura do arquivo *database.properties*.

De fato, o *HTML* não permite realizar um simples cálculo matemático ou criar uma página do nada a partir de uma base de dados. Na verdade, o *HTML*, mesmo sendo muito útil à pequena escala, é bastante limitado na hora de conceber grandes sites ou portais. A partir desta deficiência, fez-se necessário o emprego de outras linguagens acessórios muito mais versáteis e de uma aprendizagem relativamente mais complicada, capazes de responder de maneira inteligente às demandas do navegador e que permitem a automatização de determinadas tarefas tediosas e irremediáveis como podem ser as atualizações e listagens. Desta forma, utilizando da técnica de desenvolvimento de sistemas com *Servlet*, conseguimos superar essas deficiências e criar páginas que podem

ser geradas dinamicamente em tempo de requisição, adequando o conteúdo a cada solicitação em particular.

4.2 Banco com JSP e JDBC

Neste tópico desenvolvemos o mesmo exemplo da aplicação bancária utilizando *JSP* e *JDBC*.

JSP, *JavaServer Pages*, é uma tecnologia utilizada no desenvolvimento de aplicações para *web*. Esta tecnologia permite ao desenvolvedor produzir aplicações que, acessam o banco de dados, manipulam arquivos no formato texto, capturar informações a partir de formulários e sobre o servidor.

Como podemos observar na implementação anterior, conseguimos obter uma aplicação dinâmica utilizando *Servlets* que realizasse todas as operações desejadas, porém, a geração das páginas de respostas na própria classe de controle dificulta e muito uma manutenção, como também, faz com que o programador se preocupe com a área que não o seu domínio, a de apresentação *(design)*.

Para evitar esse problema, desconsideramos a geração das páginas de respostas pelas classes de controle, e desenvolvemos um conjunto de arquivos independentes de respostas, são responsáveis por executar as operações dos objetos de negócio, como também, mostrar resultado finais das mesmas. A figura 4.18 mostra o modelo de interação.

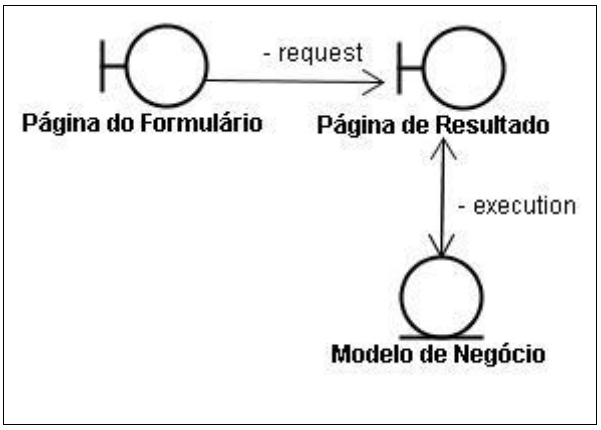

Figura 4.18 – Modelo da interação entre fronteira e fronteira de resposta.

Na figura 4.19, observamos que a estrutura continuará a mesma, diferindo apenas, na exclusão da *package* para as classes de controle.

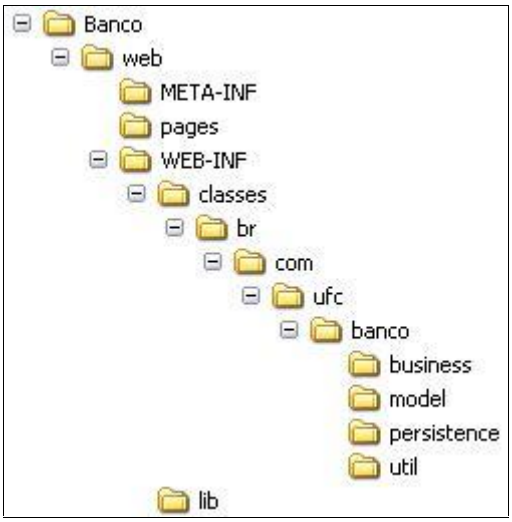

Figura 4.19 – Estrutura do diretório para a aplicação em *JSP*.

Como não temos mais os *Servlets* de controle, precisamos de três alterações. A primeira, alterar os *actions* das *tags* dos formulários para: cadastrar conta, excluir conta, transferência de valores entre contas, reder juros, render bônus, consultar saldo e consultar bônus, onde cada ação específica submeterá os dados para uma outra página em particular. A segunda foi criar, no diretório *pages*, os arquivos *JSP* de resultado para as operações: cadastrar conta, excluir conta, creditar e debitar conta, transferir valores entre contas, render juros, render bônus, consultar saldo e consultar bônus. A terceira remover do arquivo *web.xml* os mapeamentos dos *servlets*.

Alterar os *actions* das *tags forms* do formulário conforme a tabela 4.1, para que o formulário seja submetido diretamente para as páginas de respostas.

Alteração de Action	
Página	Action
cadastrarConta.html	/banco_2/pages/resultadoCadastrarConta.jsp
excluirConta.html	/banco_2/pages/resultadoExcluirConta.jsp
transferirConta.html	/banco_2/pages/resultadoTransferirConta.jsp
renderJurosConta.html	/banco_2/pages/resultadoRenderJurosConta.jsp
renderBonusConta.html	/banco_2/pages/resultadoRenderBonusConta.jsp
consultarSaldoConta.html	/banco_2/pages/resultadoConsultarSaldoConta.jsp
consultarBonusConta.html	/banco_2/pages/resultadoConsultarBonusConta.jsp

Tabela 4.1 – Alteração dos atributos *action* da *tag* dos formulários.

Incorporar conteúdo dinâmico deve no final das contas envolver algum tipo de programação para descrever como aquele conteúdo é gerado. *JSP* é um tanto quanto híbrido, entre os sistemas modelo, porque suporta dois estilos diferentes para adicionar conteúdo dinâmico a páginas da *web*. Assim como *ASP* e *PHP*, *scripts* podem ser embutidos em páginas contendo o código de programação propriamente dito, que no nosso caso, é tipicamente *Java*.

As alterações a seguir mostram *tags*, *"<% ... %>"*, que contêm os códigos *Java* a serem interpretados do lado do servidor.

O arquivo de resultado para cadastrar conta é fisicamente chamado de *resultadoCadastrarConta.jsp* e possui o conteúdo mostrado no quadro 4.32.

```
<%@page import="br.com.ufc.banco.business.*"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
        <title>Banco  -  JSP/JDBC</title>
    </head>
    <body>
            ETI - Especialização em Tecnologia da Informação<br/>
      UFC - Universidade Federal do Ceará

      <hr/>
            <%
                  String numero = request.getParameter("numeroConta");
                  String tipo = request.getParameter("tipoConta");

                  try {
                        InterBancoBusiness bancoBusiness = new
BancoBusiness();
                        bancoBusiness.cadastrarConta(new
Integer(numero), tipo);

                        out.println("Conta cadastrada com sucesso!");
                  } catch (ContaExistenteException e) {
                        out.println(e.getMessage());
                  }
            %>
            <br/>
            <br/>
            <a href="<%=request.getContextPath()%>/index.html">Menu
Principal</a>
    </body>
```

```
</html>
```

Quadro 4.32 – Conteúdo do arquivo *resultadoCadastrarConta.jsp* que representa a página de resultado de cadastrar contas.

O arquivo de resultado para excluir conta é fisicamente chamado de *resultadoExcluirConta.jsp* e possui o conteúdo mostrado no quadro 4.33.

```
<%@page import="br.com.ufc.banco.business.*"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
        <title>Banco  -  JSP/JDBC</title>
    </head>
    <body>
            ETI - Especialização em Tecnologia da Informação<br/>
      UFC - Universidade Federal do Ceará

      <hr/>

            <%
                  String numero = request.getParameter("numeroConta");

                  try {
                        InterBancoBusiness bancoBusiness = new
BancoBusiness();
                        bancoBusiness.excluirConta(new
Integer(numero));

                        out.println("Conta excluída com sucesso!");
                  } catch (ContaInexistenteException e) {
                        out.println(e.getMessage());
                  }
            %>

            <br/>
            <br/>
            <a href="<%=request.getContextPath()%>/index.html">Menu
Principal</a>
    </body>
</html>
```

Quadro 4.33 – Conteúdo do arquivo *resultadoExcluirConta.jsp* que representa a página de resultado de excluir contas.

O arquivo de resultado para creditar e debitar conta é fisicamente chamado de *resultadoCreditarDebitarConta.jsp* e possui o conteúdo mostrado no quadro 4.34.

```
<%@page import="br.com.ufc.banco.business.*"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
```

```
   "http://www.w3.org/TR/html4/loose.dtd">

<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
        <title>Banco  -  JSP/JDBC</title>
    </head>
    <body>
            ETI - Especialização em Tecnologia da Informação<br/>
      UFC - Universidade Federal do Ceará

      <hr/>

      <%
                  String numero = request.getParameter("numeroConta");
                  String valor = request.getParameter("valor");
                  String method = request.getParameter("method");

                  try {
                        InterBancoBusiness bancoBusiness = new
BancoBusiness();

                        if ("debitar".equals(method)) {
                              bancoBusiness.efetuarDebito(new
Integer(numero), new Double(
                                          valor));
                              out.println("Débito realizado com
sucesso!");
                        } else {
                              bancoBusiness.efetuarCredito(new
Integer(numero), new Double(
                                          valor));
                              out.println("Crédito realizado com
sucesso!");
                        }

                  } catch (ContaInexistenteException e) {
                        out.println(e.getMessage());
                  }
      %>

      <br/>
            <br/>
      <a href="<%=request.getContextPath()%>/index.html">Menu
Principal</a>
    </body>
</html>
```

Quadro 4.34 – Conteúdo do arquivo *resultadoCreditarDebitarConta.jsp* que representa a página de resultado de creditar/debitar contas.

O arquivo de resultado para transferência de valor entre contas é fisicamente chamado de *resultadoTransferirConta.jsp* e possui o conteúdo mostrado no quadro 4.35.

```
<%@page import="br.com.ufc.banco.business.*"%>
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
        <title>Banco  -  JSP/JDBC</title>
    </head>
    <body>
            ETI - Especialização em Tecnologia da Informação<br/>
      UFC - Universidade Federal do Ceará

      <hr/>

            <%
                  String numeroOrigem =
request.getParameter("numeroContaOrigem");
                  String numeroDestino =
request.getParameter("numeroContaDestino");
                  String valor = request.getParameter("valor");

                  try {
                        InterBancoBusiness bancoBusiness = new
BancoBusiness();
                        bancoBusiness.efetuarTransferencia(new
Integer(numeroOrigem), new Integer(
                                    numeroDestino), new Double(valor));

                        out.println("Transferência realizada com
sucesso!");
                  } catch (ContaInexistenteException e) {
                        out.println(e.getMessage());
                  } catch (LimiteContaException e) {
                        out.println(e.getMessage());
                  }
            %>

            <br/>
            <br/>
            <a href="<%=request.getContextPath()%>/index.html">Menu
Principal</a>
    </body>
</html>
```

Quadro 4.35 – Conteúdo do arquivo *resultadoTransferirConta.jsp* que representa a página de resultado de transferir contas.

O arquivo de resultado para operação de render juros de uma conta é fisicamente chamado de *resultadoRenderJurosConta.jsp* e possui o conteúdo mostrado no quadro 4.36.

```
<%@page import="br.com.ufc.banco.business.*"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<html>
```

```
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
        <title>Banco  -  JSP/JDBC</title>
    </head>
    <body>
            ETI - Especialização em Tecnologia da Informação<br/>
      UFC - Universidade Federal do Ceará

      <hr/>

            <%
                  String numero = request.getParameter("numeroConta");
                  String valor = request.getParameter("valorJuros");

                  try {
                        InterBancoBusiness bancoBusiness = new
BancoBusiness();
                        bancoBusiness.renderJuros(new Integer(numero),
new Double(valor));

                        out.println("Render Juros realizado com
sucesso!");
                  } catch (ContaInexistenteException e) {
                        out.println(e.getMessage());
                  } catch (ContaPoupancaException e) {
                        out.println(e.getMessage());
                  }
            %>

            <br/>
            <br/>
            <a href="<%=request.getContextPath()%>/index.html">Menu
Principal</a>
    </body>
</html>
```

Quadro 4.36 – Conteúdo do arquivo *resultadoRenderJurosConta.jsp* que representa a página de resultado de render juros contas.

O arquivo de resultado para operação de render bônus de uma conta é fisicamente chamado de *resultadoRenderBonusConta.jsp* e possui o conteúdo mostrado no quadro 4.37.

```
<%@page import="br.com.ufc.banco.business.*"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
        <title>Banco  -  JSP/JDBC</title>
    </head>
    <body>
            ETI - Especialização em Tecnologia da Informação<br/>
      UFC - Universidade Federal do Ceará
```

```
      <hr/>

            <%
                  String numero = request.getParameter("numeroConta");

                  try {
                        InterBancoBusiness bancoBusiness = new
BancoBusiness();
                        bancoBusiness.renderBonus(new Integer(numero));

                        out.println("Render Bônus realizado com
sucesso!");
                  } catch (ContaInexistenteException e) {
                        out.println(e.getMessage());
                  } catch (ContaBonusException e) {
                        out.println(e.getMessage());
                  }
            %>

            <br/>
            <br/>
            <a href="<%=request.getContextPath()%>/index.html">Menu
Principal</a>
    </body>
</html>
```

Quadro 4.37 – Conteúdo do arquivo *resultadoRenderBonusConta.jsp* que representa a página de resultado de render bônus contas.

O arquivo de resultado para consulta de saldo de uma conta é fisicamente chamado de *resultadoConsultarSaldoConta.jsp* e possui o conteúdo mostrado no quadro 4.38.

```
<%@page import="br.com.ufc.banco.business.*"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
        <title>Banco  -  JSP/JDBC</title>
    </head>
    <body>
            ETI - Especialização em Tecnologia da Informação<br/>
      UFC - Universidade Federal do Ceará

      <hr/>

            <%
                  String numero = request.getParameter("numeroConta");

                  try {
                        InterBancoBusiness bancoBusiness = new
BancoBusiness();
```

```
                        Double saldo = bancoBusiness.consultarSaldo(new
Integer(numero));

                        out.println("Saldo conta " + numero + " R$:" +
saldo);
                  } catch (ContaInexistenteException e) {
                        out.println(e.getMessage());
                  }
            %>

            <br/>
            <br/>
            <a href="<%=request.getContextPath()%>/index.html">Menu
Principal</a>
    </body>
</html>
```

Quadro 4.38 – Conteúdo do arquivo *resultadoConsultarSaldoConta.jsp* que representa a página de resultado de consultar saldo contas.

O arquivo de resultado para cansulta de bônus de uma conta é fisicamente chamado de *resultadoConsultarBonusConta.jsp* e possui o conteúdo mostrado no quadro 4.39.

```
<%@page import="br.com.ufc.banco.business.*"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
        <title>Banco  -  JSP/JDBC</title>
    </head>
    <body>
            ETI - Especialização em Tecnologia da Informação<br/>
      UFC - Universidade Federal do Ceará

      <hr/>

            <%
                  String numero = request.getParameter("numeroConta");

                  try {
                        InterBancoBusiness bancoBusiness = new
BancoBusiness();
                        Double saldo = bancoBusiness.consultarBonus(new
Integer(numero));

                        out.println("Bônus conta " + numero + " R$:" +
saldo);
                  } catch (ContaInexistenteException e) {
                        out.println(e.getMessage());
                  } catch (ContaBonusException e) {
                        out.println(e.getMessage());
                  }
```

```
            %>

      <br/>
            <br/>
            <a href="<%=request.getContextPath()%>/index.html">Menu
Principal</a>
    </body>
</html>
```

Quadro 4.39 – Conteúdo do arquivo *resultadoConsultarBonusConta.jsp* que representa a página de resultado de consultar bônus contas.

O arquivo *web.xml* conterá apenas o mapeamento mostrado no quadro 4.40.

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
      version="2.4">

      <session-config>
            <session-timeout>30</session-timeout>
      </session-config>
      <welcome-file-list>
            <welcome-file>index.html</welcome-file>
      </welcome-file-list>
</web-app>
```

Quadro 4.40 – Conteúdo do arquivo *web.xml*.

Com o desenvolvimento desta técnica, conseguimos, em arquivos independentes, incorporar na própria página de resultado, *scriptlets* que executam comandos *Java*. Agora, o programador deverá apenas criar os *scriptlets* necessários para realização das operações, e passá-los diretamente para as próprias páginas, reduzindo a complexidade da geração de *layout* de página dentro das classes controladoras, *Servlets*.

4.3 Banco com JSP, Servlet, MVC e JDBC

Neste tópico desenvolvemos o mesmo exemplo da aplicação bancária utilizando *JSP*, *Servlet* e *JDBC*, mas seguindo o padrão de projeto *MVC*.

MVC é um padrão de arquitetura de aplicações que visa separar a lógica da aplicação *(Model)*, da interface do usuário *(View)* e do fluxo da aplicação *(Controller)*,

permitindo que a mesma lógica de negócios possa ser acessada e visualizada por várias interfaces.

Cada padrão descreve um problema no nosso ambiente e o cerne da solução, de tal forma que você passa a usar essa solução mais de um milhão de vezes, sem nunca fazê-lo da mesma maneira[22].

Como podemos observar na implementação anterior, conseguimos resumir a complexidade da geração das páginas de respostas dentro dos *Servlets* de controle, porém, passamos para o profissional de *design* a complexidade de implementações de negócio, ou seja, o mesmo não tem familiaridade com as estruturas de programa que passaram a fazer parte das *tags* de apresentação *HTML*.

Para resolver ambos os problemas, desenvolvemos a aplicação dividida em três camadas: Uma camada de visão que procura reduzir ao máximo *scripts Java* dentro do corpo da página, uma camada de modelo, que se propõe unificar as regras de negócio, e outra de controle que rebece as solicitações da camada de visão, executa a regras de negócio e redireciona para uma página de resultado. A figura 4.20, mostra o modelo de interação entre as camadas.

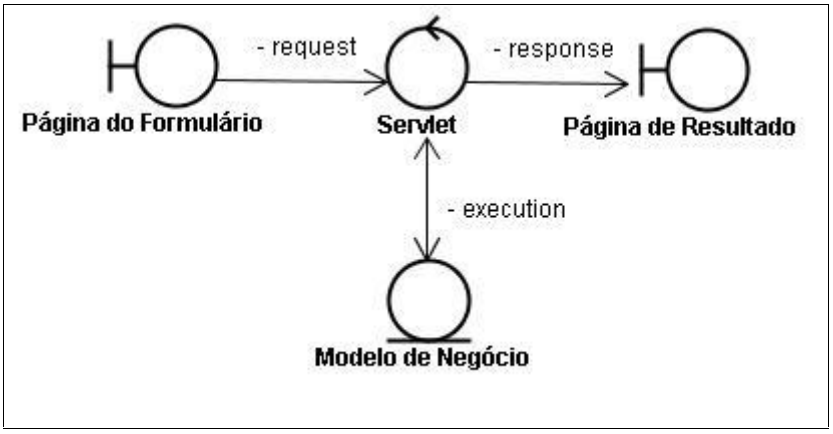

Figura 4.20 – Modelo da interação no modelo *MVC*.

Utilizaremos a estrutura inicial, figura 4.4, onde estaremos novamente utilizando a *package* controle.

Várias alterações foram feitas para se adequar ao padrão *MVC*. No diretório *pages*, fazemos dois tipos de alterações: A primeira foi relacionda as páginas que possui

formulário de entrada de dados, onde todos as *tags* de formulário, os *actions* estão apontando para um único *servlet*. O quadro 4.41 mostra o exemplo correspondente à alteração.

```
<form name="conta" action="/banco_3/servlet/FrontControllerServlet"
method="post">
```

Quadro 4.41 – Exemplo da alteração do atributo de ação.

E para cada formulário de entrada a dados, tem um componente *HTML* do tipo *hidden*, logo abaixo da abertura da *tag* de formulário que indica qual o método de ação a ser executado. O quadro 4.42 mostra o exemplo da inclusão da tag *hidden*.

```
<form name="conta" action="/banco_3/servlet/FrontControllerServlet"
method="post">
      <input type="hidden" name="method" value="cadastrar"/>
</form>
```

Quadro 4.42 – Exemplo da inclusão da *tag* do tipo *hidden*.

O valor de cada *tag hidden* é específico para formulário, ou seja, porque cada formulário solicita a execução de ação em particular. A tabela 4.2 mostra os valores para cada arquivo de entrada a dados. Nas páginas em que não forem mostrados os valores, é porque ele será informado a partir da execução de uma fução *javascript*, mesmo assim, as *tags hidden* têm que ser incluídas no formulário, mesmo contendo valores vazios.

Valores dos campos hidden dos formulários	
Página	Valor do campo
cadastrarConta.html	cadastrar
excluirConta.html	excluir
transferirConta.html	transferir
renderJurosConta.html	renderJuros
renderBonusConta.html	renderBonus
consultarSaldoConta.html	consultarSaldo
consultarBonusConta.html	consultarBonus

Tabela 4.2 – Valores dos atributos dos campos do tipo *hidden*.

A segunda alteração foi feita nas páginas de resultado: *resultadoCadastrarConta.jsp*, *resultadoExcluirConta.jsp*, *resultadoTransferirConta.jsp*, *resultadoRenderJurosConta.jsp*, *resultadoRenderBonusConta.jsp*, *resultadoConsultarSaldoConta.jsp* e *resultadoConsultarBonusConta.jsp*. Em todas essas páginas, foi apenas inserido um *scriptlet* para escrever a mensagem. O quadro 4.43 mostra o exemplo da alteração.

```
<%=request.getAttribute("mensagem")%>
```

Quadro 4.43 – *Scriptlet* para mensagem do resultado de ação.

No diretório *WEB-INF*, alteramos a *package* de controle. Agora passaremos a ter apenas um *servlet* de controle e um conjunto de classe de comando para execução das ações. O *servlet FrontControllerServlet* recebe todas as solicitações dos formulários de entrada de dados, e executa para cada formulário específico, suas ações. As classes para cada ação: *CadastrarContaCommand, ExcluirContaCommand, TransferirContaCommand, RenderJurosCommand, RenderBonusCommand, ConsultarSaldoCommand* e *ConsultarBonusCommand*. As classes de comando, implementa a interface *Command*.

A classe *FrontControllerServlet*, responsável pela centralização e execução das operações, sua implementação é mostrada no quadro 4.44.

```
package br.com.ufc.banco.control;

import java.io.IOException;
import java.util.HashMap;
import java.util.Map;

import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class FrontControllerServlet extends HttpServlet {

	private Map<String, Command> map = new HashMap<String, Command>();

	public FrontControllerServlet() {

		map.put("cadastrar", new CadastrarContaCommand());
		map.put("excluir", new ExcluirContaCommand());
		map.put("creditar", new CreditarDebitarContaCommand());
		map.put("debitar", new CreditarDebitarContaCommand());
		map.put("transferir", new TransferirContaCommand());
		map.put("renderJuros", new RenderJurosContaCommand());
		map.put("renderBonus", new RenderBonusContaCommand());
		map.put("consultarSaldo", new ConsultarSaldoContaCommand());
```

```
                map.put("consultarBonus", new ConsultarBonusContaCommand());
        }

        protected void processRequest(HttpServletRequest request,
                        HttpServletResponse response) throws ServletException,
IOException {

                String method = request.getParameter("method");

                Command command = (Command) (map.get(method));
                String retorno = (String) command.execute(request, response);

                RequestDispatcher requestDispatcher = super.getServletContext()
                                .getRequestDispatcher("/pages/" + retorno);
                requestDispatcher.forward(request, response);
        }

        protected void doGet(HttpServletRequest request,
                        HttpServletResponse response) throws ServletException,
IOException {
                processRequest(request, response);
        }

        protected void doPost(HttpServletRequest request,
                        HttpServletResponse response) throws ServletException,
IOException {
                processRequest(request, response);
        }
}
```

Quadro 4.44 – Implementação do *servlet* central.

A interface *Command*, responsável por determinar o formato dos métodos de execução, sua implementação é mostrada no quadro 4.45.

```
package br.com.ufc.banco.control;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public interface Command {

        public Object execute(HttpServletRequest request,
                        HttpServletResponse response);
}
```

Quadro 4.45 – Implementação da interface de comando.

A classe *CadastrarContaCommand*, responsável pela operação de cadastrar conta, sua implementação é mostrada no quadro 4.46.

```
package br.com.ufc.banco.control;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import br.com.ufc.banco.business.BancoBusiness;
import br.com.ufc.banco.business.ContaExistenteException;
import br.com.ufc.banco.business.InterBancoBusiness;

public class CadastrarContaCommand implements Command {

	public Object execute(HttpServletRequest request,
			HttpServletResponse response) {

		String numero = request.getParameter("numeroConta");
		String tipo = request.getParameter("tipoConta");

		try {
			InterBancoBusiness bancoBusiness = new BancoBusiness();
			bancoBusiness.cadastrarConta(new Integer(numero), tipo);

			request.setAttribute("mensagem", "Conta cadastrada com
sucesso!");
		} catch (ContaExistenteException e) {
			request.setAttribute("mensagem", e.getMessage());
		}

		return "resultadoCadastrarConta.jsp";
	}
}
```

Quadro 4.46 – Implementação do comando para cadastrar conta.

A classe *ExcluirContaCommand*, responsável pela operação de excluir conta, sua implementação é mostrada no quadro 4.47.

```
package br.com.ufc.banco.control;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import br.com.ufc.banco.business.BancoBusiness;
import br.com.ufc.banco.business.ContaInexistenteException;
import br.com.ufc.banco.business.InterBancoBusiness;

public class ExcluirContaCommand implements Command {
```

```
	public Object execute(HttpServletRequest request,
			HttpServletResponse response) {

		String numero = request.getParameter("numeroConta");

		try {
			InterBancoBusiness bancoBusiness = new BancoBusiness();
			bancoBusiness.excluirConta(new Integer(numero));

			request.setAttribute("mensagem", "Conta excluída com
sucesso!");
		} catch (ContaInexistenteException e) {
			request.setAttribute("mensagem", e.getMessage());
		}

		return "resultadoExcluirConta.jsp";
	}
}
```

Quadro 4.47 – Implementação do comando para excluir conta.

A classe *TransferirContaCommand*, responsável pela operação de transferir valores entre contas, sua implementação é mostrada no quadro 4.48.

```
package br.com.ufc.banco.control;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import br.com.ufc.banco.business.BancoBusiness;
import br.com.ufc.banco.business.ContaInexistenteException;
import br.com.ufc.banco.business.InterBancoBusiness;
import br.com.ufc.banco.business.LimiteContaException;

public class TransferirContaCommand implements Command {

	public Object execute(HttpServletRequest request,
			HttpServletResponse response) {

		String numeroOrigem = request.getParameter("numeroContaOrigem");
		String numeroDestino = request.getParameter("numeroContaDestino");
		String valor = request.getParameter("valor");

		try {
			InterBancoBusiness bancoBusiness = new BancoBusiness();
			bancoBusiness.efetuarTransferencia(new
Integer(numeroOrigem),
					new Integer(numeroDestino), new Double(valor));
```

```
                        request.setAttribute("mensagem",
                                        "Transferência realizada com sucesso!");
                } catch (ContaInexistenteException e) {
                        request.setAttribute("mensagem", e.getMessage());
                } catch (LimiteContaException e) {
                        request.setAttribute("mensagem", e.getMessage());
                }

                return "resultadoTransferirConta.jsp";
        }
}
```

Quadro 4.48 – Implementação do comando para transferir valores entre conta.

A classe *RenderJurosContaCommand*, responsável pela operação de render juros conta, sua implementação é mostrada no quadro 4.49.

```
package br.com.ufc.banco.control;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import br.com.ufc.banco.business.BancoBusiness;
import br.com.ufc.banco.business.ContaInexistenteException;
import br.com.ufc.banco.business.ContaPoupancaException;
import br.com.ufc.banco.business.InterBancoBusiness;

public class RenderJurosContaCommand implements Command {

        public Object execute(HttpServletRequest request,
                        HttpServletResponse response) {

                String numero = request.getParameter("numeroConta");
                String valor = request.getParameter("valorJuros");

                try {
                        InterBancoBusiness bancoBusiness = new BancoBusiness();
                        bancoBusiness.renderJuros(new Integer(numero), new
Double(valor));

                        request.setAttribute("mensagem",
                                        "Render Juros realizado com sucesso!");
                } catch (ContaInexistenteException e) {
                        request.setAttribute("mensagem", e.getMessage());
                } catch (ContaPoupancaException e) {
                        request.setAttribute("mensagem", e.getMessage());
                }

                return "resultadoRenderJurosConta.jsp";
        }
```

```
}
```

Quadro 4.49 – Implementação do comando para render juros conta.

A classe *RenderBonusCommand*, responsável pela operação de render bônus conta, sua implementação é mostrada no quadro 4.50.

```
package br.com.ufc.banco.control;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import br.com.ufc.banco.business.BancoBusiness;
import br.com.ufc.banco.business.ContaBonusException;
import br.com.ufc.banco.business.ContaInexistenteException;
import br.com.ufc.banco.business.InterBancoBusiness;

public class RenderBonusContaCommand implements Command {

	public Object execute(HttpServletRequest request,
			HttpServletResponse response) {

		String numero = request.getParameter("numeroConta");

		try {
			InterBancoBusiness bancoBusiness = new BancoBusiness();
			bancoBusiness.renderBonus(new Integer(numero));

			request.setAttribute("mensagem",
					"Render Bônus realizado com sucesso!!");
		} catch (ContaInexistenteException e) {
			request.setAttribute("mensagem", e.getMessage());
		} catch (ContaBonusException e) {
			request.setAttribute("mensagem", e.getMessage());
		}

		return "resultadoRenderBonusConta.jsp";
	}
}
```

Quadro 4.50 – Implementação do commando para render bônus conta.

A classe *ConsultarSaldoContaCommand*, responsável pela operação de consultar saldo conta, sua implementação é mostrada no quadro 4.51.

```
package br.com.ufc.banco.control;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

```
import br.com.ufc.banco.business.BancoBusiness;
import br.com.ufc.banco.business.ContaInexistenteException;
import br.com.ufc.banco.business.InterBancoBusiness;

public class ConsultarSaldoContaCommand implements Command {

	public Object execute(HttpServletRequest request,
			HttpServletResponse response) {

		String numero = request.getParameter("numeroConta");

		try {
			InterBancoBusiness bancoBusiness = new BancoBusiness();
			Double saldo = bancoBusiness.consultarSaldo(new
Integer(numero));

			request.setAttribute("mensagem", "Saldo conta " + numero + "
R$:"
					+ saldo);
		} catch (ContaInexistenteException e) {
			request.setAttribute("mensagem", e.getMessage());
		}

		return "resultadoConsultarSaldoConta.jsp";
	}
}
```

Quadro 4.51 – Implementação do commando para consultar saldo conta.

A classe *ConsultarBonusContaCommand*, responsável pela operação de consultar bônus conta, sua implementação é mostrada no quadro 4.52.

```
package br.com.ufc.banco.control;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import br.com.ufc.banco.business.BancoBusiness;
import br.com.ufc.banco.business.ContaBonusException;
import br.com.ufc.banco.business.ContaInexistenteException;
import br.com.ufc.banco.business.InterBancoBusiness;

public class ConsultarBonusContaCommand implements Command {

	public Object execute(HttpServletRequest request,
			HttpServletResponse response) {

		String numero = request.getParameter("numeroConta");

		try {
```

```
                        InterBancoBusiness bancoBusiness = new BancoBusiness();
                        Double saldo = bancoBusiness.consultarBonus(new
Integer(numero));

                        request.setAttribute("mensagem", "Bônus conta " + numero + "
R$:"
                                        + saldo);
                } catch (ContaInexistenteException e) {
                        request.setAttribute("mensagem", e.getMessage());
                } catch (ContaBonusException e) {
                        request.setAttribute("mensagem", e.getMessage());
                }

                return "resultadoConsultarBonusConta.jsp";
        }
}
```

Quadro 4.52 – Implementação do commando para consultar bônus conta.

O arquivo foi alterado conforme o quadro 4.53.

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
      version="2.4">

      <servlet>
      <servlet-name>FrontControllerServlet</servlet-name>
      <servlet-
class>br.com.ufc.banco.control.FrontControllerServlet</servlet-class>
      </servlet>
      <servlet-mapping>
      <servlet-name>FrontControllerServlet</servlet-name>
      <url-pattern>/servlet/FrontControllerServlet</url-pattern>
      </servlet-mapping>

      <session-config>
            <session-timeout>30</session-timeout>
      </session-config>
      <welcome-file-list>
            <welcome-file>index.html</welcome-file>
      </welcome-file-list>
</web-app>
```

Quadro 4.53 – Conteúdo do arquivo *web.xml*.

Utilizando-se do modelo *MVC*, conseguimos resolver as deficiências dos dois primeiros desenvolvimentos. Assim o programador não precisa se preocupar com conhecimentos específicos do profissional de *design* e o profissional de *design* não precisa se preocupar com conhecimentos específicos de programação. Também facilita

a manutenção de classe e página, como também, permite que ambas as partes trabalhem independentes.

4.4 Banco com Struts, MVC e JDBC

Neste tópico desenvolvemos o mesmo exemplo da aplicação bancária utilizando o *framework Struts* que implementa o padrão de projeto *MVC*.

A implementação do sistema bancário no tópico anterior é a forma mais correta de se desenvolver uma aplicação *web*, mas ainda é um desenvolvimento muito artesanal, onde necessita de um investimento, em decorrência do tempo, muito alto. Portanto, a partir deste tópico, mostramos os benefícios que um *framework* agrega em um desenvolvimento de *software*.

O *Struts* é um *framework*, de muitos no mercado, que implementa o padrão de projeto *MVC*. As páginas submetem os formulário para o controlado do próprio *Struts*, e esse, se encarrega de executar as ações. A figura 4.21 mostra o modelo de interação entre as camadas.

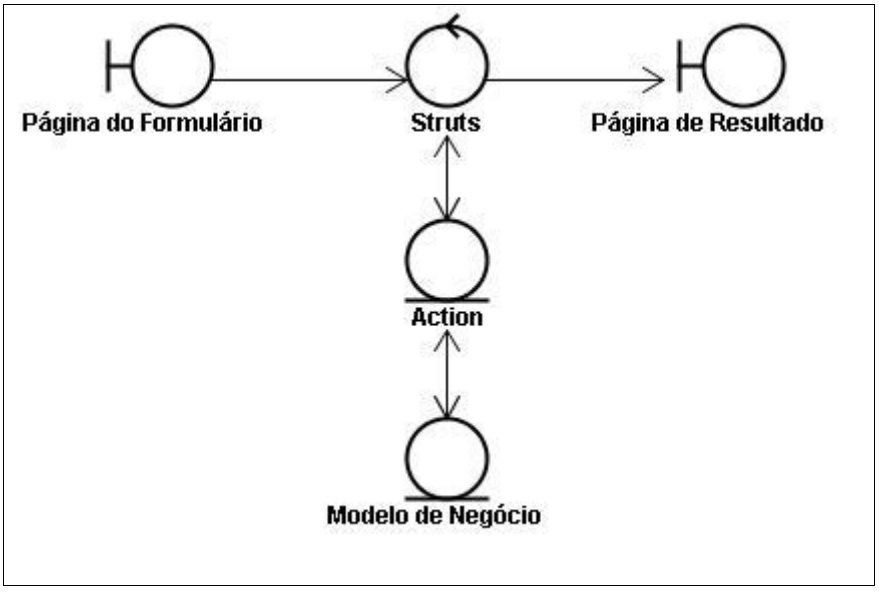

Figura 4.21 – Modelo da interação no modelo *MVC* com *Struts*.

Na figura 4.22, mostra a estrutua dos diretórios.

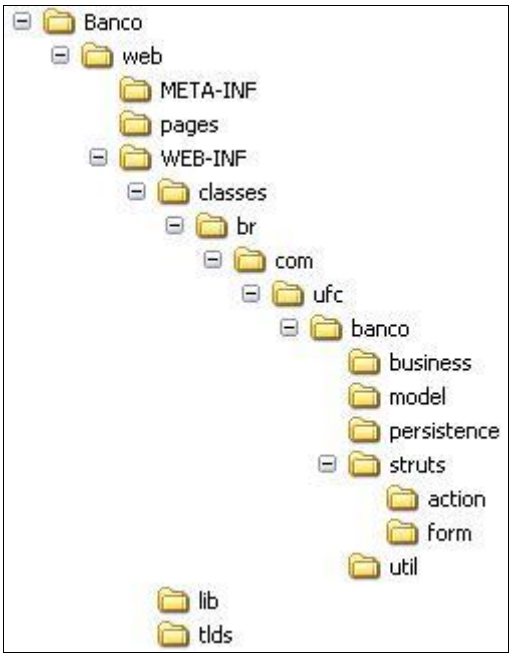

Figura 4.22 – Estrutura do diretório para a aplicação com *Struts*.

As *tags JSP* nas bibliotecas de *tags Struts* oferecem vários recursos comuns que ajudam a tornar as *tags* convenientes de usar, inclusive o escopo automático, nomes comuns da propriedade, sintaxe estendida, expressões da execução e um atributo de erro comum[11]. São basicamente quatro tipos: *bean*, *html*, *logic*, *nesting*. A *tag bean* é útil ao acessar os *JavaBeans* e suas propriedades, assim como ao definir novos componentes, a *tag html* é usada para criar formulários de entradas *HTML* que podem interagir com o *framework Struts* e as *tags HTML* afins, a *tag logic* é usada para gerenciar a geração condicional da saída, fazer um *loop* nas coleções de objetos para uma geração repetida da saída e o gerenciamento do fluxo da aplicação, e por fim, a *tag nesting*, que fornece capacidades de aninhamento melhoradas para as outras *tags Struts*.

No diretório *web*, foi alterado o arquivo *index.jsp* conforme mostrado no quadro 4.54.

```
<%@taglib uri="/tags/struts-bean" prefix="bean"%>
<%@taglib uri="/tags/struts-html" prefix="html"%>
<%@taglib uri="/tags/struts-logic" prefix="logic"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<html:html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
        <title><bean:message key="BANCO"/></title>
    </head>
```

```
    <body>
      <bean:message key="ETI"/><br/>
            <bean:message key="UFC"/>

      <h1><bean:message key="LABEL.TITLE.MENU"/></h1>
        <a
href="<%=request.getContextPath()%>/pages/cadastrarConta.jsp"><bean:me
ssage key="LABEL.LINK.CADASTRAR.CONTA"/></a><br/>
            <a
href="<%=request.getContextPath()%>/pages/excluirConta.jsp"><bean:mess
age key="LABEL.LINK.EXCLUIR.CONTA"/></a><br/>
            <a
href="<%=request.getContextPath()%>/pages/creditarDebitarConta.jsp"><b
ean:message key="LABEL.LINK.CREDITAR.DEBITAR.CONTA"/></a><br/>
            <a
href="<%=request.getContextPath()%>/pages/transferirConta.jsp"><bean:m
essage key="LABEL.LINK.TRANSFERENCIA"/></a><br/>
            <a
href="<%=request.getContextPath()%>/pages/renderJurosConta.jsp"><bean:
message key="LABEL.LINK.RENDE.JUROS"/></a><br/>
            <a
href="<%=request.getContextPath()%>/pages/renderBonusConta.jsp"><bean:
message key="LABEL.LINK.RENDER.BONUS"/></a><br/>
            <a
href="<%=request.getContextPath()%>/pages/consultarSaldoConta.jsp"><be
an:message key="LABEL.LINK.CONSULTAR.SALDO"/></a><br/>
            <a
href="<%=request.getContextPath()%>/pages/consultarBonusConta.jsp"><be
an:message key="LABEL.LINK.CONSULTAR.BONUS"/></a>
    </body>
</html:html>
```

Quadro 4.54 – Conteúdo do arquivo *index.jsp*.

No diretório pages, foram feitas alterações para utilizar as *tag* do *Struts*.

O arquivo *cadastrarConta.jsp* foi alterado conforme mostrado no quadro 4.55.

```
<%@taglib uri="/tags/struts-bean" prefix="bean"%>
<%@taglib uri="/tags/struts-html" prefix="html"%>
<%@taglib uri="/tags/struts-logic" prefix="logic"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<html:html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
        <title><bean:message key="BANCO"/></title>
    </head>
    <body>
      <bean:message key="ETI"/><br/>
            <bean:message key="UFC"/>

      <h1><bean:message key="LABEL.TITLE.CADASTRAR.CONTA"/></h1>
      <hr/>
      <html:form action="/CadastrarConta" method="post">
            <html:hidden property="method" value="cadastrar"/>
            <table>
                  <tr>
```

```
                        <td><bean:message
key="LABEL.FORM.CONTA"/><br/></td>
                        <td>
                              <html:text property="numeroConta"/>
                        </td>
                  </tr>
                  <tr>
                        <td><bean:message key="LABEL.FORM.TIPO"/></td>
                        <td>
                              <html:select property="tipoConta">
                                    <html:option
value=""></html:option>
                                    <html:option
value="C"><bean:message
key="LABEL.LIST.ITEM.CONTA.SIMPLES"/></html:option>
                                    <html:option
value="P"><bean:message
key="LABEL.LIST.ITEM.CONTA.POUPANCA"/></html:option>
                                    <html:option
value="B"><bean:message
key="LABEL.LIST.ITEM.CONTA.BONUS"/></html:option>
                              </html:select>
                        </td>
                  </tr>
            </table>
            <html:button property="btnCadastra" onclick="submit();">
                  <bean:message key="LABEL.BUTTON.CADASTRAR"/>
            </html:button>
            <input type="button" name="btnLimpar" value="Limpar"
onclick="limpar();"/>
                  <br/>
                  <br/>
            <a
href="<%=request.getContextPath()%>/index.jsp"><bean:message
key="LABEL.LINK.MENU.PRINCIPAL"/></a>
      </html:form>
    </body>
    <script type="text/javascript" language="javascript">
      function limpar(){
            document.forms[0].elements["numeroConta"].value = "";
            document.forms[0].elements["tipoConta"].value = "";
      }
    </script>
</html:html>
```

Quadro 4.55 – Conteúdo do arquivo *cadastrarConta.jsp*.

O arquivo *excluirConta.jsp* foi alterado conforme mostrado no quadro 4.56.

```
<%@taglib uri="/tags/struts-bean" prefix="bean"%>
<%@taglib uri="/tags/struts-html" prefix="html"%>
<%@taglib uri="/tags/struts-logic" prefix="logic"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<html:html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
        <title><bean:message key="BANCO"/></title>
    </head>
```

```
    <body>
      <bean:message key="ETI"/><br/>
            <bean:message key="UFC"/>

      <h1><bean:message key="LABEL.TITLE.EXCLUIR.CONTA"/></h1>
      <hr/>
      <html:form action="ExcluirConta">
            <html:hidden property="method" value="excluir"/>
            <table>
                  <tr>
                        <td><bean:message key="LABEL.FORM.CONTA"/></td>
                        <td>
                              <html:text property="numeroConta"/>
                        </td>
                  </tr>
            </table>
            <html:button property="btnExcluir" onclick="submit();">
                  <bean:message key="LABEL.BUTTON.EXCLUIR"/>
            </html:button>
            <input type="button" name="btnLimpar" value="Limpar"
onclick="limpar();"/>
            <br/>
                  <br/>
            <a
href="<%=request.getContextPath()%>/index.jsp"><bean:message
key="LABEL.LINK.MENU.PRINCIPAL"/></a>
      </html:form>
    </body>
    <script type="text/javascript" language="javascript">
        function limpar(){
            document.forms[0].elements["numeroConta"].value = "";
      }
    </script>
</html:html>
```

Quadro 4.56 – Conteúdo do arquivo *excluirConta.jsp*.

O arquivo *transferirConta.jsp* foi alterado conforme mostrado no quadro 4.57.

```
<%@taglib uri="/tags/struts-bean" prefix="bean"%>
<%@taglib uri="/tags/struts-html" prefix="html"%>
<%@taglib uri="/tags/struts-logic" prefix="logic"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<html:html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
        <title><bean:message key="BANCO"/></title>
    </head>
    <body>
      <bean:message key="ETI"/><br/>
            <bean:message key="UFC"/>

      <h1><bean:message key="LABEL.TITLE.TRANSFERIR.CONTA"/></h1>
      <hr/>
      <html:form action="TransferirConta" method="post">
            <html:hidden property="method" value="transferir"/>
            <table>
```

```
                    <tr>
                          <td><bean:message
key="LABEL.FORM.CONTA.ORIGEM"/><br/></td>
                          <td>
                                <html:text property="numeroContaOrigem"/>
                          </td>
                    </tr>
                    <tr>
                          <td><bean:message
key="LABEL.FORM.CONTA.DESTINO"/><br/></td>
                          <td>
                                <html:text
property="numeroContaDestino"/>
                          </td>
                    </tr>
                    <tr>
                          <td><bean:message
key="LABEL.FORM.VALOR"/><br/></td>
                          <td>
                                <html:text property="valor"/>
                          </td>
                    </tr>
              </table>
              <html:button property="btnTransferir" onclick="submit();">
                    <bean:message key="LABEL.BUTTON.TRANSFERIR"/>
              </html:button>
              <input type="button" name="btnLimpar" value="Limpar"
onclick="limpar();"/>
              <br/>
                    <br/>
              <a
href="<%=request.getContextPath()%>/index.jsp"><bean:message
key="LABEL.LINK.MENU.PRINCIPAL"/></a>
        </html:form>
    </body>
    <script type="text/javascript" language="javascript">
        function limpar(){
            document.forms[0].elements["numeroContaOrigem"].value = "";
            document.forms[0].elements["numeroContaDestino"].value =
"";
            document.forms[0].elements["valor"].value = "";
      }
    </script>
</html:html>
```

Quadro 4.57 – Conteúdo do arquivo *transferirConta.jsp*.

O arquivo *renderJurosConta.jsp* foi alterado conforme mostrado no quadro 4.58.

```
<%@taglib uri="/tags/struts-bean" prefix="bean"%>
<%@taglib uri="/tags/struts-html" prefix="html"%>
<%@taglib uri="/tags/struts-logic" prefix="logic"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<html:html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
        <title><bean:message key="BANCO"/></title>
    </head>
```

```
    <body>
      <bean:message key="ETI"/><br/>
            <bean:message key="UFC"/>

      <hr/>
            <bean:write name="mensagem"/>
            <br/>
            <br/>
            <a
href="<%=request.getContextPath()%>/index.jsp"><bean:message
key="LABEL.LINK.MENU.PRINCIPAL"/></a>
    </body>
</html:html>
```

Quadro 4.58 – Conteúdo do arquivo *renderJurosConta.jsp*.

O arquivo *renderBonusConta.jsp* foi alterado conforme mostrado no quadro 4.59.

```
<%@taglib uri="/tags/struts-bean" prefix="bean"%>
<%@taglib uri="/tags/struts-html" prefix="html"%>
<%@taglib uri="/tags/struts-logic" prefix="logic"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<html:html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
        <title><bean:message key="BANCO"/></title>
    </head>
    <body>
      <bean:message key="ETI"/><br/>
            <bean:message key="UFC"/>

      <hr/>
            <bean:write name="mensagem"/>
            <br/>
            <br/>
            <a
href="<%=request.getContextPath()%>/index.jsp"><bean:message
key="LABEL.LINK.MENU.PRINCIPAL"/></a>
    </body>
</html:html>
```

Quadro 4.59 – Conteúdo do arquivo *renderBonusConta.jsp*.

O arquivo *consultarSaldoConta.jsp* foi alterado conforme mostrado no quadro 4.60.

```
<%@taglib uri="/tags/struts-bean" prefix="bean"%>
<%@taglib uri="/tags/struts-html" prefix="html"%>
<%@taglib uri="/tags/struts-logic" prefix="logic"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<html:html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
        <title><bean:message key="BANCO"/></title>
    </head>
```

```
    <body>
      <bean:message key="ETI"/><br/>
            <bean:message key="UFC"/>

      <h1><bean:message key="LABEL.TITLE.CONSULTAR.SALDO.CONTA"/></h1>
      <hr/>
            <html:form action="ConsultarSaldoConta" method="post">
                  <html:hidden property="method"
value="consultarSaldo"/>
            <table>
                  <tr>
                        <td><bean:message key="LABEL.FORM.CONTA"/></td>
                        <td>
                              <html:text property="numeroConta"/>
                        </td>
                  </tr>
            </table>
            <html:button property="btnConsultarSaldo"
onclick="submit();">
                  <bean:message key="LABEL.BUTTON.CONSULTAR.SALDO"/>
            </html:button>
            <input type="button" name="btnLimpar" value="Limpar"
onclick="limpar();"/>
            <br/>
                  <br/>
            <a
href="<%=request.getContextPath()%>/index.jsp"><bean:message
key="LABEL.LINK.MENU.PRINCIPAL"/></a>
      </html:form>
    </body>
    <script type="text/javascript" language="javascript">
        function limpar(){
            document.forms[0].elements["numeroConta"].value = "";
      }
    </script>
</html:html>
```

Quadro 4.60 – Conteúdo do arquivo *consultarSaldoConta.jsp*.

O arquivo *consultarBonusConta.jsp* foi alterado conforme mostrado no quadro 4.61.

```
<%@taglib uri="/tags/struts-bean" prefix="bean"%>
<%@taglib uri="/tags/struts-html" prefix="html"%>
<%@taglib uri="/tags/struts-logic" prefix="logic"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<html:html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
        <title><bean:message key="BANCO"/></title>
    </head>
    <body>
      <bean:message key="ETI"/><br/>
            <bean:message key="UFC"/>

      <h1><bean:message key="LABEL.TITLE.CONSULTAR.BONUS.CONTA"/></h1>
      <hr/>
            <html:form action="ConsultarBonusConta" method="post">
```

```
                  <html:hidden property="method"
value="consultarBonus"/>
            <table>
                  <tr>
                        <td><bean:message key="LABEL.FORM.CONTA"/></td>
                        <td>
                              <html:text property="numeroConta"/>
                        </td>
                  </tr>
            </table>
            <html:button property="btnConsultarBonus"
onclick="submit();">
                  <bean:message key="LABEL.BUTTON.CONSULTAR.BONUS"/>
            </html:button>
            <input type="button" name="btnLimpar" value="Limpar"
onclick="limpar();"/>
            <br/>
                  <br/>
            <a
href="<%=request.getContextPath()%>/index.jsp"><bean:message
key="LABEL.LINK.MENU.PRINCIPAL"/></a>
      </html:form>
    </body>
    <script type="text/javascript" language="javascript">
        function limpar(){
            document.forms[0].elements["numeroConta"].value = "";
      }
    </script>
</html:html>
```

Quadro 4.61 – Conteúdo do arquivo *consultarBonusConta.jsp*.

Os arquivos *resultadoCadastrarConta, resultadoExcluirConta, transferirConta, resultadoRenderJurosConta, resultadoRenderBonusConta, resultadoConsultarSaldoConta,* e *resultadoConsultarBonusConta* foram alterados conforme o mostrado no quadro 4.62.

```
<%@taglib uri="/tags/struts-bean" prefix="bean"%>
<%@taglib uri="/tags/struts-html" prefix="html"%>
<%@taglib uri="/tags/struts-logic" prefix="logic"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<html:html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
        <title><bean:message key="BANCO"/></title>
    </head>
    <body>
      <bean:message key="ETI"/><br/>
            <bean:message key="UFC"/>

      <hr/>
            <bean:write name="mensagem"/>
            <br/>
            <br/>
```

```
            <a
href="<%=request.getContextPath()%>/index.jsp"><bean:message
key="LABEL.LINK.MENU.PRINCIPAL"/></a>
    </body>
</html:html>
```

Quadro 4.62 – Conteúdo do arquivo geral de resultados.

No diretório *WEB-INF* foi criada a *package br.com.ufc.banco.struts.form* onde foram incluídas as classes que possuem os atributos e métodos beans de recuperação de dados de um formulário do *Struts*.

Os *ActionForm* são objetos versáteis, que permitem desenpenhar o papel de coleta do campo, *buffer* de dados, transformador de tipos e objeto de transferência – tudo dentro da extensão de uma única solicitação[11].

O arquivo *CadastrarContaForm* é mostrado no quadro 4.63.

```
package br.com.ufc.banco.struts.form;

import org.apache.struts.action.ActionForm;

public class CadastrarContaForm extends ActionForm {

        private String numeroConta;

        private String tipoConta;

        private String method;

        public String getNumeroConta() {
                return numeroConta;
        }

        public void setNumeroConta(String numeroConta) {
                this.numeroConta = numeroConta;
        }

        public String getTipoConta() {
                return tipoConta;
        }

        public void setTipoConta(String tipoConta) {
                this.tipoConta = tipoConta;
        }

        public String getMethod() {
```

```
		return method;
	}

	public void setMethod(String method) {
		this.method = method;
	}
}
```

Quadro 4.63 – Conteúdo da classe *CadastrarContaForm*.

O arquivo *ExcluirContaForm* é mostrado no quadro 4.64.

```
package br.com.ufc.banco.struts.form;

import org.apache.struts.action.ActionForm;

public class ExcluirContaForm extends ActionForm {

	private String numeroConta;

	private String method;

	public String getNumeroConta() {
		return numeroConta;
	}

	public void setNumeroConta(String numeroConta) {
		this.numeroConta = numeroConta;
	}

	public String getMethod() {
		return method;
	}

	public void setMethod(String method) {
		this.method = method;
	}
}
```

Quadro 4.64 – Conteúdo da classe *ExcluirContaForm*.

O arquivo *TransferirContaForm* é mostrado no quadro 4.65.

```
package br.com.ufc.banco.struts.form;

import org.apache.struts.action.ActionForm;

public class TransferirContaForm extends ActionForm {

	private String numeroContaOrigem;

	private String numeroContaDestino;
```

```
	private String valor;

	private String method;

	public String getNumeroContaOrigem() {
		return numeroContaOrigem;
	}

	public void setNumeroContaOrigem(String numeroContaOrigem) {
		this.numeroContaOrigem = numeroContaOrigem;
	}

	public String getNumeroContaDestino() {
		return numeroContaDestino;
	}

	public void setNumeroContaDestino(String numeroContaDestino) {
		this.numeroContaDestino = numeroContaDestino;
	}

	public String getValor() {
		return valor;
	}

	public void setValor(String valor) {
		this.valor = valor;
	}

	public String getMethod() {
		return method;
	}

	public void setMethod(String method) {
		this.method = method;
	}
}
```

Quadro 4.65 – Conteúdo da classe *TransferirContaForm*.

O arquivo *RenderJurosContaForm* é mostrado no quadro 4.66.

```
package br.com.ufc.banco.struts.form;

import org.apache.struts.action.ActionForm;

public class RenderJurosContaForm extends ActionForm {

	private String numeroConta;

	private String valor;
```

```
	private String method;

	public String getNumeroConta() {
		return numeroConta;
	}

	public void setNumeroConta(String numeroConta) {
		this.numeroConta = numeroConta;
	}

	public String getValor() {
		return valor;
	}

	public void setValor(String valor) {
		this.valor = valor;
	}

	public String getMethod() {
		return method;
	}

	public void setMethod(String method) {
		this.method = method;
	}
}
```

Quadro 4.66 – Conteúdo da classe *RenderJurosContaForm*.

O arquivo *RenderBonusContaForm* é mostrado no quadro 4.67.

```
package br.com.ufc.banco.struts.form;

import org.apache.struts.action.ActionForm;

public class RenderBonusContaForm extends ActionForm {

	private String numeroConta;

	private String method;

	public String getNumeroConta() {
		return numeroConta;
	}

	public void setNumeroConta(String numeroConta) {
		this.numeroConta = numeroConta;
	}

	public String getMethod() {
```

```
		return method;
	}

	public void setMethod(String method) {
		this.method = method;
	}
}
```

Quadro 4.67 – Conteúdo da classe *RenderBonusContaForm*.

O arquivo *ConsultarSaldoContaForm* é mostrado no quadro 4.68.

```
package br.com.ufc.banco.struts.form;

import org.apache.struts.action.ActionForm;

public class ConsultarSaldoContaForm extends ActionForm {

	private String numeroConta;

	private String method;

	public String getNumeroConta() {
		return numeroConta;
	}

	public void setNumeroConta(String numeroConta) {
		this.numeroConta = numeroConta;
	}

	public String getMethod() {
		return method;
	}

	public void setMethod(String method) {
		this.method = method;
	}
}
```

Quadro 4.68 – Conteúdo da classe *ConsultarSaldoContaForm*.

O arquivo *ConsultarBonusContaForm* é mostrado no quadro 4.69.

```
package br.com.ufc.banco.struts.form;

import org.apache.struts.action.ActionForm;

public class ConsultarBonusContaForm extends ActionForm {

	private String numeroConta;

	private String method;
```

```
	public String getNumeroConta() {
		return numeroConta;
	}

	public void setNumeroConta(String numeroConta) {
		this.numeroConta = numeroConta;
	}

	public String getMethod() {
		return method;
	}

	public void setMethod(String method) {
		this.method = method;
	}
}
```

Quadro 4.69 – Conteúdo da classe *ConsultarBonusContaForm*.

Também foi criada a *package br.com.ufc.banco.struts.action* onde foram incluídas as classes de ação que executam as operações de formulário do *Struts*.

Os *Actions* são as classes mais flexíveis no frameworks *Struts*, e podem ser usadas para criar qualquer funcionalidade necessária sempre que você precisar, tais como:

- Acessar a camada de negócio;
- Preparar os objetos de dados para a camada de apresentação;
- Lidar com qualquer erro que surja ínterim.

O arquivo *CadastrarContaAction* é mostrado no quadro 4.70.

```
package br.com.ufc.banco.struts.action;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.actions.DispatchAction;

import br.com.ufc.banco.business.BancoBusiness;
import br.com.ufc.banco.business.ContaExistenteException;
import br.com.ufc.banco.business.InterBancoBusiness;
import br.com.ufc.banco.struts.form.CadastrarContaForm;
```

```
public class CadastrarContaAction extends DispatchAction {

	public ActionForward cadastrar(ActionMapping mapping, ActionForm form,
			HttpServletRequest request, HttpServletResponse response)
			throws Exception {

		CadastrarContaForm cadastrarContaForm = (CadastrarContaForm) form;

		try {
			InterBancoBusiness bancoBusiness = new BancoBusiness();
			bancoBusiness.cadastrarConta(new Integer(cadastrarContaForm
					.getNumeroConta()),
cadastrarContaForm.getTipoConta());

			request.setAttribute("mensagem", "Conta cadastrada com
sucesso!");

		} catch (ContaExistenteException e) {
			request.setAttribute("mensagem", e.getMessage());
		}

		return mapping.findForward("result");
	}
}
```

Quadro 4.70 – Conteúdo da classe *CadastrarContaAction*.

O arquivo *ExcluirContaAction* é mostrado no quadro 4.71.

```
package br.com.ufc.banco.struts.action;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.actions.DispatchAction;

import br.com.ufc.banco.business.BancoBusiness;
import br.com.ufc.banco.business.ContaInexistenteException;
import br.com.ufc.banco.business.InterBancoBusiness;
import br.com.ufc.banco.struts.form.ExcluirContaForm;

public class ExcluirContaAction extends DispatchAction {

	public ActionForward excluir(ActionMapping mapping, ActionForm form,
			HttpServletRequest request, HttpServletResponse response)
			throws Exception {
```

```
                ExcluirContaForm excluirContaForm = (ExcluirContaForm) form;

                try {
                        InterBancoBusiness bancoBusiness = new BancoBusiness();
                        bancoBusiness.excluirConta(new Integer(excluirContaForm
                                        .getNumeroConta()));

                        request.setAttribute("mensagem", "Conta excluída com
sucesso!");
                } catch (ContaInexistenteException e) {
                        request.setAttribute("mensagem", e.getMessage());
                }

                return mapping.findForward("result");
        }
}
```

Quadro 4.71 – Conteúdo da classe *ExcluirContaAction*.

O arquivo *TransferirContaAction* é mostrado no quadro 4.72.

```
package br.com.ufc.banco.struts.action;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.actions.DispatchAction;

import br.com.ufc.banco.business.BancoBusiness;
import br.com.ufc.banco.business.ContaInexistenteException;
import br.com.ufc.banco.business.InterBancoBusiness;
import br.com.ufc.banco.business.LimiteContaException;
import br.com.ufc.banco.struts.form.TransferirContaForm;

public class TransferirContaAction extends DispatchAction {

        public ActionForward transferir(ActionMapping mapping, ActionForm form,
                        HttpServletRequest request, HttpServletResponse response)
                        throws Exception {

                TransferirContaForm transferirContaForm = (TransferirContaForm)
form;

                try {
                        InterBancoBusiness bancoBusiness = new BancoBusiness();
                        bancoBusiness.efetuarTransferencia(new
Integer(transferirContaForm
```

```
                                        .getNumeroContaOrigem()), new
Integer(transferirContaForm
                                        .getNumeroContaDestino()), new
Double(transferirContaForm
                                        .getValor()));

                        request.setAttribute("mensagem",
                                        "Transferência realizada com sucesso!");
                } catch (ContaInexistenteException e) {
                        request.setAttribute("mensagem", e.getMessage());
                } catch (LimiteContaException e) {
                        request.setAttribute("mensagem", e.getMessage());
                }

                return mapping.findForward("result");
        }
}
```

Quadro 4.72 – Conteúdo da classe *TransferirContaAction*.

O arquivo *RenderJurosContaAction* é mostrado no quadro 4.73.

```
package br.com.ufc.banco.struts.action;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.actions.DispatchAction;

import br.com.ufc.banco.business.BancoBusiness;
import br.com.ufc.banco.business.ContaInexistenteException;
import br.com.ufc.banco.business.ContaPoupancaException;
import br.com.ufc.banco.business.InterBancoBusiness;
import br.com.ufc.banco.struts.form.RenderJurosContaForm;

public class RenderJurosContaAction extends DispatchAction {

        public ActionForward renderJuros(ActionMapping mapping, ActionForm form,
                        HttpServletRequest request, HttpServletResponse response)
                        throws Exception {

                RenderJurosContaForm renderJurosContaForm =
(RenderJurosContaForm) form;

                try {
                        InterBancoBusiness bancoBusiness = new BancoBusiness();
                        bancoBusiness.renderJuros(new Integer(renderJurosContaForm
```

```
                                        .getNumeroConta()), new
Double(renderJurosContaForm
                                        .getValor()));

                        request.setAttribute("mensagem",
                                        "Render Juros realizado com sucesso!");
                } catch (ContaInexistenteException e) {
                        request.setAttribute("mensagem", e.getMessage());
                } catch (ContaPoupancaException e) {
                        request.setAttribute("mensagem", e.getMessage());
                }

                return mapping.findForward("result");
        }
}
```

Quadro 4.73 – Conteúdo da classe *RenderJurosContaAction*.

O arquivo *RenderBonusContaAction* é mostrado no quadro 4.74.

```
package br.com.ufc.banco.struts.action;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.actions.DispatchAction;

import br.com.ufc.banco.business.BancoBusiness;
import br.com.ufc.banco.business.ContaBonusException;
import br.com.ufc.banco.business.ContaInexistenteException;
import br.com.ufc.banco.business.InterBancoBusiness;
import br.com.ufc.banco.struts.form.RenderBonusContaForm;

public class RenderBonusContaAction extends DispatchAction {

        public ActionForward renderBonus(ActionMapping mapping, ActionForm form,
                        HttpServletRequest request, HttpServletResponse response)
                        throws Exception {

                RenderBonusContaForm renderBonusContaForm =
(RenderBonusContaForm) form;

                try {
                        InterBancoBusiness bancoBusiness = new BancoBusiness();
                        bancoBusiness.renderBonus(new Integer(renderBonusContaForm
                                        .getNumeroConta()));

                        request.setAttribute("mensagem",
```

```
                                        "Render Bônus realizado com sucesso!");
                } catch (ContaInexistenteException e) {
                        request.setAttribute("mensagem", e.getMessage());
                } catch (ContaBonusException e) {
                        request.setAttribute("mensagem", e.getMessage());
                }

                return mapping.findForward("result");
        }
}
```

Quadro 4.74 – Conteúdo da classe *RenderBonusContaAction*.

O arquivo *ConsultarSaldoContaAction* é mostrado no quadro 4.75.

```
package br.com.ufc.banco.struts.action;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.actions.DispatchAction;

import br.com.ufc.banco.business.BancoBusiness;
import br.com.ufc.banco.business.ContaInexistenteException;
import br.com.ufc.banco.business.InterBancoBusiness;
import br.com.ufc.banco.struts.form.ConsultarSaldoContaForm;

public class ConsultarSaldoContaAction extends DispatchAction {

        public ActionForward consultarSaldo(ActionMapping mapping, ActionForm
form,
                        HttpServletRequest request, HttpServletResponse response)
                        throws Exception {

                ConsultarSaldoContaForm consultarSaldoContaForm =
(ConsultarSaldoContaForm) form;

                try {
                        InterBancoBusiness bancoBusiness = new BancoBusiness();
                        Double saldo = bancoBusiness.consultarSaldo(new Integer(
                                        consultarSaldoContaForm.getNumeroConta()));

                        request
                                        .setAttribute("mensagem", "Saldo conta "
                                                        +
consultarSaldoContaForm.getNumeroConta() + " R$:"
                                                        + saldo);
                } catch (ContaInexistenteException e) {
```

```
                request.setAttribute("mensagem", e.getMessage());
            }

            return mapping.findForward("result");
        }
}
```

Quadro 4.75 – Conteúdo da classe *ConsultarSaldoContaAction*.

O arquivo *ConsultarBonusContaAction* é mostrado no quadro 4.76.

```
package br.com.ufc.banco.struts.action;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.actions.DispatchAction;

import br.com.ufc.banco.business.BancoBusiness;
import br.com.ufc.banco.business.ContaBonusException;
import br.com.ufc.banco.business.ContaInexistenteException;
import br.com.ufc.banco.business.InterBancoBusiness;
import br.com.ufc.banco.struts.form.ConsultarBonusContaForm;

public class ConsultarBonusContaAction extends DispatchAction {

        public ActionForward consultarBonus(ActionMapping mapping, ActionForm
form,
                HttpServletRequest request, HttpServletResponse response)
                throws Exception {

            ConsultarBonusContaForm consultarBonusContaForm =
(ConsultarBonusContaForm) form;

            try {
                InterBancoBusiness bancoBusiness = new BancoBusiness();
                Double saldo = bancoBusiness.consultarBonus(new Integer(
                        consultarBonusContaForm.getNumeroConta()));

                request
                        .setAttribute("mensagem", "Bônus conta "
                                +
consultarBonusContaForm.getNumeroConta() + " R$:"
                                + saldo);
            } catch (ContaInexistenteException e) {
                request.setAttribute("mensagem", e.getMessage());
            } catch (ContaBonusException e) {
                request.setAttribute("mensagem", e.getMessage());
```

```
                }

                return mapping.findForward("result");
        }
}
```

Quadro 4.76 – Conteúdo da classe *ConsultarBonusContaAction*.

No diretório classes, foi criado o arquivo *MessageResources.properties*, que possui todos os *labels* do formulários. Conteúdo mostrado no quadro 4.77.

```
errors.header=<UL>
errors.prefix=<LI>
errors.suffix=</LI>
errors.footer=</UL>

errors.invalid={0} is invalid.
errors.maxlength={0} can not be greater than {1} characters.
errors.minlength={0} can not be less than {1} characters.
errors.range={0} is not in the range {1} through {2}.
errors.required={0} is required.
errors.byte={0} must be an byte.
errors.date={0} is not a date.
errors.double={0} must be an double.
errors.float={0} must be an float.
errors.integer={0} must be an integer.
errors.long={0} must be an long.
errors.short={0} must be an short.
errors.creditcard={0} is not a valid credit card number.
errors.email={0} is an invalid e-mail address.

errors.cancel=Operation cancelled.
errors.detail={0}
errors.general=The process did not complete. Details should follow.
errors.token=Request could not be completed. Operation is not in
sequence.

BANCO=Banco  -  Struts/MVC2/JDBC
ETI=ETI - Especialização em Tecnologia da Informação
UFC=UFC - Universidade Federal do Ceará

LABEL.BUTTON.CADASTRAR=Cadastrar
LABEL.BUTTON.CONSULTAR.SALDO=Consultar Saldo
LABEL.BUTTON.CONSULTAR.BONUS=Consultar Bônus
LABEL.BUTTON.CREDITAR=Creditar
LABEL.BUTTON.DEBITAR=Debitar
LABEL.BUTTON.EXCLUIR=Excluir
LABEL.BUTTON.RENDER.BONUS=Render Bônus
LABEL.BUTTON.RENDER.JUROS=Render Juros
LABEL.BUTTON.TRANSFERIR=Transferir

LABEL.LIST.ITEM.CONTA.BONUS=Conta Bônus
LABEL.LIST.ITEM.CONTA.POUPANCA=Conta Poupança
LABEL.LIST.ITEM.CONTA.SIMPLES=Conta Simples

LABEL.FORM.CONTA=Conta
LABEL.FORM.CONTA.DESTINO=Conta Destino
LABEL.FORM.CONTA.ORIGEM=Conta Origem
LABEL.FORM.JUROS=Juros
```

```
LABEL.FORM.TIPO=Tipo
LABEL.FORM.VALOR=Valor

LABEL.LINK.CADASTRAR.CONTA=Cadastrar Conta
LABEL.LINK.CONSULTAR.BONUS=Consultar Bônus
LABEL.LINK.CONSULTAR.SALDO=Consultar Saldo
LABEL.LINK.CREDITAR.DEBITAR.CONTA=Creditar/Debitar Conta
LABEL.LINK.EXCLUIR.CONTA=Excluir Conta
LABEL.LINK.MENU.PRINCIPAL=Menu Principal
LABEL.LINK.RENDER.BONUS=Render Bônus
LABEL.LINK.RENDE.JUROS=Render Juros
LABEL.LINK.TRANSFERENCIA=Transferência

LABEL.TITLE.CADASTRAR.CONTA=Cadastrar Conta
LABEL.TITLE.CONSULTAR.SALDO.CONTA=Consultar Saldo Conta
LABEL.TITLE.CONSULTAR.BONUS.CONTA=Consultar Bônus Conta
LABEL.TITLE.CREDITAR.DEBITAR.CONTA=Creditar/Debitar Conta
LABEL.TITLE.EXCLUIR.CONTA=Excluir Conta
LABEL.TITLE.MENU=Menu
LABEL.TITLE.RENDER.BONUS.CONTA=Render Bônus Conta
LABEL.TITLE.RENDER.JUROS.CONTA=Render Juros Conta
LABEL.TITLE.TRANSFERIR.CONTA=Transferência Conta
```

Quadro 4.77 – Conteúdo do arquivo *MessageResources.properties*.

No diretório *WEB-INF* incluir os arquivos *struts-config.xml*, *tiles-def.xml*, *validation.xml*, *validation-rules.xml*.

O arquivo *struts-config* é usado para carregar vários componentes críticos do *framework*. Esses objetos compõem a configuração do *Struts*. Ele possui, resumidamente, os mapeamentos das classes de formulário, as classes de ação e o a importação do arquivo de propriedades, mostrado no quadro 4.78.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<!DOCTYPE struts-config PUBLIC
          "-//Apache Software Foundation//DTD Struts Configuration
1.2//EN"
          "http://jakarta.apache.org/struts/dtds/struts-
config_1_2.dtd">

<struts-config>

      <form-beans>
            <form-bean name="CadastrarContaForm"
type="br.com.ufc.banco.struts.form.CadastrarContaForm" />
            <form-bean name="ConsultarBonusContaForm"
type="br.com.ufc.banco.struts.form.ConsultarBonusContaForm" />
            <form-bean name="ConsultarSaldoContaForm"
type="br.com.ufc.banco.struts.form.ConsultarSaldoContaForm" />
            <form-bean name="CreditarDebitarContaForm"
type="br.com.ufc.banco.struts.form.CreditarDebitarContaForm" />
            <form-bean name="ExcluirContaForm"
type="br.com.ufc.banco.struts.form.ExcluirContaForm" />
```

```
            <form-bean name="RenderBonusContaForm"
type="br.com.ufc.banco.struts.form.RenderBonusContaForm" />
            <form-bean name="RenderJurosContaForm"
type="br.com.ufc.banco.struts.form.RenderJurosContaForm" />
            <form-bean name="TransferirContaForm"
type="br.com.ufc.banco.struts.form.TransferirContaForm" />
      </form-beans>

      <global-exceptions></global-exceptions>

      <global-forwards></global-forwards>

      <action-mappings>

            <action path="/CadastrarConta"

      type="br.com.ufc.banco.struts.action.CadastrarContaAction"
                        name="CadastrarContaForm"
                        input="/pages/cadastrarConta.jsp"
                        scope="request"
                        parameter="method">
                  <forward name="result"
path="/pages/resultadoCadastrarConta.jsp" />
            </action>

            <action path="/ConsultarBonusConta"

      type="br.com.ufc.banco.struts.action.ConsultarBonusContaAction"
                        name="ConsultarBonusContaForm"
                        input="/pages/consultarBonusConta.jsp"
                        scope="request"
                        parameter="method">
                  <forward name="result"
path="/pages/resultadoConsultarBonusConta.jsp" />
            </action>

            <action path="/ConsultarSaldoConta"

      type="br.com.ufc.banco.struts.action.ConsultarSaldoContaAction"
                        name="ConsultarSaldoContaForm"
                        input="/pages/consultarSaldoConta.jsp"
                        scope="request"
                        parameter="method">
                  <forward name="result"
path="/pages/resultadoConsultarSaldoConta.jsp" />
            </action>

            <action path="/CreditarDebitarConta"

      type="br.com.ufc.banco.struts.action.CreditarDebitarContaAction"
                        name="CreditarDebitarContaForm"
                        input="/pages/creditarDebitarConta.jsp"
                        scope="request"
                        parameter="method">
                  <forward name="result"
path="/pages/resultadoCreditarDebitarConta.jsp" />
            </action>

            <action path="/ExcluirConta"

      type="br.com.ufc.banco.struts.action.ExcluirContaAction"
```

```
                        name="ExcluirContaForm"
                        input="/pages/excluirConta.jsp"
                        scope="request"
                        parameter="method">
                    <forward name="result"
path="/pages/resultadoExcluirConta.jsp" />
            </action>

            <action path="/RenderBonusConta"

      type="br.com.ufc.banco.struts.action.RenderBonusContaAction"
                        name="RenderBonusContaForm"
                        input="/pages/renderBonusConta.jsp"
                        scope="request"
                        parameter="method">
                    <forward name="result"
path="/pages/resultadoRenderBonusConta.jsp" />
            </action>

            <action path="/RenderJurosConta"

      type="br.com.ufc.banco.struts.action.RenderJurosContaAction"
                        name="RenderJurosContaForm"
                        input="/pages/renderJurosConta.jsp"
                        scope="request"
                        parameter="method">
                    <forward name="result"
path="/pages/resultadoRenderJurosConta.jsp" />
            </action>

            <action path="/TransferirConta"

      type="br.com.ufc.banco.struts.action.TransferirContaAction"
                        name="TransferirContaForm"
                        input="/pages/transferirConta.jsp"
                        scope="request"
                        parameter="method">
                    <forward name="result"
path="/pages/resultadoTransferirConta.jsp" />
            </action>

      </action-mappings>

      <controller
      processorClass="org.apache.struts.tiles.TilesRequestProcessor"
/>

      <message-resources parameter="MessageResources" />

      <plug-in className="org.apache.struts.tiles.TilesPlugin">
            <!-- Path to XML definition file -->
            <set-property property="definitions-config"
                  value="/WEB-INF/tiles-defs.xml" />
            <!-- Set Module-awareness to true -->
            <set-property property="moduleAware" value="true" />
      </plug-in>

      <plug-in
className="org.apache.struts.validator.ValidatorPlugIn">
            <set-property property="pathnames"
```

```
                  value="/WEB-INF/validator-rules.xml,/WEB-
INF/validation.xml" />
      </plug-in>

</struts-config>
```

Quadro 4.78 – Conteúdo do arquivo *struts-config.xml*.

O arquivo *web.xml* foi inserido o mapeamento dos *servlets* do *Struts*. Mostrado no quadro 4.79.

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_2.xsd"
      version="2.4">

      <servlet>
            <servlet-name>action</servlet-name>
            <servlet-class>
                  org.apache.struts.action.ActionServlet
            </servlet-class>
            <init-param>
                  <param-name>config</param-name>
                  <param-value>/WEB-INF/struts-config.xml</param-value>
            </init-param>
            <init-param>
                  <param-name>debug</param-name>
                  <param-value>2</param-value>
            </init-param>
            <init-param>
                  <param-name>detail</param-name>
                  <param-value>2</param-value>
            </init-param>
            <load-on-startup>2</load-on-startup>
      </servlet>

    <servlet-mapping>
      <servlet-name>action</servlet-name>
      <url-pattern>*.do</url-pattern>
    </servlet-mapping>

      <taglib>
            <taglib-uri>/tags/struts-bean</taglib-uri>
            <taglib-location>/WEB-INF/tlds/struts-bean.tld</taglib-
location>
      </taglib>

      <taglib>
            <taglib-uri>/tags/struts-html</taglib-uri>
            <taglib-location>/WEB-INF/tlds/struts-html.tld</taglib-
location>
      </taglib>

      <taglib>
            <taglib-uri>/tags/struts-logic</taglib-uri>
```

```
            <taglib-location>/WEB-INF/tlds/struts-logic.tld</taglib-
location>
      </taglib>

      <session-config>
            <session-timeout>30</session-timeout>
      </session-config>
      <welcome-file-list>
            <welcome-file>index.jsp</welcome-file>
      </welcome-file-list>

</web-app>
```

Quadro 4.79 – Conteúdo do arquivo *web.xml*.

Inicialmente, montar uma estrutura de uma aplicação no *Struts* é um pouco trabalhosa, mas depois de tudo bem definido, o *Struts* agiliza o desenvolvimento com a reusabilidade de suas bibliotecas.

4.5 Banco com JavaServer Faces, MVC e JDBC

Neste tópico, apresentaremos o *framework JavaServer Faces*. O *JSF* também um *framework* com base no padrão de projeto *MVC*.

O *JSF* ajuda no desenvolvimento fácil de aplicativos *web*, trazendo um rico e poderoso conjunto de componentes de interface, como por exemplo: caixas de textos, lista, tabelas entre outros entros recursos já pré-definidos[21]. Também permite validação de entrada de dados pelo usuário, controle de erro, controle dos *JavaBeans* e suporte a internacionalização.

As páginas submetem os formulários para o controlador do próprio *JSF*, e esse, se encarrega de executar as ações. A figura 4.23 mostra o modelo de interação entre as camadas.

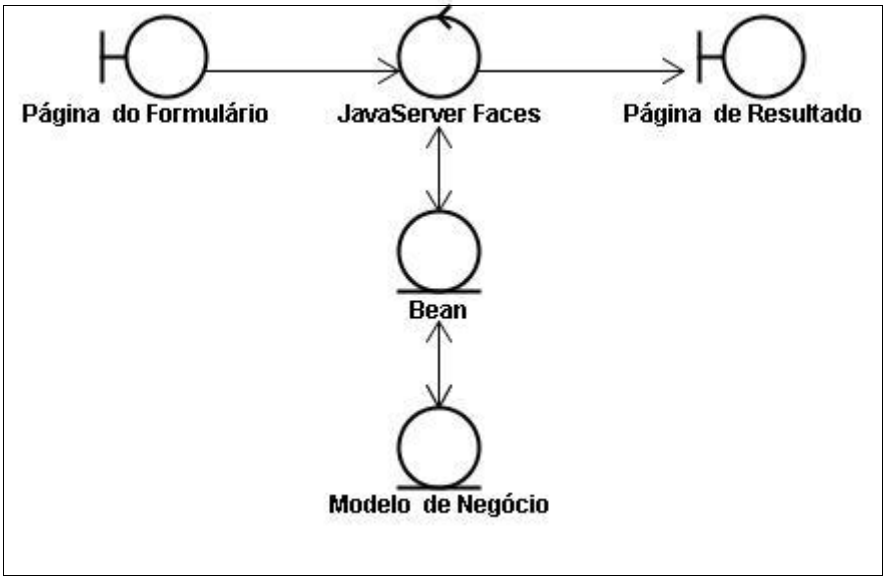

Figura 4.23 – Modelo da interação no modelo *MVC* com *JSF*.

Na figura 4.24, mostra a estrutura de pastas.

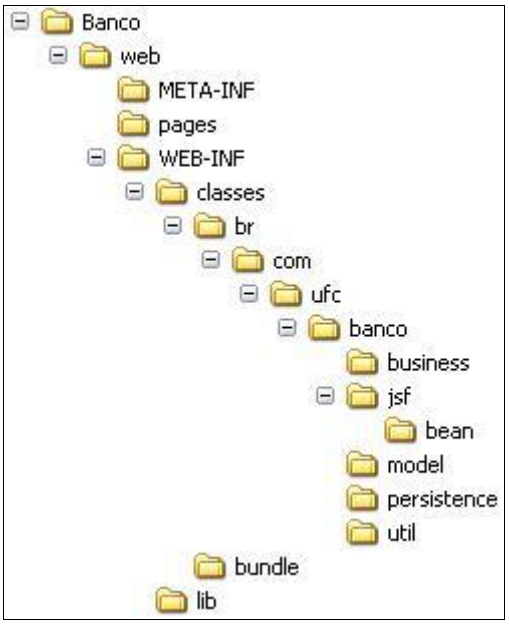

Figura 4.24 – Estrutura do diretório para a aplicação com *Javaserver Faces*.

No diretório *web*, foi alterado o arquivo *index.jsp* conforme mostrado no quadro 4.80.

```
<%@taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<%@taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<f:loadBundle basename="bundle.Messages" var="Message"/>
<html>
      <f:view>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
```

```
        <title><h:outputText value="#{Message.BANCO}"/></title>
    </head>
    <body>
      <h:outputText value="#{Message.ETI}"/><br/>
            <h:outputText value="#{Message.UFC}"/>

      <h1><h:outputText value="#{Message.LABEL_TITLE_MENU}"/></h1>
        <a
href="<%=request.getContextPath()%>/faces/pages/cadastrarConta.jsp"><h
:outputText value="#{Message.LABEL_LINK_CADASTRAR_CONTA}"/></a><br/>
            <a
href="<%=request.getContextPath()%>/faces/pages/excluirConta.jsp"><h:o
utputText value="#{Message.LABEL_LINK_EXCLUIR_CONTA}"/></a><br/>
            <a
href="<%=request.getContextPath()%>/faces/pages/creditarDebitarConta.j
sp"><h:outputText
value="#{Message.LABEL_LINK_CREDITAR_DEBITAR_CONTA}"/></a><br/>
            <a
href="<%=request.getContextPath()%>/faces/pages/transferirConta.jsp"><
h:outputText value="#{Message.LABEL_LINK_TRANSFERENCIA}"/></a><br/>
            <a
href="<%=request.getContextPath()%>/faces/pages/renderJurosConta.jsp">
<h:outputText value="#{Message.LABEL_LINK_RENDE_JUROS}"/></a><br/>
            <a
href="<%=request.getContextPath()%>/faces/pages/renderBonusConta.jsp">
<h:outputText value="#{Message.LABEL_LINK_RENDER_BONUS}"/></a><br/>

            <a
href="<%=request.getContextPath()%>/faces/pages/consultarSaldoConta.js
p"><h:outputText
value="#{Message.LABEL_LINK_CONSULTAR_SALDO}"/></a><br/>
            <a
href="<%=request.getContextPath()%>/faces/pages/consultarBonusConta.js
p"><h:outputText value="#{Message.LABEL_LINK_CONSULTAR_BONUS}"/></a>
    </body>
    </f:view>
</html>
```

Quadro 4.80 – Conteúdo do arquivo *index.jsp*.

No diretório *pages*, foram alteradas as páginas *cadastrarConta.jsp, excluirConta.jsp, transferenciaConta.jsp, renderJurosConta.jsp, renderBonusConta.jsp, consultarSaldoConta.jsp, consultarBonusConta.jsp, resultadoCadastrarConta.jsp, resultadoExcluirConta.jsp, resultadoTransferirConta.jsp, resultadoRenderJurosConta.jsp, resultadoRenderBonusConta.jsp, resultadoConsultarSaldoConta.jsp* e *resultadoConsultarBonusConta.jsp*.

A alteração do arquivo *cadastrarConta.jsp* é mostrado no quadro 4.81.

```
<%@taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<%@taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
```

```
<f:loadBundle basename="bundle.Messages" var="Message"/>
<html>
      <f:view>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
        <title><h:outputText value="#{Message.BANCO}"/></title>
    </head>
    <body>
      <h:outputText value="#{Message.ETI}"/><br/>
            <h:outputText value="#{Message.UFC}"/>

      <h1><h:outputText
value="#{Message.LABEL_TITLE_CADASTRAR_CONTA}"/></h1>
      <hr/>
      <h:form id="ContaForm">
            <table>
                  <tr>
                        <td><h:outputText
value="#{Message.LABEL_FORM_CONTA}"/><br/></td>
                        <td>
                              <h:inputText id="numeroConta"
value="#{CadastrarContaBean.numeroConta}"/>
                        </td>
                  </tr>
                  <tr>
                        <td><h:outputText
value="#{Message.LABEL_FORM_TIPO}"/></td>
                        <td>
                              <h:selectOneMenu id="tipoConta"
value="#{CadastrarContaBean.tipoConta}">
                                    <f:selectItem itemValue=""
itemLabel=""/>
                                    <f:selectItem itemValue="C"
itemLabel="#{Message.LABEL_LIST_ITEM_CONTA_SIMPLES}"/>
                                    <f:selectItem itemValue="P"
itemLabel="#{Message.LABEL_LIST_ITEM_CONTA_POUPANCA}"/>
                                    <f:selectItem itemValue="B"
itemLabel="#{Message.LABEL_LIST_ITEM_CONTA_BONUS}"/>
                              </h:selectOneMenu>
                        </td>
                  </tr>
            </table>
            <h:commandButton action="#{CadastrarContaBean.cadastrar}"
value="#{Message.LABEL_BUTTON_CADASTRAR}"/>
            <input type="button" name="btnLimpar" value="Limpar"
onclick="limpar();"/>
                  <br/>
                  <br/>
            <a
href="<%=request.getContextPath()%>/faces/index.jsp"><h:outputText
value="#{Message.LABEL_LINK_MENU_PRINCIPAL}"/></a>
      </h:form>
    </body>
    </f:view>
    <script>
      function limpar(){
            document.forms[0].elements["ContaForm:numeroConta"].value =
"";
            document.forms[0].elements["ContaForm:tipoConta"].value =
"";
```

```
      }
    </script>
</html>
```

Quadro 4.81 – Conteúdo do arquivo *cadastrarConta.jsp*.

A alteração do arquivo *excluirConta.jsp* é mostrado no quadro 4.82.

```
<%@taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<%@taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<f:loadBundle basename="bundle.Messages" var="Message"/>
<html>
      <f:view>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
        <title><h:outputText value="#{Message.BANCO}"/></title>
    </head>
    <body>
      <h:outputText value="#{Message.ETI}"/><br/>
            <h:outputText value="#{Message.UFC}"/>

      <h1><h:outputText
value="#{Message.LABEL_TITLE_EXCLUIR_CONTA}"/></h1>
      <hr/>
      <h:form id="ContaForm">
            <table>
                  <tr>
                        <td><h:outputText
value="#{Message.LABEL_FORM_CONTA}"/></td>
                        <td>
                              <h:inputText id="numeroConta"
value="#{ExcluirContaBean.numeroConta}"/>
                        </td>
                  </tr>
            </table>
            <h:commandButton action="#{ExcluirContaBean.excluir}"
value="#{Message.LABEL_BUTTON_EXCLUIR}"/>
            <input type="button" name="btnLimpar" value="Limpar"
onclick="limpar();"/>
            <br/>
                  <br/>
            <a
href="<%=request.getContextPath()%>/faces/index.jsp"><h:outputText
value="#{Message.LABEL_LINK_MENU_PRINCIPAL}"/></a>
      </h:form>
    </body>
    </f:view>
    <script>
        function limpar(){
            document.forms[0].elements["ContaForm:numeroConta"].value =
"";
      }
    </script>
</html>
```

Quadro 4.82 – Conteúdo do arquivo *excluirConta.jsp*.

A alteração do arquivo *transferirConta.jsp* é mostrado no quadro 4.83.

```
<%@taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<%@taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<f:loadBundle basename="bundle.Messages" var="Message"/>
<html>
      <f:view>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
        <title><h:outputText value="#{Message.BANCO}"/></title>
    </head>
    <body>
      <h:outputText value="#{Message.ETI}"/><br/>
            <h:outputText value="#{Message.UFC}"/>

      <h1><h:outputText
value="#{Message.LABEL_TITLE_TRANSFERIR_CONTA}"/></h1>
      <hr/>
      <h:form id="ContaForm">
            <table>
                  <tr>
                        <td><h:outputText
value="#{Message.LABEL_FORM_CONTA_ORIGEM}"/><br/></td>
                        <td>
                              <h:inputText id="numeroContaOrigem"
value="#{TransferirContaBean.numeroContaOrigem}"/>
                        </td>
                  </tr>
                  <tr>
                        <td><h:outputText
value="#{Message.LABEL_FORM_CONTA_DESTINO}"/><br/></td>
                        <td>
                              <h:inputText id="numeroContaDestino"
value="#{TransferirContaBean.numeroContaDestino}"/>
                        </td>
                  </tr>
                  <tr>
                        <td><h:outputText
value="#{Message.LABEL_FORM_VALOR}"/><br/></td>
                        <td>
                              <h:inputText id="valor"
value="#{TransferirContaBean.valor}"/>
                        </td>
                  </tr>
            </table>
            <h:commandButton action="#{TransferirContaBean.transferir}"
value="#{Message.LABEL_BUTTON_TRANSFERIR}"/>
            <input type="button" name="btnLimpar" value="Limpar"
onclick="limpar();"/>
            <br/>
                  <br/>
            <a
href="<%=request.getContextPath()%>/faces/index.jsp"><h:outputText
value="#{Message.LABEL_LINK_MENU_PRINCIPAL}"/></a>
      </h:form>
    </body>
    </f:view>
```

```
    <script>
        function limpar(){

      document.forms[0].elements["ContaForm:numeroContaOrigem"].value
= "";

      document.forms[0].elements["ContaForm:numeroContaDestino"].value
= "";
            document.forms[0].elements["ContaForm:valor"].value = "";
      }
    </script>

</html>
```

Quadro 4.83 – Conteúdo do arquivo *transferirConta.jsp*.

A alteração do arquivo *renderJurosConta.jsp* é mostrado no quadro 4.84.

```
<%@taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<%@taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<f:loadBundle basename="bundle.Messages" var="Message"/>
<html>
      <f:view>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
        <title><h:outputText value="#{Message.BANCO}"/></title>
    </head>
    <body>
      <h:outputText value="#{Message.ETI}"/><br/>
            <h:outputText value="#{Message.UFC}"/>

      <hr/>
            <h:outputText value="#{RenderJurosContaBean.mensagem}"/>
            <br/>
            <br/>
            <a
href="<%=request.getContextPath()%>/faces/index.jsp"><h:outputText
value="#{Message.LABEL_LINK_MENU_PRINCIPAL}"/></a>
    </body>
    </f:view>
</html>
```

Quadro 4.84 – Conteúdo do arquivo *renderJurosContaConta.jsp*.

A alteração do arquivo *renderBonusConta.jsp* é mostrado no quadro 4.85.

```
<%@taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<%@taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<f:loadBundle basename="bundle.Messages" var="Message"/>
<html>
      <f:view>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
```

```
        <title><h:outputText value="#{Message.BANCO}"/></title>
    </head>
    <body>
      <h:outputText value="#{Message.ETI}"/><br/>
            <h:outputText value="#{Message.UFC}"/>

      <hr/>
            <h:outputText value="#{RenderBonusContaBean.mensagem}"/>
            <br/>
            <br/>
            <a
href="<%=request.getContextPath()%>/faces/index.jsp"><h:outputText
value="#{Message.LABEL_LINK_MENU_PRINCIPAL}"/></a>
    </body>
    </f:view>
</html>
```

Quadro 4.85 – Conteúdo do arquivo *renderBonusConta.jsp*.

A alteração do arquivo *consultarSaldoConta.jsp* é mostrado no quadro 4.86.

```
<%@taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<%@taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<f:loadBundle basename="bundle.Messages" var="Message"/>
<html>
      <f:view>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
        <title><h:outputText value="#{Message.BANCO}"/></title>
    </head>
    <body>
      <h:outputText value="#{Message.ETI}"/><br/>
            <h:outputText value="#{Message.UFC}"/>

      <h1><h:outputText
value="#{Message.LABEL_TITLE_CONSULTAR_SALDO_CONTA}"/></h1>
      <hr/>
            <h:form id="ContaForm">
            <table>
                  <tr>
                        <td><h:outputText
value="#{Message.LABEL_FORM_CONTA}"/></td>
                        <td>
                              <h:inputText id="numeroConta"
value="#{ConsultarSaldoContaBean.numeroConta}"/>
                        </td>
                  </tr>
            </table>
            <h:commandButton
action="#{ConsultarSaldoContaBean.consultarSaldo}"
value="#{Message.LABEL_BUTTON_CONSULTAR_SALDO}"/>
            <input type="button" name="btnLimpar" value="Limpar"
onclick="limpar();"/>
            <br/>
                  <br/>
```

```
            <a
href="<%=request.getContextPath()%>/faces/index.jsp"><h:outputText
value="#{Message.LABEL_LINK_MENU_PRINCIPAL}"/></a>
            </h:form>
    </body>
    </f:view>
    <script>
        function limpar(){
            document.forms[0].elements["ContaForm:numeroConta"].value =
"";
      }
    </script>
</html>
```

Quadro 4.86 – Conteúdo do arquivo *consultarSaldoConta.jsp*.

A alteração do arquivo *consultarBonusConta.jsp* é mostrado no quadro 4.87.

```
<%@taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<%@taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<f:loadBundle basename="bundle.Messages" var="Message"/>
<html>
      <f:view>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
        <title><h:outputText value="#{Message.BANCO}"/></title>
    </head>
    <body>
      <h:outputText value="#{Message.ETI}"/><br/>
            <h:outputText value="#{Message.UFC}"/>

      <h1><h:outputText
value="#{Message.LABEL_TITLE_CONSULTAR_BONUS_CONTA}"/></h1>
      <hr/>
            <h:form id="ContaForm">
            <table>
                  <tr>
                        <td><h:outputText
value="#{Message.LABEL_FORM_CONTA}"/></td>
                        <td>
                              <h:inputText id="numeroConta"
value="#{ConsultarBonusContaBean.numeroConta}"/>
                        </td>
                  </tr>
            </table>
            <h:commandButton
action="#{ConsultarBonusContaBean.consultarBonus}"
value="#{Message.LABEL_BUTTON_CONSULTAR_BONUS}"/>
            <input type="button" name="btnLimpar" value="Limpar"
onclick="limpar();"/>
            <br/>
                  <br/>
            <a
href="<%=request.getContextPath()%>/faces/index.jsp"><h:outputText
value="#{Message.LABEL_LINK_MENU_PRINCIPAL}"/></a>
            </h:form>
    </body>
```

```
    </f:view>
    <script>
        function limpar(){
            document.forms[0].elements["ContaForm:numeroConta"].value =
"";
      }
    </script>
</html>
```

Quadro 4.87 – Conteúdo do arquivo *consultarBonusConta.jsp*.

As alterações dos arquivos *resultadoCadastrarConta.jsp, resultadoExcluirConta.jsp, resultadoTransferirConta.jsp, resultadoRenderJurosConta.jsp, resultadoRenderBonusConta.jsp, resultadoConsultarSaldoConta.jsp* e *resultadoConsultarBonusConta.jsp* são mostrados no quandro 4.88.

```
<%@taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<%@taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<f:loadBundle basename="bundle.Messages" var="Message"/>
<html>
      <f:view>
      <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
        <title><h:outputText value="#{Message.BANCO}"/></title>
    </head>
    <body>
      <h:outputText value="#{Message.ETI}"/><br/>
            <h:outputText value="#{Message.UFC}"/>

      <hr/>
            <h:outputText value="#{CadastrarContaBean.mensagem}"/>
            <br/>
            <br/>
            <a
href="<%=request.getContextPath()%>/faces/index.jsp"><h:outputText
value="#{Message.LABEL_LINK_MENU_PRINCIPAL}"/></a>
    </body>
    </f:view>
</html>
```

Quadro 4.88 – Conteúdo do arquivo geral de resultados.

Na *package br.com.ufc.banco.jsf.bean* foram colocadas as classes *CadastrarContaBean, ExcluirContaBean, TransferirContaBean, RenderJurosContaBean, RenderBonusContaBean, ConsultarSaldoContaBean* e *ConsultarBonusContaBean*.

O conteúdo da classe *CadastrarContaBean* é mostrado no quadro 4.89.

```
package br.com.ufc.banco.jsf.bean;

import br.com.ufc.banco.business.BancoBusiness;
import br.com.ufc.banco.business.ContaExistenteException;
import br.com.ufc.banco.business.InterBancoBusiness;

public class CadastrarContaBean {

	private Integer numeroConta;

	private String tipoConta;

	private String mensagem;

	public String cadastrar() {

		try {
			InterBancoBusiness bancoBusiness = new BancoBusiness();
			bancoBusiness.cadastrarConta(this.numeroConta, this.tipoConta);

			this.mensagem = "Conta cadastrada com sucesso!";
		} catch (ContaExistenteException e) {
			this.mensagem = e.getMessage();
		}

		return "result";
	}

	public Integer getNumeroConta() {
		return numeroConta;
	}

	public void setNumeroConta(Integer numeroConta) {
		this.numeroConta = numeroConta;
	}

	public String getTipoConta() {
		return tipoConta;
	}

	public void setTipoConta(String tipoConta) {
		this.tipoConta = tipoConta;
	}

	public String getMensagem() {
		return mensagem;
	}

	public void setMensagem(String mensagem) {
```

```
		this.mensagem = mensagem;
	}
}
```

Quadro 4.89 – Conteúdo da classe *CadastrarContaBean*.

O conteúdo da classe *ExcluirContaBean* é mostrado no quadro 4.90.

```
package br.com.ufc.banco.jsf.bean;

import br.com.ufc.banco.business.BancoBusiness;
import br.com.ufc.banco.business.ContaInexistenteException;
import br.com.ufc.banco.business.InterBancoBusiness;

public class ExcluirContaBean {

	private Integer numeroConta;

	private String mensagem;

	public String excluir() {

		try {
			InterBancoBusiness bancoBusiness = new BancoBusiness();
			bancoBusiness.excluirConta(this.numeroConta);

			this.mensagem = "Conta excluída com sucesso!";
		} catch (ContaInexistenteException e) {
			this.mensagem = e.getMessage();
		}

		return "result";
	}

	public Integer getNumeroConta() {
		return numeroConta;
	}

	public void setNumeroConta(Integer numeroConta) {
		this.numeroConta = numeroConta;
	}

	public String getMensagem() {
		return mensagem;
	}

	public void setMensagem(String mensagem) {
		this.mensagem = mensagem;
	}
}
```

Quadro 4.90 – Conteúdo da classe *ExcluirContaBean*.

O conteúdo da classe *TransferirContaBean* é mostrado no quadro 4.91.

```
package br.com.ufc.banco.jsf.bean;

import br.com.ufc.banco.business.BancoBusiness;
import br.com.ufc.banco.business.ContaInexistenteException;
import br.com.ufc.banco.business.InterBancoBusiness;
import br.com.ufc.banco.business.LimiteContaException;

public class TransferirContaBean {

	private Integer numeroContaOrigem;

	private Integer numeroContaDestino;

	private Double valor;

	private String mensagem;

	public String transferir() {

		try {
			InterBancoBusiness bancoBusiness = new BancoBusiness();
			bancoBusiness.efetuarTransferencia(this.numeroContaOrigem,
					this.numeroContaDestino, this.valor);

			this.mensagem = "Transferência realizada com sucesso!";
		} catch (ContaInexistenteException e) {
			this.mensagem = e.getMessage();
		} catch (LimiteContaException e) {
			this.mensagem = e.getMessage();
		}

		return "result";
	}

	public Integer getNumeroContaOrigem() {
		return numeroContaOrigem;
	}

	public void setNumeroContaOrigem(Integer numeroContaOrigem) {
		this.numeroContaOrigem = numeroContaOrigem;
	}

	public Integer getNumeroContaDestino() {
		return numeroContaDestino;
	}

	public void setNumeroContaDestino(Integer numeroContaDestino) {
```

```
		this.numeroContaDestino = numeroContaDestino;
	}

	public Double getValor() {
		return valor;
	}

	public void setValor(Double valor) {
		this.valor = valor;
	}

	public String getMensagem() {
		return mensagem;
	}

	public void setMensagem(String mensagem) {
		this.mensagem = mensagem;
	}
}
```

Quadro 4.91 – Conteúdo da classe *TransferirContaBean*.

O conteúdo da classe *RenderJurosContaBean* é mostrado no quadro 4.92.

```
package br.com.ufc.banco.jsf.bean;

import br.com.ufc.banco.business.BancoBusiness;
import br.com.ufc.banco.business.ContaInexistenteException;
import br.com.ufc.banco.business.ContaPoupancaException;
import br.com.ufc.banco.business.InterBancoBusiness;

public class RenderJurosContaBean {

	private Integer numeroConta;

	private Double valor;

	private String mensagem;

	public String renderJuros() {

		try {
			InterBancoBusiness bancoBusiness = new BancoBusiness();
			bancoBusiness.renderJuros(this.numeroConta, this.valor);

			this.mensagem = "Render Juros realizado com sucesso!";
		} catch (ContaInexistenteException e) {
			this.mensagem = e.getMessage();
		} catch (ContaPoupancaException e) {
			this.mensagem = e.getMessage();
		}
```

```
                return "result";
        }

        public Integer getNumeroConta() {
                return numeroConta;
        }

        public void setNumeroConta(Integer numeroConta) {
                this.numeroConta = numeroConta;
        }

        public Double getValor() {
                return valor;
        }

        public void setValor(Double valor) {
                this.valor = valor;
        }

        public String getMensagem() {
                return mensagem;
        }

        public void setMensagem(String mensagem) {
                this.mensagem = mensagem;
        }
}
```

Quadro 4.92 – Conteúdo da classe *RenderJurosContaBean*.

O conteúdo da classe *RenderBonusContaBean* é mostrado no quadro 4.93.

```
package br.com.ufc.banco.jsf.bean;

import br.com.ufc.banco.business.BancoBusiness;
import br.com.ufc.banco.business.ContaBonusException;
import br.com.ufc.banco.business.ContaInexistenteException;
import br.com.ufc.banco.business.InterBancoBusiness;

public class RenderBonusContaBean {

        private Integer numeroConta;

        private String mensagem;

        public String renderBonus() {

                try {
                        InterBancoBusiness bancoBusiness = new BancoBusiness();
                        bancoBusiness.renderBonus(this.numeroConta);
```

```
                        this.mensagem = "Render Bônus realizado com sucesso!";
                } catch (ContaInexistenteException e) {
                        this.mensagem = e.getMessage();
                } catch (ContaBonusException e) {
                        this.mensagem = e.getMessage();
                }

                return "result";
        }

        public Integer getNumeroConta() {
                return numeroConta;
        }

        public void setNumeroConta(Integer numeroConta) {
                this.numeroConta = numeroConta;
        }

        public String getMensagem() {
                return mensagem;
        }

        public void setMensagem(String mensagem) {
                this.mensagem = mensagem;
        }
}
```

Quadro 4.93 – Conteúdo da classe *RenderBonusContaBean*.

O conteúdo da classe *ConsultarSaldoContaBean* é mostrado no quadro 4.94.

```
package br.com.ufc.banco.jsf.bean;

import br.com.ufc.banco.business.BancoBusiness;
import br.com.ufc.banco.business.ContaInexistenteException;
import br.com.ufc.banco.business.InterBancoBusiness;

public class ConsultarSaldoContaBean {

        private Integer numeroConta;

        private String mensagem;

        public String consultarSaldo() {

                try {
                        InterBancoBusiness bancoBusiness = new BancoBusiness();
                        Double saldo =
bancoBusiness.consultarSaldo(this.numeroConta);
```

```
                        this.mensagem = "Saldo conta " + numeroConta + " R$:" + saldo;
                } catch (ContaInexistenteException e) {
                        this.mensagem = e.getMessage();
                }

                return "result";
        }

        public Integer getNumeroConta() {
                return numeroConta;
        }

        public void setNumeroConta(Integer numeroConta) {
                this.numeroConta = numeroConta;
        }

        public String getMensagem() {
                return mensagem;
        }

        public void setMensagem(String mensagem) {
                this.mensagem = mensagem;
        }
}
```

Quadro 4.94 – Conteúdo da classe *ConsultarSaldoContaBean*.

O conteúdo da classe *ConsultarBonusContaBean* é mostrado no quadro 4.95.

```
package br.com.ufc.banco.jsf.bean;

import br.com.ufc.banco.business.BancoBusiness;
import br.com.ufc.banco.business.ContaBonusException;
import br.com.ufc.banco.business.ContaInexistenteException;
import br.com.ufc.banco.business.InterBancoBusiness;

public class ConsultarBonusContaBean {

        private Integer numeroConta;

        private String mensagem;

        public String consultarBonus() {

                try {
                        InterBancoBusiness bancoBusiness = new BancoBusiness();
                        Double saldo =
bancoBusiness.consultarBonus(this.numeroConta);

                        this.mensagem = "Bônus conta " + numeroConta + " R$:" +
saldo;
```

```
                } catch (ContaInexistenteException e) {
                        this.mensagem = e.getMessage();
                } catch (ContaBonusException e) {
                        this.mensagem = e.getMessage();
                }

                return "result";
        }

        public Integer getNumeroConta() {
                return numeroConta;
        }

        public void setNumeroConta(Integer numeroConta) {
                this.numeroConta = numeroConta;
        }

        public String getMensagem() {
                return mensagem;
        }

        public void setMensagem(String mensagem) {
                this.mensagem = mensagem;
        }
}
```

Quadro 4.95 – Conteúdo da classe *ConsultarBonusContaBean*.

Na *package bundle* encontra-se o arquivo de mensagens *Messages.properties*, conforme mostrado no quadro 4.96.

```
BANCO=Banco  -  JavaServer Faces/MVC2/JDBC
ETI=ETI - Especialização em Tecnologia da Informação
UFC=UFC - Universidade Federal do Ceará

LABEL_BUTTON_CADASTRAR=Cadastrar
LABEL_BUTTON_CONSULTAR_SALDO=Consultar Saldo
LABEL_BUTTON_CONSULTAR_BONUS=Consultar Bônus
LABEL_BUTTON_CREDITAR=Creditar
LABEL_BUTTON_DEBITAR=Debitar
LABEL_BUTTON_EXCLUIR=Excluir
LABEL_BUTTON_RENDER_BONUS=Render Bônus
LABEL_BUTTON_RENDER_JUROS=Render Juros
LABEL_BUTTON_TRANSFERIR=Transferir

LABEL_LIST_ITEM_CONTA_BONUS=Conta Bônus
LABEL_LIST_ITEM_CONTA_POUPANCA=Conta Poupança
LABEL_LIST_ITEM_CONTA_SIMPLES=Conta Simples

LABEL_FORM_CONTA=Conta
LABEL_FORM_CONTA_DESTINO=Conta Destino
LABEL_FORM_CONTA_ORIGEM=Conta Origem
LABEL_FORM_JUROS=Juros
LABEL_FORM_TIPO=Tipo
```

```
LABEL_FORM_VALOR=Valor

LABEL_LINK_CADASTRAR_CONTA=Cadastrar Conta
LABEL_LINK_CONSULTAR_BONUS=Consultar Bônus
LABEL_LINK_CONSULTAR_SALDO=Consultar Saldo
LABEL_LINK_CREDITAR_DEBITAR_CONTA=Creditar/Debitar Conta
LABEL_LINK_EXCLUIR_CONTA=Excluir Conta
LABEL_LINK_MENU_PRINCIPAL=Menu Principal
LABEL_LINK_RENDER_BONUS=Render Bônus
LABEL_LINK_RENDE_JUROS=Render Juros
LABEL_LINK_TRANSFERENCIA=Transferência

LABEL_TITLE_CADASTRAR_CONTA=Cadastrar Conta
LABEL_TITLE_CONSULTAR_SALDO_CONTA=Consultar Saldo Conta
LABEL_TITLE_CONSULTAR_BONUS_CONTA=Consultar Bônus Conta
LABEL_TITLE_CREDITAR_DEBITAR_CONTA=Creditar/Debitar Conta
LABEL_TITLE_EXCLUIR_CONTA=Excluir Conta
LABEL_TITLE_MENU=Menu
LABEL_TITLE_RENDER_BONUS_CONTA=Render Bônus Conta
LABEL_TITLE_RENDER_JUROS_CONTA=Render Juros Conta
LABEL_TITLE_TRANSFERIR_CONTA=Transferência Conta
```

Quadro 4.96 – Conteúdo do arquivo *Messages.properties*.

Conteúdo do arquivo *faces-config.xml* é mostrado no quadro 4.97.

```
<?xml version="1.0"?>
<!DOCTYPE faces-config PUBLIC
  "-//Sun Microsystems, Inc.//DTD JavaServer Faces Config 1.0//EN"
  "http://java.sun.com/dtd/web-facesconfig_1_0.dtd">

<faces-config>

      <managed-bean>
            <description>Cadastrar Conta</description>
            <managed-bean-name>CadastrarContaBean</managed-bean-name>
      <managed-bean-
class>br.com.ufc.banco.jsf.bean.CadastrarContaBean</managed-bean-
class>
      <managed-bean-scope>request</managed-bean-scope>
      </managed-bean>
      <managed-bean>
            <description>Excluir Conta</description>
            <managed-bean-name>ExcluirContaBean</managed-bean-name>
      <managed-bean-
class>br.com.ufc.banco.jsf.bean.ExcluirContaBean</managed-bean-class>
      <managed-bean-scope>request</managed-bean-scope>
      </managed-bean>
      <managed-bean>
            <description>Creditar/Debitar Conta</description>
            <managed-bean-name>CreditarDebitarContaBean</managed-bean-
name>
      <managed-bean-
class>br.com.ufc.banco.jsf.bean.CreditarDebitarContaBean</managed-
bean-class>
      <managed-bean-scope>request</managed-bean-scope>
      </managed-bean>
      <managed-bean>
            <description>Tranferir Conta</description>
            <managed-bean-name>TransferirContaBean</managed-bean-name>
```

```
	<managed-bean-
class>br.com.ufc.banco.jsf.bean.TransferirContaBean</managed-bean-
class>
	<managed-bean-scope>request</managed-bean-scope>
	</managed-bean>
	<managed-bean>
		<description>Render Juros Conta</description>
		<managed-bean-name>RenderJurosContaBean</managed-bean-name>
	<managed-bean-
class>br.com.ufc.banco.jsf.bean.RenderJurosContaBean</managed-bean-
class>
	<managed-bean-scope>request</managed-bean-scope>
	</managed-bean>
	<managed-bean>
		<description>Render Bônus Conta</description>
		<managed-bean-name>RenderBonusContaBean</managed-bean-name>
	<managed-bean-
class>br.com.ufc.banco.jsf.bean.RenderBonusContaBean</managed-bean-
class>
	<managed-bean-scope>request</managed-bean-scope>
	</managed-bean>
	<managed-bean>
		<description>Consultar Saldo Conta</description>
		<managed-bean-name>ConsultarSaldoContaBean</managed-bean-
name>
	<managed-bean-
class>br.com.ufc.banco.jsf.bean.ConsultarSaldoContaBean</managed-bean-
class>
	<managed-bean-scope>request</managed-bean-scope>
	</managed-bean>
	<managed-bean>
		<description>Imprimir Bônus Conta</description>
		<managed-bean-name>ConsultarBonusContaBean</managed-bean-
name>
	<managed-bean-
class>br.com.ufc.banco.jsf.bean.ConsultarBonusContaBean</managed-bean-
class>
	<managed-bean-scope>request</managed-bean-scope>
	</managed-bean>

	<navigation-rule>
		<from-view-id>/index.jsp</from-view-id>
	<navigation-case>
			<to-view-id>/index.jsp</to-view-id>
	</navigation-case>
	</navigation-rule>
	<navigation-rule>
		<from-view-id>/pages/cadastrarConta.jsp</from-view-id>
	<navigation-case>
			<from-action>#{CadastrarContaBean.cadastrar}</from-
action>
			<from-outcome>result</from-outcome>
			<to-view-id>/pages/resultadoCadastrarConta.jsp</to-
view-id>
	</navigation-case>
	</navigation-rule>
	<navigation-rule>
		<from-view-id>/pages/excluirConta.jsp</from-view-id>
	<navigation-case>
			<from-action>#{ExcluirContaBean.excluir}</from-
action>
```

```
                  <from-outcome>result</from-outcome>
                  <to-view-id>/pages/resultadoExcluirConta.jsp</to-
view-id>
      </navigation-case>
      </navigation-rule>
      <navigation-rule>
            <from-view-id>/pages/creditarDebitarConta.jsp</from-view-
id>
      <navigation-case>
                  <from-
action>#{CreditarDebitarContaBean.creditar}</from-action>
                  <from-outcome>result</from-outcome>
                  <to-view-
id>/pages/resultadoCreditarDebitarConta.jsp</to-view-id>
      </navigation-case>
      <navigation-case>
                  <from-
action>#{CreditarDebitarContaBean.debitar}</from-action>
                  <from-outcome>result</from-outcome>
                  <to-view-
id>/pages/resultadoCreditarDebitarConta.jsp</to-view-id>
      </navigation-case>
      </navigation-rule>
      <navigation-rule>
            <from-view-id>/pages/transferirConta.jsp</from-view-id>
      <navigation-case>
                  <from-action>#{TransferirContaBean.transferir}</from-
action>
                  <from-outcome>result</from-outcome>
                  <to-view-id>/pages/resultadoTransferirConta.jsp</to-
view-id>
      </navigation-case>
      </navigation-rule>
      <navigation-rule>
            <from-view-id>/pages/renderJurosConta.jsp</from-view-id>
      <navigation-case>
                  <from-
action>#{RenderJurosContaBean.renderJuros}</from-action>
                  <from-outcome>result</from-outcome>
                  <to-view-id>/pages/resultadoRenderJurosConta.jsp</to-
view-id>
      </navigation-case>
      </navigation-rule>
      <navigation-rule>
            <from-view-id>/pages/renderBonusConta.jsp</from-view-id>
      <navigation-case>
                  <from-
action>#{RenderBonusContaBean.renderBonus}</from-action>
                  <from-outcome>result</from-outcome>
                  <to-view-id>/pages/resultadoRenderBonusConta.jsp</to-
view-id>
      </navigation-case>
      </navigation-rule>
      <navigation-rule>
            <from-view-id>/pages/consultarSaldoConta.jsp</from-view-id>
      <navigation-case>
                  <from-
action>#{ConsultarSaldoContaBean.consultarSaldo}</from-action>
                  <from-outcome>result</from-outcome>
                  <to-view-
id>/pages/resultadoConsultarSaldoConta.jsp</to-view-id>
```

```
      </navigation-case>
      </navigation-rule>
      <navigation-rule>
            <from-view-id>/pages/consultarBonusConta.jsp</from-view-id>
      <navigation-case>
                  <from-
action>#{ConsultarBonusContaBean.consultarBonus}</from-action>
                  <from-outcome>result</from-outcome>
                  <to-view-
id>/pages/resultadoConsultarBonusConta.jsp</to-view-id>
      </navigation-case>
      </navigation-rule>

</faces-config>
```

Quadro 4.97 – Conteúdo do *arquivo faces-config.xml*.

Conteúdo do arquivo *web.xml* é mostrado no quadro 4.98.

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
      version="2.4">

    <context-param>
        <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
        <param-value>client</param-value>
    </context-param>
    <context-param>
        <param-name>javax.faces.application.CONFIG_FILES</param-name>
        <param-value>/WEB-INF/faces-config.xml</param-value>
    </context-param>
    <servlet>
        <servlet-name>Faces Servlet</servlet-name>
        <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
        <load-on-startup> 1 </load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>/faces/*</url-pattern>
    </servlet-mapping>

      <session-config>
            <session-timeout>30</session-timeout>
      </session-config>
      <welcome-file-list>
            <welcome-file>index.jsp</welcome-file>
      </welcome-file-list>
</web-app>
```

Quadro 4.98 – Conteúdo do arquivo *web.xml*.

Como foi visto, essa técnica permite uma programação a eventos, semelhante a algumas linguagens exclusivas para *desktop*.

4.6 Banco com JavaServer Faces, MVC, JDBC e DAO

Neste tópico, aplicamos o padrão de projeto *DAO* no tratamento da persistência dos dados.

O padrão *DAO*, *Data Access Objetc*, cria uma camada na aplicação, separando as rotinas de acesso a banco de dados, geração de *SQL*, das rotinas de negócio de tal modo que os objetos de negócio não saibam como estão sendo gravados. Com esta solução, desenvolvedores implementam um objeto que é unicamente responsável por receber informação de um armazenamento persistente, onde quer que ele esteja. Isto abstrai a visão do dado usada por uma aplicação do layout da tabela, esquema XML ou arquivo em disco.

Na *package br.com.ufc.banco.persistence* passaremos a ter as classes *BancoSQL*, *ContaDAO*, *OracleContaDAO*, *DAOFactory*, *OracleDAOFactory*.

A classe *BancoSQL* foi alterada conforme mostrado no quadro 4.99.

```
package br.com.ufc.banco.persistence;

import br.com.ufc.banco.model.Conta;

public class BancoSQL {

	DAOFactory daoFactory;

	ContaDAO contaDao;

	public BancoSQL() {

		daoFactory = DAOFactory.getDAOFactory(DAOFactory.ORACLE);
		contaDao = daoFactory.getContaDAO();
	}

	public void inserir(Conta conta) {

		contaDao.inserir(conta);
	}

	public void excluir(Conta conta) {
```

```
                contaDao.excluir(conta);
        }

        public void atualizar(Conta conta) {

                contaDao.atualizar(conta);
        }

        public Conta procurar(Conta conta) {

                return contaDao.procurar(conta);
        }
}
```

Quadro 4.99 – Conteúdo da classe *BancoSQL*.

A figura 4.25, mostra o diagrama de classe do modelo das classes *DAO*.

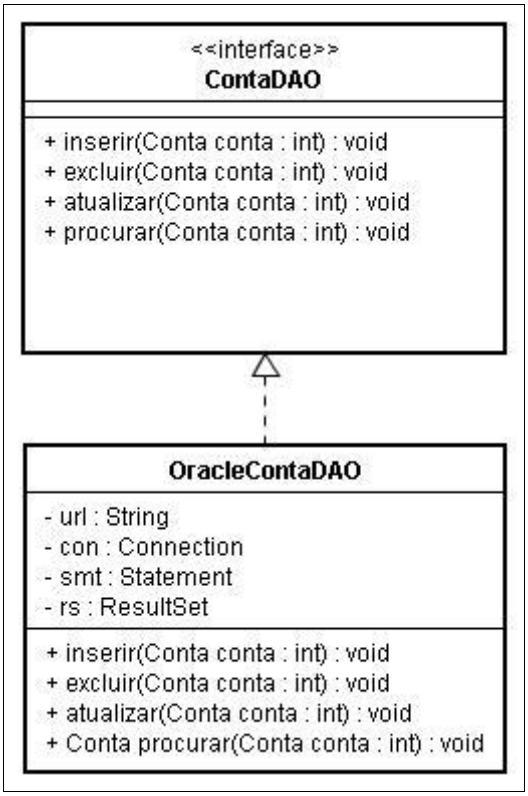

Figura 4.25 – Modelagem das classes *DAO*.

A interface *ContaDAO* é mostrada no quadro 4.100.

```
package br.com.ufc.banco.persistence;

import br.com.ufc.banco.model.Conta;

public interface ContaDAO {

        public void inserir(Conta conta);
```

```
	public void excluir(Conta conta);

	public void atualizar(Conta conta);

	public Conta procurar(Conta conta);
}
```

Quadro 4.100 – Conteúdo da interface *ContaDAO*.

A classe *OracleContaDAO* é mostrada no quadro 4.101.

```
package br.com.ufc.banco.persistence;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import br.com.ufc.banco.model.Conta;
import br.com.ufc.banco.model.ContaBonus;
import br.com.ufc.banco.model.ContaPoupanca;
import br.com.ufc.banco.util.UtilProperties;

public class OracleContaDAO implements ContaDAO {

	private String url;

	private Connection con;

	private Statement stmt;

	private ResultSet rs;

	public OracleContaDAO() {
		try {
			Class.forName(UtilProperties.getAcessoMapCode("DRIVER"));
			url = UtilProperties.getAcessoMapCode("URL");
			con = DriverManager.getConnection(url, UtilProperties
					.getAcessoMapCode("USERNAME"),
UtilProperties
					.getAcessoMapCode("PASSWORD"));
			stmt = con.createStatement();
		} catch (ClassNotFoundException e) {
			System.out.println(e.getMessage());
		} catch (SQLException e) {
			System.out.println(e.getMessage());
		}
	}
```

```
public void inserir(Conta conta) {

        String tipoConta;

        if (conta instanceof ContaBonus) {
                tipoConta = "B";
        } else if (conta instanceof ContaPoupanca) {
                tipoConta = "P";
        } else {
                tipoConta = "C";
        }

        String clausula;
        if ("B".equals(tipoConta)) {
                clausula = "insert into conta (numero, tipo,saldo, bonus) values("
                                + conta.getNumero() + ",\'" + tipoConta + "\',0,0)";
        } else {
                clausula = "insert into conta (numero, tipo,saldo) values("
                                + conta.getNumero() + ",\'" + tipoConta + "\',0)";
        }

        try {
                stmt.executeUpdate(clausula);
        } catch (SQLException e1) {
                System.out.println(e1.getMessage());
        } catch (Exception e1) {
                System.out.println(e1.getMessage());
        }
}

public void excluir(Conta conta) {

        String clausula = "delete from conta where numero ="
                        + conta.getNumero();

        try {
                stmt.executeUpdate(clausula);
        } catch (SQLException e1) {
                System.out.println(e1.getMessage());
        }
}

public void atualizar(Conta conta) {

        String clausula;

        if (conta instanceof ContaBonus) {
                clausula = "update conta set saldo=" + conta.getSaldo()
                                + ", bonus= " + ((ContaBonus) conta).getBonus()
                                + "where numero = " + conta.getNumero();
```

```
                } else {
                        clausula = "update conta set saldo=" + conta.getSaldo()
                                        + "where numero = " + conta.getNumero();
                }

                try {
                        stmt.executeUpdate(clausula);
                } catch (SQLException e1) {
                        System.out.println(e1.getMessage());
                }
        }

        public Conta procurar(Conta conta) {

                Conta c = null;

                String clausula = "select * from conta where numero = "
                                + conta.getNumero();
                try {
                        rs = stmt.executeQuery(clausula);

                        if (rs.next()) {

                                switch (rs.getString("tipo").charAt(0)) {

                                case 'C':
                                        c = new Conta(new
Integer(rs.getString("numero")));
                                        c.setSaldo(new Double(rs.getString("saldo")));
                                        break;

                                case 'P':
                                        c = new ContaPoupanca(new
Integer(rs.getString("numero")));
                                        c.setSaldo(new Double(rs.getString("saldo")));
                                        break;

                                case 'B':
                                        c = new ContaBonus(new
Integer(rs.getString("numero")));
                                        c.setSaldo(new Double(rs.getString("saldo")));
                                        ((ContaBonus) c)
                                                        .setBonus(new
Double(rs.getString("bonus")));

                                        break;

                                }
                        }
                } catch (SQLException e1) {
```

```
                        System.out.println(e1.getMessage());
                }

                return c;
        }
}
```

Quadro 4.101 – Conteúdo da classe *OracleContaDAO*.

A figura 4.26, mostra o diagrama de classe do modelo das classes *Factory*.

A idéia é simples do padrão *Factory* é que em vez de um cliente que precisa de um objeto chamar *new* e assim especificar a classe concreta que ele instancia, o cliente chama um método abstrato (*Factory Method*) especificado em alguma classe abstrata (ou interface) e a subclasse concreta vai decidir que tipo exato de objeto criar e retornar.

Os benefícios dessa técnica basicamente são:

- Mudar a subclasse concreta que cria o objeto permite mudar a classe do objeto criado sem que o cliente saiba;
- Permite estender a funcionalidade através da construção de subclasses sem afetar os clientes.

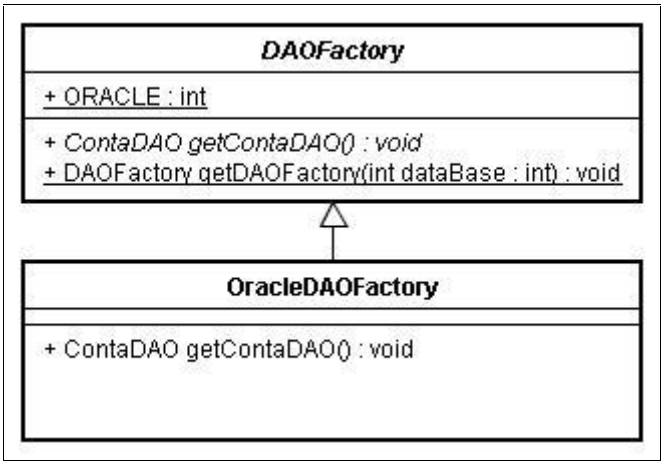

Figura 4.26 – Modelagem das classes *Factory*.

A classe *DAOFactory* é mostrada no quadro 4.102.

```
package br.com.ufc.banco.persistence;
```

```
public abstract class DAOFactory {

	public static final int ORACLE = 1;

	public abstract ContaDAO getContaDAO();

	public static DAOFactory getDAOFactory(int dataBase) {

		switch (dataBase) {
		case ORACLE:
			return new OracleDAOFactory();
		default:
			return null;
		}
	}
}
```

Quadro 4.102 – Conteúdo da classe *DAOFactory*.

A classe *OracleDAOFactory* é mostrada no quadro 4.103.

```
package br.com.ufc.banco.persistence;

public class OracleDAOFactory extends DAOFactory {

	public ContaDAO getContaDAO() {
		return new OracleContaDAO();
	}
}
```

Quadro 4.103 – Conteúdo da classe OracleDAOFactory.

Com essa implementação, se necessário, permite a uma fácil mudança de banco de dados.

4.7 Banco com JavaServer Faces, MVC, JDBC, DAO e Validação

Neste tópico, mostraremos o recurso de validação dos dados dos formulários.

A estrura passa a ter mais uma *package* que contém as classes de validação. A figura 4.27 mostra essa nova estrutura.

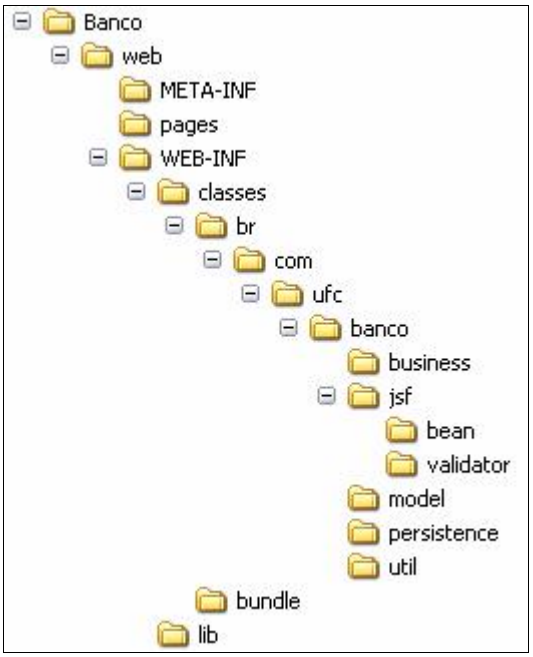

Figura 4.27 – Estrutura do diretório para a aplicação com *Javaserver Faces* com validação.

No diretório *pages*, foram alterados as páginas, *cadastrarConta.jsp*, *excluirConta.jsp*, *transferirConta.jsp, renderJurosConta.jsp, renderBonusConta.jsp, consultarSaldoConta.jsp* e *consultarBonusConta.jsp*. Foram feitas, basicamente, duas alterações, uma para tornar os campos obrigatórios e outra para validar os tipos dos dados.

O arquivo *cadastrarConta.jsp* foi alterado conforme quadro 4.104.

```
            <table>
                  <tr>
                        <td><h:outputText
value="#{Message.LABEL_FORM_CONTA}"/><br/></td>
                        <td>
                              <h:inputText id="numeroConta"
value="#{CadastrarContaBean.numeroConta}" required="true">
                                    <f:validator
validatorId="IntegerValidador"/>
                              </h:inputText>
                              <h:message for="numeroConta"
style="color:red"/>
                        </td>
                  </tr>
                  <tr>
                        <td><h:outputText
value="#{Message.LABEL_FORM_TIPO}"/></td>
                        <td>
                              <h:selectOneMenu id="tipoConta"
value="#{CadastrarContaBean.tipoConta}" required="true">
```

```
                                    <f:selectItem itemValue=""
itemLabel=""/>
                                    <f:selectItem itemValue="C"
itemLabel="#{Message.LABEL_LIST_ITEM_CONTA_SIMPLES}"/>
                                    <f:selectItem itemValue="P"
itemLabel="#{Message.LABEL_LIST_ITEM_CONTA_POUPANCA}"/>
                                    <f:selectItem itemValue="B"
itemLabel="#{Message.LABEL_LIST_ITEM_CONTA_BONUS}"/>
                              </h:selectOneMenu>
                              <h:message for="tipoConta"
style="color:red"/>
                        </td>
                  </tr>
            </table>
```

Quadro 4.104 – Conteúdo do arquivo *cadastrarConta.jsp*.

O arquivo *excluirConta.jsp* foi alterado conforme quadro 4.105.

```
            <table>
                  <tr>
                        <td><h:outputText
value="#{Message.LABEL_FORM_CONTA}"/></td>
                        <td>
                              <h:inputText id="numeroConta"
value="#{ExcluirContaBean.numeroConta}" required="true">
                                    <f:validator
validatorId="IntegerValidador"/>
                              </h:inputText>
                              <h:message for="numeroConta"
style="color:red"/>
                        </td>
                  </tr>
            </table>
```

Quadro 4.105 – Conteúdo do arquivo *excluirConta.jsp*.

O arquivo *transferirConta.jsp* foi alterado conforme quadro 4.106.

```
            <table>
                  <tr>
                        <td><h:outputText
value="#{Message.LABEL_FORM_CONTA_ORIGEM}"/><br/></td>
                        <td>
                              <h:inputText id="numeroContaOrigem"
value="#{TransferirContaBean.numeroContaOrigem}" required="true">
                                    <f:validator
validatorId="IntegerValidador"/>
                              </h:inputText>
                              <h:message for="numeroContaOrigem"
style="color:red"/>
                        </td>
                  </tr>
                  <tr>
                        <td><h:outputText
value="#{Message.LABEL_FORM_CONTA_DESTINO}"/><br/></td>
                        <td>
                              <h:inputText id="numeroContaDestino"
value="#{TransferirContaBean.numeroContaDestino}" required="true">
                                    <f:validator
validatorId="IntegerValidador"/>
```

```
                              </h:inputText>
                                    <h:message for="numeroContaDestino"
style="color:red"/>
                        </td>
                  </tr>
                  <tr>
                        <td><h:outputText
value="#{Message.LABEL_FORM_VALOR}"/><br/></td>
                        <td>
                              <h:inputText id="valor"
value="#{TransferirContaBean.valor}" required="true">
                                          <f:validator
validatorId="DoubleValidador"/>
                              </h:inputText>
                              <h:message for="valor"
style="color:red"/>
                        </td>
                  </tr>
            </table>
```

Quadro 4.106 – Conteúdo do arquivo *transferirConta.jsp*.

O arquivo *renderJurosConta.jsp* foi alterado conforme quadro 4.107.

```
            <table>
                  <tr>
                        <td><h:outputText
value="#{Message.LABEL_FORM_CONTA}"/><br/></td>
                        <td>
                              <h:inputText id="numeroConta"
value="#{RenderJurosContaBean.numeroConta}" required="true">
                                    <f:validator
validatorId="IntegerValidador"/>
                              </h:inputText>
                              <h:message for="numeroConta"
style="color:red"/>
                        </td>
                  </tr>
                  <tr>
                        <td><h:outputText
value="#{Message.LABEL_FORM_JUROS}"/><br/></td>
                        <td>
                              <h:inputText id="valor"
value="#{RenderJurosContaBean.valor}" required="true">
                                          <f:validator
validatorId="DoubleValidador"/>
                              </h:inputText>
                              <h:message for="valor"
style="color:red"/>
                        </td>
                  </tr>
            </table>
```

Quadro 4.107 – Conteúdo do arquivo *renderJurosConta.jsp*.

O arquivo *renderBonusConta.jsp* foi alterado conforme quadro 4.108.

```
            <table>
                  <tr>
                        <td><h:outputText
value="#{Message.LABEL_FORM_CONTA}"/><br/></td>
```

```
                        <td>
                              <h:inputText id="numeroConta"
value="#{RenderBonusContaBean.numeroConta}" required="true">
                                    <f:validator
validatorId="IntegerValidador"/>
                              </h:inputText>
                              <h:message for="numeroConta"
style="color:red"/>
                        </td>
                  </tr>
            </table>
```

Quadro 4.108 – Conteúdo do arquivo *renderBonusConta.jsp*.

O arquivo *consultarSaldoConta.jsp* foi alterado conforme quadro 4.109

```
            <table>
                  <tr>
                        <td><h:outputText
value="#{Message.LABEL_FORM_CONTA}"/></td>
                        <td>
                              <h:inputText id="numeroConta"
value="#{ConsultarSaldoContaBean.numeroConta}" required="true">
                                    <f:validator
validatorId="IntegerValidador"/>
                              </h:inputText>
                              <h:message for="numeroConta"
style="color:red"/>
                        </td>
                  </tr>
            </table>
```

Quadro 4.109 – Conteúdo do arquivo *consultarSaldoConta.jsp*.

O arquivo *consultarBonusConta.jsp* foi alterado conforme quadro 4.110

```
            <table>
                  <tr>
                        <td><h:outputText
value="#{Message.LABEL_FORM_CONTA}"/></td>
                        <td>
                              <h:inputText id="numeroConta"
value="#{ConsultarBonusContaBean.numeroConta}" required="true">
                                    <f:validator
validatorId="IntegerValidador"/>
                              </h:inputText>
                              <h:message for="numeroConta"
style="color:red"/>
                        </td>
                  </tr>
            </table>
```

Quadro 4.110 – Conteúdo do arquivo *consultarBonusConta.jsp*.

Na *package br.com.ufc.banco.jsf.validator* foram incluídas as classes de validação de dados do tipo inteiro e real.

A classe de validação do tipo inteiro é mostrado no quadro 4.111

```
package br.com.ufc.banco.jsf.validator;

import javax.faces.application.FacesMessage;
import javax.faces.component.UIComponent;
import javax.faces.context.FacesContext;
import javax.faces.validator.Validator;
import javax.faces.validator.ValidatorException;

public class IntegerValidator implements Validator {

	public void validate(FacesContext context, UIComponent comp, Object object)
			throws ValidatorException {

		try {
			Integer.parseInt(object.toString());
		} catch (RuntimeException e) {
			FacesMessage message = new FacesMessage();
			message.setSummary("O número não é um inteiro.");
			throw new ValidatorException(message);
		}
	}
}
```

Quadro 4.111 – Conteúdo da classe de validação do tipo inteiro.

A classe de validação do tipo real é mostrada no quadro 4.112

```
package br.com.ufc.banco.jsf.validator;

import javax.faces.application.FacesMessage;
import javax.faces.component.UIComponent;
import javax.faces.context.FacesContext;
import javax.faces.validator.Validator;
import javax.faces.validator.ValidatorException;

public class DoubleValidador implements Validator {

	public void validate(FacesContext context, UIComponent comp, Object object)
			throws ValidatorException {

		try {
			Double.parseDouble(object.toString());
		} catch (RuntimeException e) {
			FacesMessage message = new FacesMessage();
			message.setSummary("O número não é um real.");
			throw new ValidatorException(message);
		}
	}
}
```

Quadro 4.112 – Conteúdo da classe de validação do tipo real.

No arquivo *faces-config.xml* foi adicionado o mapeamento de validação, mostrado no quadro 4.113.

```
<validator>
      <validator-id>
            IntegerValidador
      </validator-id>
      <validator-class>
            br.com.ufc.banco.jsf.validator.IntegerValidator
      </validator-class>
</validator>
<validator>
      <validator-id>
            DoubleValidador
      </validator-id>
      <validator-class>
            br.com.ufc.banco.jsf.validator.DoubleValidador
      </validator-class>
</validator>
```

Quadro 4.113 – Parte do conteúdo do arquivo *faces-config.xm*.

Observamos que foi simples criar nossas validações utilizando o *framework JSF*.

4.8 Banco com JavaServer Faces, MVC, JDBC, DAO, Validação e Hibernate

Neste tópico utilizamos o *Hibernate* para persistência dos dados.

O *Hibernate* é um projeto ambicioso que aponta para um solução completa para o problema de gerenciamente de dados persistentes em *Java*. Fica no meio, entre o aplicativo e um banco de dados relacional, enquanto deixa o desenvolvedor livre para concentrar-se no problema do negócio. Não é uma solução difícil, ou seja, o programador não precisa seguir muitas regras específicas do próprio *Hibernate* e padrões de projeto ao escrever sua lógica de negócio e classes persistentes. Ele se integra suavemente com aplicações mais novas e existentes, não requerendo mudanças bruscas para o restante da aplicação[23].

A estrura passa a ter mais duas *packages* que contém o mapeamento do modelo, e o mapeamento das consultas. A figura 4.28 mostra essa nova estrutura.

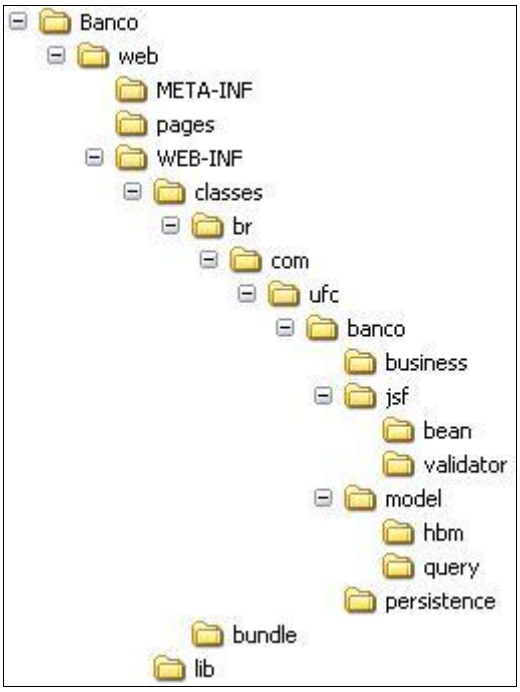

Figura 4.28 – Estrutura do diretório para a aplicação com *Hibernate*.

Na *package br.com.ufc.banco.persistence* teremos agora apenas a classe *BancoSQL*. O quadro 4.114 mostra as alterações da classe.

```
package br.com.ufc.banco.persistence;

import java.util.List;

import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

import br.com.ufc.banco.model.Conta;

public class BancoSQL {

	private SessionFactory factory;

	public BancoSQL() {

		factory = new Configuration().configure().buildSessionFactory();
	}

	public void inserir(Conta conta) {

		Session session = factory.openSession();
		Transaction transaction = session.beginTransaction();
		session.save(conta);
```

```
                transaction.commit();
                session.flush();
                session.close();
        }

        public void excluir(Conta conta) {

                Session session = factory.openSession();
                Transaction transaction = session.beginTransaction();
                session.delete(conta);
                transaction.commit();
                session.flush();
                session.close();
        }

        public void atualizar(Conta conta) {

                Session session = factory.openSession();
                Transaction transaction = session.beginTransaction();
                session.update(conta);
                transaction.commit();
                session.flush();
                session.close();
        }

        public Conta procurar(Conta conta) {

                Session session = factory.openSession();
                Transaction transaction = session.beginTransaction();

                Query query = session

        .getNamedQuery("br.com.ufc.banco.model.query.ContaQuery.contaPorNumero"
);
                query.setString("numeroConta", conta.getNumero().toString());

                List contas = query.list();

                transaction.commit();
                session.close();

                if (contas.size() == 0) {
                        return null;
                } else {
                        return (Conta) contas.get(0);
                }
        }
}
```

Quadro 4.114 – Conteúdo da classe *BancoSQL*.

Toda a configuração do *Hibernate* é feita através de arquivos *XML*, os quais contêm mapeamentos de tabelas e classes *Java*, detalhes de *pooling* de conexões, fornecendo total configurabilidade a aplicação. O *Hibernate* precisa saber como carregar e gravar os objetos de uma classe de persistência. É aí que o *XML* de mapeamento entra.

Os arquivos seguintes indicam ao *Hibernate* qual tabela e seus campos serão persistidos ou consultados.

Na *package br.com.ufc.banco.model.hbm* foi inserido o arquivo *Conta.hbm.xml* de mapeamento do objeto *Conta* para o modelo relacional da tabela, mostrado no quadro 4.115.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
"-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
      <class name="br.com.ufc.banco.model.Conta" table="conta"
discriminator-value="C">

            <id name="numero" column="numero"/>
            <discriminator column="tipo"/>
            <property name="saldo" column="saldo" />

            <subclass name="br.com.ufc.banco.model.ContaPoupanca"
                  discriminator-value="P">
            </subclass>

            <subclass name="br.com.ufc.banco.model.ContaBonus"
                  discriminator-value="B">
                  <property name="bonus" column="bonus" />
            </subclass>

      </class>
</hibernate-mapping>
```

Quadro 4.115 – Conteúdo do arquivo *Conta.hbm.xml*.

Na *package br.com.ufc.banco.model.query* foi inserido o arquivo *ContaQuery.hbm.xml* que possui consulta *HQL*, mostrado no quadro 4.116.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
"-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>

      <query
name="br.com.ufc.banco.model.query.ContaQuery.contaPorNumero">
            select cont
            from br.com.ufc.banco.model.Conta cont
            where cont.numero = :numeroConta
```

```
      </query>

</hibernate-mapping>
```

Quadro 4.116 – Conteúdo do arquivo *ContaQuery.hbm.xml*.

Na diretório classes, foi inserido o arquivo *hibernate.cfg.xml* de configurações, basicamente, do banco de dados e dos mapeamentos de modelo e *query*, mostrado no quadro 4.117.

```
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
      <session-factory>

            <property name="hibernate.dialect">
                  org.hibernate.dialect.OracleDialect
            </property>
            <property name="hibernate.connection.driver_class">
                  oracle.jdbc.driver.OracleDriver
            </property>
            <property name="hibernate.connection.url">
            jdbc:oracle:thin:@localhost:1521:xe
            </property>
            <property name="hibernate.connection.username">
                  system
            </property>
            <property name="hibernate.connection.password">
                  123456
            </property>

            <property name="hibernate.c3p0.max_size">10</property>
            <property name="hibernate.c3p0.min_size">2</property>
            <property name="hibernate.c3p0.timeout">5000</property>

            <property
name="hibernate.c3p0.max_statements">10</property>
            <property
name="hibernate.c3p0.idle_test_period">3000</property>
            <property
name="hibernate.c3p0.acquire_increment">2</property>

            <property name="show_sql">true</property>
        <property name="use_outer_join">true</property>
        <property name="hibernate.generate_statistics">true</property>
        <property name="hibernate.use_sql_comments">true</property>

            <mapping
resource="br/com/ufc/banco/model/hbm/Conta.hbm.xml"/>
            <mapping
resource="br/com/ufc/banco/model/query/ContaQuery.hbm.xml"/>

      </session-factory>
</hibernate-configuration>
```

Quadro 4.117 – Conteúdo do arquivo *hibernate.cfg.xml*.

Observamos que com a utilização deste *framework*, não precisamos mais escrever as *querys* para serem executada no banco de dados, ou seja, o próprio *Hibernate* já faz isso implicitamente, como também utilizamos diretamente os próprios objetos para as operações no banco de dados.

5 Conclusão

Como podemos observar no decorrer do trabalho, através da implementação do estudo de caso, os *frameworks*, possibilitam obter um grau de reúso que de fato reduz o tempo de desenvolvimento, assegurando um elevado padrão de qualidade e uma maior agilidade para o desenvolvimento de uma aplicação *web*. Além disso, a utilização de *framewoks* também proporcionou uma maior simplicidade na resolução de determidados tipos de problemas, onde normalmente necessitaríamos de muito esforço para implementação, testes e correções.

Através das nove refatorações (*refactoring*) apresentadas, introduzimos a utilização dos principais *frameworks* que fornecem suporte ao desenvolvimento *web*. Destacamos as vantagens obtidas e discutimos a integração entre os *frameworks* utilizados. Desta forma, esperamos contribuir para diminuir a curva de aprendizado do desenvolvimento *web* baseado em *frameworks*, incentivar a adoção desta tecnologia e facilitar a transferência de conhecimento dos especialistas em desenvolvimento *web* para os desenvolvedores mais jovens ou iniciantes. Além disso, fornecemos subsídios para uma escolha mais criteriosa, tranqüila e adequada dos *frameworks* a serem utilizados numa aplicação real. Escolha esta que deve levar em consideração os aspectos particulares do sistema a ser desenvolvido.

Atualmente, encontramos vários *frameworks* disponíveis no mercado, entre eles gratuitos e pagos, outros de forte recomendação por parte não só dos grupos de desenvolvedores, como também, por parte das próprias empresas criadoras das linguagens utilizadas. Existe muita literatura disponível, cada uma procurando oferecer um estudo mais direcionado como para a utilização e recursos disponíveis oferecidos pelos *frameworks*, as conhecidas *Bíblias*, outras que se dedicam à integração com outras ferramentas e *frameworks*, e também, aquelas que são mais procuradas pelos programadores que estão se iniciando na tecnologia, mostrando o passo a passo o mínimo necessário para se obter êxito no uso da mesma. Uma grande quantidade de listas de discursões, entre nacionais e internacionais, que se propõem a comentar, estudar e disponibilizar as melhores soluções para os membros da sua e outras comunidades, também podem ser facilmente encontradas. Várias empresas de

consultoria trabalham diretamente no acompanhamento da escolha e implementação da tecnologia mais adequada ao negócio por parte das empresas contratantes.

Por fim, utilizar-se dos benefícios dos *frameworks*, de forma consciente, já tornou-se vital na concorrência e fator de qualidade nas empresas de desenvolvimento de sistemas. Logo, conhecer e dominar esta tecnologia é hoje de fundamental importância para os profissionais em Tecnologia da Informação.

Referências

[1] LARMAN, Craig. Utilizando UML e Padrões. 3ª. Edição, Bookman, 2007.
[2] DSC – Departamento de Sistema e Computação. Disponível em: http://www.dsc.ufcg.edu.br/~jacques/cursos/map/html/frame/oque.htm. Acessado em 01 de janeiro de 2007.
[3] LEITE, Alessandro Ferreira. Frameworks e Padrões de Projeto. Rio de Janeiro: DevMedia Group, 2006.
[4] OLIVEIRA, Eric C. M. O Universo dos Frameworks Java. São Paulo, 2005.
[5] DSC – Departamento de Sistema e Computação. Disponível em: http://www.dsc.ufcg.edu.br/~jacques/cursos/map/html/frame/tipos.htm. Acessado em 01 de janeiro de 2007.
[6] WIKIPÉDIA. Disponível em: http://pt.wikipedia.org/wiki/Apache_Cocoon. Acessado em 04 de janeiro de 2007.
[7] WIKIPÉDIA. Disponível em: http://pt.wikipedia.org/wiki/Struts . Acessado em 04 de janeiro de 2007.
[8] WIKIPÉDIA. Disponível em: http://pt.wikipedia.org/wiki/Hibernate . Acessado em 01 de janeiro de 2007.
[9] NEKI TECHNOLOGIES. Disponível em: http://www.neki-technologies.com.br/jsp/training/catalog.jsp . Acessado em 04 de janeiro de 2007.
[10] WIKIPÉDIA. Disponível em: http://pt.wikipedia.org/wiki/JavaServer_Faces . Acessado em 04 de janeiro de 2007.
[11] HUSTED, Ted; DUMOULIN, Cedric; FRANCISCUS, George; WINTERFELDT, David. Struts em Ação. Rio de Janeiro:Ed. Ciência Moderna, 2004.
[12] WIKIPÉDIA. Disponível em: http://pt.wikipedia.org/wiki/Spring . Acessado em 01 de janeiro de 2007.
[13] MARTIN, Fernando. Velocity. iMasters, 2005. Disponível em: http://www.imasters.com.br/artigo/3240 . Acessado em 04 de janeiro de 2007.
[14] WIKIPÉDIA. Disponível em: http://en.wikipedia.org/wiki/Java_Database_Connectivity. Acessado em 04 de janeiro de 2007.
[15] JOHNSON, Raph E. Documenting frameworks using patterns. Conference on Object Oriented Programming Systems Languages and Applications. Vancouver, British Columbia, Canada, 1992.
[16] GIMENES, I. M. S e HUZITA, E.H.M. Desenvolvimento Baseado em Componentes: Conceitos e Técnicas. Rio de Janeiro, Ciência Moderna, 2005.
[17] JSF (2006): JavaServer Faces. Implementação de referência da Sun disponível em http://java.sun.com/j2ee/javaserverfaces. Última visita em 27/11/2006.
[18] SPRING (2006): Página oficial do framework Spring disponível em http://www.springframework.org. Última visita em 27/11/2006.
[19] Hibernate (2006): Página oficial do framework Hibernate disponível em http://www.hibernate.org. Última visita em 27/11/2006.
[20] HORSTMANN, Cay S. e CORNELL, Gary. Core Java 2 – Volume I – Fundamentos. São Paulo, Makron Books, 2003.
[21] Mann, Kito D. JavaServer Faces in Action, Greenwich, Manning, 2004.
[22] GAMMA, Erich; HELM, Richard; JONHSON, Ralph; VLISSIDES, John. Padrões de Projeto – Soluções reutilizáveis de software orientado a objetos. São Paulo, Bookman, 1995.

[23] BAUER, Christian; KING, Gavin. Hibernate em Ação. Rio de Janeiro, Ciências Moderna, 2005.