ELABORATA

INFORMÁTICA

JEE

Curitiba / 2018

Srs. Alunos

A Elaborata Informática, com o objetivo de continuar prestando-lhe um excelente nível de atendimento e funcionalidade, informa a seguir algumas regras que devem ser observadas quando do uso do labor atório e seus equipamentos, visando mantê-los sempre em um perfeito estado de funcionamento para um melhor aproveitamento de suas aulas.

É proibido:

- Atender celular. Por favor, retire-se da sala, voltando assim que desligar.
- Fazer cópias ilegais de software (piratear), com quaisquer objetivos.
- Retirar da sala de treinamento quaisquer materiais, mesmo que a título de empréstimo.
- Divulgar ou informar produtos ou serviços de outras empresas sem autorização por escrito da direção Elaborata.
- Trazer para a sala de treinamento, qualquer tipo de equipamento pessoal de informática, como por exemplo:
    - Computadores de uso pessoal
    - Notebooks
    - Placas de vídeo
    - Placas de modem
    - Demais periféricos
    - Peças avulsas como memória RAM, ferramentas, etc.
- O consumo de alimentos ou bebidas
- Fumar

Atenciosamente

Elaborata Informática

Sumário

CAPÍTULO 1

INTRODUÇÃO

JEE

1.1 Linguagem Java

A linguagem Java teve início a partir de 1991 com a Sun Microsystems. Inicialmente era parte de outro projeto, denominado de Green Project, e objetivava possibilitar a convergência entre computadores, equipamentos eletrônicos e eletrodomésticos.

O primeiro nome foi OAK (“carvalho”), pois era essa espécie de árvore que James Gosling(criador da linguagem Java) e um pequeno grupo de funcionários, via através de sua janela, sede do projeto, toda vez que a visitava.

Finalizado o projeto o resultado foi um controle remoto chamado de *7(StarSeven), que trazia uma interface gráfica sensível ao toque e interagia com vários equipamentos. Mas o grupo percebeu que não poderia ficar preso a plataformas, pois nem todos os clientes estavam interessados no tipo de processador utilizado e para ficar fazendo uma versão de projeto para cada plataforma era inviável. Desta forma desenvolveram o sistema operacional GreenOS com a linguagem OAK.

A linguagem Oak, surgiu para controlar internamente esses equipamentos. Na época a linguagem era chamada de Oak (em português, “carvalho”).

Como o green project não emplacou no mercado, James Gosling foi chamado para adaptar a linguagem OAK para a internet, surgindo em 1995 a plataforma Java.

Para a época, Java ainda era uma grande incerteza, pois a viabilidade técnica ainda estava sendo estudada para verificar se o hardware seria ou não suficiente para gerenciar uma Virtual Machine. Uma das principais diferenças entre Java e as outras linguagens de programação existentes é que Java era executado numa JVM, ou Java Virtual Machine. Não importava qual plataforma seria executado, pois não haveria problemas.

Isso justifica o slogan criado “write once, run anywhere”, ou em português, “escreva uma vez, rode em qualquer lugar”.

1.2 Instalação e configuração do ambiente de desenvolvimento

Antes de iniciar o desenvolvimento precisamos configurar o ambiente de desenvolvimento JEE(Java Enterprise Edition). Você aprenderá como instalar e

configurar os softwares que ajudarão na montagem de um ambiente de desenvolvimento de um aplicativo web utilizando **Java**, **Apache Tomcat** e **Eclipse**.

Também veremos como instalar e configurar os aplicativos utilizados para a montagem do ambiente de desenvolvimento e explicar o uso de cada uma destas ferramentas.

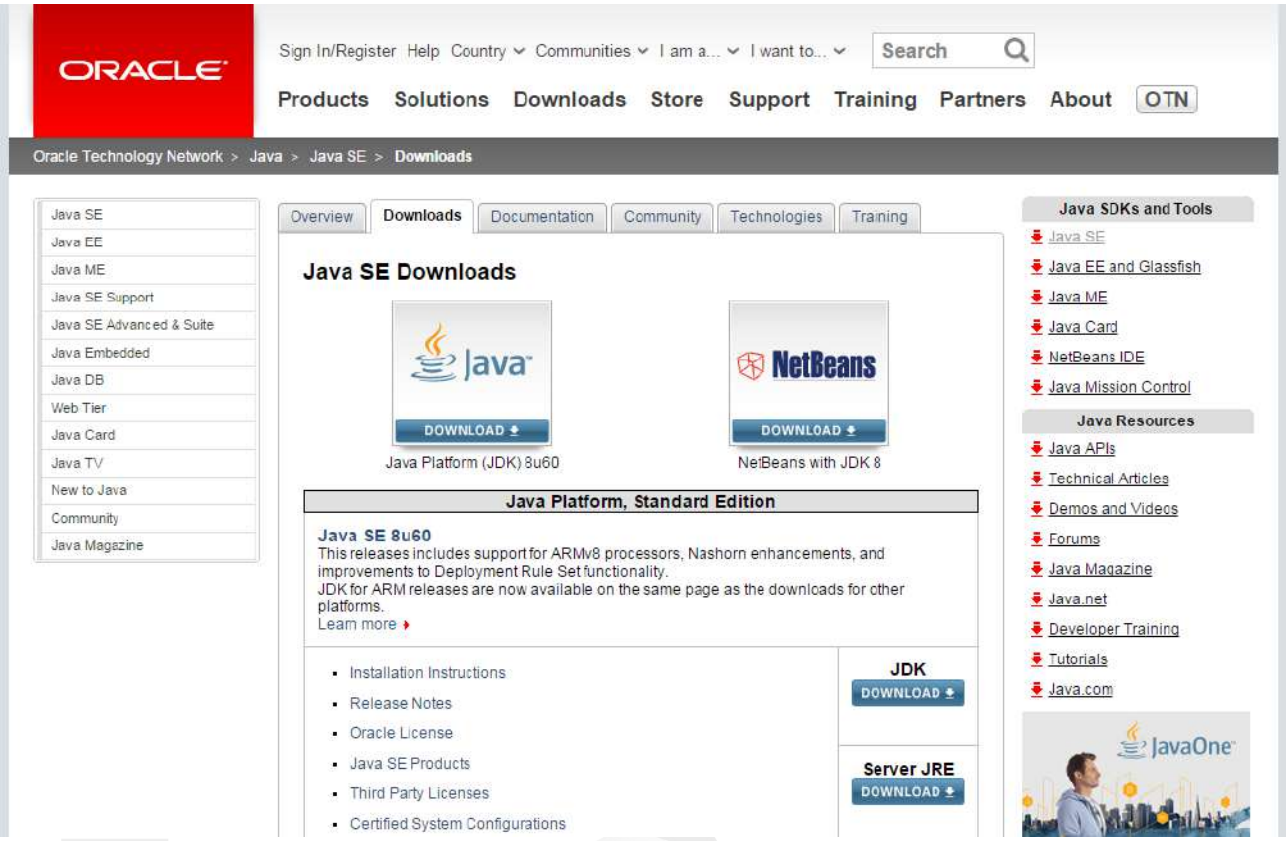
A escolha se deu pela popularidade e a experiência de uso. São muitos os aspectos que podemos utilizar para a escolha das ferramentas a empregar em um projeto de software. O envolvimento da comunidade de desenvolvedores em determinado projeto é sempre um ponto importante a ser utilizado. Esse é um dos pontos mais fortes de Java, porque além de instituições privadas que criam IDE's excelentes existe uma comunidade forte e atuante nos projetos criados para a plataforma.

O Eclipse e o Apache Tomcat são exemplos de ferramentas onde a comunidade é altamente atuante, o que vem contribuindo de maneira expressiva na evolução da IDE. Para hospedar seu projeto de software, onde pessoas de qualquer lugar do mundo possam acessar, você tem mais facilidade de encontrar servidores com serviços de hospedagem que tenha exatamente a configuração para utilização do Tomcat e o MySql.

1.2.1 Instalação do Java

O mais indicado é a utilização da versão mais recente da linguagem Java, no caso Java 5 em diante, pois trazem alterações, atualizações e vários ajustes de segurança. Na montagem deste material a versão mais atualizada é a Java SE 8u60. Podemos encontrar o arquivo para a instalação do Java e seus pacotes no endereço http://www.oracle.com/technetwork/java/javase/downloads/index.html,

abrindo diretamente a opção Java SE no menu **Downloads.**



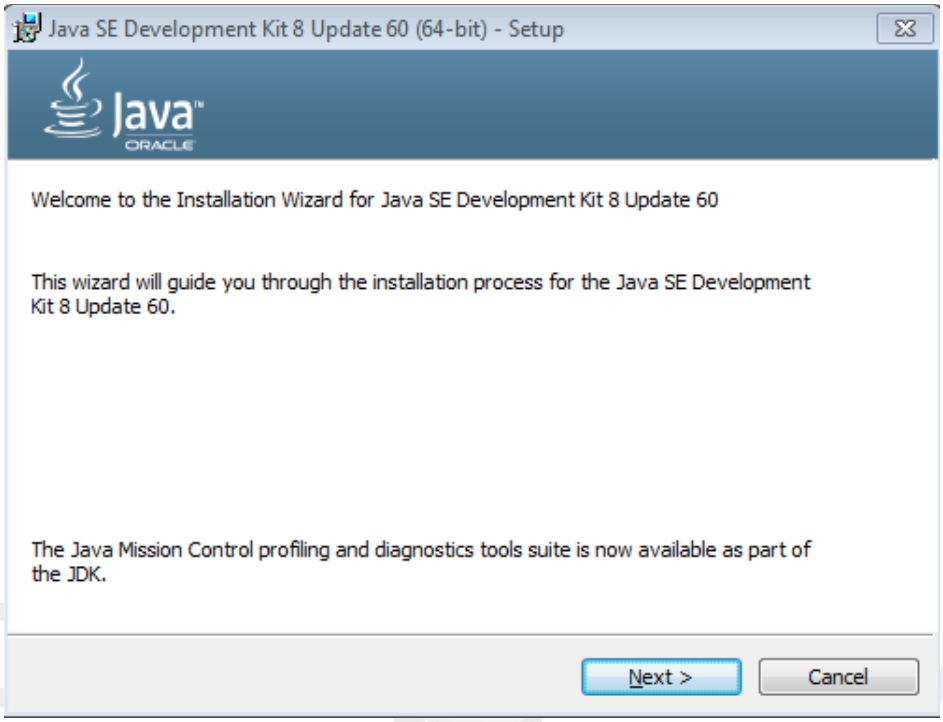
Opção de pacotes para instalação do Java.

1.2.1.1 Instalação

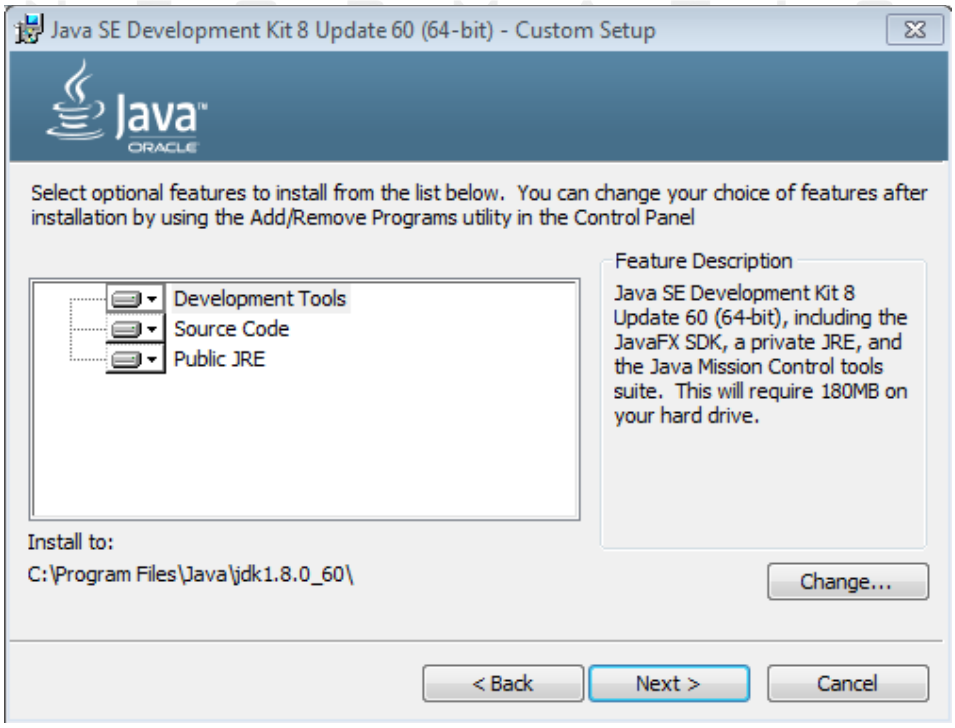
Serão listados todos os pacotes de instalação do Java, que são disponbilizados pela Oracle. As duas versões disponíveis são Java SE (ou JRE) e JDK. O pacote JRE (sigla de Java Runtime Environment) é o que chamamos de pacote mínimo e todo sistema operacional precisa dele instalado para a execução dos aplicativos desenvolvidos em Java. Já o pacote JDK (sigla de Java Development Kit) é utilizado pelos desenvolvedores de sistema e é necessário para que se consiga desenvolver algum aplicativo Java. O pacote JDK (que também inclui o JRE) já foi criado para que os desenvolvedores tenham uma IDE integrada, para que não seja necessário realizar muitas configurações e correr o risco de instalar coisas a mais ou no pior dos casos coisa de menos. Dessa forma, clique em Download. Na página seguinte, aceite os termos e selecione o ambiente (sistema operacional) de destino da instalação do pacote e a opção de idioma.

Continue o download conforme as instruções até a finalização do processo estar concluída por completo.

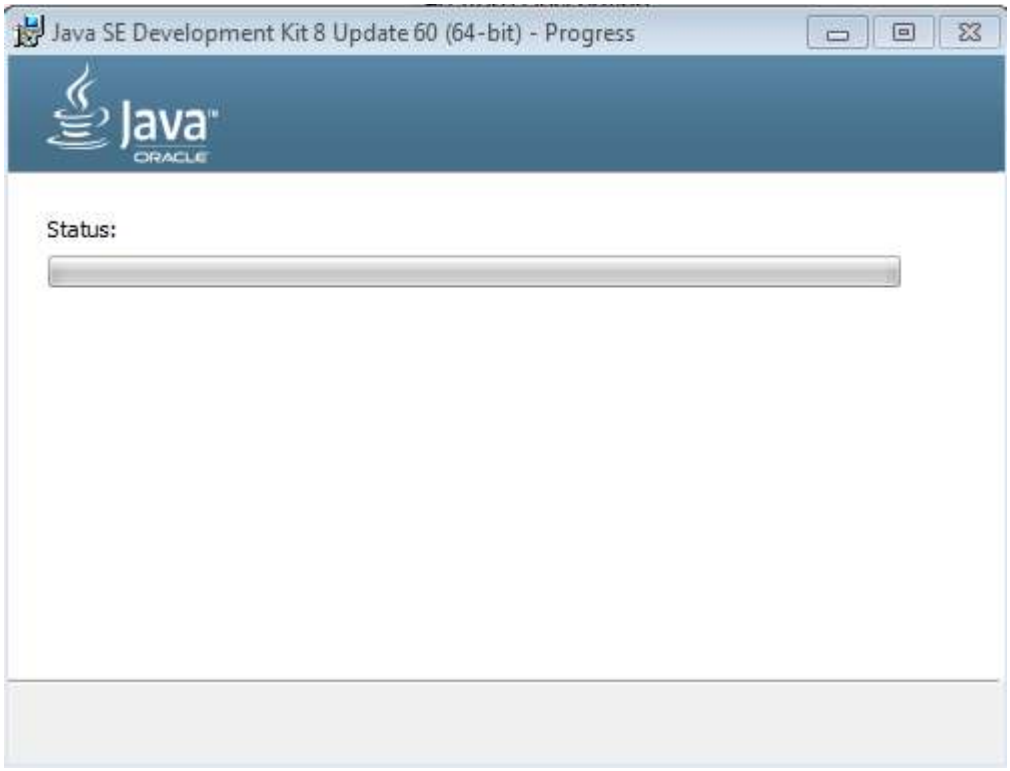
Depois de concluído o download, execute o arquivo baixado e aguarde a instalação. Siga os passos conforme indicações e a instalação estará concluída, conforme a próxima figura. Ao ser solicitada a instalação do JRE, aceite e siga todos os passos mantendo o padrão.(next, next, next...).



Iniciando a instalação do Java.



Itens do Java a serem instalados.



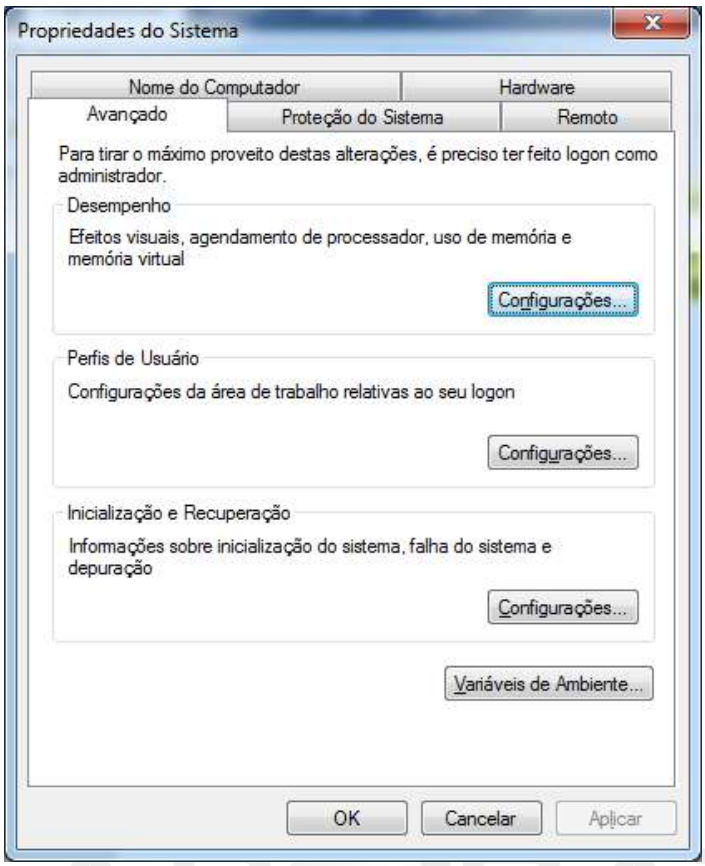
Levantamento das informações para a instalação do Java.

Escolha a pasta de destino para a instalação do Java.(recomendo default)

1.2.1.2 Configuração

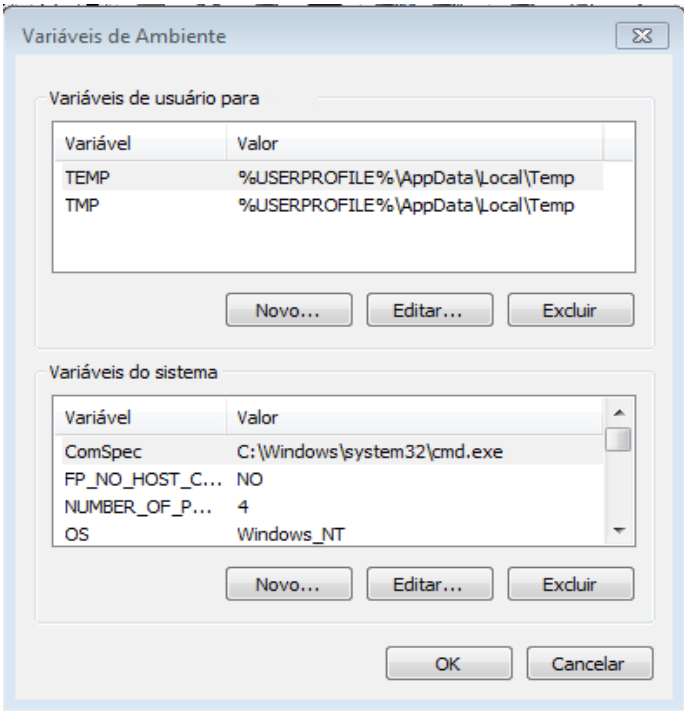
Mesmo que a instalação do Java tenha ocorrido sem erros ainda são necessários alguns procedimentos de configuração. Essa etapa deve ser realizada manualmente pois não é feita automaticamente. Esse procedimento não é obrigatório, pois utilizaremos a ferramenta Eclipse para fazer o desenvolvimento Java e ela já está configurada para executar o java. Porém, é interessante manter essa configuração para uma necessidade futura.

No Windows, clique em Iniciar → Painel de Controle → Sistema. Clique na guia Avançado e no botão Variáveis de Ambiente, conforme a figura



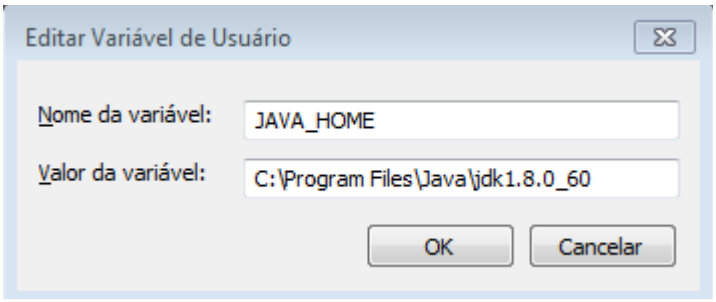
Tela de acesso às variáveis de ambiente.

A tela seguinte exibe as variáveis de ambiente do Windows. Agora vamos criar a variável que define o local onde foi instalado o Java, que é utilizada por muitos outros programas que precisam saber onde o Java se encontra, inclusive o Apache Tomcat.



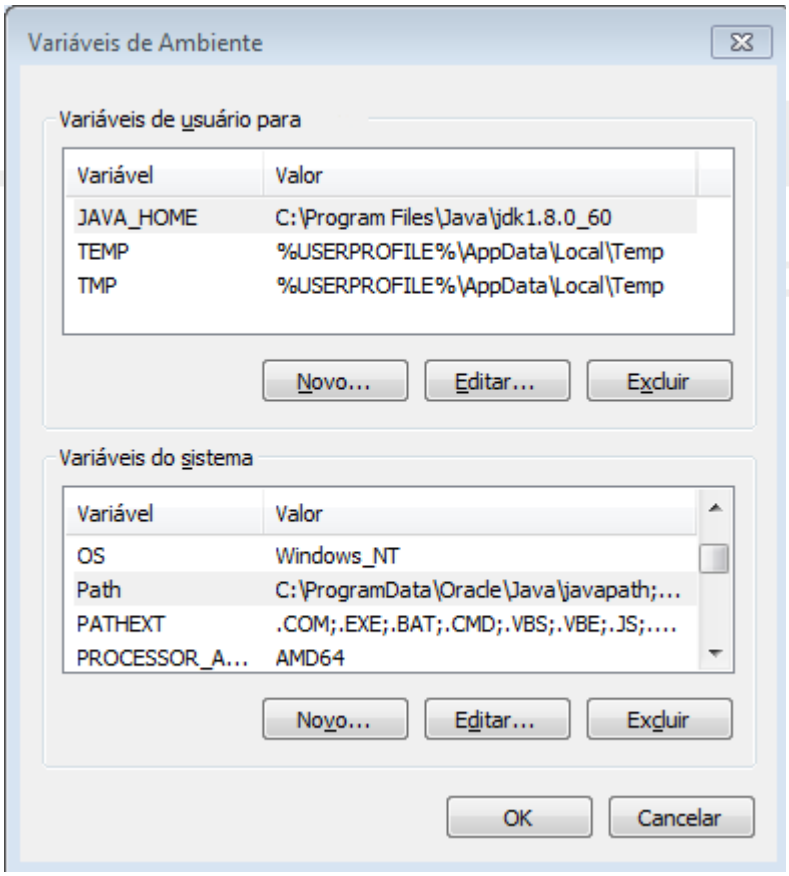
Tela para selecionar a variável do usuário.

Na região de Variáveis de Usuário, clique em Novo e preencha os campos com o nome da variável igual a **JAVA_HOME** e no valor da variável o caminho em que o Java foi instalado, **C:\Program Files\Java\jdk1.8.0_60** e depois pressionar o botão OK.



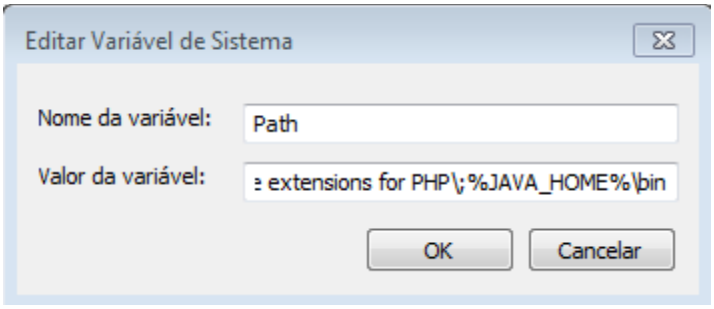
Editar variável de sistema

Agora vamos alterar a variável Path no quadro "Variáveis do Sistema" dentro do quadro Variáveis de Ambiente, que conterá o caminho para os programas executáveis do Java. Selecione a variável Path e clique em Editar.



Tela de variáveis do sistema.

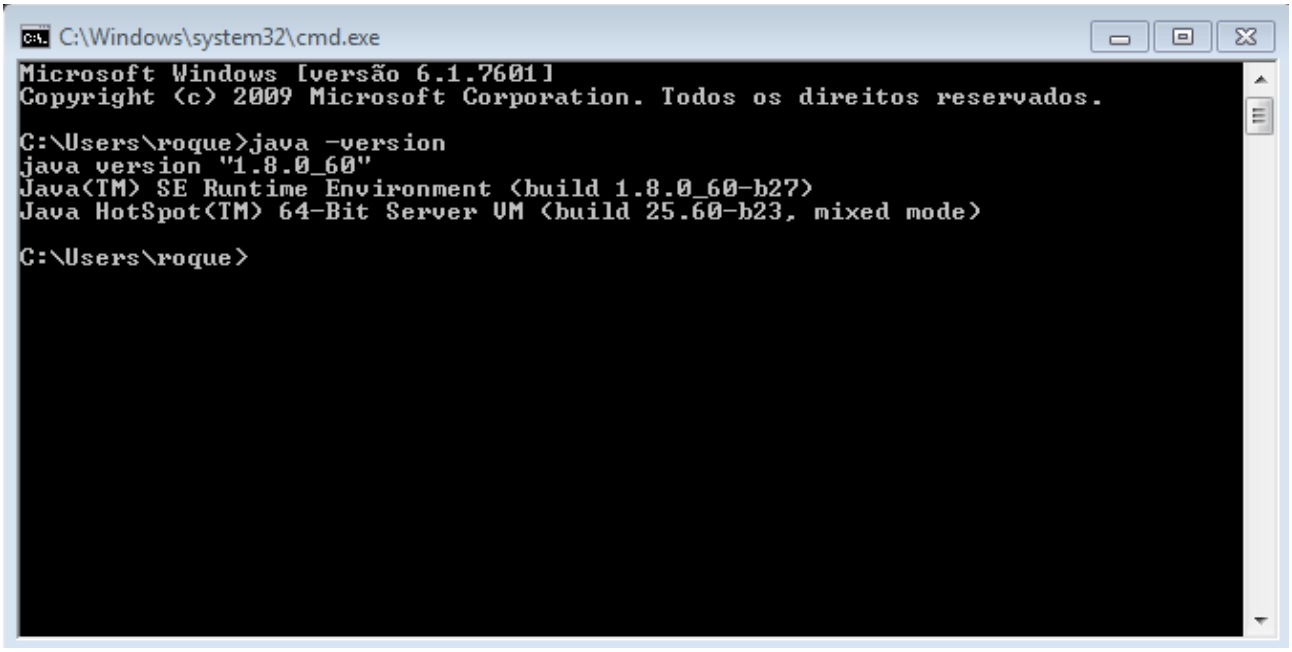
Vá até o campo Valor da Variável e adicione um ";" (ponto e vírgula) no final da linha (se não houver). Acrescente o texto %JAVA_HOME%\bin conforme mostra a figura a seguir.



Editar variável de sistema

	Nota: *A variável PATH serve para que os programas possam ser executados por linha de comando (DOS) em qualquer diretório, e não apenas naquele em que se encontra. Por isso é o caminho definido nessa variável.*

Para testarmos a instalação do Java, abra um Prompt de Comando (Iniciar ➤ Programas ➤ Acessórios ➤ Prompt de Comando) e digite java –version. Esse comando exibirá a versão atual instalada do Java, conforme a figura abaixo, indicando o sucesso da instalação.



Verificar se variável de ambiente está configurada corretamente.

1.2.2 Instalação do Apache Tomcat

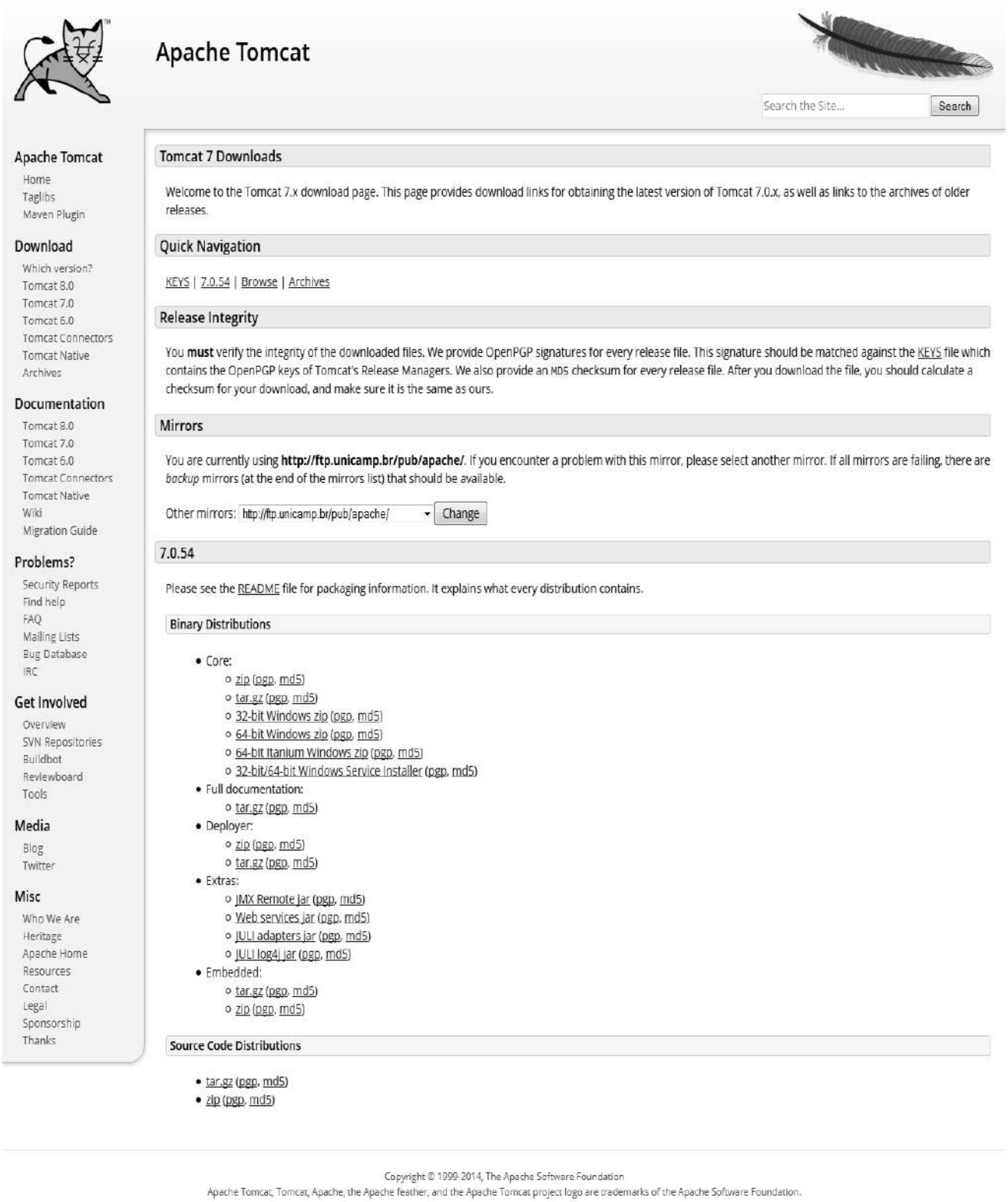
O Apache Tomcat é um servidor Java web e também um dos mais utilizados para pequenos projetos. Suporta as tecnologias Java Servlet e Java Server Pages (JSP), permitindo que o Java funcione num ambiente web, o que não encarece sua contratação. Podemos dizer que o Apache Tomcat é um contêiner de Servlets.

Ele não é um servidor de EJBs mesmo sendo um servidor de aplicações JEE. Ele implementa tecnologias menores, o termo definido pelo JEE6 é “container server web

profile", e representa softwares que representam somente parte do JEE. Desenvolvido pela Apache Software Foundation, é distribuído como Software Livre dentro do Apache Jakarta.

Para realizar a instalação do Apache Tomcat é necessário acessar o site http://tomcat.apache.org/. Acesse o link Tomcat 7.x no menu Downloads e você será direcionado para a página de download. A página mostrará sempre a última versão liberada para download do Apache Tomcat.

Faremos o download do pacote em Binary Distributions\Core, conforme a figura na próxima página. Escolha o pacote ZIP ou service installer para Windows e o TAR.GZ para Linux.



Obtenção do pacote de instalação do Apache Tomcat.

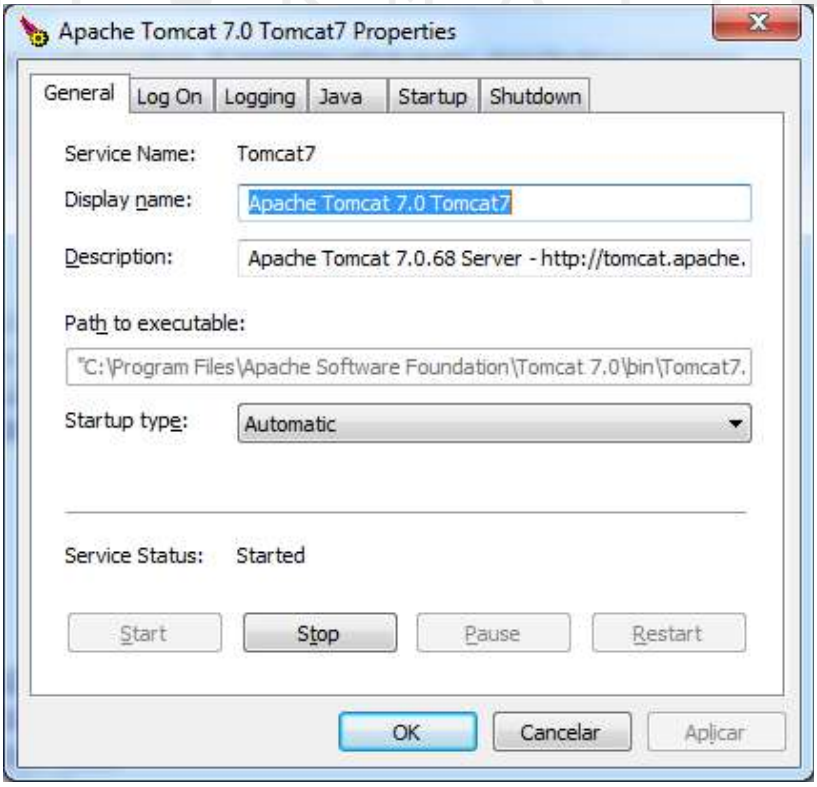
1.2.2.1 Instalação

Depois de descompactar o arquivo no disco de preferência no drive c:\, a estrutura de pasta deve se parecer com a da figura a seguir.

Estrutura de pastas da instalação do Apache Tomcat.

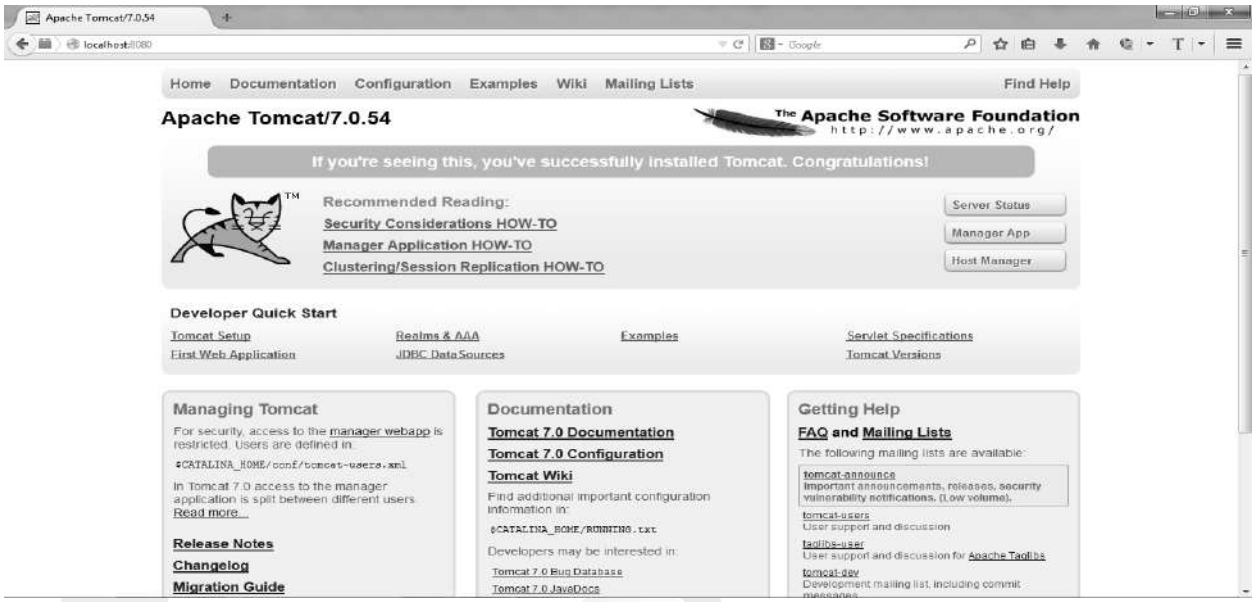
Na instalação do Apache Tomcat, é aconselhável definir uma nova variável de ambiente se a instalação for feita pelo arquivo zip. Isso é necessário para trabalhar com o Apache Tomcat seja iniciado sem a necessidade de abrir o eclipse. Para isso, crie a variável CATALINA_HOME tendo como valor o caminho de instalação do Tomcat, no caso, C:\apache-tomcat-7.0 conforme figura acima.

Agora que a variável está configurada, você pode fazer um teste para se certificar de que a instalação ocorreu normalmente. Entre na pasta bin do diretório de instalação do Tomcat e execute o arquivo Tomcat7w.exe: o resultado da execução deverá se parecer com o da figura abaixo.



Console do Apache Tomcat, depois de ser inicializado com sucesso.

Após iniciar o serviço pressionando o botão start, caso não esteja acionado, na figura da página anterior, abra o navegador de sua preferência e digite http://localhost:8080. Deverá aparecer uma página com a logo do Apache Tomcat como a da figura a seguir.



Página inicial padrão do Apache Tomcat.

Para desligar o servidor, pressione o botão stop da figura da página anterior, isso fará com que o servidor tomcat pare de responder.

1.2.3 Instalação do Eclipse IDE

Teoricamente podemos desenvolver qualquer programa em Java usando apenas um editor de texto simples, como por exemplo o bloco de notas(Windows) ou até mesmo o notepad++. Neste caso precisamos realizar a compilação do projeto de forma descentralizada, o que deixaria o nosso processo de implementação muito mais demorado e exigindo muito mais experiência técnica para a montagem, compilação, e disponibilização do projeto no ambiente de servidor de aplicação.

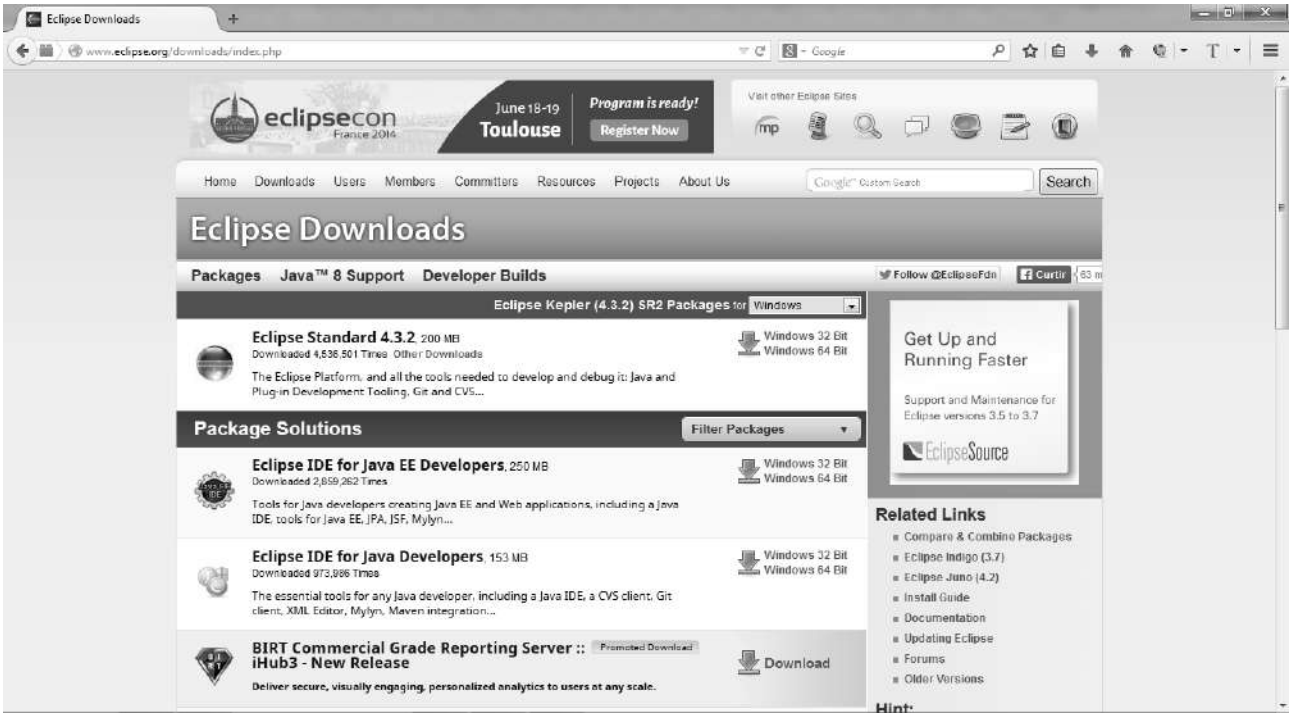
Por isso, a exemplo de outras linguagens, necessitamos utilizar um ambiente integrado de desenvolvimento que é popularmente conhecido como IDE(integrated Development Environment).

Tem como objetivo aumentar a produtividade no desenvolvimento de aplicações. São bem mais complexas que editores de textos e apresenta várias ferramentas embarcadas.

JEE

O Eclipse é uma plataforma de desenvolvimento de aplicações que é composta por uma IDE e um conjunto de plugins. É praticamente escrita em Java e usada para desenvolver aplicações nesta linguagem e por meio dos diversos plugins existentes, é capaz de desenvolver para C e C++, Cobol, PHP, Python entre outras.

Então temos que baixa e instalar o Eclipse. Para isso acesse o site http://www.eclipse.org/downloads/index.php onde será exibida uma página como a da figura da próxima página.



Tela da sessão de downloads do site do Eclipse.

Como já apresentado na instalação do JDK, precisamos escolher o Sistema Operacional adequado para baixa e montar a nossa plataforma de desenvolvimento, então devemos baixar a versão “Eclipse IDE for Java EE Developers”.

1.2.3.1 Instalação

O Eclipse tem um instalador, que faz todo o processo automaticamente e configurando todo o processo sem a necessidade de intervenção do desenvolvedor.

Temos também a versão que você baixa e só descompacta o arquivo no local que desejar. A descompactação pode ser em sua unidade C:\, conforme a figura.

Caso a instalação tenha sido pelo arquivo zip, você precisa criar um atalho para esse arquivo e posicioná-lo em sua área de trabalho do Windows.

Na primeira execução do Eclipse, aparecerá uma tela para a definição do workspace que você deseja trabalhar. O workspace é a localização física dos arquivos e diretórios no qual serão criados todos os projetos e onde ficará também todas as configurações efetuadas dentro do Eclipse. Podemos definir o workspace através do menu “File → Switch Workspace → Other...”

O eclipse permite trabalhar com vários workspaces para uma só versão do eclipse. Você precisa selecionar qual o workspace será usado para o desenvolvimento da aplicação.

1.2.3.2 Teclas de atalho

Existe no eclipse uma gama muito grande de teclas de atalho que devem ser utilizadas com frequência para aumentar a produtividade no desenvolvimento da aplicação. Abaixo algumas das teclas mais utilizadas pelos desenvolvedores.

Tecla de atalho	Descrição
Ctrl+Espaço	Mostra uma lista de classes ou métodos que são sugestões disponíveis que podem ser utilizadas. Isso ajuda a não implementar duas vezes a mesma coisa.
Ctrl+K	Ao selecionar uma palavra e pressionar esse conjunto de teclas, o cursor será posicionado na próxima ocorrência desta palavra percorrendo todo o texto.
Ctrl+Shift+T	Se precisar localizar uma classe mais rapidamente, pode usar essa combinação que pesquisará somente classes Java dentro do seu projeto.
Ctrl+Shift+R	Para essa combinação são trazidos além das classes Java, todos os outros arquivos existentes no projeto
Ctrl+O	Algumas classes Java contém uma quantidade muito grande de atributos e métodos. Essa combinação permite o agrupamento e assim a localização mais rápida casa necessário.
Ctrl+Shift+O	Uma das combinações de teclas mais utilizadas no Eclipse, serve para organizar os imports do projeto. Se estiver faltando algum import ou existir um import que não está sendo utilizado este comando resolve de forma rápida e segura.
Ctrl+I	Comando serve para a identar o código e ressaltar a estrutura do algoritmo.
Ctrl+Shift+F	Parecido com o comando Ctrl+I, mas realiza uma formatação onde o código alem de sofrer sua identação, também quebra textos muito extensos para que fiquem

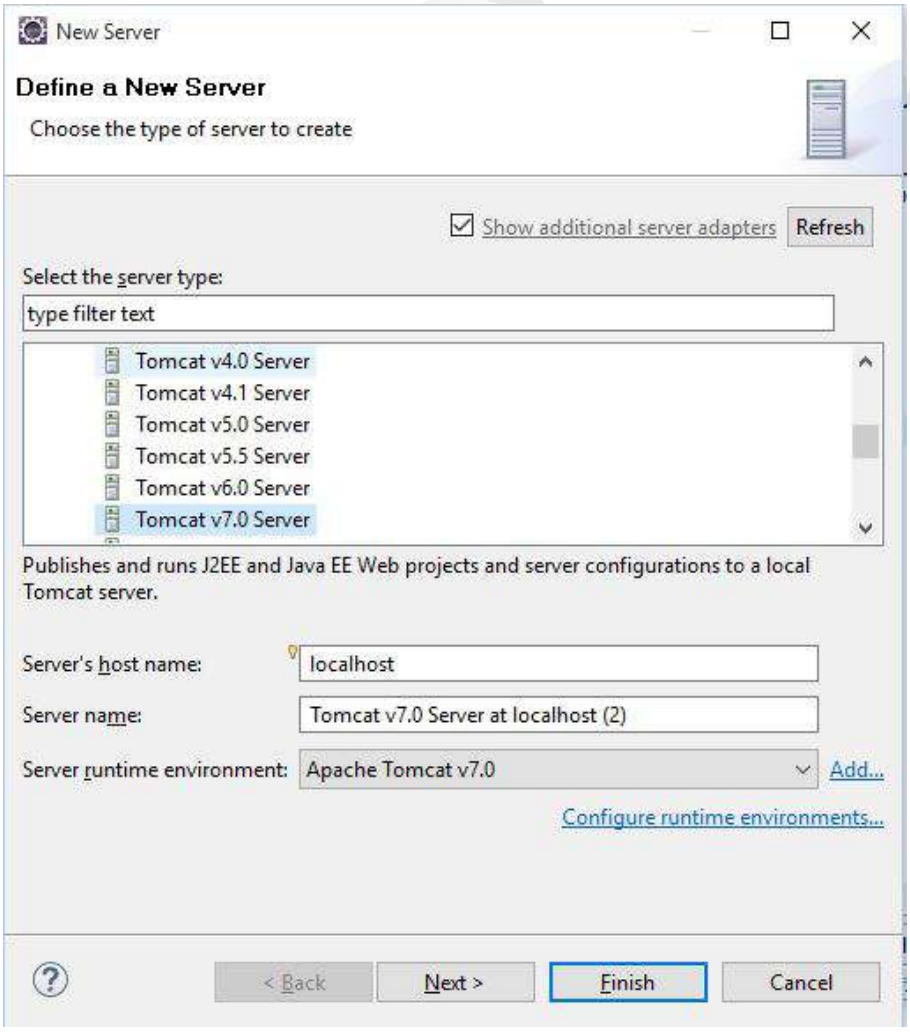
	dentro do campo de visão do desenvolvedor.
Ctrl+/	Adiciona um comentário de linha, importante para adicionar comentários no código.
Ctrl+Shift+/	Adiciona um comentário de bloco, que permite comentar uma série de linhas. Eficiente para comentar métodos inteiros, sem a necessidade de ir linha a linha realizando os comentários.
Alt+Shift+R	Se precisa trocar o nome de uma variável por exemplo e quer que isso reflita em todo o projeto, você deve usar essa combinação. Isso faz com que você altere o nome em todos os arquivos que tem essa referência.
Ctrl+Shift+L	Utilizando essa combinação de teclas, o eclipse mostrará uma caixa com todas as teclas de atalho existente.

1.2.4 Configurando o Apache Tomcat dentro do Eclipse

Agora que já temos instalados e configurados o Apache Tomcat e o Eclipse IDE de forma individual, vamos agora ver como integrar os dois, isto é, para que Tomcat possa ser executado a partir do Eclipse. Isso permitirá que uma classe ou JSP que já é executada por padrão no servidor também possa ser executada por dentro do ambiente interno da IDE Eclipse que também está pré-configurado.

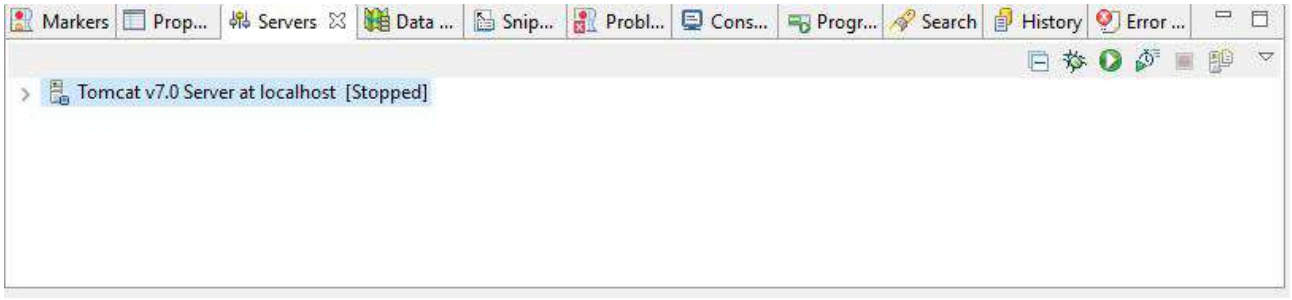
O primeiro passo que teremos que executar é ir até a view Servers, pressionar o botão direito do mouse e escolher New → Server. Isso dará início ao assistente de criação de servidores que está disponível dentro do Eclipse. Selecione Tomcat v7.0 Server e clique em Next, conforme a figura da próxima página.

Na tela seguinte, informe o local em que foi instalado ou descompactado o Apache Tomcat, e pressione Finish. O caminho definido nessa última tela será o responsável para que sua aplicação rode a partir do serviço do Apache Tomcat e é o mesmo que já foi definido para a variável CATALINA_HOME.



Seleção do servidor a ser criado.

Pronto: agora o servidor que executará nossos projetos web já está criado e pronto para ser utilizado. Nele colocaremos os projetos que serão criados no decorrer do aprendizado.



View Server do Eclipse exibindo o servidor recém-criado.

CAPÍTULO 2

JEE – JAVA ENTERPRISE EDITION

JEE

Plataforma de programação Java para internet. Amplamente utilizada e com vários casos de sucesso.

2.1 Características

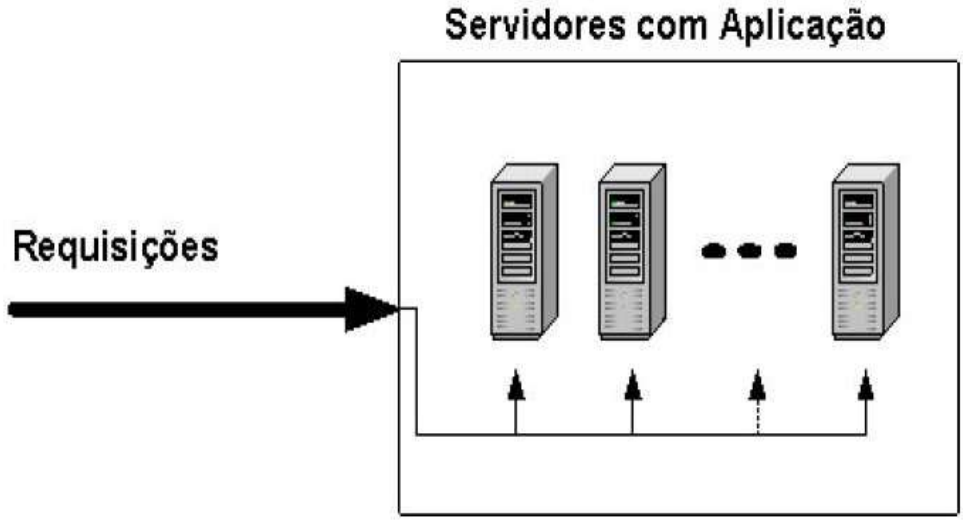
Java Plataform, Enterprise Edition, é uma plataforma de programação para servidores na linguagem de programação Java. A plataforma exibe uma API e um ambiente de execução para desenvolvimento de softwares corporativos, incluindo serviços web, e outras aplicações de rede, multicamadas, escaláveis, confiáveis e seguras.

Esta plataforma oferece diversas vantagens em relação ao uso de outras tecnologias (como PHP, ASP e CGI). As principais vantagens são herdadas da própria linguagem Java:

- **Portabilidade**: a aplicação desenvolvida roda em diversas plataformas, como por exemplo Windows, Unix e Macintosh, sem que haja a necessidade de criar um sistema para cada plataforma ou mesmo ter que recompilar novamente a aplicação para rodar em determinada plataforma.
- **Facilidade de programação**: a programação é orientada a objetos, o que facilita a criação e manutenção no desenvolvimento de sistemas complexos. Além disso, a linguagem oferece outras facilidades, como por exemplo o gerenciamento automático de memória (estruturas alocadas são automaticamente liberadas, sem a intervenção do desenvolvedor).
- **Flexibilidade**: o Java já se é bastante conhecido, e tem uma enorme comunidade de desenvolvedores, ampla documentação e diversas bibliotecas e códigos prontos, dos quais o desenvolvedor pode usufruir.

A arquitetura de Servlets e páginas JSP possibilita alguns benefícios adicionais:

- **Escalabilidade**: em grande parte dos servidores de aplicações modernos, é possível distribuir a carga de processamento de aplicações desenvolvidas em diversos servidores, sendo possível adicionar ou remover de maneira a acompanhar o aumento ou decréscimo dessa carga de processamento.



Servidor de aplicação.

- **Eficiência**: os Servlets carregados por um servidor persistem em sua memória até que ele seja finalizado. Assim, ao contrário de outras tecnologias, não são iniciados novos processos para atender cada requisição recebida; por outro lado, uma mesma estrutura alocada em memória pode ser utilizada no atendimento das diversas requisições que chegam a esse mesmo Servlet.
- **Recompilação automática**: páginas JSP modificadas podem ser automaticamente recompiladas, de maneira que passem a incorporar imediatamente as alterações sem que seja necessário interromper o funcionamento da aplicação como um todo.

O Java EE foi criado para programação em containers, responsáveis por fornecer todos os serviços necessários para as aplicações corporativas.

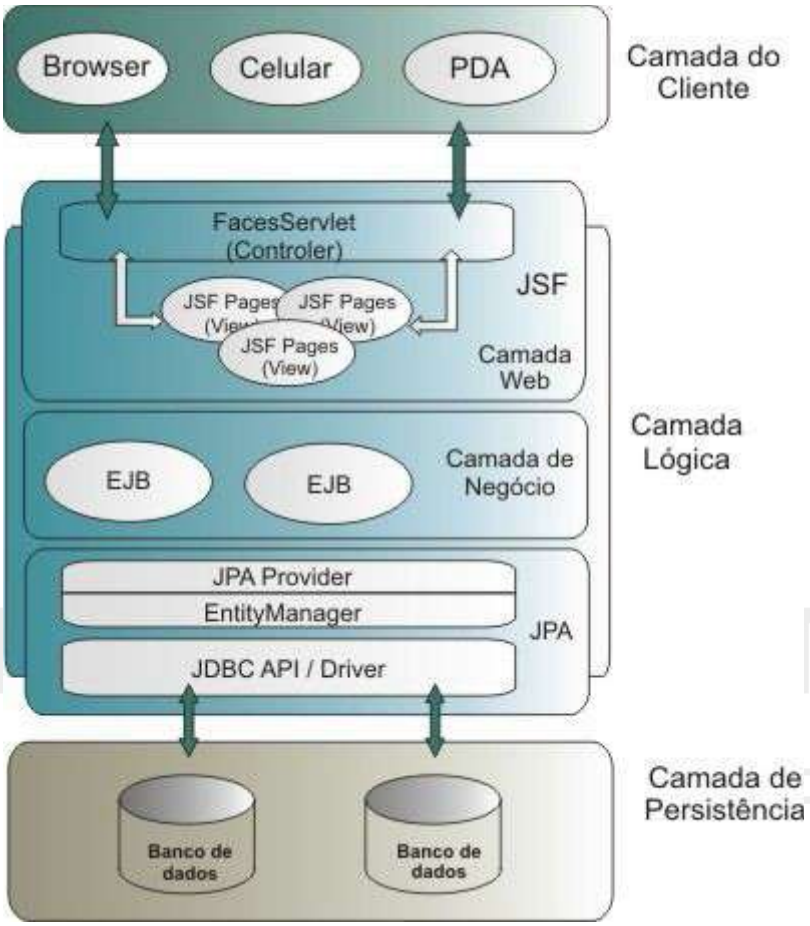
Sendo assim o programador escreve menos código, diminui o tempo com o desenvolvimento, os riscos do projeto e os problemas de manutenção e deixa que a plataforma de serviços e APIs já existente faça todo o trabalho pesado.

Quando padronizamos a plataforma de desenvolvimento, conseguimos evitar muitos problemas, referente a migração de fornecedores.

Podemos dizer que Java Entrerprise Edition é formada por um conjunto de especificações publicadas pela Oracle e pela Java community Process(JSRs), responsáveis por criar a infraestrutura que gira em torno do desenvolvimento das aplicações distribuídas corporativas.

As aplicações do modelo Java EE são vistas como um modelo de camada com separação de conceitos(tiers).

A Figura mostra a arquitetura da plataforma JEE dividida em 3 partes lógicas: camada do cliente, camada lógica e camada de persistência.



Exemplo de arquitetura distribuída com Servlets e Páginas JSP.

2.2 As tecnologias do JavaEE

A plataforma JavaEE é bastante completa e vem com vários serviços implementados pelas diversas tecnologias incorporadas ao framework. Vamos ver as principais na tabela abaixo.

Java Servlet 3.0(JSR 315)	Permite a criação de classes servidoras que podem ser instaladas em Containers Java EE
Java Server Faces 2.0(JSR 314)	Modelo de desenvolvimento de páginas RIA(Rich Internet Application)
Java Server Pages 2.2(JSR 245)	Permite a criação de páginas dinâmicas de

	maneira fácil e prática.
Entrerprise JavaBeans 3.1(JSR 318)	Criação de componentes de negócio remotos.
Java Persistence 2.0(JSR 317)	Abstração objeto-realcional para acesso a bancos de dados.
Java API for XML-Based Web Services(JAX-WS) 2.2(JSR 224)	Extensão do JAX-RPC para criação de Web Services.
Java API for RESTful Web Services(JAX-RS) 1.1(JSR 311)	Implementação da arquitetura REST em Web Services.

O Java EE na versão 6.0 contem duas das novidades citadas acima e outras duas que são:

- **Java API for RESTful Web Services(JAX-RS) 1.1(JSR 311):** para a criação de Web Services utilizando a arquitetura REST(Representational State Transfer);
- **Java Server Faces 2.0(JSR 314) incorporando o Facelets:** aumenta a produtividade do desenvolvimento Web utilizando páginas em formato XHTML, compatível com W3C;
- **Contexts and Dependency Injection for Java(Web Beans 1.0) (JSR 299):** desacopla a aplicação, permitindo que o container incorpore os recursos necessários. Nesse caso a aplicação lida somente com as abstrações;
- **Bean Validation 1.0(JSR 303):** permite a criação de metadados de validação em beans de entidade.

2.3 A evolução do desenvolvimento para web

No início dos anos 90 estava ocorrendo o surgimento da internet no Brasil. Naquela época a utilização da web era um luxo. Os usuários tinham o habito de desabilitar as imagens, de modo a diminuir o tempo de resposta dos modens da época.

Os sites eram praticamente coleções de textos e imagens, com interações diminuídas, mas não chegavam a ser monótonas pois a novidade tomava toda a atenção. O mais complexo em termos de usabilidade era o preenchimento de um form-to-mail, que enviava as informações inseridas por e-mail para o webmaster.

JEE

Com o passar do tempo surgiram os sites com scripts CGI, feitos em Perl ou C e a interação era feita baseada em formulários. Foi uma época de inovação e porque não dizer, de heroísmo tornando fácil fazer sucesso com a internet.

Porém, hoje tudo mudou. As tecnologias avançaram e fizeram que os usuários ficassem muito mais exigentes e tornando a vida do programador web muito mais difícil e complicada.

O HTML ainda era muito básico. Para se ter uma ideia somente em 1996 foi lançado o HTML 2.0 que dava suporte a tabelas. O lançamento do JavaScript, na versão 2 do Netscape Navigator em 1995, causou uma revolução na criação de páginas web e foi nesta versão que houve o lançamento do suporte a Applets Java possibilitando gerar páginas bem mais funcionais.

No final da década de 90, muitas tecnologias estavam surgindo com o intuito de competir com os sites baseados em HTML. Abaixo vamos listar algumas delas:

- **NSAPI –** Nestscape Applications Program Interface – Um pouco mais avançado que o CGI e que consome menos recursos, pois rodava dentro do processo do servidor web.
- Borland **Delphi** NSAPI/ISAPI – Uma versão do Delphi para a criação de aplicações WEB.
- **ASP –** Active Server Pages – Linguagem de páginas dinâmicas, criada pela Microsoft e lançada juntamente, com o Internet Information Server 3.0.
- **ISAPI –** Internet Information Server API o IIS na versão Microsoft.
- **PHP –** É uma linguagem interpretada para a criação de páginas dinâmicas.]
- **Java Servlet –** Serve para criar componentes servidores em Java.

CAPÍTULO 3

SERVLETS

3.1 Características

Servlets são, nada mais nada menos que classes Java que trabalham em parceria com a Web. Todo o conteúdo visto nos cursos Java básico e Java avançado pode ser aplicado aqui. Lembrando que uma Servlet não tem a classe main, necessária para iniciar um programa simples em Java, pois está sob o controle de outra aplicação Java, chamada de container como por exemplo o Apache Tomcat.

No mundo Java a primeira e também principal tecnologia a gerar páginas dinâmicas são as Servlets, que surgiu em 1997. Utilizamos Servlet quando necessitamos abrir um programa em qualquer navegador web utilizando a linguagem Java.

Apesar de os Servlets poderem responder a quaisquer tipos de requisições, eles geralmente operam em aplicações hospedadas por servidores web, assim eles podem ser imaginados como Applets Java que rodam em servidores em vez de rodarem nos navegadores web. Estes tipos de Servlets são os equivalentes Java a outras tecnologias de conteúdo Web dinâmico, sendo os mais conhecidos o PHP e ASP.NET.

Também pode ser definido como um componente semelhante a um servidor, que gera dados HTML e XML para a camada de apresentação de uma aplicação Web sem a necessidade de um arquivo html. Ele processa dinamicamente requisições e respostas.

Quando uma aplicação web recebe uma chamada para um Servlet o servidor entrega a solicitação ao container ao qual o Servlet é distribuído. O container entrega a solicitação e a resposta HTTP e é responsável também pela pela chamada dos métodos doGet() e doPost().

3.1.1 Container

O container é a representação do gerenciador de requisições da aplicação, representada aqui pelo Apache Tomcat(Nosso container).

A requisição é enviada para o container que após receber, identifica pela URL, a classe que representa o Servlet, carrega o objeto na memória e o método init() é chamado. Com o Servlet instanciado e inicializado, o container gera um objeto HttpServletRequest e um HttpServletResponse e posteriormente chama o método service() do Servlet, passando os objetos como parâmetro.

Ao fim do processo, quando o container finaliza, ou se por ocasião a quantidade de memória não for suficiente, o Servlet é destruído e para que isso aconteça o cantainer invoca o método destroy(). Parecido com o garbage collector o container percebe quando um Servlet está há um determinado tempo sem receber requisições e nesse caso também pode chamar o método destroy(), para liberar recursos.

Oferece uma maneira bastante simples para a comunicação das Servlets com o servidor de aplicação, isso fornece suporte a comunicação evitando a utilização de um ServerSocket para ter que ficar escutando alguma porta. O container conhece o protocolo entre o servidor e ele mesmo tirando a responsabilidade do Servlet de controlar essa comunicação, tendo que se preocupar somente com a lógica de negócio que está contida em seu Servlet.

O container tem suporte a multithreads e cria automaticamente uma nova thread em Java para cada solicitação Servlet recebida. Ele também pode usar XML para realizar toda a configuração para a segurança, deixando de escrever qualquer linha de código na classe Servlet, além de ter suporte a JSP sendo o responsável por traduzir o código JSP em Java.

A especificação do Java EE é implementada em container(servidor) e também em servidores de aplicação. Existem diversos containers e servidores de aplicação que podemos utilizar para o desenvolvimento Java para a web. Abaixo segue uma lista de alguns containers e servidores de aplicação que podemos utilizar:

- **Tomcat * (Apache)** – Implementa Servlet e JSP e não é um container EJB.
- **Jetty * (Eclipse Foundation)** – É um Servlet container 100% escrito em Java e o maior concorrente do Tomcat.
- **JonAS (Object Web)** – É uma implementação open-source para JEE desenvolvido pelo Consórcio europeu ObjectWeb, sem fins lucrativos.
- **JBoss AS (JBoss - Red Hat)** – Baseado na plataforma JEE e implementado 100% em JAVA é hoje um dos mais utilizados.
- **Geronimo (Apache)** – Está entre um dos melhores projetos de código aberto. A distribuição é implementada em JEE 6.

- **TomEE (Apache)** – É uma abreviação de Tomcat + Java EE e tem uma abordagem diferente do tomcat, sendo que incorpora as tecnologias EJB, CGI entre outras o que resulta num Servidor Web Profile.
- **Resin (Caucho Technology)** – É um servidor de aplicações Java para ambientes corporativos e de produção e suporta JEE e PHP.
- **Blazix (Desiderata Software)** – É um servidor de aplicações Java completo e de alto desempenho. Pode ser usado somente como servidor de aplicações ou como servidor Web completo.
- **Enhydra Server (Enhydra.org)** – É um servidor independente que foi construido sobre a servidor JOnAS.
- **GlassFish (Oracle)** – Servidor de código aberto compatível com JEE e desenvolvido pela gigante Oracle.
- **WebLogic (Oracle)** – Servidor para ambientes de nuvem e convencionais, se destaca por simplificar as operações de gerenciamento de nuvem nativo.
- **WebSphere (IBM)** – Muito utilizado para programação orientada a serviços utilizando SOA, permite processos dinâmicos e toda a estrutura para as situações de negócios.

3.1.2 Servlet e web.xml

Podemos ressaltar que o arquivo web.xml pode declarar vários Servlets. A tag <servlet-name> é o nome conhecido pelo container, contem o nome da Servlet que servirá para realizar o preenchimento da tag <servlet-mapping>, a tag <servlet-class> serve para declarar a classe Java que representa a Servlet e a última tag do exemplo é <url-pattern> onde o conteúdo é o que o cliente utiliza na solicitação.

Exemplo:

Estrutura de um Servlet – Padrão

```
PadraoServletjava


import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
```

```
/* 99% de todos os servlets sai HttpServlet */
public class PadraoServlet extends HttpServlet{
    /* É aqui que seu Servlet consegue as referências dos objetos solicitados e a resposta
    No mundo real 99% de todos os Servlet anulam ou o método doGet() ou o doPost()
     */
    public void doGet(HttpServletRequest request,
           HttpServletResponse response) throws IOException,
           ServletException{
     /* Utilize PrintWriter para escrever texto html no objeto de resposta*/
     PrintWriter out = response.getWriter();
     out.println("Hello World");
    }
}
```

Exemplo de um XML para um Servlet

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.5"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
    <welcome-file-list>
     <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
    <!--chamando um Servlet-->
    <servlet>
     <servlet-name>servletteste</servlet-name>
     <servlet-class>PadraoServlet</servlet-class>
```

```
    </servlet>
    <servlet-mapping>
     <servlet-name>servletteste</servlet-name>
     <url-pattern>/teste.do</url-pattern>
    </servlet-mapping>
</web-app>
```

Trabalhar com Servlet permite relacionar mapeamentos entre classes Java, arquivos html e xml, que proporcionam a integração entre páginas de internet de maneira rápida e fácil. Essa configuração permite flexibilidade na segurança da aplicação e mapeando o nome em vez de colocar o verdadeiro caminho garantem flexibilidade ao movimentar as coisas e também evita aquela complicação quando for necessário realizar alterações do código que aponta para a antiga localização dos Servlets.

Em relação a segurança, como a internet é livre e contém todos os tipos de usuários, existem aqueles que sempre tem um motivo para prejudicar alguém e vai escolher sempre o caminho mais fácil. Se queremos manter a privacidade, nada melhor que ocultar informações ou despistar de alguma forma esses maus elementos. Passar um caminho falso é zelar pela segurança.

Exercícios Servlet:

1 – Criar um DynamicWebProject nomeando como Exercicio1, configure o arquivo web.xml, nomeando e configurando a Servlet corretamente. Crie a classe Java que chamará MinhaServlet configurando o que a Servlet executará quando for chamada.

2 – Criar um DynamicWebProject nomeando como Exercicio2, configure o arquivo web.xml, nomeando e configurando a Servlet corretamente. Crie a classe Java configurando a Servlet para receber uma condição que se for verdadeira retorne uma página jsp chamada de pagina1.jsp, senão uma página jsp chamada pagina2.jsp.

3 – Criar um DynamicWebProject nomeando como Exercicio3, composto por um formulário de cadastro de contatos, onde após o envio do cadastro, seja mostrado uma página com os contatos gravados. Após a montagem do projeto deve ser feita a reorganização dos imports.

3.2 JSP – JavaServer Pages

Em 2001 ocorreu o estouro da “bolha” da internet e a partir daí tudo mudou na Web. Quem já não aparecia muito deixou definitivamente de aparecer e os grandes vieram para ficar. O evento foi chamado de fim da “nova economia” causando também impacto direto na Web.

Os web sites passaram a ser aplicações corporativas gerando a necessidade de utilização de tecnologias mais robustas e eficientes. Foi aí que a plataforma Java se firmou como a mais promissora das linguagens, pois conseguíamos uma combinação entre JSP e Servlets o qual gerou o framework Struts e diversos outros frameworks, para o melhor uso e gerenciamento da linguagem Java para Web facilitando assim a vida dos programadores pois os frameworks buscavam aumentar a produtividade.

As JSPs são páginas web, em linguagem fonte, que são processadas pelo WebContainer, que é responsável por transformar uma jsp em uma Servlet. Quando chamamos a JSP estamos invocando o Servlet gerado a partir dela.

Podemos considerar a JSP, como um tipo de linguagem de criação de script no servidor, embora, como você verá mais adiante, ela opera de maneira bem diferente por detrás dos panos. JavaServer Pages são arquivos textos, normalmente com a extensão ".jsp"(mas é normalmente possível configurar o container JSP para reconhecer extensões de arquivos adicionais ou alternativas.), que substituem as páginas HTML tradicionais. Os arquivos JSP contêm HTML junto com o código embutido que permitem que o designer de páginas acessem dados do código Java rodando no servidor. Quando a página é solicitada por um usuário e processada pelo servidor HTTP (HyperText Transport Protocol), a parte HTML da página é então transmitida. No entanto, as partes de código das páginas são executadas no momento em que a solicitação é recebida e o conteúdo dinâmico gerado por este código é unido na página, antes de ser enviado para o usuário. Isto propicia uma separação dos aspectos de apresentação HTML da página, da lógica de programação contida no código.

JSP é baseado em Java e simplifica o processo de desenvolvimento de sites da web dinâmicos. Veio para auxiliar principalmente os designers da web e programadores que rapidamente incorporam elementos dinâmicos em páginas da web, utilizando Java embutido e algumas tags de marcação simples. Estas tag(s) fornecem ao designer de

HTML um meio de acessar dados e lógica de negócio, armazenados em objetos Java sem ter que dominar as complexidades do desenvolvimento de aplicações.

Incorporar conteúdo dinâmico deve no final das contas envolver algum tipo de programação para descrever como aquele conteúdo é gerado. No entanto, o código gerado tende a ser caro para criar e manter, logo, minimizar a necessidade de programação é frequentemente um objetivo a ser alcançado. A combinação deste objetivo com o objetivo da SUN por um suporte robusto e repleto de recursos para Java no servidor, ou seja, um sistema modelo baseado em Java, o JSP, foi um resultado que teve boa aceitação pela comunidade.

Com o conhecimento de Servlets veremos que o JSP não oferece nada que você não possa conseguir com os Servlets puros. O JSP, entretanto, oferece a vantagem de ser facilmente codificado, facilitando assim a elaboração e manutenção de uma aplicação.

Muitos dizem que JSP é um tanto quanto híbrido entre os sistemas de modelo, porque suporta dois estilos diferentes para adicionar conteúdo dinâmico as páginas da web. Assim com o ASP, PHP e SSJS (Server-Side Javascript), scripts podem ser embutidos em páginas JSP contendo o código de programação tipicamente Java. (Assim é possível dizer que a especificação JSP permite a possibilidade de diferentes linguagens de criação de scripts).

Exemplo de uma JSP utilizando “scriptlets”:

Criando e imprimindo uma variável Java dentro de uma jsp :

```
<html>
<body>
    <%
     //Isto é um scriptlet
     System.out.println("Criando o objeto data");
     java.util.Date data = new java.util.Date();
    %>
    <br /> Olá agora o horário local é:
    <br />
    <%
```

```
      out.println(String.valueOf(data));
     %>
     <br />
</body>
</html>
```

JSP oferece diversos benefícios como um sistema para geração de conteúdo dinâmico. É uma tecnologia baseada em Java, então se aproveita de todas as características e vantagens desta linguagem como orientação a objetos, herança, encapsulamento, tratamento de exceções e gerenciamento de memória automática (Garbage Colector) conduzindo assim a um código mais robusto e flexível e de fácil manutenção.

Exemplo de uma JSP comum utilizando “expression”:

```
<html>
 <body>
     Data de hoje: <%=new java.util.Date() %>
 </body>
</html>
```


Neste exemplo, o código Java é uma expressão, delimitada por <%= e %>. Assim, depois do container consumir a JSP a expressão é calculada e convertida numa String que será apresentada na tela.

Existe uma API padrão publicada para JSP, e pelo fato do bytecode Java compilado ser portável através de todas as plataformas que aceitam uma JVM (Java Virtual Machine),o uso de JSP não restringe ao uso de uma plataforma de hardware, sistema operacional ou software servidor específico. Se uma mudança de qualquer um destes componentes se tornar necessária, todas as páginas JSP e classes de Java associadas podem ser migradas no estado em que se encontram sem precisar de uma nova compilação.

Outra maneira de utilizar linguagem Java dentro de um arquivo jsp utilizando a própria request auxiliando.

```
<html>
<body>
```

```
    <%
     // Isto é um scriptlet.
     System.out.println("Criando o objeto data");
     java.util.Date data = new java.util.Date();
    %>
        Olá! Agora o horário local é:
     <%
     out.println(String.valueOf(data));
     out.println("<BR> e seu endereço IP é: ");
     out.println(request.getRemoteHost());
     %>
</body>
</html>
```

A tecnologia JSP tem agora um grande papel na evolução contínua desta tecnologia, um exemplo disso é o novo conceito para desenvolvimento gráfico das interfaces, o JavaServer Faces, e o surgimento de frameworks que auxiliarão os desenvolvedores no dia a dia.

Além de expressões inseridas no texto da página, é possível inserir trechos de código Java que são executados no servidor ao se processar a página. Como vimos anteriormente esses trechos de código são chamados 'scriptlets' e delimitados por <% e %>.

A execução de um 'scriptlet' ocorre durante a apresentação do texto correspondente à página HTML. Os 'scriptlets' são portanto executados na ordem em que aparecem na página.

Quando abrimos uma página de internet em nosso navegador, realizamos a solicitação HTTP (Hypertext Transfer Protocol).

O HTTP é o protocolo responsável pelo pedido e também pela resposta. Isso é realizado quando o computador envia um pedido solicitando uma página ou arquivo e o servidor da aplicação na internet devolve a página ou arquivo solicitado.

- request – contém a informação da solicitação HTTP e é representada pelo HttpServletRequest.
- URL: http://...../pagina.jsp?parametro=conteudo
- String valor = request.getParameter(“parametro”);
- URL: http://...../pagina.jsp?p1=v1&p2=v2…
- response – contém a informação da resposta HTTP que será devolvida ao cliente e é representada pelo HttpServletResponse. Não é utilizado com muita frequência.
- out – responsável por escrever no corpo(body) da resposta HTTP e é representado pelo objeto JspWriter.
- session – responsável por guardar todas as informações do usuário e é representado pelo objeto do tipo HttpSession.
- application – é obtido pelo objeto de configuração de uma Servlet e é representado pelo objeto do tipo ServletContext.
- exception – toda e qualquer situação de erro de uma página JSP é tratado e representado pelo objeto do tipo Throwable.
- config – geralmente utilizado para obter parâmetro de inicialização a partir do arquivo web.xml e é representado pelo objeto do tipo ServletConfig.
- page – representa o operador “this” do objeto do tipo HttpJspPage e não utilizado com muita freqüência.
- pageContext – contém todo o contexto da página e é representado pelo objeto do tipo PageContext. Pode ser usado para definir, obter ou remover atributos da página JSP.

3.2.1 Diretivas

As diretivas do JSP são representadas por tags que afetam a estrutura dos servlets a partir da construção da página JSP.

Um exemplo:

<%@ diretiva atributo="valor" %>

Existem três tipos de diretivas: page, include e taglib.

- A diretiva page permite o controle da estrutura do servlet e pode ser colocada em qualquer ponto da página.
- A diretiva include permite incluir um arquivo na página JSP em tempo de interpretação e deve ser colocada no ponto de inserção do código.
- A diretiva taglib foi implementada no JSP 1.1 e é responsável por uma biblioteca de tags personalizada.

<%@ page import = "package1.*, package2,*,..." %>

<%@ include file="URL" %>

<%@ page errorPage="relativeURL" %>

<%@ page isErrorPage="true|false" %>

Exemplo:

pagina1.jsp

```
<%@page import="java.util.*"%>
<HTML>
<BODY>
     <%      System.out.println("Criando o objeto data");
      Date data = new Date(); %>
     Olá! Agora o horário local é:  <%=data%>
</BODY>
</HTML>
```

pagina2.jsp

```
<HTML>
<BODY>
     Incluindo pagina1.jsp...<BR>
     <%@ include file="pagina1.jsp"%>
</BODY>
</HTML>
```

3.2.2 Declarações

As declarações são representadas pelos delimitadores <%! e %>. Servem para que possamos definir variáveis e métodos específicos para uma página JSP. Os métodos e variáveis que forem declarados entre os delimitadores podem ser referenciados por elementos de criação de scriptlets na mesma página. Temos que lembrar que a cada declaração há a necessidade de finalizar ou separar por "ponto-e-vírgula".

Vejamos alguns exemplos:

<%! int x = 8; %>

<%! double a, b; long c; %>

<%! Circle a = new Circle(2.0); %>

```
<%@ page import="java.util.*"%>
<HTML>
<BODY>
     <%!Date data = new Date();


     Date getData() {
      System.out.println("Dentro do metodo getData()");
      return data;
     }%>
     Olá! Agora o horário local é:
     <%=getData()%>
</BODY>
</HTML>
```

3.2.3 Ações JSP

Podemos utilizar as ações em JSP para executar operações externas ao Servlet em tempo de execução. Também é possível concatenar várias páginas numa única resposta, além de incluir JavaBeans.

As principais ações são:

- forward
- include
- useBean
- setProperty
- getProperty

```
Exemplo de Tag com conteúdo
<alguma:tag>
conteúdo
</alguma:tag>
```

```
Exemplo de Tag sem conteúdo
<alguma:tag />
```

```
Exemplo de Tag com parâmetro
<alguma:tag nome="valor" />
```

```
Exemplo de Include simples
<HTML>
<BODY>
    <BR>Incluindo pagina2.jsp...
    <jsp:include page="pagina2.jsp" />
</BODY>
</HTML>
```

```
Exemplo de Include com parâmetros
<HTML>
<BODY>
    <BR>Incluindo pagina2.jsp passando parametros...
    <jsp:include page="pagina2.jsp">
    <jsp:param name="ParamName1" value="ParamValue1" />
```

```
    <jsp:param name="ParamName2" value="ParamValue2" />
    </jsp:include>
</BODY>
</HTML>
```

3.2.4 Escopos

Para cada requisição (link, url, etc) feita, é executado um request. E para cada requisição há um escopo de request que dura uma página, ou seja, uma página é acessada com parâmetros n1, n2. Se em vez de um forward, fizesse um redirecionamento, utilizando a tag <a href="http://qualquercoisa">, para outra página, os parâmetros não estariam mais disponíveis, pois, objetos dentro de request duram apenas uma requisição, como conhecemos os objetos de criação local que só existem até a finalização do bloco.

Há um segundo escopo chamado escopo de Sessão. Basicamente, é um escopo onde para cada cliente no servidor, é gerado um espaço de armazenamento temporário dos dados de sessão do usuário. Por padrão, cada escopo de sessão dura 30 minutos (pode ser configurado) e dessa forma, com escopo de sessão, é possível pegar valores em qualquer página, pois ele existe enquanto a sessão existir.

Escopo Application é válido para toda a aplicação, ou seja, coisas colocadas em Application podem ser acessadas à qualquer momento, em qualquer parte do sistema. Similar ao "global" de outras linguagens (asp, php).

Além destes escopos, também temos um escopo chamado "page" que é utilizado somente no ciclo de vida das JSP's da sua aplicação. Nele os objetos são definidos em e para cada página e existem apenas para elas mesmas.

- Pagina de chamada

```
<HTML>
<BODY>
    <FORM METHOD="POST" ACTION="mostrar_nome.jsp">
     Qual seu nome?<INPUT TYPE="TEXT" NAME="username" SIZE="20">
     <P><INPUT TYPE="SUBMIT" VALUE="ENVIAR"></P>
    </FORM>
</BODY>
```

```
</HTML>
```

- Pagina de resposta que chama proxima_pagina.jsp

```
    <%
    String nome = request.getParameter("username");
    session.setAttribute("seuNome", nome);
%>
<HTML>
<BODY>
    <A HREF="proxima_pagina.jsp">Continue</A>
</BODY>
</HTML>
```

- próxima_pagina.jsp

```
<HTML>
<BODY>
    Olá,<%=session.getAttribute("seuNome")%>
</BODY>
</HTML>
```

3.2.5 Beans

Segundo Java EE 6, a definição comum está na especificação de Managed Beans, onde são definidos como objetos que são gerenciados no contêiner com pouquíssimas restrições de programação, também conhecidos pelo acrônimo POJO (Plain Old Java Object). Com isso podemos dizer que um bean ou POJO, é o objeto mais simples em Java, onde tem o nome da classe, seus atributos e métodos.

Com pouquíssimas exceções, quase toda classe Java concreta que possui um construtor com nenhum parâmetro (ou um construtor designado com a anotação `@Inject`) é um bean.

```
DadosUsuario.java
package beans;
```

```
public class DadosUsuario {

    private String username;

    private String email;

    private int idade;

    public String getUsername() {

     return username;

    }

    public void setUsername(String username) {

     this.username = username;

    }

    public String getEmail() {

     return email;

    }

    public void setEmail(String email) {

     this.email = email;

    }

    public int getIdade() {

     return idade;

    }

    public void setIdade(int idade) {

     this.idade = idade;

    }

}
```

dados_usuario.jsp

```
<HTML>

<BODY>

    <FORM METHOD="POST" ACTION="dados_sessao.jsp">

     nome: <INPUT TYPE="text" NAME="username" SIZE="20"><BR>

     e-mail: <INPUT TYPE="text" NAME="email" SIZE="20"><BR>
```

```
      idade: <INPUT TYPE="text" NAME="idade" SIZE="4">
      <P>
            <INPUT TYPE="submit" value="OK">
      </P>
     </FORM>
</BODY>
</HTML>
```

dados_sessao.jsp

```
     <jsp:useBean id="usuario" class="beans.DadosUsuario" scope="session" />
<jsp:setProperty name="usuario" property="*" />
<HTML><BODY>
     <A HREF="proxima_pagina.jsp">Continuar</A>
</BODY></HTML>
```

proxima_pagina.jsp

```
<jsp:useBean id="usuario" class="beans.DadosUsuario" scope="session" />
<HTML>
<BODY>
     Você informou:<BR>
     Nome:<%=usuario.getUsername()%><BR>
     Email:<%=usuario.getEmail()%><BR>
     Idade:<%=usuario.getIdade()%><BR>
</BODY>
</HTML>
```

3.2.6 Tratamento de erros

Uma exceção é uma situação provocada pela interrupção no fluxo normal de execução de uma aplicação, como por exemplo, acessar um arquivo que não existe ou tentando se conectar numa base de dados inexistente.

Tratamento de erros é a manipulação da exceção imprevista que foi resultada de alguma interrupção em tempo de execução. A maneira mais comum de tratamento em páginas JSP seria redirecionar o usuário para uma página de erro padrão.

Temos várias maneiras de lidar com o tratamento de exceções em JSP:

- Podemos usar os atributos errorPage e isErrorPage
- Usando o elemento <error-page> no arquivo web.xml.
- Usando scriplets com codificação java para o tratamento da exceção.

pagina_erro.jsp

```
<%@page errorPage="error_page.jsp" contentType="text/html"%>
<html>
<body>
    <%String s = null;
    out.println(s.trim());%>
</body>
</html>
```

error_page.jsp

```
<%@page isErrorPage="true" contentType="text/html"%>
<html>
<body>
    Request que falhou: ${pageContext.errorData.requestURI}
    <br /> Código de Status: ${pageContext.errorData.statusCode}
    <br /> Exceção: ${pageContext.errorData.throwable}
    <br /> ${pageContext.errorData.servletName}
</body>
</html>
```

3.2.7 Expressions Language

Expression Language (Linguagem de Expressão) se iniciou na 1ª versão da JSTL (JSP Standard Tag Library), antes eram usados scriptles para manipular dados nas aplicações. JSTL introduziu o conceito de Linguagem de Expressão – EL, que simplifica muito o desenvolvimento de páginas JSP disponibilizando vários recursos através de tags. Agora faz parte da especificação JSP 2.0.

Existem inúmeras tags que provêm recursos para tarefas estruturadas como iterações e condições, processamento de arquivos XML, internacionalização e acesso a dados usando SQL.

A forma de representação de uma Expression Language(EL) é ${}, a expressão será inserida entre as chaves. A EL apresenta resultados no FrontEnd onde favorece o desenvolvimento abordando o MVC.

```
jsp:setProperty name="retangulo" property="area" value="50" />

<jsp:setProperty  name="retangulo"  property="area"  value="${retangulo.base  *
retangulo.altura}" />

<h1>A área do retângulo é: 100 cm 2</h1>

<h1>A área do retângulo é: ${retangulo.base * retangulo.altura} cm2</h1>
```


3.2.7.1 Objetos Implícitos – EL

- pageScope
- requestScope
- sessionScope
- applicationScope
- param
- paramValues
- header
- headerValues
- initParam
- cookie
- pageContext

3.2.8 Taglibs

São bibliotecas que foram criadas para serem integradas e usadas nas páginas JSP com intenção de substituir os scriptles gerados. Uma TagLib executa ações que são utilizadas nas páginas JSP como etiquetas que manipulam os dados e variáveis das JSPs.

É definido por um TLD(Tag Librarie Descriptor) e uma classe Java que implementa a inteface da Tag no JSP(TagSupport). O arquivo descritor é representado por um arquivo XML com extensão tld que descreve a configuração e o mapeamento entre as classes em java e otimiza o código que será escrito na página JSP.

```
Exemplos de representação de uma Tag.
<ex:Saudacao />

<ex:Saudacao>
Bom Dia
</ex:Saudacao>

<ex:Saudacao message="Bom Dia" />
```

```
<TLD>

<taglib>
<tlib-version>1.0</tlib-version>
<jsp-version>2.0</jsp-version>
<short-name>Exemplo de TLD</short-name>

    <tag>
      <name>Saudacao</name>
      <tag-class>tagLibs.Saudacao</tag-class>
      <body-content>empty</body-content>
    </tag>

</taglib>
```

```
<CLASSE>
package tagLibs;


import java.io.IOException;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.SimpleTagSupport;


public class Saudacao extends SimpleTagSupport {
     // ------------------------------------
     // o nome deve ser sempre doTag() devido
     // ao extends da SimpleTagSupport
     // ------------------------------------
     public void doTag() throws JspException, IOException {
      // JspWriter é uma classe do Java EE
      // As JSPs utilizam para mostrar algo na tela
      // getJspContext é um método que está dentro
      // da SimpleTagSupport.
      JspWriter out = getJspContext().getOut();
      out.println("<h1>Bom Dia</h1>");
     }
}
```

```
<EXEMPLO.JSP>


<%@taglib prefix="ex" uri="WEB-INF/Ex_12_minhas_tags.tld" %>
<html>
<body>
     <ex:Saudacao />
</body>
</html>
```

SaudacaoTag.java

```
package taglibs;

import javax.servlet.jsp.tagext.*;
import javax.servlet.jsp.*;
import java.io.*;
public class SaudacaoTag extends SimpleTagSupport {
     public void doTag() throws JspException, IOException {
      JspWriter out = getJspContext().getOut();
      out.println("Bom Dia");
     }
}
```

minhas_tags.tld

```
<taglib>
     <tlib-version>1.0</tlib-version>
     <jsp-version>2.0</jsp-version>
     <short-name>Exemplo de TLD</short-name>
     <tag>
      <name>Saudacao</name>
      <tag-class>taglibs.SaudacaoTag</tag-class>
      <body-content>empty</body-content>
     </tag>
</taglib>
```

teste.jsp

```
<%@taglib prefix="ex" uri="WEB-INF/minhas_tags.tld"<html>
```

```
<body>
     <ex:Saudacao />
</body>
</html>
```

ConteudoTag.java

```
package taglibs;

import javax.servlet.jsp.tagext.*;
import javax.servlet.jsp.*;
import java.io.*;
public class ConteudoTag extends SimpleTagSupport {
     StringWriter sw = new StringWriter();
     public void doTag() throws JspException, IOException {
      getJspBody().invoke(sw);
      getJspContext().getOut().println(sw.toString());
     }
}
```

minhas_tags.tld

```
<taglib>
     <tlib-version>1.0</tlib-version>
     <jsp-version>2.0</jsp-version>
     <short-name>Exemplo de TLD com conteúdo</short-name>
     <tag>
      <name>Conteudo</name>
      <tag-class>taglibs.ConteudoTag</tag-class>
      <body-content>scriptless</body-content>
     </tag>
```

```
</taglib>
```

teste.jsp

```
<%@taglib prefix="ex" uri="WEB-INF/minhas_tags.tld"%>
<html>
<body>
     <ex:Conteudo>
      Conteúdo da Tag
     <ex:Conteudo />
</body>
</html>
```

AtributoTag.java

```
package taglibs;
import javax.servlet.jsp.tagext.*;
import javax.servlet.jsp.*;
import java.io.*;
public class AtributoTag extends
SimpleTagSupport {
     private String mensagem;
     public void setMensagem(String msg) {
      this.mensagem = msg;
     }
     StringWriter sw = new StringWriter();
     public void doTag() throws JspException,
     IOException {
      JspWriter out = getJspContext().getOut();
      if (mensagem != null) {
```

```
            out.println( mensagem );
      } else {
            getJspBody().invoke(sw);
            out.println(sw.toString());
      }
     }
}
```

minhas_tags.tld

```
<taglib>
     <tlib-version>1.0</tlib-version>
     <jsp-version>2.0</jsp-version>
     <short-name>Exemplo de TLD com conteúdo</short-name>
     <tag>
      <name>Atributo</name>
      <tag-class>taglibs.AtributoTag</tag-class>
      <body-content>scriptless</body-content>
      <attribute>
            <name>mensagem</name>
      </attribute>
     </tag>
</taglib>
```

teste.jsp

```
<%@ taglib prefix="ex" uri="WEB-INF/minhas_tags.tld"%>
<html>
<body>
     <ex:Atributo mensagem="Bom Dia" />
</body>
</html>
```

Exercícios JSP:

4 – Criar uma página jsp que mostre o resultado de 6 (seis) operações matemáticas e 1 (um) número aleatório entre 0 e 100 como no exemplo abaixo:

2 + 2 = 4

13 – 7 = 6

8 x 3 = 24

26 ÷ 4 = 6,5

3 elevado a 4ª potência é 81

A raiz quadrada de 81 é 9

O número aleatório é 53

5 – Crie uma JSP que leia um parâmetro “nome” e escreva o conteúdo na página.

6 – Crie uma JSP que leia 2 (dois) parâmetros “n1, n2” e mostre o resultado da soma entre os dois na página.

7 – Crie uma JSP que gere um número aleatório, exiba e diga em seguida se o número gerado é par ou ímpar. Modifique a JSP para que quando o número gerado for par seja apresentado em azul e quando ímpar seja apresentado em vermelho. Crie uma JSP que leia 3 parâmetros (2 números e uma operação) e mostre na página o resultado do calculo.

8 – Crie 2 JSPs “cabecalho.jsp e rodape.jsp” e as inclua dentro de uma terceira página chamada “conteudo.jsp“ onde será mostrada a data atual do sistema.

9 – Modifique o exercício anterior para que a data atual do sistema seja retornada através de um método.

10 – Crie uma JSP que receba um número como parâmetro e caso seja par redirecione para uma página “par.jsp” ou senão redirecione para “impar.jsp”.

11 – Crie um JSP com um formulário para registro de usuários, após o preenchimento devera ser exibido um termo de licença com as opções “Aceitar, Recusar”, após a escolha do usuário deverá ser exibida uma página exibindo os dados informados pelos usuários.

12 – Modifique o exercício anterior para que o mesmo utilize beans para armazenar as informações.

13 – Crie uma página JSP onde possa ser informado as medidas da base e altura de um triângulo. Guarde as informações em um bean e utilize EL para mostrar os valores informados e o valor calculado da área em outra página.

14 – Crie 2 (duas) TagLibs para efetuar conversões entre Real e Dollar. A tag deve receber 2 parâmetros, “cotacao” e “valor”, e caso aconteça algum erro deverá ser exibida uma mensagem configurada através do conteúdo da tag.

Exercício Livre.

Crie uma JSP para efetuar a soma de 2 números informados em um formulário e mostre o resultado da soma na própria página. Caso ocorra alguma exceção informe ao usuário o problema ocorrido em uma página JSP de erro.

CAPÍTULO 4

JSF – JAVASERVER FACES 2.0

4.1 Conhecendo JSF

JavaServer Faces, ou JSF, é a tecnologia indicada para o desenvolvimento da camada web que conta com vários benefícios, entre eles o desacoplamento de apresentação(view) e a lógica de negócio(model), contando com um template de componentes bastante interessante.

O JSF permite criar componentes web, como a Microsoft fazia com os componentes visuais OCX(antigo VB) e hoje com o DotNet que possui os Web Controls e os Windows Controls, deixando o desenvolvimento das páginas mais elegantes e funcionais.

Existem desenvolvedores que acham que os benefícios que o Struts apresenta não chegam a compensar o seu uso, acham que burocratiza o desenvolvimento e agrega pouco valor. Mas então surgiu o JSF, e como toda a novidade, a comunidade web de desenvolvedores esboçou rejeição inicialmente, mas em pouco tempo conseguiram entender o que essa nova tecnologia trazia e então a aceitação passou a ser bem razoável. O JSF veio para facilitar o desenvolvimento JAVA para web, e traz o conceito de desenvolvimento web com a utilização de componentes(component-based).

A transição da tecnologia Struts para a tecnologia JSF pode causar estranheza, dependendo da pessoa, mas verá que vale a pena investir nesta nova forma de desenvolvimento web. O Struts é mais um “organizador/controlador” de sites, trabalhando sempre com a camada de páginas web. Já o JSF, trata o desenvolvimento web bem mais profissionalmente.

A estrutura do JSF fica em destaque pois nela temos o ciclo de vida, comunicação entre managed beans, escopos de conversação, árvore de componentes, converters e validators, phase listeners, modelo dos componentes etc.

Devemos tomar cuidado ao fazer o componente ou bloco de código funcionar a todo custo sem nem ao menos ler a documentação do mesmo [“brute-force”]. Não adianta correr para tentar usar os componentes se o desenvolvedor não possuir os fundamentos básicos sobre JSF ou mesmo sobre desenvolvimento Java para web.

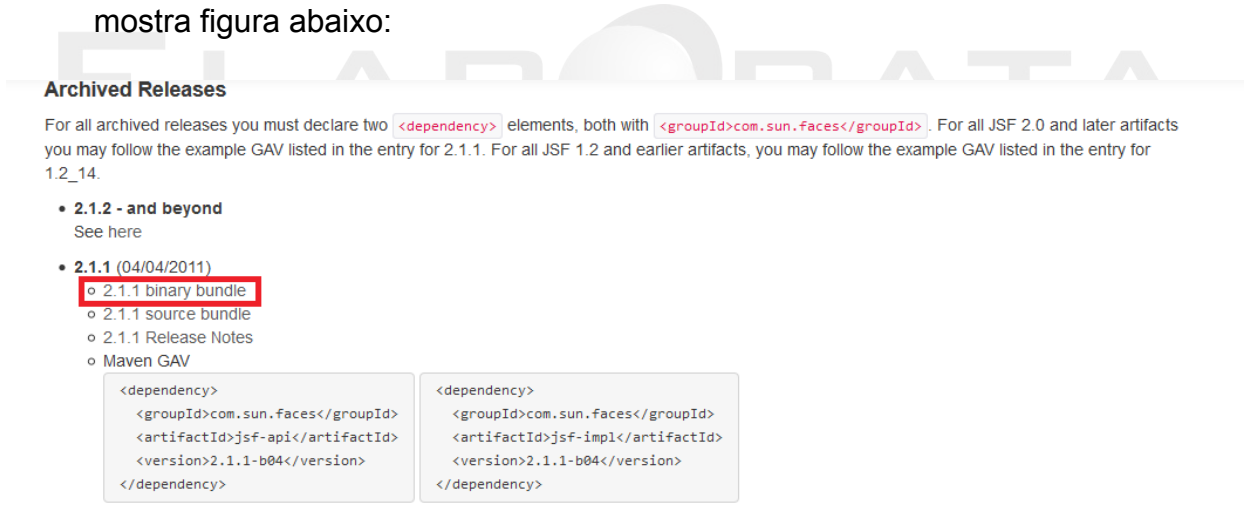
4.1.1 Desenvolvendo com JSF

Como falaremos sobre programação utilizando o JSF para manipulação de páginas jsp, precisaremos de um ambiente de desenvolvimento já preparado para iniciarmos os trabalhos. Utilizaremos o ambiente que configuramos no início do módulo.

Isso já nos permite iniciar o aprendizado de Java Server Faces(JSF), pois aqui ainda estamos tratando a manipulação das páginas, ou seja, a parte da visão do MVC. Dessa forma fica fácil entender por que não precisaremos de tanta configuração para o nosso ambiente de desenvolvimento.

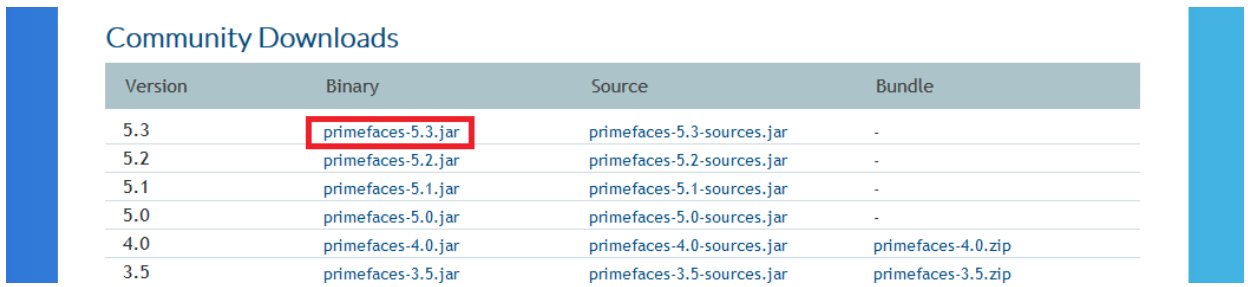
Se a versão do eclipse que estiver usando não tiver o JSF ou primefaces, devemos baixar e configurar o ambiente no eclipse antes de começarmos a criação do primeiro projeto em JSF.

1. Acesse o seguinte link: https://javaserverfaces.java.net/download.html
2. Selecionamos a versão estável mais atual que é a 2.1.1 source bundle conforme mostra figura abaixo:



Baixando jar do JSF para instalação manual.

3. Acesse o seguinte link: http://www.primefaces.org/downloads.html
4. Selecionamos a versão estável mais atual que é a 5.3 conforme mostra figura abaixo:



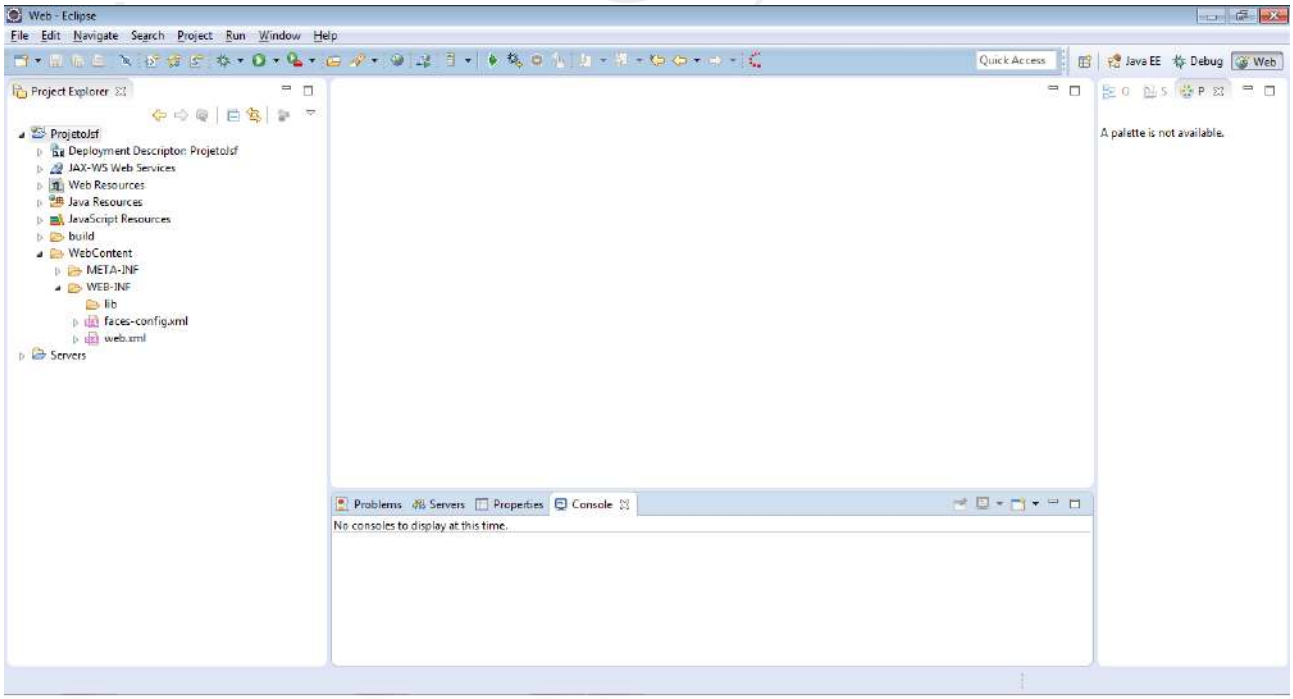
Version	Binary	Source	Bundle
5.3	primefaces-5.3.jar	primefaces-5.3-sources.jar	-
5.2	primefaces-5.2.jar	primefaces-5.2-sources.jar	-
5.1	primefaces-5.1.jar	primefaces-5.1-sources.jar	-
5.0	primefaces-5.0.jar	primefaces-5.0-sources.jar	-
4.0	primefaces-4.0.jar	primefaces-4.0-sources.jar	primefaces-4.0.zip
3.5	primefaces-3.5.jar	primefaces-3.5-sources.jar	primefaces-3.5.zip

Baixando jar do PrimeFaces para instalação manual.

Então agora podemos criar nosso projeto utilizando JSF 2.0:

1. Selecione no Eclipse o menu “file/new/Dynamic Web Project”;
2. Na janela de configuração, digite o nome do projeto ”ProjetoJsf”;
3. No campo “Target runtime” escolhemos o nosso servidor de aplicação que aqui será o Tomcat e em “Configuration” deixar selecionado o Default Configuration for Apache Tomcat.
4. Agora devemos pressionar o botão “Modify” no campo “Configuration”.
5. Em Projetct Faces marque Java Server Faces e altere para a versão 2.0
6. Pronto, agora é só finalizar. Clique em “ok” e “finish”.
7. Antes de iniciar a criação dos arquivos devemos descompactar o arquivo baixado no tópico 3, do tema anterior, procurar a pasta lib e copiar os jars para a pasta lib do projeto.

Verifique se os elementos foram criados, e se tem o arquivo “faces-config.xml”, dentro da pasta WEB-INF, conforme figura abaixo.



Elementos da estrutura de pastas.

Podemos iniciar nosso trabalho ajustando o código do arquivo web.xml que é gerado pelo plugin do eclipse e gera código desnecessário.

Com o aquivo web.xml aberto, selecione a aba “source” e em seguida precisamos alterar o código, iniciando pelo mapeamento.

```
<servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>??????</url-pattern>
 </servlet-mapping></web-app>
```

Agora é só trocar a url no mapeamento.

```
 <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>*.jsf</url-pattern>
 </servlet-mapping></web-app>
```

Através desse mapeamento que o container será informado que todo o pedido HTTP para uma página que termine com “.jsf” será encaminhada ao Faces Servlet que é a Servlet controler.

O arquivo “web.xml” deverá conter o código abaixo.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xmlns="http://java.sun.com/xml/ns/javaee"
     xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
     xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
     http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" version="3.0">
  <display-name>ProjetoJSF</display-name>
    <context-param>
        <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
        <param-value>server</param-value>
    </context-param>
    <servlet>
```

```
        <servlet-name>Faces Servlet</servlet-name>
        <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
    </servlet>
  <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>*.jsf</url-pattern>
    </servlet-mapping></web-app>
```

Utilizaremos o parâmetro de inicialização nomeado de "javax.faces.STATE_SAVING_METHOD", e que funciona parecido com um atributo de contexto que já fica disponível desde a inicialização do website e informa que o estado da sessão será salvo. O valor do parâmetro que escolhemos é o "server" que mantém o estado no HTTP. Poderíamos também escolher o parâmetro "client" e o estado ficaria salvo no Cliente, o que demandaria mais tráfego de rede.

Como a grande maioria das aplicações web criaremos o arquivo de inicialização "index.jsp". Substituiremos pelo código abaixo:

```
<% response.sendRedirect("inicio.jsf"); %>
```

O arquivo "index.jsp" redirecionará o pedido para um arquivo ao qual demos o nome de "inicio.jsf" e esse pedido será redirecionado automaticamente para o Faces Servlet. Vamos aproveitar e criar um gabarito facelets(template). A tecnologia facelets cria gabaritos que especificam áreas do site a serem substituídas dinamicamente, muito parecido com o struts com tiles.

Crie na pasta "Webcontent" do projeto uma nova XHTML Page em "new--> XHTML Page". Informe o nome "template" para o arquivo e clique em "next". Na próxima caixa de diálogo selecione "New facelet template" na lista de templates e pressione "Finish".

O arquivo "template.xhtml", deverá conter o seguinte conteúdo:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
          "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets">
```

```
<head>
        <title><ui:insert    name="title">Template    dinâmico    de    pagina
web</ui:insert></title>
</head>
     <body>
      <div id="header"><ui:insert name="header"/></div>
      <div id="content"><ui:insert name="content"/></div>
      <div id="footer"><ui:insert name="footer"/></div>
     </body>
</html>
```

Agora que já temos nosso gabarito, criaremos a página “inicio.xhtml”. Quando a página index.jsp redirecionar para a página inicio.jsf, o Faces Servlet vai procurar a página inicio.xhtml. O procedimento de criação é igual ao do template.xhtml visto anteriormente.

O conteúdo do arquivo inicio.xhtml deverá ser modificado conforme segue abaixo:

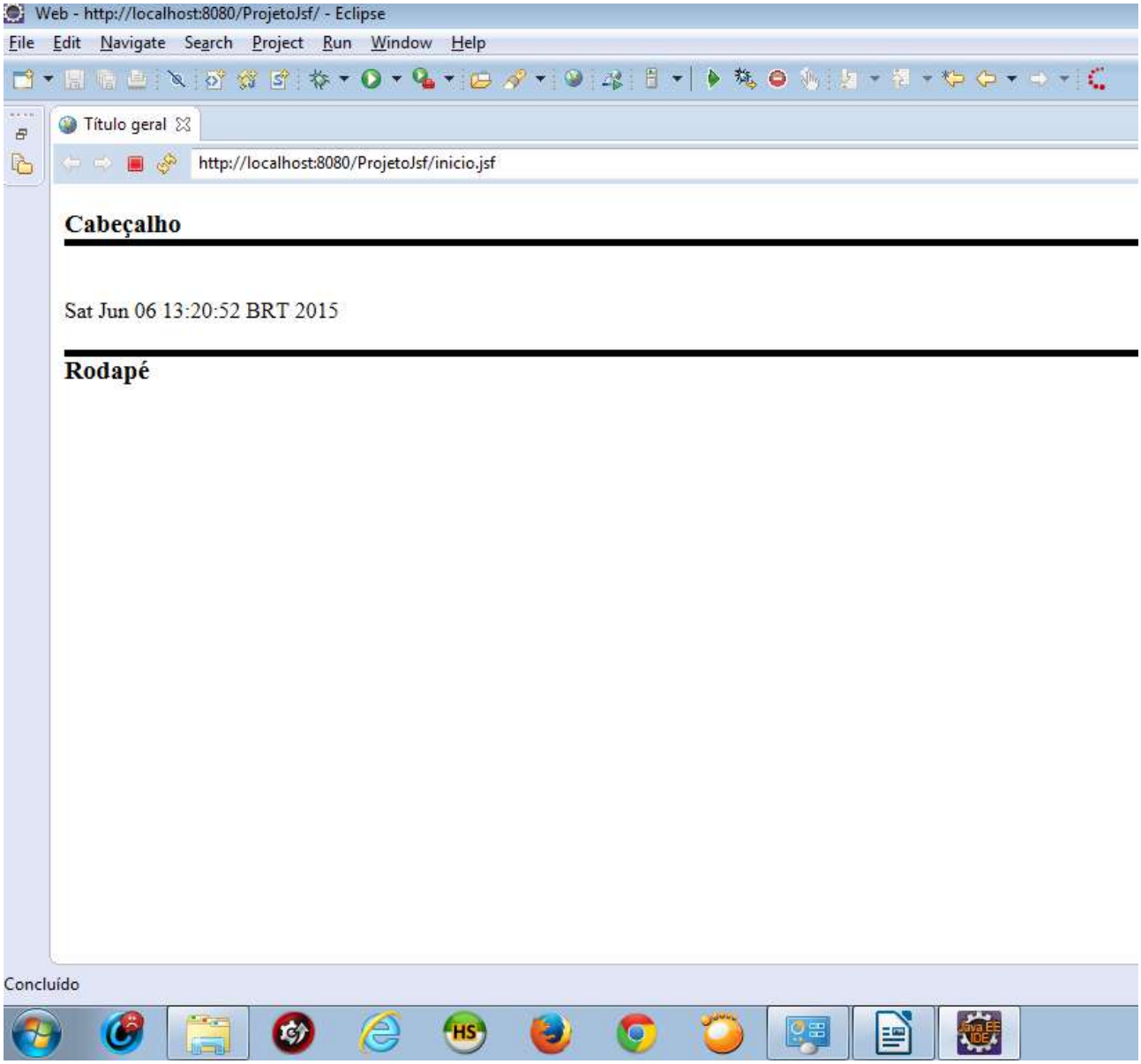
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
     xmlns:ui="http://java.sun.com/jsf/facelets"
     xmlns:h="http://java.sun.com/jsf/html"
     xmlns:f="http://java.sun.com/jsf/core">
<ui:composition template="template.xhtml">
     <ui:define name="header">
      <h3 style="border-bottom:thick solid #000000;">Cabeçalho</h3>
     </ui:define>
     <ui:define name="content">
      <br/><h:outputText value="#{bean.mensagem}"/>
     </ui:define>
     <ui:define name="footer">
      <h3 style="border-top:thick solid #000000;">Rodapé</h3>
```

```
    </ui:define>
</ui:composition>
</html>
```

Finalmente criaremos o Bean ”bean” que será o Managed Bean. Criar um pacote na pasta resources(src) “br.com.elaborata” onde criaremos a classe chamada de “ClasseBean”. Note que acrescentamos a anotação “@ManagedBean” responsável pela manipulação do objeto bean.

```
package br.com.elaborata;


import java.util.Date;
import javax.faces.bean.ManagedBean;
@ManagedBean(name="bean")
public class ClasseBean {
    public String getMensagem() {
     return new Date().toString();
    }
}
```

Ao realizar a chamada da página o resultado deverá ser o que segue:



Resultado da aplicação funcionando.

4.2 Que é JSF?

Arquitetura que utiliza componentes para a construção de interfaces com o usuário(GUI-Grafical User Interface) e também orienta seus eventos. Foi criado para o desenvolvimento de aplicações Web em Java tornando mais fácil a implementação da programação em três camadas(MVC – Model, View, Control).

JSF é um framework de fácil manuseio e para quem conhece este fundamento ou tem conhecimento em desenvolvimento web terá uma certa intimidade ao manipular os componentes do JSF. Permite a elaboração de interfaces de usuário web onde é possível colocar componentes em um formulário e fazer uma ligação com objetos Java o que permite que a regra de negócio fique separada da lógica.

Segundo a oracle, JSF pode ser definido como um novo modelo de programação para web, onde podemos gerenciar mais facilmente os componentes visuais que são, cada vez mais requisitados. O modelo utilizado é o de orientação a eventos. Além de possuir um conjunto de APIs muito eficiente que auxilia na representação dos componentes visuais realiza o gerenciamento de seu estado, manipula eventos e realiza a validação da entrada de dados.

Também existe um conjunto de APIs diferente um pouco da API de orientação a eventos, pois essa é responsável pela definição e controle da navegação entre páginas web, e outro conjunto, mas agora de Tags que são utilizadas em páginas JSP e/ou XHTML(facelets) que representam os componentes visuais.

O JSF para muitos serve somente para a criação de páginas web, mas isso é um ledo engano, pois sua arquitetura trabalha com o desacoplamento entre os componentes e o código que efetivamente exibe o componente(renderizador), deixando livre para que a mesma aplicação seja portada para outras interfaces. É importante saber que existe renderizador JSF para diversas tecnologias, tais como o HTML, WML(Wireless Markup Language), SVG(Scalable Vector Graphics) entre outras.

4.3 Bibliotecas de componentes

As bibliotecas de interface do JSF são baseadas em uma tecnologia poderosa criada em 2002 pela Macromedia, denominada Rich Internet Application (RIA). Quando utilizados de forma adequada constroem aplicações complexas, auxiliando no trabalho do desenvolvedor. Anteriormente, tínhamos conceitos semelhantes, como o Remote Scripting, da Microsoft, em 1998, e o X Internet, da Forrester Research, em 2000.

O maior objetivo de uma RIA é contar com características e funcionalidades que já existem apenas em aplicações desktop auxiliando na aplicação de projetos web, proporcionando uma experiência mais agradável para o usuário. Estas soluções geralmente rodam em um navegador – portanto não precisam ser instaladas diretamente no computador – adicionando uma camada intermediária entre cliente e servidor a qual denominamos de “client engine”.

Além dos componentes padrões, já inseridos na implementação JSF, podemos contar com várias bibliotecas produzidas por terceiros, que podem ser utilizadas por qualquer aplicação JSF.

Provavelmente a mais famosa delas seja a biblioteca "RichFaces", que é fornecida pela Red Hat(JBoss), PrimeFaces, ICEfaces, OpenFaces e Oracle ADF.

4.3.1 Richfaces

O RichFaces é um framework RIA criado por Alexander Smirnov. para o JavaServer Faces (JSF) com suporte a AJAX, skins prontos para diversos usos (que podem ser customizados), e conta ainda com componentes de interface gráfica.

Essa biblioteca surgiu pela pesquisa e desenvolvimento da empresa Exadel, que em 2006 liberou a primeira versão disponível para o público em um pacote comercial. Já no ano seguinte, a Exadel foi dividida e um acordo de cooperação com a JBoss possibilitou a incorporação da biblioteca de componentes Ajax4jsf, além de uma mudança na forma de distribuição, já que o pacote passou a ser gratuito e de código aberto. Isso resolveu muitos problemas de versões e compatibilidades existentes. Atualmente o framework é mantido pela gigante americana JBOSS responsável por projetos como Hibernate com releases constantes e uma comunidade bastante ativa.

Richfaces baseia-se no apoio AJAX, faz uso do Jquery(Framework JavaScript), no controle de eventos e é padronizado com o JSF2. Os componentes estão divididos em duas partes:

- A4j – controle de componentes AJAX da biblioteca. Servem para submeter dados de forma assíncrona e controlar o comportamento da página nesse tipo de requisição.
- Rich – componentes visuais autônomos da biblioteca com suporte nativo a AJAX. Destinados a exibir, receber e organizar informações na tela. Em relação a dependências externas, para funcionar o Richfaces precisa de quatro arquivos para funcionar:
    - richfaces-core-api.jar;
    - richfaces-core-impl.jar;
    - richfacescomponents-api.jar;
    - richfaces-components-ui.jar

Entre as características presentes no Richfaces podemos destacar:

- Controle avançado de requisições AJAX baseado em fila para organizar as solicitações assíncronas;
- Validação do lado do cliente, evitando tráfego de rede para o servidor no caso de preenchimento incorreto de campos no formulário;
- Kit próprio para desenvolvimento de componentes;
- Integração com o plugin JBoss Tools para Eclipse e NetBeans;
- Facilidade para criação de testes automatizados em componentes, action, listeners e páginas.

4.3.2 ICEFaces

Lançado pela empresa canadense ICEsoft Technologies em 2004, é baseada no JQuery e no YahooUI(User Interface Library – YUI).

ICEfaces possui três bibliotecas de componentes AJAX:

- ICEfaces Advanced Components (ACE) – devem ser preferencialmente utilizados por apresentarem melhor performance. São componentes novos e com baixo tráfego de rede.
- ICEfaces ICE Components – ideal para aplicações e navegadores mais antigos, possui os componentes das primeiras versões do ICEfaces que foram atualizados para serem compatíveis com o JSF2;
- ICEfaces Enterprise Comonents (AEE) – incluída como oferta comercial é composta por mais de 30 componentes com classes mais aprimoradas e recursos e funcionalidades extras. Porém, como dito anteriormente, esses benefícios só são oferecidos em versões pagas.

Em relação a dependências externas, para funcionar o ICEfaces precisa de quatro arquivos:

icefaces.jar,

icepush.jar,

icefaces-ace.jar,

icefaces-compat.jar.

A versão mais recente, ICEfaces 3, possui dentre suas características:

- Ajax Automático – eliminando completamente a necessidade de desenvolvedores de, utilizarem o padrão JSF <f:ajax> tags;
- Adição de window scope – possibilitando o gerenciamento de uma janela ou aba do navegador;
- Ajax push atualização assíncrona baseada em fila. Permite atualizar de forma incremental qualquer parte da página a qualquer momento, para qualquer grupo de usuários;
- Versão mobile para desenvolvimento de aplicações para dispositivos portáteis;
- Integração através de plugins com as IDES Eclipse e Netbeans:

4.3.3 PrimeFaces

O PrimeFaces é uma biblioteca de componentes RIA que surgiu em 2008 através da empresa de origem turca chamada Prime Technology, especializada em desenvolvimento ágil, treinamento e consultoria Java EE.

O projeto tem o código aberto, encontra-se na versão 3.4.1 e a comunidade é bastante ativa, sugerindo constantemente possíveis melhorias, reportando bugs e fornecendo pequenos patches de correção, quando aplicáveis.

O framework possui, além do tradicional ambiente desktop, um kit voltado para dispositivos móveis chamado PrimeFaces Mobile, que se encontra na versão 0.9.3.

4.4 Navegação JSF

No Java Server Faces existe um modelo de navegação que pode ser feito de duas maneiras: “ad-hoc” e também a controlada.

A primeira, navegação “ad-hoc” é feita através do atributo “action” em uma tag de componente JSF.

Quando trabalhamos com JSF e queremos realizar uma navegação entre páginas, devemos incluir a tag de navegação dentro de um formulário(h:form).

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"

    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core">
<ui:composition template="template.xhtml">
      <ui:define name="header">
       <h3 style="border-bottom:thick solid #000000;">Cabeçalho</h3>
      </ui:define>
      <ui:define name="content">
      <h:form>
       <h:outputLabel value="Nome:"/>
       <h:inputText value="#{bean.nome}"/>  <br/>
       <!-- Para acessar a página diretamente -->
       <h:commandButton action="pagina2" value="Entrar" />
       <!-- Para acessar a página através de um método o qual iremos utilizar
-->
       <h:commandButton action="#{bean.navegar}" value="Entrar" />
      </h:form>
      </ui:define>
      <ui:define name="footer">
       <h3 style="border-top:thick solid #000000;">Rodapé</h3>
      </ui:define>
</ui:composition>
</html>
```

O atributo “action” não especifica o nome de uma página, mas uma action JSF, que será tratada pelo componente “NavigationHandler”. Se houver uma action registrada será desviado para ela. Senão ele procurará uma página com o nome especificado.

```
<h:form>
      <h:outputLabel value="Nome:"/>
      <h:inputText value="#{bean.nome}"/>  <br/>
```

```
      <!-- Para acessar a página através de um método o qual iremos utilizar
-->

      <h:commandButton action="#{bean.navegar}" value="Entrar" />

    </h:form>
```

Podemos realizar a navegação através de um “Action Method” no Managed Bean do projeto. O método não pode receber argumentos e deve retornar um “java.lang.Object”.

Definimos agora os atributos e métodos no nosso Managed Bean para a demonstração de navegação entre páginas.

```
package br.com.elaborata;


import javax.faces.bean.ManagedBean;
@ManagedBean(name="bean")
public class ClasseBean {
    private String nome;
    public Object navegar() {
     return "pagina2";
    }
    public Object voltar() {
     return "inicio";
    }
    public String getNome() {
     return nome;
    }
    public void setNome(String nome) {
     this.nome = nome;
    }
}
```

Se quisermos efetuar uma validação para que a página escolhida no caso de sucesso seja a página2 e se o caso for de falha redirecionará para a página de início com uma mensagem informativa, podemos acrescentar na página inicio.xhtml o campo

que receberá a mensagem, na ClasseBean um atributo mensagem, adicionar os métodos get e set e alterar o método navegar como segue abaixo:

```
    <h:form>
     <h:outputLabel value="${bean.mensagem}"/><br/>
     <h:outputLabel value="Nome:"/>
     <h:inputText value="#{bean.nome}"/>  <br/>
    <h:commandButton action="#{bean.navegar}" value="Entrar" />
    </h:form>
```

```
private String mensagem;
public String getMensagem() {
    return mensagem;
}
public void setMensagem(String mensagem) {
    this.mensagem = mensagem;
}
public Object navegar() {
    if("".equals(getNome())){
     setMensagem("O campo nome é obrigatório");
     return "inicio";
    }
     return "pagina2";
}
```

Existe uma forma de executar o código acima por fora do Managed Bean onde são criadas condições para invocar determinadas páginas, dependendo de um resultado. Para que isso seja possível devemos registrar as ações dentro do arquivo "faces-config.xml".

Neste arquivo, definimos a lógica de navegação, especificando as condições e a página que deverá ser invocada. No JSF 2.0, isto é opcional, pois podemos controlar a navegação em nossos Managed Beans, retornando o nome da página a ser chamada.

Vamos alterar nossa ClasseBean para retornar outro resultado que será resultante de uma condicional.

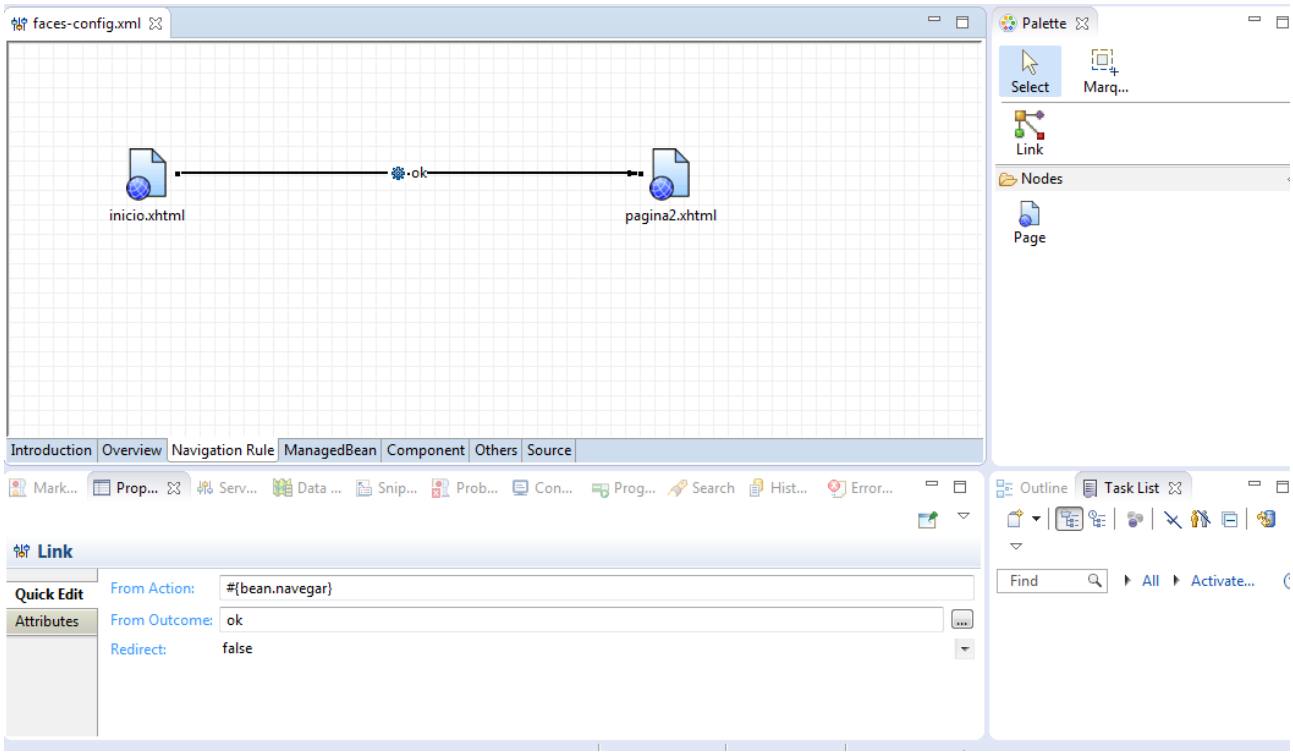
```
package br.com.elaborata;


import javax.faces.bean.ManagedBean;
@ManagedBean(name="bean")
@SessionScoped
public class ClasseBean {
    private String nome;
    private String mensagem;
    public String getMensagem() {
     return mensagem;
    }
    public void setMensagem(String mensagem) {
     this.mensagem = mensagem;
    }
    public Object navegar() {
     if("".equals(getNome())){
           setMensagem("O campo nome é obrigatório");
           return "inicio";
     }
           return "pagina2";
    }
    public Object voltar() {
     return "inicio";
    }
    public String getNome() {
     return nome;
    }
    public void setNome(String nome) {
     this.nome = nome;
```

```
    }
}
```

Note que acrescentamos a annotation “@SessionScoped” para informar que o bean deverá ser mantido na sessão do usuário.

A partir deste momento criaremos uma navegação dentro do arquivo de configuração do faces-config.xml. Quando abrir o arquivo no eclipse surgirão abas inferiores onde selecionaremos a aba Navigation Rule (Eclipse Neon).

Arraste as páginas, click em link do lado direito e crie uma conexão entre inicio e pagina2 preenchendo as propriedades, conforme a figura da próxima página.



Arquivo faces-config.xml.

Selecione o link e defina as propriedades como:

- De “inicio.xhtml” para “pagina2.xhtml”:
  - From Action: #{bean.navegar}
  - From Outcome:ok

Agora quando clicar na aba “source” teremos o seguinte código:

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

<faces-config
```

```
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                        xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-facesconfig_2_0.xsd"
    version="2.0">
     <navigation-rule>
      <display-name>inicio.xhtml</display-name>
      <from-view-id>/inicio.xhtml</from-view-id>
      <navigation-case>
            <from-action>#{bean.navegar}</from-action>
            <from-outcome>ok</from-outcome>
            <to-view-id>/pagina2.xhtml</to-view-id>
      </navigation-case>
     </navigation-rule>
</faces-config>
```

O mal que assola todo o começo e familiarização de uma nova tecnologia, nos traz alguns questionamentos; Como podemos desenvolver aplicações JSF? Onde exatamente devemos colocar o código? Como funciona a interação das páginas?

Veremos a seguir que não precisamos de muito para que esse fantasma seja afastado do processo de aprendizagem, pois este é um framework bem completo que permitirá a criação de um programa, apenas com o conhecimento básico, utilizando esta tecnologia.

4.4.1 Ciclo de vida

Começaremos a estudar como funciona o JSF por dentro, e isso acontece através de uma Servlet que controla o que chamamos de “Faces Servlet”. Entender o ciclo de vida de qualquer componente ou framework, possibilita a diminuição de falhas no decorrer do desenvolvimento.

A versão 2.0 do JSF, na especificação (JSR 314) trouxe melhorias e grande facilidade na compreensão do ciclo de vida. Temos aqui, duas macro-atividades que são executadas num pedido HTTP: “execute” e “render”. Elas trazem várias sub-atividades que auxiliam no trabalho do pedido HTTP. Em algumas fases existem

eventos que interceptam o fluxo de processamento, levando à renderização imediata da página ou ao encerramento da resposta.

O “execute” é responsável por processar o pedido e o “render” prepara a resposta gerando o código através do “render kit” padrão para gerar XHTML.

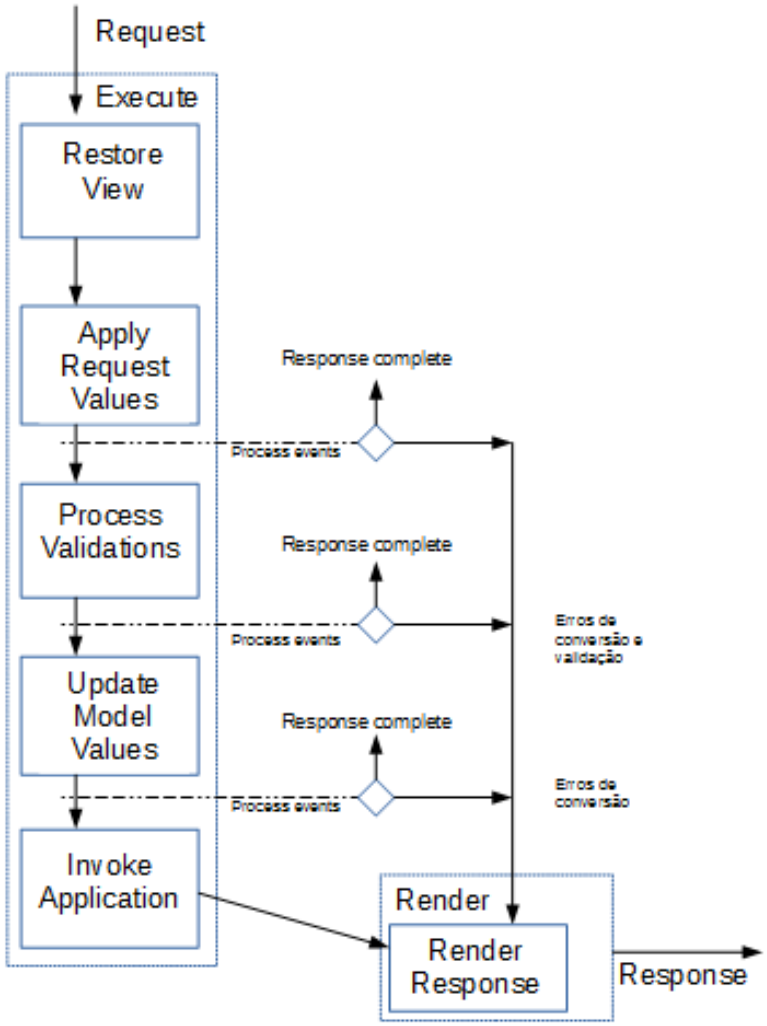


Ilustração do Ciclo de vida JSF 2.0.

4.4.1.1 Restore view

Essa é a fase inicial de uma requisição de página JSF que pode ser feita através de um link ou botão pressionado.

Aqui a JSF recria toda a árvore de componentes da página(UIViewRoot), dispara os validadores nestes componentes e atualiza dentro do objeto FacesContext, lembrando que todos os componentes podem acessá-lo.

4.4.1.2 Apply Request Values

Essa é a fase onde a árvore de componentes da página é restaurada com os valores iniciais enviados na requisição. É através da requisição que os valores iniciais são obtidos, sendo aplicada ao componente em memória e não à propriedade da classe Bean para a qual o componente aponta.

Todo componente da UIViewRoot analisa o valor passado para ele dentro do pedido HTTP, deixando armazenado localmente. Uma mensagem é armazenada no FacesContext, caso haja um erro para posterior exibição.

Logo após, podem ocorrer situações que levem a renderização imediata da página ou ao término da resposta. Existem listeners que são avisados quando estes eventos ocorrem.

4.4.1.3 Process validations

Esse é o momento em que o JSF processa todos os “validators” que estão registrados, que possuem a configuração do campo com o atributo required=”true” para cada componente na UIViewRoot.

A preocupação aqui é se todas as regras estão corretamente aplicadas aos valores da fase anterior. Também podem gerar erros de validação e conversão nesta fase, gerando mensagens de erros.

4.4.1.4 Update model values

Nessa fase todos os valores já foram registrados e validados nos componentes de atribuição da propriedade da classe Bean. O backing bean é o modelo associado à página. O JSF percorre cada componente e tenta aplicar o valor local à propriedade do bean associado, convertendo assim o valor, se necessário.

Também podemos criar nossos conversores ou utilizar os padrões. Se não for possível converter o valor, o ciclo será direcionado para a fase Render Response e exibirá as mensagens de erro.

4.4.1.5 Invoke application

Se existir algum evento de aplicação, como por exemplo o envio de um formulário HTTP, ou mesmo o acionamento de outra página JSF, então é processado neste momento.

O JSF manipula qualquer nível de evento da aplicação, tal como o envio de um formulário ou realizar chamada para outra página através de link. Assim que os valores estiverem validados, convertidos e atribuídos para as propriedades da classe Bean, se for o caso, o JSF adicionará o método da classe Bean que adicionou a requisição.

4.4.1.6 Render response

Essa é a última fase do ciclo de vida e nesse passo é exigido que a partir do momento que a página for criada e devolvida para o navegador, o JSF solicitará que todos os componentes de tela que tem suas propriedades, comportamento e forma, realizem a geração do próprio HTML.

Durante essa fase, o “render kit” é chamado para cada componente e a tecnologia de visualização é invocada. Podendo ser JSP ou Facelets, que vem como default na versão JSF 2.0.

Exercícios JSF:

14 – Criar uma aplicação que contenha um template de páginas, uma página de login, uma página de boas vindas. Se o login e senha estiverem corretos encaminhar para a página de boas vindas.

15 – Acrescentar ao exercício anterior uma condição que se for verdadeira encaminhe para a página de boas vindas, mas se for falsa mostra uma mensagem de erro na página de login, por exemplo se o login e senha forem inválidos mostrará uma mensagem.

Exercício Livre.

Criar um CRUD que realizará a manutenção de produtos e que conterá os seguintes atributos: id, código, tipo, qtde, valor, dataEntrada, descricao.

CAPÍTULO 5

RESOLUÇÃO DOS EXERCÍCIOS PROPOSTOS

1 – Criar um DynamicWebProject nomeando como Exercicio1, configure o arquivo web.xml, nomeando e configurando a Servlet corretamente. Crie a classe Java que chamará MinhaServlet configurando o que a Servlet executará quando for chamada.

****

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"              id="WebApp_ID"
version="2.5">
  <display-name>Exercicio1</display-name>
  <servlet>
    <servlet-name>NomeDaServlet</servlet-name>
    <servlet-class>br.com.elaborata.servlet.MinhaServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>NomeDaServlet</servlet-name>
    <url-pattern>/qualquer</url-pattern>
  </servlet-mapping>
</web-app>
```

****

MinhaServlet.java

```
package br.com.elaborata.servlet;


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;
```

```
/**
* Classe que representa a servlet que montará a página de internet com o
* seguinte conteúdo: Olá Boa Noite!!!
*/
public class MinhaServlet extends HttpServlet{

    private static final long serialVersionUID = 1L;

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
     resp.setContentType("text/html");

     PrintWriter out = resp.getWriter();

     out.println("<html>");
     out.println("    <body>");
     out.println("          <h1>Olá, Boa Noite!!!</h1>");
     out.println("    </body>");
     out.println("</html>");
    }

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
     doGet(req, resp);
    }
```

2 – Criar um DynamicWebProject nomeando como Exercicio2, configure o arquivo web.xml, nomeando e configurando a Servlet corretamente. Crie a classe Java configurando a Servlet para receber uma condição que se for verdadeira retorne uma página jsp chamada de pagina1.jsp, senão uma página jsp chamada pagina2.jsp.

****

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"              id="WebApp_ID"
version="2.5">
  <display-name>Exercicio1</display-name>
  <servlet>
    <servlet-name>NomeDaServlet</servlet-name>
    <servlet-class>br.com.elaborata.servlet.MinhaServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>NomeDaServlet</servlet-name>
    <url-pattern>/qualquer</url-pattern>
  </servlet-mapping>
</web-app>
```

****

MinhaServlet.java

```
package br.com.elaborata.servlet;


import java.io.IOException;


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

```
/**
 *
 */
public class MinhaServlet extends HttpServlet{

    private static final long serialVersionUID = 1L;

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
     String condicao = "true";
     String redir;

     if("true".equals(condicao)){
           redir = "/Exercicio2/pagina1.jsp";
     } else {
           redir = "/Exercicio2/pagina2.jsp";
     }
     resp.sendRedirect(redir);
    }

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
     doGet(req, resp);
    }
}
```

****

pagina1.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<title>Pagina 1</title>
<body>
    Olá, você está na pagina 1
</body>
</html>
```

****

pagina2.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<title>Pagina 2</title>
<body>
    Olá, você está na pagina 2
</body>
</html>
```

3 – Criar um DynamicWebProject nomeando como Exercicio3, composto por um formulário de cadastro de contatos, onde após o envio do cadastro, seja mostrado uma página com os contatos gravados. Após a montagem do projeto deve ser feita a reorganização dos imports.

****

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    id="WebApp_ID"
    version="2.5">
  <display-name>Exercicio3</display-name>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
  <servlet>
    <servlet-name>AdicionaContato</servlet-name>
    <servlet-class>br.com.elaborata.AdicionaContatoServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>AdicionaContato</servlet-name>
    <url-pattern>/adicionaContato</url-pattern>
  </servlet-mapping>
</web-app>
```

****

index.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!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=ISO-8859-1">
<title>Cadastro de contatos</title>
```

```
</head>

<body>

     <form action="adicionaContato" method="POST">

      Nome: <input type="text" name="nome"/><br/>

      Email: <input type="text" name="email"/><br/>

      Endereço: <input type="text" name="endereco"/><br/>


      <input type="submit" value="Gravar"/>

     </form>

</body>

</html>
```

****

Contato.java

```
package br.com.elaborata;


public class Contato {


     private String nome;

     private String email;

     private String endereco;


     public String getNome() {

      return nome;

     }

     public void setNome(String nome) {

      this.nome = nome;

     }

     public String getEmail() {

      return email;

     }

     public void setEmail(String email) {
```

```
      this.email = email;
    }
    public String getEndereco() {
      return endereco;
    }
    public void setEndereco(String endereco) {
      this.endereco = endereco;
    }
}
```

****

ContatoDAO.java

```
package br.com.elaborata;

public class ContatoDAO {
    List<Contato> contatos = new ArrayList<Contato>();
    public ContatoDAO() {
      // TODO Auto-generated constructor stub
    }

    public ContatoDAO(List<Contato> contatos){
      this.contatos = contatos;
    }

    public List<Contato> salva(Contato contato) {
      contatos.add(contato);
      return contatos;
    }
}
```

****

AdicionaContatoServlet.java

```
package br.com.elaborata;


public class AdicionaContatoServlet extends HttpServlet{


    private static final long serialVersionUID = 1L;
    ContatoDAO contatoDAO = new ContatoDAO(new ArrayList<Contato>());


    @Override
    protected void service(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
     PrintWriter out = response.getWriter();
     List<Contato> contatos = new ArrayList<Contato>();
     String nome = request.getParameter("nome");
     String email = request.getParameter("email");
     String endereco = request.getParameter("endereco");


     Contato contato = new Contato();
     contato.setNome(nome);
     contato.setEmail(email);
     contato.setEndereco(endereco);
     contatos= contatoDAO.salva(contato);
     if(!contatos.isEmpty()) {
           out.println("<html>");
           out.println("<body>");
           out.println("Contato adicionado: </br>");
           for(Contato cont : contatos) {
                 out.println(cont.getNome()+"</br>");
           }
           out.print("<a            href=\"javascript:window.history.go(-1)\"
>Voltar</a>");
           out.println("</body>");
           out.println("</html>");
```

```
      }
    }
}
```

4 – Criar uma página jsp que mostre o resultado de 6 (seis) operações matemáticas e 1 (um) número aleatório entre 0 e 100 como no exemplo abaixo:

2 + 2 = 4

13 – 7 = 6

8 x 3 = 24

26 ÷ 4 = 6,5

3 elevado a 4 potência é 81

A raiz quadrada de 81 é 9

O número aleatório é 53

****

exercicio4.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!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=ISO-8859-1">
<title>Exercicio 4</title>
</head>
<body>
    2 + 2 = <%= 2 + 2 %> <br>
    13 - 7 = <%= 13 - 7 %> <br>
```

```
    8 x 3  =   <%= 8 * 3 %>              <br>
    26 &divide 4 =   <%= 26 / 4 %> <br>
    <!-- &divide é comando HTML para sinal de dividir     -->

    3 elevado à 4 potência é <%= Math.pow(3, 4) %>   <br>
    A raiz quadrada de 81 é <%= Math.sqrt(81) %>     <br>
    O número aleatório é <%= Math.random()*100 %>    <br>
</body>
</html>
```

5 – Crie uma JSP que leia um parâmetro “nome” e escreva o conteúdo na página(expressions).

****

exercicio5.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!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=ISO-8859-1">
<title>Exercicio 5</title>
</head>
<body>
    <!--
    Para testar:
    http://localhost:8080/projeto1/Exercicio5.jsp?nome=ALUNO
     -->

    <%
    String nome = request.getParameter("nome");
    %>
```

```
    Nome: <%= nome %>
</body>
</html>
```

6 – Crie uma JSP que leia 2 (dois) parâmetros “n1, n2” e mostre o resultado da soma entre os dois na página(request).

****

exercicio6.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!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=ISO-8859-1">
<title>Exercicio 6</title>
</head>
<body>
<!--
Para testar:
http://localhost:8080/Projeto/exercicio6.jsp?valor1=100&valor2=50
 -->
    <%// Request ### SEMPRE ### soma uma String
    String valor1string = request.getParameter("valor1");
    String valor2string = request.getParameter("valor2");
    int valor1Int = Integer.parseInt(valor1string);
    int valor2Int = Integer.parseInt(valor2string);
    int resultado = valor1Int + valor2Int;
    %>
    SOMA de<%=valor1string%>+<%=valor2string%>=<%=resultado%>
</body>
</html>
```

7 – Crie uma JSP que gere um número aleatório, exiba e diga em seguida se o número gerado é par ou ímpar. Modifique a JSP para que quando o número gerado for par seja apresentado em azul e quando ímpar seja apresentado em vermelho. Crie uma JSP que leia 3 parâmetros (2 números e uma operação) e mostre na página o resultado do calculo.

****

exercicio7_1.jsp

```
<html>
<body>
     <%
      int numero_aleatorio = (int) (Math.random()*100);
     %>
     <br> O número aleatório é
     <%= numero_aleatorio %>
     <%= numero_aleatorio%2==0?"PAR":"IMPAR" %>
     <br>
</body>
</html>
```

****

exercicio7_2.jsp

```
<html>
<body>
     <%
      int numero_aleatorio = (int) (Math.random()*100);
     %>
     <br> O número é:
     <%= numero_aleatorio%2==0?"<font
      color='blue'>"+numero_aleatorio+"</font>":"<font
     color='red'>"+numero_aleatorio+"</font>"  %>
     <br>
</body>
</html>
```

****

exercicio7_3.jsp

```
<html>
<body>
     <%
     int n1 = Integer.parseInt( request.getParameter("n1"));
     int n2 = Integer.parseInt( request.getParameter("n2"));
     String operacao = request.getParameter("op");
     int resultado = 0;
     %><br>
     <%
     if (operacao.equals("a"))
      resultado = n1 + n2;
     else if (operacao.equals("s"))
         resultado = n1 - n2;
     else if (operacao.equals("m"))
      resultado = n1 * n2;
     else if (operacao.equals("d"))
      resultado = n1 / n2;
     %>
     O resultado é :    <%= resultado %>  <br>
</body>
</html>
```

8 – Crie 2 JSPs “cabecalho.jsp e rodape.jsp” e as inclua dentro de uma terceira página chamada “conteudo.jsp“ onde será mostrada a data atual do sistema.

****

exercicio8_cabecalho.jsp

```
<!--  CABECALHO -->
<table width="400" height="50" bgcolor="bbffee">
     <tr>
```

```
      <td width="100%">
            Cabeçalho: Exercicio_diretivas_cabeçalho.jsp
      </td>
     </tr>
     <tr>
      <td width="100%"><hr></td>
     </tr>
</table>
```

****

exercicio8_rodape.jsp

```
<!--  RODAPE -->
<table width="400" height="50" bgcolor="aaffee">
     <tr>
      <td width="100%"><hr></td>
     </tr>
     <tr>
      <td width="100%">
            Rodapé: Exercicio_diretivas_rodape.jsp
      </td>
     </tr>
</table>
</body>
</html>
```

****

exercicio8_conteudo.jsp

```
<!--  CONTEUDO  -->
<html>
<body>
     <%@ include file="exercicio8_cabecalho.jsp"  %>

     <table width="400" height="100" bgcolor="eeeeee">
```

```
      <tr>
            <td width="100%">Conteúdo da página:</td>
      </tr>
      <tr>
            <td width="100%">Exercicio_diretivas_conteudo.jsp</td>
      </tr>
     </table>
     <%@ include file="exercicio8_rodape.jsp"  %>
```

9 – Modifique o exercício anterior para que a data atual do sistema seja retornada através do método.

****

exercicio9.jsp

```
<%@page import="java.util.Date"%>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
     pageEncoding="ISO-8859-1"%>
<!DOCTYPE    html    PUBLIC    "-//W3C//DTD    HTML    4.01    Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<body>
     <%!
     int visitas = 0;
     void novaVisita() {
      System.out.println("Dntro do método");
      visitas++;
     }
     Date data = new Date();
     Date getData() {
      System.out.println("Data...dentro do metodo getData()");
      return new Date();
     }
     Date getDataCorrente() {
      return new Date();
```

```
    }
    Date updateData() {
     return new Date();
    }
%>
</body>
Olá, você é o visitante
<%= visitas %>
<% novaVisita(); %>
<%@include file="exercicio8_cabecalho.jsp"%>
Hora início:
    <%= getData() %>
    <br> hora atual.:
    <%= getDataCorrente() %>
    <br>
    <%= updateData() %>
    horario final:
    <%= getData() %>
</html>
```

10 – Crie uma JSP que receba um número como parâmetro e caso seja par redirecione para uma página “par.jsp” ou senão redirecione para “impar.jsp”.

****

exercicio10.jsp

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Exercicio10</title>
</head>
<body>
    <%String valorString = request.getParameter("meuparametro");
     int valorInt = Integer.parseInt(valorString);
```

```
      String paginadestino = "";
      if (valorInt % 2 == 0) {
            paginadestino = "Exercicio10_par.jsp";
      } else {
            paginadestino = "Exercicio10_impar.jsp";
      }
     %>
     <jsp:forward page="<%=paginadestino%>">
      <jsp:param value="<%=valorInt%>" name="meuparametro" />
     </jsp:forward>
</body>
</html>
```

****

exercicio10_par.jsp

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>PÁGINA: PAR</title>
</head>
<body>
     <br>   PÁGINA: PAR
     <br>   Resultado passado é <%= request.getParameter("meuparametro") %>
</body>
</html>
```

****

exercicio10_impar.jsp

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
```

```
<title>PÁGINA: IMPAR</title>
</head>
<body>
    <br> PÁGINA: IMPAR
    <br> Resultado passado é <%= request.getParameter("meuparametro") %>
</body>
</html>
```

11 – Crie um JSP com um formulário para registro de usuários, após o preenchimento devera ser exibido um termo de licença com as opções “Aceitar, Recusar”, após a escolha do usuário deverá ser exibida uma página exibindo os dados informados pelos usuários.

****

escopo 1

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<body>
    <form method="post" action="Exercicio_11_escopo_2.jsp">
     Nome.: <input type="TEXT" name="nome" size="20"><br>
     Email: <input type="TEXT" name="mail" size="20"><br>
     Idade: <input type="TEXT" name="idade" size="3">
     <input type="submit" value="Enviar">
    </form>
</body>
</html>
```

****

escopo 2

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<%
     String nome  = request.getParameter( "nome" );
     String mail  = request.getParameter( "mail" );
     String idade = request.getParameter( "idade" );

     session.setAttribute( "seuNome" , nome  );
     session.setAttribute( "seuMail" , mail  );
     session.setAttribute( "suaIdade", idade );
%>
<html>
<body>
     Termo de Licença:
     <br>
     <hr>
     <br><br>
     Nome: <%= request.getParameter("nome")  %><br>
     Email: <%= request.getParameter("mail")  %><br>
     Idade: <%= request.getParameter("idade") %><br>
     <hr>
     <form method="post" action="Exercicio_08_escopo_3.jsp">
      <input type="radio" name="OpcaoRadio" value="ACEITO"/> Aceitar
      <input type="radio" name="OpcaoRadio" value="RECUSADO"/> Recusar

      <input type="submit" value="Enviar">
     </form>
</body>
</html>
```

****

escopo 3

```
<HTML>
<BODY>
Registro de:
<br>
<br>
<font face="courier">
    Nome.: <%= session.getAttribute( "seuNome" ) %><br>
    Email: <%= session.getAttribute( "seuMail" ) %><br>
    Idade: <%= session.getAttribute( "suaIdade") %>
</font>
<br>
<br>
<font size="3">
<b>
    Termo de Licença: <%= request.getParameter("OpcaoRadio")  %>
</b>
</font>
</BODY>
</HTML>
```

12 – Modifique o exercício anterior para que o mesmo utilize beans para armazenar as informações.

****

exercicio12_escopo_1.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
```

```
<body>
     <form method="post" action="exercicio12_escopo_2.jsp">
      <font face="Courier New" size="2">
      Nome..: <input type="TEXT" name="name"  size="50">    <br>
      Email.: <input type="TEXT" name="email" size="50">    <br>
      Idade.: <input type="TEXT" name="idade" size="3">
      </font>
      <input type="submit" value="Enviar">
     </form>
</body>
</html>
```

****

exercicio12_escopo_2.jsp

```
<%@page import="beans.Usuario"%>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>

<%
     String name  = request.getParameter( "name"  );
     String email = request.getParameter( "email" );
     String idade = request.getParameter( "idade" );

     Usuario user = new Usuario();

     user.setNome(name);
     user.setEmail(email);
     user.setIdade(Integer.parseInt(idade));
     // Todos os dados são enviados para a SESSAO.
     session.setAttribute( "usuario" , user  );
%>
```

```
<html>
<body>
     TERMO LICENÇA:
     <br>
     <hr>
     <br>
     <font face="Courier New" size="2">
      Nome: <b> <%= user.getNome()    %>   </b><br>
      Email:<b> <%= user.getEmail()   %>   </b><br>
      Idade:<b> <%= user.getIdade()   %>   </b><br>
     </font>
     <hr>
     <form method="post" action="exercicio12_escopo_3.jsp">
      <input type="radio" name="OpcaoRadio" value="ACEITO"/> Aceitar
      <input type="radio" name="OpcaoRadio" value="RECUSADO"/> Recusar
      <input type="submit" value="Enviar">
     </form>
</body>
</html>
```

****

exercicio12_escopo_3.jsp

```
<%@page import="beans.Usuario" %>


<HTML>
<BODY>
     <font face="Courier New" size="2"><b>
      TERMO DE LICENÇA DE: </b>
     </font>
     <br><br>
     <% Usuario user = (Usuario)session.getAttribute("usuario"); %>
     <font face="Courier New" size="2">
```

```
    Nome.: <%= user.getNome()       %> <br>
    Email: <%= user.getEmail()   %> <br>
    Idade: <%= user.getIdade()   %> <br>
    </font>
    <br><br>
    <font size="5">
    <b>
     Termo de Licença: <%= request.getParameter("OpcaoRadio")   %>
    </b>
    </font>
</BODY>
</HTML>
```

13 – Crie uma página JSP onde possa ser informado as medidas da base e altura de um triângulo. Guarde as informações em um bean e utilize EL para mostrar os valores informados e o valor calculado da área em outra página.

****

Classe Triangulo

```
package beans;

public class Triangulo {

    private int base;
    private int altura;
    private int area;

    public int getBase() {
     return base;
    }
    public void setBase(int base) {
     this.base = base;
    }
```

```
    public int getAltura() {
     return altura;
    }
    public void setAltura(int altura) {
     this.altura = altura;
    }
    public int getArea() {
     return area;
    }
    public void setArea(int area) {
     this.area = area;
    }

}
```

****

exercicio13_expression_language.jsp

```
<jsp:useBean id="triangulo" class="beans.Triangulo" scope="session" />
<jsp:setProperty name="triangulo" property="base"    value = "50" />
<jsp:setProperty name="triangulo" property="altura" value = "5"  />

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<html>
<body>
    <jsp:setProperty      name="triangulo"      property="area"      value="$
    {triangulo.base*triangulo.altura/2}"/>

    A     área     do     triângulo     é:<jsp:getProperty     name="triangulo"
property="area"/><br>
    A área do triângulo é: ${triangulo.base * triangulo.altura / 2}<br>
    ${pageContext.request.queryString }<br>
```

```
    Parâmetro p1:  ${param["p1"]}
</body>
</html>
```

14 – Crie 2 (duas) TagLibs para efetuar conversões entre Real e Dollar. A tag deve receber 2 parametros, “cotacao” e “valor”, e caso aconteça algum erro deverá ser exibida uma mensagem configurada através do conteúdo da tag.

****

exercicio14.jsp

```
<%@ taglib prefix="moeda" uri="WEB-INF/ex12_tags.tld"%>
<html>
  <body>
    Dollar p/ Real:
    <moeda:DollarParaReal valor="13" cotacao="a">
     Ocorreu um erro na conversão de Dollar p/ Real.
    </moeda:DollarParaReal><br>
     Real p/ Dollar:
    <moeda:RealParaDollar valor="3.00" cotacao="2.00">
     Ocorreu um erro na conversão de Real p/ Dollar.
    </moeda:RealParaDollar>
  </body>
</html>
```

****

DollarRealTag.java

```
package taglibs;

import java.io.IOException;
import java.io.StringWriter;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
```

```
import javax.servlet.jsp.tagext.SimpleTagSupport;


public class DollarRealTag extends SimpleTagSupport {

    private Double valor;

    private String cotacao;

    private StringWriter sw = new StringWriter();


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

    public void setCotacao(String cotacao) {this.cotacao = cotacao;}

    @Override

    public void doTag() throws JspException, IOException {

     JspWriter out = getJspContext().getOut();

     if (valor != null && cotacao != null) {

           try {

                 out.println(valor*Double.parseDouble(cotacao));

           }

           catch(Exception e) {

                 getJspBody().invoke(sw);

                 out.println("<font          color=\"#ff0000\">"+sw.toString()
+"</font>");

           }

     } else {

           getJspBody().invoke(sw);

           out.println(sw.toString());

     }

    }
}
```

****

RealDollarTag.java

```
package taglibs;


import java.io.IOException;

import java.io.StringWriter;

import javax.servlet.jsp.JspException;

import javax.servlet.jsp.JspWriter;

import javax.servlet.jsp.tagext.SimpleTagSupport;


public class RealDollarTag extends SimpleTagSupport {

    private Double valor;

    private Double cotacao;

    private StringWriter sw = new StringWriter();


    public void setValor(Double valor) {

     this.valor = valor;

    }

    public void setCotacao(Double cotacao) {

     this.cotacao = cotacao;

    }


    @Override

    public void doTag() throws JspException, IOException {

     JspWriter out = getJspContext().getOut();

     if (valor != null && cotacao != null) {

           try {

                 out.println(valor/cotacao);

           }

           catch(Exception e) {

                 getJspBody().invoke(sw);
```

```
                out.println(sw.toString());
            }
        } else {
            getJspBody().invoke(sw);
            out.println(sw.toString());
        }
    }
}
```

****

web.xml **Pasta WEB-INF

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"              id="WebApp_ID"
version="2.5">
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

****

exercicio12_tags.tld **Pasta WEB-INF

```
<taglib>
    <tlib-version>1.0</tlib-version>
    <jsp-version>2.0</jsp-version>
    <short-name>Exercicio 12</short-name>
    <tag>
      <name>DollarParaReal</name>
      <tag-class>taglibs.DollarRealTag</tag-class>
      <body-content>scriptless</body-content>
      <attribute>
            <name>valor</name>
```

```
            <type>Double</type>
      </attribute>
      <attribute>
            <name>cotacao</name>
      </attribute>
    </tag>
    <tag>
      <name>RealParaDollar</name>
      <tag-class>taglibs.RealDollarTag</tag-class>
      <body-content>scriptless</body-content>
      <attribute>
            <name>valor</name>
            <type>Double</type>
      </attribute>
      <attribute>
            <name>cotacao</name>
            <type>Double</type>
      </attribute>
    </tag>
</taglib>
```

14 – Criar uma aplicação que contenha um template de páginas, uma página de login, uma página de boas vindas. Se o login e senha estiverem corretos encaminhar para a página de boas vindas.

****

faces-config.xml

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

<faces-config
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
                        xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-facesconfig_2_0.xsd"
    version="2.0">
     <navigation-rule>
      <display-name>login.xhtml</display-name>
      <from-view-id>/login.xhtml</from-view-id>
      <navigation-case>
            <from-action>#{loginBean.logar}</from-action>
            <from-outcome>ok</from-outcome>
            <to-view-id>/welcome.xhtml</to-view-id>
      </navigation-case>
     </navigation-rule>

</faces-config>
```

****

template_basico.xhtml

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
          "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets">
<head>
  <title><ui:insert name="title">Facelets Tutorial</ui:insert></title>
</head>
<body>
<div id="header">
    <ui:insert name="header">
        <ui:include src="/WEB-INF/templates/header.xhtml"/>
    </ui:insert>
</div>
<div id="content">
    <ui:insert name="content"/>
```

```
</div>


<div id="footer">
    <ui:insert name="footer">
        <ui:include src="/WEB-INF/templates/footer.xhtml"/>
    </ui:insert>
</div>
</body>
</html>
```

****

header.xhtml

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<body>
<div style="width:100%;font-size:36px;line-height:48px;background-
color:navy;color:white">Minha aplicação JSF</div>
</body>
</html>
```

****

footer.xhtml

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
</head>
<body>
```

```
    <h3 style="border-top:thick solid #000000;">Rodapé</h3>
</body>
</html>
```

****

index.jsp

```
<% response.sendRedirect("login.jsf"); %>
```

****

login.xhtml

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:ui="http://java.sun.com/jsf/facelets"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:f="http://java.sun.com/jsf/core">
<ui:composition template="/WEB-INF/templates/template_basico.xhtml">
    <ui:define name="content">
     <h:form>
          <h:panelGrid columns="2">
               <h:outputLabel value="Login:" />
               <h:inputText id="login" value="#{loginBean.login}"  />
               <h:outputLabel value="Senha:" />
               <h:inputSecret   id="senha"   value="#{loginBean.senha}"   />
 <br />
          </h:panelGrid>
          <h:commandButton action="#{loginBean.logar}" value="Logar" />
     </h:form>
    </ui:define>
</ui:composition>
</html>
```

****

LoginBean.java

```
package br.com.elaborata;

import javax.faces.bean.ManagedBean;

import javax.faces.bean.SessionScoped;


@ManagedBean(name="loginBean")

@SessionScoped

public class LoginBean {

    private String login;

    private String senha;

    public Object logar() {

     if("aluno".equals(getLogin()) & "123".equals(getSenha())){

           setMensagem("");

           return "ok";

     }

     return null;

    }

    public Object voltar() {

     setMensagem("");

     return "login";

    }

    public String getLogin() {

     return login;

    }

    public void setLogin(String login) {

     this.login = login;

    }

    public String getSenha() {

     return senha;
```

```
    }
    public void setSenha(String senha) {
     this.senha = senha;
    }
}
```

****

welcome.xhtml

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:ui="http://java.sun.com/jsf/facelets"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:f="http://java.sun.com/jsf/core">
<ui:composition template="WEB-INF/templates/template_basico.xhtml">
    <ui:define name="header">
     <h3 style="border-bottom:thick solid #000000;">Cabeçalho</h3>
    </ui:define>
    <ui:define name="content">
    <h:form>
     <br/><h1><h:outputText value="Bem vindo
#{loginBean.login}!!!"/></h1> 
     <br/>
     <h:commandButton action="#{loginBean.voltar}" value="Voltar" />
    </h:form>
    </ui:define>
    <ui:define name="footer">
     <h3 style="border-top:thick solid #000000;">Rodapé</h3>
    </ui:define>
</ui:composition>
</html>
```

15 – Acrescentar ao exercício anterior uma condição que se for verdadeira encaminhe para a página de boas vindas, mas se for falsa mostra uma mensagem de erro na página de login, por exemplo se o login ou senha forem inválidos mostrará uma mensagem.

****

login.xhtml

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
     xmlns:ui="http://java.sun.com/jsf/facelets"
     xmlns:h="http://java.sun.com/jsf/html"
     xmlns:f="http://java.sun.com/jsf/core">
<ui:composition template="/WEB-INF/templates/template_basico.xhtml">
     <ui:define name="content">
      <h:form>
            <h:outputLabel value="#{loginBean.mensagem}" /><br />
            <h:panelGrid columns="2">
                  <h:outputLabel value="Login:" />
                  <h:inputText id="login" value="#{loginBean.login}"  />
                  <h:outputLabel value="Senha:" />
                  <h:inputSecret     id="senha"     value="#{loginBean.senha}"
/> <br />
            </h:panelGrid>
            <h:commandButton action="#{loginBean.logar}" value="Logar" />
      </h:form>
     </ui:define>
</ui:composition>
</html>
```

****

LoginBean.java

```
package br.com.elaborata;


import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;
@ManagedBean(name="loginBean")
@SessionScoped
public class LoginBean {
    private String login;
    private String senha;
    private String mensagem;

    public String getMensagem() {
     return this.mensagem;
    }
    public void setMensagem(String msg) {
     this.mensagem = msg;
    }
    public Object logar() {
     if("roque".equals(getLogin()) & "123".equals(getSenha())){
           setMensagem("");
           return "ok";
     } else {
           setMensagem("Verifique Login ou Senha!!!");
           return null;
     }
    }
    public Object voltar() {
     setMensagem("");
     return "login";
    }
    public String getLogin() {
```

```
      return login;
    }
    public void setLogin(String login) {
      this.login = login;
    }
    public String getSenha() {
      return senha;
    }
    public void setSenha(String senha) {
      this.senha = senha;
    }
}
```

CAPÍTULO 6

REFERÊNCIAS BIBLIOGRÁFICAS

- https://pt.wikipedia.org/wiki/Java_(linguagem_de_programação)
- https://www.novatec.com.br/livros/javaparaweb/capitulo9788575222386.pdf
- http://fabrica.ms.senac.br/2013/06/o-que-e-jsf-java-server-faces/
- https://pablonobrega.wordpress.com
- http://www.devmedia.com.br/
- https://docs.jboss.org/weld/reference/1.1.5.Final/pt-BR/html/intro.html
- http://www.devmedia.com.br/ciclo-de-vida-do-javaserver-faces-jsf/27893
- COSTA, Aécio, Java para desenvolvimento WEB.
- ALVIM, Paulo, Tirando o máximo do Java EE 5 Open Source. 2. ed. Belo Horizonte -MG 641p.
- Sampaio, Cleuton, Java Enterprise Edition 6: desenvolvendo aplicações corporativas / Cleuton Sampaio: prefácio de BrYan Basham. - Rio de Janeiro: Brasport, 2011
- Coelho, Hérbert, Casa do Código, JSF Eficaz As melhores práticas para o desenvolvedor web Java.

Rua Monsenhor Celso, 256 - 1º andar - Curitiba – Paraná

41.3324.0015

41.99828.2468

Agradecimentos:

Equipe Elaborata Informática