Universidade Federal da Bahia

Instituto de Matemática

Departamento de Ciência da Computação

Disciplina Linguagem de Aplicação Comercial

Alexandre Silva

Anderson Martiniano

Tiago Trocoli Leite de Souza

Trabalho Semestral: Java Server Faces

Salvador-Ba

28 de abril de 2009

Alexandre Silva

Anderson Martiniano

Tiago Trocoli Leite de Souza

Trabalho Semestral: Java Server Faces

Atividade apresentada como requisito para avaliação no curso de Ciência da Computação da Universidade Federal da Bahia na disciplina Laboratório de Aplicação Comercial orientada pelo professor doutor Adonai Medrado.

Prof. Adonai Medrado

Salvador-Ba

28 de abril de 2009

RESUMO

Este trabalho apresenta informações a respeito do framework Java Server Faces. A motivação que a criou, licença, exemplo de aplicação. Encontra-se nesse trabalho informações sobre a linguagem de programação Java.

ABSTRACT

This work presents information about the Java Server Faces framework. The motivation that created it, its license, application example. It has information about Java language too.

SUMÁRIO

1 INTRODUÇÃO

Desenvolver aplicações em tempo menores, e atender a requisitos de qualidade, manutenabilidade e usabilidade são fundamentais para um software comercial bem sucedido.

Pensando nisso, comunidades de desenvolvedores e empresas aprimoram e criam ferramentas. Essas ferramentas variam de modelos de projetos até novas linguagens ágeis.

Java Server Faces é uma dessas ferramentas desenvolvida. Pensada em conjunto por várias empresas e comunidades, esse framework veio pra atender o modelo MVC em aplicações web e resolver problemas de codificação em páginas, o que, anteriormente, dificultava a manutenção.

Baseada na linguagem Java, o Java Server Faces tem uma linguagem fácil e prática. E possui uma série de vantagens que serão listadas e comentadas nos itens presentes neste artigo.

2. JAVA SERVER FACES

2.1 MOTIVAÇÃO

Com a necessidade de desenvolver páginas dinâmicas, que modificam todo seu conteúdo ou apenas partes da página, muitas tecnologias foram desenvolvidas para atender a esse novo modelo.

O Java Server Pages[1] foi uma das tecnologias criadas. Porém ela era um pouco inconivente, pois se aplicava código Java junto com HTML na página. Isso dificultava o desenvolvimento, já que o desenvolvedor responsável pela interface gráfica e o desenvolvedor da lógica por trás da página teriam de saber Java. Outro problema apresentado era não poder modularizar o sistema em camadas.

Com o surgimento do modelo MVC(Model View Control)[2], muitos frameworks surgiram, entre eles o JSF (Java Server Faces)[3].

2.2 CARACTERISTICAS

O JSF é uma tecnologia que incorpora características de um framework MVC para web e de um modelo de interfaces gráficas baseado em eventos. Por basear-se no padrão de projeto MVC, uma de suas melhores vantagens é a clara separação entre a visualização e regras de negócio (modelo).

No JSF, o controle é composto por um servlet denominado *FacesServlet*, por arquivos de configuração e por um conjunto de manipuladores de ações e observadores de eventos. O *FacesServlet* é responsável por receber requisições da WEB, redirecioná-las para o modelo e então remeter uma resposta. Os arquivos de configuração são responsáveis por realizar associações e mapeamentos de ações e pela definição de regras de navegação. Os manipuladores de eventos são responsáveis por receber os dados vindos da camada de visualização, acessar o modelo, e então devolver o resultado para o *FacesServlet*.

O modelo representa os objetos de negócio e executa uma lógica de negócio ao receber os dados vindos da camada de visualização. Finalmente, a visualização é composta por *component trees*(hierarquia de componentes), tornando possível unir um componente ao outro para formar interfaces mais complexas. A Figura 1 mostra a arquitetura do Java Server Faces baseada no modelo MVC.[4]

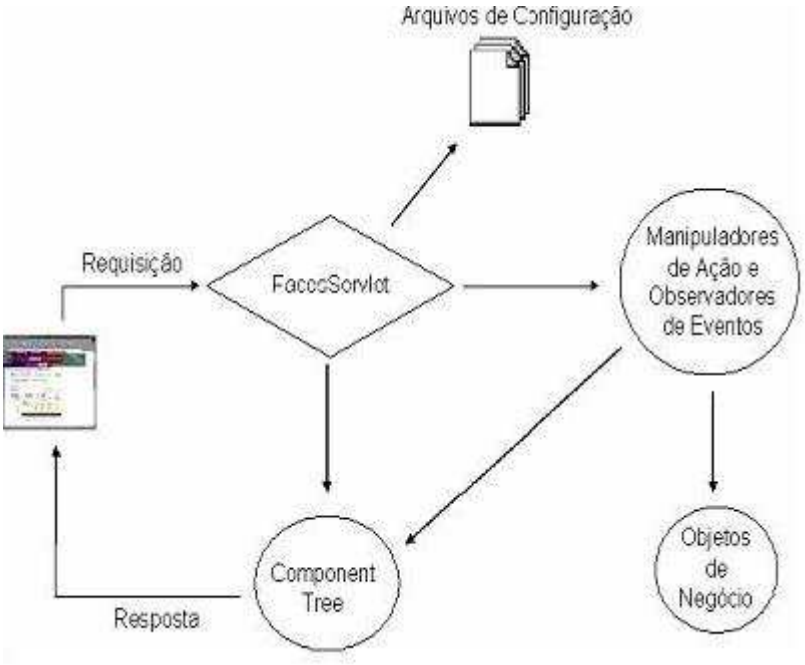

Figura 1

JavaServer Faces oferece ganhos no desenvolvimento de aplicações WEB por diversos motivos:

- Personalização de componentes. É possível fazer sua própria biblioteca de componentes;
- Fornece separação de funções que envolvem a construção de aplicações WEB;
- Reusa componentes da página;
- Suporte a eventos;
- Mecanismos de validação e conversão;
- Suporte a criação de componentes;
- Suporte a internacionalização.

2.3 LICENÇA

A propriedade intelectual (Copyright) do JSF é da SUN[5]. Porém ela não priva qualquer pessoa de ter acesso a essa especificação, podendo qualquer um implementá-la.

Existe uma série de regras que o desenvolvedor que implementa a especificação do JSF deve cumprir. Como não é nosso foco a implementação do JSF, vamos apenas citar a mais importante das regras que é: “Qualquer implementação do JSF não pode ter fins comerciais”. Ou seja, as implementações devem possuir uma licença gratuita.

Um exemplo de implementação do JSF é a MyFaces[6], da Apache Software Foundation[7]. Ela está sob a licença General Public License (GPL)[8].

Pela GPL, o desenvolvedor tem a liberdade de executar o programa para qualquer propósito, estudar como o programa funciona e adaptá-lo às suas necessidades. O acesso ao código-fonte é um pré-requisito para essa liberdade, assim como redistribuir cópias, e aperfeiçoar o programa, liberando os seus aperfeiçoamentos, de modo que outras pessoas também sejam beneficiadas.

2.3.1 Regras da Licença

As regras mais importantes da licença são:

- O código fonte tem que estar disponível.
- As modificações são regidas pela mesma licença
- Diferenciar as modificações feitas, identificando-se como contribuinte.

2.3.2 Obrigações do desenvolvedor

A licença é bastante flexível para o desenvolvedor que utiliza a tecnologia sob a GPL. Ele pode desenvolver um novo aplicativo e liberá-lo sobre outra licença.

2.4 PLATAFORMA

Para executar o JSF é necessário ter um servidor web para Java. Como os servidores web são feitos em Java também, é necessário ter a máquina virtual Java (JVM) para ser executados. A máquina virtual Java está disponível para as seguintes plataformas:

- Intel x86, x64
- Intel Intanium
- SPARC

A JVM está disponível para os seguintes sistemas operacionais:

- Linux
- Solaris
- Windows

2.5 EXEMPLO DE APLICAÇÃO

2.5.1 Instalando o Servidor de Aplicações

O servidor de aplicações sugerido é o Jboss 5.0.1.GA. Este servidor é bastante robusto e está sobre a especificação do J2EE 5, possuindo código aberto e gratuito. Suporta JSF, EJB 3, JPA, entre outras tecnologias. É mantida pela Red Hat e tem sua licença sob a GPL[9].

Para obter o servidor, basta ir ao site www.jboss.org e acessar a seção de downloads.

Como o Jboss é desenvolvido todo em Java, é independente do sistema operacional, basta ter a maquina virtual Java para ser executado.

2.5.2 Organização de Pastas

As pastas precisam ter uma organização definida. A figura 2 abaixo mostra o esquema de pasta como devem ser.

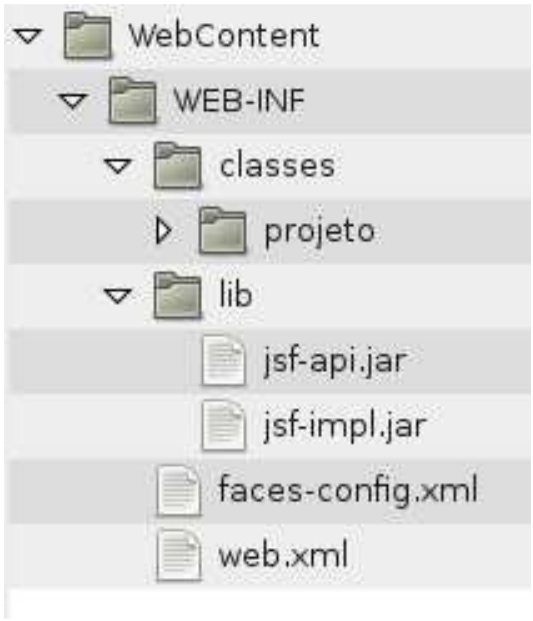

Figura 2

A pasta WebContent pode ter qualquer nome. Geralmente o nome dado a ela é o nome do projeto. WEB-INF é onde de fato ficam os arquivos de codificação de páginas web, as classes Java e as bibliotecas usadas que não são padrão do Java. A pasta classes deve conter apenas as classes Java e os seus pacotes. A pasta lib deve conter apenas as bibliotecas que não são padrão da máquina virtual ou que o servidor de aplicação não dê suporte.

Na pasta WEB-INF ainda existem dois arquivos XML. Esses arquivos são arquivos de configuração do JSF e definem como vai ser o seu comportamento. O web.xml é o arquivo de configuração do JSF, nele é definido o caminho do servlet que será executado, o nome da extensão que irá aparecer nas páginas, entre outras configurações que sejam necessárias. No faces-config.xml será definido o comportamento da aplicação JSF desenvolvida, como definição de navegação entre páginas e mapeamento de classes Java que irão manipular as páginas. Essas classes são chamadas de manager beans (MB). Na Figura 2, é possível visualizar dois arquivos dentro da pasta lib. Esses arquivos são as bibliotecas do JSF.

2.5.3 Criando Páginas

Para a aplicação exemplo serão criadas 4 paginas com a extensão JSP. As páginas serão index.jsp, menu.jsp, inserir.jsp, buscar.jsp.

A página index.jsp é apenas realiza o redirecionamento para menu.jsp. Ela se faz necessária por uma questão de padrão.

A página menu.jsp possui duas escolhas para ações diferentes.

A página inserir.jsp como o próprio nome já diz, insere um objeto em uma lista.

A página buscar.jsp possui uma tabela, que lista todos os objetos inseridos e faz uma busca pelo campo escolhido.

A codificação das páginas index.jsp, menu.jsp, inserir.jsp e buscar.jsp estão nos apêndices A.1, A.2, A.3 e A.4, respectivamente.

2.5.4 Criando manager bean

O manager bean, como dito anteriormente, é onde fica a lógica da página. Esse MB pode ser de três tipos diferentes:

- Session, quando se quer que o MB se mantenha com os dados carregados em memória.
- Request, quando não há a necessidade de manter os dados em memória.
- Application, semelhante à session, porém com algumas pequenas restrições.

Geralmente é feito um MB para cada página. No exemplo, por se tratar de algo simples, será necessário apenas um MB.

A codificação do MB.java está no apêndice A.5.

2.5.5 Configurando os XMLs

O arquivo web.xml irá receber configurações sobre o servlet, endereço raiz da aplicação e a extensão da página que será exibida.

O arquivo faces-config.xml irá possuir os dados das navegações entre as páginas, mapeamento e o escopo do MB. O faces-config.xml pode ter outras configurações, como a definição de variáveis globais na aplicação, novos componentes, entre outras.

A codificação do web.xml está na apêndice A.6 e a do faces-config.xml está no apêndice A.7.

2.5.6 Executando

Após ter executado os passos acima e ter posto todas as bibliotecas necessárias, basta agora criar o pacote com a aplicação. Esse pacote deve ser do tipo Web Archive (WAR) e deve conter a pasta WEB-INF. Agora basta executar o Jboss e copiar esse arquivo do tipo war para a pasta deploy que está dentro de default, dentro de server na raiz de da pasta onde foi instalado o Jboss.

No navegador web, acessamos o endereço http://localhost:8080/nome, onde nome se refere ao nome do arquivo do tipo war criado.

3 JAVA

Java[10] é uma plataforma de desenvolvimento orientada a objetos, utilizada em todos os principais segmentos da indústria, estando presente em uma ampla gama de dispositivos, computadores e redes. Java é multiplataforma, e para isso faz uso de uma máquiva virtual. Assim, o código é compilado para bytecodes, que por sua vez são executados na máquina virtual. Além do caráter multiplaforma, o Java traz um vasto ambiente de programação - o que permite aos programadores desenvolver aplicações de forma simples, utilizando o conjunto de bibliotecas (APIs) que acompanham a plataforma, ou adquirida com terceiros. Entre essas facilidades podemos citar a desalocação de memória automática através do coletor de lixo (garbage collector) e os recursos de internacionalização.

3.1 CONTEXTO DE CRIAÇÃO

Em 1991, a Sun Microsystems, percebendo o grande impacto que os microprocessadores tinham em dispositivos inteligentes eletrônicos voltados para o consumidor, financiou um projeto de pesquisa corporativa interna de nome Green[11], que resultou na criação de uma linguagem baseada em C++. Inicialmente chamada de Oak, o nome teve que ser trocado pois já havia uma linguagem com este nome. O nome sugerido então foi Java.

O projeto Green teve algumas dificuldades, e por pouco não foi cancelado, por conta do não desenvolvimento de dispositivos eletrônicos voltados para o mercado consumidor, como a Sun havia previsto, no início dos anos 90. Em 1993, com o advento da explosão de popularidade da World Wide Web, a equipe desenvolvedora viu o enorme potencial do Java para adicionar conteúdo dinâmico às páginas da web, dando nova vida ao projeto.

O Java foi então formalmente anunciado em 1995, e chamou bastante atenção por causa do enorme interesse na World Wide Web.

3.2 PESSOAS E EMPRESAS ENVOLVIDAS

A linguagem Java foi desenvolvida por uma equipe chefiada por James Gosling, da Sun Microsystems. Esta equipe, além de James, contava com mais cinco desenvolvedores, considerados alguns dos melhores da empresa.

3.3 VERSÕES E EVOLUÇÕES

A versão base do java surgiu em janeiro de 1996 com o nome de JDK 1.0, Em fevereiro de 1997 foi lançada a versão do JDK 1.1 que trouxe várias melhorias no AWT, além de “inner classes”, JavaBeans, RMI e JDBC. Em dezembro de 1998 aparece o J2SE 1.2, adicionando ao Java suporte a reflection, integração do Swing nas classes do core (núcleo da linguagem), incorporação do compilador JIT na JVM, java plugin, java IDL e Collections. O J2SE 1.3 surgiu em maio de 2000 e incluiu recursos como o RMI, Java Sound, JNDI (Java Naming and Directory Interface) e recursos para debug JPDA (Java Platform Debugger Architecture). Em fevereiro de 2002 surge o J2SE 1.4 que incluiu suporte de expressões regulares (baseado nas expressões regulares do Perl), NIO sem bloqueamento, API da logging, parser de XML e processador de XSLT (JAXP) integrados, além de incorporação de extensões de criptografia e segurança. O J2SE 5.0 surgiu em dezembro de 2006 e incluiu generics, meta-dados, autoboxing/unboxing, enumerações e varargs. A mais recente versão do java, chamada de Java SE 6 foi lançada em dezembro de 2006 e incluiu suporte de scripting, suporte para anotações, além de várias melhorias no Swing. O Java SE 7, está em desenvolvimento e estima-se que seja lançada em 2010 e incluirá melhorias e correção de bugs.

3.4 EXIGÊNCIAS EM CONCURSOS PÚBLICOS

Por ser uma plataforma vastamente usada em todo o mundo, o Java é bastante explorado em concursos públicos[12]. A maioria dos concursos atuais da área de TI exige conhecimentos da linguagem Java e do modelo de programação orientado a objetos que a linguagem ajudou difundir. No Apêndice B listamos e discutimos algumas questões cobradas em concursos públicos de algumas instituições brasileiras.

3.5 TIPO DE DESENVOLVIMENTO

A linguagem Java é interpretada e multiplataforma. Por esta razão, não é necessário recompilar o código para rodar a mesma aplicação quando houver mudança de sistema operacional.

3.6 SINTAXE

A sintaxe[13] do Java é derivada do C++, por isso se assemelha bastante a esta linguagem, como também à linguagem C#. A diferença para o C++ é o fato de

Java ser uma linguagem exclusivamente orientada a objetos[14]. Todo o código está dentro de uma classe e tudo são um objeto, com exceção dos números ordinais e reais, valores booleanos e caracteres, que não são classes por questão de desempenho.

3.6.1 Tipos básicos e declaração

Os tipos de dados em Java estão divididos em tipos primitivos e tipos por referência (tipos não primitivos).

Os tipos primitivos são boolean, byte, char, short, int, long, float e double. Uma variável desse tipo pode armazenar exatamente um valor de seu tipo por vez. As variáveis de instância de tipo primitivo têm um valor padrão de inicialização, as numéricas são inicializadas como 0, e a booleana é iniciada como false.

Todos os tipos não primitivos são tipos por referência, assim as classes, que especificam o tipo dos objetos, são tipos por referência. As variáveis desse tipo referenciam objetos no programa. Os objetos que são referenciados podem conter muitas variáveis de instância e métodos.

3.6.2 Declaração de estruturas de dados

A principal estrutura de dados de um software em java são as classes, já que toda a lógica de um programa deve estar dentro de uma classe. A classe é uma unidade básica em qualquer aplicação java. A declaração de uma classe em Java segue a seguinte sintaxe.

<modificador de acesso> class <nome da classe> [extends <classe pai>] [implements <interface>]

{

[<atributos>]

[<construtor>]

[<métodos>]

}

Bastante conhecidos os modificadores de acesso public/private controlam o acesso às variáveis e métodos da classe.

A parte envolvida por colchetes indica partes da declaração que são opcionais. Se uma classe faz uso de herança, a palavra chave extends deve ser usada seguida da classe pai. Se a classe deve implementar alguma interface, a palavra chave implements é usada seguida do nome da interface. Dentro do escopo da classe inserimos seus atributos, métodos e construtores, todos opcionais.

Para se criar uma instância de uma classe em java, um objeto, usamos a seguinte sintaxe.

<Nome da classe> <Nome do objeto> = new <Nome da classe>([<parametros de construtor>]);

Além das classes, também temos estruturas como os vetores, matrizes e enumerações.

Vetores são estruturas de dados que armazenam usualmente uma quantidade fixa de dados de um certo tipo. Segue sintaxe de declaração de vetores:

<tipo de dados do vetor> <nome do vetor>[ ] = new <tipo de dados do vetor>[<tamanho do vetor>];

exemplo:

```
int vetor[] = new int[10]; // declaração de um vetor
```

Matrizes são vetores multidimensionais e são declarados da seguinte maneira.

<tipo de dados do vetor> <nome do vetor>[ ][ ] = new <tipo de dados do vetor>[<tamanho do vetor][<tamanho do vetor interno>];

Exemplo:

```
int matriz[][] = new int[3][5];
```

Outra estrutura que o Java possui é a Enumeração, semelhante a outras linguagens como C/C++. Uma enumeração é um tipo especial de classe que é introduzida pela palavra chave enum e um nome do tipo. Dentro de uma enum devem ficar, separadas por vírgula, um conjunto de constantes, cada uma representando um valor único. Abaixo temos um exemplo de declaração.

```
Enum <nome da enum>{
        <lista de constantes>
}
```

Exemplo:

```
public enum Nivel {
        ADMIN, MOD, USER, BAN
}
```

3.6.3 Instruções condicionais

O java possui três tipos de instruções condicionais. São eles o if/else, o switch e o operador ternário (?:).

O comando if executa um trecho de código apenas uma vez, dependendo de sua condição. A sintaxe esta abaixo.

```
if ( <condição>  ) {
        <comandos>
 } else {
         <comandos>
 }
```

A condição é uma expressão da linguagem Java e pode ser qualquer conjunto de operações que retornem um valor booleano. Na área de comandos ficam um ou mais comandos da linguagem Java. No caso de usar apenas um comando o uso de {} é opcional.

O comando switch é semelhante ao do C/C++. Ele permite selecionar o bloco de código a ser executado baseado no teste lógico de uma expressão. O switch é a forma evoluída do if, podendo tratar mais de dois blocos de execução. A sintaxe do switch segue logo abaixo.

```
Switch ( <expressão> ) {
        case <valor1>:<comandos 1>
        [break;]
        case <valor2>:<comandos 2>
        [break;]
        case <valor3>:<comandos 3>
        [break;]
```

```
        case <valor4>:<comandos 4>
        [break;]
        ...
        default: <comandos default>
}
```

O comando switch executa a expressão e compara o valor encontrado com os valores de cada caso. Quando encontra a igualdade ele executa o bloco de comandos daquele valor. A execução continuará até o final do switch ou até que ele encontre um break;.

A única limitação do switch é que a expressão deve retornar um valor numérico inteiro, qualquer um de seus quatro tipos primitivos. O switch não funciona com String, float, char e boolean por exemplo.

O operador ternário também é semelhante ao de C/C++. Esse comando tem comportamento bastante semelhante ao do comando □o. Sintaxe abaixo.

<□ondição> ? <comando1> : <comando2>;

A condição será testada, e caso seja verdadeira, a sequência de comandos 1 será executada, caso contrário, será executada comandos 2.

3.6.4 Instruções de repetição

Temos também 3 importantes instruções de repetição no Java. Que são: for, while, do/while;

O comando for cria um laço de repetição no fluxo do programa baseado em três parâmetros:

<expressão inicial>: Onde é executado apenas uma vez, na entrada do laço.
<condição>: É executado a cada iteração do laço e determina quando o programa deve sair do mesmo. Caso a condição seja verdadeira, repete-se os do laço uma vez, caso seja falsa, o programa pula para a próxima instrução seguinte ao laço.
<incremento>: É uma operação normal, executada a cada iteração. Geralmente é usada para incrementar contadores ou configurar variáveis.

Sintaxe da instrução for

```
for (<expressão inicial>; <condição>; <incremento>) {
        <comandos>
}
```

O comando while é utilizado quando não se quer que o corpo do laço seja necessariamente executado. A expressão de comparação é avaliada antes que o laço seja executado, enquanto ela for verdadeira os são repetidos. Sintaxe abaixo.

```
while (<condição>) {
        <comandos>
}
```

O comando do/while é utilizado quando se quer que o corpo do laço seja executado pelo menos uma vez. A expressão de comparação é avaliada depois que o laço foi executado, enquanto ela for verdadeira os são repetidos. Sintaxe abaixo.

```
do {
        <comandos>
} while (<condição>);
```

3.6.5 Definição de funções

Em Java qualquer função ou método deve ser declarado dentro de uma classe. A definição de funções em Java segue o padrão:

```
<modificador de acesso> <tipo de retorno> <nome da função>(<parâmetros>)
{
        <comandos>
}
```

O modificador de acesso indica a visibilidade de desse método na classe para os demais objetos. Pode ser public, private ou protected. É um valor opcional, e quando não usado, o valor padrão (private) é assumido.

O tipo de retorno indica o tipo de variável que o método irá retornar, por exemplo, um inteiro (int), um ponto flutuante, ao um tipo objeto declarado pelo usuário. Quando um método não retornar valor a palavra reservada void deve ser usada como tipo de retorno.

Após o modificador de acesso ainda podemos usar a palavra chave static. Quando declaramos um método como static, ele passa a não ser mais um método de cada objeto, e sim um método da classe, a informação fica guardada pela classe, não é mais individual para cada objeto.

Para acessarmos um método estático, não usamos a palavra chave this, mas sim o nome da classe.

3.6.6 Definição de comentário

Os programadores utilizam o comentário para tornar o código mais legível, ajudando que outras pessoas possam entendê-lo. Tudo que está dentro de um comentário é ignorado pela linguagem, portanto ele não provoca nenhuma ação no computador quando o programa é executado. O comentário em Java pode ser feito em mais de uma forma.

A primeira forma se inicia com // e é chamado de comentário de fim de linha (ou de uma única linha), pois termina na mesma linha. Este tipo de comentário pode ser feito no início da linha ou após um trecho de código, continuando até o final da linha.

A segunda forma é o comentário tradicional (também chamado de comentário de múltiplas linhas). Este tipo de comentário inicia com o delimitador /* e termina com */, e pode se estender por várias linhas.

Comentários no estilo Javadoc, que são delimitados por /** e */ , também são fornecidos por Java. Este tipo de comentário permite que a documentação da linguagem seja incorporada diretamente ao programa.

3.6.7 Exceções

Exceções[15] constituem uma maneira elegante e simples para tratamento de condições excepcionais durante a execução de um programa Java, por exemplo: entrada de dados inválida; falhas no tratamento de arquivos; falhas na comunicação entre processos; reativação de threads; erros aritméticos; estouro de limites de arrays; etc.

Exceções em java possuem a seguinte sintaxe.

```
try {
        <comandos>
} catch (Exception 1 ex) {
        <tratamento da Exception 1>
} catch (Exception 2 ex) {
        <tratamento da Exception 2>
}
...
```

```
catch (Exception n ex) {
        <tratamento da Exception n>
}
finally {
        <bloco finally>
}
```

Os blocos de comandos passíveis de erros são envolvidos pelo bloco try/catch. Se no decorrer da execução desses comandos algum deles lançar uma exceção o bloco deixará de ser executado e o fluxo de instruções será delegado ao bloco catch que trata a exceção recentemente lançada. Ao final podemos também inserir um bloco finally que é opcional, e sempre será executado independente do que tenha ocorrido dentro do bloco try. Finally geralmente é usado para fazer operações de limpeza e que devem ser executados independentemente de ocorrer erro ou não.

3.6.8 Mecanismo de acesso a banco de dados e arquivos

Os programas em Java se comunicam com banco de dados e manipulam seus dados utilizando a API do JDBC. Um driver JDBC permite aos aplicativos Java conectar-se a um banco de dados em um DBMS (Database Management System) particular e permite aos programadores manipular esse banco de dados utilizando a API do JDBC. O JDBC é quase sempre utilizado com um banco de dados relacional. Entretanto, ele pode ser utillizado com qualquer origem de dados baseada em tabelas. A maioria dos bancos de dados populares já fornece seus próprios drivers JDBC. Além disso, muitos fornecedores independentes fornecem drivers JDBC para uma variedade de bancos de dados.

Java possui poderosos recursos de processamento de arquivos e de fluxos de entrada/saída. Toda essa arquitetura se encontra no pacote java.io. Neste pacote encontram-se todas as classes necessárias à manipulação de arquivos de qualquer tipo, sejam de acesso sequêncial ou aleatório. Assim como a maioria das linguagens orientadas a objetos o java também permite a gravação de objetos em arquivos através do processo de serialização.

3.6.9 Mecanismo de acesso a dispositivos externos

O Java fornece APIs para acessar dispositivos externos conectados a uma porta de entrada/saída de um computador pessoal ou a partir de um computador remoto. Entre essas APIs podemos citar como principal a JNI (Java Native Interface). O JNI é um padrão de programação que permite que a máquina virtual do Java acesse bibliotecas construídas com o código nativo de um sistema. Isso possibilita acesso às principais portas de entrada e saída no computador, como por exemplo, a porta paralela, serial e USB.

A plataforma JME permite que os programadores tenham acesso e façam softwares para dispositivos móveis, como celulares e palmtops.

3.7 REQUISITOS MINIMOS PARA EXECUTAR UM PROGRAMA

Para executar um programa em Java, é necessário ter instalado na máquina o JRE (Java Runtime Environment), ambiente de execução Java, que é composto pela JVM (Java Virtual Machine) e por bibliotecas (APIs).

3.8 COMO CONECTAR O BANCO DE DADOS COM A LINGUAGEM

Para fazer conexão com um banco de dados é necessário o uso da API do JDBC[16]. Cada banco de dados fornece a sua própria API JDBC. Logo, se o programador deseja fazer conexão com um banco de dados MySQL por exemplo, é necessário que ele tenha nas bibliotecas de seu projeto o driver JDBC do MySQL, que geralmente pode ser adquirido no site do fornecedor do MySQL. Uma vez com o driver à sua disposição também é preciso que o programador consulte a documentação do driver e veja as instruções e passos necessários para se estabelecer uma conexão com o banco de dados em questão.

3.9 INTERAÇÕES COM BIBLIOTECAS DE OUTRAS LINGUAGENS

O java permite que rotinas escritas em outra linguagem sejam executadas a a partir da máquina virtual e para isso faz uso da API JNI (Java Native Interface). JNI é uma suíte de programação que permite que código em Java rodando sob uma máquina virtual Java (JVM) chamar e ser chamada por aplicações nativas. Aplicações nativas são programas escritos para plataformas de hardware e sistema operacional específicos. É possível fazer chamadas para C, C++ e Assembly.

3.10 UTILITÁRIOS PARA DOCUMENTAÇÃO

A plataforma Java vem acompanhada por um utilitário chamado Javadoc. O Javadoc é um gerador de documentação para documentar a API dos programas em Java, a partir do código-fonte. A documentação resultante está em formato HTML. O javadoc consegue gerar documentação a partir de algumas marcações simples , as Tags, inseridas nos comentários do código pelo próprio programador.

A tabela abaixo lista as principais Tags do Javadoc:

Tag	Descrição
@author	Nome do desenvolvedor
@deprecated	Marca o método como *deprecated*. Algumas IDEs exibirão um alerta de compilação se o método for chamado.
@exception	Documenta uma exceção lançada por um método.
@param	Define um parâmetro do método. Requerido para cada parâmetro.
@return	Documenta o valor de retorno. Essa tag não deve ser usada para construtores ou métodos definidos com o tipo de retorno *void*.
@see	Documenta uma associação a outro método ou classe.
@since	Documenta quando o método foi adicionado a a classe.
@throws	Documenta uma exceção lançada por um método. É um sinônimo para a @exception introduzida no Javadoc 1.2.
@version	Exibe o número da versão de uma classe ou um método.

As Tags do Javadoc é o padrão de documentação de classes em Java, e muitas dos IDEs do Java automaticamente geram um Javadoc em HTML.

3.11 DESENVOLVER APLICAÇÕES WEB E DESKTOP

O foco da linguagem Java é desenvolver aplicativos de grande porte, tanto sistemas web quanto desktop. A linguagem tornou-se popular principalmente pelo seu uso na internet, e hoje possui aplicações em web browsers, mainframes, sistemas operacionais, celulares, palmtops e cartões inteligentes, entre outros.

3.12 AMBIENTES DE DESENVOLVIMENTO INTEGRADO

Os IDEs fornecem muitas ferramentas que suportam o processo de desenvolvimento de software, com editores para escrever e editar programas e depuradores para localizar erros lógicos de programas.

Existem muitos IDEs disponíveis para a linguagem Java. Destacam-se:

- NetBeans, desenvolvido pela Sun Microsystems, gratuito e de código aberto. Pode ser baixado em www.netbeans.org
- jEdit, desenvolvido por Slava Pestov, disponível sob a GNU General Public License. Pode ser baixado em www.jedit.org
- JCreator, desenvolvido pela Xinox, possui duas versões: Lite Edition (LE), gratuita, e Pro Edition (Pro), que custa $89 após um período Trial de 30 dias. Pode ser baixado/comprado em www.jcreator.com
- BlueJ, desenvolvido por Michael Kölling, disponível sob a GNU General Public License. Pode ser baixado em www.bluej.org
- Eclipse, inicialmente desenvolvido pela IBM, agora mantido pela comunidade de Software Livre (Free Software Community), gratuito e de código aberto, disponível sob a Eclipse Public License. Pode ser baixado em www.eclipse.org
- JBuilder, desenvolvido pela Borland/CodeGear, possui uma versão gratuita, JBuilder 2008 Turbo, e outras duas pagas, JBuilder 2008 Professional, que custa R$1.497 e JBuilder 2008 Enterprise, que custa R$4.497. Pode ser baixado/comprado em www.codegear.com/Products/JBuilder
- JDeveloper, desenvolvido pela Oracle, gratuito e disponível sob a OTN JDeveloper License. Pode ser baixado em www.oracle.com/technology/products/jdev/index.html
- IntelliJ IDEA, comercial, desenvolvida pela JetBrains, considerada por muitos a melhor IDE do mercado, possui cinco tipos de licença: Commercial License, para empresas e organizações, custando $599; Personal License, para desenvolvedores, custando $249; Academic License, para estudantes e professores, custando $99; Classroom License, para instituições educacionais, gratuita, e Open Source License, para projetos de código aberto. Pode ser baixado/comprado em http://www.jetbrains.com/idea/

- Outras IDEs menos populares são: Gel, Greenfoot, JGRASP, Java Studio Creator/Enterprise, Workshop for WebLogic, WebSphere Studio Application Developer.

3.13 PADRONIZAÇÃO DA LINGUAGEM

A Sun Microsystems já tentou submeter a linguagem à padronização pelos órgãos ISO/IEC e ECMA, mas acabou desistindo. A plataforma Java é padronizada e controlada através da JCP (Java Community Process), trata-se de um processo formalizado que permite que interessados se envolvam nas definições de versões futuras e funcionalidades da plataforma Java. Atualmente o JCP possui como principais membros grandes produtores de softwares mundiais. Entre eles Oracle, IBM, Red Hat, Apache Software Foundation, Adobe Systems, Sony, além claro, da própria Sun Mycrosystems.

Praticamente todo o código Java está como software de código aberto, sob os termos da GNU General Public License (GPL), menos uma pequena porção da linguagem a qual a Sun não possui copyright.

4 CONCLUSÂO

Pode-se perceber que o Java é uma linguagem bastante poderosa e que está conquistando mais espaço na área de desenvolvimento de software. Como existe uma grande empresa que a mantém e muitas comunidades que se dedicam para desenvolver soluções em sua plataforma, há grandes ferramentas de desenvolvimento, principalmente para software empresarias e aplicações web.

JSF é apenas um dos muitos frameworks que foram desenvolvidos em Java. Além de JSF, para web ainda existe o Struts, Facelets. Também existem outros frameworks que atuam em níveis diferentes nas camadas de aplicação.

Assim, podemos afirmar que Java agora representa mais que uma linguagem, e sim uma tecnologia com uma gama de ferramentas, que vão de interface com o usuário, até a camada mais baixa de comunicação com o banco de dados.

REFERÊNCIAS

[1] JavaServer Pages. Disponível em <http://java.sun.com/products/jsp/>. Acessado em 26 de abril 2009.

[2] Model View Control. Disponível em <http://pt.wikipedia.org/wiki/MVC>. Acessado em 26 de abril 2009.

[3] JavaServer Faces. Disponível em <http://java.sun.com/javaee/javaserverfaces/>. Acessado em 26 de abril 2009.

[4] JavaServer Faces: A mais nova tecnologia Java para desenvolvimento web. Disponível em <www.guj.com.br/content/articles/jsf/jsf.pdf>. Acessado em 26 de abril 2009.

[5] SUN Microsystem Brasil. Disponível em <http://br.sun.com/>. Acessado em 26 de abril 2009.

[6] MyFaces Wiki. Disponível em <http://wiki.apache.org/myfaces/>. Acessado em 26 de abril 2009.

[7] Apache Software Foundation. Disponível em <http://www.apache.org/>. Acessado em 26 de abril 2009.

[8] General Public License. Disponível em <http://www.gnu.org/copyleft/gpl.html>. Acessado em 26 de abril 2009.

[9] Jboss. Disponível em <http://en.wikipedia.org/wiki/JBoss>. Acessado em 26 de abril 2009.

[10] Java (linguagem de programação). Disponível em <http://pt.wikipedia.org/wiki/Java_(linguagem_de_programação)>. Acessado em 25 de abril de 2009.

[11] Java Version History. Disponível em <http://en.wikipedia.org/wiki/Java_version_history>. Acessado em 25 de abril 2009.

[12] Questões de Concursos em Java. Disponível em <http://www.questoesdeconcursos.com.br/pesquisar/marcador/98>. Acessado em 25 de abril 2009.

[13] Java 2 – Características Básicas. Disponível em <http://javafree.uol.com.br/artigo/871496/Tutorial-Java-2-Caracteristicas-Basicas>. Acessado em 24 de abril 2009.

[14] Java e Orientação a Objetos. Disponível em <http://www.caelum.com.br/downloads/apostila/caelum-java-objetos-fj11.pdf>. Acessado em 25 de abril 2009.

[15] Exceções em Java. Disponível em <http://www.di.ufpe.br/~java/jai/aula7/index.html>. Acessado 25 de abril 2009.

[16] API do JDBC. Disponível em <java.sun.com/products/jdbc>. Acessado em 28 de abril 2009.

APÊNDICES

APENDICE A – Exemeplo de Programa

Apêndice A.1 – Index.jsp

```
<jsp:forward page="menu.lac"></jsp:forward>
```

Apêndice A.2 – menu.jsp

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<html>
  <head>
    <title>SEMINARIO LAC</title>
  </head>
  <body>
    <f:view>
      <h:form>
       <h:outputText value="Agenda"/>
         <h:panelGrid columns="2" >
            <h:commandLink value="Inserir" action="inserir"/>
            <h:commandLink value="Buscar" action="buscar"/>
         </h:panelGrid>
       </h:form>
    </f:view>
  </body>
</html>
```

Apêndice A.3 – inserir.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>SEMINARIO LAC</title>
</head>
<body>
<f:view>
<h:form>
        <h:outputText value="Inserir Pessoas na Agenda"/>
        <h:panelGrid columns="2">
                <h:outputText value="Nome"/>
                <h:inputText value="#{mB.pessoa.nome}" />
                <h:outputText value="EndereÃ§o"/>
                <h:inputText value="#{mB.pessoa.endereco}" />
                <h:outputText value="Telefone"/>
                <h:inputText value="#{mB.pessoa.telefone}" />
                <h:commandButton value="Inserir" actionListener="#{mB.inserirPessoa}"/>
                <h:commandLink value="Voltar" action="menu"/>
        </h:panelGrid>

</h:form>
</f:view>
</body>
</html>
```

Apêndice A.4 – buscar.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
	pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>SEMINARIO LAC</title>
</head>
<body>
<f:view>
	<h:form>
		<h:outputText value="Buscar Pessoas" />
		<h:panelGrid columns="3">
			<h:selectOneMenu value="#{mB.opcao}">
				<f:selectItem itemLabel="Selecione" itemValue="0" />
				<f:selectItem itemLabel="Nome" itemValue="1" />
				<f:selectItem itemLabel="EndereÃ§o" itemValue="2" />
				<f:selectItem itemLabel="Telefone" itemValue="3" />
			</h:selectOneMenu>
			<h:inputText value="#{mB.valor}" />
			<h:commandButton value="Buscar" actionListener="#{mB.buscarPessoas}"/>
		</h:panelGrid>
		<h:dataTable value="#{mB.pessoasBuscadas}" var="pessoa">
			<h:column>
				<f:facet name="header">
					<h:outputText value="Nome" />
				</f:facet>
				<h:outputText value="#{pessoa.nome}" />
			</h:column>
			<h:column>
				<f:facet name="header">
					<h:outputText value="EndereÃ§o" />
				</f:facet>
				<h:outputText value="#{pessoa.endereco}" />
			</h:column>
			<h:column>
				<f:facet name="header">
					<h:outputText value="Telefone" />
				</f:facet>
				<h:outputText value="#{pessoa.telefone}" />
			</h:column>
		</h:dataTable>
		<h:commandLink value="Voltar" action="menu" />
	</h:form>
</f:view>

</body>
</html>
```

Apêndice A.5 – MB.java e Pessoa.Java

```
package projeto;

import java.util.ArrayList;
import java.util.List;

import javax.faces.event.ActionEvent;

public class MB {

	private String opcao;
	private String valor;
	private Pessoa pessoa;
	private List<Pessoa> pessoas;
	private List<Pessoa> pessoasBuscadas;


	public MB(){
		pessoas = new ArrayList<Pessoa>();
		pessoa = new Pessoa();
		valor="";
	}

	public void buscarPessoas(ActionEvent actionEvent){
		pessoasBuscadas = new ArrayList<Pessoa>();
		int i = Integer.parseInt(opcao);
		switch (i) {
		case 1:
				for(Pessoa p:pessoas){
					if(p.getNome().contains((CharSequence)valor)){
						pessoasBuscadas.add(p);
					}
				}
			break;
		case 2:
			for(Pessoa p:pessoas){
				if(p.getEndereco().contains((CharSequence)valor)){
					pessoasBuscadas.add(p);
				}
			}
			break;
		case 3:
			for(Pessoa p:pessoas){
				if(p.getTelefone().contains((CharSequence)valor)){
					pessoasBuscadas.add(p);
				}
			}
			break;
		}

	}

	public void inserirPessoa(ActionEvent actionEvent){
		pessoas.add(pessoa);
		pessoa= new Pessoa();
	}
```

```
	public String getOpcao() {
		return opcao;
	}
	public void setOpcao(String opcao) {
		this.opcao = opcao;
	}
	public List<Pessoa> getPessoas() {
		return pessoas;
	}
	public void setPessoas(List<Pessoa> pessoas) {
		this.pessoas = pessoas;
	}
	public Pessoa getPessoa() {
		return pessoa;
	}
	public void setPessoa(Pessoa pessoa) {
		this.pessoa = pessoa;
	}

	public String getValor() {
		return valor;
	}

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

	public List<Pessoa> getPessoasBuscadas() {
		if(valor.equals("")){
			return pessoas;
		}else{
		return pessoasBuscadas;
		}
	}

	public void setPessoasBuscadas(List<Pessoa> pessoasBuscadas) {
		this.pessoasBuscadas = pessoasBuscadas;
	}
}

//Classe Pessoa
package projeto;

public class Pessoa {

	private String nome;
	private String endereco;
	private String telefone;

	public Pessoa(){
		this.nome = "";
		this.endereco = "";
		this.telefone = "";

	}

	public Pessoa(String nome, String endereco, String telefone){
	this.nome = nome;
	this.endereco = endereco;
	this.telefone = telefone;
```

```
	}

	public String getNome() {
		return nome;
	}

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

	public String getEndereco() {
		return endereco;
	}

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

	public String getTelefone() {
		return telefone;
	}

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


}
```

Apêndice A.6 – web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-
app_2_4.xsd">
	<display-name>
	lac</display-name>
	<servlet>
		<servlet-name>Faces Servlet</servlet-name>
		<servlet-class>
		javax.faces.webapp.FacesServlet</servlet-class>
		<load-on-startup>1</load-on-startup>
	</servlet>
	<servlet-mapping>
		<servlet-name>Faces Servlet</servlet-name>
		<url-pattern>*.lac</url-pattern>
	</servlet-mapping>
	<welcome-file-list>
		<welcome-file>index.jsp</welcome-file>
	</welcome-file-list>
</web-app>
```

Apêndice A.7 – faces-config.xml

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

<!DOCTYPE faces-config PUBLIC
   "-//Sun Microsystems, Inc.//DTD JavaServer Faces Config 1.1//EN"
   "http://java.sun.com/dtd/web-facesconfig_1_1.dtd">

<faces-config>
        <managed-bean>
                <managed-bean-name>mB</managed-bean-name>
                <managed-bean-class>projeto.MB</managed-bean-class>
                <managed-bean-scope>session</managed-bean-scope>
        </managed-bean>
        <navigation-rule>
                <display-name>buscar</display-name>
                <from-view-id>/buscar.jsp</from-view-id>
                <navigation-case>
                        <from-outcome>menu</from-outcome>
                        <to-view-id>/menu.jsp</to-view-id>
                </navigation-case>
        </navigation-rule>
        <navigation-rule>
                <display-name>menu</display-name>
                <from-view-id>/menu.jsp</from-view-id>
                <navigation-case>
                        <from-outcome>buscar</from-outcome>
                        <to-view-id>/buscar.jsp</to-view-id>
                </navigation-case>
        </navigation-rule>
        <navigation-rule>
                <display-name>inserir</display-name>
                <from-view-id>/inserir.jsp</from-view-id>
                <navigation-case>
                        <from-outcome>menu</from-outcome>
                        <to-view-id>/menu.jsp</to-view-id>
                </navigation-case>
        </navigation-rule>
        <navigation-rule>
                <display-name>menu</display-name>
                <from-view-id>/menu.jsp</from-view-id>
                <navigation-case>
                        <from-outcome>inserir</from-outcome>
                        <to-view-id>/inserir.jsp</to-view-id>
                </navigation-case>
        </navigation-rule>

</faces-config>
```

APÊNDICE B – Questões de Concursos.

1 - Em que porção da JVM (Java Virtual Machine) são armazenados objetos instanciados em um programa JAVA ?

a) Heap

b) GUnit

c) Stack Pool

d) Dump Buffer

e) Text Segment

A resposta correta está alternativa a). Os objetos instanciados em um programa java são armazenados na Heap.

Esta questão caiu na prova do CAPES aplicada pela banca CESGRANRIO para o cargo de Analista de Sistemas no ano de 2008.

2 - Quando um *servlet* é carregado pela primeira vez para a máquina virtual Java do servidor

a) ocorre um destroy() no processo cliente.

b) o seu método init() é invocado.

c) o método service() é definido.

d) ocorre a execução do método getOutputStream().

e) o seu método stream() é invocado.

A resposta correta está na alternativa b). Essa questão exigiu do candidado conhecimentos a respeito da plataforma do java para web com um uso de Servlets.

Esta questão caiu na prova do TRE-SE aplicada pela banca FCC para o cargo de Analista Judiciário – Especialidade de Sistemas – Desenvolvimento no ano de 2007.

3 - Observe o seguinte programa JAVA:

```
package ex;

public class caso1 {

     public caso1() {
     }

      public static void main(String[] args)
{
          try {
                System.out.println(3/0);
          }
           catch (ArithmeticException e1) {
                System.out.print("M");
          }
           catch (Exception e2) {
                System.out.print("G");
          }
          finally {
                System.out.print("X");
          }
           System.out.print("Fim");
     }
}
```

A saída da execução desse programa é:

a) GXFim

b) 0Fim

c) MGXFim

d) MXFim

e) M

A resposta correta está na alternativa d). O programa tenta imprimir o resultado de uma operação matemática. Como ocorre divisão por 0 – o java lança é a exceção ArithmeticException que em seguinda é capturada no bloco logo abaixo, que imprime a string “M”. Como a exceção já foi capturada o bloco catch logo abaixo não é executado. Finalmente o bloco finally é executado e imprime a string “X” . Ao sair do bloco try/catch/finally o programa imprime a string “Fim”. Ao final do programa obtivemos a saida “MXFim”.

Esta questão caiu na prova do REFAP SA aplicada pela banca CESGRANRIOpara o cargo de Programador de Computador no ano de 2007.

4 - Os métodos Java que não retornam valores devem possuir no parâmetro tipo-de-retorno a palavra

a) static.

b) public.

c) void.

d) main.

e) string args.

A resposta correta está na alternativa b). Métodos java que não retornam valores devem declarar void como tipo de retorno.

Esta questão caiu na prova do **TRE-MG** aplicada pela banca FCC para o cargo de Técnico Judiciário – Especialidade – Programação de Sistemas no ano de 2005.

5 - A portabilidade é a capacidade que um programa tem de poder ser executado em diferentes plataformas. O sucesso de Java vem em parte de sua portabilidade, que é fruto da sua capacidade de ligação tardia (*dynamic binding*), que viabiliza a carga e a ligação de código em tempo de execução, adaptando-se, assim, à plataforma em que está inserido.

Certo ou Errado?

A preposição está Errada. A portabilidade da linguagem Java deve-se ao fato dela ser uma linguagem interpretada. O compilador Java traduz o código-fonte (arquivo .java) para uma linguagem intermediária independente da plataforma, gerando assim o bytecode (arquivo.class). O bytecode depois é traduzido para a linguagem máquina pelo interpretador Java (Máquina Virtual Java) e executado.

Esta questão caiu na prova do **Polícia Federal** aplicada pela banca CESPE para o cargo de Perito Criminal Federal – Informática no ano de 2004.