

Curso

Pacotes, Lambdas, Streams, Interfaces Gráficas

Atualizado até o Java 21 &
Eclipse 2023-09



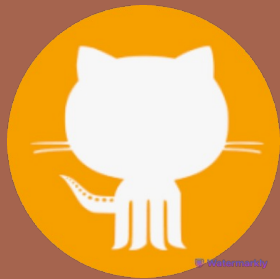
Prof. Msc. Antonio B. C. Sampaio Jr
ENGENHEIRO DE SOFTWARE & PROFESSOR

@abctreinamentos
@amazoncodebr

www.abctreinamentos.com.br
www.amazoncode.com.br



REPOSITÓRIO GITHUB



antonio-sampaio-jr / **pacotes-lambdas-streams**

CONTEÚDO PROGRAMÁTICO



- UNIDADE 1 – PACOTES, ERROS E EXCEÇÕES
 - Pacotes em Java
 - Instalação do JAVA 21 & ECLIPSE 2023-09
[NOVO]
 - Modularidade [NOVO]
 - Formato Java ARchive
 - Erros e Exceções
 - Helpful NullPointerException [NOVO]
 - Throws e Throw
 - JavaDoc
 - Novidades JavaDoc [NOVO]



UNIDADE 1

PACOTES, ERROS E EXCEÇÕES

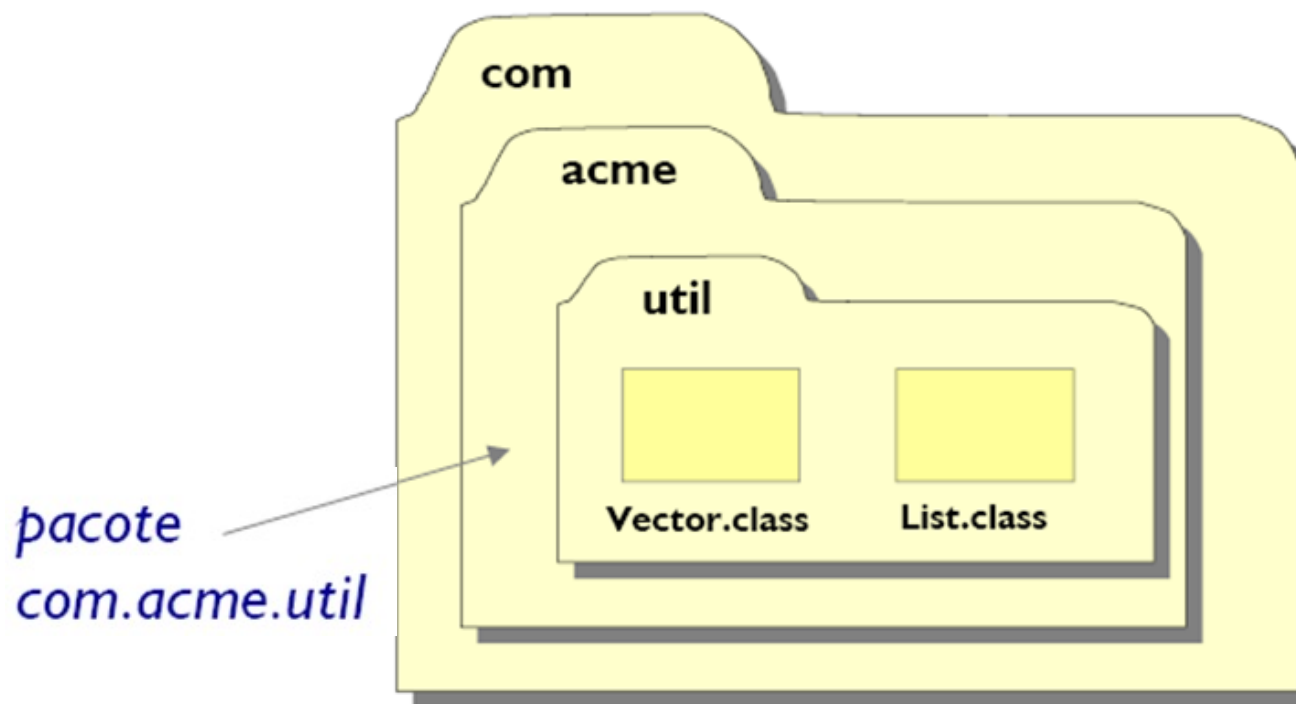


Pacotes em Java

Conceito de Pacotes

Um Pacote é:

- Um conjunto de classes e interfaces relacionadas e outros pacotes que provê **acesso protegido** e **gerenciamento de espaço de nomes (*namespaces*)**.



Conceito de Pacotes

Acesso Protegido

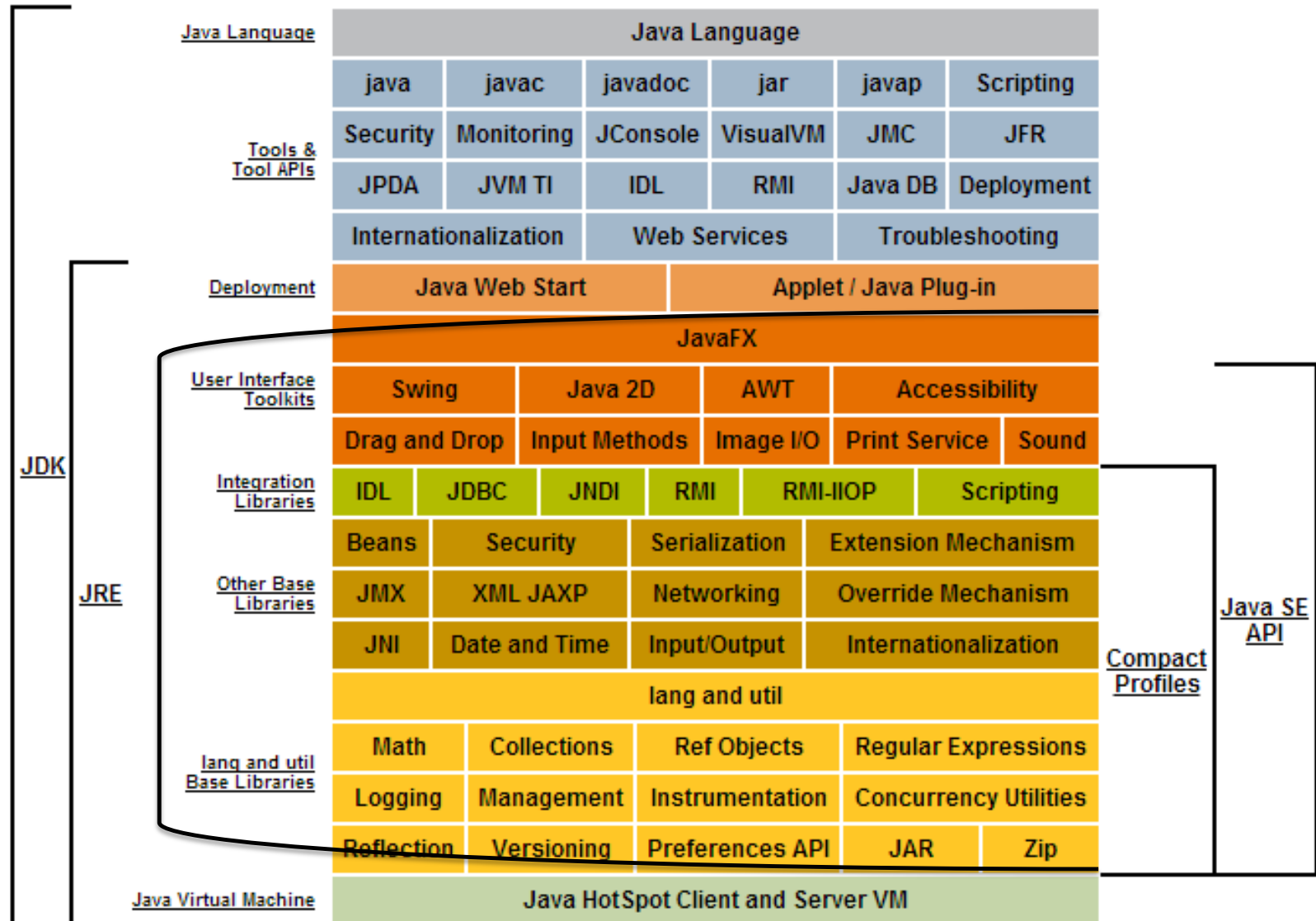
- Os pacotes são organizados em diretórios.
- Há um mecanismo de controle de acesso (*package*) às classes e interfaces de um determinado pacote.
- Somente as classes e interfaces de um mesmo pacote podem acessar uns aos outros de forma irrestrita, porém restringindo o acesso a outros tipos de pacotes.

Gerenciamento de Espaço de Nomes

- Para que os nomes de seus tipos não entrem em conflito com os nomes de tipos de outros pacotes, cada qual cria o seu próprio espaço de nomes (*namespaces*).

```
java.lang.Object e org.omg.CORBA.Object
```

Pacotes Java



Pacotes Java

- O Java 8 define mais de 200 pacotes para serem utilizados no desenvolvimento de aplicativos Java.

Principais Pacotes Java

java.lang: classes e interfaces fundamentais – importado automaticamente

java.applet: classes e interfaces para a criação de applets na Internet

java.util: classes e interfaces utilitárias

java.io: classes e interfaces para entrada e saída

java.nio: classes e interfaces para buffers

java.net: classes e interfaces para uso em rede (TCP/IP)

java.rmi: classes e interfaces para acesso remoto

java.sql: classes e interfaces para acesso ao Banco de Dados

java.awt: interface gráfica universal nativa

java.text: internacionalização, transformação e formatação de texto

java.time: classes para datas e calendários

javax.crypto: classes e interfaces para operações de criptografia

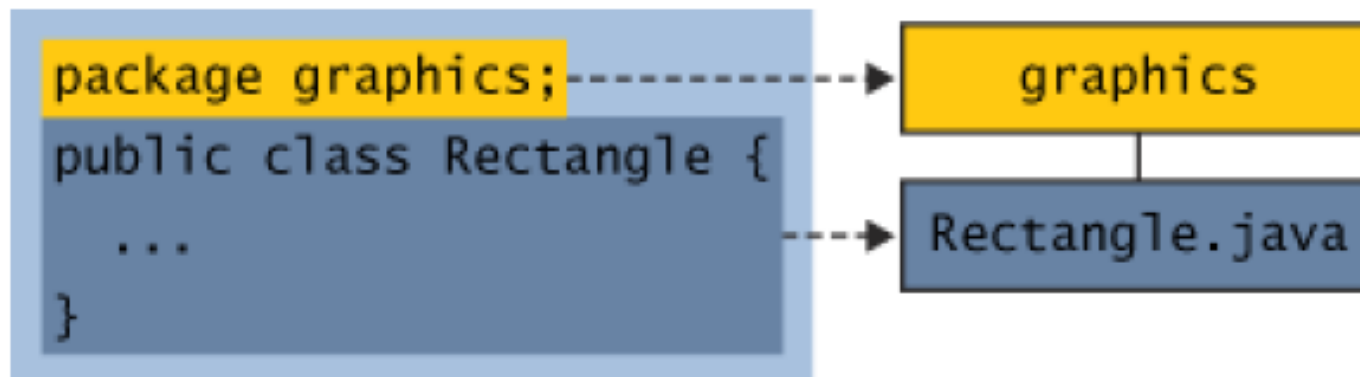
javax.xml: classes e interfaces para manipulação de XML

Pacotes Java

- O nome dos pacotes são hierárquicos e separados por pontos.

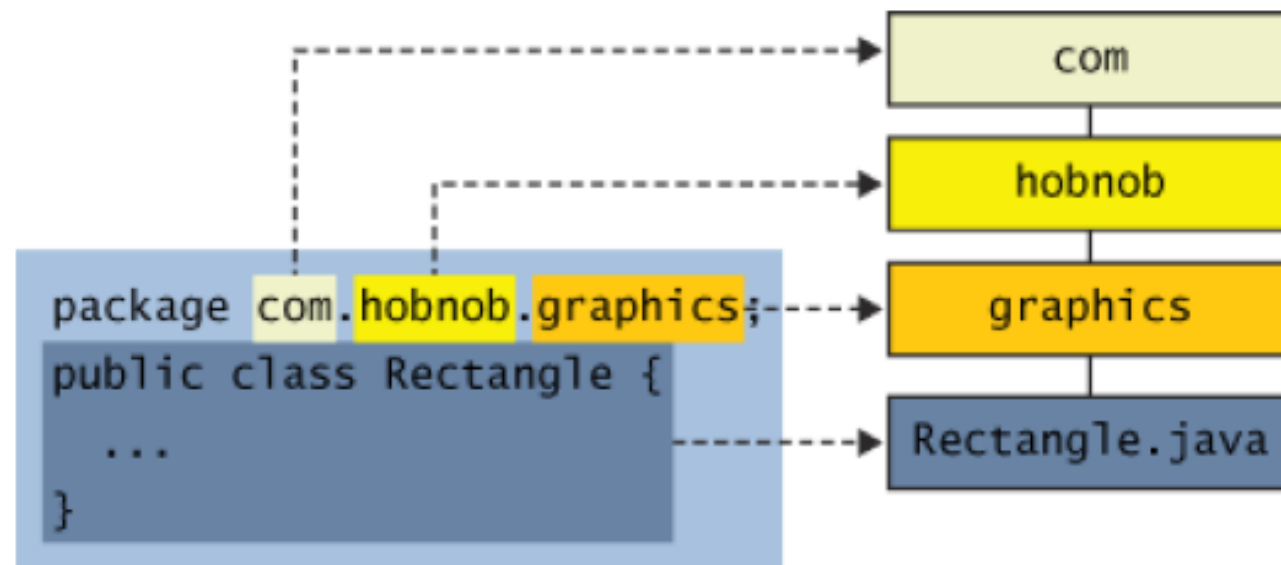
```
br.com.nomedaempresa.nomedoprojeto.subpacote  
br.com.nomedaempresa.nomedoprojeto.subpacote2  
br.com.nomedaempresa.nomedoprojeto.subpacote2.subpacote3
```

- Convencionalmente, os elementos dos nomes de pacotes estão normalmente em letras minúsculas.
- Para indicar que as classes de um arquivo pertencem a um determinado pacote, utiliza-se a palavra reservada **package**.



Pacotes Java

- A declaração de pacote, se houver, deve estar sempre no início do arquivo de origem.
- Hierarquias de pacotes são construídas através de hierarquias de diretórios.



- Caminho da classe Java: **com/hobnob/graphics/Rectangle.java**

Pacotes Java

- Para se criar uma classe como pertencente a um pacote, deve-se:
 - Declará-la em um arquivo dentro do diretório que representa o pacote
 - Declarar, na primeira linha, que a classe pertence ao pacote, usando-se a palavra reservada **package**.

```
package com.hobnob.graphics;  
class Triangulo{  
    ...  
}
```

Importação de Pacotes Java

- Para se usar uma classe/interface que está dentro de um pacote, precisa-se referenciar o nome completo desse pacote no acesso à classe:

```
com.hobnob.graphics.Triangulo objeto =  
new com.hobnob.graphics.Triangulo();
```

- Pode-se usar o comando **import** para simplificar a chamada à classe Triangulo.

```
import com.hobnob.graphics.Triangulo;  
Triangulo objeto = new Triangulo();
```

- É possível importar as classes/interfaces de um pacote inteiro:

```
import com.hobnob.graphics.*;
```

Importação de Pacotes Java

- Com o coringa ‘*’ é possível importar todas as classes/interfaces que estão definidas em um pacote, **exceto os seus subpacotes**.
- Importar todas as classes de um pacote não implica em perda de performance em tempo de execução. Contudo, importar de um em um é considerada uma boa prática, pois facilita a leitura do código para os outros desenvolvedores.
- No Eclipse, a importação unitária já é feita de forma automática, bem como a organização dos pacotes em diretórios.

Controle de Acesso

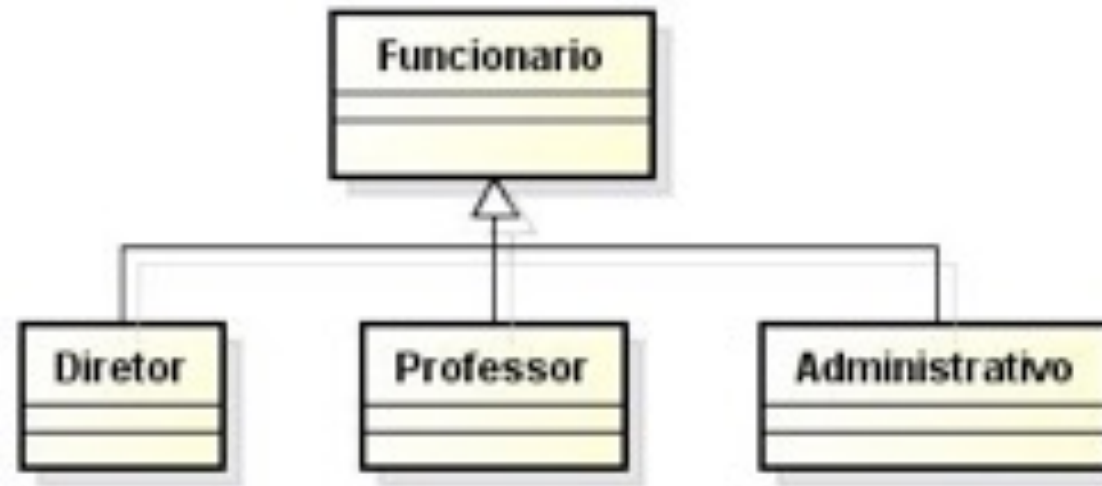
- Um membro de pacote só pode ser acessado por classes/interfaces declaradas no mesmo pacote.
- Quando o modificador de acesso é omitido, diz-se que o membro (atributos/métodos) é do tipo **package**.
- Todas as classes do mesmo pacote têm acesso aos atributos e métodos umas das outras.

Resumo de Visibilidade



Visibilidade	public	protected	package	private
Classe	Sim	Sim	Não	Sim
SubClasse	Sim	Sim	Não	Não
Pacote	Sim	Sim	Sim	Não
Geral	Sim	Não	Não	Não

Exercícios

- 1) Criar a hierarquia de classes abaixo pertencentes ao pacote **br.abctreinamentos.rh**



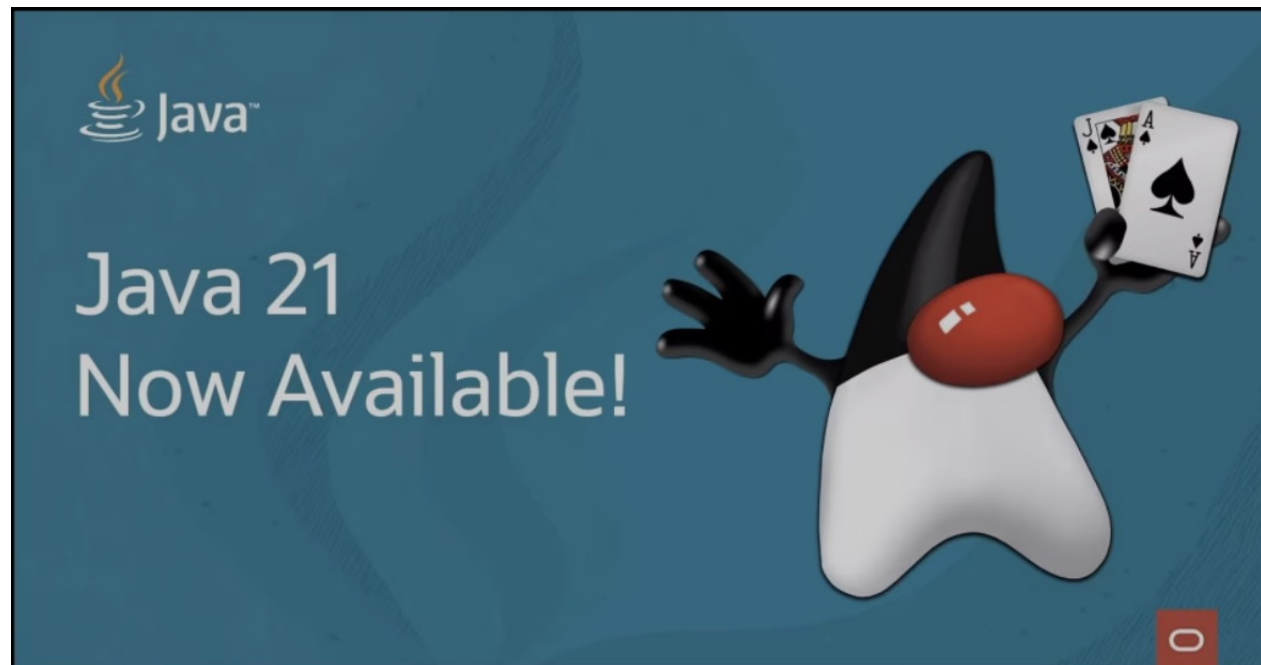
- 2) Criar a classe **Universidade** pertencente ao pacote **br.abctreinamentos** que possui diversos **Funcionários**, com o método de classe **gerarRelatorioProfessoresMestresDoutores(...)**.
- 3) Criar a classe **Mec** – não pertencente ao pacote **br.abctreinamentos** - que fará a avaliação da **Universidade** em questão pelo método **avaliarEnsino(...)**.



Instalação do JAVA 21 & ECLIPSE 2023-09

Instalação do JAVA 21

18



<https://www.oracle.com/br/java/technologies/downloads/>

Instalação do Eclipse 2023-09

19



<https://www.eclipse.org/downloads/packages/>

Static Import

- Estende as funcionalidades do comando **import**, possibilitando que membros de classe (atributos e métodos) sejam chamados.
- Código sem **Static Import**

```
int raio = 5;  
double comp = Math.PI*Math.pow(raio, 2);  
System.out.println("Comp. do circulo="+comp);
```

- Código com **Static Import**

```
import static java.lang.Math.*;  
...  
int raio = 5;  
double comp = PI*pow(raio, 2);  
System.out.println("Comp. do circulo="+comp);
```

Static Import

OBSERVAÇÃO: Importações estáticas misturam os *namespaces* das classes, podendo causar conflitos de nome e perder legibilidade.

Exercício

- 1) Incluir na classe **Mec** o acesso ao método de classe **gerarRelatorioProfessoresMestresDoutores(...)**.
- 2) Alterar a chamada a esse método fazendo-se uso da **importação estática**.



Modularidade

- Java SE modules**
- 303Mb**



MODULARIDADE

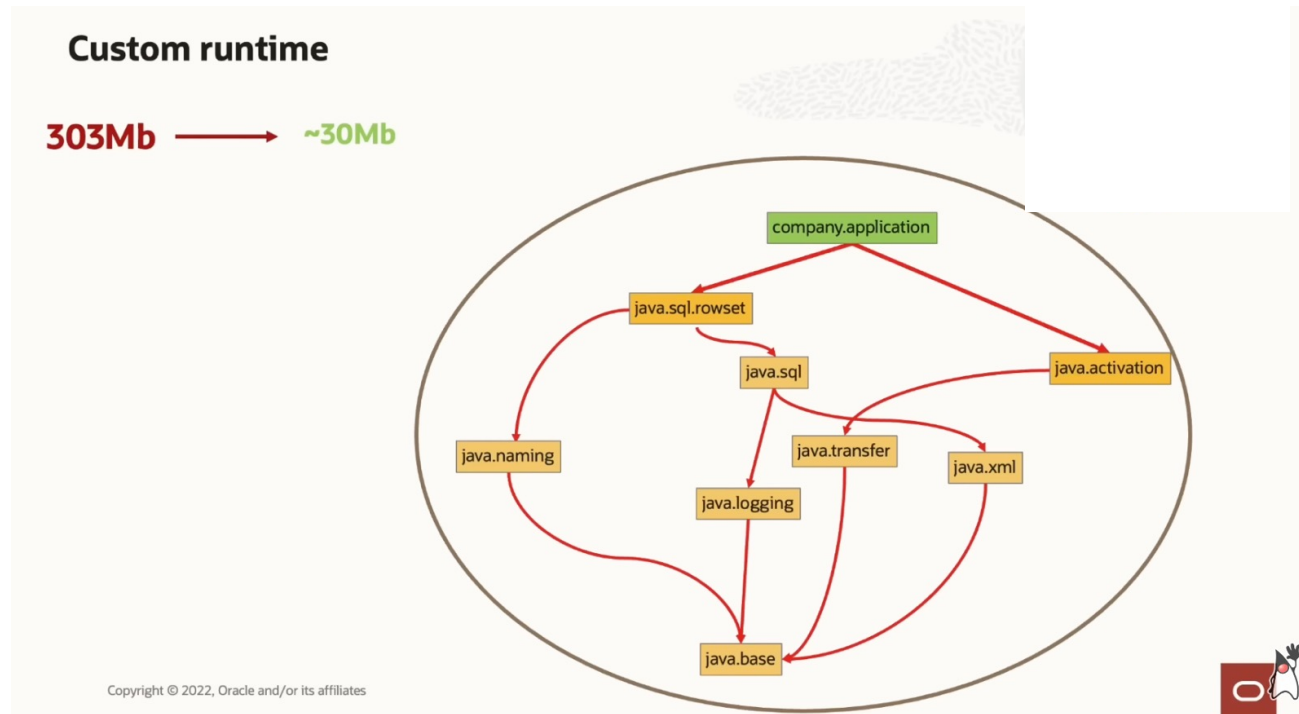
- **Módulo java.base**

- É o único módulo que deve estar presente em todas as implementações. Este define e exporta todos os pacotes que fazem parte do núcleo da plataforma, incluindo o próprio sistema de módulos.

```
module java.base {  
    exports java.io;  
    exports java.lang;  
    exports java.lang.annotation;  
    exports java.lang.invoke;  
    exports java.lang.module;  
    exports java.lang.ref;  
    exports java.lang.reflect;  
    exports java.math;  
    exports java.net;  
    ...  
}
```

MODULARIDADE

- **Custom runtime**
 - É uma configuração personalizada de um ambiente de tempo de execução projetada para atender aos requisitos específicos de um aplicativo ou sistema, oferecendo otimizações e configurações especializadas.



MODULARIDADE

- **Exemplo**

- Neste exemplo, o módulo **meuModulo** exporta o pacote **com.example.meuModulo**, permitindo que outras partes do código tenham acesso a ele. A classe **MinhaClasse** possui um método simples chamado método que imprime uma mensagem.

```
module meuModulo {  
    exports com.example.meuModulo; // Exporta o pacote com.example.meuModulo  
}
```

```
package com.example.meuModulo;  
  
public class MinhaClasse {  
    public void metodo() {  
        System.out.println("Olá do meuModulo!");  
    }  
}
```

PACOTES x MÓDULOS

- **Resumo**

- Em resumo, a utilização de módulos traz um nível de organização e encapsulamento mais avançado em relação ao sistema de pacotes tradicional, especialmente em projetos maiores e complexos. Isso promove a manutenibilidade, a confiabilidade e a escalabilidade do código.

PACOTES x MÓDULOS

- Resumo

Característica	Módulos	Pacotes
Escopo de Organização	Permitem organizar o código em unidades maiores e independentes.	Permitem organizar o código em unidades menores dentro de um projeto.
Encapsulamento	Fornecem um nível mais alto de encapsulamento, permitindo controlar explicitamente quais partes do código são visíveis para outros módulos.	O encapsulamento é mais leve e baseado principalmente em modificadores de acesso (public, protected, private).
Resolução de Dependências	Possibilitam especificar claramente quais outros módulos são necessários para o funcionamento correto.	A resolução de dependências é menos explícita e pode ser mais sujeita a conflitos de classpath.
Evita Conflitos de Nomes	Permite ter classes com o mesmo nome em módulos diferentes sem conflitos.	Requer nomes únicos para classes em um mesmo classpath.

PACOTES x MÓDULOS

- Resumo

Característica	Módulos	Pacotes
Acoplamento	Reduz o acoplamento entre diferentes partes do código, pois você pode definir interfaces e exportar apenas o necessário.	Pode haver acoplamento mais forte entre classes em diferentes pacotes.
Segurança e Confiabilidade	Permite restringir o acesso a certas partes do código, proporcionando um maior controle de acesso.	O controle de acesso é baseado principalmente em modificadores de acesso, mas pode ser mais difícil de aplicar em grande escala.
Facilita a Manutenção e Escalabilidade	Facilita a manutenção, pois as dependências são explicitamente definidas e limitadas a outros módulos.	A manutenção pode ser mais complexa, especialmente em projetos grandes com muitas dependências.
Aprimoramentos de Desempenho	Pode otimizar a inicialização e execução do programa, carregando apenas os módulos necessários.	O carregamento de classes pode ser menos otimizado, pois todo o classpath é examinado.

PACOTES x MÓDULOS

- **Resumo**

Característica	Módulos	Pacotes
Facilita a Distribuição e Empacotamento	Módulos podem ser empacotados com todas as suas dependências, facilitando a distribuição.	Dependências podem precisar ser gerenciadas manualmente, o que pode complicar a distribuição.
Introduzido no Java	Java 9 (Jigsaw Project)	Versões iniciais do Java

JLINK

- **Definição**

- O **jlink** é uma ferramenta introduzida no Java 9 que permite criar imagens personalizadas da JRE que incluem apenas os módulos necessários para uma aplicação específica. Isso ajuda a reduzir o tamanho da distribuição da sua aplicação e a torna mais eficiente, pois não inclui módulos desnecessários.
- A principal finalidade do jlink é facilitar a criação de distribuições JRE leves e personalizadas para aplicações Java. Antes do Java 9, você precisaria distribuir a JRE completa com sua aplicação, o que poderia ser grande e incluir módulos que sua aplicação não utilizava. Com o jlink, é possível criar uma JRE customizada que inclui apenas os módulos necessários para a sua aplicação, reduzindo o tamanho da distribuição e, por conseguinte, o espaço em disco e o tempo de download.

JLINK

- **Comando**

```
jlink --module-path $JAVA_HOME/jmods:meus-modulos  
--add-modules meu.modulo --output minha-jre
```

- Depois de criar a imagem customizada com o jlink, é possível distribuí-la com sua aplicação. Isso ajuda a garantir que a JRE necessária esteja disponível, independentemente da versão do Java instalada no sistema do usuário.



Formato Java ARchive

Formato .JAR

- O formato .JAR (**J**ava **AR**chive) possui um conjunto de classes e arquivos de configurações compactados, no estilo de um arquivo **zip**. A hierarquia de diretórios dos pacotes é mantida.

```
Files contained in jdbc.jar:

META-INF/INDEX.LIST
META-INF/MANIFEST.MF
com.mysql.jdbc.AssertionFailedException.class
com.mysql.jdbc.Blob.class
com.mysql.jdbc.BlobFromLocator.class
com.mysql.jdbc.Buffer.class
com.mysql.jdbc.CallableStatement.class
com.mysql.jdbc.CharsetMapping.class
com.mysql.jdbc.Clob.class
com.mysql.jdbc.CommunicationsException.class
com.mysql.jdbc.CompressedInputStream.class
com.mysql.jdbc.Connection.class
com.mysql.jdbc.ConnectionFeatureNotAvailableException.class
com.mysql.jdbc.ConnectionProperties.class
com.mysql.jdbc.ConnectionPropertiesTransform.class
com.mysql.jdbc.Constants.class
com.mysql.jdbc.CursorRowProvider.class
```

- O arquivo **jar** pode ser criado por qualquer compactador **zip** disponível no mercado, inclusive o aplicativo **jar** que vem junto com o JDK.

Formato .JAR

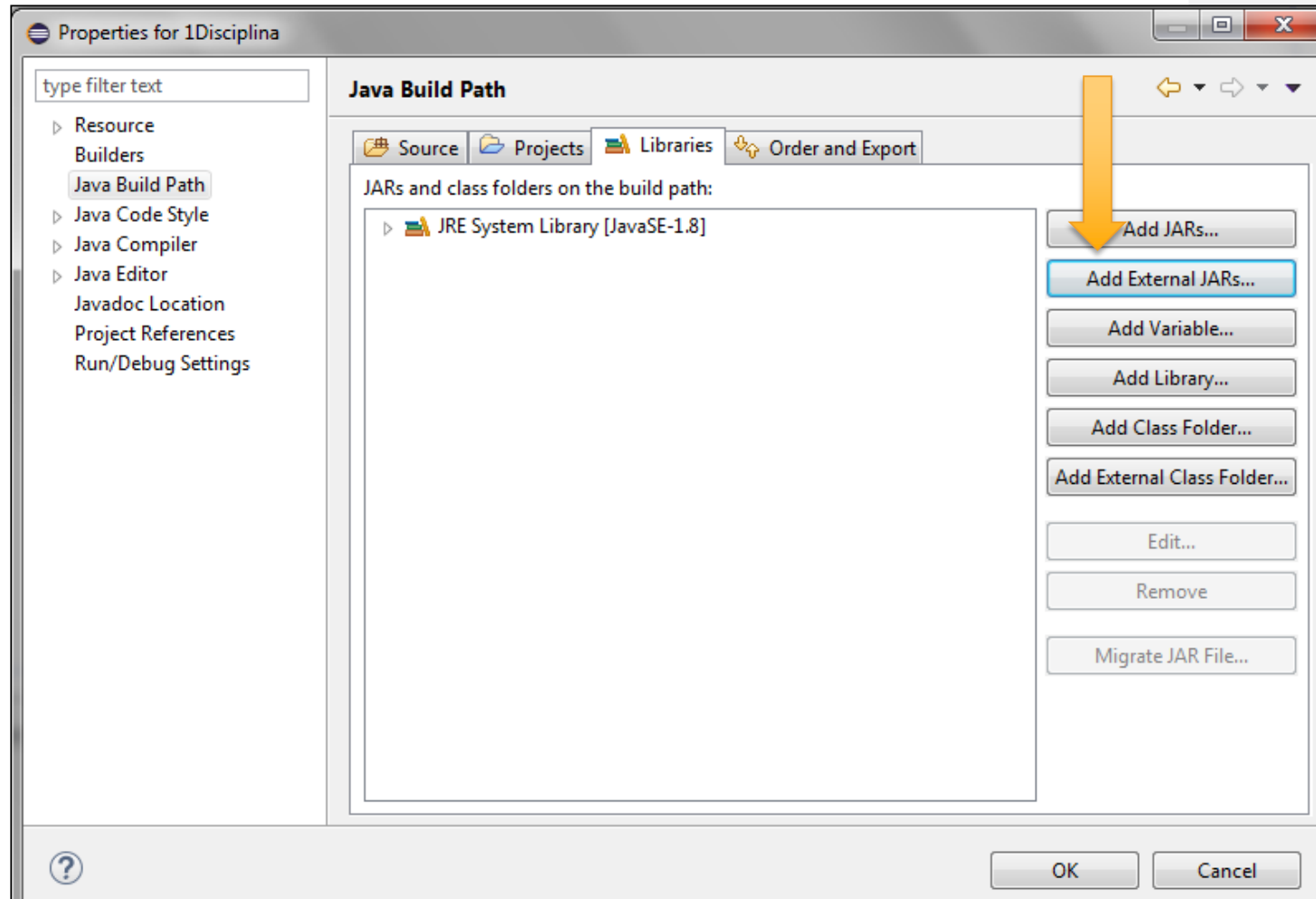
- É muito comum a distribuição de aplicações Java em arquivos **.jar**.
- Para usar um arquivo **jar** é necessário que o mesmo esteja no CLASSPATH da aplicação.
- O **CLASSPATH** é uma variável de ambiente que informa onde os arquivos **.jar** utilizados estão armazenados.
- Os mecanismos mais comuns para a configuração do CLASSPATH no ambiente de execução Java são três:

1) via parâmetro **-classpath** do interpretador Java

```
java -classpath biblioteca.jar;biblioteca2.jar  
NomeDaClasse
```

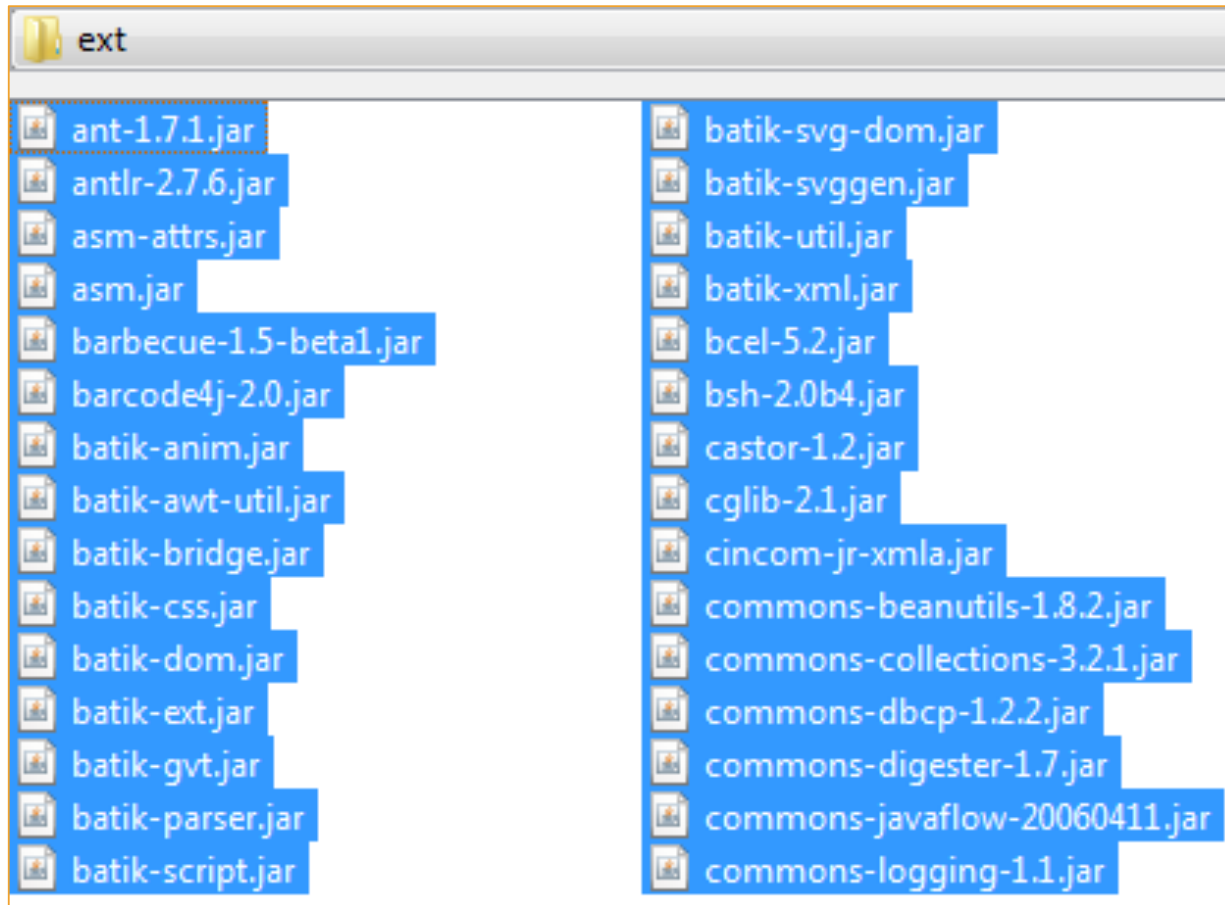
Formato .JAR

2) Incluindo o(s) arquivo(s) .jar no projeto Eclipse



Formato .JAR

3) Copiando o arquivo jar no diretório **%JAVA_HOME%/jre/lib/ext**



Exercícios

- 1) Criar o arquivo **abc.jar** a partir do pacote **br.abctreinamentos**.
- 2) Criar um novo projeto Java **TesteJar** e copiar para o mesmo a classe **Mec**. Quais são os erros apresentados?
- 3) Corrigir os erros acima com a importação do arquivo **abc.jar** no CLASSPATH do projeto **TesteJar**.



Erros e Exceções

Erros de Software Conhecidos

1. Bug do Milênio (1999)

Custo: \$500 bilhões

Desastre: Empresas gastaram bilhões com programadores para corrigir uma falha no software legado. Embora nenhuma falha significativa tenha ocorrido, a preparação para o Bug do Milênio teve um custo significativo para todas as indústrias que fazem uso intensivo de tecnologia computacional.

Causa: Para economizar espaço de armazenamento de computador, softwares legados muitas vezes armazenavam anos para datas com números de dois dígitos, como 99 para 1999. Esses softwares também interpretavam 00 para significar 1900, em vez de 2000, por isso, quando o ano de 2000 veio, bugs apareceriam.

Fonte: <http://www.profissionaisti.com.br/2012/01/alguns-dos-mais-famosos-erros-de-softwares-da-historia/>

Erros de Software Conhecidos

2. Tratamento de Câncer Mortal (2000)

Custo: 8 pessoas mortas e 20 seriamente feridas

Desastre: O software de radiação da empresa Multidate calculou mal a dosagem de radiação que deveria ser enviada, expondo pacientes a níveis fatais de radiação. Os físicos que foram indicados para checar as máquinas foram condenados à morte.

Causa: O software calculava a dosagem de radiação baseando-se na ordem de entrada dos dados, e algumas vezes, enviava o dobro da dose recomendada.

Erros de Software Conhecidos

3. Destruição do Foguete Ariane (1996)

Custo: \$500 milhões

Desastre: Ariane 5, o mais novo foguete da Europa não-tripulado, foi intencionalmente destruído segundos após seu lançamento em seu voo inaugural.

Causa: O desligamento ocorreu quando o computador de orientação tentou converter a velocidade do foguete de 64-bits para um formato de 16 bits. O número era muito grande, o que resultou em erro de estouro. Quando o sistema de orientação desligou, o controle passou para uma unidade idêntica redundante, que também falhou porque nele estava correndo o mesmo algoritmo.

Erros de Software Conhecidos

4. Sistema 'Patriot' Mata Soldados (1991)

Custo: 28 soldados mortos e 100 feridos.

Desastre: Durante a primeira Guerra do Golfo, o sistema de defesa americano Patriot localizado na Arábia Saudita falhou ao interceptar um míssil vindo do Iraque. O míssil destruiu acampamentos americanos.

Causa: Um erro de arredondamento no software calculou incorretamente o tempo, fazendo com que o sistema Patriot ignorasse os mísseis Scud de entrada.

Erros de Software Conhecidos

5. Crash em Wall Street (1987)

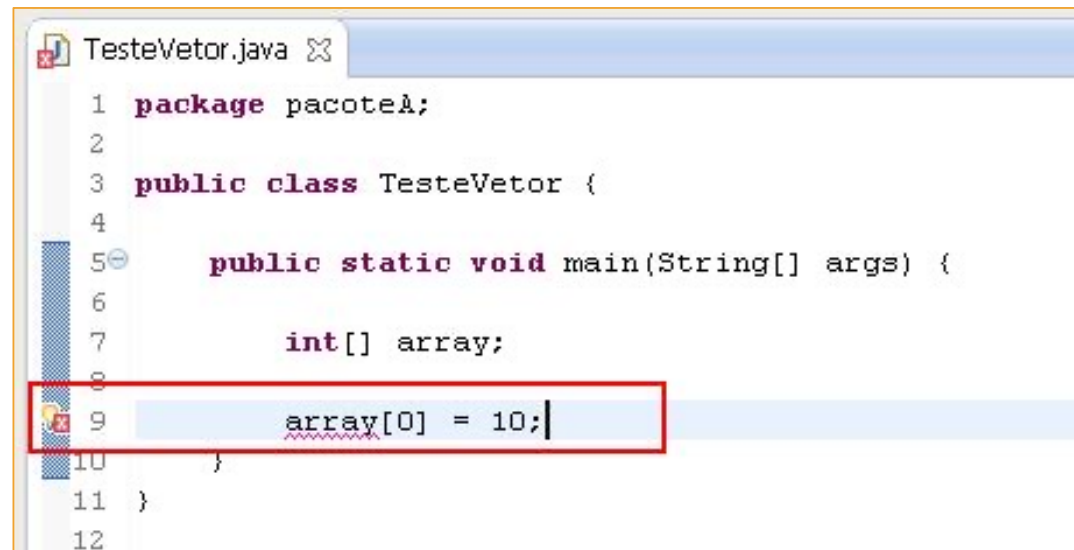
Custo: \$500 bilhões em um dia

Desastre: Em 19 de outubro de 1987, o índice Dow Jones caiu 508 pontos, perdendo 22,6% de seu valor total. Esta foi a maior perda que Wall Street já sofreu em um único dia.

Causa: O mercado americano foi atingido por uma série de investigações conduzidas pela SEC (*Securities and Exchange Commission*). Como os investidores fugiram de ações investigadas, um número muito grande de ordens de venda foi gerado pelos computadores, 'derrubando' os principais sistemas da bolsa de valores americana e deixando os investidores sem ação.

Erros de Software em Java

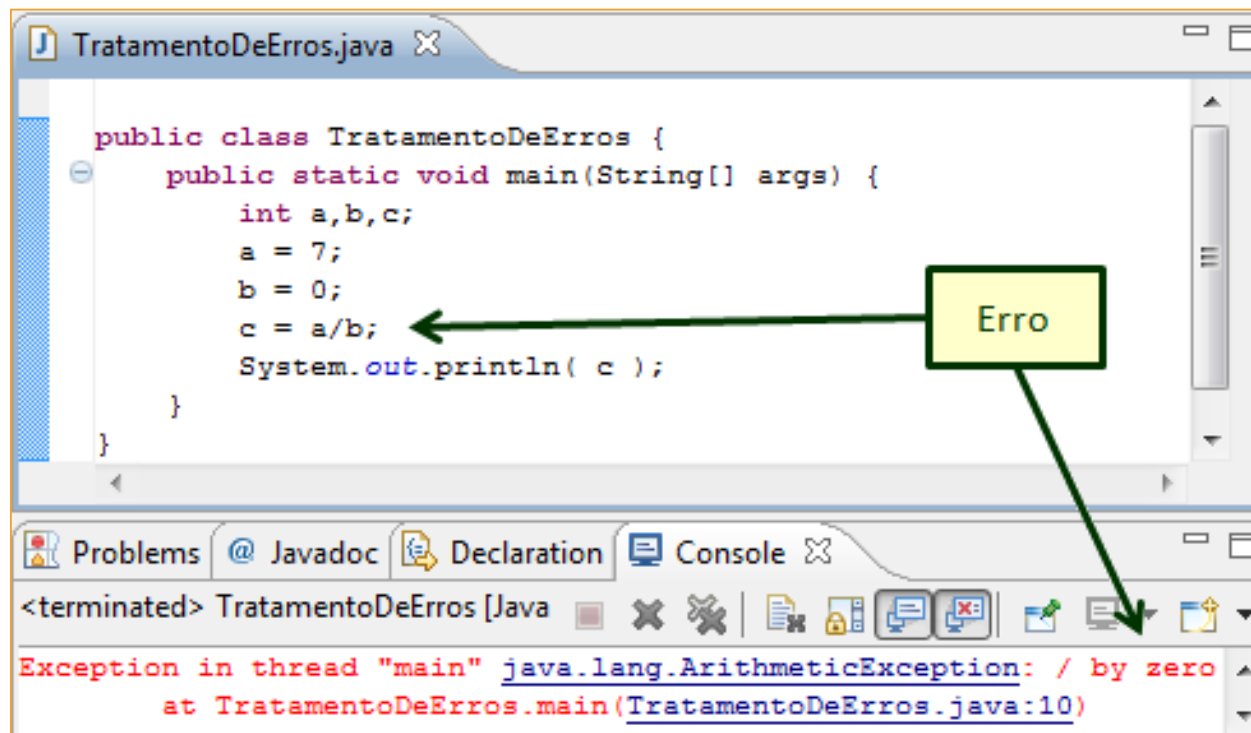
- São dois os principais tipos de erros que podem ocorrer no desenvolvimento de software em Java:
 - **Erros de Compilação**
 - **Erros em Tempo de Execução**
- **Erros de Compilação**
 - Problemas de sintaxe no código-fonte



```
TesteVetor.java ✕
1 package pacoteA;
2
3 public class TesteVetor {
4
5     public static void main(String[] args) {
6
7         int[] array;
8
9         array[0] = 10;|
10    }
11 }
12
```

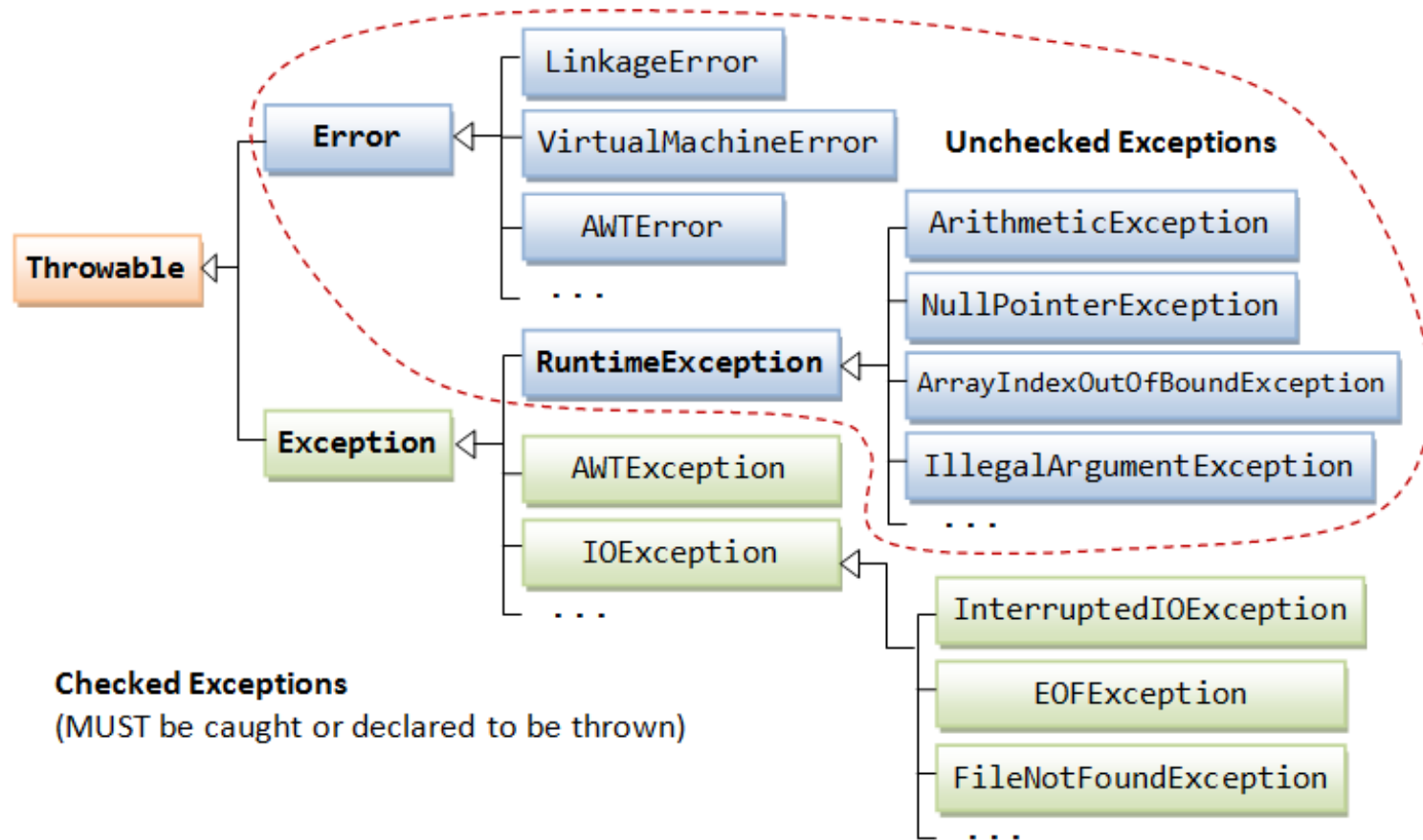
Erros em Java

- **Erros em Tempo de Execução**
 - Problemas na lógica de programação
 - Problemas no ambiente de execução



Exceções e Erros

- O Java define uma classe **Exception** utilizada para representar uma situação inesperada na aplicação, isto é, um **Erro em Tempo de Execução**.

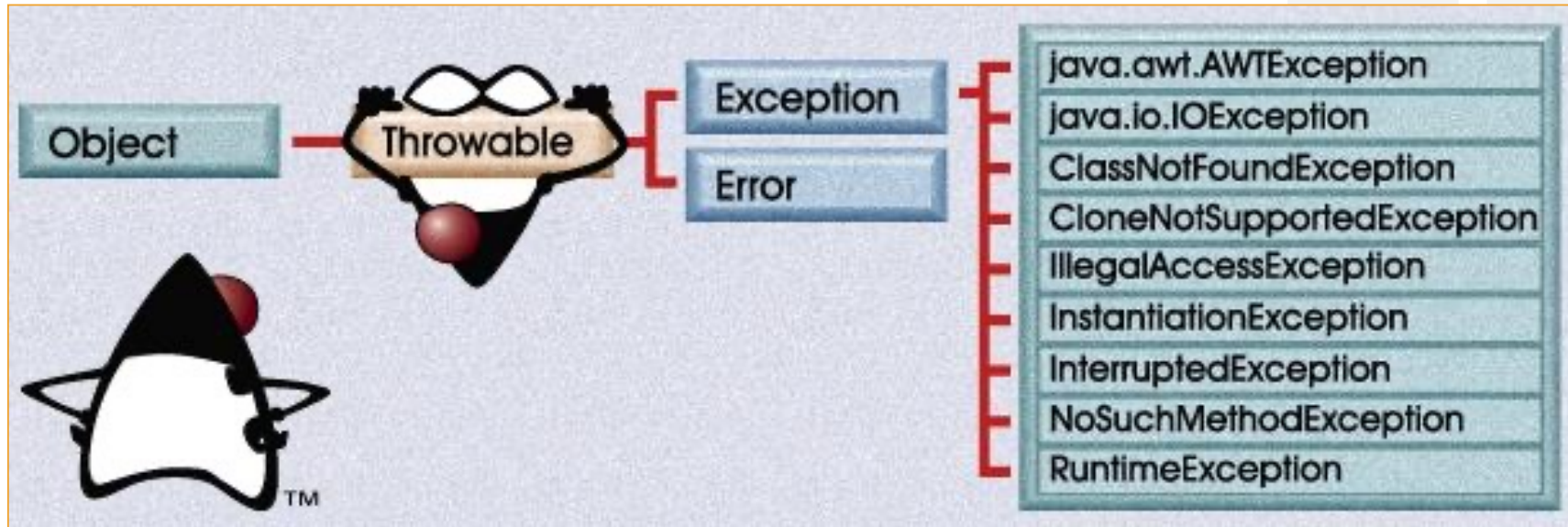


<http://www3.ntu.edu.sg>

Exceções e Erros

- *Unchecked Exceptions (UE)* são o grupo de Exceções que não obrigam o programador a fazer o tratamento prévio delas com uso de **try/catch** ou **throws**.
- *Checked Exceptions (CE)* são o grupo de Exceções que obrigam o programador a fazer o tratamento prévio delas com uso de **try/catch** ou **throws**.
- Exceções comuns:
 - Divisão por zero (**ArithmeticException**) (UE)
 - Acesso a objeto não instanciado (**NullPointerException**) (UE)
 - Acesso a um índice inválido de array (**ArrayIndexOutOfBoundsException**) (UE)
 - Erro de acesso a dispositivo de E/S (**IOException**) (CE)
- O Java define uma classe **Error** que define as condições consideradas muito graves (tais como problemas na máquina virtual), que tem pouca possibilidade de serem recuperadas.

Exceções e Erros



- A classe **Throwable** é a superclasse de **Error** e **Exception**.
- A superclasse **Exception** define as subclasses que tratam **Erros em Tempo de Execução**.

Tratamento de Exceções

- Os Erros em Tempo de Execução do tipo *Checked Exceptions (CE)* devem ser obrigatoriamente tratados em uma aplicação Java.
- São duas as soluções utilizadas:
 - Tratar a Exceção com os blocos **try, catch e finally**
 - Declarar a Exceção com a palavra **throws**
- **Tratar a Exceção com os blocos try, catch e finally**

```
try {  
    // instruções: executa até linha onde ocorrer exceção  
} catch (TipoExcecao1 ex) {  
    // executa somente se ocorrer TipoExcecao1  
}  
catch (TipoExcecao2 ex) {  
    // executa somente se ocorrer TipoExcecao2  
}  
finally {  
    // executa sempre ...  
}  
  
// executa se exceção for capturada ou se não ocorrer
```

Exercícios

- 1) Dada a classe Teste abaixo, incluir o tratamento de Exceção adequado.

```
class Teste {  
    public static void main (String args[]) {  
        int i=0;  
        String frases[] = {"um","dois","três"};  
        while (i<4) { // o índice vai de 0 a 2 !  
            System.out.println(frases[i]);  
            i++;  
        }  
    }  
}
```

Exercícios

- 2) Dada a classe TesteLeitura abaixo, incluir o tratamento de Exceção adequado.

```
import java.io.*;
public class TesteLeitura {
    public static void main(String[] args) {
        byte [] bytes = new byte[10];
        System.out.print("Digite algo: ");
        System.in.read(bytes);
        System.out.println("\nEco: " + new String(bytes));
    }
}
```

Tratamento de Exceções

- **Multicatch**

```
try {  
    // código  
} catch (Exceção 1 | Exceção 2 | ... | Exceção n) {  
    // tratamento da exceção  
} finally {  
    //}
```

- O Multicatch é útil quando se deseja evitar código repetido.
- O **finally** é muito útil quando se trabalha com acesso a Banco de Dados ou manipulação de arquivos, pois é necessário que a aplicação libere os recursos alocados após o seu uso.
- A partir do Java 7, é possível fazer uso do **Gerenciamento Automático de Recursos**.

Tratamento de Exceções

- **Gerenciamento Automático de Recursos**

- O uso do **finally** é muito comum quando se trabalha com acesso a banco de dados, manipulação de arquivos, entre outras situações. São em casos como esses que é necessário que o programa adquira recursos do SO quando necessário e os libere posteriormente.
- Para tirar esse trabalho do desenvolvedor, a partir do Java 7 há o gerenciamento automático de recursos.

```
public static void main(String[] args)
                                throws Exception {
    try(MyBufferedReader br = new MyBufferedReader
(new FileReader("classes.txt"))) {
        String line;
        while ((line = br.readLine()) != null)
            instantiate(line);
    }
```

Tratamento de Exceções

- **Gerenciamento Automático de Recursos**
- Com a mudança, a declaração e a inicialização do `MyBufferedReader` são feitas no próprio `try`. Isso determina que a instância terá seu método `close()` chamado automaticamente, de forma semelhante àquela obtida quando usamos o `finally`. Isso certamente evita confusões e erros no momento de liberar recursos.
- Em decorrência da chamada implícita do `close()`, esse `try` deve fazer parte de outro `try` contendo um `catch` para `Exception`; ou então o método deve lançar `Exception`.

Exercício

- 1) Dada a classe TesteExcecao abaixo, tratar as exceções com uma única instrução catch.

```
import java.io.*;
public class TesteExcecao {
    public static void main(String args[]) {
        int i = 50;
        i = i / 0;
        System.out.println("O resultado " + i);
        Object c = null;
        System.out.println("O resultado " + c.toString());
    }
}
```

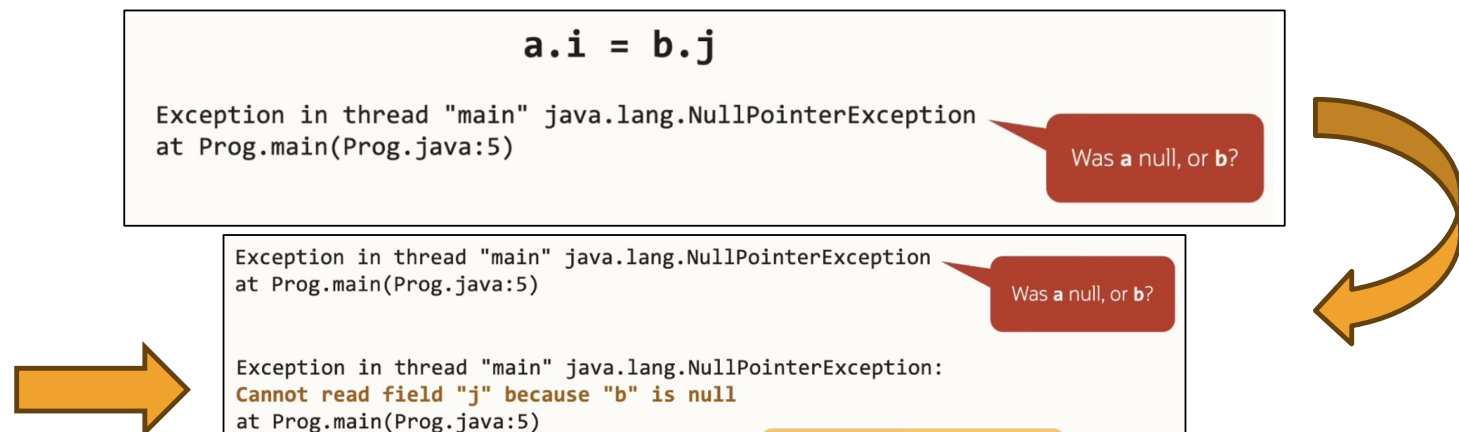


Helpful NullPointerException

HELPFUL NULLPOINTEREXCEPTION

- **Definição**

- A exceção *NullPointerException* ocorre quando se tenta acessar ou chamar um método em um objeto que possui o valor '*null*', ou seja, que não foi inicializado ou não aponta para uma instância válida de objeto.
- Até o Java 13, descobrir qual instância gerava esse tipo de exceção era um problema.
- A partir do Java 14, a JVM passou a descrever com precisão qual variável é nula.





Throws e Throw

Tratamento de Exceções

- Declarar a Exceção com a palavra **throws**
- A declaração **throws** é obrigatória em métodos e construtores que deixam de capturar uma ou mais exceções (do tipo *Checked Exceptions*) que ocorrem em seu interior.

```
public void m() throws Excecao1, Excecao2 {...}  
public Circulo() throws ExcecaoDeLimite {...}
```

- **throws** declara que o método pode provocar exceções do tipo declarado (ou de qualquer subtipo).

```
public static void main (String args[])  
    throws ArrayIndexOutOfBoundsException {  
    int i=0;  
    String frases[] = {"um", "dois", "tres"};  
    while (i<4) {i++};  
}
```

Novas Exceções

- É possível criar novas exceções criando subclasses de **Exception**.

```
public class ServerTimeoutException extends Exception
{
    ...
}
```

- Para 'lançar' as próprias exceções, é necessário utilizar a palavra **throw**.

```
...
public connect( ) throws ServerTimeoutException
{
    if (sucess == -1)
        throw new
            ServerTimeoutException("Impossível conectar");
}
```

Novas Exceções

- **DIFERENÇAS DO `throws` e `throw`**
- **`throws`** declara que o método pode provocar exceções do tipo declarado. Não é obrigatório nas *Unchecked Exceptions*.
- **`throw`** utilizado para 'lançar' uma exceção específica ou a sua própria exceção no meio do código.

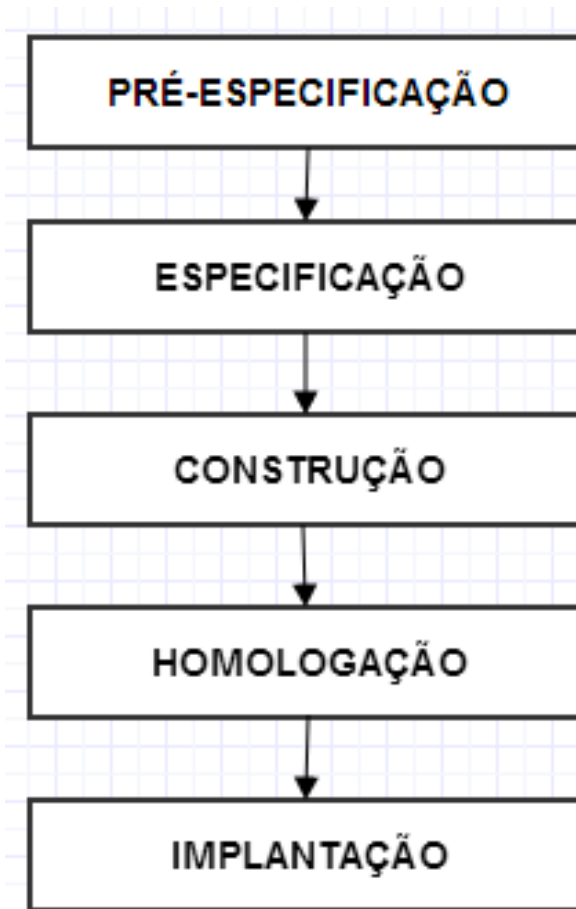
```
public void metodo() throws ExcecaoSimples {  
    try {  
        // instruções  
    } catch (ExcecaoSimples ex) {  
        // faz alguma coisa para lidar com a exceção  
        throw ex; // relança exceção  
    }  
}
```

Exercícios

- 1) Altere as classes **Teste** e **TesteLeitura** para fazer uso da palavra reservada **throws**.
- 2) Criar uma classe **Calculadora** que realiza a divisão de dois números. Realize o tratamento de exceção adequado, fazendo uso de **throws** e **throw**.
- 3) Criar uma exceção própria **DivisaoZeroException** e utilizar o **throw** para lançá-la.

Qualidade de Software

- **METODOLOGIA TRADICIONAL DE DESENVOLVIMENTO DE SOFTWARE**



Qualidade de Software

- Um bom Projeto de Software (Especificação) proporciona menos erros de codificação.
- É importante ressaltar que a Qualidade de um Software não pode ser mensurada apenas pela pouca incidência ou mesmo a inexistência de bugs, pois é possível ter um código sem erros porém difícil de utilizar e que não atende às necessidades dos usuários finais.

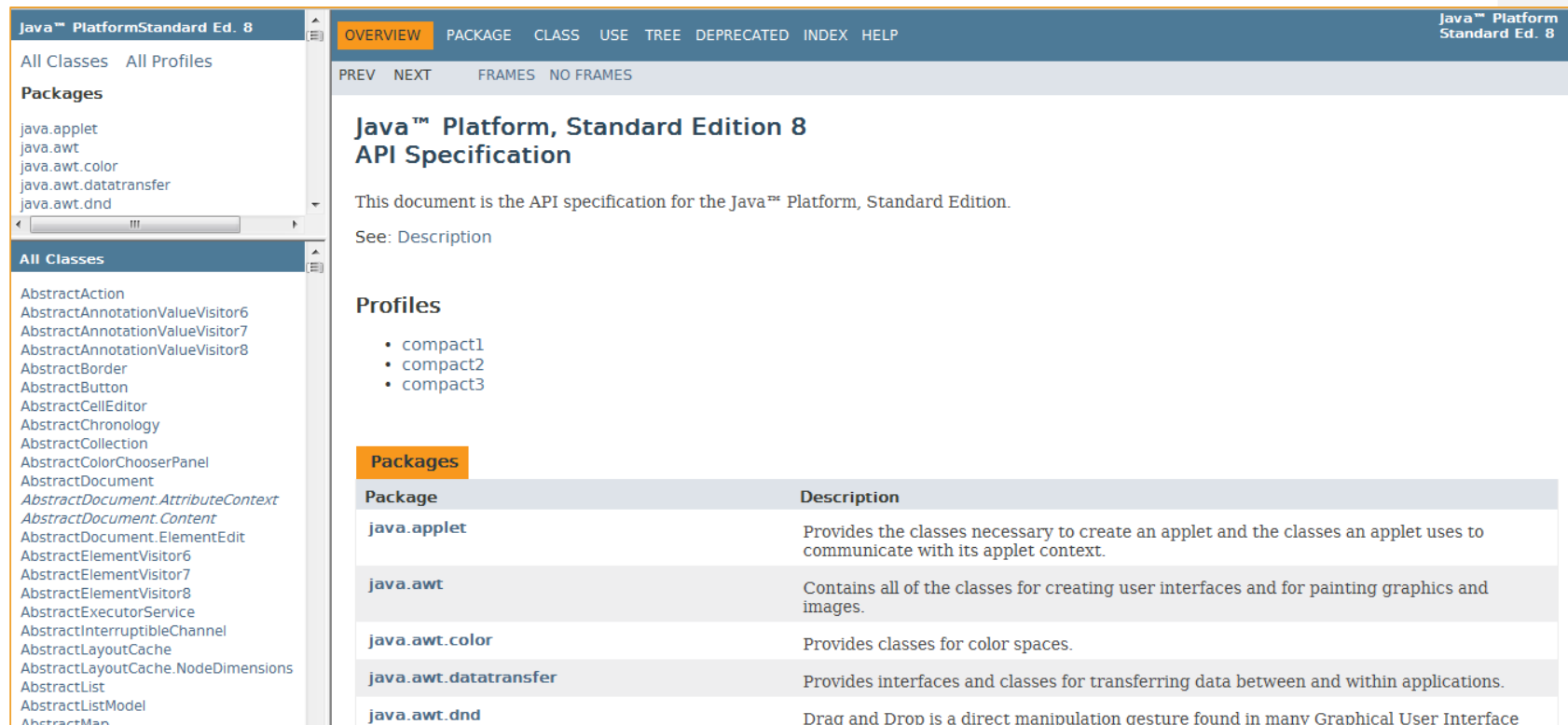
Dica: Pensar bem o Projeto de Software antes de construí-lo sempre ajuda e o torna mais barato de manter.



JavaDoc

Documentação com Javadoc

- O JDK fornece a ferramenta **javadoc** para a criação de documentação do software construído em Java para a Internet.
- A Java 8 API é criada com essa tecnologia.



The screenshot displays the Java Platform Standard Edition 8 API Specification website. The left sidebar contains a list of packages and classes. The main content area shows the 'Overview' tab with a table of packages.

Package	Description
java.applet	Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.
java.awt	Contains all of the classes for creating user interfaces and for painting graphics and images.
java.awt.color	Provides classes for color spaces.
java.awt.datatransfer	Provides interfaces and classes for transferring data between and within applications.
java.awt.dnd	Drag and Drop is a direct manipulation gesture found in many Graphical User Interface

Documentação com Javadoc

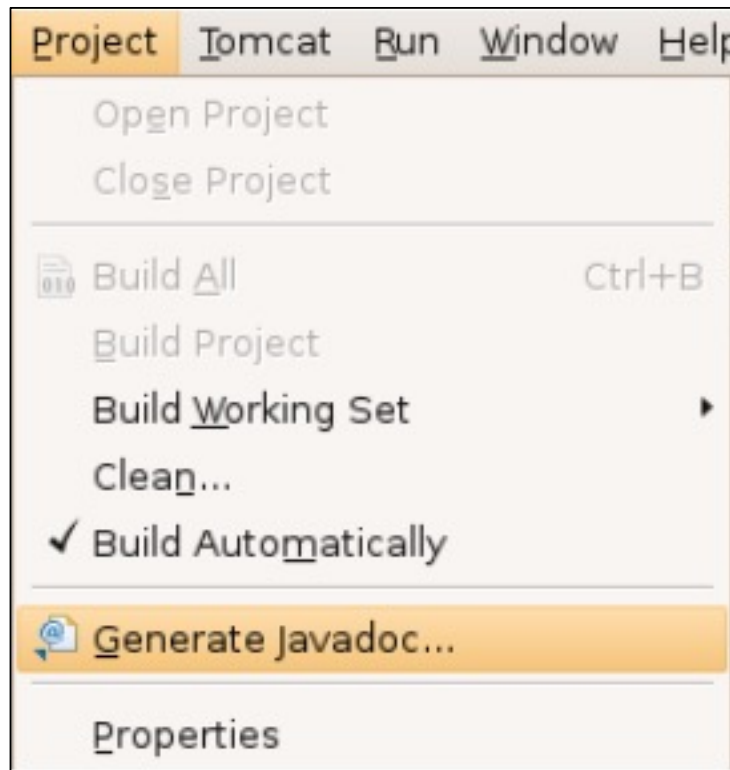
- Para o correto uso da ferramenta, torna-se necessário a inclusão de comentários na linha anterior à definição de classe, interface, construtor, método e atributo.
- Sintaxe Padrão

```
/**  
 * Classe  
 * @author Antonio Benedito  
 */
```

```
/**  
 * Método  
 * @param valor  
 */
```

Documentação com Javadoc

- Para executar o Javadoc a partir do Eclipse é muito simples. Basta selecionar no menu 'Project' a funcionalidade 'Generate Javadoc...'



- Posteriormente, o projeto desejado para gerar a documentação deverá ser selecionado.

Exercício

- 1) Criar a documentação do projeto **2Disciplina** com a ferramenta **javadoc**. Incluir os comentários necessários nas classes, interfaces, construtores, métodos e atributos do projeto em questão.



Novidades JavaDoc

Abaixo algumas melhorias no Javadoc

- **(1) Suporte a Módulos (Java 9+)**

```
/**
 * Este é um exemplo de documentação para um módulo no Java 9+.
 * @moduleNome meuModulo
 * @moduleVisão geral Este módulo fornece funcionalidades relacionadas a
 */
module meuModulo {
    // ...
}
```

- **(2) Estilos de HTML5 (Java 9+)**

- O Javadoc gerado a partir do Java 9 utiliza marcação HTML5 semântica e moderna.

Abaixo algumas melhorias no Javadoc

- **(3) Recursos de Pesquisa Aprimorados (Java 9+)**
 - O Javadoc no Java 9+ possui uma funcionalidade de pesquisa melhorada, permitindo aos usuários encontrar informações específicas mais facilmente.
- **(4) HTML Customizável (Java 9+)**
 - Os desenvolvedores podem personalizar a aparência do Javadoc gerado usando folhas de estilo CSS personalizadas.
- **(5) HTML5 e CSS3 (Java 7+)**
 - O Javadoc gerado a partir do Java 7 utiliza marcação HTML5 e suporta estilização através de CSS3.

Abaixo algumas melhorias no Javadoc

- (6) Suporte a Anotações (Java 8+)

```
/**
 * Exemplo de uso de anotação.
 * @Deprecated Use a classe {@link NovaClasse} em vez desta.
 */
@Deprecated
public class ClasseAntiga {
    // ...
}
```

Abaixo algumas melhorias no Javadoc

- **(7) Inclusão de Comentários Lambda (Java 8+)**

```
/**
 * Executa a operação especificada no contexto desta thread.
 * Exemplo de uso:
 * {@code minhaThread.executar(() -> System.out.println("Operação
 */
public void executar(Runnable operacao) {
    // ...
}
```

Abaixo algumas melhorias no Javadoc

- **(8) Suporte a Referências de Método (Java 8+)**

```
/**
 * Obtém o comprimento da string.
 * Exemplo de uso:
 * {@code int tamanho = minhaString.length();}
 */
public int length() {
    // ...
}
```

Exercício

- 1) Criar uma nova documentação do projeto **cursoLambdasStreams** com a ferramenta **javadoc**. Incluir alguns dos novos recursos apresentados nesta aula.