


Laboratório de Engenharia de Software

INF1636 – PROGRAMAÇÃO ORIENTADA A OBJETOS

Departamento de Informática – PUC-Rio

Ivan Mathias Filho
ivan@inf.puc-rio.br

Programa – Capítulo 2




Laboratório de Engenharia de Software

- Visão Geral da Tecnologia Java
- Aplicação **Hello World!**
- Operadores e Comandos da Linguagem Java
- A Ferramenta Eclipse

© LES/PUC-Rio 2

Programa – Capítulo 2




Laboratório de Engenharia de Software

- **Visão Geral da Tecnologia Java**
- Aplicação **Hello World!**
- Operadores e Comandos da Linguagem Java
- A Ferramenta Eclipse

© LES/PUC-Rio

3

A Linguagem Java




Laboratório de Engenharia de Software

Orientada a Objetos

Multithreaded

Portável


ORACLE

Distribuída

Segura

Robusta

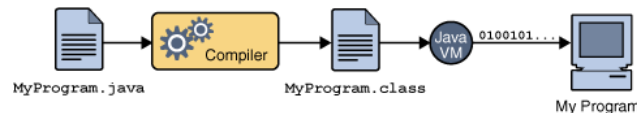
© LES/PUC-Rio

4

O desenvolvimento de aplicações



- O código fonte é escrito em arquivos ASCII puros com a extensão .java ;
- Os arquivos fontes são compilados para arquivos .class pelo compilador Java (javac);
- Os arquivos .class contêm bytecodes – a linguagem de máquina da Java Virtual Machine (JVM);
- Os arquivos .class são carregados e executados por uma instância da máquina virtual Java (JVM).



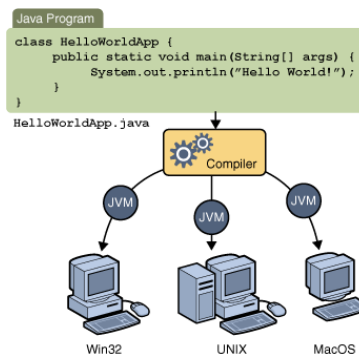
© LES/PUC-Rio

5

Suporte a múltiplas plataformas




- A máquina virtual Java está disponível para vários sistemas operacionais;
- Isso permite que um mesmo conjunto de arquivos .class possa ser executado em diferentes plataformas.



© LES/PUC-Rio

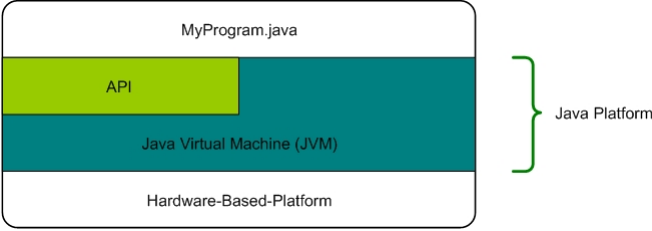
6

A plataforma Java




Laboratório de Engenharia de Software

- A plataforma Java possui dois componentes:
 - A máquina virtual Java (JVM);
 - A Application Programming Interface (API).



© LES/PUC-Rio
7


Edições da plataforma Java



Laboratório de Engenharia de Software

- **Micro Edition (Java ME):** fornece um ambiente robusto e flexível para aplicativos executados em dispositivos embarcados e móveis: microcontroladores, sensores, gateways, telefones celulares, assistentes digitais pessoais (PDAs), aparelhos de TV, top boxes e impressoras.
- **Standard Edition (Java SE):** fornece a infraestrutura necessária para o desenvolvimento e a implantação de aplicativos Java em desktops e servidores, bem como em ambientes de tempo real. Inclui classes que apoiam o desenvolvimento de Web Services e fornece a base para a Enterprise Edition.
- **Enterprise Edition (Java EE):** conjunto de tecnologias para o desenvolvimento, implantação e gestão de aplicações multicamadas e centradas em servidores.

© LES/PUC-Rio
8

Elementos da plataforma		
Laboratório de Engenharia de Software	<ul style="list-style-type: none">• Ferramentas de Desenvolvimento: as principais são o compilador (javac), o interpretador (java) e a ferramenta de documentação (javadoc).• API: funcionalidades de Java prontas para uso em aplicações: GUIs, sockets, RMI, segurança, XML e banco de dados.• Tecnologias de Implantação: ferramentas (Web Start e Java Plug-In) para a implantação de sistemas para os usuários finais.• GUI: bibliotecas de classes (Swing e Java 2D) para a criação de sofisticadas interfaces gráficas (GUIs).• Bibliotecas de Integração: JDBC, JNDI, RMI e etc.	
	© LES/PUC-Rio	
	9	

Programa – Capítulo 2		
Laboratório de Engenharia de Software	<ul style="list-style-type: none">• Visão Geral da Tecnologia Java• Aplicação Hello World!• Operadores e Comandos da Linguagem Java• A Ferramenta Eclipse	
	© LES/PUC-Rio	
	10	

Requisitos

LES

Laboratório de Engenharia de Software

- Ferramentas necessárias na plataforma Windows:
 - Java SE Development Kit 17 (JDK 17, **não** JRE 17);
 - um editor de textos, como o NotePad;
 - em vez de um editor, é possível também utilizar um Integrated Development Environment (IDE): Eclipse, NetBeans e etc.
- Passos para criar a aplicação **HelloWorldApp**:
 - criar o código fonte:
 - texto na linguagem Java escrito no editor de textos ou no IDE;
 - compilar o código fonte:
 - o compilador **javac** traduz o código fonte para bytecodes;
 - executar a aplicação:
 - o interpretador **java** usa o JVM para executar os bytecodes.

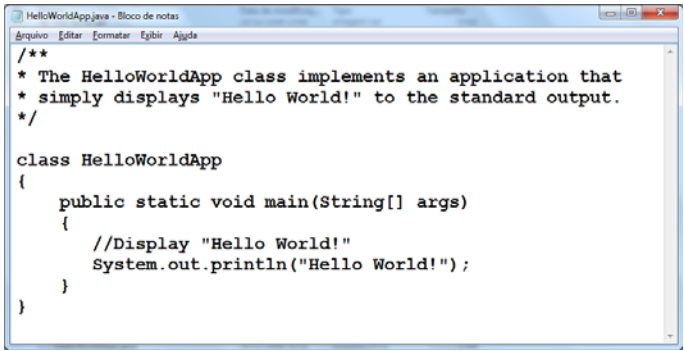
© LES/PUC-Rio

11

Edição

LES

Laboratório de Engenharia de Software



```

HelloWorldApp.java - Bloco de notas
Arquivo  Editar  Formatar  Exibir  Ajuda

/**
 * The HelloWorldApp class implements an application that
 * simply displays "Hello World!" to the standard output.
 */

class HelloWorldApp
{
    public static void main(String[] args)
    {
        //Display "Hello World!"
        System.out.println("Hello World!");
    }
}

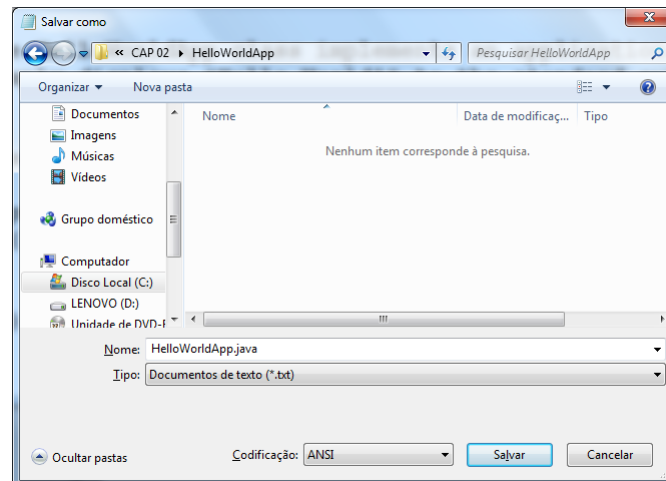
```

- O arquivo texto tem de ter o mesmo nome da classe principal (**HelloWorldApp**) – a que contém o método **main**

© LES/PUC-Rio

12

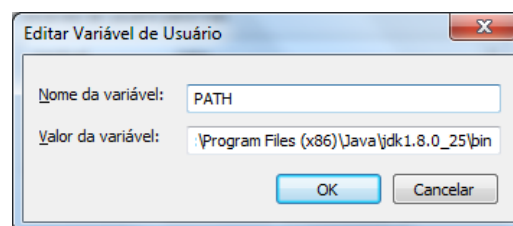
Salvar o código fonte



© LES/PUC-Rio

13

Path

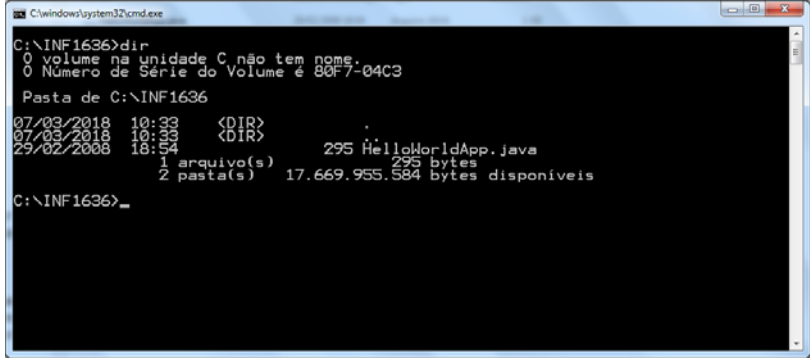


- Para compilar e executar uma aplicação Java é recomendável colocar o diretório **\\jdk\\bin** na variável de sistema **path**

© LES/PUC-Rio

14

O diretório da aplicação



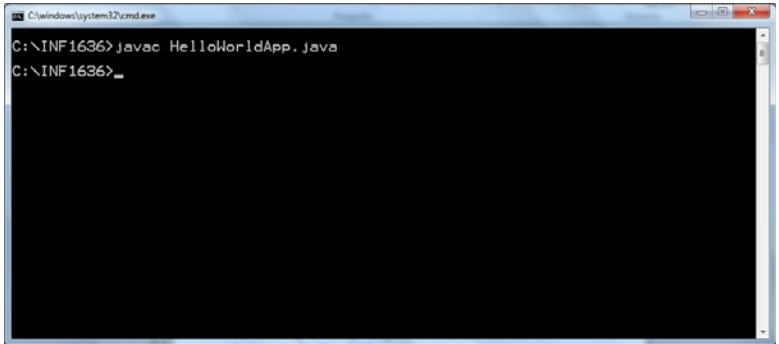
```
C:\windows\system32\cmd.exe
C:\INF1636>dir
0 volume na unidade C não tem nome.
0 Número de Série do Volume é 80F7-04C3

Pasta de C:\INF1636
07/03/2018 10:33 <DIR> .
07/03/2018 10:33 <DIR> ..
29/02/2008 18:54      295 HelloWorldApp.java
                295 bytes
                2 pasta(s) 17.669.955.584 bytes disponíveis

C:\INF1636>
```

© LES/PUC-Rio 15

Compilação da aplicação



```
C:\windows\system32\cmd.exe
C:\INF1636>javac HelloWorldApp.java
C:\INF1636>
```

- Para compilar a aplicação, execute o comando **javac HelloWorldApp.java**

© LES/PUC-Rio 16

O arquivo .class



- O compilador gera o arquivo **HelloWorldApp.class**

```
C:\windows\system32\cmd.exe
C:\INF1636>javac HelloWorldApp.java
C:\INF1636>dir
0 volume na unidade C: não tem nome.
0 Número de Série do Volume é 80F7-04C3

Pasta de C:\INF1636
07/03/2018 10:35 <DIR>      .
07/03/2018 10:35 <DIR>      ..
07/03/2018 10:35             432 HelloWorldApp.class
29/02/2008 18:54             295 HelloWorldApp.java
                727 bytes
2 arquivo(s)
2 pasta(s) 17.664.180.224 bytes disponíveis

C:\INF1636>
```

© LES/PUC-Rio

17

Execução da aplicação




- Para executar a aplicação, execute o comando **java HelloWorldApp**

```
C:\windows\system32\cmd.exe
C:\INF1636>java HelloWorldApp
Hello World!
C:\INF1636>
```

© LES/PUC-Rio

18

Análise do código



Laboratório de Engenharia de Software

```

/**
 * The HelloWorldApp class implements an application that
 * simply displays "Hello World!" to the standard output.
 */


class HelloWorldApp {
    public static void main(String[] args) {
        //Display "Hello World!"
        System.out.println("Hello World!");
    }
}

```

- ← Classe
- ← Método main
- ← Comentários
- ← Comandos Java

© LES/PUC-Rio 19

Comentários




Laboratório de Engenharia de Software

- Os comentários utilizados são de dois tipos:
 - */** documentation */*
 - Comentário de documentação, usado pela ferramenta **javadoc** para gerar documentação automaticamente.
 - *// text*
 - O compilador ignora tudo a partir de *//* até o final da linha.
- Existe ainda o */* text */*, significando que o compilador deve ignorar o texto em uma ou mais linhas entre */** e **/*.

© LES/PUC-Rio 20

Laboratório de Engenharia de Software

O método `main` (1)




- Toda aplicação (**não** toda classe!) deve possuir um método **`public static void main(String[] args) {...}`**
- O argumento pode ter qualquer nome, mas **`args`** é padrão;
- Os modificadores **`public`** e **`static`** podem ser escritos em qualquer ordem, mas **`public static`** é padrão;
- O cabeçalho do método contém três modificadores:
 - **`public`** - pode ser invocado por qualquer objeto;
 - **`static`** - método de classe (oposto a método de instância);
 - **`void`** - o método não retorna valor algum.

© LES/PUC-Rio

21

Laboratório de Engenharia de Software

O método `main` (2)




- Quando o interpretador executa uma aplicação, começa por chamar o método **`main`**, que, por sua vez, chama os demais métodos existentes na aplicação.

© LES/PUC-Rio

22

Laboratório de Engenharia de Software

A classe `System`



- **HelloWorldApp** usa **System**, classe Java que fornece funcionalidades independentes de plataforma;
- Em **HelloWorldApp**, o comando `System.out.println()` ilustra o uso de uma variável de classe e de um método de instância de **System** (`out` e `println`, respectivamente);
- Variáveis e métodos de classe são precedidos, na declaração, pelo modificador de acesso **static**;
- **out** é uma variável de classe que se refere a uma instância de **PrintStream**, classe Java que implementa a saída padrão.

© LES/PUC-Rio

23

Laboratório de Engenharia de Software

Programa – Capítulo 2



- Visão Geral da Tecnologia Java
- Aplicação **Hello World!**
- **Operadores e Comandos da Linguagem Java**
- A Ferramenta Eclipse

© LES/PUC-Rio

24

Tipos primitivos			LES
Tipo	Descrição	Formato	
<i>Integers</i>			
byte	Byte-length integer	8-bit two's complement	
short	Short integer	16-bit two's complement	
int	Integer	32-bit two's complement	
long	Long integer	64-bit two's complement	
<i>Real numbers</i>			
float	Single-precision floating point	32-bit IEEE 754	
double	Double-precision floating point	64-bit IEEE 754	
<i>Other types</i>			
char	A single character	16-bit Unicode character	
boolean	A boolean value (true or false)	true or false	

© LES/PUC-Rio

25

LES

O modificador final

- A linguagem C usa a palavra chave **const** para definir uma constante (**const int x=100**);
- Java, por sua vez, usa a palavra chave **final** com o mesmo propósito;
- Uma variável é dita **final** quando seu valor não pode ser alterado após sua inicialização;
- A inicialização de uma variável final pode ser feita posteriormente à sua declaração.

Laboratório de Engenharia de Software


© LES/PUC-Rio

26

© LES/PUC-Rio

26

Operadores aritméticos (1)			Laboratório de Engenharia de Software
Operador	Uso	Descrição	
+	op1 + op2	Adiciona op1 e op2; também usado para concatenação de strings	
-	op1 - op2	Subtrai op2 de op1	
*	op1 * op2	Multiplica op1 por op2	
/	op1 / op2	Divide op1 por op2	
%	op1 % op2	Calcula resto da divisão de op1 por op2	
© LES/PUC-Rio			27



Operadores aritméticos (2)


Tipo do Resultado	Tipos dos Operandos
long	Nenhum operando é float ou double (aritmética de inteiros) e pelo menos um é long .
int	Nenhum operando é float , double ou long .
double	Pelo menos um operando é double .
float	Pelo menos um operando é float e nenhum operando é double .

Quando um inteiro e um real são usados como operandos em uma mesma operação aritmética, o resultado é um real.

© LES/PUC-Rio

28

Laboratório de Engenharia de Software




Operadores aritméticos (3)

Operador	Uso	Descrição
++	op++	Incremento de 1; avalia o valor de op antes do incremento
++	++op	Incremento de 1; avalia o valor de op depois do incremento
--	op--	Decremento de 1; avalia o valor de op antes do incremento
--	--op	Decremento de 1; avalia o valor de op depois do decremento

© LES/PUC-Rio

29

Laboratório de Engenharia de Software




Operadores relacionais

Operador	Uso	Descrição
>	op1 > op2	Retorna true se op1 é maior que op2
>=	op1 >= op2	Retorna true se op1 é maior/igual a op2
<	op1 < op2	Retorna true se op1 é menor que op2
<=	op1 <= op2	Retorna true se op1 is menor que ou igual a op2
==	op1 == op2	Retorna true se op1 e op2 são iguais
!=	op1 != op2	Retorna true se op1 e op2 não são iguais

Laboratório de Engenharia de Software

© LES/PUC-Rio

30



Operadores lógicos

Operador	Uso	Descrição
&&	op1 && op2	Retorna true se op1 e op2 são ambos verdadeiros; condicionalmente avalia op2
	op1 op2	Retorna true se op1 ou op2 é verdadeiro; condicionalmente avalia op2
!	!op	Retorna true se op é false


Laboratório de Engenharia de Software

© LES/PUC-Rio

31

© LES/PUC-Rio

31



Operadores de atribuição

Laboratório de Engenharia de Software


Operador	Uso	Descrição
=	op1 = op2	Atribui o valor da direita à esquerda
+=	op1 += op2	Equivalente a op1 = op1 + op2
-=	op1 -= op2	Equivalente a op1 = op1 - op2
*=	op1 *= op2	Equivalente a op1 = op1 * op2
/=	op1 /= op2	Equivalente a op1 = op1 / op2
%=	op1 %= op2	Equivalente a op1 = op1 % op2
&=	op1 &= op2	Equivalente a op1 = op1 & op2
=	op1 = op2	Equivalente a op1 = op1 op2
^=	op1 ^= op2	Equivalente a op1 = op1 ^ op2

© LES/PUC-Rio

32

© LES/PUC-Rio

32




Outros operadores

Operador	Uso	Descrição
?:	op1 ? op2 : op3	Se op1 é true, retorna op2; se não, retorna op3
[]	Vetores	Usado para criar vetores, bem como acessar seus elementos
(<i>params</i>)	Métodos	Delimita um lista de parâmetros separados por vírgula
(<i>type</i>)	(<i>type</i>) op	Converte op para o tipo especificado; uma exceção é levantada se o tipo de op é incompatível com <i>type</i>
new	Objetos e Vetores	Cria um novo objeto ou vetor
instanceof	op1 instanceof op2	Retorna true se op1 é uma instância de op2

© LES/PUC-Rio

33

Laboratório de Engenharia de Software

LES

Comandos de controle de fluxo

Tipo de Comando	Palavra-reservada
Repetição	while, do-while, for
Decisão	if-else, switch-case
Tratamento de Exceção	try-catch-finally, throw
Desvio de Fluxo	break, continue, label: , return

Laboratório de Engenharia de Software

© LES/PUC-Rio

34

Uso de expressões booleanas



- A linguagem Java possui um tipo booleano (**boolean**), cujas constantes são **true** e **false**;
- Dessa forma, as expressões associadas aos comandos **if** e **while** devem resultar em valores booleanos, e não em valores integrais, como é o caso da linguagem C.

```
int x=10;
```

```
while(x>0)
```

```
{
    System.out.println(x);
    x--;
}
```

Certo

```
int x=10;
```

```
while(x)
```

```
{
    System.out.println(x);
    x--;
}
```

Errado

© LES/PUC-Rio

35

Saída no console – Funções println e printf



```
public class Main
{
    public static void main(String[] args)
    {
        System.out.println("Exemplos" + " - " + "Varios Formatos\n");
        System.out.printf("%d %d %d %05d\n", 3, -3, 3, 3);
        System.out.println();
        System.out.printf("%f\n", 1234567.123);
        System.out.printf("%,f\n", 1234567.123);
        System.out.printf("%,f\n", -1234567.123);
        System.out.printf("%,(f\n", -1234567.123);
        System.out.println();
        System.out.printf("% ,.2f\n% ,.2f\n", 234567.123, -1234567.123);
    }
}
```

Saída

Exemplos - Varios Formatos

3 (3) +3 00003

1234567,123000

1.234.567,123000

-1.234.567,123000

(1.234.567,123000)

234.567,12

-1.234.567,12

© LES/PUC-Rio

36

Entrada pelo teclado – classe Scanner



```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        int qtd, matric, maiorMatric=0;
        float nota, soma=0.0F, maiorNota=-1;

        Scanner e=new Scanner(System.in);
        System.out.println("Informe a quantidade de alunos\n");
        qtd=e.nextInt();

        for(int i=0;i<qtd;i++) {
            System.out.println("Informe a matricula e a nota\n");
            matric=e.nextInt();
            nota=e.nextFloat();
            soma+=nota;

            if(nota>maiorNota) {
                maiorNota=nota;
                maiorMatric=matric;
            }
        }
        System.out.printf("Media: %.2f\n", soma/qtd);
        System.out.printf("Maior nota: %d %.2f\n", maiorMatric, maiorNota);
    }
}
```

© LES/PUC-Rio

37

Programa – Capítulo 2



- Visão Geral da Tecnologia Java
- Aplicação **Hello World!**
- Operadores e Comandos da Linguagem Java
- **A Ferramenta Eclipse**

© LES/PUC-Rio

38

Eclipse – Primeiros passos

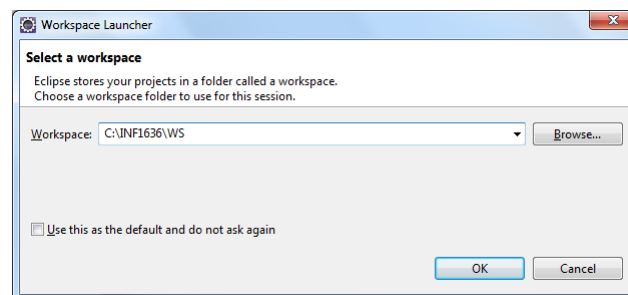


- O primeiro passo após a instalação do **Eclipse** é criar um **workspace**;
- Um **workspace** é uma pasta do sistema de arquivos;
- Ele guarda as pastas dos projetos, alguns arquivos de controle e bibliotecas usadas pelo próprio **Eclipse**;
- Pode-se utilizar quantos **workspaces** se achar necessário;
- Entretanto, apenas um **workspace** pode estar ativo em um dado momento.

© LES/PUC-Rio

39


Criação de um workspace



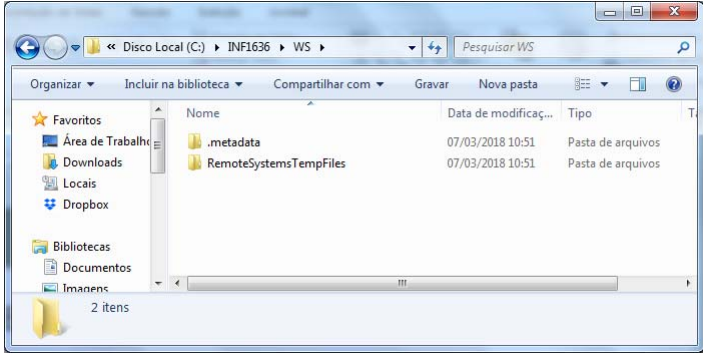
© LES/PUC-Rio

40

Estrutura de um workspace




Laboratório de Engenharia de Software



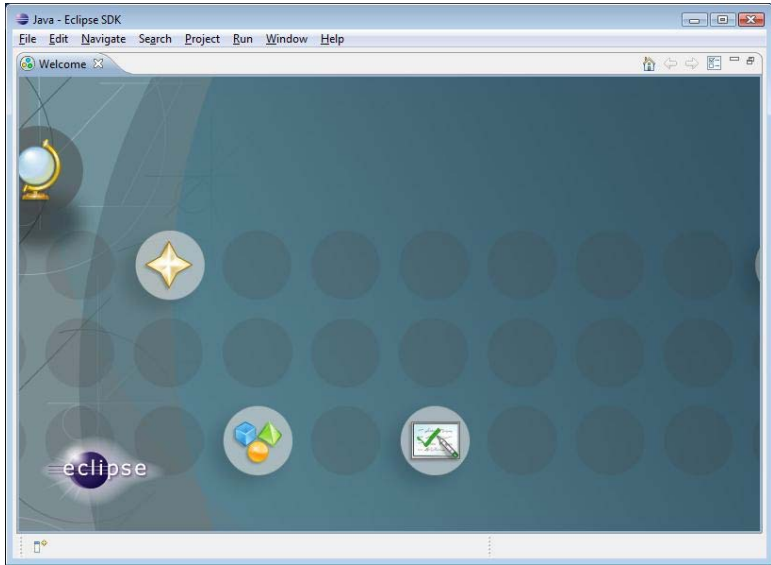
Nome	Data de modificação...	Tipo
.metadata	07/03/2018 10:51	Pasta de arquivos
RemoteSystemsTempFiles	07/03/2018 10:51	Pasta de arquivos

© LES/PUC-Rio 41

Novo workspace - Welcome



Laboratório de Engenharia de Software



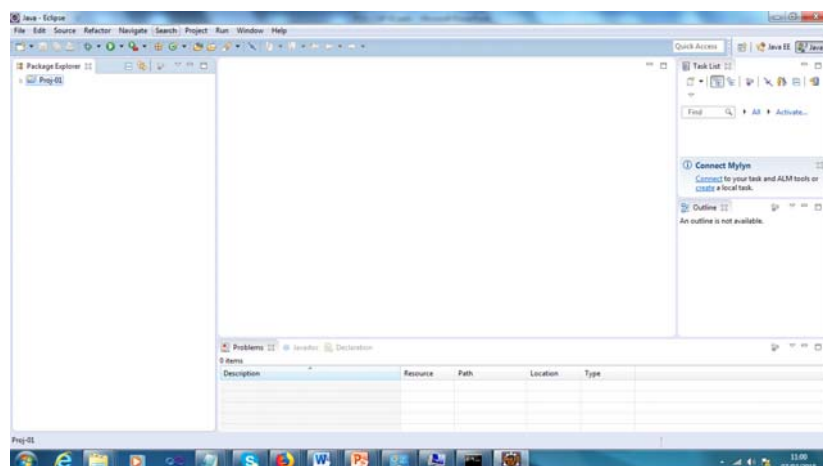
© LES/PUC-Rio 42


Perspectiva



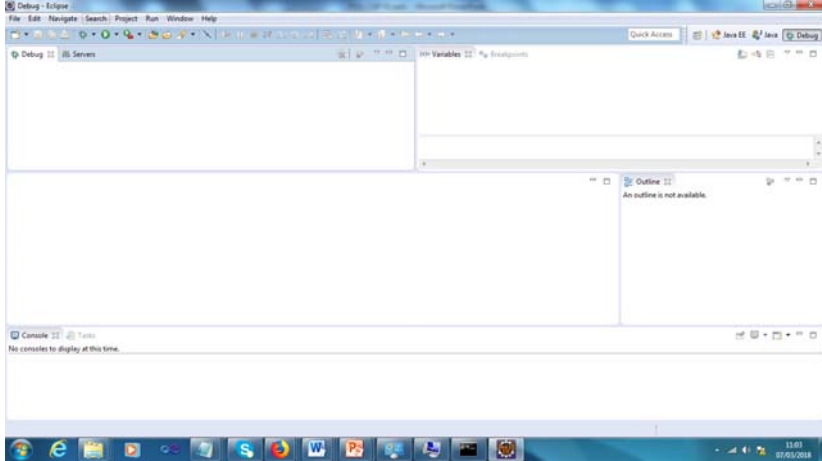
- Para realizar uma tarefa qualquer no Eclipse é preciso antes abrir uma **perspectiva**;
- Uma **perspectiva** é uma coleção de janelas e ferramentas adequadas à execução de uma tarefa específica;
- Duas das **perspectivas** que serão mais usadas no desenvolvimento de aplicações Java são:
 - **Java** – perspectiva usada para a codificação de aplicações Java;
 - **Debug** – perspectiva usada para a depuração de aplicações.

Perspectiva Java




Perspectiva Debug


Laboratório de Engenharia de Software



© LES/PUC-Rio
45

Projeto


Laboratório de Engenharia de Software

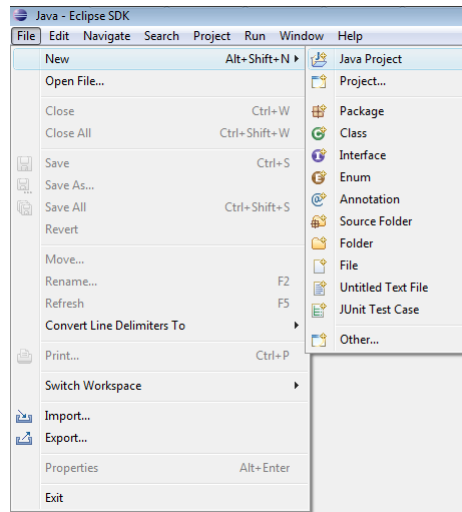
- O primeiro passo para o desenvolvimento de uma aplicação é criar um **projeto**;
- Um **projeto** é uma pasta do sistema de arquivos inserida na pasta relativa ao workspace;
- Um **projeto** contém o código fonte, o código binário e outros tipos de arquivos de uma aplicação;
- Um projeto pode ser criado de duas maneiras:

© LES/PUC-Rio
46

Criação de um projeto (1)



Por meio do menu **File**



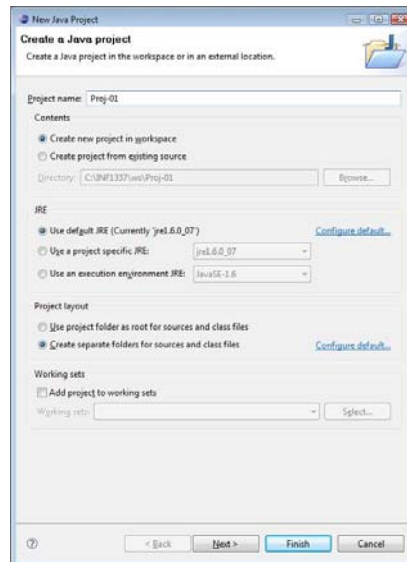
Criação de um projeto (2)



Por meio da **Barra de Ferramentas**



Criação de um projeto (3)



© LES/PUC-Rio

49

Criação da classe principal (1)

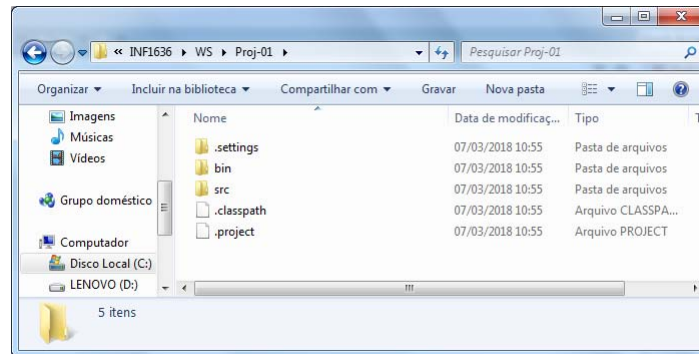


- O próximo passo é a criação da classe principal da aplicação, aquela que contém o método **main**;
- Isso pode ser feito por meio do menu **File** ou da **Barra de Ferramentas**;
- Para codificar uma aplicação deve-se usar os editores de código, que são dispostos no painel central da perspectiva Java.

© LES/PUC-Rio

50

Estrutura de um projeto



© LES/PUC-Rio

53

Execução da uma aplicação (1)



- Para executar uma aplicação deve-se usar o menu **Run** (CTRL+F11) ou a **Barra de Ferramentas**;
- Caso haja informação para ser lida ou exibida no console, deve-se usar a visão **Console**, localizada logo abaixo do painel dos editores.

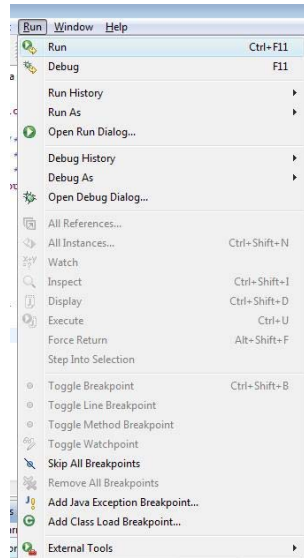
© LES/PUC-Rio

54

Execução da uma aplicação (2)



Por meio do menu **Run**



Execução da uma aplicação (3)



Por meio da **Barra de Ferramentas**



Exibição dos resultados no console

Laboratório de Engenharia de Software

© LES/PUC-Rio

57