

Programação Orientada a Objetos

Linguagem Java

Rone Ilídio
Thiago Oliveira

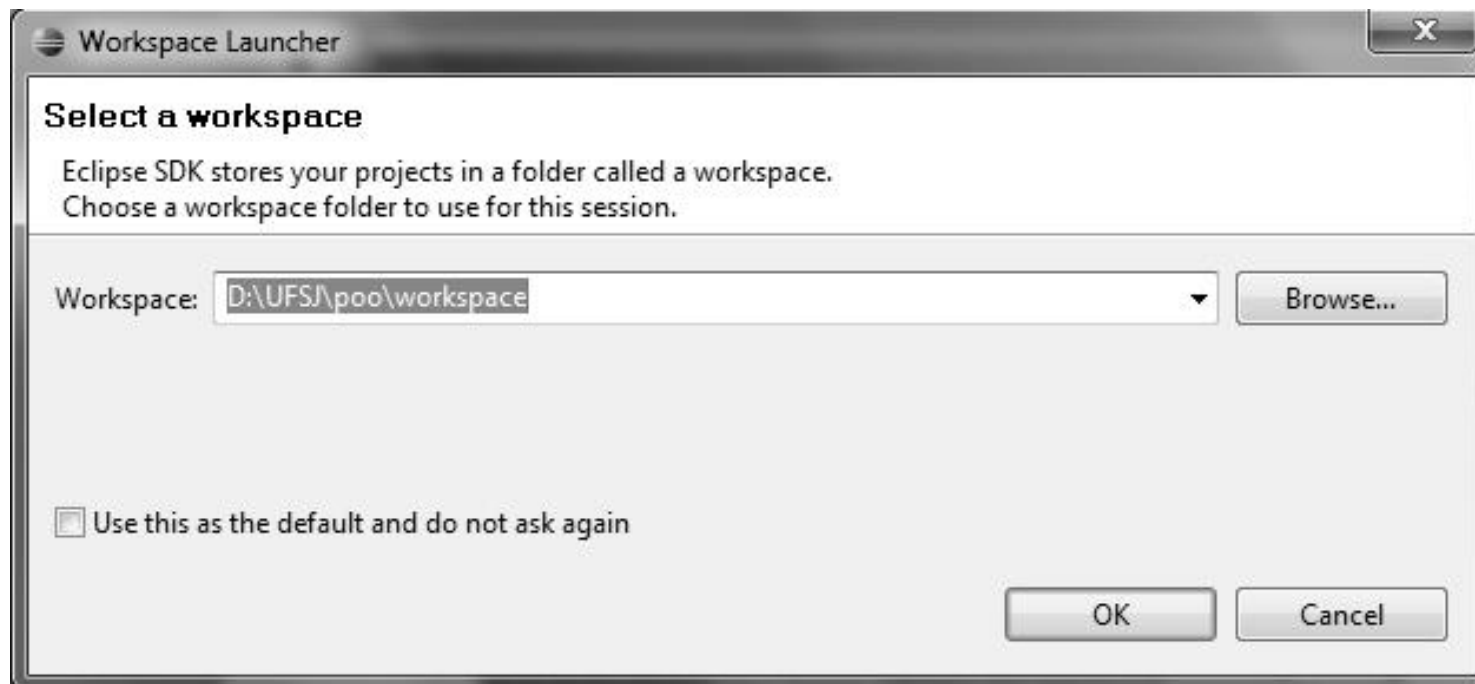
Eclipse

- Ambiente de desenvolvimento Java.
 - Usado em várias empresas.
 - Vários plugins podem ser integrados.



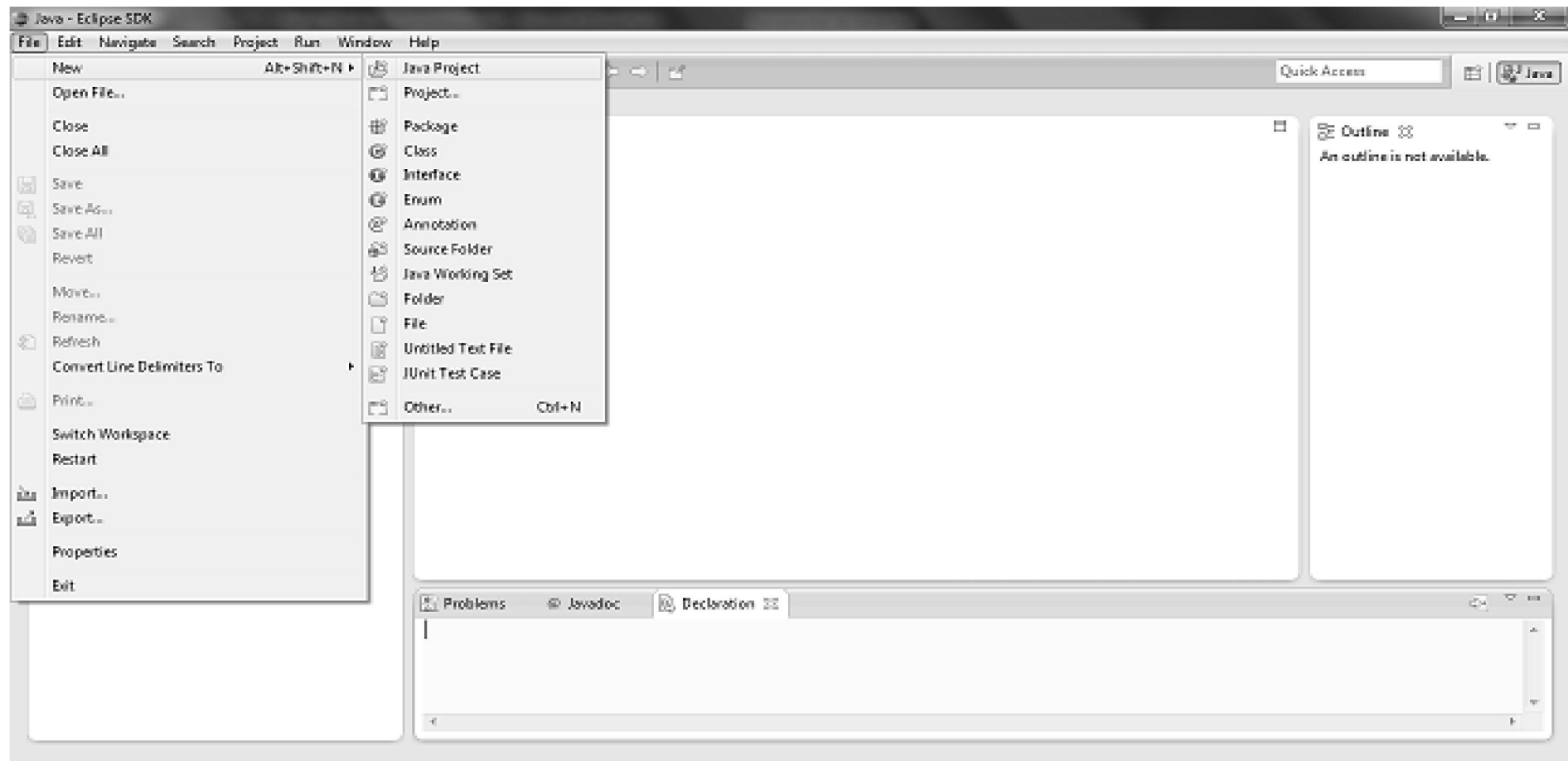
Eclipse

- Inicialmente, deve definir local de trabalho.
 - ☐ Diretório onde ficarão as classes e programas.
 - ☐ Pode ser trocado na opção File – Switch workspace.



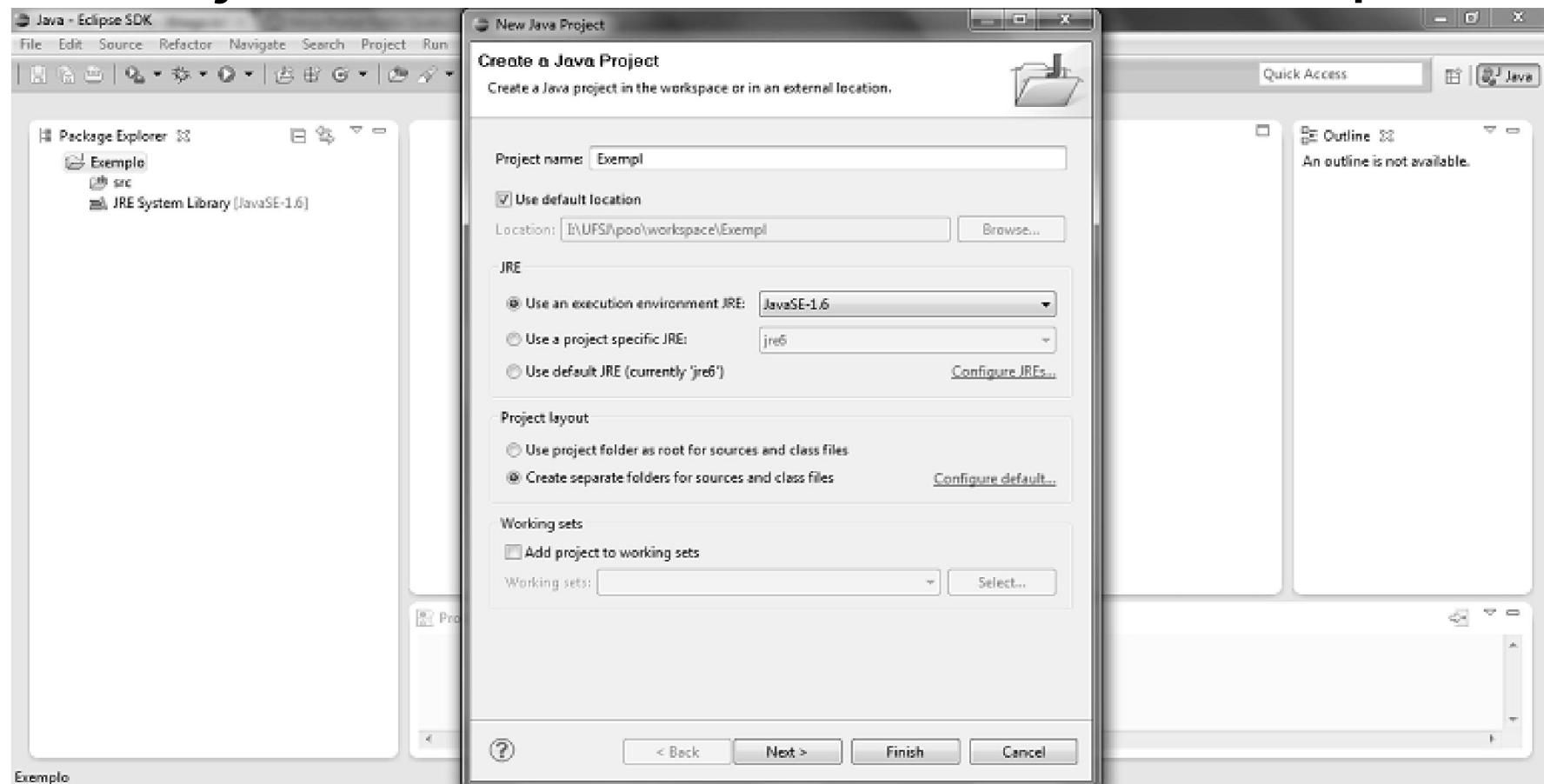
Eclipse

- Projeto armazena classes relacionadas.



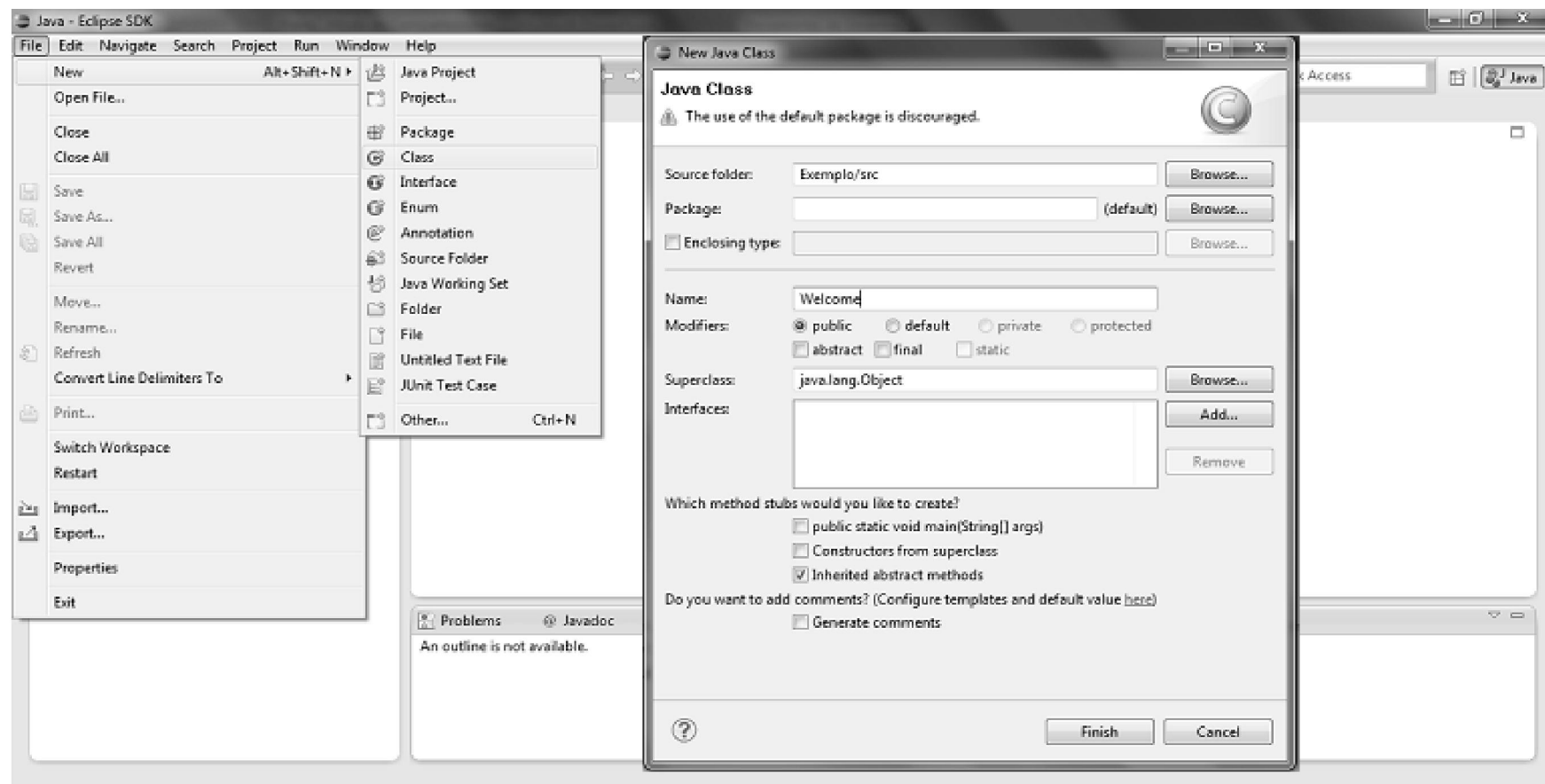
Eclipse

- Projeto criado é um diretório do workspace.



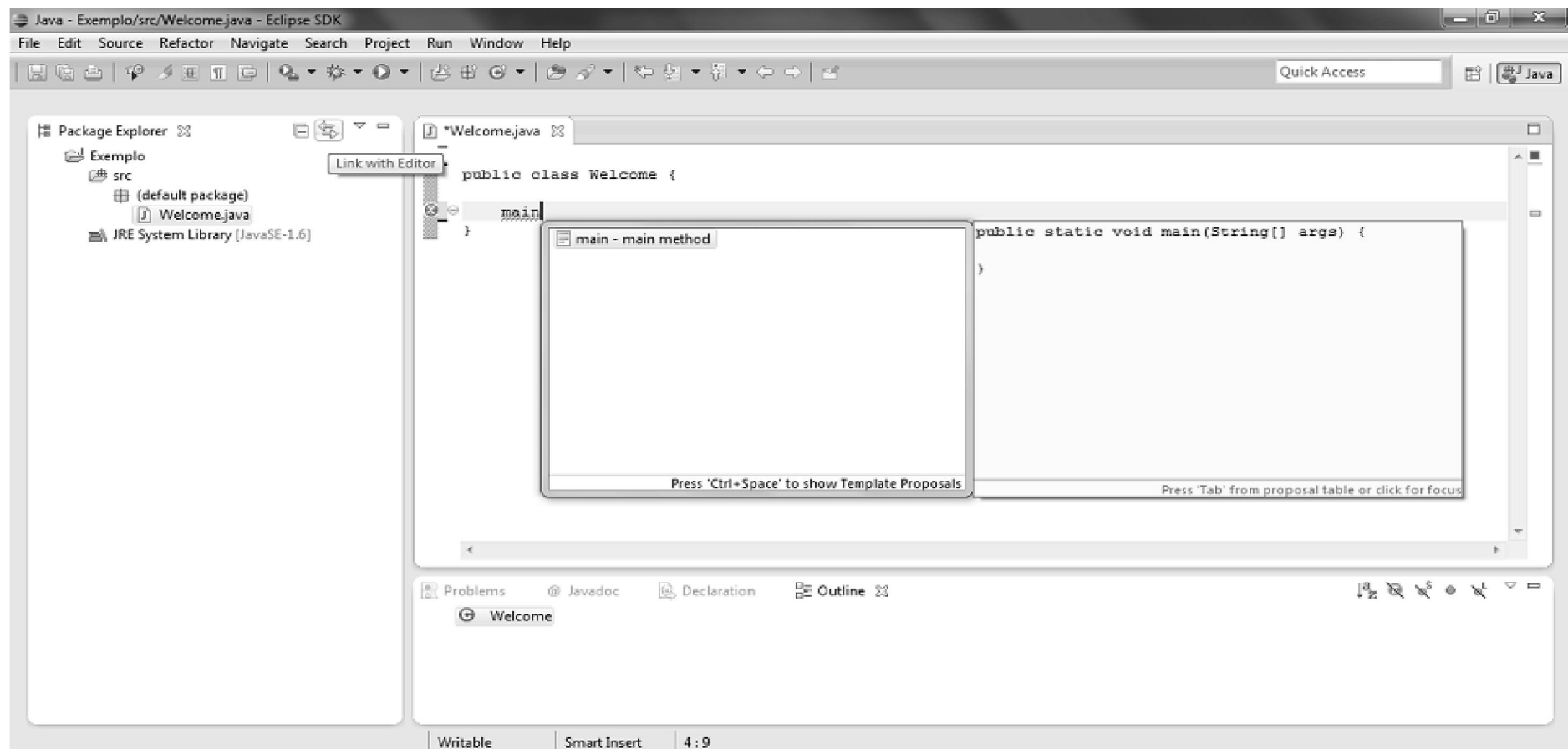
Eclipse

- Projeto pode agrupar várias classes.



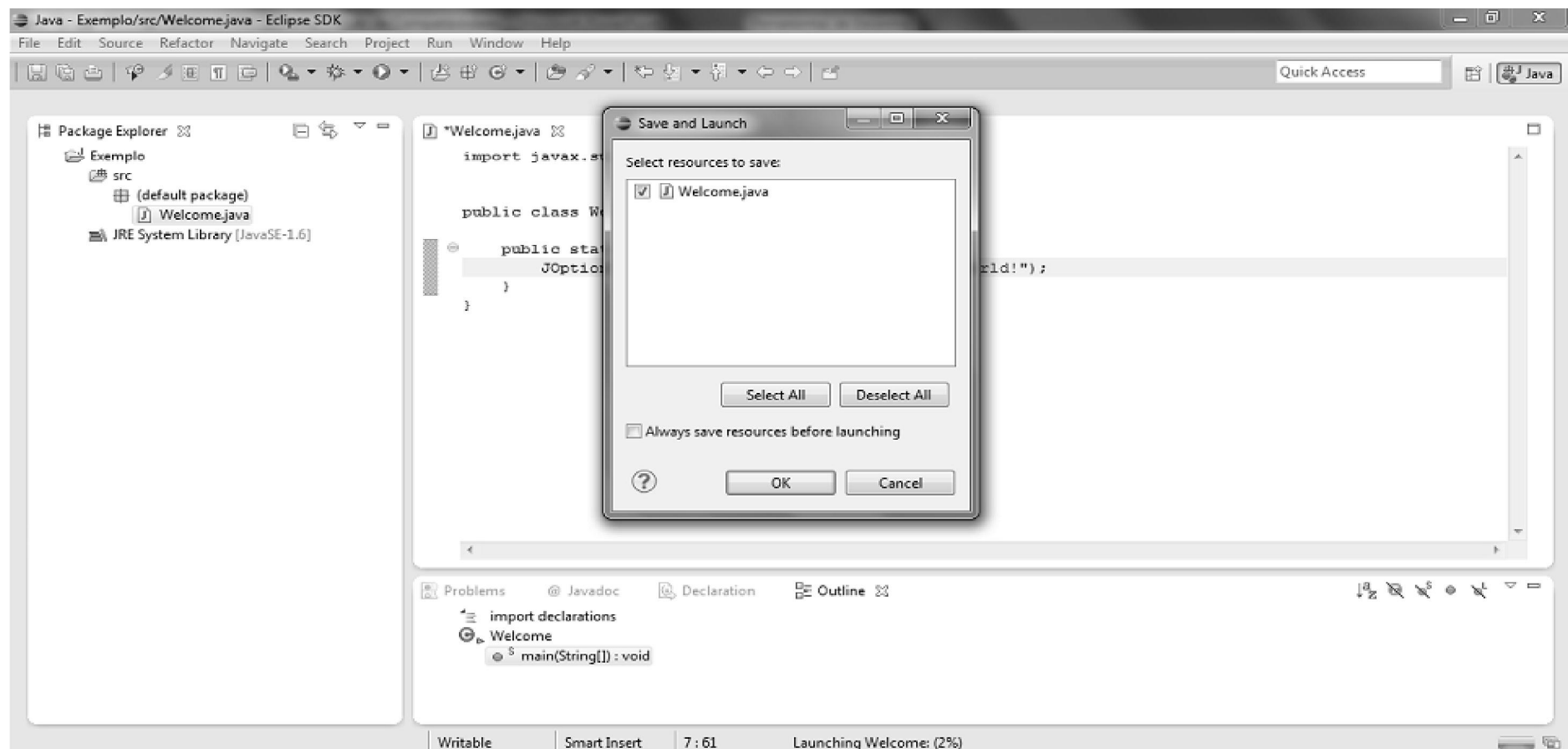
Eclipse

- CTRL+espaço completa e integra classes.



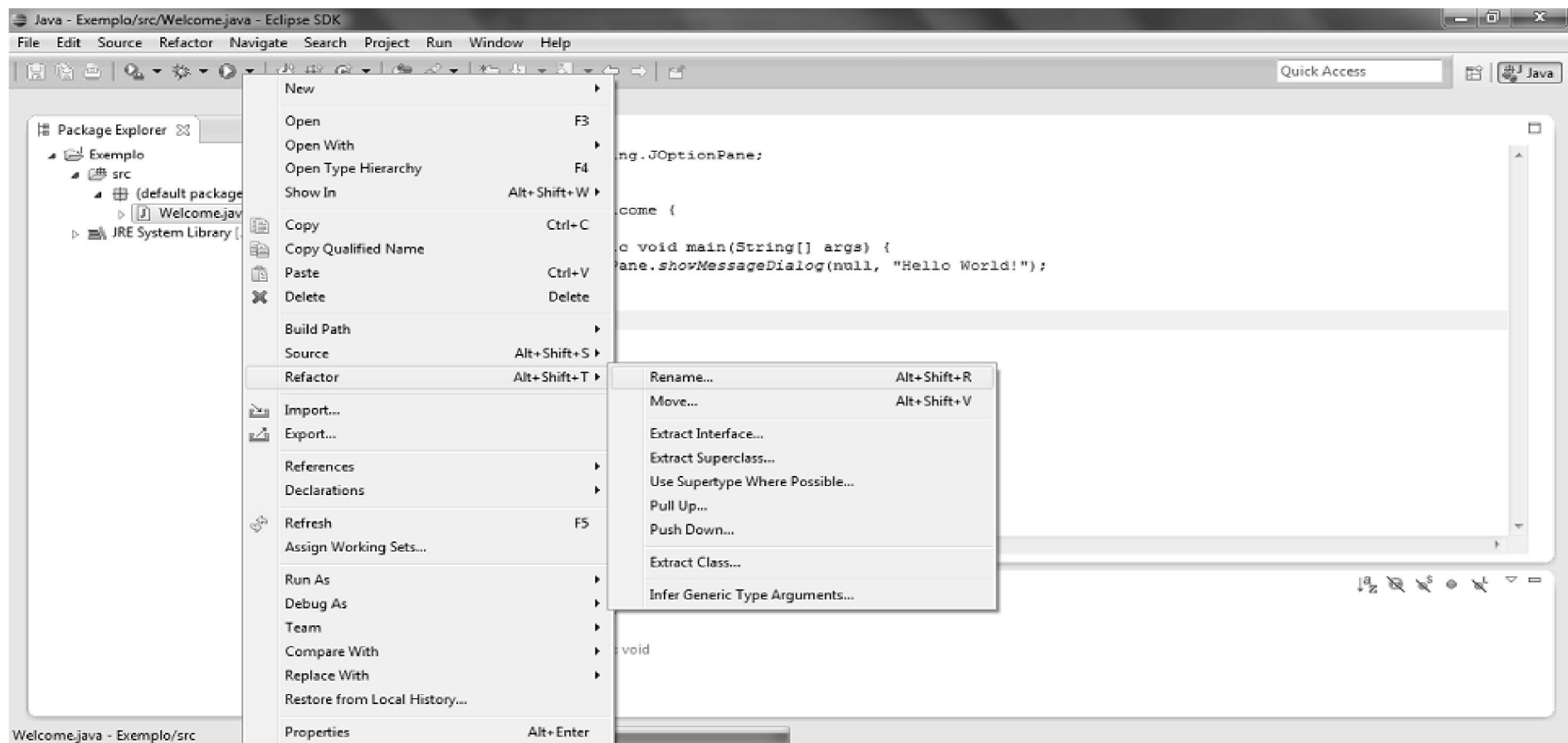
Eclipse

- Botão executar : permite salvar sempre.



Eclipse

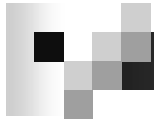
■ Package Explorer: manipular arquivos.





Tipos de dados primitivos

| Tipo | Bit | Exemplo |
|-------------|------------|-----------------------|
| boolean | 8 | true, false |
| char | 16 | 'a', 'A', '1', '@' |
| byte | 8 | -128 a 127 |
| int | 32 | -32768 a 32767 |
| long | 64 | -2^{63} a $+2^{63}$ |
| float | 32 | -3.14, 0 |
| double | 64 | -3.14, 0 |
| void | 0 | nenhum valor |



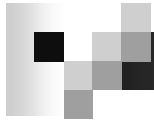
Operadores

- Atribuição =
- Aritméticos:
 - ☐ Soma +
 - ☐ Subtração -
 - ☐ Multiplicação *
 - ☐ Divisão /
 - ☐ Modulo %



Precedência

- $*$, $/$ e $\%$: mesma precedência, a maior.
- $+$ e $-$: mesma precedência.
- $=$: menor de todas.
- Em caso de empate resolve-se da esquerda para a direita.
- O uso de parênteses altera a precedência.



Comparações

■ Igualdade:

- ☐ igual a: ==
- ☐ diferente: !=

■ Relacionais:

- ☐ maior que: >
- ☐ menor que: <
- ☐ maior ou igual: >=
- ☐ menor ou igual: <=



Estruturas de controle

- Comando if – sintaxe:

```
if (condição) {  
    . . . sequência de comandos . . .  
}
```

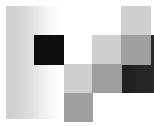
```
if (condição)  
    <comando> ;
```



Estruturas de controle

- Comando if/else – sintaxe:

```
if (condição) {  
    . . . sequência de comandos . . .  
} else  
{  
    . . . sequência de comandos . . .  
}
```



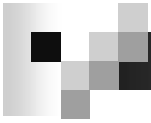
```
import javax.swing.JOptionPane;
public class Testelf{
    public static void main(String args[]){
        String entrada = JOptionPane.showInputDialog("Entre com
                                                    um número");
        int num = Integer.parseInt(entrada);
        if (num < 10) {
            JOptionPane.showMessageDialog(null,"Menor que 10 ");
        } else if (num < 100) {
            JOptionPane.showMessageDialog(null,"Menor que 100");
        }
        else JOptionPane.showMessageDialog(null,"Maior que 100");
        System.exit(0);
    }
}
```




Repetição controlada

- Comando while – sintaxe:

```
while (condição){  
    . . . sequência de comandos . . .  
}
```




```
import javax.swing.JOptionPane;
public class TesteWhile{
    public static void main(String args[]){
        String entrada = JOptionPane.showInputDialog(
            "Entre com um número");
        int num = Integer.parseInt(entrada);
        while (num <= 10){
            System.out.println("num = " + num);
            num ++;
        }
        System.exit(0);
    }
}
```



Estrutura de repetição “for”

■ Sintaxe:

```
for (variável ← inicial; condição; incremento) {  
    . . . sequência de comandos . . .  
}
```



```
public class TesteFor
{
    public static void main(String args[])
    {
        for(int i=1; i<=10; i++ ) {
            System.out.println("iteracao = " + i);
        }
        System.exit(0);
    }
}
```



Repetição controlada do/while


- Tem a mesma função do while.
- A diferença é que o bloco de comandos é executado pelo menos uma vez.

- Sintaxe:

```
do{
```

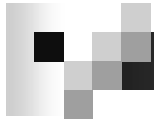
```
... sequência de comandos ...
```

```
} while (condição);
```



```
import javax.swing.JOptionPane;
public class TesteDoWhile{
    public static void main(String args[]){
        String entrada = JOptionPane.showInputDialog(
                                   "Entre com um número");
        int num = Integer.parseInt(entrada);
        do{
            JOptionPane.showMessageDialog(null,"Número é"
                                   + num);

            num = num/2;
        } while (num > 10);
        }
}
```



Estrutura de controle switch


- Utilizado quando uma variável pode assumir vários valores.
- Este comando só pode ser utilizado com variáveis dos tipos:
 - ☐ byte
 - ☐ short
 - ☐ int
 - ☐ char



Estrutura de controle switch

■ Sintaxe

```
switch (variável)
{
    case [valor]:
        . . . sequência de comandos . . .
        break;
    case [valor]:
        . . . sequência de comandos . . .
        break;
    default:
        . . . sequência de comandos . . .
}
```

```
import javax.swing.JOptionPane;
public class TesteSwitch{
    public static void main(String args[]){
        String entrada = JOptionPane.showInputDialog("Entre com 1 número");
        int num = Integer.parseInt(entrada);
        switch (num) {
            case 1:
                JOptionPane.showMessageDialog(null,"1");
                break;
            case 2:
                JOptionPane.showMessageDialog(null,"2");
                break;
            default:
                JOptionPane.showMessageDialog(null,"maior ou igual a 3");
        }
        System.exit(0);
    }
}
```



Operadores Lógicos

■ Operador “E” (and)

| A | B | A && B |
|-------|-------|--------|
| false | false | false |
| false | true | false |
| true | false | false |
| true | true | true |

■ Operador “OU” (or)

| A | B | A B |
|-------|-------|--------|
| false | false | false |
| false | true | true |
| true | false | true |
| true | true | true |



Operadores Lógicos

■ Operador “OU exclusivo” (xor)

| A | B | $A \wedge B$ |
|-------|-------|--------------|
| false | false | false |
| false | true | true |
| true | false | true |
| true | true | false |

■ Operador de negação (not)

| A | $\neg A$ |
|-------|----------|
| false | true |
| true | false |