

# Aula 2

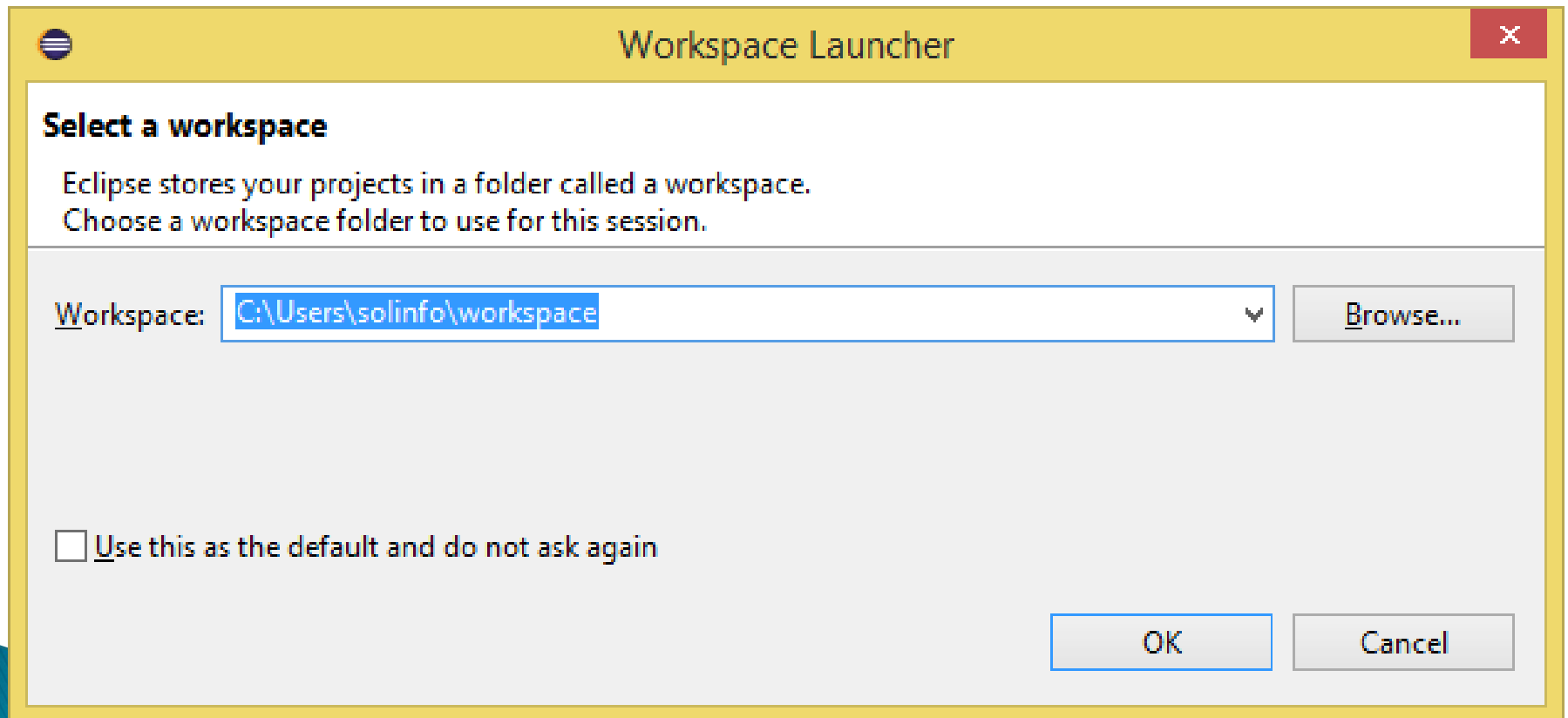
# IDE

- ▶ Eclipse



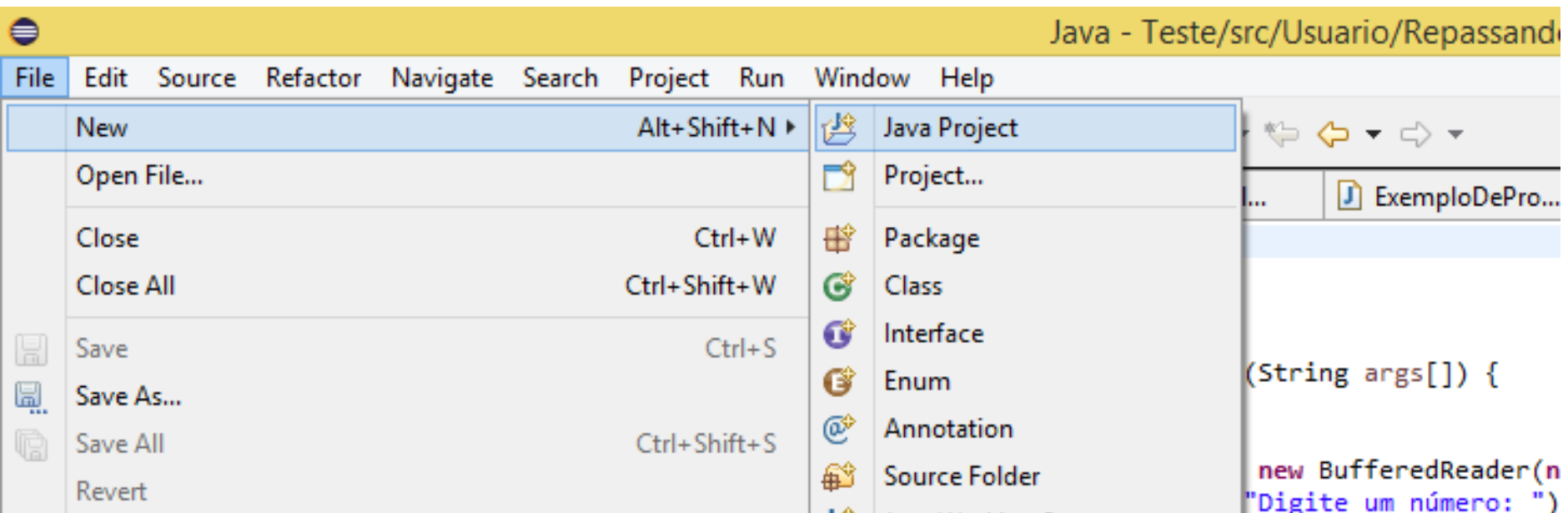
# IDE

- ▶ Selecione onde seus projetos serão armazenados



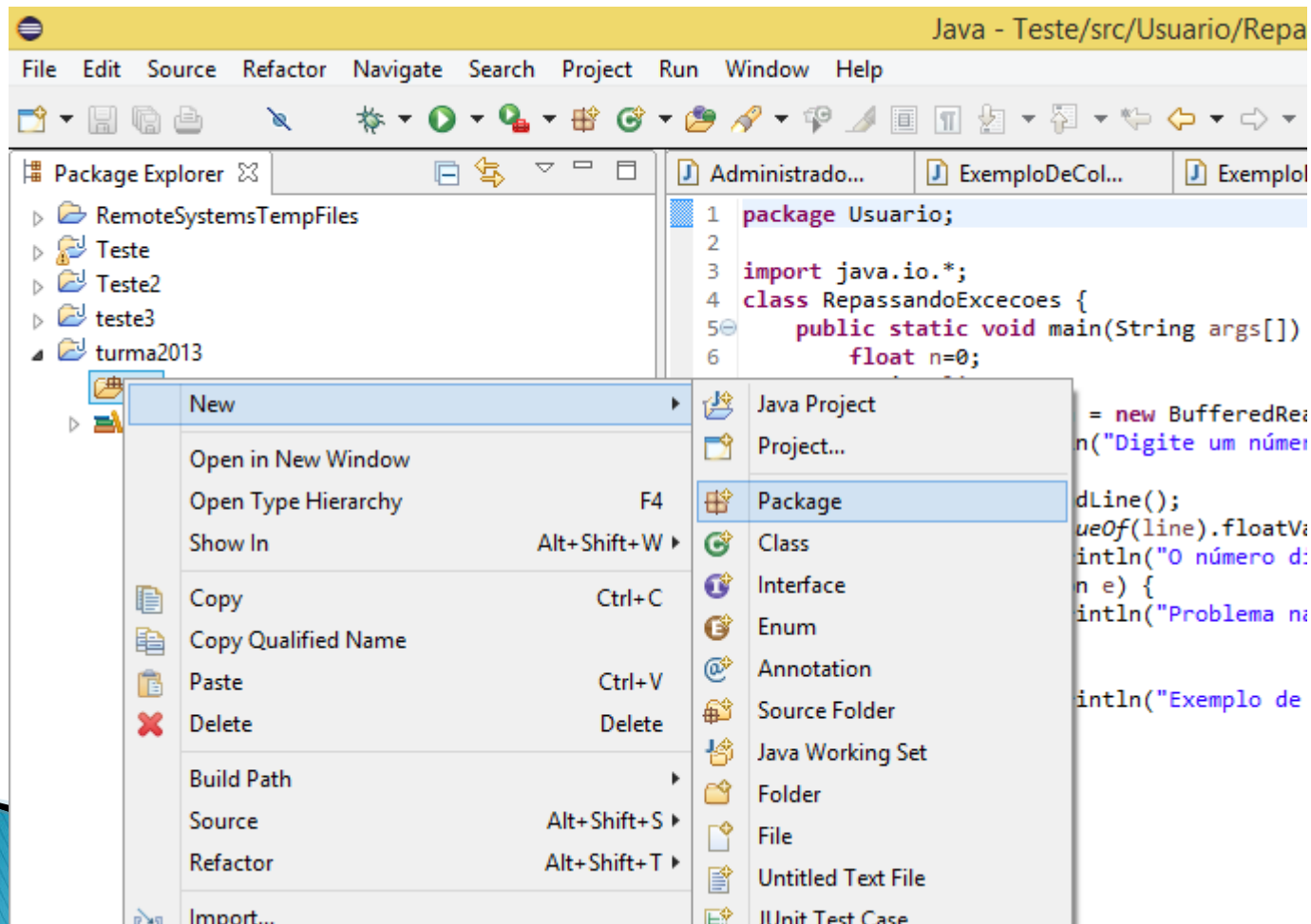
# IDE

## ► Criar um projeto Java



# IDE

## ► Criar um pacote (diretório)



# IDE

## ► Criando uma classe

The image shows an IDE interface with a 'New Java Class' dialog box open. The dialog is titled 'New Java Class' and has a subtitle 'Create a new Java class.' It contains several fields and options for creating a new class.

**Java Class Dialog Fields:**

- Source folder:** turma2013/src (with a 'Browse...' button)
- Package:** modelo (with a 'Browse...' button)
- Enclosing type:** (empty, with a 'Browse...' button)
- Name:** BemVindo
- Modifiers:** ☒ public, ☐ package, ☐ private, ☐ protected, ☐ abstract, ☐ final, ☐ static
- Superclass:** java.lang.Object (with a 'Browse...' button)
- Interfaces:** (empty, with 'Add...' and 'Remove' buttons)
- Which method stubs would you like to create?**
  - ☒ public static void main(String[] args)
  - ☐ Constructors from superclass
  - ☐ Inherited abstract methods
- Do you want to add comments? (Configure templates and default value [here](#))**
  - ☐ Generate comments

At the bottom of the dialog are 'Finish' and 'Cancel' buttons.

The IDE background shows the 'Package Explorer' on the left with a tree view containing 'RemoteSystemsTempFiles', 'Teste', 'Teste2', 'teste3', 'turma2013', 'src', and 'modelo'. The 'src' folder is expanded, and the 'modelo' package is selected. A context menu is open over the 'modelo' package, showing options like 'New', 'Open in New Window', 'Open Type Hierarchy', 'Show In', 'Copy', 'Copy Qualified Name', 'Paste', and 'Delete'. The 'New' option is selected, and a sub-menu is visible with options like 'Java Project', 'Project...', 'Package', 'Class', 'Interface', 'Enum', 'Annotation', 'Source Folder', and 'Java Working Set'. The 'Class' option is highlighted.

The editor window on the right shows a snippet of Java code:

```
1 package Usuari
2
3 import java.io
4 class Repassan
5 public sta
6 float
7 String
8 Buffer
```

# A LINGUAGEM JAVA

## ► Comentários

- `/* texto */`

O compilador ignora tudo entre `/*` e `*/`

- `/** documentacao */`

indica um comentário para documentação.  
Utilizado pela ferramenta *javadoc*

- `// texto`

O compilador ignora todos os caracteres de `//`  
até o final da linha

# A LINGUAGEM JAVA

## ▶ Tipos de dados

- Inteiros: byte / short / int / long
- Reais: float / double
- Outros: char / boolean

## ▶ Nomes de variáveis

- Série de caracteres
- Não pode ser palavra chave



# A LINGUAGEM JAVA

- ▶ Métodos e variáveis estáticos
  - Declarados com o especificador **static**
  - São comuns a todos os objetos da classe
  - Utilizados para declaração de métodos que não necessitam de uma instância da classe

```
static int FALSO      0;  
static int VERDADEIRO 1;
```

# A LINGUAGEM JAVA

- ▶ Inicialização de variáveis

```
int i = 10;
```

```
char c;
```

```
c = 'X' ;
```

- ▶ Variáveis finais (constantes)

```
final float pi = 3.14159;
```

# Exercício

- ▶ Implemente uma classe com:
  - Duas variáveis inteiras
  - Uma constante de valor 5
  - Uma variável real que começa com o valor 10
  - Faça um comentário acima de cada variável

# Orientação a Objetos em Java

```
class Bicicleta {  
    int velocidade = 0;  
    int marcha = 1;  
  
    public void mudarMarcha(int novoValor) {  
        marcha = novoValor;  
    }  
  
    void aumentarVelocidade(int incremento) {  
        velocidade = velocidade + incremento;  
    }  
  
    void aplicarFreios(int decremento) {  
        velocidade = velocidade - decremento;  
    }  
  
    public int mostrarMarcha(){  
        return marcha;  
    }  
}
```

# Orientação a Objetos – Criando objetos com Java

- Para instanciarmos um novo objeto devemos utilizar o operador *new*, conforme modelo abaixo:
- Criando dois objetos bicicleta:  
Bicicleta **bicicleta1** = new Bicicleta();  
Bicicleta **bicicleta2** = new Bicicleta();
- Invocando seus métodos:  
**bicicleta1**.mudarMarcha(2);  
**bicicleta2**.aumentarVelocidade(5);  
  
System.out.println(bicicleta1.mostrarMarcha());  
System.out.println(bicicleta2.velocidade);

# Exercício

- ▶ Para o exercício anterior imprima dentro do método main os valores das variáveis e constante
- ▶ `System.out.println(variavel);`

# OBJETOS E CLASSES EM JAVA

## ▶ Exemplo: Lâmpada

▶ `class Lampada {`

▶ `private boolean estadoLampada;`

▶ `public void acionarInterruptor(){`

▶ `estadoLampada = !estadoLampada;`

▶ `}`

▶ `public Lampada(boolean estado) {`

▶ `estadoLampada = estado;`

▶ `}`

▶ `}`

Variável

Método

Construtor

# OBJETOS E CLASSES EM JAVA

- ▶ Criando um objeto da classe Lamp

```
Lampada l;
```

```
l = new Lampada(true);
```

```
Lampada l1 = new Lampada (false);
```

```
Lampada l2 = new Lampada (true);
```

- ▶ Acessando variáveis e métodos

```
l1.acionarInterruptor();
```

```
l2.estadoLampada = true;
```

Viola definição de  
visibilidade



# OBJETOS E CLASSES EM JAVA

## ▶ Garbage Collector

- A plataforma Java periodicamente libera a memória usada por objetos que não são mais necessários
- O Garbage Collector roda em baixa prioridade e remove todos os objetos que não são mais referenciados

## ▶ Finalização

- Antes de um objeto ser destruído, o Garbage Collector executa o método *finalize* do objeto (quando existir)
- Isto permite, por exemplo, o fechamento de arquivos e conexões de rede

```
public finalize() {  
    f.close();  
}
```

# OBJETOS E CLASSES EM JAVA

## ► super

- Faz referência à superclasse
- Utilizado para chamar o construtor da superclasse, quando se tem herança

```
public LampadaFluorescente( ) {  
    super(true);  
}
```

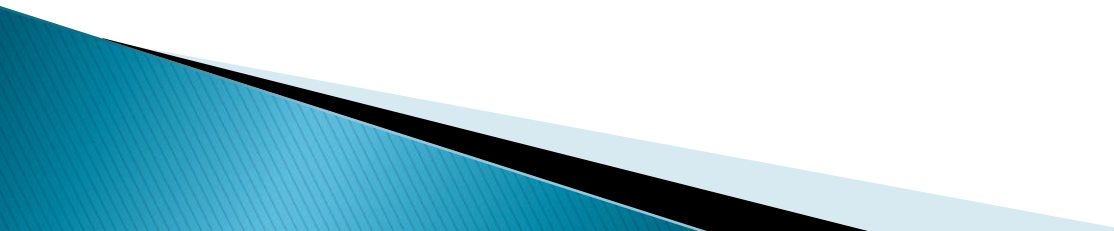
- Utilizado para chamar métodos da superclasse

```
public finalize() {  
    f.close();  
    super().finalize();  
}
```

# Operadores

# Operadores

Em Java, trabalha-se com conjuntos de operadores:

- aritméticos
  - relacionais
  - condicionais
  - lógicos e binários
  - atribuição
  - outros
- 

# Operadores Aritméticos

+      adição                       $op1 + op2$

-      subtração                    $op1 - op2$

/      divisão                       $op1 / op2$

%      resto da divisão            $op1 \% op2$

++    incremento de 1              ++op1    ou    op1++

--    decremento de 1              --op1    ou    op1--

# Operadores Relacionais

== igual                      op1 == op2

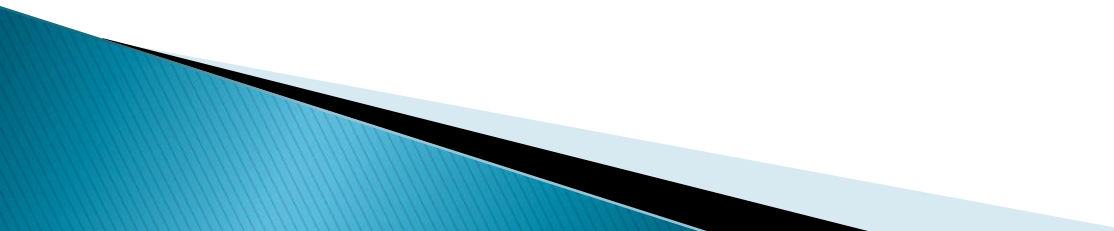
!= diferente                  op1 != op2

> maior                      op1 > op2

< menor                      op1 < op2

>= maior ou igual          op1 >= op2

<= menor ou igual          op1 <= op2



# Operadores Condicionais

**&&    “E”**

|| “OU” || op1 || op2

! “NÃO” lógico ! Op

$$\wedge \quad \text{“OU” exclusivo} \quad \text{op1} \wedge \text{op2}$$

# Operadores de Atribuição Aritméticos

$+=$  atribuição aditiva  
equivale a

$op1 += op2$   
 $op1 = op1 + op2$

$-=$  atribuição subtrativa  
equivale a

$op1 -= op2$   
 $op1 = op1 - op2$

$*=$  atribuição multiplicativa  
equivale a

$op1 *= op2$   
 $op1 = op1 * op2$

$/=$  atribuição divisiva  
equivale a

$op1 /= op2$   
 $op1 = op1 / op2$

$\%=$  atribuição de resto  
equivale a

$op1 \% = op2$   
 $op1 = op1 \% op2$



# Expressões

- ▶ Série de variáveis, operadores e chamadas a métodos, construídas de acordo com a sintaxe, que resulta em um valor simples

Ex:

$$x * (y + z) / 100$$
$$s = \text{"Meu nome é "} + \text{nome}$$

# Exercício

- ▶ Implemente uma classe com as seguintes expressões:
  - 1º –
    - $x = (x+5)*10-50$ ;
    - Com x inicializado com 10
  - 2º –
    - $Z = \text{"Anderson"}$
    - $W = \text{"Costa"}$
    - $Y = Z + \text{" "} + W$
- ▶ Implemente duas classes.
  - Cada classe deve possuir um método com uma expressão.
  - A primeira classe deve instanciar um objeto da segunda e utilizar o resultado do método da segunda classe como parte de seu método.

# Laço: while

- ▶ Repete processamento do bloco, enquanto a expressão for verdadeira. A expressão é avaliada antes do processamento do bloco:

```
while (expressão booleana) {  
    bloco(s)  
}
```

# Laço: while

▶ Ex:

```
x=0;  
while (x<5) {  
    x++;  
}
```

# Laço: while

▶ Ex:

//aumento em meses até que chegue em 5000

**float** salario = 1000;

**while** (salario < 5000) {

    salario \*= 1.5;

    System.out.println("Meu salário AINDA é de " + salario);

}



# Laço: do-while

- ▶ Repete processamento do bloco enquanto a expressão for verdadeira
- ▶ A expressão é avaliada após o processamento do bloco:

```
do {  
    bloco(s)  
} while (expressão booleana);
```

# Laço: do-while

```
//aumento menor que 5000
```

```
float salario = 1000;
```

```
float aumento = 100;
```

```
do {
```

```
    salario += aumento;
```

```
    System.out.println(" O valor atual do salario é de: " + salario);
```

```
} while (salario < 5000);
```



# Exercício

- ▶ Implemente um exemplo de while e modifique-o para do-while



# Laço: for

- ▶ Laço de interação que inclui:
  - expressão de inicialização
  - condição de término
  - expressão de incremento:

```
for (inicialização; término; incremento) {  
    bloco(s)  
}
```

# Laço: for

```
for(int i=0; i<5; i++){  
    System.out.println(i);  
}
```

# Decisão: if-then

- ▶ Executa o bloco caso a expressão seja verdadeira:

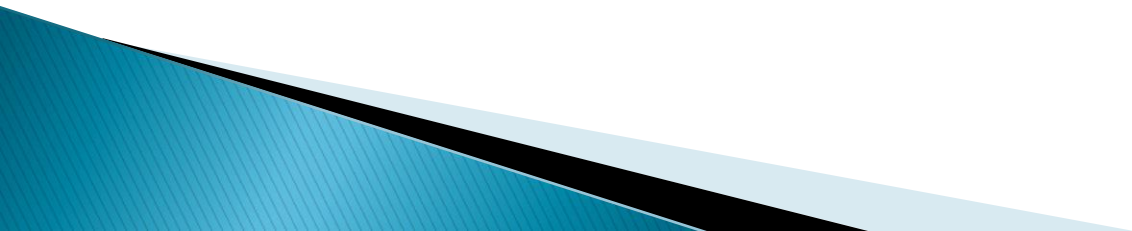
```
if (expressão booleana) {  
    bloco(s)  
}
```

# Decisão: if-then-else

- ▶ Executa o primeiro bloco caso a expressão seja verdadeira, do contrário executa o segundo bloco:

```
if (expressão booleana) {  
    bloco(s)  
} else {  
    bloco(s)  
}
```

# Exercício

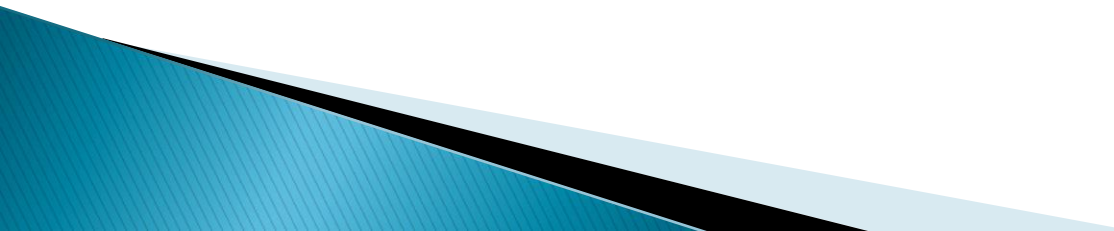
- ▶ Faça uma classe com métodos para as seguintes situações, considerando funcionalidades de um exemplo de sistema real:
    - For
    - IF-Then
    - IF-Then-Else
- 

# Decisão: Múltipla Escolha

- ▶ Avalia uma expressão inteira e, de acordo com o valor, processa o bloco correspondente.
- ▶ Caso o valor não coincida com nenhum dos valores previstos, executa o bloco default

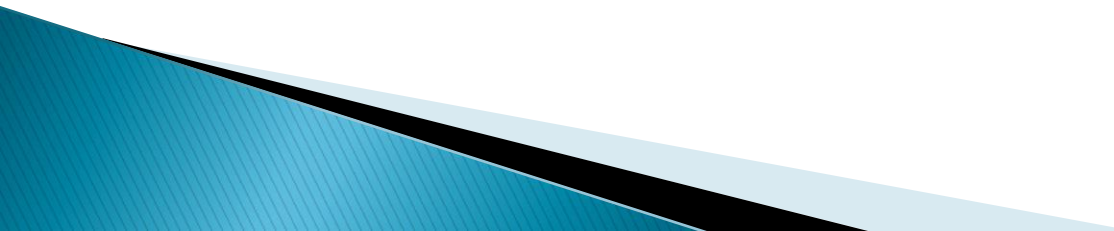
# Decisão: Múltipla Escolha

```
switch (expressão inteira) {  
    case escolha1:  
        bloco(s)  
        break;  
    case escolha2:  
        bloco(s)  
        break;  
    ...  
    default:  
        bloco(s)  
        break;  
}
```



# Decisão: Múltipla Escolha

```
int fimDeSemana = 1;  
switch (fimDeSemana) {  
case 1:  
    System.out.println("Domingo");  
    break;  
case 2:  
    System.out.println("Sábado");  
    break;  
default:  
    System.out.println("Este não é um dia válido!");  
}
```

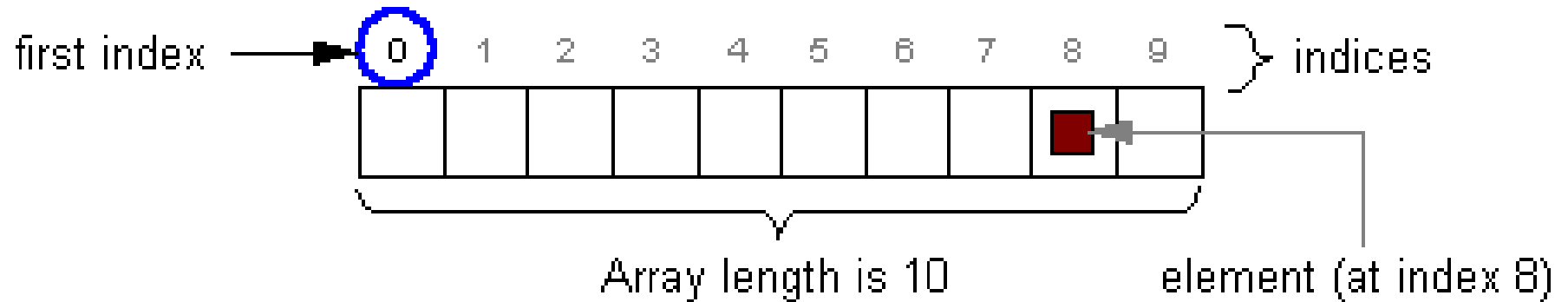




# Arrays

- ▶ Def.: São estruturas que armazenam múltiplos valores do mesmo tipo
  - O tamanho é estabelecido no momento de criação
  - Após a criação o tamanho do array é fixo

# Estrutura de um Array



# Utilizando Arrays

- ▶ Declaração de um array:

`type[ ] name;`

Ex:

`int[ ] x; // Um array de inteiros`

- ▶ Criação de um array:

Ex:

`int[] x = new int[20];`

# Utilizando Arrays

- ▶ Obtendo o tamanho de um array:

`x.length();`

- ▶ Acessando um elemento:

`x[posição]`

Ex:

```
for (int i=0; i<x.length; i++) {  
    System.out.println(x[i]);  
}
```

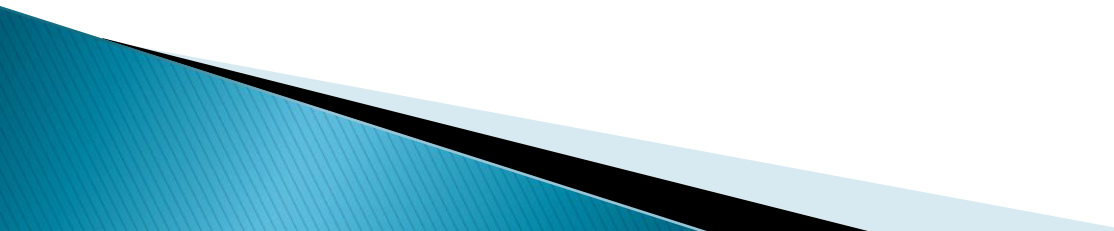
# Utilizando Arrays

- ▶ Inicializando um array:

```
type[] name = { elemento1, elemento2, ..., elementoN};
```

Ex:

```
int[] x = { 2, 4, 8, 16, 32, 64 };
```

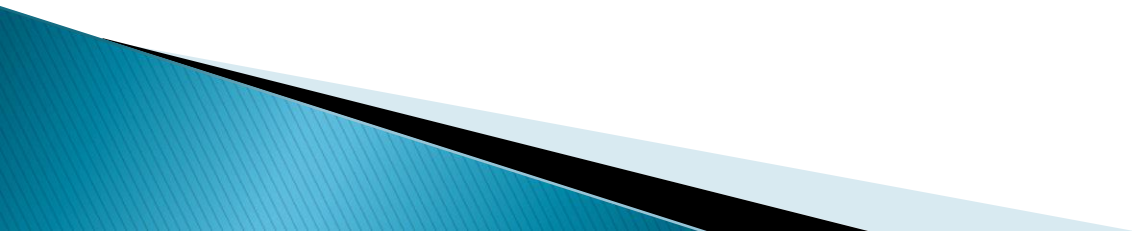


# Array de Objetos

- ▶ É possível criar arrays tanto para tipos primitivos como para objetos:

```
String[ ] semana = {"Domingo", "Segunda", ..., "Sábado"};  
Lampada lamp1 = new Lampada(true);  
Lampada lamp2 = new Lampada(false);  
Lampada[ ] lampadas = {lamp1, lamp2};
```

# Exercício

- ▶ Crie uma classe que possui um array e atribua na quinta posição o valor 10
  - ▶ Crie uma estrutura de múltipla escolha para identificar que para a escolha 1 a questão está correta e para as demais opções a escolha é errada.
- 

# Exercício

- ▶ Crie uma classe aluno com seus atributos
  - ▶ Na classe aluno crie um método para inserir nota e outro para visualizar a nota
  - ▶ Inicialize a nota do aluno com zero
  - ▶ Crie uma classe a parte com o método main
    - Instancie um aluno com a nota
    - Atribua a nota 8 ao aluno e imprima a nota
- 