

## Lab 5 – Usando Objetos em Java

Neste laboratório faremos uso de objetos Java, objetos de bibliotecas básicas, reforçando experiência de programação em Java e outros conceitos básicos como invocação de métodos (de classe e de instância), classes básicas do pacote **java.lang**.

**Duração prevista: 160 minutos**

### Exercícios

**Exercício 1:** Criando uma instância de objeto de uma classe usando a palavra chave **new** (20 minutos)

**Exercício 2:** Métodos estáticos e métodos de instância (não - estático) (20 minutos)

**Exercício 3:** Argumentos passado por valor e por referência (20 minutos)

**Exercício 4:** Escopo de variáveis (20 minutos)

**Exercício 5:** Coesão de tipos primitivos e classe **Wrapper** (30 minutos)

**Exercício 6:** Comparando objetos (30 minutos)

**Exercício 7:** Método **getClass()** e o operador **instanceof** (20 minutos)

### *Exercício 1: Criando uma instância de objeto de uma classe usando a palavra chave "new" (20 minutos)*

#### 1.1 Criando objetos da classe String

1. Usando sua IDE ou editor de arquivo crie um arquivo **ClasseEObjeto.java** com base na **Listagem-2.1** abaixo. Estude o código prestando muita atenção aos comentários.

```
public class ClasseEObjeto {  
  
    public static void main(String[] args) {  
  
        // Para criar uma instância de uma classe use a palavra chave new  
        // Por exemplo, para criar uma instância da classe String  
        // procedemos como segue  
        String strObjeto1 = new String("Este objeto é uma instância da classe String");  
        System.out.println("estado da instância de strObjeto1 = " + strObjeto1);  
  
        // A classe String é uma classe especial que permite a criação de uma  
        // instância pela atribuição de um literal string. Nenhuma outra classe  
        // em java permite este tipo de criação. Além disso para cada literal  
        // string é criada uma única instância desta string  
        String strObjeto2 = "Este objeto é uma instância da classe String";  
        System.out.println("estado da instância de strObjeto2 = " + strObjeto2);  
  
    }  
  
}
```

Listagem 2.1 – ClasseEObjeto.java

2. Compile e execute o programa **ClasseEObjeto.java**, observe o resultado da execução:

*estado da instância de strObjeto1 = Este objeto é uma instância da classe String*

*estado da instância de strObjeto2 = Este objeto é uma instância da classe String*

3. Modifique o programa **ClasseEObjeto.java** para criar outra instância da classe **String** contendo o literal string **"Eu sou outra instância da classe String"** e imprimir seu estado usando o método **System.out.println(...)**

## 1.2 Criando objetos da classe Integer

1. Com base no programa **ClasseEObjeto.java**, modifique o programa para criar um objeto da classe **Integer** (classe **Wrapper** do tipo primitivo **int**) cujo valor é 20, veja na **Listagem-2.2**.

```
public class ClasseEObjeto {  
    public static void main(String[] args) {  
        // Para criar uma instância de uma classe use a palavra chave new  
        // Por exemplo, para criar uma instância da classe String  
        // procedemos como segue  
        String strObjeto1 = new String("Este objeto é uma instância da classe String");  
        System.out.println("estado da instância de strObjeto1 = " + strObjeto1);  
  
        // A classe String é uma classe especial que permite a criação de uma  
        // instância pela atribuição de um literal string. Nenhuma outra classe  
        // em java permite este tipo de criação. Além disso para cada literal  
        // string é criada uma única instância desta string  
        String strObjeto2 = "Este objeto é uma instância da classe String";  
        System.out.println("estado da instância de strObjeto2 = " + strObjeto2);  
  
        // cria uma instância de objeto da classe Integer  
        Integer intObjeto1 = new Integer(20);  
        System.out.println("estado da instância de intObjeto1 = " + intObjeto1);  
    }  
}
```

Listagem 2.2 – ClasseEObjeto.java criando objeto Integer

2. Modifique o programa anterior para criar e imprimir uma instância da classe **Double** e imprimir seu estado.

## Exercício 2: Métodos estáticos e métodos de instância (não - estático)

Neste exercício, você aprenderá como invocar métodos **estáticos** (de classe) e **não-estáticos** (de instância) de uma classe. Para invocar métodos estáticos usamos a seguinte forma:

**<Nome da Classe>.<Nome do método estático>**

por exemplo,

*Integer.parseInt("25"); // parseInt é um método estático da classe Integer*

Um método não-estático (instância) de uma classe só pode ser invocado a partir de uma instância do objeto da classe usando a seguinte forma:

*<Nome da instância de uma classe>.<Nome do método não-estático>*

por exemplo,

*String str = new String("Brasil com z"); //cria instância do objeto*

*str.charAt(0); //chama método de instância charAt()*

*//da classe String através da instância do objeto*

1. Crie o programa **MetodosInstanciaEEstatico.java** com base na **Listagem-2.3** a abaixo.

```
public class MetodosInstanciaEEstatico {  
  
    public static void main(String[] args) {  
  
        // Cria duas instâncias da classe String  
        String strInst1 = new String("Sou uma instância de objeto da classe String");  
        String strInst2 = "Viva com paixão!";  
  
        // Invoca o método de instância charAt()  
        // através das instâncias da classe String  
        char x = strInst1.charAt(2);  
        char y = strInst2.charAt(1);  
        char z = strInst2.charAt(0);  
  
        System.out.println("O 3.o caracter da strInst1 = " + x);  
        System.out.println("O 2.o caracter da strInst2 = " + y);  
        System.out.println("O 1.o caracter da strInst2 = " + z);  
  
        // Invoca o método de instância equalsIgnoreCase(...)  
        boolean b = strInst1.equalsIgnoreCase(strInst2);  
        String strInst3 = b ? "Sim" : "Não";  
  
        System.out.println("As variáveis strInst1 " + " e strInst2 tem o mesmo " + "  
conjunto de caracteres? " + strInst3);  
        // Invoca um método estático, valueOf (int i), da classe String  
        int i = 23;  
        String strInst4 = String.valueOf(i);  
        System.out.println("valor de strInst4 = " + strInst4);  
  
        // Você já usou o método estático parseInt() da classe  
        // Integer no exercício anterior  
        // Agora cria uma instância de objeto da classe String  
        String strInst5 = new String("34");  
        int ii = Integer.parseInt(strInst5);  
        System.out.println("valor de ii = " + ii);  
  
    }  
}
```

Listagem 2.3 – MetodoInstanciaEEstatico.java

2. Compile, rode o programa e observe o resultado.

3. Modifique o programa **MetodosInstanciaEEstatico.java** para que o mesmo contenha as instruções abaixo no final do método **main()**.

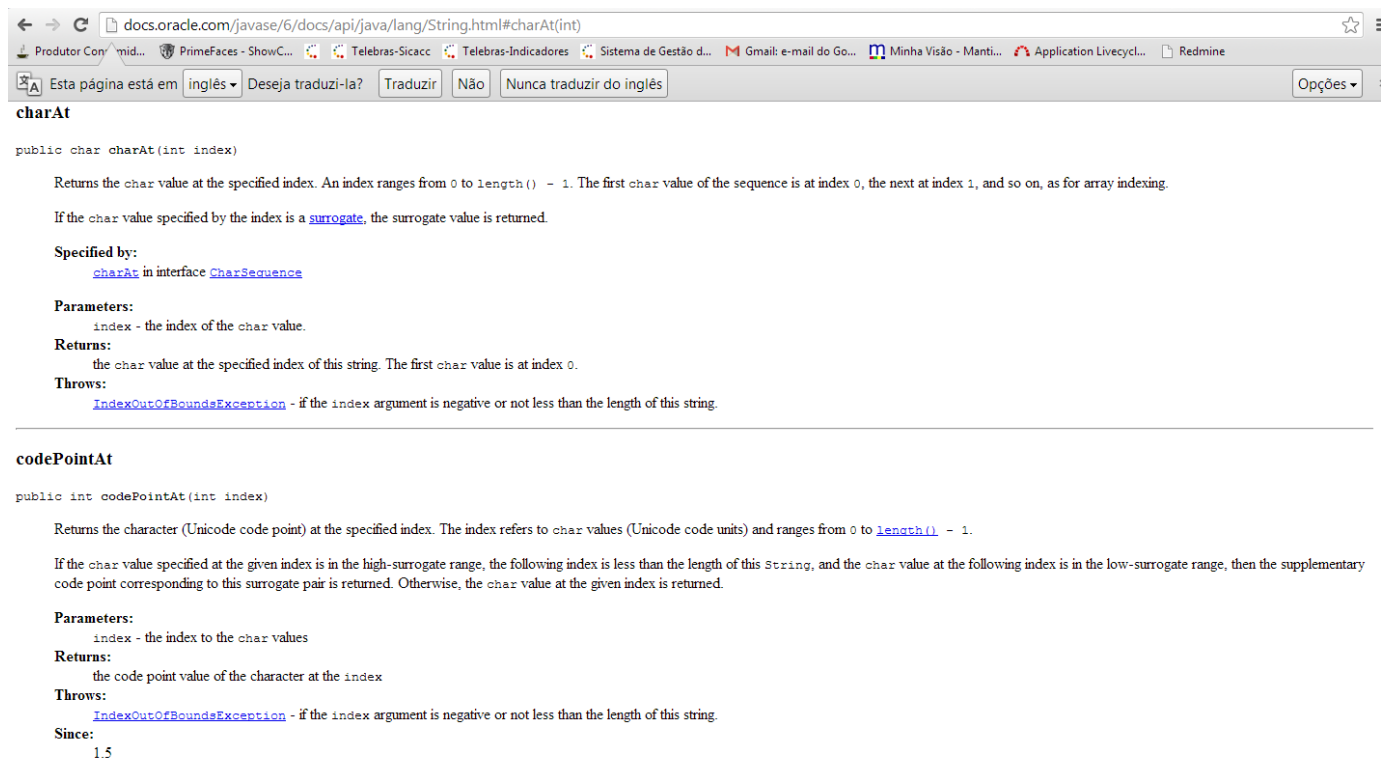
```
// O seguinte código irá gerar erro de compilação
// uma vez que ele tentará invocar um método de instância
// através do nome da classe. Corrija o este erro de compilação
char f = String.charAt(2);
```

4. Compile o programa. O seguinte erro será gerado pelo compilador:

**Cannot make a static reference to the non-static method charAt(int) from the type String**

5. Corrija o erro de compilação e novamente compile e execute o programa.

6. Como vou saber se um método qualquer na biblioteca de Java é estático ou de instância? A resposta é: consulte a documentação da **API Java** ([http://docs.oracle.com/javase/6/docs/api/java/lang/String.html#charAt\(int\)](http://docs.oracle.com/javase/6/docs/api/java/lang/String.html#charAt(int))) e veja se o método em questão possui em sua assinatura a palavra chave **static**. Veja na figura:



**charAt**

```
public char charAt(int index)
```

Returns the char value at the specified index. An index ranges from 0 to `length() - 1`. The first char value of the sequence is at index 0, the next at index 1, and so on, as for array indexing.

If the char value specified by the index is a [surrogate](#), the surrogate value is returned.

**Specified by:**

- [charAt](#) in interface [CharSequence](#)

**Parameters:**

- `index` - the index of the char value.

**Returns:**

- the char value at the specified index of this string. The first char value is at index 0.

**Throws:**

- [IndexOutOfBoundsException](#) - if the `index` argument is negative or not less than the length of this string.

---

**codePointAt**

```
public int codePointAt(int index)
```

Returns the character (Unicode code point) at the specified index. The index refers to char values (Unicode code units) and ranges from 0 to [length\(\)](#) - 1.

If the char value specified at the given index is in the high-surrogate range, the following index is less than the length of this `String`, and the char value at the following index is in the low-surrogate range, then the supplementary code point corresponding to this surrogate pair is returned. Otherwise, the char value at the given index is returned.

**Parameters:**

- `index` - the index to the char values

**Returns:**

- the code point value of the character at the `index`

**Throws:**

- [IndexOutOfBoundsException](#) - if the `index` argument is negative or not less than the length of this string.

**Since:**

- 1.5

7. Acrescente o trecho de código abaixo, **Listagem-2.4**, no programa anterior. Consulte na documentação da API se os métodos são estáticos ou de instância e faça a correta invocação do método.

```
// método endsWith()
String str = "Hello";
System.out.println( str.endsWith( "slo" ) );

// método floor()
System.out.println( Math.floor(3.14));

// método isDigit()
System.out.println( "0=" + Character.isDigit('0'));
System.out.println( "A=" +Character.isDigit('A'));
```

Listagem 2.4 – Identifique quais métodos são de classe ou de instância

### Exercício 3: Argumentos de métodos passado por valor e por referência

Neste exercício, você irá praticar o conceito da passagem de parâmetro por valor. Note que os parâmetros de tipos primitivos são passados o valor, enquanto parâmetros do tipo referência são passados a referência. Um array é considerado um tipo referência mesmo que os valores armazenados nele sejam de um tipo primitivo.

#### 3.1 Passagem de parâmetros por valor

1. Escreva, compile e execute a **Listagem-2.5, TestaPassagemValor.java**, preste bastante atenção aos comentários.

```
public class TestaPassagemValor {

    public static void main(String[] args) {

        int i = 10;

        // Imprime valor de i
        System.out.println("inicia método main e i = " + i);

        // Chama método test, que está definido abaixo
        // e passa valor inteiro como um parâmetro. Uma vez que
        // int é um tipo primitivo, este argumento é passado
        // por valor.
        test(i);

        // imprime o valor de i, note que valor de i não muda
        System.out.println("termina o método main e i = " + i);
    }

    // Método estático na classe
    public static void test(int j) {

        System.out.println("inicia método test e j = " + j);

        // muda valor parâmetro i
        j = 33;
        System.out.println("termina método test e j = " + j);
    }

}
```

Listagem 2.5 – Passagem de parâmetro por valor

2. Modifique **TestaPassagemValor.java** como segue abaixo e execute a classe.

- 2.1. Passe um segundo parâmetro de tipo primitivo para o método **test**, exemplo **test(int j, int k)** - você pode escolher qualquer parâmetro de tipo primitivo (tal como long ou boolean).
- 2.2. Atribua um valor ao segundo parâmetro antes de chamar o método **test(...)**. E mude o valor do parâmetro passado dentro do método **test(...)**, como você fez com o primeiro parâmetro na **Listagem-2.5** acima.
- 2.3. Modifique os métodos **System.out.println(...)** para indicar os valores do primeiro parâmetro e do segundo parâmetro.

### 3.2 Passagem de parâmetros por referência

1. Escreva, compile e execute a **Listagem-2.6, TestaPassagemReferencia.java**, preste bastante atenção aos comentários.

```
import java.util.Arrays;

public class TestaPassagemReferencia {

    public static void main(String[] args) {

        // criando um array de inteiros
        int[] i = { 10, 20, 30 };

        // Imprime valor de i
        System.out.println("inicia metodo main e i = " + Arrays.toString(i));

        // Chama método test que está definido abaixo,
        // e passa por referência o array de inteiro como um parâmetro.
        // Uma vez que um array é do tipo referência,
        // este argumento é passado
        // por referência.
        test(i);

        // imprime o valor de i. Note que o valor de i não muda
        System.out.println("termina o método main e i = " + Arrays.toString(i));
    }

    // Método estático na classe
    public static void test(int[] j) {

        System.out.println("inicia método test e j = " + Arrays.toString(j));

        // muda valor do parâmetro i
        j[0] = 33;
        j[1] = 66;

        System.out.println("termina método test e j = " + Arrays.toString(j));
    }

}
```

Listagem 2.6 – Passagem de parâmetros por referência

2. Observe que a classe utiliza o método **toString()** da classe **java.util.Arrays** para podermos imprimir o conjunto de elementos do array, apesar de ainda não termos visto arrays em Java o exercício demonstra bem o conceito.

3. É bom saber que a referência é tratada como um tipo primitivo, a final ele simplesmente armazena o endereço de um objeto, logo a referência é passada por valor. Pense nisso.

### Exercício 4: Escopo de variáveis

Neste exercício, você praticará o conceito de escopo de variável. Você também aprenderá como declarar três tipos de variáveis: **variável estática**, **variável de instância** e a **variável local**.

## 4.1 Um programa que demonstra escopo de variáveis

1. Escreva, compile e execute a **Listagem-2.7, EscopoDeVariavel.java**, preste bastante atenção aos comentários.

```
public class EscopoDeVariavel {  
    public static void main(String[] args) {  
        int var1 = 10;  
        if (var1 < 100) {  
            int var2 = 20;  
        } else {  
            int var2 = 21;  
        }  
  
        // Acesso a var1 é permitido, então não há erro de compilação.  
        System.out.println("valor de var1 = " + var1);  
  
        // Acesso a var2 não é permitido, então erro de compilação será gerado  
        System.out.println("valor de var2 = " + var2);  
    }  
}
```

Listagem 2.7 – Escopo de variável

2. Observe que haverá um erro de compilação. Isto era esperado porque você está tentando ter acesso a variável **var2** e ela foi declarada dentro das **{ }** do bloco **if/else**, então o escopo de acesso é somente dentro das **{ }** e não pode usada fora de seu escopo.

3. Modifique **EscopoDeVariavel.java**, como mostra a **Listagem-2.8**. Observe os comentários.

```
public class EscopoDeVariavel {  
    public static void main(String[] args) {  
        int var1 = 10;  
        // var2 agora esta definida em todo escopo do metodo main  
        int var2 = 15;  
  
        if (var1 < 100) {  
            // int var2 = 20;  
            var2 = 20;  
        } else {  
            // int var2 = 21;  
            var2 = 21;  
        }  
  
        // Acesso a var1 é permitido, então não há erro de compilação.  
        System.out.println("valor de var1 = " + var1);  
  
        // Acesso a var2 agora é permitido, não há erro de compilação  
        System.out.println("valor de var2 = " + var2);  
    }  
}
```

Listagem 2.8 – EscopoDeVariavel.java modificado

4. Compile e execute o programa modificado.

## 4.2 Um programa que usa três tipos de variáveis

1. Escreva, compile e execute a **Listagem-2.9, TresTiposDeVariaveis.java**, preste bastante atenção aos comentários.

```
public class TresTiposDeVariaveis {  
  
    // Exemplo de variável estática  
    static String staticVariable = "Variável de classe, ou estática";  
  
    // Exemplo de variável de instância  
    String instanceVariable = "variável de instância";  
  
    public static void main(String[] args) {  
  
        String localVariable = "variável local";  
        System.out.println("Variável estática = " + staticVariable);  
        System.out.println("Variável de instância = " + instanceVariable);  
        System.out.println("Variável local = " + localVariable);  
  
    }  
}
```

Listagem 2.9 – TresTiposDeVariaveis.java

2. Observe que variáveis de instância não podem ser referenciadas dentro de um contexto estático, o método **main()** é estático.

3. Comente a linha de código contendo o erro, compile e execute o programa.

4. Pense em outra forma de resolver este problema se for necessário acessar a variável de instância **instanceVariable**.

## Exercício 5: Casting de tipos primitivos e classe Wrapper

Neste exercício, você praticará a moldagem (**casting**) de tipos primitivos e como converter primitivos às classes de empacotamento (**Wrapper**) correspondentes e vice versa. **Casting** de tipos de referência serão exercitados em outro laboratório.

1. **Casting** de tipos primitivos.
2. Convertendo primitivos para classes **Wrapper**.

### 5.1. Casting de tipos primitivos

1. Escreva, compile e execute a **Listagem-2.10, CastingPrimitivos.java**, preste bastante atenção aos comentários.



```
public class CastingPrimitivos {  
  
    public static void main(String[] args) {  
  
        // casting implicito exemplo 1  
        int numInt = 10;  
        double numDouble = numInt;  
        System.out.println("int " + numInt + " e implicitamente moldado para double " +  
numDouble);  
  
        // casting implicito exemplo 2  
        int numInt1 = 3;  
        int numInt2 = 2;  
        double numDouble2 = numInt1 / numInt2;  
        System.out.println("numInt1/numInt2 " + numInt1 / numInt2 + " e implicitamente moldado  
para " + numDouble2);  
  
        // casting explicito exemplo 1  
        double valDouble = 10.12;  
        int valInt = (int) valDouble;  
        System.out.println("double " + valDouble + " e explicitamente moldado para int " +  
valInt);  
  
        // casting explicito exemplo 2  
        double x = 10.2;  
        int y = 2;  
        int resultado = (int) ( x / y );  
        System.out.println("x/y " + x / y + " e explicitamente moldado para int " + resultado);  
    }  
}
```

Listagem 2.10 – CastingPrimitivos.java

2. Compile e execute o programa, veja o resultados:

***int 10 e implicitamente moldado para double 10.0  
numInt1/numInt2 1 e implicitamente moldado para 1.0  
double 10.12 e explicitamente moldado para int 10  
x/y 5.1 e explicitamente moldado para int 5***

Em (1) e (2) ocorre uma conversão explícita dos tipos, ou seja, o Java o faz automaticamente. Em (2) ocorre uma divisão de números inteiros resultado em um número inteiro que é transformado em double. Já em (3) há uma divisão de inteiro/double resultando em um double. Neste caso dizemos que está ocorrendo uma promoção numérica ou um alargamento do tipo.

3. Modifique o programa para fazer casting dos tipos **long -> byte, float -> short, int -> char**.

## 5.2. Convertendo primitivos para classes Wrapper

1. Escreva, compile e execute a **Listagem-2.10, CastingPrimitivos.java**, preste bastante atenção aos comentários.

```
public class PrimitivaParaWrapper {  
  
    public static void main(String[] args) {  
  
        // cria uma instância de objeto Integer
```

```
Integer intObjeto = new Integer(7801);

// Converte de Integer para primitivo int usando método intValue()
int intPrimitiva = intObjeto.intValue();
System.out.println("int intPrimitiva = " + intPrimitiva);

// Usando método estático da classe empacotadora Integer
// para converter uma String para o tipo primitivo int
String strInt = "65000";
int intConvertida = Integer.parseInt(strInt);
System.out.println("int intConvertida = " + intConvertida);

// Converte int primitivo para tipo Integer
Integer intObjeto2 = new Integer(intConvertida);
System.out.println("Integer intObjeto2 = " + intObjeto2);
}

}
```

Listagem 2.10 – *PrimitivaParaWrapper.java*, converte de primitivos para empacotadora e vsv

2. Compile e rode o programa, observe e procure entender o resultado.

3. Modifique o programa da **Listagem-2.10** para criar uma variável do tipo **Long** e convertê-la em seu tipo primitivo **long**, depois mostre seu valor.

4. Em **Java 5.0** estas conversões ocorrem implicitamente, pois agora há o recurso de **Autobox** (empacotar) e **AutoUnbox** (desempacotar). Modifique a **Listagem-2.10** para que fique como na **Listagem-2.11**. Perceba como isso torna menos burocrático o processo de conversão de primitivos para **Wrapper** e vice-versa.

```
public class AutoBoxUnbox {

    public static void main(String[] args) {

        // cria uma instância de objeto Integer, autobox
        Integer intObjeto = 7801;

        // Converte de Integer para primitivo int, auto-unbox
        int intPrimitiva = intObjeto;
        System.out.println("int intPrimitiva = " + intPrimitiva);

        // Usando método estático da classe empacotadora Integer
        // para converter uma String para o tipo Integer, autobox
        String strInt = "65000";
        Integer intConvertida = Integer.parseInt(strInt);
        System.out.println("int intConvertida = " + intConvertida);

        // Converte Integer para primitivo int, autoUnbox
        int intPrimitiva2 = intConvertida;
        System.out.println("Integer intObjeto2 = " + intPrimitiva2);
    }

}
```

Listagem 2.11 - AutoBoxUnbox.java

## Exercício 6: Comparando objetos

1. Escreva, compile e execute a **Listagem-2.12, TestaIgualdadeObjeto.java**, preste bastante atenção aos comentários.

```
public class TestaIgualdadeObjeto {  
  
    public static void main(String[] args) {  
  
        // Declara duas variáveis do tipo String, str1 e str2  
        String str1, str2;  
  
        // Inicializando as variáveis. Note que as variáveis contêm  
        // ponteiros para instância de mesmo objeto atual.  
        str1 = "Viver sem Deus...não é viver!";  
        str2 = str1;  
  
        // Mostra o valor das variáveis str1 e str2  
        System.out.println("String1: " + str1);  
        System.out.println("String2: " + str2);  
  
        // A notação "==", quando é usada com variáveis de referência,  
        // se retornar true significa que as variáveis apontam para  
        // mesma instância de um objeto, e se false não, são referências diferentes  
  
        // Checa se str1 e str2 são o mesmo objeto para as duas variáveis  
        // apontando para mesma instância de um objeto do tipo String  
        System.out.println("Mesmo objeto? " + ( str1 == str2 ));  
  
        // Reinicializa variável str2. Ela agora aponta para um nova  
        // instância de objeto String  
        str2 = new String(str1);  
  
        // Mostra valor das variáveis str1 e str2  
        System.out.println("String1: " + str1);  
        System.out.println("String2: " + str2);  
  
        // Checa novamente se str1 e str2 são os mesmos  
        // objetos para as duas variáveis  
        // apontando para mesma instância de um objeto do tipo String  
        System.out.println("Mesmo objeto? " + ( str1 == str2 ));  
  
        // Checa se str1 e str2 tem o mesmo valor  
        System.out.println("Mesmo valor? " + str1.equals(str2));  
    }  
}
```

Listagem 2.12 – TestaIgualdadeObjeto.java

2. Compile e execute o programa.

3. Crie um programa como da **Listagem-2.12** para criar e comparar dois objetos da classe **Integer**.

4. String são objetos especiais em Java, podemos criar instâncias de objetos apenas atribuindo um literal a uma variável do tipo String, exemplo:

```
String str1 = "Viver sem Deus...não é viver!";
```

```
String str2 = "Viver sem Deus...não é viver!";
```

além disso, cada objeto String criado desta forma apontam para mesma instância do objeto, ou seja qualquer literal String igual ao anterior apontará para o mesmo objeto.

Modifique o programa da **Listagem-2.12** para demonstrar essa propriedade

## Exercício 7: método `getClass()` e o operador `instanceof`

Neste exercício, você usará o método do **`getClass()`** da classe **Object** para encontrar uma instância de um objeto fora da classe. Você aprenderá também como usar o operador **`instanceof`** para testar se uma instância de objeto qualquer é de um tipo particular de classe.

1. Usando método **`getClass()`**
2. Usando operador **`instanceof`**

### 7.1. Usando método `getClass()`

1. Escreva, compile e execute a **Listagem-2.13**, **TestaGetClass.java**, preste bastante atenção aos comentários.

```
public class TesteGetClass {  
  
    public static void main(String[] args) {  
  
        // Cria instância de objeto String  
        String str1 = "A vida e para ser vivida... com Deus!";  
  
        // Encontrar informação externa de uma instância de String  
        // via método getClass(). Note que ele retorna uma instância  
        // de objeto da classe Class  
        Class str1Class = str1.getClass();  
        System.out.println("A classe de str1 e uma instância de " + str1Class);  
  
        // O nome da classe da instância de objeto Class.  
        String str1ClassName = str1Class.getName();  
        System.out.println("Nome da classe e " + str1ClassName);  
  
        // Cria instância de objeto Integer  
        Integer int1 = new Integer(34);  
  
        // Encontrar informação externa de uma instância de Integer  
        // via método getClass(). Note que ele retorna uma instância  
        // de objeto da classe Class  
        Class int1Class = int1.getClass();  
        System.out.println("A classe de int1 é uma instância de " + int1Class);  
  
        // O nome da classe da instância de objeto Class.  
        String int1ClassName = int1Class.getName();  
        System.out.println("Nome da classe é " + int1ClassName);  
  
    }  
  
}
```

Listagem 2.13 – TestaGetClass.java

2. Monte, compile e execute.

3. Modifique o programa para criar uma instância de objeto da classe **java.util.Date** e mostre informações desta classe.

## 7.2. Usando operador instanceof

1. Escreva, compile e execute a **Listagem-2.14, TestaInstanceOf.java**, preste bastante atenção aos comentários.

```
public class TestaInstanceOf {  
  
    public static void main(String[] args) {  
  
        // Criar objeto String  
        String str1 = "Aprenda Java em Dez anos!";  
        Integer int1 = new Integer(40);  
  
        // Checa se str1 é do tipo String usando operador instanceof.  
        // Checa também se é do tipo Object.  
        boolean b1 = str1 instanceof String;  
        System.out.println("str1 é String: " + b1);  
        boolean b2 = str1 instanceof Object;  
        System.out.println("str1 é Object: " + b2);  
  
        // Checa se int1 é do tipo Integer usando operador instanceof.  
        // Checa também se é do tipo Object.  
        b1 = int1 instanceof Integer;  
        System.out.println("int1 é do tipo Integer: " + b1);  
        b2 = int1 instanceof Object;  
        System.out.println("int1 é Object: " + b2);  
        b2 = int1 instanceof Number;  
        System.out.println("int1 é do tipo Number: " + b2);  
  
    }  
  
}
```

Listagem 2.14 – TestaInstanceOf.java

2. Consultando a documentação da **API Java**, você vai observar que classe **Integer** é uma classe filha da classe **java.lang.Number**, e que esta é filha de **java.lang.Object**. Por isto o teste de **int1 instanceof Number** retorna **true**.

docs.oracle.com/javase/6/docs/api/java/lang/Integer.html

Produtor Con... PrimeFaces - ShowC... Telebras-Sicacc Telebras-Indicadores Sistema de Gestão d... Gmail: e-mail do Go... Minha Visão - Manti... Application Livecycl... Redmine

Esta página está em inglês Deseja traduzi-la? Traduzir Não Nunca traduzir do inglês Opções

Overview Package **Class** Use Tree Deprecated Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes  
SUMMARY: NESTED FIELD CONSTR METHOD DETAIL: FIELD CONSTR METHOD

java.lang

## Class Integer

java.lang.Object  
└ java.lang.Number  
└ java.lang.Integer

All Implemented Interfaces:  
[Serializable](#), [Comparable<Integer>](#)

```
public final class Integer
extends Number
implements Comparable<Integer>
```

The Integer class wraps a value of the primitive type `int` in an object. An object of type `Integer` contains a single field whose type is `int`.

In addition, this class provides several methods for converting an `int` to a `String` and a `String` to an `int`, as well as other constants and methods useful when dealing with an `int`.

Implementation note: The implementations of the "bit twiddling" methods (such as [highestOneBit](#) and [numberOfTrailingZeros](#)) are based on material from Henry S. Warren, Jr.'s *Hacker's Delight*, (Addison Wesley, 2002).

Since:  
JDK1.0

See Also:  
[Serialized Form](#)

3. Modifique o programa para criar uma instância de objeto da classe **Long** e mostre que o objeto é do tipo **Object**, **Number** e da própria classe **Long**.