Técnicas de Programação II

Aula 02 – Objetos e Strings

Edirlei Soares de Lima
<edirlei.lima@uniriotec.br>

Orientação a Objetos

- O ser humano se relaciona com o mundo através do conceito de **objetos**.
- Damos **nomes** a esses objetos e os classificamos em **grupos** (**classes**).
- Os objetos possuem:
  - **Características** pelas quais os identificamos (**Atributos**);
    - O objeto **Pessoa** possui RG, nome, data de nascimento...
    - O objeto **Carro** possui tipo, cor, quantidade de portas...
  - **Finalidades** para as quais os utilizamos (**Comportamentos**);
    - Um objeto do tipo **Pessoa** pode andar, correr ou dirigir carros.
    - Um objeto do tipo **Carro** pode ligar, desligar, acelerar, frear.

Classes e Objetos

- A unidade fundamental da programação orientada a objetos é a **classe**.

- A classe define o que os objetos devem ter (exemplo: tipo, cor, placa e número de portas...) e quais operações eles podem realizar.

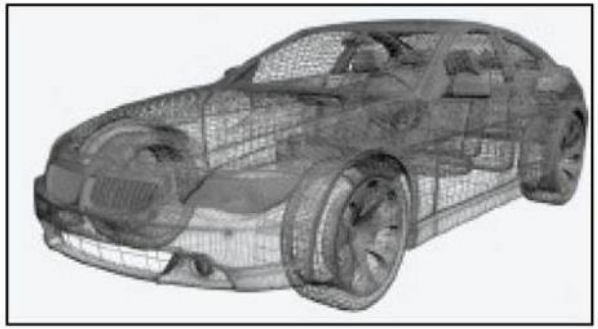
- As classes definem a **estrutura básica para a construção de objetos**.

Classe: Carro
Atributos: Tipo Cor Placa N° Portas ...
Métodos: Acelerar() Frear() TrocarMarcha(x) Buzinar() ...

Classes e Objetos

- Objetos são **instâncias** da **classe**.

Classe Carro



Objeto Carro1

Objeto Carro2

Objeto Carro3

Classes e Objetos

- A classe é o **modelo** ou **molde** de construção de objetos. Ela define as características e comportamentos que os objetos irão possuir.

- Sob o ponto de vista da programação orientada a objetos, um objeto não é muito diferente de uma **variável normal**.

- Um programa orientado a objetos é composto por um **conjunto de objetos** que interagem entre si.

Classes e Objetos

- Para manipularmos objetos em Java precisamos declarar **variáveis de objetos.**
- Uma variável de objeto é uma **referência** para um objeto.
- A **declaração de uma variável de objeto** é semelhante a declaração de uma variável normal:

```
Carro carro1;     /* carro1 não referencia nenhum objeto,
                     o seu valor inicial é null */
```

- A simples declaração de uma variável de objeto não é suficiente para a **criação de um objeto**.

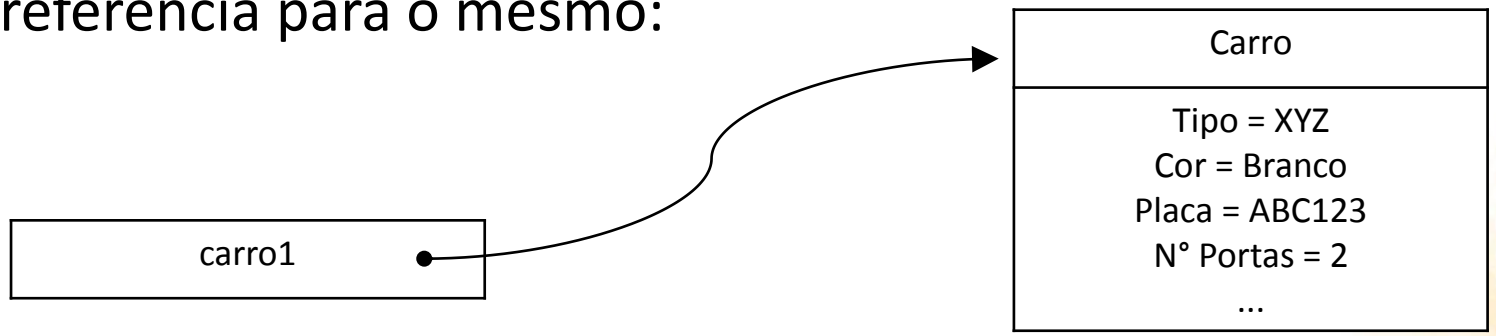
Classes e Objetos

- A criação de um objeto deve ser explicitamente feita através do operador **new**:

```
Carro carro1;     /* carro1 não referencia nenhum objeto,
                     o seu valor inicial é null */

carro1 = new Carro();  /* instancia o objeto carro1 */
```

- O operador new **aloca o objeto em memória** e retorna uma referência para o mesmo:



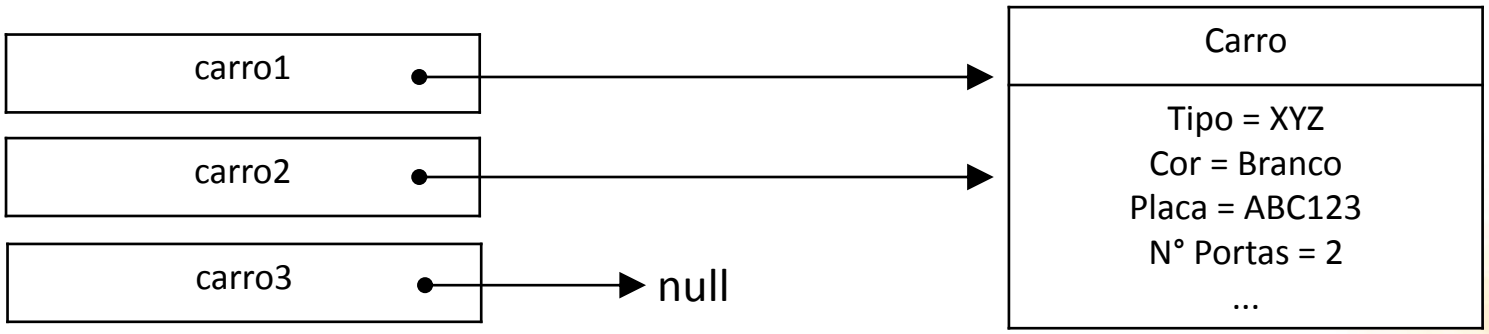
Classes e Objetos

- Como as variáveis de objeto são referências, as operações de atribuição entre elas **não criam novos objetos**:

```
Carro carro1, carro2, carro3;

carro1 = new Carro();

carro2 = carro1;  /* carro2 passa a referenciar o mesmo
                     objeto referenciado por carro1 */
```



Classes e Objetos

- Um objeto expõe o seu **comportamento** através de métodos (funções).

- É possivel **invocar métodos** dos objetos instanciados:

```
Carro carro1 = new Carro();
Carro carro2 = new Carro();

carro1.mudarMarcha(2);
carro1.aumentaVelocidade(5);

carro2.mudarMarcha(4);
carro2.aumentaVelocidade(10);
```

A Classe String

- Strings são **sequências de caracteres**. Na linguagem Java, strings são instâncias da classe **String**.

- O modo mais direto de criar uma string é:

```
String s = "Técnicas de Programação II";
```

- Sempre que o compilador encontra uma **string literal** no código de um programa ele cria um objeto da classe **String** contendo o valor do literal.

A Classe String

- Como qualquer outro objeto, uma string pode ser criada pelo operador **new**.

- A classe String possui 15 **metodos construtores**, que nos permitem fornecer o valor inicial da string usando diferentes fontes, tais como arrays de caracteres:

```
char data[] = {'U', 'N', 'I', 'R', 'I', 'O'};
String str = new String(data);
```

```
String str = new String("UNIRIO");
```

```
String str = "UNIRIO";
```

Métodos da Classe String

- Classe **String** fornece um **conjunto de métodos** para a manipulação de cadeias de caracteres.

- Um destes métodos é o método `length`, o qual retorna um valor inteiro igual ao **número de caracteres** contidos na string:

```
int length();
```

- Exemplo:

```
String str = "Técnicas de Programação II";

int tamanho = str.length();

System.out.println("Tamanho: " + tamanho);
```

Métodos da Classe String

- Outro método da classe String é o método `charAt`, o qual permite o acesso individual aos caracteres contidos na string:

```
char charAt(int index);
```

- Exemplo:

```
String str = "Técnicas de Programação II";

System.out.println("Primeira Letra: " + str.charAt(0));

System.out.println("Ultima Letra: " + str.charAt(str.length()-1));
```

Resultado:

```
Primeira Letra: T
Ultima Letra: I
```

Métodos da Classe String

- A classe String possui um **método estático**, chamado `format`, que nos permite formatar uma string.

```
static String format(String format, Object... args);
```

- Exemplo:

```
String str1;

str1 = String.format("Alunos: %d \nMédia: %.2f", 25, 7.45);

System.out.println(str1);
```

Resultado:

```
Alunos: 25
Média: 7,45
```

Métodos da Classe String

- **Alguns outros métodos da classe String:**
  - `boolean  contains(String s)`
  - `int  indexOf(String str)`
  - `String  replace(char oldChar, char newChar)`
  - `String  substring(int beginIndex, int endIndex)`
  - `String  toLowerCase()`
  - `String  toUpperCase()`
  - `String  trim()`

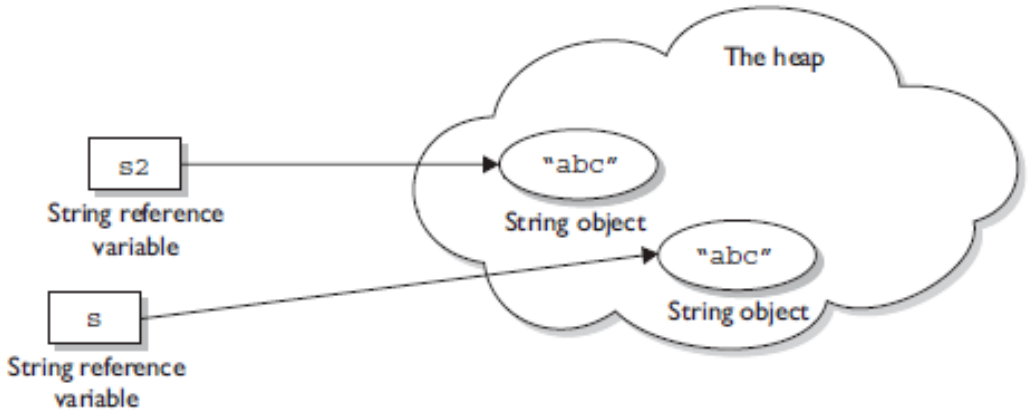
Lista completa: http://docs.oracle.com/javase/7/docs/api/java/lang/String.html

Igualdade de Objetos

- As variáveis de objetos em Java fazem referencia aos objetos instanciados em memória.

```
String s = new String("abc");

String s2 = new String("abc");
```



Igualdade de Objetos

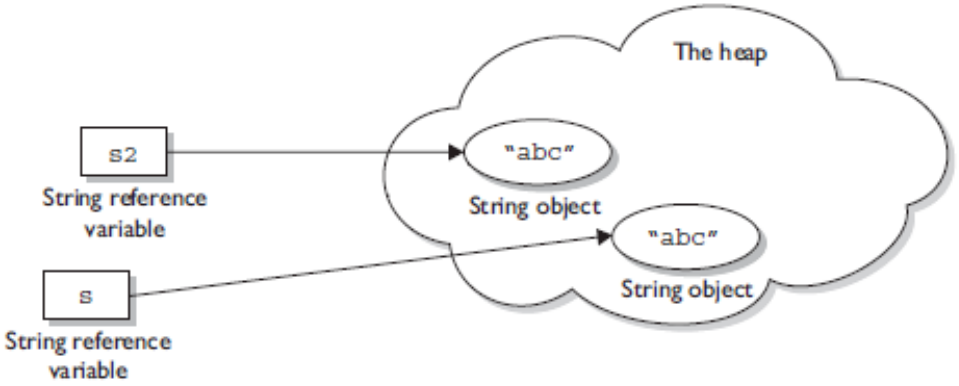
- Ao usarmos o operador == para compararmos variáveis de objetos, estamos verificando se ambas apontam para o mesmo objeto instanciado em memória.

```
String s = new String("abc");
String s2 = new String("abc");

if (s1 == s2)
  System.out.println("É igual!");
else
  System.out.println("Não é igual!");
```

Resultado:

```
Não é igual!
```



Igualdade de Strings

- Para compararmos o conteúdo de uma String devemos utilizar os métodos de comparação fornecidos pela classe String:

  - **`boolean equals(Object anObject)`** – retorna true se, e somente se, a string representa a mesma sequência de caracteres que o objeto passado como argumento. Exemplo:

```
String s = new String("abc");
String s2 = new String("abc");

if (s1.equals(s2))
  System.out.println("É igual!");
else
  System.out.println("Não é igual!");
```

Resultado:

```
É igual!
```

Igualdade de Strings

- A classe String também fornece outros métodos para **comparação de Strings**:
  - **`boolean endsWith(String suffix)`** – retorna true se a string termina com a string passada como argumento.
  - **`boolean startsWith(String prefix)`** – retorna true se a string começa com a string passada como argumento.
  - **`int compareTo(String s)`** – compara duas strings lexicograficamente. Retorna um inteiro que indica se a string maior (retorno > 0), igual (retorno = 0) ou menor (retorno < 0) que a string passada como argumento.

Imutabilidade de Strings

- Em Java, strings são imutável. Uma vez criado um objeto da classe String, ele não pode ser modificado.
  - A classe String possui vários métodos que parecem alterar o conteúdo de uma string, mas estes métodos na verdade fazem é criar e retornar uma nova string que contém o resultado da operação.

```
String s = "abc";
String s2 = s;
s = s.concat("def");

System.out.println(s);
System.out.println(s2);
```

Resultado:

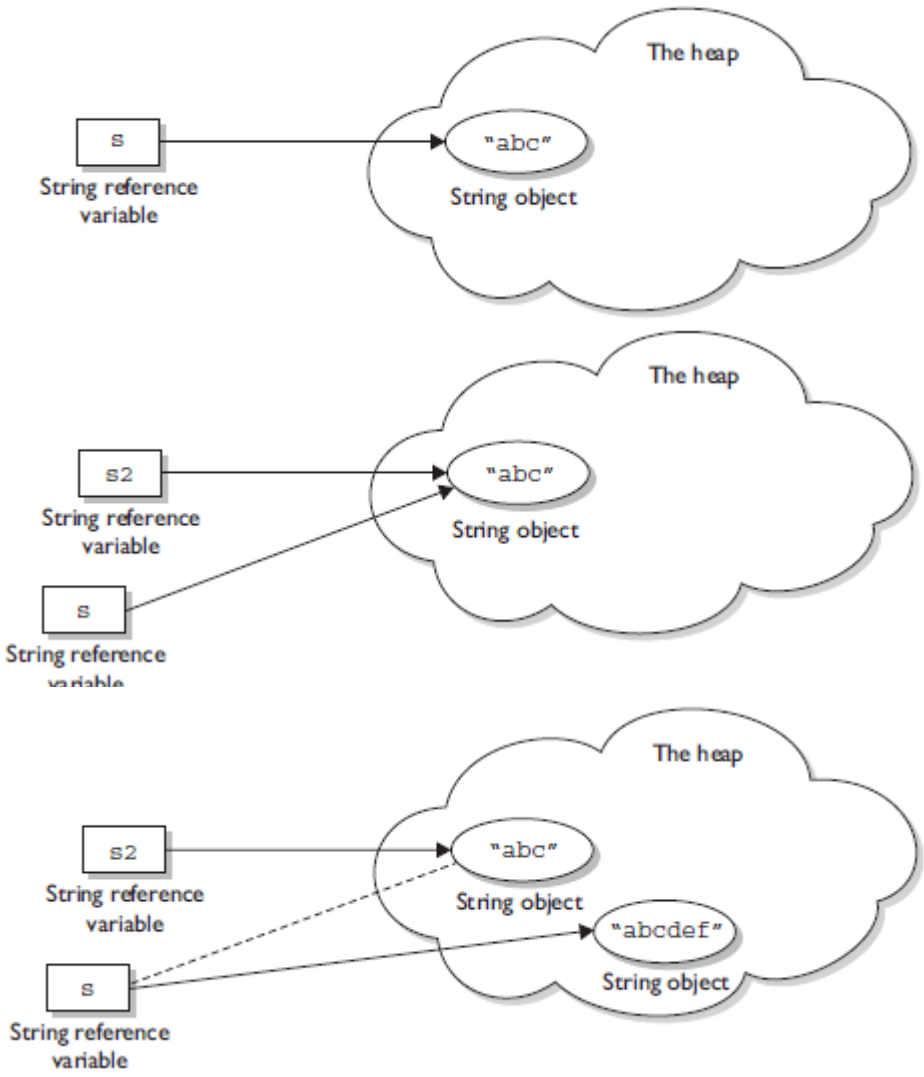
```
abcdef
abc
```

Imutabilidade de Strings

```
String s = "abc";

String s2 = s;

s = s.concat("def");
```



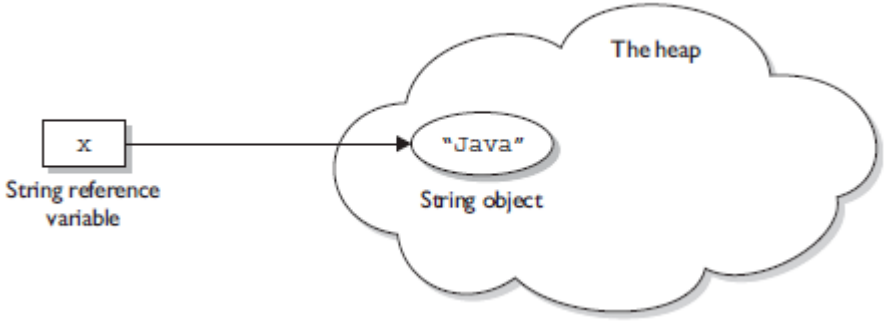
Imutabilidade de Strings

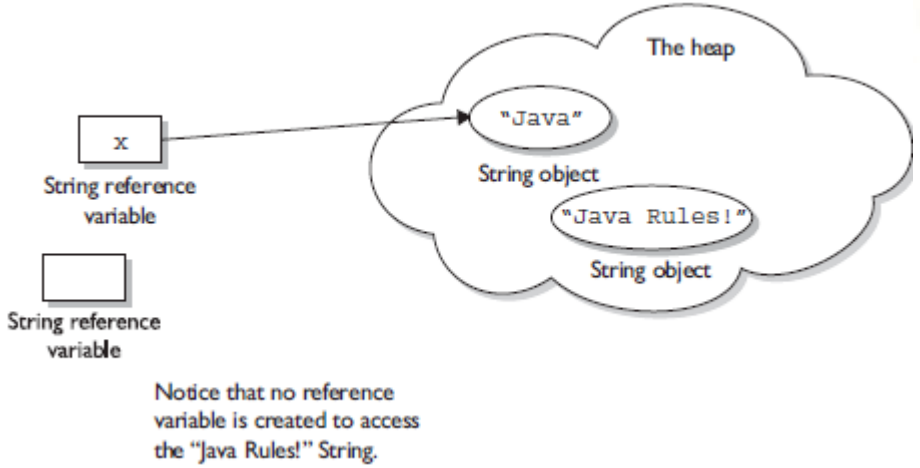
```
String x = "Java";
x.concat(" Rules!");

System.out.println(x);
```

Resultado:

```
Java
```





StringBuffer e StringBuilder

- Para se trabalhar com strings nas quais se deseja modificar o conteúdo, a biblioteca Java oferece as classes StringBuffer e StringBuilder.
  - Ambas as classes possuem a maioria dos métodos da classe String, além de métodos de alteração.

- A classe StringBuffer é sincronizada para acesso concorrente, enquanto que a StringBuilder não tem sincronização.

- Os principais métodos destas classes são:
  - `append(...)` – adiciona o parâmetro ao final da string;
  - `insert(int offset, ...)` – insere o parâmetro na posição;
  - `setCharAt(int index, char ch)` – altera o caractere na posição;

StringBuffer e StringBuilder

- Exemplo:

```
public static void main(String[] args)
{
  StringBuffer s1 = new StringBuffer("UNIRIO");

  s1.append(" - RJ");
  s1.insert(s1.indexOf("- ")+1, " BSX -");
  s1.setCharAt(s1.indexOf("X"), 'I');

  System.out.println(s1);
}
```

Resultado:

```
UNIRIO - BSI - RJ
```

Quando usar String e StringBuffer?

- Use String para manipular com valores constantes:
  - Valores literais;
  - Textos carregados de fora da aplicação;
  - Textos em geral que não serão modificados intensivamente.

- Use StringBuffer para alterar textos:
  - Acrescentar, concatenar, inserir, etc.

- Prefira usar StringBuffer para construir Strings
  - Concatenação de strings usando "+" é extremamente cara.

Leitura de Strings via Console

- A classe `java.util.Scanner`, além de fornecer funções para a leitura de variáveis simples (int, float, double...), também fornece funções para leitura de Strings **:**
  - `String nextLine()` – Retorna a ultima linha de texto digitada no console;

– **Exemplo:**

```
String nome;
Scanner entrada = new Scanner(System.in);
System.out.println("Digite o seu nome:");
nome = entrada.nextLine();

System.out.println(nome);
```

Manipulando Caracteres

- É possível acessar individual os caracteres contidos em uma Strings através do método `charAt`. Exemplo:

```
char opcao;
Scanner entrada = new Scanner(System.in);
System.out.println("Deseja continuar (s ou n):");
opcao = entrada.nextLine().charAt(0);

if (opcao == 's')
  System.out.println("Continua!");
else if (opcao == 'n')
  System.out.println("Não Continua!");
else
  System.out.println("Opção Invalida!");
```

Manipulando Caracteres

- É possível analisar os caracteres de uma Strings usando o método `charAt`. Exemplo:

```
public static int conta_letra(String str, char letra)
{
  int i, total = 0;
  for (i = 0; i < str.length(); i++)
  {
    if (str.charAt(i) == letra)
      total++;
  }
  return total;
}
```

```
String str1 = "Tecnicas de Programacao II";
System.out.println("Total: " + conta_letra(str1, 'a'));
```

Conversão de Strings

- Converter de String para int:

```
String str1 = "52";
int num1 = Integer.parseInt(str1);
```

- Converter de String para float:

```
String str1 = "28.6";
float num1 = Float.parseFloat(str1);
```

- Converter de String para double:

```
String str1 = "85.1";
double num1 = Double.parseDouble(str1);
```

Conversão de Strings

- Converter de int para String:

```
int num1 = 52;
String str1 = String.valueOf(num1);
```

- Converter de float para String:

```
float num1 = 28.6f;
String str1 = String.valueOf(num1);
```

- Converter de String para double:

```
double num1 = 85.1;
String str1 = String.valueOf(num1);
```

Exercícios

Lista de Exercícios 02 – Objetos e Strings

http://uniriodb2.uniriotec.br