Unicesumar

Programação I

Implementação de classes

Prof. Me. Fábio Perfetto

Classes e Objetos

- Classes fornecem o benefício da **reusabilidade**
- Programadores podem utilizar a mesma classe diversas vezes para criar os objetos

Conteúdo da Classe

- Atributos de Objeto
- Métodos de Objeto

- Atributos de Classe (atributos estáticos)
- Métodos de Classe (métodos estáticas)

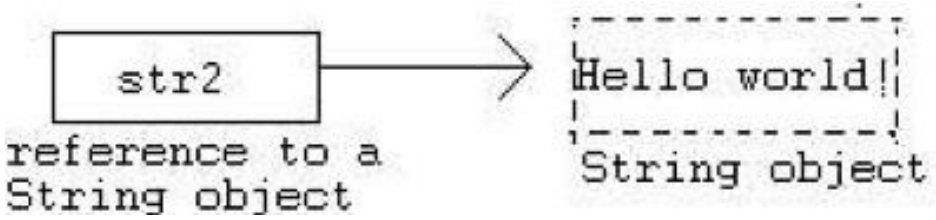
Instância de Classe

- Para criar um objeto, ou uma instância de uma classe, utilizamos o operador **new**
- Para criar um objeto da classe String, escreve-se o seguinte código:

```
String str2 = new String("Hello world!")
```

ou

```
String str = "Hello world!";
```



Instância de Classe

- Operador **new**
  - Aloca memória para um objeto e retorna uma referência
  - Ao criar um objeto, invoca-se o construtor da classe

Construtores

- Importantes na criação de um objeto
- É um método onde são colocadas todas as inicializações
- Possuem o mesmo nome da classe
- Não retornam valor
- Executados automaticamente na utilização do operador **new** durante a instanciação da classe

Exemplo

```
public class Pessoa {
    private String nome;
    private int idade;
    public Pessoa(){
    }
    public Pessoa(String nome){
       this.nome = nome;
    }
    public Pessoa(String nome, int idade){
       this.nome = nome;
       this.idade = idade;
    }
}
```

Construtor *default*

- Quando não especificamos construtores, Java provê um construtor default para nossa classe:
  - Toda classe precisa de um construtor. Se você não escreveu nenhum, Java provê um pra você;
  - Sem parâmetros e sem implementação.
- Quando especificamos construtores, o construtor *default* não é provido automaticamente:
  - Se você escreveu um construtor, Java assume que você sabe o que está fazendo e não provê um;
  - Chamar o construtor sem o parâmetro gera erro se ele não for definido explicitamente.

Construtores chamando construtores

- Podem ser cruzadas, o que significa que você pode chamar um construtor de dentro de outro construtor
- Deve sempre ocorrer na primeira linha de instrução
- Não pode ser usada fora de construtores.
- Utilizando a chamada this();

```
public Pessoa(String nome){
  this.nome = nome;
}
public Pessoa(String nome, int idade){
  this(nome);
  this.idade = idade;
}
```

Métodos

- É uma parte separada do código que pode ser chamada pelo programa principal
- Pode ou não retornar valor
- Pode aceitar tantos argumentos quantos forem necessários
- Após o fim da execução de um método, o fluxo de controle é retornado a quem o chamou

Porque usar Métodos?

- Decomposição – É a chave para a solução de problemas
- A criação de métodos resolve uma parte específica do problema
- Separar o problema em partes menores e manuseáveis

Declarando Métodos

- Para definir um método, especificamos:
  - Seu tipo de retorno;
  - Seu nome;
  - Seus parâmetros;
  - Sua implementação;

```
<modificador>* <tipoRetorno> <nome>(<argumento>*) {
<instrução>*
}
```

Assinatura de um método

- Define o método de uma forma única;
- Em Java, o nome e os tipos de parâmetros de um método formam sua assinatura;
- Não pode haver dois métodos com a mesma assinatura na mesma classe (mesmo que o tipo de retorno seja diferente);
- Algumas linguagens incluem o tipo de retorno na assinatura. Java não o faz.

Sobrecarga de Métodos

- Permite método com o mesmo nome mais diferentes argumentos(assinatura), possa ter implementações diferentes e retornar valores de diferentes tipos.
- Pode ser usado quando a mesma operação tem implementações diferentes.
- Propriedades:
  - mesmo nome;
  - argumentos diferentes;
  - tipo do retorno pode ser igual ou diferente;

Exemplo

```
public void imprimir(String nome){
System.out.println("Nome: " + nome);
}
public void imprimir(String nome, int idade){
System.out.println("Nome: " + nome + " Idade: "
  + idade);
}
public void imprimir(int idade, String nome){
System.out.println("Nome: " + nome + " Idade: "
  + idade);
}
```

Parâmetros variáveis

- partir do Java 5 se tornou possível definir métodos que recebem um número variável de argumentos;

```
public class Teste {
    void imprimir(boolean msg, String ... objs) {
        if (msg) System.out.println("Args:");
        for (int i = 0; i < objs.length; i++)
            System.out.println(objs[i]);
    }
    public static void main(String[] args) {
        Teste teste = new Teste();
        teste.imprimir(true, "Java", "Sun", "JCP");
    }
}
```

Parâmetros variáveis

- Só pode haver uma lista de parâmetros variáveis na declaração do método;
- Deve ser a última a ser declarada;
- Funciona como um vetor do tipo declarado (no exemplo, vetor de String);
- Não há limite para o número de parâmetros;
- Também aceita zero parâmetros.

```
teste.imprimir(false, "A", "B", "C", "D", "E");
teste.imprimir(true, "Um", "Dois");
teste.imprimir(false);
```

Múltiplos Comandos return

- Desde que eles não pertençam ao mesmo bloco
- Pode-se utilizar constantes para retornar valores, ao invés de atributos

```
public String getNumberInWords(int num) {
    String defaultNum = "zero";
    if (num == 1) {
      return "one";
    } else if (num == 2) {
      return "two";
    }
    return defaultNum;
}
```

Chamando Métodos Estáticos

- Métodos que podem ser invocados sem que um objeto tenha sido instanciado pela classe (sem invocar a palavra chave new)
- Pertencem à classe como um todo e não a uma instância (ou objeto) específica da classe
- São diferenciados dos métodos de instância pela declaração da palavra chave **static** na definição do método

```
NomeClasse.nomeMetodoEstatico(argumentos);
```

Chamando Métodos Estáticos

```
int i = Integer.parseInt("10");
String hexEquivalent = Integer.toHexString(i);
System.out.println(hexEquivalent);
```


Unicesumar