Retrospectiva Orientação a Objetos

O que é uma classe?

- estrutura de dados que armazena valores
- possui não apenas características (atributos), mas também possui funcionalidades (métodos)

Por que Programamos Orientado a Objetos?

Porque a linguagem não vai fornecer um tipo de dado pronto, bonitinho, “feito para você” com os atributos e métodos que você precisa.
Por conta dessa “*limitação*”, a linguagem fornece mecanismos para que você (programador) crie de forma versátil seus próprios tipos de dados
Ex: Em JAVA não existem tipos para armazenar Pessoas, Carros, BichinhosDeEstimacao, ProdutoDaLoja, Departamentos, Funcionarios, etc
Porém, java permite que você defina estes tipos

Evoluções Técnicas

***Praxis* para programadores**

- Em geral declaramos atributos como **PRIVATE**, para preservar a estrutura do objeto, blindando-a

- Por conta disso, criamos métodos de acesso. Tais métodos são conhecidos como GETTERS e SETTERS

    - Get (obter o valor do atributo)
    - Set (alterar/atribuir valor para o atributo)

- Os métodos em geral são declarados como **PUBLIC** para ficarem disponíveis e acessíveis de qualquer classe em qualquer diretório (*package*)

- Além disso, nos métodos onde há uma coincidência de nomenclatura (ex: nome de atributo coincidente com nome de parâmetro), usamos o operador **this** para nos referirmos ao atributo.

```
public void setNome(String nome){
  // aqui o "this" refere-se ao atributo nome
  this.nome = nome;
}
```

- Nomes de atributos e métodos seguem o padrão *Camel Case*, ou seja, a cada nova palavra que compõe o nome do método/atributo, colocamos uma letra maiúscula

```
  private int meuAtributoComVariasPalavras;
  public void meuMetodoQueTemVariasPalavras(){
     ...
  }
```

- Eventualmente podemos modificar a forma como instanciamos nosso objeto. Para isso criamos um método chamado *construtor*

    - Este método tem um cabeçalho específico: Obrigatoriamente não tem tipo de retorno e seu nome é exatamente igual ao nome da Classe
    - Pra que isso? Para definirmos a forma como daremos **new** no objeto

Situação Normal

```
Pesoa p = new Pessoa();
```

Com construtor

classe Pessoa.java

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

classe Principal.java

```
public static void main(String args[]){

   Pessoa p = new Pessoa("Isidro","isidro@isidro.com");
   ...
}
```

- Além disso, temos a possiblidade de criar várias “Versões” do mesmo método. O que diferencia cada versão? A quantidade e o tipo dos parâmetros. A isto damos o nome de **sobrecarga**. Sobrecarga é a capacidade de escrever vários métodos que possuem o mesmo nome e diferentes listas de parâmetros na mesma classe. Exemplos

```
public class Calculadora{
  public int soma(int x, int y){
     return x + y;
  }

  public float soma(float x, float y){
     return x + y;
  }

  public float soma(double x, double y){
     return x + y;
  }
}
```

usando a classe...

```
public class ClassePrincipal{
   public static void main(String args[]){
       Calculadora c = new Calculadora();
       int x = c.soma(1,2);
       float y = c.soma(2.0f, 3.0f);
       double z = c.soma(4.0, 5.0);
       System.out.println("X = " + x);
       System.out.println("Y = " + y);
       System.out.println("Z = " + z);
   }
}
```

Lembrando que a **sobrecarga** também aplica-se a construtores de objetos.

- Outra prática é a definição de atributos um por linha (isso é muito útil quando começamos a utilizar *frameworks*)

- Quando atribuimos um objeto a outro, na verdade estamos atribuindo referências, ou seja, não há duplicação da área de memória. O que acontece é que a nova referência aponta para o mesmo objeto da referência original. Exemplo:

```
Produto p = new Produto(...);
Produto p2 = p;
/* aqui temos o exemplo de p e p2 apontando para o mesmo objeto
 qual o efeito colateral?
 poder manipular o mesmo objeto tanto por p quanto por p2
 isso pode fazer com que alteremos algo através de p2 e isso
 seja refletido em p
 */
```

- outra coisa importante: Não precisamos nos preocupar com a liberação de memória. Quem realiza esta tarefa é o *Garbage Collector*

- comparação de Objetos (e Strings principalmente) só são possíveis através do método **equals**.