Aula 4 - Introdução à Programação Orientada a Objetos

Na aula anterior, já começamos a utilizar alguns objetos, como o objeto Scanner:

```
Scanner leitorDeDados = new Scanner(System.in);
```

Mas o que são objetos? E o que eles tem a ver com a linguagem Java?

Paradigmas de Programação

Vocês já pararam para pensar como devemos programar um computador? De que maneira precisamos instruí-lo para que ele realize suas tarefas?

Segundo o google:

Um paradigma de programação é um estilo, modelo ou metodologia de programação que apontam para a melhor forma de solucionar problemas usando uma determinada linguagem

O processador de um computador baseado na arquitetura de Von Neumann (ou seja, a quase totalidade dos computadores atuais) executa suas tarefas seguindo uma sequência de instruções, independente da linguagem de programação que escrevemos nossas aplicações. Tudo, em algum momento, é convertido para linguagem de máquina.

Então poderíamos escrever todas as nossas aplicações em linguagem de máquina, visto que é de fato o que o computador vai executar, sem intermediários.

Entretanto, a ciência da computação percebeu, ao longo de muitos anos que essa maneira de escrever programas possui diversos problemas. A complexidade dos programas aumentou exponencialmente ao longo do tempo, tornando a escrita de programas muito complexa para a grande maioria de nós.

Verificou-se que era possível a organização do código de maneiras que fossem mais fáceis dos programadores entenderem, baseadas em organização de código repetitivo, ou mesmo em metáforas mais próximas do mundo real que facilitassem o entendimento. Essas maneiras são conhecidas como paradigmas de programação.

Orientação a objetos

O paradigma OO entende que as instruções podem ser organizadas em partes, que podem ser instanciadas (os famosos objetos) e que contenham apenas o que necessário para que

funcionem, sem mais, nem menos.

Embora atingir esses objetivos nem sempre é uma tarefa fácil, podemos entender que é possível pensarmos em objetos como pequenos robôs virtuais, ou mesmo peças de uma grande máquina, com cada uma delas fazendo sua parte, podendo haver ou não peças iguais, tendo inclusive colaboração entre elas, com peças dependendo de outras peças.

O paradigma OO pode nos ajudar a entender como pensar em divisão de tarefas, seja entre classes, seja em aplicações inteiras.

Para entendermos um pouco mais o que o paradigma OO traz pra gente, vamos considerar o exemplo da aula 3, onde cadastramos um usuário usando a classe `Scanner` para leitura dos dados:

```
public class EntradaESaida {

	public static void main(String[] args) {
		Scanner leitorDeDados = new Scanner(System.in);

		System.out.println("Informe a idade do usuário:");
		int idade = leitorDeDados.nextInt();

		System.out.println("Informe o salário do usuário:");
		double salario = leitorDeDados.nextDouble();

		System.out.println("Informe o sexo do usuário:");
		char sexo = leitorDeDados.next().charAt(0);

		System.out.println("O usuário é casado?");
		boolean casado = leitorDeDados.nextBoolean();

		System.out.println("Informe o nome do usuário:");
		String nome = leitorDeDados.next();

		leitorDeDados.close();

		int umaVariavel = 45 * 3;
		int outraVariavel = 65;
		int maisOutraVariavel = umaVariavel + outraVariavel;


		double porcentagemDeAumento = 10;
		double salarioReajustado = salario + (salario / 100 *
porcentagemDeAumento);

		System.out.println(idade);
		System.out.println(salario);
```

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

            System.out.println(salarioReajustado);
            System.out.println("As informações da pessoa são:\n " +
            "Nome: " + nome
            + ", Idade: " + idade
            + ", Salário: " + salario
            + ", Sexo: " + sexo
            + ", Casado: " + casado);
      }
}
```

Esse exemplo trabalhava com cadastro de pessoas e também realizava muitas outras tarefas. Era o que precisava ser visto naquela aula.

Vamos nos concentrar nesses trechos de código:

```
public class EntradaESaida {

      public static void main(String[] args) {
            Scanner leitorDeDados = new Scanner(System.in);

            System.out.println("Informe a idade do usuário:");
            int idade = leitorDeDados.nextInt();

            System.out.println("Informe o salário do usuário:");
            double salario = leitorDeDados.nextDouble();

            System.out.println("Informe o sexo do usuário:");
            char sexo = leitorDeDados.next().charAt(0);

            System.out.println("O usuário é casado?");
            boolean casado = leitorDeDados.nextBoolean();

            System.out.println("Informe o nome do usuário:");
            String nome = leitorDeDados.next();

            leitorDeDados.close();

            System.out.println("As informações da pessoa são:\n " +
            "Nome: " + nome
            + ", Idade: " + idade
            + ", Salário: " + salario
```

```
            + ", Sexo: " + sexo
            + ", Casado: " + casado);
      }
}
```

Se tivermos que cadastrar outra pessoa e quisermos exibi-la, teríamos que repetir o código das últimas linhas.

Estamos falando de pessoas. E se usássemos uma estrutura como a abaixo para lidar com pessoas:

```
import java.util.Scanner;

public class CadastroDeUsuarios {

      public static void main(String[] args) {
            Scanner leitorDeDados = new Scanner(System.in);

            System.out.println("Informe a idade do usuário:");
            int idade = leitorDeDados.nextInt();

            System.out.println("Informe o salário do usuário:");
            double salario = leitorDeDados.nextDouble();

            System.out.println("Informe o sexo do usuário:");
            char sexo = leitorDeDados.next().charAt(0);

            System.out.println("O usuário é casado?");
            boolean casado = leitorDeDados.nextBoolean();

            System.out.println("Informe o nome do usuário:");
            String nome = leitorDeDados.next();

            leitorDeDados.close();

            Usuario usuario = new Usuario(idade, salario, sexo,
casado, nome);
            usuario.exibir();
      }
}
```

E a classe `Usuario` ficaria assim:

```
package primeiros.objetos;

class Usuario {

	int idade;
	double salario;
	char sexo;
	boolean casado;
	String nome;

	Usuario(int idade, double salario, char sexo, boolean casado,
String nome){
		this.idade = idade;
		this.salario = salario;
		this.sexo = sexo;
		this.casado = casado;
		this.nome = nome;
	}

	void exibir() {
		System.out.println("As informações da pessoa são:\n " +
				"Nome: " + nome
				+ ", Idade: " + idade
				+ ", Salário: " + salario
				+ ", Sexo: " + sexo
				+ ", Casado: " + casado);
	}
}
```

Se precisarmos criar outro usuário, podemos fazer o seguinte:

```
Usuario usuario = new Usuario(idade, salario, sexo, casado, nome);
Usuario outroUsuario = new Usuario(50, 564.45, 'M', false,
"Claudia");

usuario.exibir();
outroUsuario.exibir();
```

Percebe que não precisamos mais nos preocupar com a concatenação das informações? Escrevemos a concatenação apenas uma única vez, e usamos quantas vezes quiser.

Na verdade criamos uma classe `Usuario`, e enviamos um pedido a ela. Algo como:

- *usuário, por gentileza, exiba suas informações*

Outro exemplo:

considere que estamos trabalhando num sistema de banco. Uma das primeiras tarefas de um banco é a abertura de contas corrente.

O que costumamos fazer numa conta corrente?
Normalmente fazemos operações de saque e depósito. Então, que tal:

- criarmos uma classe ContaCorrente
- inserirmos código para fazer os pedidos de saque e depósito para um objeto da classe ContaCorrente

A classe, então, ficaria assim:

```
package primeiros.objetos;

public class ContaCorrente {

	void saque(double valor){

	}

	void deposito(double valor) {

	}
}
```

Agora que definimos o que queremos que a `ContaCorrente` faça, precisamos escrever a lógica necessária para essas operações.

Quando realizamos um saque, removemos dinheiro do saldo da conta. E, quando realizamos um depósito, adicionamos dinheiro a uma conta.

Então, é necessário que a conta tenha um saldo.
Uma vez que a conta tenha um saldo, podemos escrever as operações:

```
public class ContaCorrente {

	double saldo;

	ContaCorrente(double saldo){
		this.saldo = saldo;
	}

	void saque(double valor){
		saldo = saldo - valor;
```

```
	}

	void deposito(double valor) {
		saldo = saldo + valor;
	}
}
```

E se quisermos saber o saldo da conta? Sem problemas. Adicionamos mais uma operação à `ContaCorrente`.

```
	double saldo() {
		return saldo;
	}
```

Agora, com a ContaCorrente pronta, podemos criar uma classe `Banco`, e criar diversas contas corrente. Também vamos fazer algumas operações:

```
package primeiros.objetos;

public class Banco {

	public static void main(String[] args) {
		ContaCorrente umaConta = new ContaCorrente(1000);
		ContaCorrente outraConta = new ContaCorrente(2000);

		umaConta.saque(100);
		outraConta.deposito(200);

		System.out.println("Saldo de umaConta: " +
umaConta.saldo());
		System.out.println("Saldo de outraConta: " +
outraConta.saldo());
	}

}
```

O operador **new** nos ajuda a criar um objeto da classe `Usuario`.

E assim criamos nossos primeiros objetos, em contextos diferentes.

Atributos, Construtores e Métodos

Ao longo dos exemplos acima, tanto no caso do `Usuario` quanto no caso da `Conta Corrente` usamos essas características dos objetos. De forma resumida:

- o `saldo` é um **atributo** da classe `ContaCorrente`
- as operações de `saque()`, `deposito()` e `saldo()` são os **métodos** da classe `ContaCorrente`
- por último, o construtor de `ContaCorrente` encontra-se nas primeiras linhas da classe, e recebe o saldo como parâmetro

O método main

O método main é um método diferente de todos quando o assunto é programação orientada a objetos. De forma resumida, é o método que instancia todos os objetos necessários para a resolução do problema.

Exercícios

4.1 - Quais são os atributos, construtores e métodos da classe `Usuario`?
4.2 - A classe `ContaCorrente` já contém algumas operações. Entretanto, a equipe de produto do nosso banco entendeu que as contas corrente precisam ter rendimentos de 1% todo mes.
Crie um projeto novo no Eclipse e copie e cole as classes `Banco` e `ContaCorrente`. Implemente na `ContaCorrente` um método `render()` que não recebe parâmetro e não retorna nada. E, na classe `Banco`, chame esse método. Logo após, chame o método `saldo()` para confirmar se a conta rendeu de fato.