Programação Orientada a Objetos

Uma classe é uma abstração das características relevantes de um grupo de objetos do mundo real. Os objetos semelhantes podem ser agrupados em classe.

O conceito de classe é inspirado no conceito de classificação taxanômica da biologia. Na biologia, os organismos biológicos são agrupados com base nas suas características compartilhadas formando uma classificação taxanômica. Na biologia, as principais classificações taxonômicas são: reino, filo, classe, ordem, família , gênero e espécie.

Classificação científica Homo Sapiens

- Reino: Animalia
- Filo: Chordata
- Classe: Mammalia
- Ordem: Primates
- Família: Hominidae
- Gênero: Homo
- Espécie: Sapiens

Classificação científica Gorila

- Reino: Animalia
- Filo: Chordata
- Classe: Mammalia
- Ordem: Primates
- Família: Hominidae
- Gênero: Gorilla
- Espécia: Gorilla gorilla

No exemplo acima, o Homo Sapiens e o Gorilla coincidem até o nível da classificação taxonômica da Família.

Criando a nossa primeira Classe

Considere que as características relevantes a todas as lâmpadas são as seguintes:

- Uma lâmpada tem dois estados principais: “ligada” e “desligada”.
- Uma lâmpada começa com estado desligada.

```
//Lampada.java
public class Lampada {
    boolean estadoDaLampada = false;
}
```

No nosso arquivo teste, podemos fazer o seguinte:

```
//TesteLampada.java
public class testeLampada {
    public static void main(String[] args) {
```

```
        Lampada l1 = new Lampada();
        System.out.println(l1.estadoDaLampada);
    }
}
```

Até o momento, a nossa classe guarda uma informação sobre o estado da Lâmpada.

Criando o nosso primeiro método

Considere que as características/comportamentos relevantes a todas as lâmpadas são as seguintes:

- Uma lâmpada tem dois estados principais: "ligada" e "desligada".
- Uma lâmpada começa com estado desligada.
- Uma lâmpada começa com contador de vezes ligada igual a zero.
- Precisamos saber quantas vezes uma lâmpada foi ligada.
- Queremos modificar o estado da lâmpada.

Agora, vamos adicionar um outro atributo na nossa classe Lampada e dois métodos.

Um **método** é um trecho de código que pode ser chamado por um nome por um objeto ou por uma classe (Por enquanto, vamos nos concentrar nos métodos chamados pelos objetos). O método possui duas diferenças principais com relação as funções: * O método recebe implicitamente uma referência para o objeto que chamou o método através da palavra reservada `this`. * O método é capaz de acessar/modificar os dados contidos na classe.

```
//Lampada.Java
public class Lampada {
    boolean estadoDaLampada = false;
    int contadorVezesLigada = 0;

    void muda(){
        // estadoDaLampada e o método muda estão na mesma classe
        estadoDaLampada = !estadoDaLampada;
        // contadorVezesLigada e o método muda estão na mesma classe
        if(estadoDaLampada) contadorVezesLigada++;
    }
}
```

No nosso arquivo teste, podemos fazer o seguinte:

```
        Lampada l1 = new Lampada();
        l1.muda();
        l1.muda();
        l1.muda();
        System.out.println(l1.estadoDaLampada);
```

```
        System.out.println(l1.contadorVezesLigada);
    }
}

/*
Saída:
true
2
*/
```

Criando mais métodos

Sempre que possível, a classe deve prover métodos para acessar/alterar seu atributos. Vamos criar mais dois métodos: * Um método para mostrar as informações sobre a lâmpada * Um método para devolver o número de vezes que ela foi ligada.

```
//Lampada.java
public class Lampada {
    boolean estadoDaLampada = false;
    int contadorVezesLigada = 0;

    void muda(){
        // estadoDaLampada e o método muda estão na mesma classe
        estadoDaLampada = !estadoDaLampada;
        // contadorVezesLigada e o método muda estão na mesma classe
        if(estadoDaLampada) contadorVezesLigada++;
    }

    void info(){
        System.out.println("Status: " + (estadoDaLampada? "ligada" : "desligada" ));
        System.out.println("Contador: " + contadorVezesLigada);
    }

    int numVezesLigada(){
        return contadorVezesLigada;
    }

}
```

Customizando a criação de objetos

Em alguns casos, queremos inicializar os dados de uma maneira customizada. Para isso, podemos utilizar uma função especial usada no momento da construção. Todas as lâmpadas criadas começam desligadas e com o contador de vezes ligada igual a zero.

Considere agora que vamos adicionar como característica relevante a localização da lâmpada. Note que a localização da lâmpada não será a mesma para todas as lâmpadas.

```
public class Lampada {
    boolean estadoDaLampada = false;
    int contadorVezesLigada = 0;
    String local;

    Lampada(String local){
        //Para acessar a variável local que pertence ao objeto
        // utilizaremos a referência para o objeto que chamou o método
        // através da palavra reservada this.
        this.local = local;
    }

    void muda(){
        // estadoDaLampada e o método muda estão na mesma classe
        estadoDaLampada = !estadoDaLampada;
        // contadorVezesLigada e o método muda estão na mesma classe
        if(estadoDaLampada) contadorVezesLigada++;
    }

    void info(){
        System.out.println("Status: " + (estadoDaLampada? "ligada" : "desligada" ));
        System.out.println("Local: " + local);
        System.out.println("Contador: " + contadorVezesLigada);
    }

    int numVezesLigada(){
        return contadorVezesLigada;
    }

}
```

Acessando meus ancestrais

Na linguagem Java, a classe Object é a raiz da hierarquia de classe. Toda classe tem a classe Object como uma classe ancestral.

Essa classe ancestral possui alguns métodos, como:

- toString() devolve uma representação de string do objeto.
- getClass() devolve a classe do objeto.
- hashCode() devolve um código associado ao objeto.

Esses métodos definidos nas classes ancestrais podem ser acessados pelas classes descendentes:

```
public class testeLampada {
    public static void main(String[] args) {
        Lampada l1 = new Lampada("cozinha");
        System.out.println( l1.getClass() );
        System.out.println( l1.hashCode() );
        System.out.println( l1.toString() );
        System.out.println(l1);

    }
}

Saída:
class Lampada
1933863327
Lampada@7344699f
Lampada@7344699f
```

Conta Bancária Simplificada

As informações relevantes para uma conta bancária será nome do correntista, o saldo da conta e se a conta é especial ou não.

Restrições:

- Se a conta for especial, o correntista terá o direito de retirar mais dinheiro do que tem no saldo (ficar com o saldo negativo)

```
public class ContaBancariaSimplificada {
    String nomeCorrentista;
    double saldo;
    boolean contaEhEspecial;

    ContaBancariaSimplificada(String nomeCorrentista){
        this.nomeCorrentista = nomeCorrentista;
        this.saldo = 0;
        this.contaEhEspecial = false;
    }

    ContaBancariaSimplificada(String nomeCorrentista, double saldo){
        this.nomeCorrentista = nomeCorrentista;
        this.saldo = saldo;
        this.contaEhEspecial = false;
    }

    ContaBancariaSimplificada(String nomeCorrentista, double saldo, boolean contaEhEspecial)
        this.nomeCorrentista = nomeCorrentista;
        this.saldo = saldo;
```

```
        this.contaEhEspecial = contaEhEspecial;
    }

    public String toString(){
        return "nome: " + nomeCorrentista + "\n"
        + "saldo: " + saldo + "\n" +
        "ehEspecial: " + (contaEhEspecial ? "sim" : "nao");
    }

}
```

Classe para testar a classe ContaBancariaSimplificada:

```
public class TesteContaBancariaSimplificada {
    public static void main(String[] args) {
        ContaBancariaSimplificada c1 = new ContaBancariaSimplificada("Alef");
        ContaBancariaSimplificada c2 = new ContaBancariaSimplificada("Yarlei", 3000);
        ContaBancariaSimplificada c3 = new ContaBancariaSimplificada("Janaina", 150000, true

        System.out.println(c1);
        System.out.println(c2);
        System.out.println(c3);
    }
}
```

Fração

```
public class Fracao {
    private int numerador;
    private int denominador;

    public String toString(){
        return numerador + "/"+ denominador;

    }

    Fracao(int num){
        numerador = num;
        denominador = 1;
    }

    Fracao(int num, int den){
        int d = mdc(num, den);
        numerador = num/d;
        denominador = den/d;
```

```
}

private int mdc(int a, int b){
    while(b != 0){
        int r = a % b;
        a = b;
        b = r;
    }
    return a;
}

public int getDenominador(){
    return denominador;
}

public int getNumerador(){
    return numerador;
}


public Fracao somaFracao(Fracao f){
    return new Fracao(numerador * f.getDenominador() + f.getNumerador() * denominador,
    denominador * f.getDenominador());
}

public Fracao subFracao(Fracao b){
    return new Fracao(numerador * b.getDenominador() - b.getNumerador() * denominador,
    denominador * b.getDenominador());
}

public Fracao multFracao(Fracao c){
    return new Fracao(numerador * c.getNumerador(), denominador * c.getDenominador());
}

public static void main(String[] args) {
    Fracao f1 = new Fracao(5,3);
    System.out.println(f1);
    Fracao f2 = new Fracao(30,4);
    System.out.println(f2);
    Fracao f3 = f1.somaFracao(f2);
    System.out.println(f3);
    Fracao f4 = f1.subFracao(f2);
    System.out.println(f4);
    Fracao f5 = f1.multFracao(f2);
    System.out.println(f5);
}
```

```
}
```