Classes Abstratas e Interfaces

Métodos e Classes Abstratas

Vamos lembrar das classes que fizemos quando estávamos estudando `polimorfismo`.

```
public class Animal{
    // atributos ...

    public void mover(){
        // como o objeto se move
    }
}

public class Cachorro extends Animal{
    // atributos ...

    // override
    public void mover(){
        // correr
    }

    // sorbecarga
    public void mover(boolean isAgua){
        if(isAgua){
            // nadar
        } else{
            mover();
        }
    }
}

public class Passaro extends Animal{
    // atributos ...

    //override
    public void mover(){
        //voar
    }
```

```
}

public class Peixe extends Animal{
    // atributos ...

    public void mover(){
        //nadar
    }
}
```

O método `mover()` é o que está sofrendo `polimorfismo`. Perceba que, teoricamente, o método foi implementado na classe `Animal`. Porém, existem situações em que queremos métodos "vazios" apenas para serem herdados. O método `mover()` pode se encaixar nisso. De repente seria melhor apenas declarar o método na `superclasse` para que seja herdado e implementado somente nas `subclasses`. Isso é possível? **SIM**, através dos `métodos abstratos`.

Para declarar um `método` como abstrato, basta escrever a palavra reservada `abstract` antes e finalizar com um `;`, ou seja, o método declarado não tem um `corpo`. Entretanto, como um `método abstrato` é um método propositadamente vazio e feito para ser herdado e nem possui corpo, um possível `objeto` dessa `classe` teria problemas de funcionamento. Por causa disso, **toda classe que possui algum método abstrato, deve ser também declarada como abstrada**.

A partir disso temos a seguinte definição:

> Uma `classe abstrata` é uma classe que é declarada como abstrata --- ela pode, ou não, incluir `métodos abstratos`. `Classes abstratas` **não podem ser instanciadas**, mas seu conteúdo pode ser herdado.

Vejamos os exemplos de código.

Exercícios

1. Escreva um programa em Java com uma classe abstrata chamada `Animal`, com um método abstrato chamado `som()`. Crie duas subclasses de Animal, uma chamada `Leao` e outra chamada `Cavalo`, e implemente o método `som()` para que cada animal faça seu som específico.

2. Crie uma classe abstrata chamada `Forma`, a qual possui dois métodos abstratos: `calculaArea()` e `calculaPerimentro()`. Depois crie as classes `Círculo` e `Triangulo` as quais serão subclasses de `Forma`. Implemente os métodos para calcular a área e o perímetro de cada forma.
3. Crie uma classe abstrata chamada `ContaBancaria` com os métodos abstratos `depositar()` e `sacar()`. Crie as subclasses `ContaPoupanca` e `ContaCorrente`, as quais devem implementar os métodos respectivos para depósito e saque.
4. Acrescente à classe abstrata `Animal` os métodos `comer()` e `dormir()`. Implemente esses novos métodos nas subclasses já existentes e crie outra subclasse chamada `Vaca`, a qual deve implementar todos os métodos declarados como abstratos em `Animal`.
5. Crie uma classe abstrata chamada `Empregado` com os métodos abstratos `calcularSalario()` e `imprimirInformacoes()`. Crie as as subclasses `Gerente` e `Programador` as quais devem implementar os respectivos métodos para calcular salário e mostrar as informações de cada um.

Interfaces

De acordo com a própria `Oracle`:

> Na linguagem de programação Java, uma `interface` é um tipo de referência, similar a uma classe, e pode conter apenas constantes, assinaturas de métodos, métodos *default*, métodos estáticos e tipos aninhados. Os corpos dos métodos existem apenas no caso dos métodos *default* e estáticos. As `interfaces` não podem ser instanciadas --- elas podem apenas ser implementadas por classes ou extendidas por outras interfaces.

De forma mais fácil, uma `interface` é **como se fosse** uma classe **completamente** abstrata. Basicamente serve como um *esqueleto*.

Obs.: `métodos default e estáticos não precisam ser entendidos ainda .`

Exemplo de uma `interface`:

```
public interface OperateCar {
```

```
    // declaração de constantes, se tiver

    // assinatura dos métodos

    int dobrar(Direcao direcao, double raio, double velocidadeInicio, double velocid

    int trocarFaixa(Direcao direcao, double raio, double velocidadeInicio, double ve

    int dobrarNoSinal(Direcao direcao, boolean sinalVerde);

    int getRadarFrente(double distanceParaOCarro, double velocidadeDoCarro);

    int getRadarTraseira(double distanceParaOCarro, double velocidadeDoCarro);
          ......
    // more method signatures
}
```

Para "usar" uma `interface`, basta criar uma classe que a `implementa`. Quando uma classe **instanciável** implementa uma `interface`, ela provê um bloco/corpo para cada um dos métodos declarados na `interface`. Por exemplo:

```
public class OperateBMW760i implements OperateCar {

    public int dobrarNoSinal(Direcao direcao, boolean sinalVerde) {
        // código para a BMW dobrar à esquerda e ligar a seta
       // código para a BMW dobrar à esquerda e desligar a seta
       // código para a BMW dobrar à direita e ligar a seta
       // código para a BMW dobrar à direita e desligar a seta
    }
}
```

No exemplo do carro robótico acima, as fabricantes do automóvel é que irão **implementar** as `interface`s. A implementação da Chevrolet será substancialmente diferente das implementações da Toyota, por exemplo, porém ambas as fabricantes utilizarão a mesma `interface`. Os fabricantes da parte de orientação dos veículos, os quais são clientes da `interface`, construirão sistemas que usam os dados de GPS da localização do carro, mapas digitais e dados de tráfego para dirigir o carro. Ao fazer essas coisas, os sistemas de orientação invocarão os métodos da `interface`: `dobrar`, `trocarFaixa`, `frear`, `acelerar`, etc.

Exercícios

1. Crie uma `interface` chamada `Forma` com um método `getArea()`. Crie três classes `Retangulo`, `Circulo` e `Triangulo`, e implemente a `interface Forma`. Implemente o método `getArea()` para cada uma das três classes.
2. Crie uma `interface` chamada `Animal` com um método chamado `latir()` que não requer qualquer argumento e retorna `void`. Crie uma classe `Cachorro` que implementa `Animal` e sobrescreve `latir()` para imprimir "O cachorro está latindo".
3. Crie uma `interface` chamada `Voavel` com um método chamado `voaObj()`. Crie três classes `EspacoNave`, `Aviao` e `Helicoptero` que implementam a `interface Voavel`. Implemente o método `voaObj()` para cada uma das três classes.
4. Crie um programa em Java para criar um sistema bancário com quatro classes: `Banco`, `ContaBancaria`, `ContaPoupanca` e `ContaCorrente`. O banco deve ter uma lista de contas e métodos para adicioná-los. `ContaBancaria` deve ser uma `interface` com métodos para depositar, sacar, calcular os juros e ver o balanço. `ContaPoupanca` e `ContaCorrente` devem implementar a `interface ContaBancaria` e terem seus métodos implementados de forma única.
5. Cria uma `interface` chamada `Redimensionavel` com os métodos `redimensionarLargura(int largura)` e `redimensionarAltura(int altura)` que permitem a um objeto ser redimensionado. Crie uma classe `Retangulo` que implementa a `interface Redimensionavel` e implemente seus métodos.