# Programação Orientada a Objetos

Aula 6 – Classe abstrata e Interface

Profa. Ana Patrícia F. Magalhães Mascarenhas
anapatriciamagalhaes@gmail.com

apmagalhaes@uneb.br

# Plano de Aula

- Objetivo
  - Explicar as características de uma classe abstrata
  - Analisar contextos para decidir sobre o uso ou não de classes abstratas
  - Construir classes abstratas
  - Discutir o uso e as vantagens da utilização de interfaces e seu relacionamento com o conceito de abstração
  - Projetar soluções utilizando interfaces
  - Construir interfaces e classes que a realizem

- Bibliografia básica
  - SEPE, Adriano e Roque Maitino Neto**. Programação orientada a objetos.** Londrina: Editora e Distribuidora Educacional AS, 2017. 176p.

# Classes Abstratas

São classes compostas por implementações genéricas e especificações de procedimentos. Como não são totalmente implementadas, não produzem instâncias, ou seja, não podemos criar objetos dessas classes. Elas agrupam características e comportamentos que serão herdados por outras classes e fornecem padrões de comportamento que serão implementados nas subclasses.

A classe passa a servir como modelo genérico para as classes que serão dela derivadas.

Para se definir classes abstratas usa-se a palavra chave **abstract.**

# Métodos abstratos

Um método abstrato será apenas o molde de uma implementação a ser provida pelas classes filhas concretas.

Cada classe filha poderá prover sua implementação de uma forma particular.

Métodos abstratos só podem existir em classes abstratos.

Métodos que não são abstratos podem ser usados normalmente pelos objetos das classes que derivam da classe abstrata.

As classes que derivarem de um classe abstrata devem obrigatoriamente implementar todos os métodos abstratos definidos na classe Pai.

# Exemplo

```
abstract class Figura
{   int x;   // coordenada x
    int y;   // coordenada y

    public Figura (int x1, int y1)
    {   x = x1;
        y = y1;
    }
    public abstract void desenha();
    public abstract void apaga();

    public void move (int x1, int y1)
    {   apaga();
        x = x1;
        y = y1;
        desenha();
    }
}
```

Definição da classe abstrata, através da palavra **abstract**.

A classe pode ter um construtor, mesmo que não possa instanciar objetos.

Declaração dos métodos abstratos. Esses métodos devem obrigatoriamente ser implementados nas classes derivadas.

Declaração de um método normal que poderá ser utilizado pelos objetos de classes derivadas.

# Exemplo (cont.)

```
class Quadrado extends Figura
{   public Quadrado(int x1, int y1)
    {    super(x1, y1);
    }

    public void desenha()
    {  System.out.println("Desenhando quadrado (" + x + "," + y + ")");
    }
    public void apaga()
    {  System.out.println("Apagando quadrado (" + x + "," + y + ")");
    }
}
```

Utilização do construtor da classe Pai.

Implementação obrigatória dos métodos definidos na classe abstrata.

Classe para testar o exemplo.

```
class TesteAbstract
{ public static void main (String args[])
  {   Quadrado q = new Quadrado(10,10);
      q.desenha();
      q.move(50,50);
      q.apaga();
   }
}
```

# Exercícios

Uma seguradora vende produtos para seus clientes. Os produtos vendidos pela seguradora podem ser: seguro de vida, residencial ou de automóvel. Os seguros de vida possuem o nome do beneficiário, valor da apólice e idade do segurado. O seguro residencial possui o nome do beneficiário, valor da apólice, endereco do imóvel e ano de construção do mesmo. O seguro de automóveis possui o nome do beneficiário, valor da apólice, o numero do chassi do carro e o ano de fabricação. Sabe-se que o valor do premio de um seguro de vida corresponde ao valor da apólice acrescido de 10% caso o segurado venha a falecer com menos que 50 anos; Sabe-se que o valor do premio no seguro residencial é o valor da apólice menos a depreciação do imóvel, que corresponde a 0.2 % do valor da apólice por ano de construção. Sabe-se que o valor do premio no seguro do automóvel corresponde a 90% do valor da apólice menos 2% de depreciação por ano de fabricação do automóvel.

Para o cenário descrito acima, apresente o esquema de classes e implemente-as em Java
A classe produto deverá conter um método abstrato calcula, que será responsável por calcular o valor do prêmio de um seguro.
Construir também métodos toString para todas as classes.

# Interface

Classes abstratas podem conter métodos abstratos e não abstratos

Se uma classe abstrata tiver APENAS métodos abstratos, podemos criá-la como uma Interface

Interface não pode ser instanciada

Todos os métodos são implicitamente abstract e public

Atributos serão implicitamente static e final e devem ser inicializados na declaração

# Interface x Herança

Uma classe filha pode herdar de apenas uma classe (abstrata ou não), ou seja, não é permitida herança múltipla.

Uma classe pode implementar várias interfaces

Interfaces são usadas também para implementar uma biblioteca de constantes

# Exemplo (1)

```
 interface objetoGeometrico

{   double calculaArea();
    double calculaPerimetro();
    Ponto2D centro();
}
```

Para criar uma interface usar a palavra interface, NÃO usa a palavra class

Os métodos não precisam de modificador pois são public  e abstract

Interfaces NÃO tem construtores

Interface pode usar outras classes como tipo de dado

# Exemplo (2)

```
Class circulo implements ObjetoGeometrico
{   private Ponto2D centro;
    private double raio;

    public Circulo (Ponto2D centro, double raio){
      setCentro (centro);
      setRaio(raio);
}
    public  Ponto2D centro(){
       return getCentro;
}
    public double calculaArea(){
     return 3.14*getRaio()*getRaio();
}
    public double calculaPerimetro(){
    return 2*3.14*getRaio();
}
}
```

# Sobre a classe que implementa a interface

A relação da interface com a classe que usa a interface é definida com palavra **implements**

A classe deve implementar todos os métodos da interface

Os métodos devem ter o modificador explicitamente. NÃO é possível declarar o método como private.

# Herança múltipla usando interface

**Interface Exemplo1{**
**....**
**}**
**Class exemplo2{**
**....**
**}**

**Pode-se fazer herança multipla de duas formas:**

**Class CirculoEscalavel implements ObjetoGeometrico,Escalavel{**
**}**

Ou

Class exemplo3 extends exemplo2 implements exemplo1

# Exercícios

Considere uma biblioteca. Os livros da biblioteca devem ser identificados por um titulo, autor, numero de paginas, ano de edição e se o livro esta emprestado. Criar uma classe livro com esta especificação e com um método toString que retorne todos os dados de um livro.

Sabe-se que o número máximo de dias para emprestimo de um livro é de 14 dias. Criar uma interface (ItemDeBibliotece) para representar este limite e definir as principais operacoes da biblioteca: emprestimo e devolucao

Criar uma classe Movimentacao que herda de livro e implementa os métodos de ItemDeBiblioteca.

# Pacotes

Muitas aplicações são formadas por diversas classes

É importante agrupar estas classes para organizá-las de acordo com uma relação específica.

Ex. Um sistema pode usar classes de outros sistemas (reutilização)

# Criação de Pacotes

Pacotes requerem que as classes que o compõe sejam armazenadas em um mesmo diretório.

Pode-se criar um pacote criando-se um diretório e armazenando nele todos os códigos fontes das classes que serão agrupadas.

As classes pertencentes a um mesmo pacote deve ter no início de seu código a declaração **package** seguida do nome do diretório (ou pacote).

ex. package Academico

# Exemplo de Pacotes

```
package Calendario;

public class Data{
private byte dia, mês;
private short ano;

public Data(){
}
... gets e sets ....

}
```

```
package Calendario;

public class Hora{
private byte hora, min, seg;

public Hora(){
}
... gets e sets ....

}
```

# Criação de Pacotes

Pode-se criar um pacote com o mesmo nome de uma das classes

Classes que estão no mesmo pacote podem acessar umas as outras através de composição (declaração de variáveis) sem a necessidade de nenhuma declaração adicional

Classes que não fazem parte do pacote podem acessar a classe de outro pacote caso importe este pacote explicitamente a partir da palavra reservada import.

Neste caso apenas os métodos públicos podem ser acessados da classe que a importa.

# Utilizando o pacote criado

```
Import Calendario.*;

Class DemoCalendario{

Public static void main (String[] args){

Data d1= new Data();
}

}
```

# Exercícios

Desenvolver o modelo de classes do trabalho semestral e implementar as classes em Java