

### Aula: 01

Assunto: Classes Abstratas, Interfaces, Polimorfismo, Anotações

## Conceitos Básicos

### Classe Abstrata

- Não pode ser instanciada
- Pode ou não ter métodos concretos (com implementação)
- Pode ou não ter métodos abstratos
- Um método abstrato é aquele que é declarado como abstrato e não possui implementação, isto é, o corpo do método está ausente (method body)

```
public abstract double salario();
```

```
public abstract class Empregado{  
    private String nome;  
  
    public Empregado(String nome){  
        this.nome = nome;  
    }  
  
    public String getNome(){  
        return nome;  
    }  
  
    public void setNome(String nome){  
        this.nome = nome;  
    }  
  
    public abstract double salario();  
}
```

### Interfaces

- É parecida com uma classe abstrata, mas não pode ter métodos concretos, isto é, com implementação, a não ser que sejam default ou static.

- Não é necessário usar a palavra `abstract` nos métodos sem implementação.
- Mas é necessário usar `default` ou `static` nos métodos com implementação; `static` para métodos utilitários invocados sem instanciação e `default` para os métodos de instância. Os métodos `default` podem ser sobrescritos e os `static`, escondidos.
- O uso de `public` é opcional, pois os métodos são sempre implicitamente `public` em uma interface.

```
public interface Bonus {
    static double valor = 1000.0;

    double bonus();

    default double imposto(double aliquota) {
        return aliquota * bonus();
    }

    static double limite() {
        return valor;
    }
}
```

### Como usar Interfaces e Classes Abstratas

- Uma classe concreta que estende uma classe abstrata precisa fazer a sobreposição dos métodos abstratos herdados.
- Uma classe concreta que implementa uma interface precisa fazer a sobreposição dos métodos abstratos herdados.

```
public class Gerente extends Empregado implements Bonus {

    private double salario;

    public Gerente(String nome, double salario) {
        super(nome);
        this.salario = salario;
    }

    public void setSalario(double salario) {
        this.salario = salario;
    }

    @Override
    public double bonus() {
        return salario * 0.5;
    }

    @Override
    public double salario() {
        return salario;
    }
}
```

## Quando usar classe abstrata e quando usar interface?

### Classe abstrata:

- Para compartilhar código entre classes bem parecidas

### Interface

- As classes que irão implementar a interface não tem muito uma a ver com a outra  
Você quer implementar um tipo de comportamento independente de quem for adota-lo
- Classes que já são subclasses de outra precisam também se comportar como a interface (herança múltipla)

## Exemplo de uso de código polimórfico usando classes abstratas e interfaces:

### Classe base - não sabe calcular o salário

```
package folha;
public abstract class Empregado{
    private String nome;

    public Empregado(String nome){
        this.nome = nome;
    }

    public String getNome(){
        return nome;
    }

    public void setNome(String nome){
        this.nome = nome;
    }

    public abstract double salario();
}
```

### Classe que estende a classe base e implementa o cálculo de salário para um mensalista

```
package folha;
public class Mensalista extends Empregado{
```

```

private double salario;

public Mensalista(String nome, double salario){
    super(nome);
    this.salario = salario;
}

@Override
public double salario(){
    return this.salario;
}
}

```

### Classe que estende a classe base e implementa o cálculo de salário para um comissionado

```

package folha;
public class Comissionado extends Mensalista{
    private double comissao;

    public Comissionado(String nome, double salario, double comissao){
        super(nome, salario);
        this.comissao = comissao;
    }

    @Override
    public double salario(){
        return super.salario()+comissao;
    }
}

```

### Interface que define o cálculo do bônus

```

package folha;

public interface Bonus {
    static double valor = 1000.0;

    double bonus();

    default double imposto(double aliquota) {
        return aliquota * bonus();
    }

    static double limite() {
        return valor;
    }
}

```

### Classe que estende a classe base e implementa a interface, isto é, define como se calcula o salário e o bônus de um gerente

```

package folha;
public class Gerente extends Empregado implements Bonus {

    private double salario;

    public Gerente(String nome, double salario) {

```

```

        super(nome);
        this.salario = salario;
    }

    public void setSalario(double salario) {
        this.salario = salario;
    }

    @Override
    public double bonus() {
        return salario * 0.5;
    }

```

**Classe que estende a classe base e implementa a interface, isto é, define como se calcula o salário e o bônus de um diretor**

```

    @Override
    public double salario() {
        return salario;
    }
}
package folha;
public class Diretor extends Mensalista implements Bonus {

    public Diretor(String nome, double salario) {
        super(nome, salario);
    }

    @Override
    public double bonus() {
        return salario() * 0.1;
    }
}

```

**Teste. Note que todos estão sendo instanciados em um arraylist do tipo Empregado. O operador instanceof garante que o método bonus não seja chamado em classes que não tem o método bonus().**

```

package folha;
import java.util.ArrayList;

public class TesteEmpregado {
    public static void main(String[] args) {
        ArrayList<Empregado> empregados = new ArrayList<>();
        empregados.add(new Mensalista("Jose Pereira", 3500.00));
        empregados.add(new Gerente("Roberto Almeida", 7000.00));
        empregados.add(new Diretor("Joao Silva", 25000.00));
        empregados.add(new Comissionado("Maria Pereira", 1500.00, 5000.00));

        for(Empregado emp:empregados){

```

```
        System.out.println(emp.getNome());
        System.out.println(emp.salario());
        if(emp instanceof Bonus){
            System.out.println(((Bonus)emp).bonus());
        }
    }
}
```

**Resultado: apesar de serem do tipo empregado, todos sabem calcular seu salário e bonus (aqueles que o tem) corretamente dependendo do seu tipo.**

```
Jose Pereira
3500.0
Roberto Almeida
7000.0
3500.0
Joao Silva
25000.0
2500.0
Maria Pereira
6500.0
```

## Anotações

- São um tipo de metadado, fornecendo dados sobre o programa que não são parte do programa propriamente dito.

Usos comuns:

- Informações para o compilador
- Processamento em tempo de compilação e de implantação
- Processamento em tempo de execução
- Usados intensivamente em frameworks, como Spring e Hibernate

### Algumas anotações básicas

**@Override**

o compilador verifica se o método está sobrecarregando um método da superclasse

**@SuppressWarnings**

desliga os avisos do compilador (e do Eclipse)

**@WebServlet**

informa o endereço de um servlet

**@WebFilter**

informa qual servlet será filtrado por um filtro

### Problemas Propostos:

1. Crie a seguinte hierarquia de classes de figuras geométricas. Veja na figura as fórmulas:
  - a. A classe abstrata Figura deve ter o método abstrato `area`.
  - b. A classe concreta `Circulo` é subclasse de `Figura`.
  - c. A classe abstrata `Poligono` é subclasse de `Figura` e deve ter os atributos `base` e `altura`.
  - d. As classes concretas `Triangulo`, `Losango`, `Retangulo` e `Quadrado` são subclasses de `Poligono`. Tente criar mais uma generalização aqui olhando as fórmulas da área.
  - e. Os polígonos `Retangulo` e `Quadrado` devem implementar a interface `Diagonal`, que deve ter um método que calcula a diagonal.
  - f. Crie uma classe `Geometria` com um `ArrayList` de Figuras com pelo menos uma figura de cada e imprima suas áreas, perímetros e diagonais.

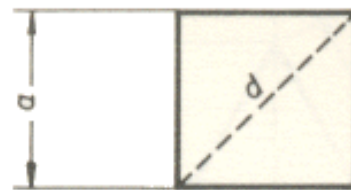
Obs: use a anotação `@Override` nos métodos sobrepostos (ou sobrescritos).
2. Implemente agora o método abstrato `perimetro` na classe `Figura`. Como consequência, você terá que implementá-lo em todas as classes concretas.
3. Crie a classe concreta `Trapezio`. De quem ela deve ser subclasse?
4. Altere a sua herarquia de classes para agora calcular também o volume. Crie as classes `Cubo`, `Esfera`, `Cilindro` e `Piramide`. É melhor resolver por herança ou por composição?

$$A = a^2$$

$$a = \sqrt{A}$$

$$d = a\sqrt{2}$$

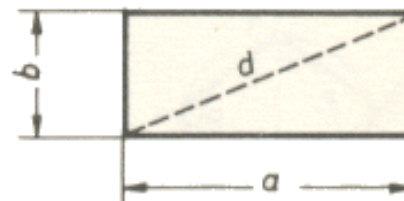
Quadrado

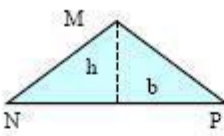
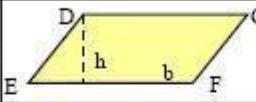
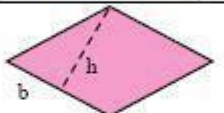
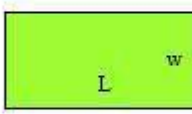
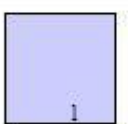
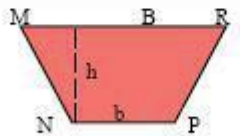
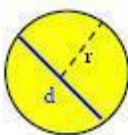


Retângulo

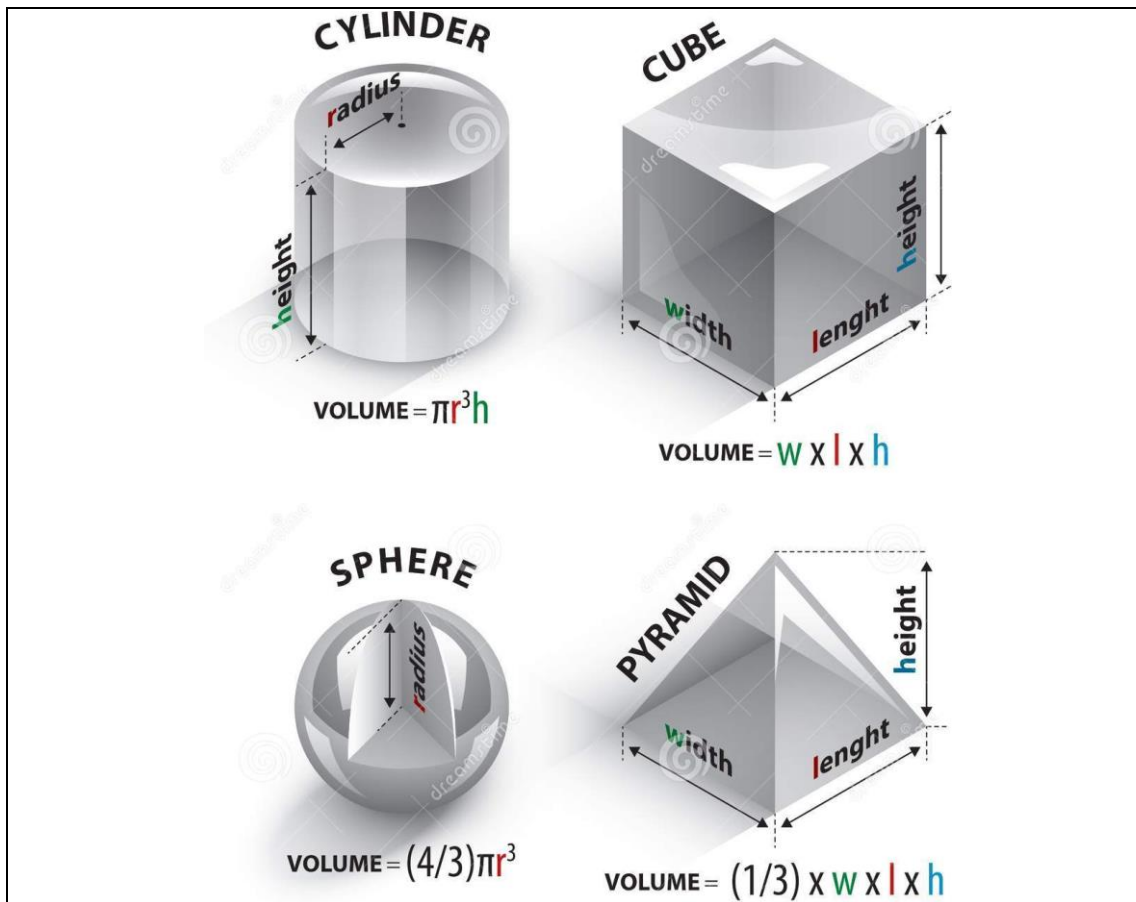
$$A = a b$$

$$d = \sqrt{a^2 + b^2}$$



NAME	FIGURE	AREA	PERIMETER CIRCUMFERENCE
TRIANGLE		$A = \frac{b \times h}{2}$	$P = MN + NP + PM$
PARALLELOGRAM		$A = b \times h$	$P = DE + EF + FG + GD$
RHOMBUS		$A = b \times h$	$P = b + b + b + b$ $P = 4b$
RECTANGLE		$A = L \times w$	$P = L + w + L + w$ $P = 2L + 2w$
SQUARE		$A = l^2$	$P = l + l + l + l$ $P = 4l$
TRAPEZOID		$A = \frac{(B + b) \times h}{2}$	$P = MN + NP + PR + RM$
CIRCLE		$A = \pi r^2$	$C = 2\pi r = \pi d$





### Bibliografia

DEITEL, P. DEITEL, H. Java: como programar. 8 Ed. São Paulo: Prentice – Hall (Pearson), 2010.