7 – Classes Abstractas e Interfaces

Classe Abstracta

– Classe em que pelo menos um dos métodos de instância não é implementado.

Exemplo:

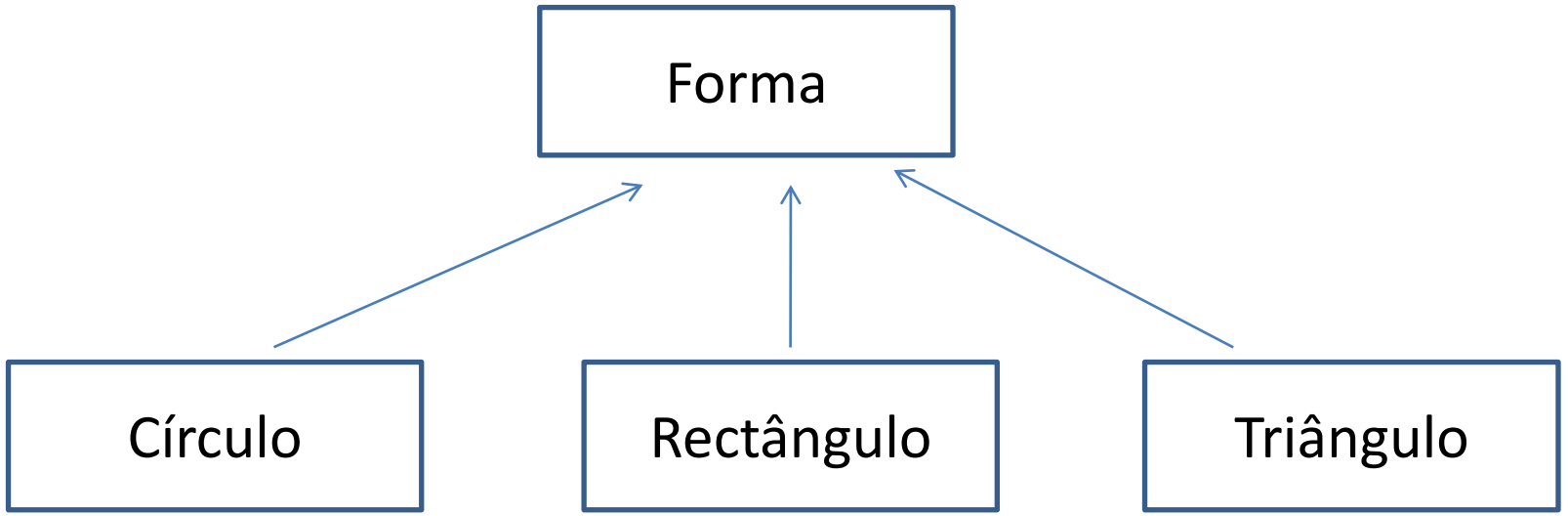
```
public abstract class Forma{
        public abstract double area();
        public abstract double perimetro();
}
```

- Não é possível criar instâncias de uma classe abstracta;
- Mecanismo de herança mantém-se;
- Princípio da substitutividade mantém-se;
- Se uma subclasse de uma classe abstracta implementar todos os métodos, passará a ser uma classe concreta (não abstracta).

Para que servem?

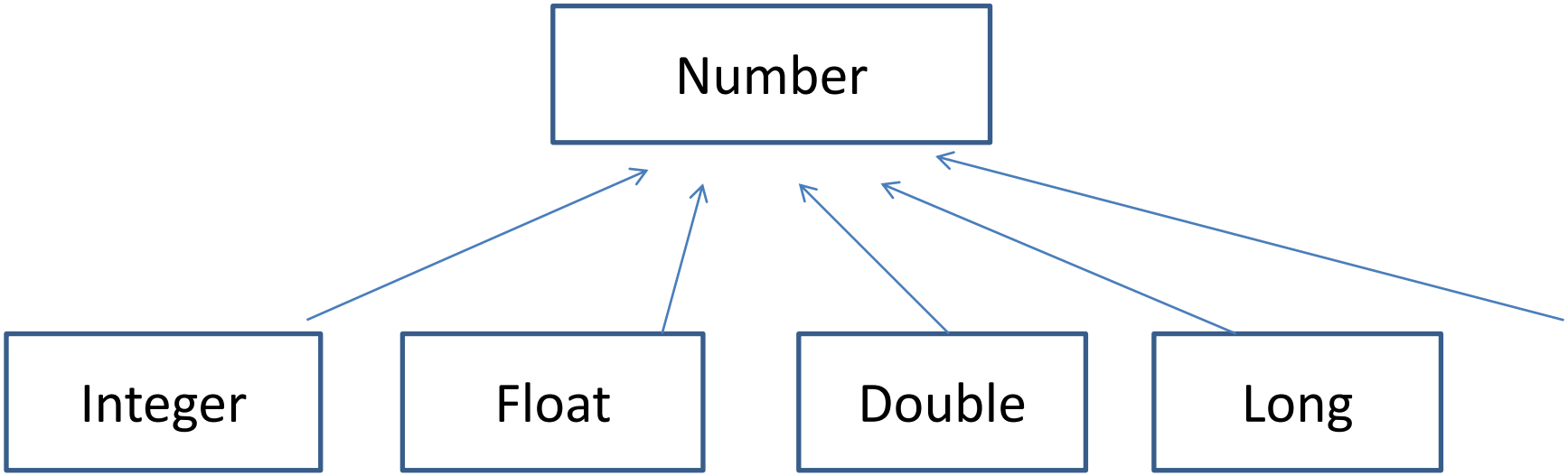
Definir uma linguagem comum a um conjunto de classes que herdam a classe abstracta.

Exemplos:




```
public  class Circulo extends Forma {
 ...
}
...
```

Exemplos:



Notas:

- *Variáveis não são abstractas;*
- *Construtores não são abstractos;*
- *Métodos de classe não são abstractos;*
- *Métodos privados não são abstractos.*

Interfaces (em Java)

"Interface":

- especificação sintáctica de um conjunto de métodos e constantes

Permite definir um comportamento comum a duas ou mais classes que não possuam qualquer relação hierárquica entre si

Declaração de uma interface (exemplo):

```
public abstract interface Ordem{
        public abstract boolean igual (Ordem elemento);
        public abstract boolean maior (Ordem elemento);
        public abstract boolean menor (Ordem elemento);
}
```

Uma interface é (implícita e) obrigatoriamente abstracta.

Os métodos declarados numa interface são (implícita e) obrigatoriamente públicos e abstractos.

Uma classe que implemente uma dada interface tem obrigatoriamente que implementar todos os métodos declarados na interface.

```
public class MyInteger implements Ordem{
        ...
                public boolean igual (Ordem e){…}
                public boolean maior (Ordem e){…}
                public boolean menor (Ordem e){…}
// outros métodos
}
```

Todas as classes que implementam a interface Ordem têm em comum o comportamento definido em Ordem.

As interfaces têm a sua própria hierarquia:

```
public interface Amovivel {
        public abstract void movimento ( double x, double y);
}

public interface ComMotor extends Amovivel

        public static final int limiteVel = 120;
        public abstract String motor();
}
```

As constantes declaradas numa interface são implícita e obrigatoriamente: *public static final*

-Uma classe que implemente a interface ComMotor terá obrigatoriamente que implementar:

- todos os métodos da interface e
- todos os métodos de todas as super interfaces

```
public class Veiculo implements ComMotor {
  …
  public String motor () {…}

  public void movimento (double x, double y){ …}

  …
}
```

Uma interface pode ser sub-interface de várias interfaces:

```
public interface Transformavel extends
                                    Escalavel,  Rodavel, Desenhavel {
...

}
```

A interface Transformavel herda todas as definições sintácticas das 3 interfaces especificadas.

➔ Mecanismo de herança múltipla

Classes Abstractas versus Interfaces

- uma classe abstracta pode ter métodos implementados

- *numa interface todos os métodos são abstractos*

- uma subclasse de uma classe abstracta pode ser ou não uma classe abstracta

- *numa subinterface todos os métodos são abstractos*

Classes Abstractas versus Interfaces

- uma classe abstracta pode ser usada para escrever software genérico, cada subclasse vai fazendo a sua implementação num processo de especialização sucessiva.

- uma interface serve para especificar um comportamento comum a todas as classes que a implementam.

Exemplo (1) - Classe abstrata

```
public abstract class Conta {
   private double saldo;

   public void setSaldo(double saldo) { this.saldo = saldo; }

   public double getSaldo() { return saldo; }

   public abstract void imprimeExtrato ();
}
```

(1) https://www.devmedia.com.br/polimorfismo-classes-abstratas-e-interfaces-fundamentos-da-poo-em-java/26387

Exemplo (1)

```
import java.texto.SimpleDataFormat;
import java.util.Date;
public class ContaPoupanca extends Conta
{ @Override
  public void imprimeExtrato() {
    System.out.println("### Extrato da Conta ###");
    SimpleDateFormat sdf =
        new SimpleDateFormat("dd/MM/aaaa HH:mm:ss");
    Date date =
        new Date(); System.out.println("Saldo: "+this.getSaldo());
    System.out.println("Data: "+sdf.format(date)); }
 }
```

Exemplo (1)

```
public class TestaConta {

 public static void main(String[] args) {

    Conta cp = new ContaPoupanca();
    cp.setSaldo(2121);
    cp.imprimeExtrato(); } }

}
```

Exemplo (1) interface

```
interface Conta{

    public void depositar(double valor);
   public void sacar(double valor);
   public double getSaldo();

 }
```

Exemplo (1)

```
public class ContaCorrente implements Conta {
 private double saldo;
 private double taxaOperacao = 0.45;
@Override
 public void deposita(double valor) {
      this.saldo += valor - taxaOperacao; }
@Override
public double getSaldo() { return this.saldo; }
@Override
public void sacar(double valor) {
  this.saldo -= valor + taxaOperacao; }
}
```

Exemplo (1)

```
public class ContaPoupanca implements Conta {
   private double saldo;

   @Override
   public void deposita(double valor) { this.saldo += valor; }

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

   @Override
    public void sacar(double valor) { this.saldo -= valor; }
}
```

Exemplo (1)

```
public class GeradorExtratos {

 public void geradorConta(Conta conta){

      System.out.println("Saldo Atual: "+conta.getSaldo());

 }
}
```

Exemplo (1)

```
public class TestaContas {
  public static void main(String[] args) {
    ContaCorrente cc = new ContaCorrente();
    cc.deposita(1200.20);
    cc.sacar(300);

    ContaPoupanca cp = new ContaPoupanca();
    cp.deposita(500.50);
    cp.sacar(25);

    GeradorExtratos gerador = new GeradorExtratos();
    gerador.geradorConta(cc);
    gerador.geradorConta(cp); }
}
```