Metodologia e Linguagem de Programação Orientada a Objetos

Classes Abstratas e Interface

Drª. Alana Morais

Aula Passada

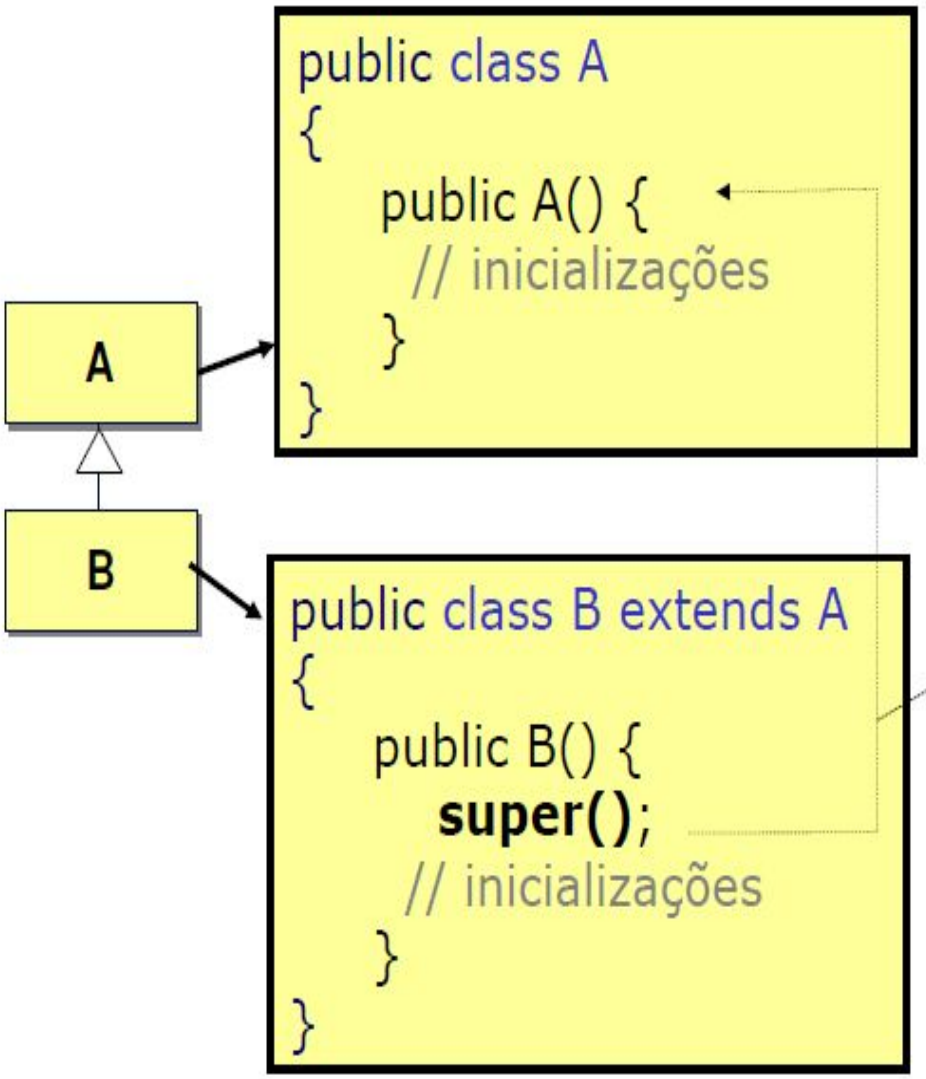
- Listas
  - ArrayList e Array
- Herança

Herança em java

- Herança é um tipo de **relacionamento** entre 2 classes onde:
  - Uma classe herda (compartilha) todas as propriedades e métodos de outra classe.
  - Subclasse herda de superclasse.

- Em Java, a palavra-chave utilizada para criar

```
[Subclasse a ser criada] extends [Superclasse existente]
```



- O construtor da superclasse não é herdado,
- Logo, a subclasse deverá chamá-lo, na **primeira linha** de seu construtor, através da palavra **super**

Roteiro

- Herança
  - Classes Abstratas
  - Métodos Abstratos
- Interface
  - Sintaxe
  - Utilidade da Interface
- Herança múltipla

Classes Abstratas e Herança

Classes Abstratas

- Serve apenas como modelo para uma classe concreta.

- Poderoso Mecanismo de Abstração:
  - Permite a herança do código sem violar a noção de subtipo
  - Diz o que deve ter a subclasse, **mas não diz como**!

- São classes que não podem ser instanciadas.
  - Não usamos o operador “new” para essa classe

Classes Abstratas

- O uso de classe abstrata é necessário:
  - Quando a classe é genérica e contém propriedades e métodos comuns a várias subclasses
  - Quando há pelo menos um *método abstrato:*
    - Métodos Abstratos:
      - Encapsulamento, Indicação de abstração, Retorno, Nome do método, Parâmetros e “;”

```
public abstract void nomeMetodoAbstrato();
```

Classes Abstratas

- A classe abstrata:
  - código genérico, livre de particularidades
  - Serve como guia para a definição do comportamento dos herdeiros, das subclasses.

- As subclasses:
  - Detalhes particulares.
  - Deve, obrigatoriamente, implementar todos os métodos abstratos, assim definidos na superclasse.

Classes Abstratas
Sintaxe

```
public abstract class Contribuinte{

   private String nome;
   private float total_receita;
      .
      .
      .
}
```

Classes Abstratas - Exemplo I

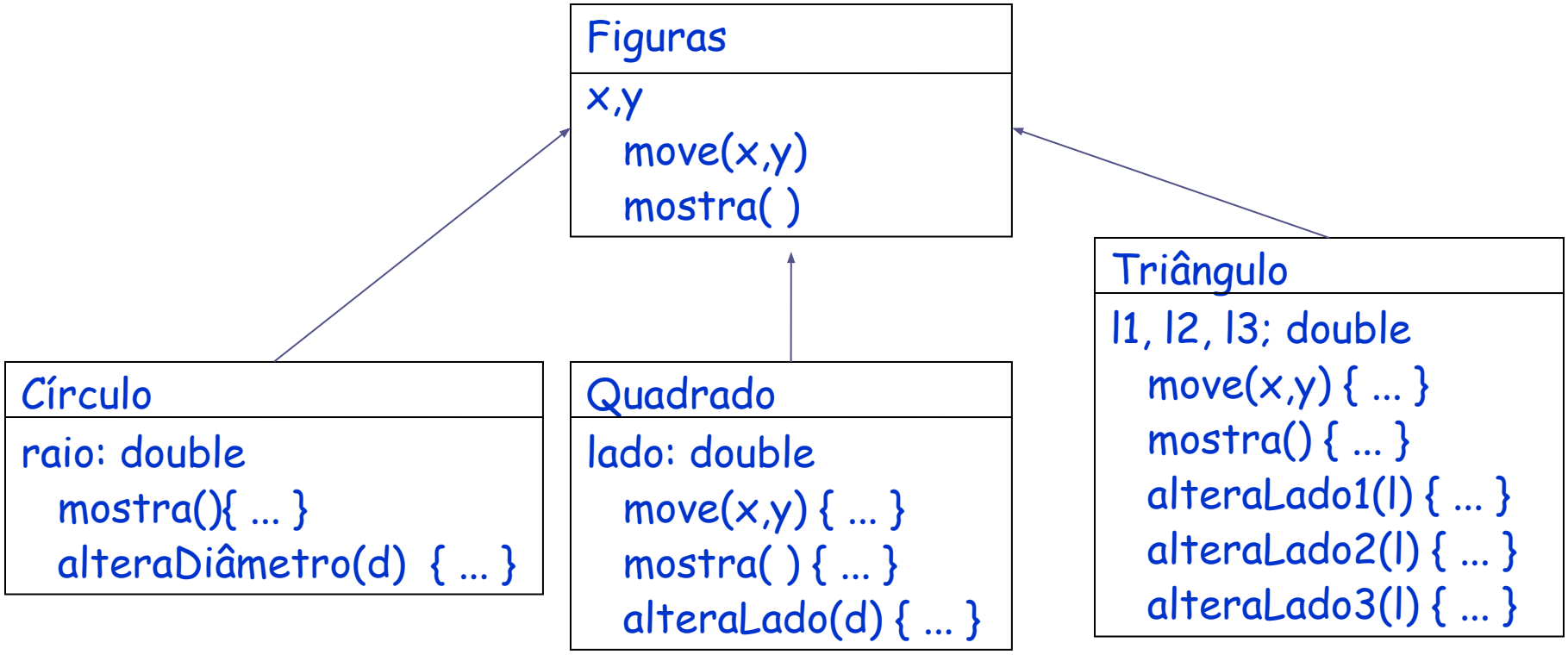
- Exemplo:
  - Círculos, Quadrados e Triângulos.

Círculo
x,y raio move(x,y) { ... } mostra( ) { ... } alteraDiâmetro(d) { ... }

Quadrado
x,y lado move(x,y) { ... } mostra( ) { ... } alteraLado(d) { ... }

Triângulo
x,y l1,l2,l3 move(x,y) mostra() alteraLado1(l) { ... } alteraLado2(l) { ... } alteraLado3(l) { ... }

Classes Abstratas - Exemplo I



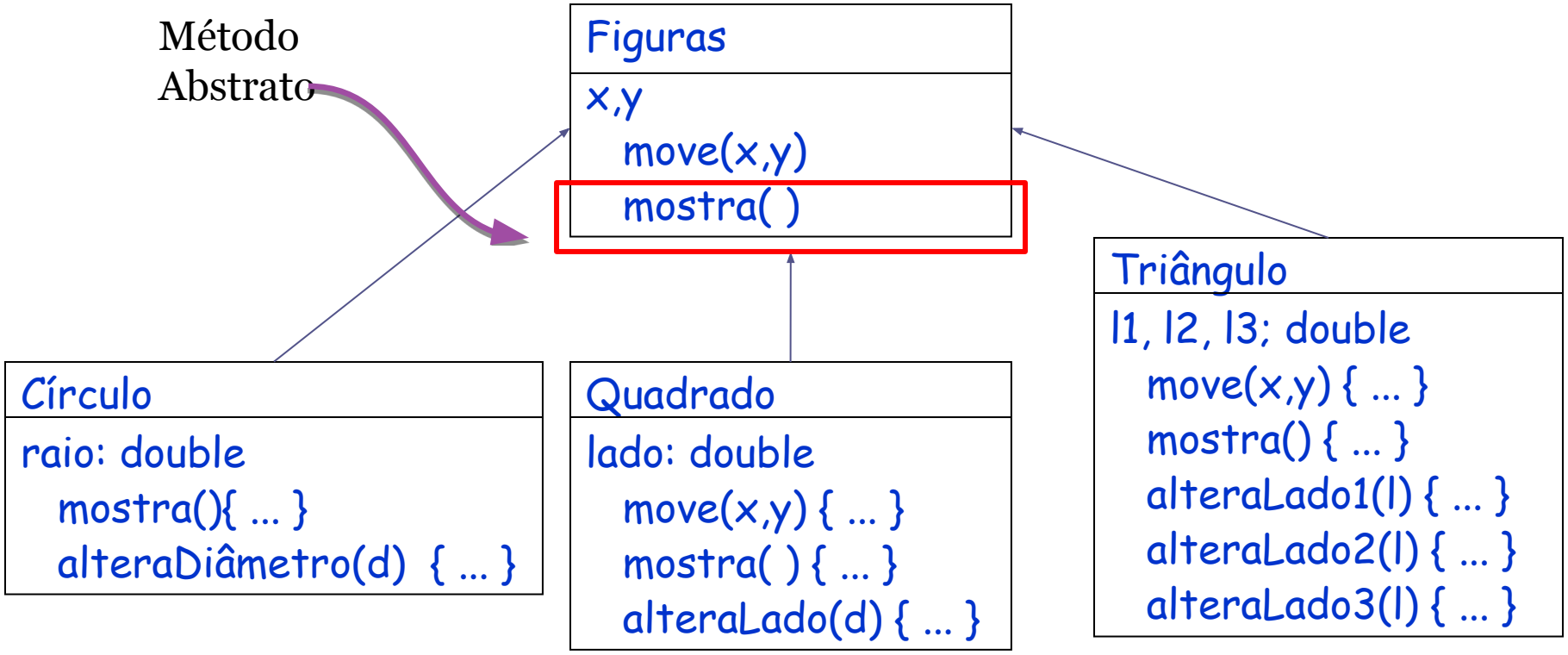
Classes Abstratas - Exemplo I

```
public abstract class Figuras{

        private int x,y;

        public Figuras(int xT, int yT){
            this.x = xT; this.y =yT;
        }

        public String mostra(){
            return String.valueOf(this.x);
        }

        public void move(int x, int y){
            x+=3;
            y+=-10;
        }
}
```

Classes Abstratas - Exemplo I

- Métodos Abstratos
  - O uso de métodos abstratos:
    - Torna uma classe abstrata.
    - Obriga que todas as subclasses concretas o implemente.
    - Permite o Polimorfismo, pois, define um protocolo a ser usado em todas as subclasses

Classes Abstratas - Exemplo I



Classes Abstratas - Exemplo I

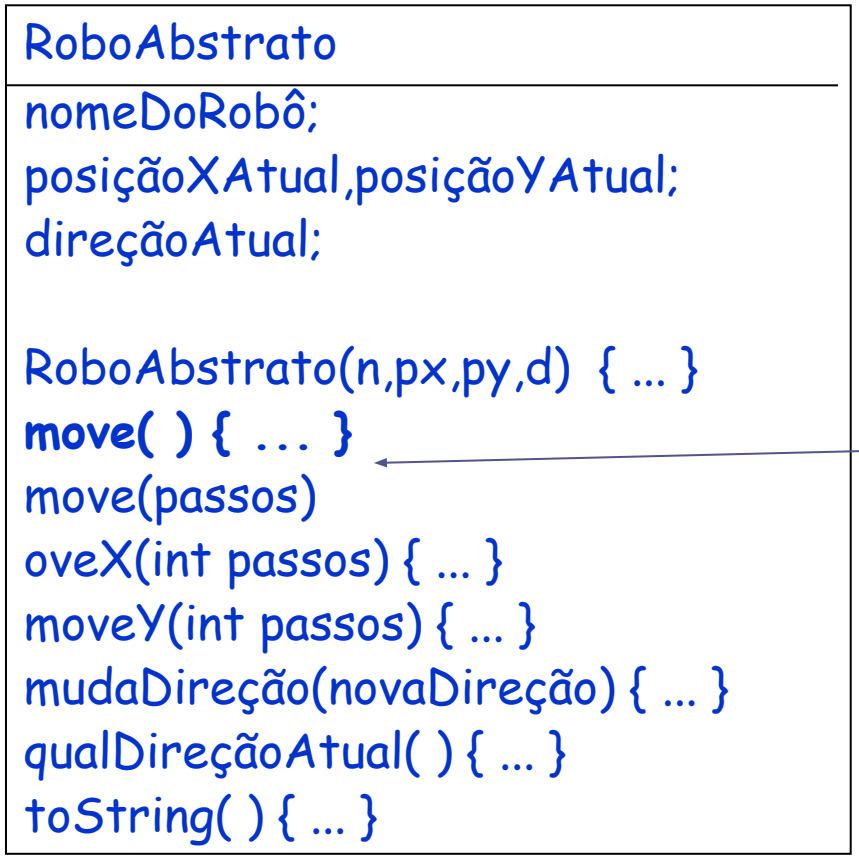
- Agora com métodos abstratos!!

```
public abstract class Figuras{
     private int x,y;

     public abstract void mostra();

     public void mover(int x, int y){ … }
}
```

Classes Abstratas - Exemplo I

```
public class Circulo extends Figuras{
      private double raio;

      public Circulo(int x, int y, double r){
         super(x,y);
         this.raio = r;
      }

      public void mostra(){
         System.out.println
             ( String.valueOf(getX()) +
               String.valueOf(getY()));
      }
}
```

Classes Abstratas - Exemplo II



Método abstrato: não sabemos exatamente como um robô irá implementar seus movimentos.

Classes Abstratas - Exemplo II

```
abstract class RoboAbstrato {
  private String nomeDoRobô;
  private int posiçãoXAtual,posiçãoYAtual;
  private short direçãoAtual;

  public RoboAbstrato(String n,int px,int py,short d)
  { ... }
  public void move()    { move(1); }

  public abstract void move(int passos);
  public void moveX(int passos)  { ... }
  public void moveY(int passos)  { ... }
  public void mudaDireção(short novaDireção) { ... }
  public short qualDireçãoAtual()    { ... }
  public String toString()   { ... }
}
```

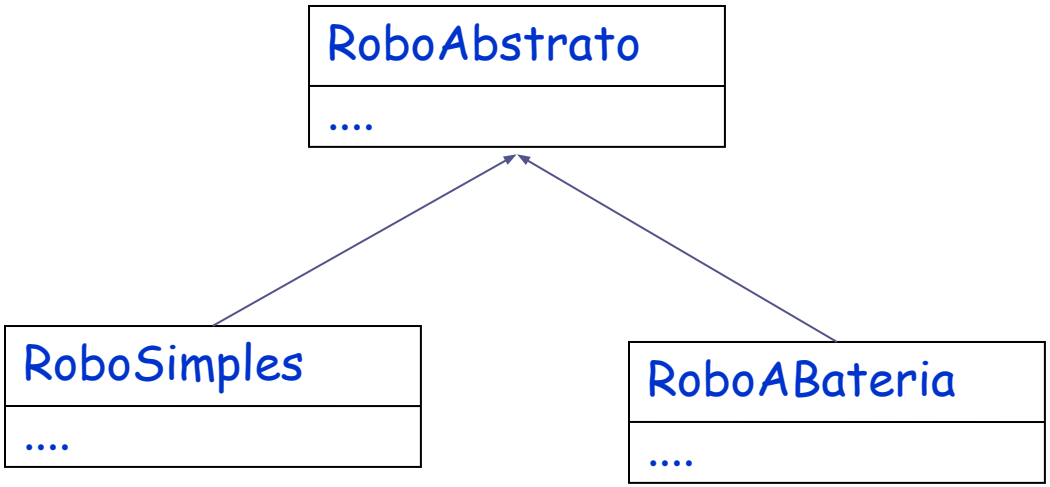
A classe é abstrata, pois possui ao menos um método abstrato.

Não existem "campos abstratos".

Um construtor não pode ser abstrato: seu código é necessário para inicializar corretamente os campos da classe abstrata.

Um método abstrato pode ser chamado a partir de outro: no momento da chamada, a subclasse haverá sobreposto o método abstrato.

Classes Abstratas – Exemplo II



Cada subclasse de RoboAbstrato implementa move(passos) de uma forma particular:

Robô simples apenas atualiza as coordenadas,

Robô a bateria consome energia.

Classes Abstratas – Exemplo II

```
class RoboSimples extends RoboAbstrato {
   RoboSimples(String n,int px,int py,short d) {
     super(n,px,py,d);
    }

   public void move(int passos){
     switch(qualDireçãoAtual()) {
        case   0:   moveX(+passos); break;
        case  90:  moveY(+passos); break;
        case 180: moveX(-passos); break;
        case 270: moveY(-passos); break;
    }
   }
}
```

RobôSimples herda todas as caracerísticas de RobôAbstrato, mas deve implementar move(passos) para que possa ser instanciada.

Classes Abstratas – Exemplo II

```
class RoboABateria extends RoboAbstrato {
      private long energia;
      RoboABateria(String n,int px,int py,short d,long e)  {
           super(n,px,py,d); energia = e;
      }
      public void move(int passos)     {
          long energiaASerGasta = passos*10;
          if (energiaASerGasta <= energia) {
              switch(qualDireçãoAtual()) {
                case   0: moveX(+passos);
                       break;
                ...
                case 315: moveY(-passos);
                        moveX(+passos);
                        break;
             }
                energia -= energiaASerGasta;
        }
}}
```

Classes Abstratas

```
class DemoRobos{
 public static void main(String[]
          argumentos){
    ....
  RoboAbstrato imag = new RoboAbstrato
       ("Imaginário",0,0,(short)180);

  } // fim do método main
} // fim da classe DemoRobos
```

Classes Abstratas

- Exercício de Fixação:
  - Implementar a classe abstrata Guarita, e as subclasses GuaritaDeEntrada e GuaritaDeSaída.
  - A estrutura de classe deve conter um método abstrato mostrarIdentificacao( ) sem retorno
  - Testar as novas classes usando em uma classe de teste chamada Estacionamento.

Interface

Interface

- Conjunto de protocolos que são implementados por uma ou mais classes.
  - Cada classe pode executar de forma diferente este métodos.

- Recurso de orientação a objeto que permite definir ações que devem ser obrigatoriamente executadas nas classes.

Interface

- Uma **interface** é similar a uma **classe abstrata** com algumas diferenças:
  - Não podem ter métodos concretos ou variável de instância.
    - Isso significa que elas não possuem atributos nem implementação de métodos, apenas as assinaturas dos mesmos.
  - Podem ter constantes (inicializadas).

Interface

- Classes Java é capaz de implementar **quantas interfaces** forem necessárias.

- Enquanto que ao utilizar classes abstratas, o programador está restrito ao fato de herdar de apenas **uma** classe ancestral.

Interface

- Regras:
  - A classe que implementa a interface concorda com a implementação de todos os métodos definidos pela interface.
  - Uma definição de interface é uma “coleção de definições de métodos, sem implementação, e de variáveis que são obrigatoriamente constantes”.

Interface
Sintaxe

- O formato para uso de interface é descrito a seguir:

```
[Classe] implements [Interface 1], [Interface 2]
{

    (...)

}
```

Interface

- Declara uma Interface
- Implementa uma Interface

Interface - Declaração

```
public interface Interface1 {
    public tipo atributo=valor;

    public tipo m1(…);

    public tipo m2(…);

}
```

Interface - Implementação

- Implementar uma interface significa implementar numa classe todos os métodos especificados pela interface.
- Em Java, utiliza-se a palavra **implements**

```
public class A implements Interface1 {
   // …atributos de A
   // …métodos de A

   public tipo m1(…) { … };
   public tipo m2(…) { … };
}
```

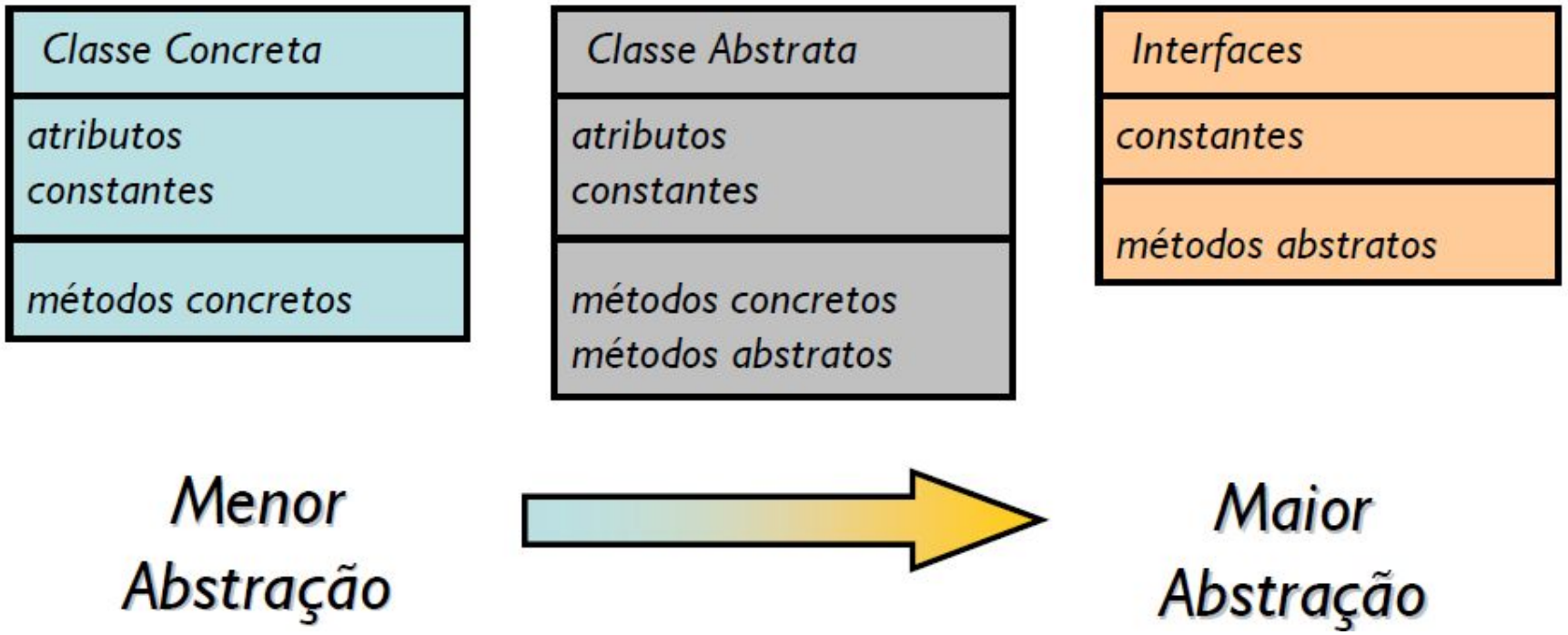
Interface

- Permitem generalizar ao máximo uma classe
  - MAIOR FLEXIBILIDADE
- Representam uma classe totalmente abstrata
  - Não têm variáveis
  - Não têm construtores
  - Não têm métodos implementados
  - Podem ter constantes

Interface

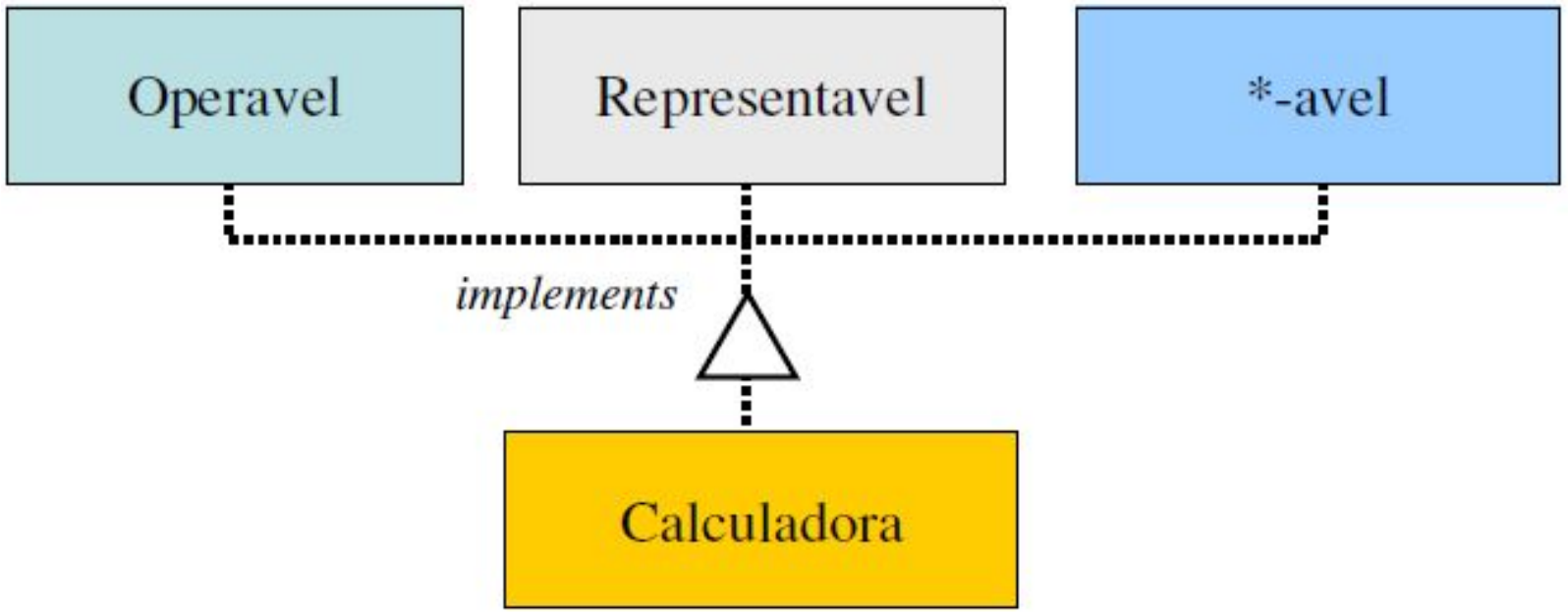
- “Programar para a Interface, não para a implementação”.
  - Maior flexibilidade
  - Implementações mudam, conceitos ficam
  - Programar para o futuro e não para o presente

Interface
Grau de Abstração



Interface - Exemplo

- *Podemos implementar mais de uma interface*



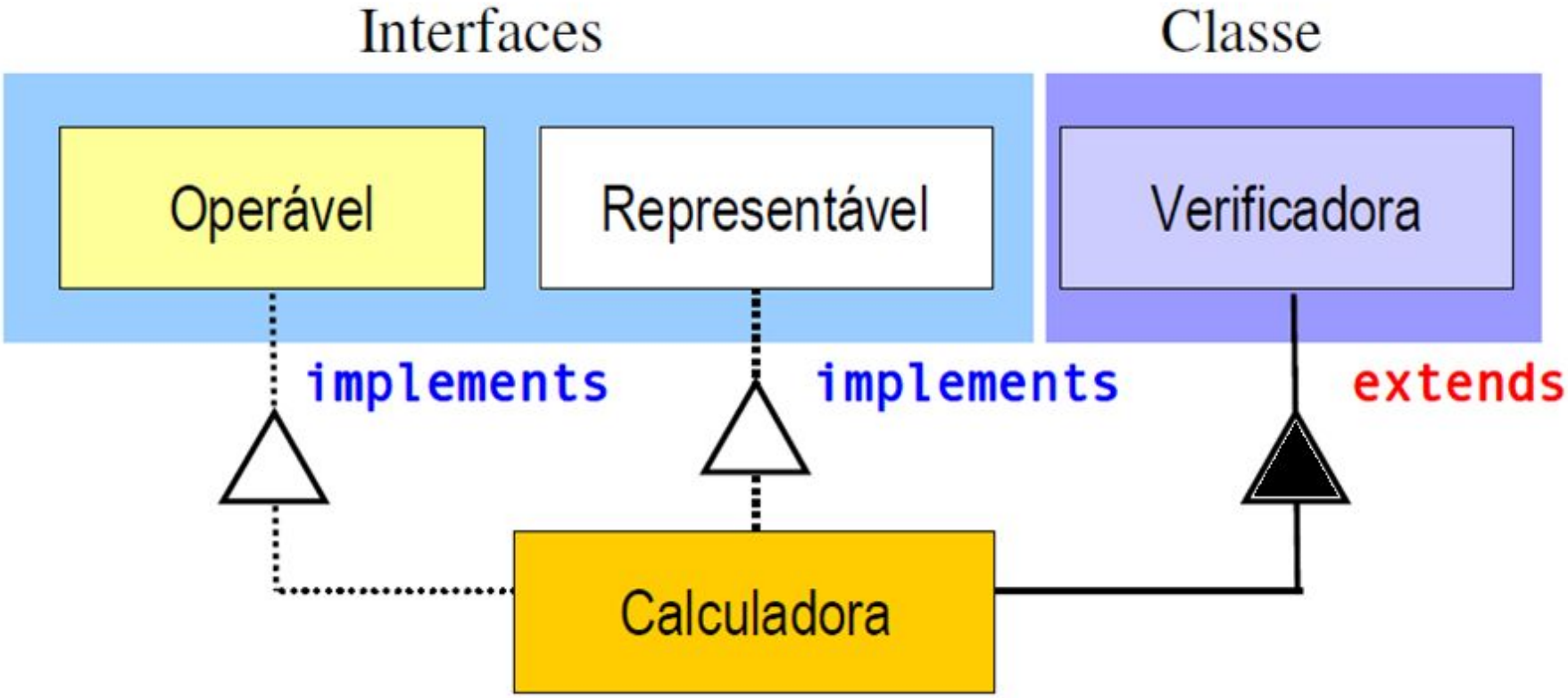
Interface - Exemplo

```
class Calculadora implements Operavel,Representavel
{
      // métodos de Operavel
      public int soma(int a, int b){return a + b;}
      public int subtrai(int a, int b){return a - b;}
      public int multiplica(int a, int b){return a*b;}
      public int divide(int a, int b){return a / b;}

      // método de Representavel
      public void exibeNaTela(int valor){
         System.out.println("O valor é " + valor);
      }
}
```

Herança Múltipla

- A herança múltipla é o fato de uma classe possuir mais de uma superclasse.
- Em outros termos, significa construir uma classe que herde as características de mais de uma classe ao mesmo tempo.
- Linguagens de programação como C++ permitem tal recurso, embora Java não o permita diretamente.

Herança Múltipla - Exemplo



Herança Múltipla - Exemplo

```
class calculadora implements Operavel,Representavel
                    extends Verificadora {

   // métodos de Operavel
   public int soma(int a, int b){return a+b;}
   public int subtrai(int a, int b){return a-b;}
   public int multiplica(int a, int b){return a*b;}
   public int divide(int a, int b){return a/b;}

   // método de Representavel
   public void exibeNaTela(int valor){
       System.out.println("O valor é " + valor);
   }
}
```

Exercício:

Crie a seguinte hierarquia de classes

- Uma interface para representar uma forma geométrica, definindo métodos para cálculo do perímetro e cálculo da área da forma;
- Uma classe abstrata para representar quadriláteros. Seu construtor deve receber os tamanhos dos 4 lados e o método de cálculo do perímetro já pode ser implementado;
- Classes para representar retângulos e quadrados. A primeira deve receber o tamanho da base e da altura no construtor, enquanto a segunda deve receber apenas o tamanho do lado;
- Uma classe para representar um círculo. Seu construtor deve receber o tamanho do raio.
- No programa principal, pergunte ao usuário qual forma ele deseja criar. Em seguida, para cada forma, solicite os dados necessários para criar a forma. Todas as formas criadas devem ser armazenadas em um vetor. Finalmente, imprima: (a) os dados (lados ou raio); (b) os perímetros; e (c) as áreas de todas as formas.

Referências

- Livro Use a cabeça! JAVA
- Material do Professor Fausto IFPB
- Material da Profª Isabel Cafezeiro
  - http://www.dcc.ic.uff.br/~isabel

Dúvidas

alanamm.prof@gmail.com