Criação e Uso de Classes

- Em Java, (quase) tudo é Classe
- Criar cada Classe em arquivo separado
- O método main() deve existir apenas na classe pela qual iniciará a execução do programa
- Classes possuem:
  - Atributos => armazenam dados
  - Métodos => definem comportamento
- Para usar uma classe é necessário instanciá-la, ou seja, declarar e inicializar (ou construir) um objeto com o comando **new**

Criação e Uso de Classes

```
public class Retangulo {
  double base;
  double altura;


  public double calcArea() {
    return base * altura;
  }

}
```

```
public class POO {

  public static void main(String[] arqs) {
    Retangulo r = new Retangulo();
    r.base = 10.3;
    r.altura = 5.9;
    System.out.println("Area Ret. = " + r.calcArea());

    System.out.println("Fim do Programa");
  }

}
```

Encapsulamento

- Não convém que os atributos base e altura da classe possam ser diretamente acessados fora da classe. Para evitar que isso ocorra é que existe o encapsulamento.
- Modificador de acesso: private
  - De `double base;` passa para `private double base;`
  - Porém, a partir daí ocorrerão erros de compilação pois os atributos não estão mais acessíveis fora da classe
- O acesso aos atributos passa a ser feito através de métodos públicos usualmente conhecidos como: getters e setters

Encapsulamento

```
public class Retangulo {
  private double base;
  private double altura;
  public void setBase(double valor) {
    base = valor;
  }
  public double getBase() {
    return base;
  }
  public void setAltura(double valor) {
    altura = valor;
  }
  public double getAltura() {
    return altura;
  }
  public double calcArea() {
    return base * altura;
  }

}
```

```
public class POO {

  public static void main(String[] arqs) {
    Retangulo r = new Retangulo();
    r.setBase (10.3);
    r.setAltura (5.9);
    System.out.println("Area Ret. = " + r.calcArea());

    System.out.println("Fim do Programa");
  }

}
```

Encapsulamento - Benefícios

- Permite usar os atributos sem a necessidade de conhecer os detalhes internos das Classes;
- Alterações nas regras de implementação de uma Classe não afetam seu uso;
- Permite a validação e a consistência do conteúdo no momento da atribuição de valor a um atributo;
- Garante o acesso seguro e consistente ao conteúdo dos atributos;

Exercício

- Implemente as classes Retangulo e POO e teste o programa
- Faça uma alteração nos métodos set???? para que números negativos não sejam aceitos

Construtor de uma Classe

- Construtor é um método público de uso especial que é invocado única e exclusivamente quando um objeto é instanciado
- Ele tem o mesmo nome da Classe
- Pode receber parâmetros ou não
  - Caso receba parâmetros, os mesmos podem ser usados para inicializar atributos
  - Caso não receba parâmetros, o construtor pode inicializar atributos com valores padrão (default)

Construtor de uma Classe

```
public class Retangulo {
  private double base;
  private double altura;

  public Retangulo() {
    setBase(0);
    setAltura(0);
  }

  public Retangulo(double b, double a) {
    setBase(b);
    setAltura(a);
  }

  ... segue a classe
```

Sobrecarga de métodos

- Lembrando: construtores são métodos
- Na tela anterior foram apresentados dois diferentes construtores
  - Um, sem qualquer parâmetro;
  - O outro, com dois parâmetros;
- Isso se chama Sobrecarga de Método
  - Ela consiste em declarar dois ou mais métodos com o mesmo nome, porém com diferentes quantidades e/ou tipos de parâmetros
  - No momento da chamada do método, o interpretador Java decide qual das versões usar em função dos parâmetros que foram passados
  - Na chamada os parâmetros passados, obrigatoriamente, devem coincidir com alguma opção sobrecarregada disponível. Se isso não ocorrer, haverá erro

Exercício

- Crie duas novas Classes semelhantes à Classe Retangulo, Triangulo e Circulo, conforme especificado abaixo e escreva um programa para testá-las

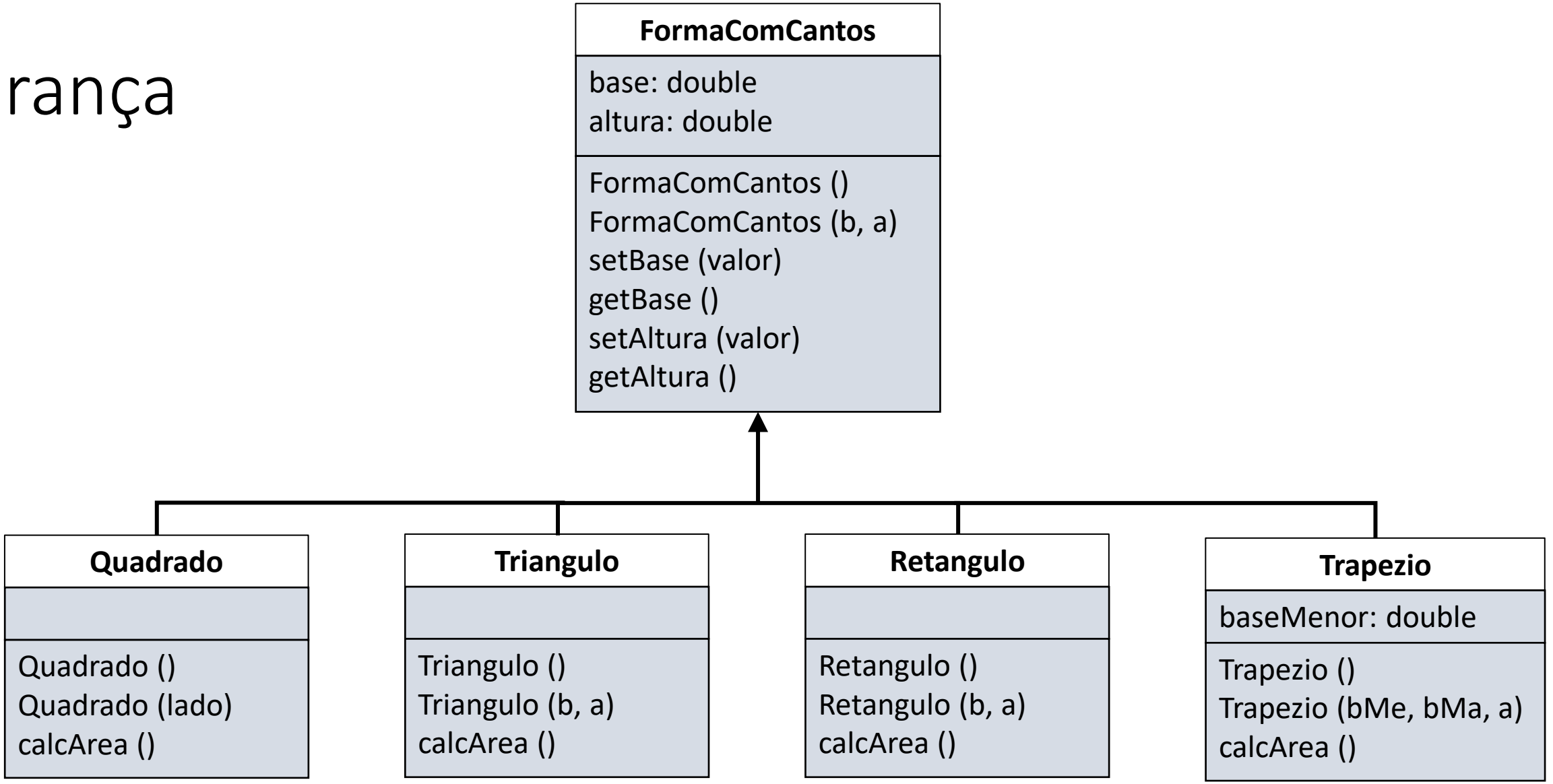
Triangulo
base: double altura: double
Triangulo () Triangulo (b, a) setBase (valor) getBase () setAltura (valor) getAltura () calcArea ()

Circulo
raio: double
Circulo () Circulo (r) setRaio (valor) getRaio () calcArea ()

Herança

- É a característica da POO que permite derivar nova Classe a partir de uma Classe pré existente
- Neste contexto a Classe pré existente é chamada de Superclasse
- A Classe derivada herda todos os atributos e métodos da Superclasse e pode implementar os seus próprios
- A Herança pode ser praticada em tantos níveis quanto necessário para que as Classes modelem um conjunto de requisitos de software
- A notação UML (Unified Modeling Language) é usada para representar graficamente a hierarquia de classes

Herança



Exercício: Implemente as novas classes Quadrado e Trapézio derivadas de FormaComCantos

Polimorfismo

- É a característica da POO que permite alterar o comportamento de um objeto (instância)
- Há dois tipos de Polimorfismo
  - Estático: que é implementado através da Sobrecarga de métodos e é resolvido em tempo de compilação do programa
  - Dinâmico: que é implementado através de Sobreposição de métodos e está associado ao uso da Herança e é resolvido em tempo de execução do programa

- Já vimos a Sobrecarga quando tratamos de Construtores (embora a mesma possa ser usada para quaisquer métodos)

Polimorfismo - Sobreposição

- Ocorre quando é feita a sobrescrita de métodos, ou seja, uma classe filha sobrescreve um método presente na superclasse
- Assim a superclasse deve possuir o método em questão
- O método sobrescrito na subclasse tem que ser idêntico ao método da superclasse (mesmo nome, tipo de retorno e parâmetros)
- Caso haja alguma diferença o interpretador tratará o método como sobrecarregado e não sobreposto
- E se o método não existir na superclasse?
  - Por exemplo: o método calcArea() dos nossos exemplos não existe na classe FormaComCantos

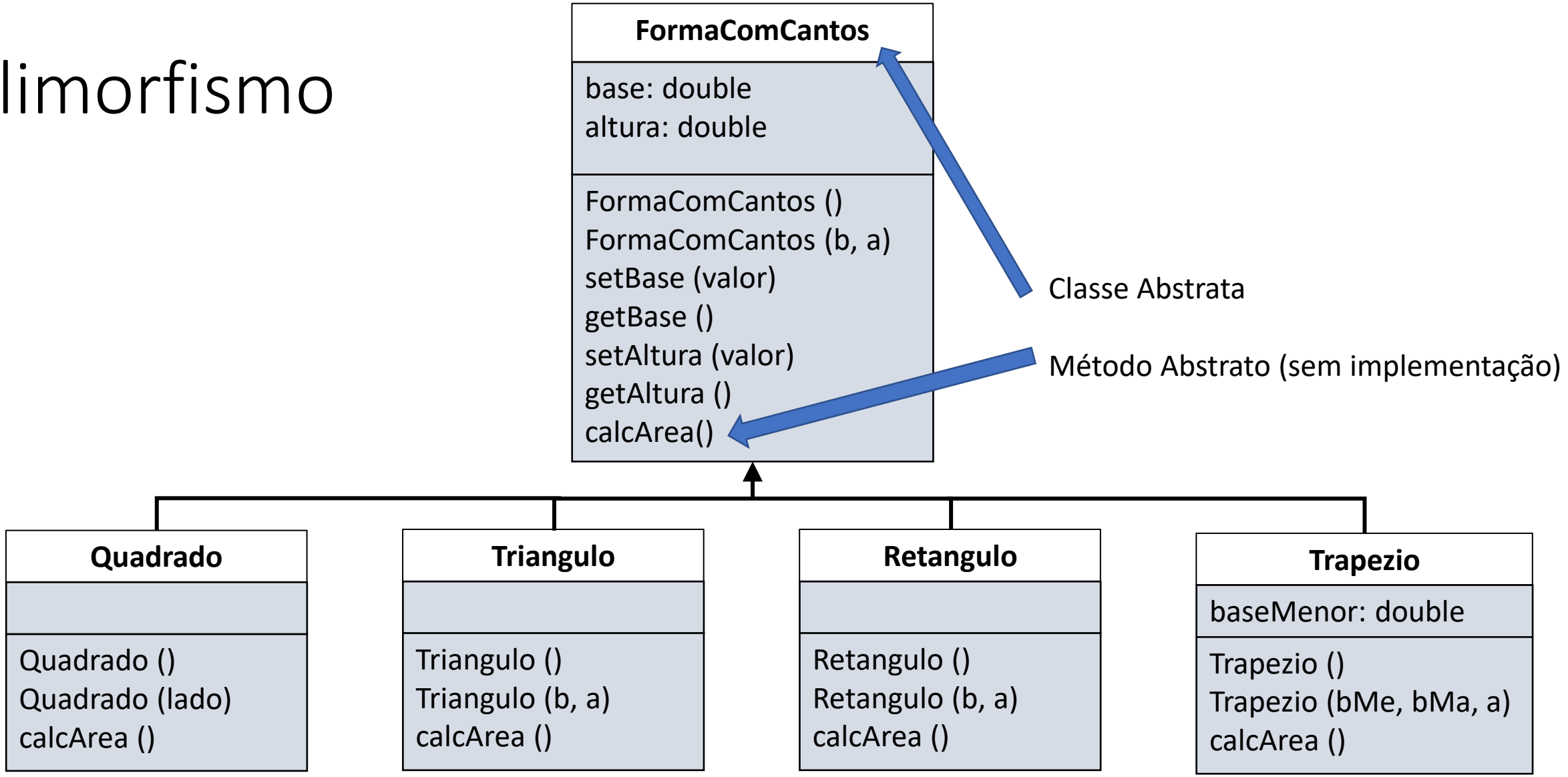
Polimorfismo – Classes Abstratas

- A resposta para a questão anterior são as Classes Abstratas
- Nessas classes declara-se o método, mas não é feita sua implementação
- Neste caso é necessário usar o modificador `abstract` para indicar que tanto a classe como o método são abstratos
- A classe FormaComCantos ficará assim:

```
public abstract class FormaComCantos {
   . . .
   . . .
   public abstract double calcArea();

}
```

Polimorfismo

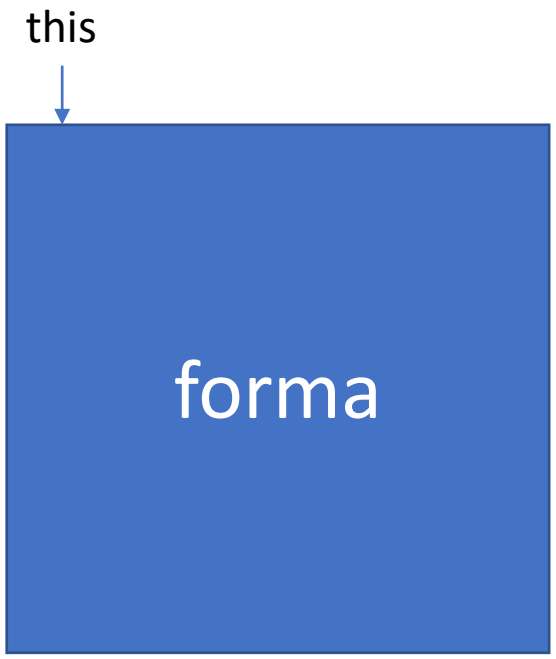


Polimorfismo – Exercício

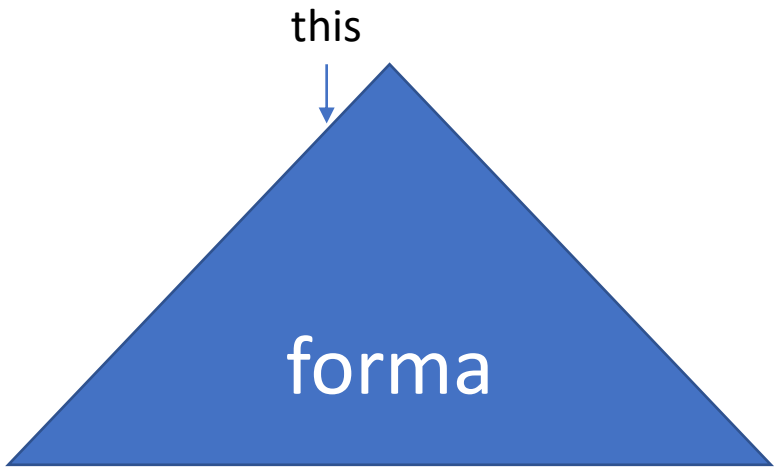
- Altere todas as classes do nosso exemplo incluindo um método chamado nomeForma() que retorna um string indicando o nome da forma geométrica a que se refere a classe.
- Use esse nome na mensagem de exibição da Área no programa de teste. Verifique a variação polimórfica da mensagem conforme a classe instanciada.

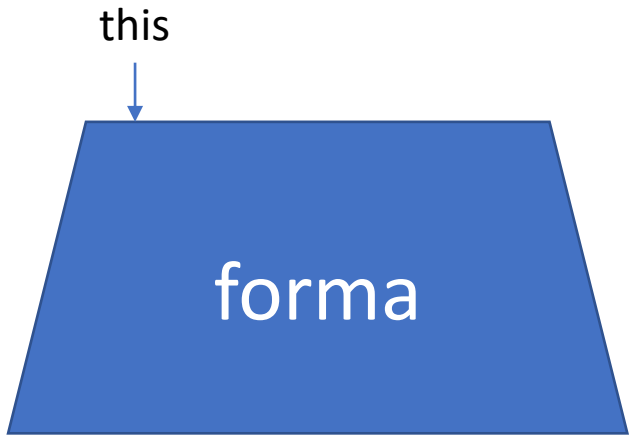
Uso da palavra chave "**this**"

- Trata-se de elemento frequente no contexto da linguagem Java
- Diferentes objetos (instâncias) da mesma classe tem os mesmos atributos. Então, se existirem dois objetos da Classe Circulo, haverá em memória dois atributos **raio**
- Como ter certeza de que nos referimos a um atributo de um objeto específico? Usando **this**
- **this** é uma palavra chave usada num método como referência para o objeto corrente, ela tem o significado de: “o objeto para o qual este trecho de código está sendo executado”.


this
forma


this
forma

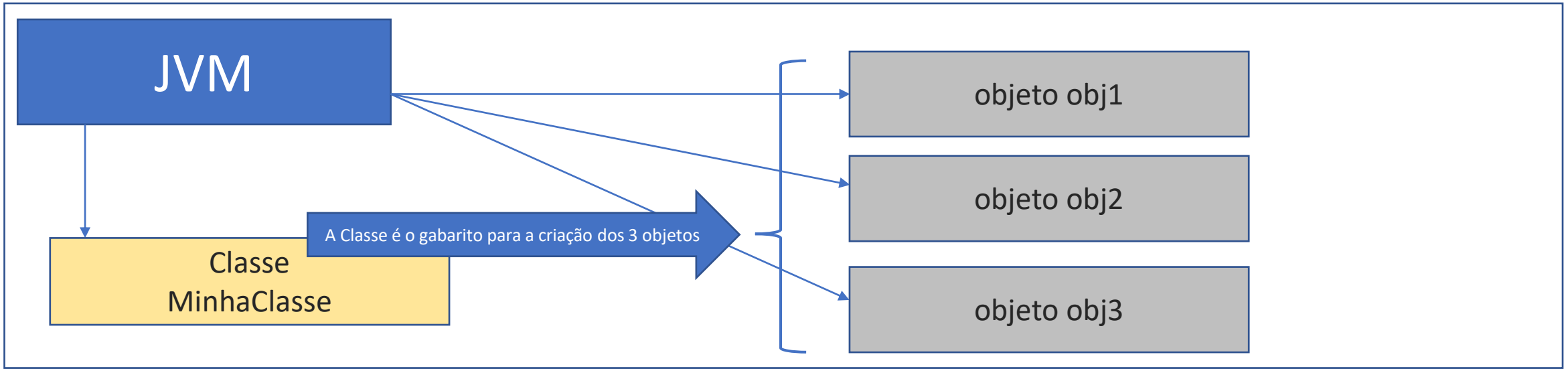

this
forma


this
forma

Situação: o programa Java contém 1 classe e cria 3 objetos dessa classe

```
MinhaClasse obj1 = new MinhaClasse()
MinhaClasse obj2 = new MinhaClasse()
MinhaClasse obj3 = new MinhaClasse()
```

Memória RAM



Em Java há elementos que pertencem aos objetos e há elementos que pertencem à classe.

Todos os elementos que pertencem a um objeto podem ser acessados através do “this”

E todos os elementos de Classe estão automaticamente disponíveis a todos os objetos criados com base na Classe