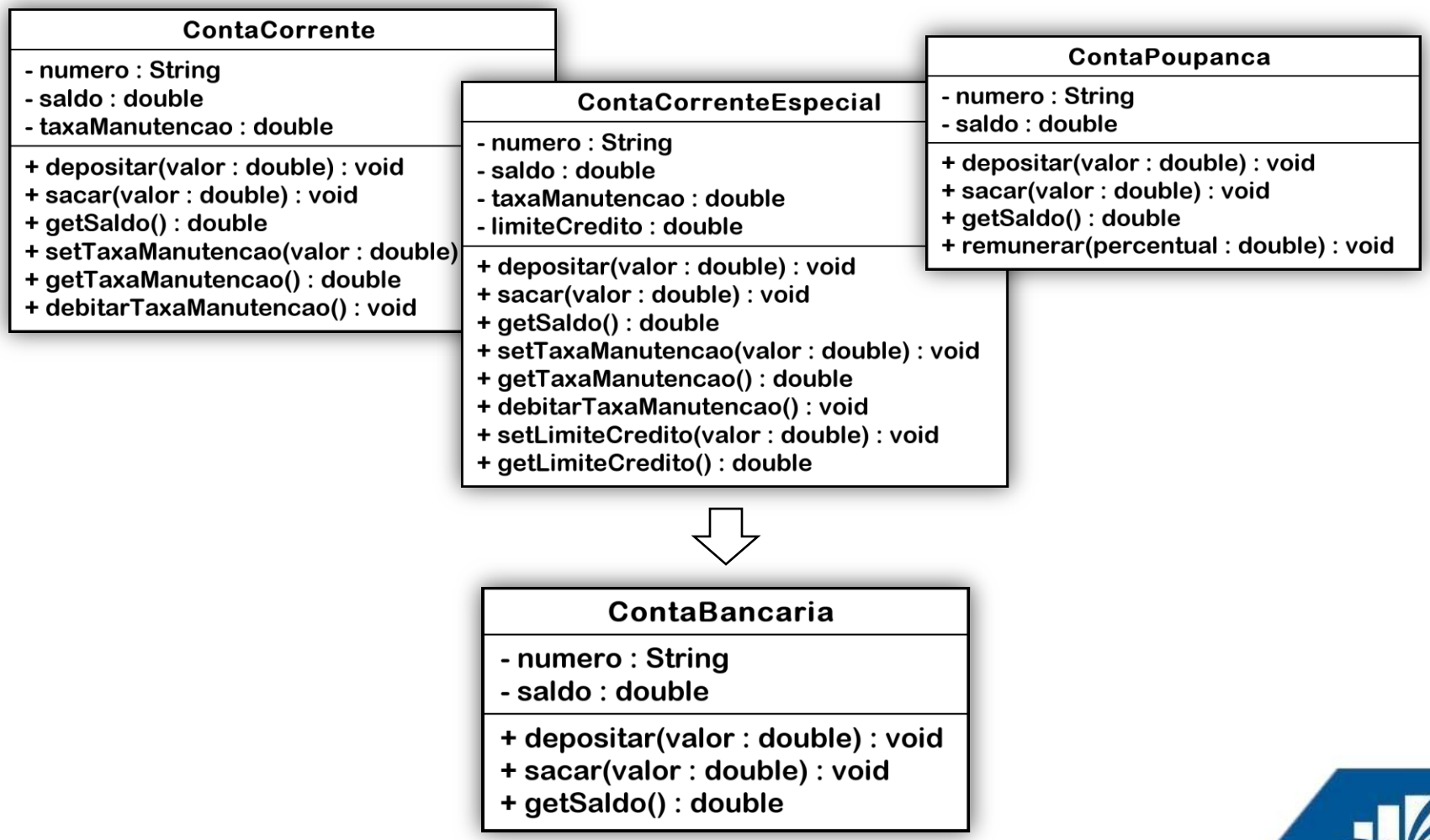
Herança

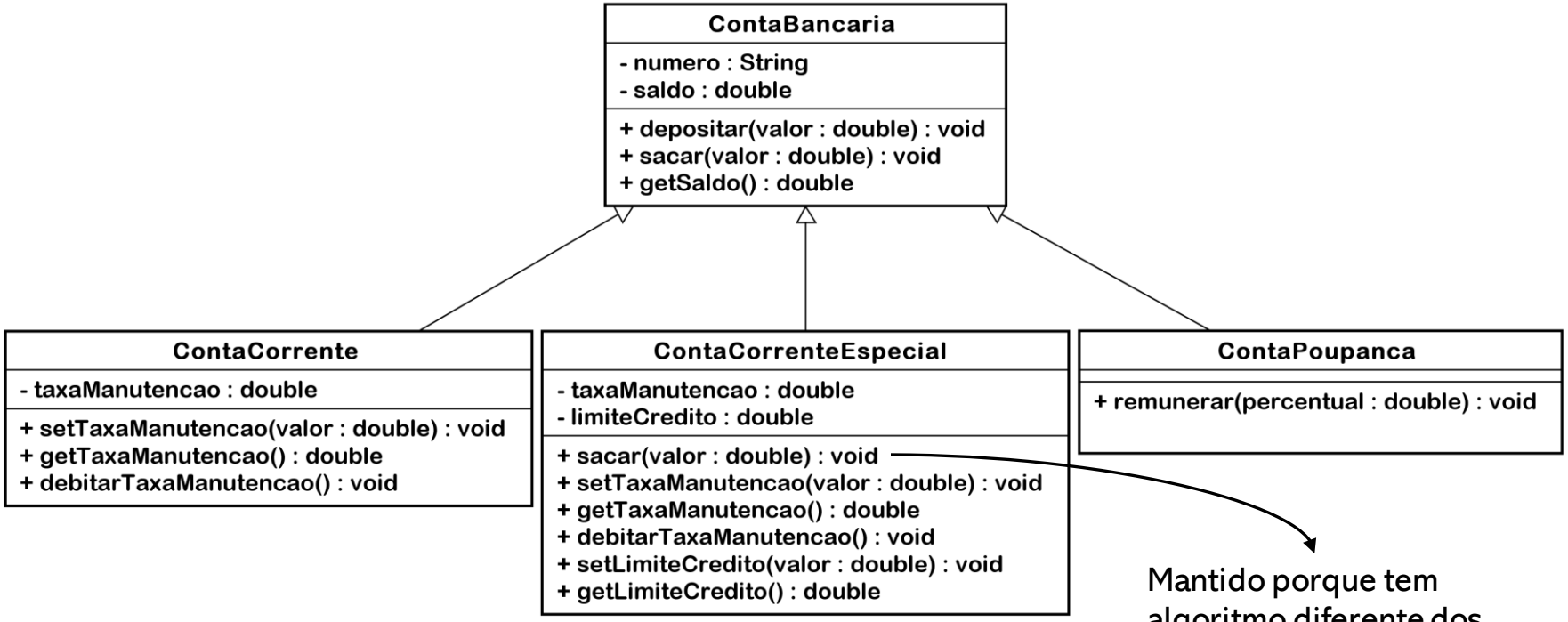
Generalização

- É um processo de encontrar e criar superclasses:
  - Num conjunto de classes relacionadas, localizar os membros (métodos e atributos) comuns entre as classes;
  - Mover os membros comuns para uma classe, tornando-a superclasse das classes originais
  - Para os métodos que tiverem comportamento diferenciado, manter na classe original

Exemplo de aplicação



Exemplo


ContaBancaria
- numero : String
- saldo : double
+ depositar(valor : double) : void
+ sacar(valor : double) : void
+ getSaldo() : double
ContaCorrente
- taxaManutencao : double
+ setTaxaManutencao(valor : double) : void
+ getTaxaManutencao() : double
+ debitarTaxaManutencao() : void
ContaCorrenteEspecial
- taxaManutencao : double
- limiteCredito : double
+ sacar(valor : double) : void
+ setTaxaManutencao(valor : double) : void
+ getTaxaManutencao() : double
+ debitarTaxaManutencao() : void
+ setLimiteCredito(valor : double) : void
+ getLimiteCredito() : double
ContaPoupanca
+ remunerar(percentual : double) : void
Mantido porque tem algoritmo diferente dos demais

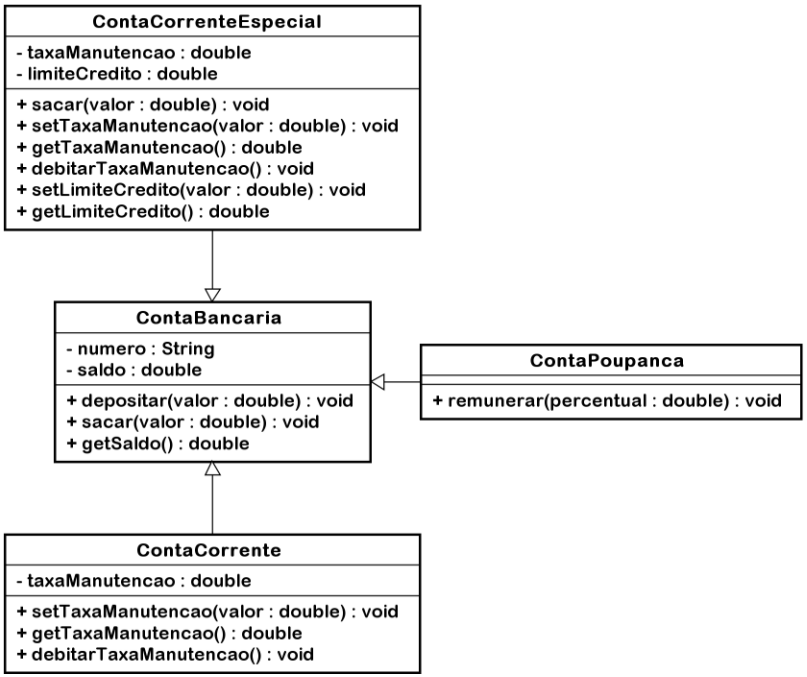
Generalização

- O processo de generalização pode ser aplicado mais vezes, com um conjunto menor de classes

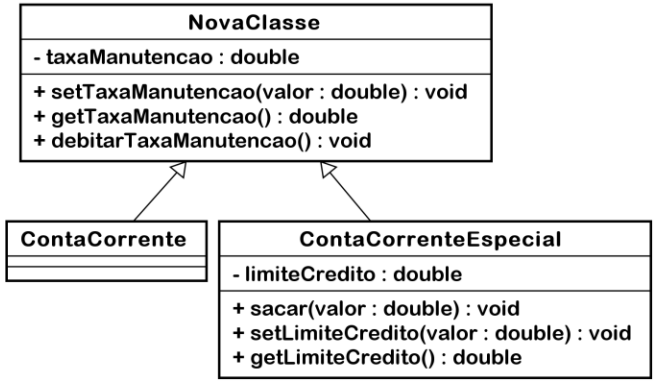
- Se após aplicar o processo de generalização, uma classe não possuir membro algum, esta classe pode se promovida para ser superclasse das demais classes.

Exemplo

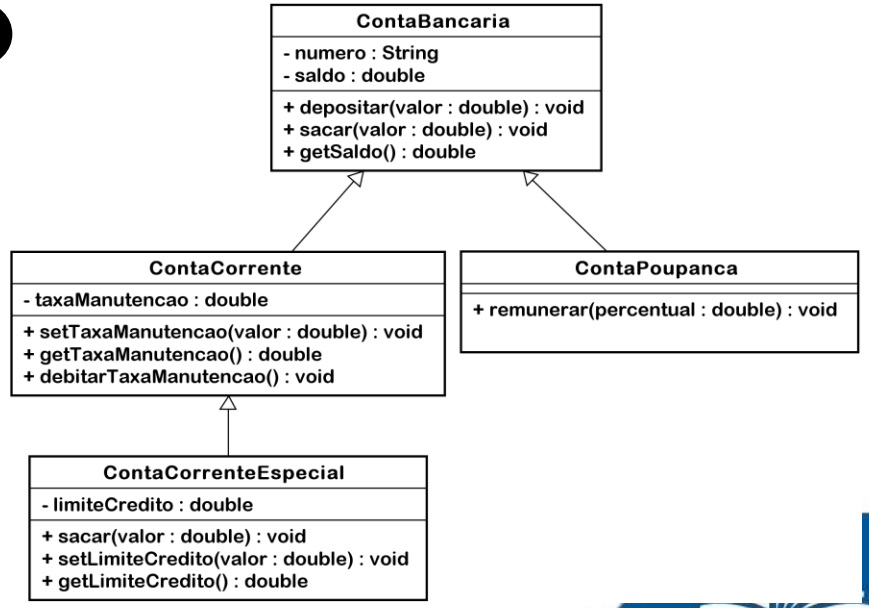
1


ContaCorrenteEspecial
- taxaManutencao : double
- limiteCredito : double
+ sacar(valor : double) : void
+ setTaxaManutencao(valor : double) : void
+ getTaxaManutencao() : double
+ debitarTaxaManutencao() : void
+ setLimiteCredito(valor : double) : void
+ getLimiteCredito() : double
ContaBancaria
- numero : String
- saldo : double
+ depositar(valor : double) : void
+ sacar(valor : double) : void
+ getSaldo() : double
ContaPoupanca
+ remunerar(percentual : double) : void
ContaCorrente
- taxaManutencao : double
+ setTaxaManutencao(valor : double) : void
+ getTaxaManutencao() : double
+ debitarTaxaManutencao() : void

2


NovaClasse
- taxaManutencao : double
+ setTaxaManutencao(valor : double) : void
+ getTaxaManutencao() : double
+ debitarTaxaManutencao() : void
ContaCorrente
ContaCorrenteEspecial
- limiteCredito : double
+ sacar(valor : double) : void
+ setLimiteCredito(valor : double) : void
+ getLimiteCredito() : double

3


ContaBancaria
- numero : String
- saldo : double
+ depositar(valor : double) : void
+ sacar(valor : double) : void
+ getSaldo() : double
ContaCorrente
- taxaManutencao : double
+ setTaxaManutencao(valor : double) : void
+ getTaxaManutencao() : double
+ debitarTaxaManutencao() : void
ContaPoupanca
+ remunerar(percentual : double) : void
ContaCorrenteEspecial
- limiteCredito : double
+ sacar(valor : double) : void
+ setLimiteCredito(valor : double) : void
+ getLimiteCredito() : double

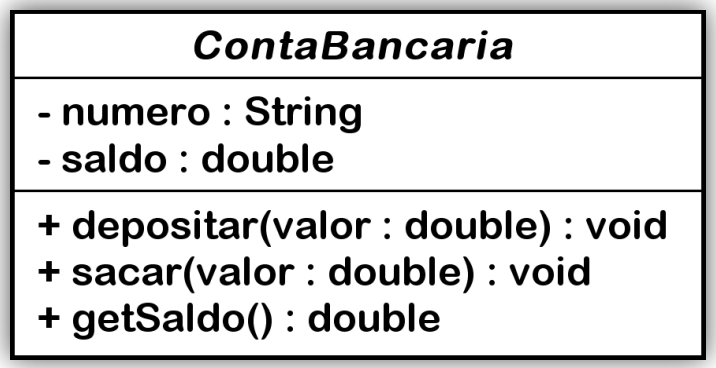
Generalização

- Em muitas ocasiões, a superclasse que surgiu a partir do processo de generalização não tem um significado no mundo real
  - Trata-se de um efeito do processo de generalização
  - A superclasse pode representar um conceito abstrato que não existe no mundo real

Classe abstrata

Classe abstrata

- Uma classe que representa um conceito genérico
- Em UML, é expressa com o nome da classe em itálico



- Em Java, utiliza-se o modificador *abstract,* como em:

```
public abstract class ContaBancaria {

    // ...

}
```

Classe Abstrata

- Por ser uma abstração, não é possível criar objetos de classes abstratas

```
public abstract class Classe1 {

    // ...

}
```

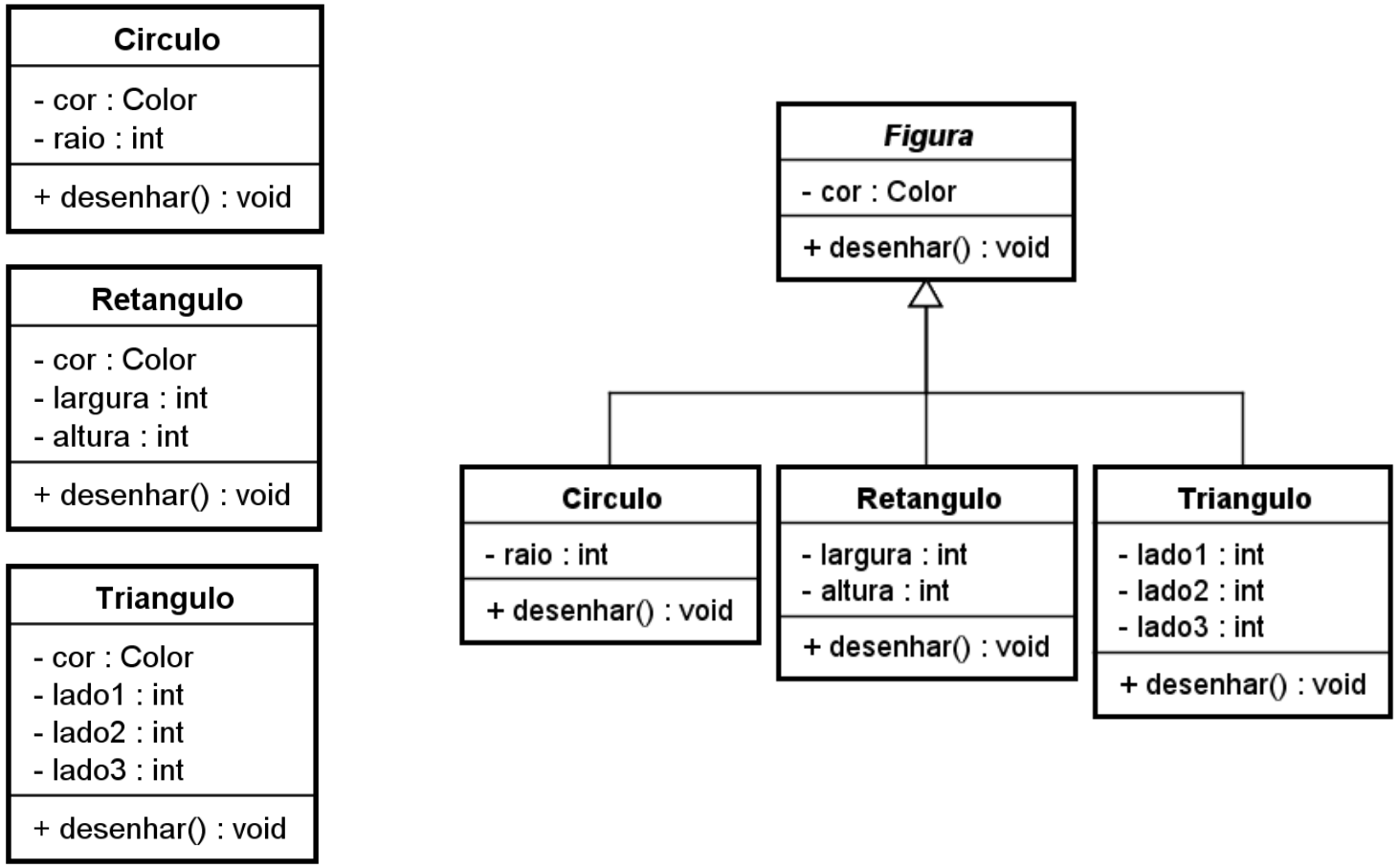
```
Classe1 c1 = new Classe1();
```

Erro de compilação: "não pode instanciar Classe1"

- Classes abstratas precisam ser estendidas para serem reusadas.
  - Classes que não são abstratas são conhecidas como **classes concretas**
  - Somente classes concretas podem ser instanciadas

Métodos abstratos

Motivação - Métodos abstratos



Métodos abstratos

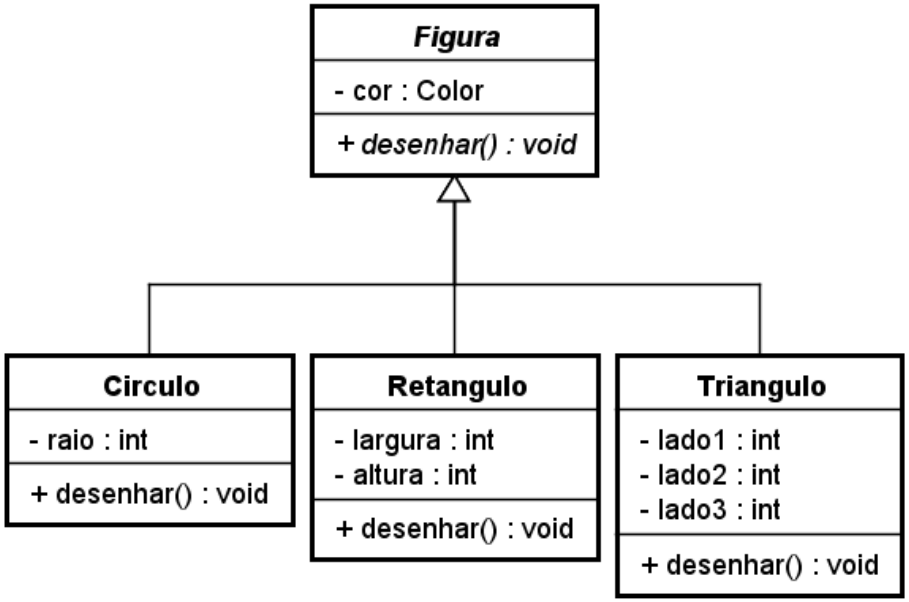
- Como evitar esta implementação?

```
public class Figura {

    private Color cor;

    public void desenhar() {

    }

}
```

```
public class Pentagono extends Figura {

}
```

Métodos abstratos

- Definindo um método como abstrato, instruímos o compilador a exigir que as subclasses implementem um método com tal assinatura
- Em UML, os métodos abstratos são escritos em itálico



Exemplo

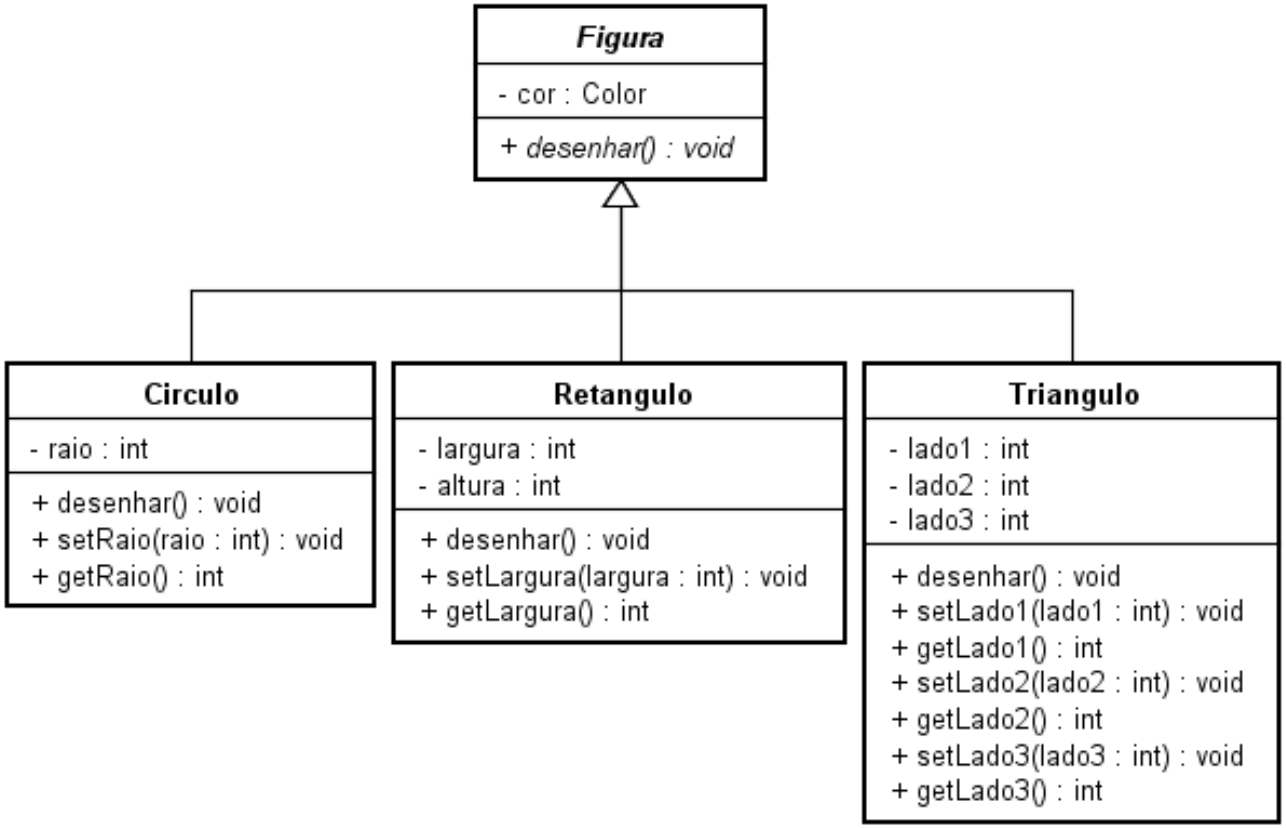


Figura
- cor : Color
+ desenhar() : void
Circulo
- raio : int
+ desenhar() : void
+ setRaio(raio : int) : void
+ getRaio() : int
Retangulo
- largura : int
- altura : int
+ desenhar() : void
+ setLargura(largura : int) : void
+ getLargura() : int
Triangulo
- lado1 : int
- lado2 : int
- lado3 : int
+ desenhar() : void
+ setLado1(lado1 : int) : void
+ getLado1() : int
+ setLado2(lado2 : int) : void
+ getLado2() : int
+ setLado3(lado3 : int) : void
+ getLado3() : int

Métodos abstratos

- Ao especificar o método desenhar() como abstrato:

```
public class Pentagono extends Figura {

}
```

Erro de compilação: "Pentagono precisa implementar o método abstrato desenhar()"

Métodos abstratos

- Um método abstrato é um método que não possui implementação, pois espera-se que as subclasses se encarreguem de implementá-lo.
- Declaramos um método abstrato para indicar que o método deve existir na subclasse, embora não há implementação para o método na superclasse
- Qualquer classe que contém um método abstrato deve ser abstrata também

Métodos abstratos

- Métodos abstratos são declarados com o modificador **abstract** antes do tipo de retorno
- Não possuem corpo

```
public abstract class Figura {

    private Color cor;

    public abstract void desenhar();

}
```

Classes abstratas

- Uma classe abstrata pode ser especializada a partir de uma classe concreta

Classes e métodos que não podem ser estendidos/sobrescritos

Impedir que um método seja sobrescrito

- Em algumas ocasiões, um método possui uma implementação que não deveria estar sujeito a ser alterado através da sobrescrita de métodos em subclasses, pois a mudança de algoritmo pode tornar o estado do objeto inconsistente.
- Para impedir que um método seja sobrescrito na subclasse, utilizar a palavra reservada `final`, antes do tipo de dado de retorno, como em:

```
public final void metodo1() {
    // ...
}
```

Impedir que uma classe seja estendida

- É possível impedir a extensão de uma classe. Para isso, utilizar a palavra reservada `final` antes da palavra `class`, como em:

```
public final class Classe1 {
   // ...
}
```

```
public class Classe2 extends Classe1 {

}
```

Erro de compilação

- Este recurso é útil, por exemplo, para criar classes de objetos imutáveis, como os objetos da classe String.