

Programação orientada a objetos

Classes abstratas

Relacionamentos entre classes

- Classes podem se relacionar entre si, definindo um vínculo entre os objetos dessas classes.




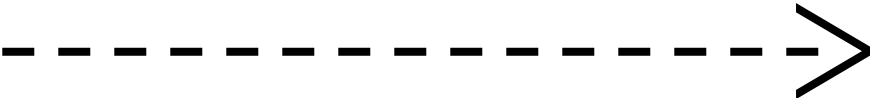


Exemplos

- Um **cliente** possui um **endereço**.
- Uma **empresa** é composta por **funcionários**.
- Uma **moto** é um tipo de **veículo**.
- Um **restaurante** possui **pratos**.
- Uma **correspondência** possui um **remetente** e um **destinatário**.

Relacionamentos entre classes

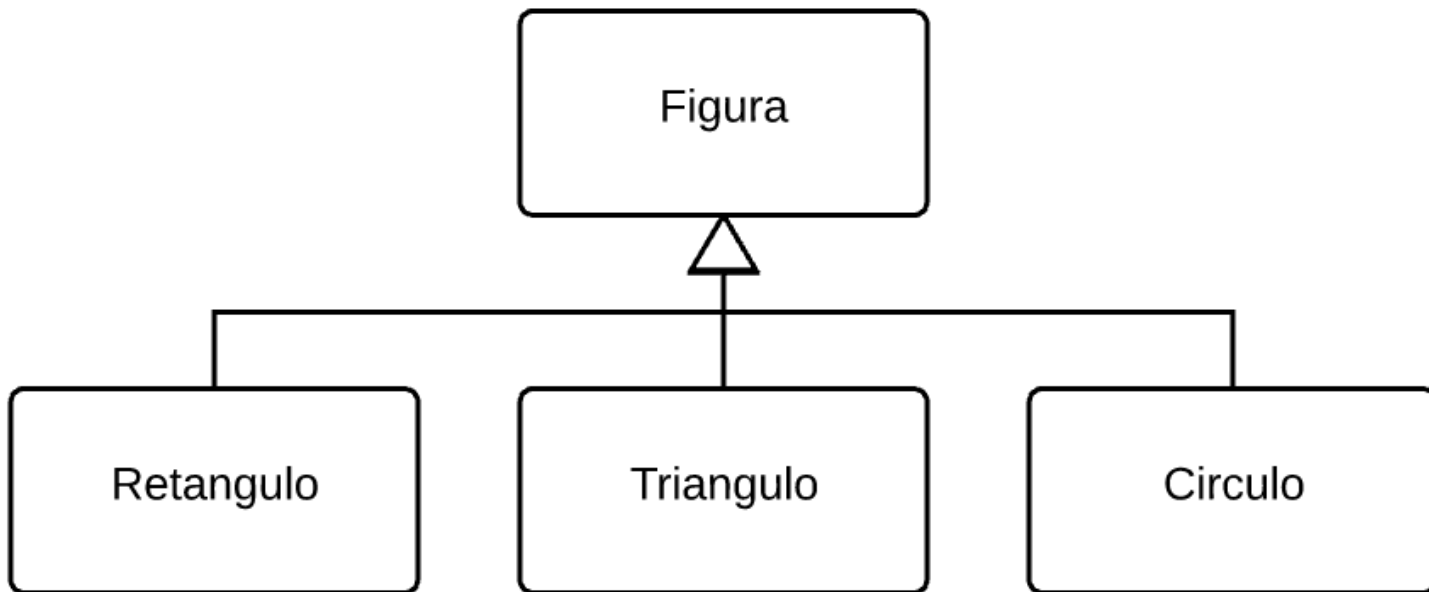
- **Associação:** conexão entre classes.
- **Agregação e composição:** especialização de uma associação onde um todo é relacionado com suas partes (relacionamento “parte-de”).
- **Dependência:** um objeto depende de alguma forma de outro (relacionamento de utilização).
- **Herança (generalização):** um dos princípios da orientação a objetos, permite a reutilização, uma nova classe pode ser definida a partir de outra já existente.
- **Realização:** um contrato que a classe segue (obrigação).

Relacionamentos entre classes

- Associação: 
- Agregação: 
- Composição: 
- Dependência: 
- Herança (generalização): 
- Realização: 

Classes abstratas

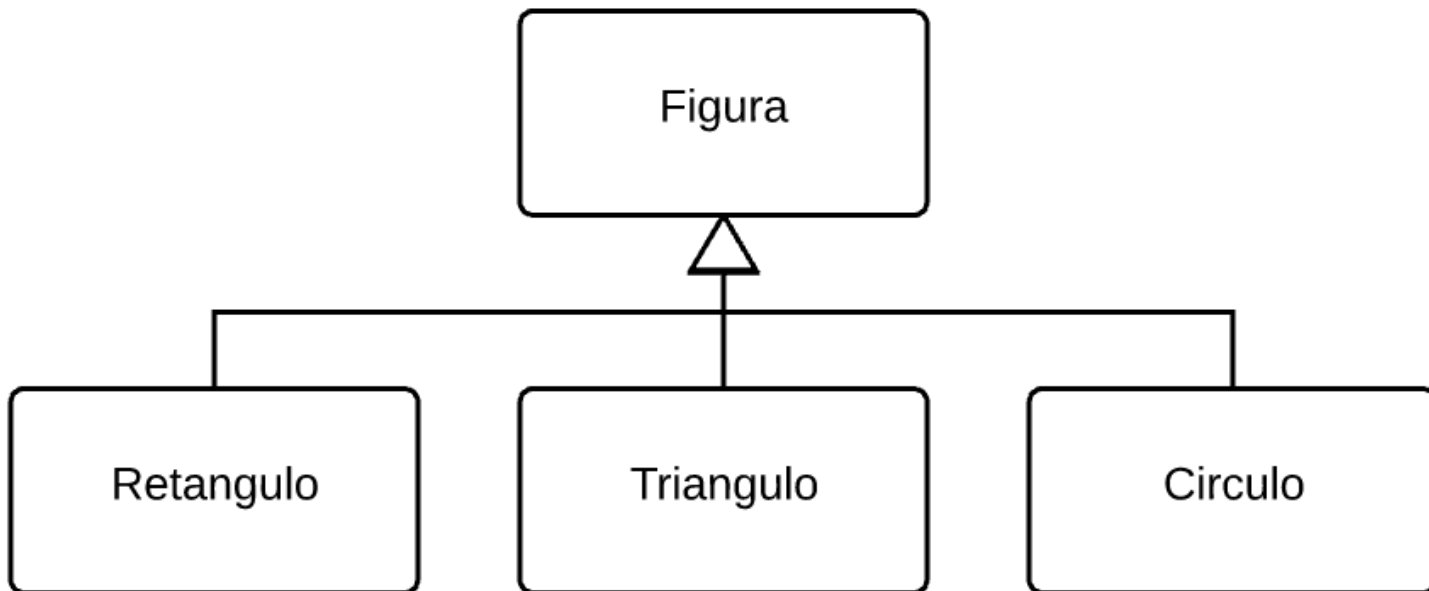
- O maior benefício oriundo da generalização/especialização de classes através da herança é o **polimorfismo**. Ele é um recurso poderoso, capaz de garantir ao sistema flexibilidade.
- **Exemplo:** figuras geométricas, que compartilham entre si uma cor e possuem os métodos para cálculo da área e perímetro.



Classes abstratas

```
public class Figura {  
    private String cor;  
  
    public double area() {  
        return 0;  
    }  
  
    public double perimetro() {  
        return 0;  
    }  
}
```

```
public class Retangulo extends Figura {  
    private double lado1, lado2;  
  
    public double area() {  
        return lado1 * lado2;  
    }  
  
    public double perimetro() {  
        return (lado1 * 2) + (lado2 * 2);  
    }  
}
```



Classes abstratas

- Neste exemplo, não faz sentido termos uma instância da classe Figura, pois uma figura é sempre um triângulo, um retângulo, um círculo, um trapézio, etc.
- A classe Figura define uma entidade abstrata, que só existe para definir uma estrutura comum a todas as figuras concretas, além de estabelecer os métodos que todas devem apresentar (cálculo da área e do trapézio).
- Tecnicamente, usamos a classe figura para garantir a estrutura e o comportamento de todas as figuras e para nos fornecer polimorfismo.
- Não faz sentido termos uma instância de Figura e calcularmos sua área ou perímetro, por exemplo.

```
Figura figura = new Figura();  
figura.area();  
figura.perimetro();
```

Classes abstratas

- Uma dica para identificar casos onde não faz sentido termos uma instância da superclasse é analisar seus métodos. Quando não é possível definir sua implementação, é um bom indicativo de que a classe não deve ser instanciada.
- No exemplo das figuras geométricas, cada classe concreta (Triângulo, Retângulo, etc.) possui sua própria forma de calcular a área e o perímetro em função dos seus atributos. Isso não acontece na classe Figura, que é incapaz de prover uma implementação dos métodos supracitados.

```
public class Figura {  
  
    private String cor;  
  
    public double area() {  
        return 0;  
    }  
  
    public double perimetro() {  
        return 0;  
    }  
}
```


Classes abstratas

- Nestes casos, podemos definir a classe como **abstrata**, o que implica na impossibilidade de ser instanciada.
- Com isso, a classe se restringe a definir a estrutura e o comportamento das classes que a estenderem e fornecer polimorfismo.
- Por exemplo, se tivermos uma lista de figuras (**List<Figura>**) estamos fazendo uso do polimorfismo. Caso a classe **Figura** seja abstrata, temos a garantia de que todos os objetos da lista são figuras concretas (triângulos, retângulos, etc.) e, com isso, faz sentido chamarmos os métodos **area()** e **perimetro()** para qualquer um deles.

Classes abstratas

Implementação

- A palavra **abstract** define uma classe como abstrata.

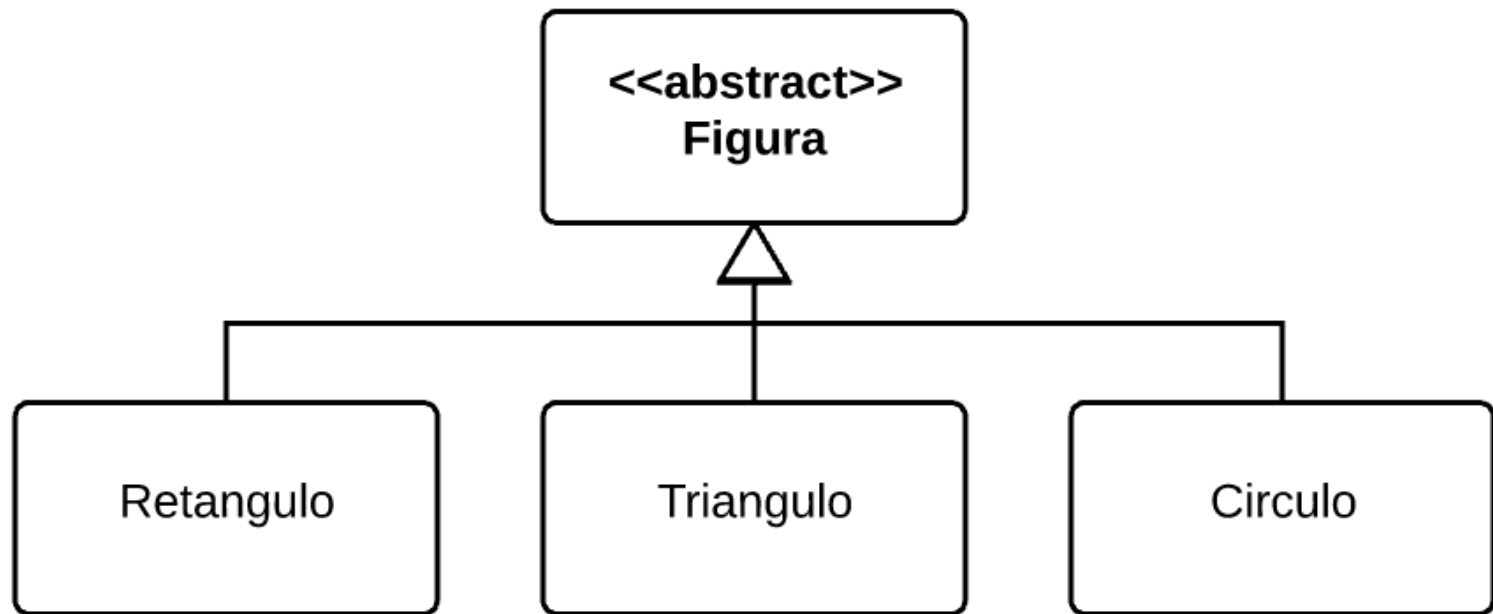
```
public abstract class Figura {  
    private String cor;  
  
    public double area() {  
        return 0;  
    }  
  
    public double perimetro() {  
        return 0;  
    }  
}
```

- Com isso, não é possível instanciar objetos dessa classe.

```
Figura f1 = new Figura();           //Erro de compilação  
Figura f2 = new Circulo();          //Funciona normalmente
```

Classes abstratas

- No diagrama de classes, as classes abstratas devem ser representadas como tal, para que esta característica fique facilmente visível.



Métodos abstratos

- Resolvemos o problema de não fazer sentido termos uma instância da classe *Figura*. Porém, a implementação padrão dos métodos **area()** e **perimetro()** ainda não fazem sentido na classe *Figura*, uma vez que trata-se de uma entidade abstrada.
- O ideal seria:
 - Não haver implementação do método, uma vez que não faz sentido.
 - Exigir que as subclasses implementem o método, uma vez que a implementação padrão não faz sentido.
- Isso é possível definindo os **métodos** supracitados como **abstratos**. Com isso, apenas a assinatura do método é definido na superclasse, exigindo que as subclasses os implementem.
- Com isso, ao obtermos um objeto do tipo *Figura*, sabemos que ele é uma referência a um dos tipos concretos e que os métodos **area()** e **perimetro()** estão disponíveis.

Métodos abstratos

- Para definir um método abstrato, basta inserir a palavra **abstract** na sua assinatura e omitir sua implementação, colocando ponto-e-vírgula após o fechamento de parêntesis.

```
public abstract class Figura {  
    private String cor;  
    public abstract double area();  
    public abstract double perimetro();  
}
```

- Só é possível definir um método como abstrato se ele pertence a uma classe abstrata. Classes concretas não podem conter métodos abstratos e devem implementar todos os métodos abstratos herdados.
- Uma classe abstrata pode definir métodos abstratos e métodos concretos simultaneamente;

Referências

CAELUM. **Apostila Java e Orientação a Objetos**. Curso FJ-11, 2016.

DEITEL, H. M. **Java: como programar**. H. M Deitel e P. J. Deitel - 8a ed. Porto Alegre: Prentice-Hall, 2010.

Leitura complementar

TutorialsPoint Java (<http://www.tutorialspoint.com/java>).