Tipos de classe Java:
- Classe principal (main)
- Entidade (tipo do dominio)
  * Classe abstrata (Conta, Funcionario)
- Interface
- Enum (constantes)

Estoque de produtos:

```
public class Estoque {
  //quantidades de produtos
}

public class TesteEstoque {
  public static void main(String[] args) {
      ArrayList<Bicicleta> bicicletas = new ArrayList<>({....});
      ArrayList<Celular> celulares  = new ArrayList<>({....});
     //estoque único
      Estoque e1 = new Estoque();
      Estoque e2 = new Estoque();
      e1.adiciona(bicicletas);
      e2.adiciona(celulares);
  }
}
```

Design Pattern - Como fazer (Solução)
*Singleton

```
1:  public class Estoque {
2:
3:     private static final Estoque instance = new Estoque(); //(2)
4:     private List<Produto> produtos;
5:
6:
7:     private Estoque() { //(1)
8:         produtos = new ArrayList<>();
9:     }
10:
11:    public static Estoque getInstance() { //(3)
12:        return instance;
13:    }
14: }
```

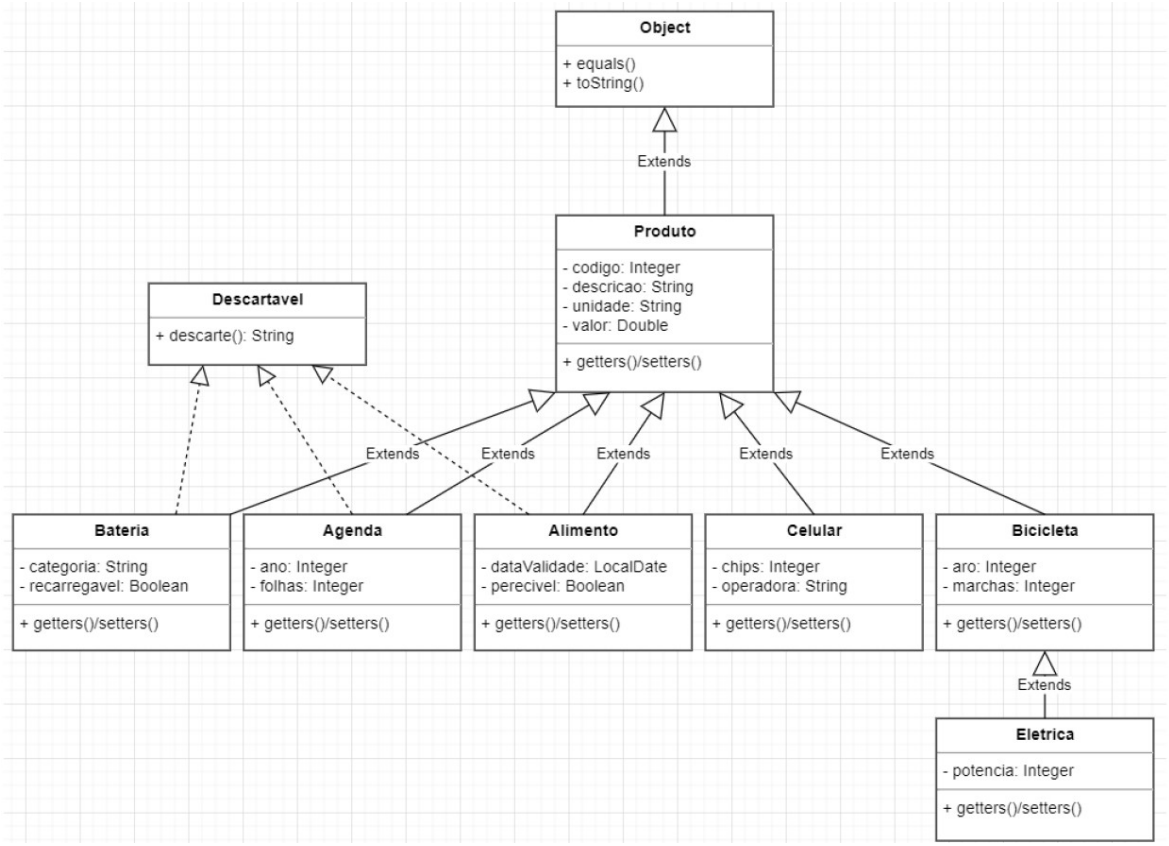
```
private Estoque() {
    if (instance == null) {
        instance = new Estoque();
    }
}
```

Estoque de produtos:

```
public class Estoque {
  //quantidades de produtos
}
public class TesteEstoque {
  public static void main(String[] args) {
      ArrayList<Bicicleta> bicicletas = new ArrayList<>({....});
      ArrayList<Celular> celulares  = new ArrayList<>({....});
      //estoque único
      Estoque e1 = Estoque.getInstance();
      Estoque e2 = Estoque.getInstance();
      e1.adiciona(bicicletas);
      e2.adiciona(celulares);
  }
}
```

0xaab11f: Endereço objeto Estoque

Interface




```java
public interface Descartavel {

    public abstract String descarte();

}
```

```java
public class Bateria extends Produto implements Descartavel {

    private String categoria;
    private Boolean recarregavel;

    public Bateria(String descricao, Double valor, String categoria, Boolean recarregavel) {
        super(descricao, valor);
        this.categoria = categoria;
        this.recarregavel = recarregavel;
    }

    @Override
    public String descarte() {
        return "Pode ser descartado em lojas Kalunga ou drogarias São Paulo.";
    }
//outros métodos da classe
}
```

Classe Aninhada

- **Inner Classes**
  - Member classes
  - Local classes
  - Anonymous inner classes
- **Static Classes**

Paradigma funcional:

Funcionamento básico de um calculadora:

```
calcula(num1, num2, operacao)
  - calcula(5, 10, soma) => 15
  - calcula(5, 10, subtracao) => -5
  - calcula(5, 10, multiplicacao) => 50
```

```
soma: efetua(n1, n2) return n1 + n2;
subtracao: efetua(n1, n2) return n1 - n2;
multiplicacao: efetua(n1, n2) return n1 * n2;
```

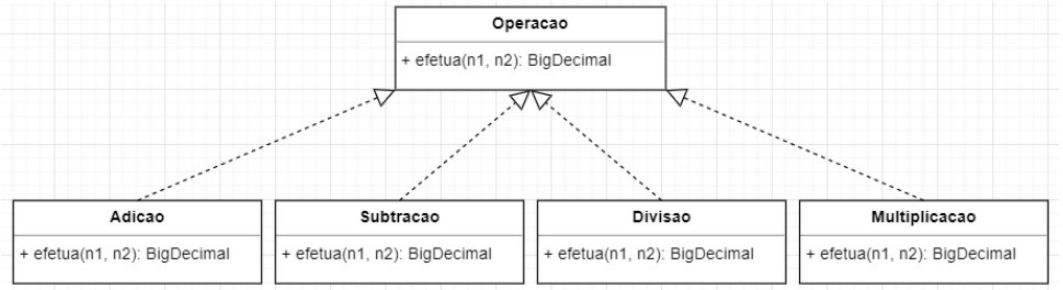
```
public interface Operacao {
    public abstract double efetua(double n1, doublen2);
}
```

```
public class Calculadora {
    public double calcula(double n1, double n2, Operacao op) {
        return op.efetua(n1,n2);
    }
}
```

```
public class Soma implements Operacao {
    public double efetua(n1, n2) {
        return n1 + n2;
    }
}
```

```
public class Subtracao  implements Operacao {
    public double efetua(n1, n2) {
        return n1 - n2;
    }
}
```

```
public class TesteCalculadora {
    public static void main(String args[]) {
        Calculadora calc = new Calculadora();
        Soma soma = Soma();
        Subtracao sub = new Subtracao();
        //....
        sysout(calc.calcula(5, 10, soma);
        sysout(calc.calcula(5, 10, sub);
        //.....
    }
}
```




```java
public interface Operacao {

   BigDecimal efetua(BigDecimal num1, BigDecimal num2);

}
```

```java
public class Calculadora {

   static public BigDecimal calcula(BigDecimal num1, BigDecimal num2, Operacao operacao) {
      return operacao.efetua(num1, num2);
   }

}
```

```java
public class Calculadora {

   static public BigDecimal calcula(BigDecimal num1, BigDecimal num2, Operacao operacao) {
      return operacao.efetua(num1, num2);
   }

   public class Adicao implements Operacao {
      @Override
      public BigDecimal efetua(BigDecimal num1, BigDecimal num2) {
         return num1.add(num2);
      }
   }

   public class Subtracao implements Operacao {
      @Override
      public BigDecimal efetua(BigDecimal num1, BigDecimal num2) {
         return num1.subtract(num2);
      }
   }

   public class Multiplicacao implements Operacao {
      @Override
      public BigDecimal efetua(BigDecimal num1, BigDecimal num2) {
         return num1.multiply(num2);
      }
   }

   public class Divisao implements Operacao {
      @Override
      public BigDecimal efetua(BigDecimal num1, BigDecimal num2) {
         return num1.divide(num2);
      }
   }
}
```

```java
public class TesteCalculadora01 {

    public static void main(String[] args) {

       Operacao operacoes[] = {
          new Calculadora().new Adicao(),
          new Calculadora().new Subtracao(),
          new Calculadora().new Multiplicacao(),
          new Calculadora().new Divisao()
       };

       BigDecimal num1 = new BigDecimal(10);
       BigDecimal num2 = new BigDecimal(2);

       for (Operacao operacao : operacoes) {
          System.out.printf("Operação (classe): %s - Resposta: %.2f \n",
                operacao.getClass().getSimpleName(),
                Calculadora.calcula(num1, num2, operacao));
       }

    }

}
```

```java
public class Calculadora {
   static public BigDecimal calcula(BigDecimal num1, BigDecimal num2, Operacao operacao) {
      return operacao.efetua(num1, num2);
   }
   static public class Adicao implements Operacao {
      @Override public BigDecimal efetua(BigDecimal num1, BigDecimal num2) {
         return num1.add(num2);
   }}
   static public class Subtracao implements Operacao {
      @Override public BigDecimal efetua(BigDecimal num1, BigDecimal num2) {
         return num1.subtract(num2);
   }}
   static public class Multiplicacao implements Operacao {
      @Override public BigDecimal efetua(BigDecimal num1, BigDecimal num2) {
         return num1.multiply(num2);
   }}
   static public class Divisao implements Operacao {
      @Override public BigDecimal efetua(BigDecimal num1, BigDecimal num2) {
         return num1.divide(num2);
   }}
}
```

```java
public class TesteCalculadora01 {

   public static void main(String[] args) {

      Operacao operacoes[] = {
         new Calculadora.Adicao(),
         new Calculadora.Subtracao(),
         new Calculadora.Multiplicacao(),
         new Calculadora.Divisao()
      };

      BigDecimal operador1 = new BigDecimal(10);
      BigDecimal operador2 = new BigDecimal(2);

      for (Operacao operacao : operacoes) {
         System.out.printf("Operação (classe): %s - Resposta: %.2f \n",
               operacao.getClass().getSimpleName(),
               Calculadora.calcula(operador1, operador2, operacao));
      }
   }
}
```

Local class:

```java
public class TesteCalculadora02 {

   public static void main(String[] args) {

      class Adicao implements Operacao {
         @Override
         public BigDecimal efetua(BigDecimal num1, BigDecimal num2) {
            return num1.add(num2);
         }
      }

      BigDecimal n1 = new BigDecimal(10);
      BigDecimal n2 = new BigDecimal(2);

      System.out.printf("10 + 2 = %.2f\n",
            Calculadora.calcula(n1, n2, new Adicao()));
   }

}
```

Classe interna anônima

```
1:  public class TesteCalculadora03 {
2:
3:     public static void main(String[] args) {
4:
5:        Operacao adicao = new Operacao() {
6:           @Override
7:           public BigDecimal efetua(BigDecimal num1, BigDecimal num2) {
8:              return num1.add(num2);
9:           }
10:       };
11:
12:       BigDecimal n1 = new BigDecimal(10);
13:       BigDecimal n2 = new BigDecimal(2);
14:       System.out.printf("10 + 2 = %.2f\n",
15:             Calculadora.calcula(n1, n2, adicao));
16:
17:    }
18:
19: }
```

Java 8 : operador lâmbda (-> ou ::)

```
1:  public interface Operacao {
2:
3:     BigDecimal efetua(BigDecimal num1, BigDecimal num2);
4:
5:  }
```

```
1:  public class TesteCalculadora04 {
2:
3:     public static void main(String[] args) {
4:        Operacao adicao = (BigDecimal num1, BigDecimal num2) -> num1.add(num2);
5:
6:        //...restante do código
7:     }
8:  }
```

```
Operacao adicao = (num1, num2) -> num1.add(num2);
Operacao subtracao = (num1, num2) -> num1.subtract(num2);
Operacao multiplicacao = (num1, num2) -> num1.multiply(num2);
Operacao divisao = (num1, num2) -> num1.divide(num2);
```

A partir da versão 8 do Java, interfaces podem possuir métodos ***default*** e ***static*** e no Java 9 foram incluídos métodos ***private*** e ***static private***.

Os métodos ***default*** são herdados pelas classes que implementam a interface e podem ser **sobrescritos**.

Atenção: modificadores para métodos ***default***

Métodos default não podem receber modificadores **abstract**, **final** ou **static**.

Listas (List):

```
ArrayList<Cliente> clientes = new ArrayList<>();
clientes.add(new Cliente....);
ArrayList<Integer> numeros  = new ArrayList<>();
numeros.add(10);
numeros.add(20);
ArrayList<Produto> produtos = new ArrayList<>();
produtos.add(new Bicicleta(....));
produtos.add(new Celular(....));
```

Operdor diamante:

< ? > => Generics

```
ArrayList<int> numeros = new ArrayList<>();
```