CLASSE

- É constituída de métodos (comportamento) e variáveis (atributos).

- Inicia com a palavra class seguida por um nome, iniciando com uma letra maiúscula.

```
class Teste{
.
.
.
}
```

- As chaves abrem a fecham o corpo de uma classe, tudo o que estiver dentre elas será referente a classe.

- O Java possui classes já definidas e permite que sejam criadas novas classes para resolver problemas específicos.

VARIÁVEIS.

- Os componentes de uma declaração de uma variável são:

access	static	final	transient	Volatile	Type	name

- Nem todos os componentes são necessários, porém a ordem deve ser obedecida.

TIPOS DE VARIÁVEIS

- Locais: existem apenas dentro de um método. Seu escopo é o método onde foi declarada, ou na cláusula do laço for em que for declarada.

```
1 - class VarLocal{
2 -   void imprimeVar(){
3 -        int i = 1;   // variável local
4 -        String s = "bom dia";   //variável local
5 -        System.out.println(s);
6 -        System.out.println(i);
7 -       }
8 - public static void main(String args[]){
9 -       VarLocal v = new VarLocal();
10-       v.imprimeVar();
11-       }
12- }
```

- De Instância: são as declaradas dentro da classe, mas fora de um método. Precisam de um objeto para manipulá-las

```
1- class VarInstancia{
2-       int i = 0;  //variável de instância.
3-  void imprimeVar1(){
4-        System.out.println(i);
5-        i += 2;
6-    }
7-  void imprimeVar2(){
8-       System.out.println(i);
9-       i += 3;
10-  }
11-    public static void main(String args[]){
12-     VarInstancia v = new VarInstancia();
13-     v.imprimeVar1();
14-     v.imprimeVar2();
 15-   System.out.println(v.i); //variável i sendo manipulada por um objeto da sua classe
16-     }
17- }
```

- De Classe: são declaradas dentro da classe, mas fora do método, porém elas não pertencem a um objeto, podem ser manipuladas a partir da classe. É necessário o uso da palavra reservada static.

```
1 - class VarClass{
2 -   static int i = 0; //variável de classe não precisa de um objeto para ser manipulada
3-    public static void main(String args[]){
4-   System.out.println(VarClasse.i);
5-  }
6-}
```

A sentença Classe variável = new Classe(), serve para criar um objeto da classe (instanciar). A variável passar a referenciar um objeto da classe.

MÉTODOS.

- É a parte funcional da classe, implementa seus comportamentos.

- Um método é constituído de um nome, lista de parâmetros e tipo de retorno.

- Os componentes de uma declaração de um método são:

Access	Static	Abstract/final	Native	Synchronized	Return type	Name	Param list	throws

- O nome do método, preferencialmente, deve indicar o que é faz.

Lista de Parâmetros do Método:

- Define quais são elementos, e seus respectivos tipos, necessários para que o método possa executar corretamente o que ele foi planejado.

- Fica entre parêntese a direita do nome do método, pode ter quantos elementos quanto for preciso.

```
1- class Metodo{
2-  void soma(int op1, int op2){
3-   System.out.println(op1+op2);
4- }
5 - public static void main(String args[]){
6-        Metodo m = new Metodo();
7-        m.soma(1,2);
8-        }
9- }
```

Tipo de retorno do Método:

- Diferentemente de linguagens como o Pascal, o Java não faz uma diferença explicita entre procedimento e função, existe apenas método.

- Todo o método deve dizer em sua declaração qual é o seu tipo de retorno, isso equivaleria ao uso de uma função.

```
1- class Funcao{
2- float divisao(float op1, float op2){
3-   return op1/op2;
4-   }
5-  public static void main(String[] args){
6-    Funcao f = new Funcao();
7-    System.out.println(f.divisao(5,2));
8-   }
9- }
```

- A palavra reservada *return* indica define qual será o valor que deverá ser retornado pelo método.

- Se não existir nenhum tipo de retorno deve-se usar a palavra reservada void que define que o método é vazio. É equivalente ao procedimento.

```
1- class Procedimento{
2-   void imprime(String s){
3-        System.out.println(s);
4-        }
5-   public static void main(String[] args){
6-  Procedimento p = new Procedimento();
7-        p.imprime("funcionou!!!");
8-        }
9-}
```

Assinatura do Método:

- É o conjunto formado pelo nome do método e sua lista de parâmetros.

- A assinatura deve ser única para cada método

- O tipo de retorno não faz parte da assinatura.

Sobrecarga de métodos.

- Em uma classe, podemos ter métodos com o mesmo nome, desde que tenham as listas de parâmetros diferentes, isso torna a assinatura diferente.

- A lista de parâmetros é distinta entre si pelo número, tipo e ordem dos parâmetros.

- O compilador é encarregado de encontrar o método correto.

Tipos de Métodos:

- <u>De Instância</u>: são aqueles que só podem ser executados por um objeto. Pode-se fazer o uso da palavra reservada this para manipular o objeto atual.

```
1- class MetodoInstancia{
2-   String s;
3-   void soma(int a, int b){
4-      System.out.println(this.s + a+b);
5-        }
6-   public static void main(String[] args){
7-        MetodoInstancia m = new MetodoInstancia();
8-        MetodoInstancia mi = new MetodoInstancia();
9-        m.s = "a soma e de a + b: ";
10-       m.soma(2,3);
11-       mi.s = "A SOMA DE A + B: ";
12-       mi.soma(4,5);
13-       }
14- }
```

- De Classe: são métodos estáticos (static) não precisam de um objeto para executa-los, porém só podem manipular parâmetros e/ou variáveis de classe (static).

```
1- class MetodoClasse{
2-      static String s = "a * b: ";
3-      static String mult(float a, float b){
4-      return s+a*b;
5-      }
6-      public static void main(String args[]){
7-              System.out.println(MetodoClasse.mult(2,3));
8-      }
9-}
```

Método construtor.

- Às vezes, ao criar um objeto, é necessário definir um estado inicial (variáveis de instância) à ele.

- O método construtor é o responsável pela criação de objetos de uma classe.

- Ele é chamado antes que se possa fazer qualquer manipulação com o objeto.

- Ele não especifica nenhum tipo de retorno, nem mesmo se é void.

- Possui lista de parâmetros.

- Ele possui o mesmo nome da classe, porém diferentemente dos outros métodos, ele começa com a letra maiúscula.

- O método construtor só pode ser chamado com o uso do new.

- Quando não é definido um construtor para uma classe, o compilador cria um para executar funções mínimas necessárias para criar um objeto, como chamada do construtor de uma superclasse.

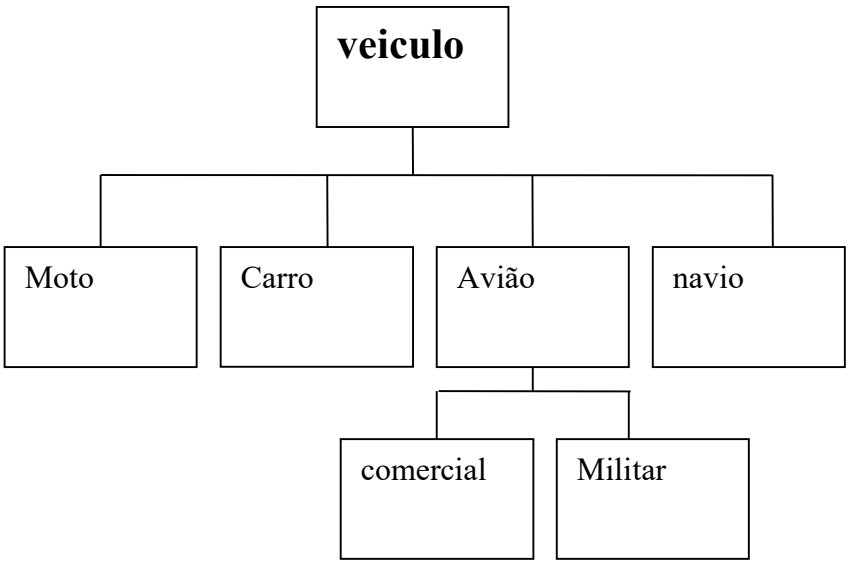
```
1- class Construtor{
2-       int a;
3-       int b;
4-       String s;
5-   Construtor(){
6-       a = 1;
7-       b = 20;
8-       s = "as variáveis a = " + a+", b = "+ b+" , foram iniciadas pelo construtor";
9-       }
10-      public static void main(String args[]){
11-       Construtor c = new Construtor();
12-        System.out.println(c.s);
13-      }
14- }
```

Programa com construtor com lista de parâmetros

```
1- import java.util.Scanner;
2- class Construtor2{
3-       String s;
4-   Construtor2(int x, int y){
5-       s = "o construtor chamado recebeu inteiros como parametros " +x+" ,"+y;
6-       }
7-       Construtor2(Double x, Double y){
 8-        s = "o construtor chamado recebeu pontos flutuantes como parametros "
9-        +x+" ,"+y;
10-      }
11-  public static void main(String args[]){
12-   int a;
13-    int b;
14-    // Double a;
15-   // Double b;
16-       Scanner dado = new Scanner(System.in);
17-      a = dado.nextInt();
18-      b = dado.nextInt();
19-      //a = dado.nextDouble();
20-      //b = dado.nextDouble();
21-              Construtor2 c = new Construtor2(a, b);
22-              System.out.println(c.s);
23-              }
24- }
```

Herança.

- serve para reutilizar classes pré-existentes, sem a necessidade de reescrever código.



- permite uma estrutura hierárquica entre as classes, quanto mais profunda for a classe mais especializada ela é.

- a classe que herda características de outra classe se chama subclasse.

- a classe que está acima de subclasse chama-se superclasse.

- o Java permite apenas herança simples, uma subclasse pode ter apenas uma superclasse.

- para realizar a herança basta colocar a cláusula *extends:*

```
Class MyClass extends MySuperClass{

        .
        .
        .
}
```

- toda a classe obrigatoriamente são subclasses de alguma classe, quando não é especificado a superclasse, a subclasse herda a classe Object.

```
1.  class SuperClasse{
2.   int var;
3.   void imprime(String s){
4.       System.out.println("metodo herdado da superclasse "+s);
5.       }
6.   }
7.  class SubClasse extends SuperClasse{
8.       public static void main(String[] args){
9.           SubClasse sb = new SubClasse();
10.           sb.var = 1;
11.           sb.imprime("pela subclasse");
12.       }
13. }
```

Métodos anulados (Sobre escritos).

-mesmo que um método seja especificado na superclasse, ele pode ser reescrito na subclasse.

-isso permite que as classes possam especializar mais um método que é mais abrangente na sua superclasse.

- quando um método é chamado, o compilador começa a procura a partir da base ao topo, ele executa a primeira que encontrar.

```
1.  class SuperClass{
2.      void imprime(){
3.           System.out.println("se voce ler essa mensagem, o metodo da
    superclasse foi chamado");
4.      }
5.  }
6.  class SubClass{
7.       void imprime(){
8.        System.out.println("se voce ler essa mensagem, o metodo da subclasse
    foi chamado");
9.       }
10. }
11. class MetodoAnulado{
12.      public static void main(String[] args){
13.        SubClass sb = new SubClass();
14.        sb.imprime();
15.     }
16. }
```

Especificadores de acesso.

- Definem qual o tipo de acesso dos membros de uma classe, ou seja, quais outras classes podem acessar os métodos e variáveis de uma classe.

- São colocadas em frente ao nome do método ou variável.

Public: **são membros de classe que podem ser acessados por qualquer outra classe. Ex: o método main deve ser público para que o ambiente de execução possa acessá-lo.**

```
1.  class Teste{
2.      public String s = "essa é uma variável pública";
3.      public void imprime(){
4.        System.out.println("esse método é publico");
5.     }
6.  }
7.     class AcessoPublico{
8.          public static void main(String[] args){
9.          Teste t = new Teste();
10.           System.out.println(t.s);
11.           t.imprime();
12.     }
13. }
```

Obs. Quando duas classes estiverem no mesmo pacote ou no mesmo programa, o uso da palavra public pode ser omitido.

Protected: **os objetos da subclasse tem acesso aos membros protegidos da sua superclasse.**

```
1.  class Teste{
2.       protected String s = "Essa variável é protegida";
3.       protected void imprime(){
4.           System.out.println("esse método é protegido");
5.     }
6.  }
7.  class AcessoProtegido extends Teste{
8.       public static void main(String[] args){
9.          AcessoProtegido ap = new AcessoProtegido();
10.      ap.imprime();
11.       System.out.println(ap.s);
12.     }
13. }
```

Private: **apenas objetos criados dentro da própria classe, ou instruções internas a classe podem acessar membros privados.**

```
1.  class Teste{
2.       private String s = "essa variável é privada";
3.       private void imprime(){
4.           System.out.println("esse método é privado");
5.       }
6.  }
7.       class AcessoPrivado{
8.          public static void main(String[] args){
9.           Teste t = new Teste();
10.           System.out.println(t.s);
11.      }
12. }
```

```
1.  class AcessoPrivado{
2.     private String s = "essa variável é privada";
3.     private void imprime(){
4.          System.out.println("esse método é privado");
5.      }
6.     public static void main(String[] args){
7.        AcessoPrivado t = new AcessoPrivado();
8.         System.out.println(t.s);
9.        t.imprime();
10.    }
11. }
```

UPCASTING.

- em uma hierarquia de classes, uma subclasse é um tipo de uma superclasse.

- quando um objeto de uma subclasse é tratado como um objeto de sua superclasse ocorre upcasting.

- um objeto de uma superclasse não pode ser tratado como um objeto de sua subclasse.

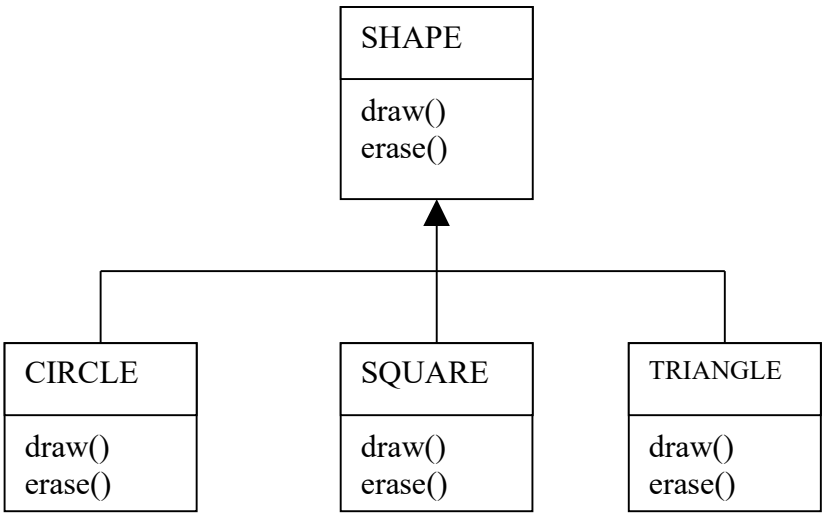
```
1.  class Instrumento{
2.    void afinacao(){
3.       System.out.println("o instrumento esta afinado");
4.     }
5.    void nota(String s){
6.       System.out.println(s);
7.    }
8.  }
9.  class Flauta extends Instrumento{
10.   String tipoFlauta;
11.  }
12. class Violao extends Instrumento{
13.    String tipoViolao;
14.  }
15.     class Musica{
16.        static void nomeInstrumento(Instrumento i){
17.           System.out.println(String.valueOf(i));
18.        }
19.    public static void main(String args[]){
20.     Violao v = new Violao();
21.     Flauta f = new Flauta();
22.     Musica.nomeInstrumento(v);
23.     Musica.nomeInstrumento(f);
24.  }
25. }
```

POLIMORFISMO.

- permite que uma superclasse possa instanciar um objeto de uma subclasse.

- a subclasse deve anular os métodos da superclasse que vai utilizar.

- traz a vantagem de que um único objeto da subclasse pode referenciar qualquer objeto das subclasses e invocar seus métodos.



```
class Shape {
void draw() {}
void erase() {}
}
class Circle extends Shape {
void draw() {
System.out.println("Circle.draw()");
}
void erase() {
System.out.println("Circle.erase()");
}
}
class Square extends Shape {
void draw() {
System.out.println("Square.draw()");
}
void erase() {
System.out.println("Square.erase()");
}
}
class Triangle extends Shape {
void draw() {
System.out.println("Triangle.draw()");
}
void erase() {
System.out.println("Triangle.erase()");
}
}
public class Shapes {
public static void main(String[] args) {


        Shape s = new Triangle();
        s.draw();
        s = new Circle();
        s.draw();
        s = new Square();
        s.draw();
}
} ///:~
```