



Tecnologia em Análise e Desenvolvimento de Sistemas

Programação Orientada a Objetos



©SuhasDwarakanath

1º/2018





Índice

- Detalhes da disciplina – Slide 3
- Introdução a Plataforma Java – Slide 9
- Programação em Java – Slide 21
- Introdução à POO – Slide 33



Detalhes da disciplina



Grupo de discussão no Whatsapp

- Foi criado um grupo no Whatsapp para que os alunos postem dúvidas durante os intervalos de aula.
 - <https://chat.whatsapp.com/6NWaE4igc202YDa8tckynp>





Sobre a disciplina

- Números:
 - 66,7 horas
 - 80 aulas no semestre
 - 4 aulas semanais
- Ementa:
 - Desenvolvimento de sistemas de software baseados no paradigma orientado a objetos.
- Objetivos
 - Tornar o aluno apto a entender e aplicar os conceitos de orientação a objetos no desenvolvimento de sistemas.
- Conteúdo Programático
 - Fundamentos da orientação a objetos. Aplicação dos conceitos de orientação a objetos.



Sobre a disciplina

- BIBLIOGRAFIA BÁSICA:

- DEITEL, P. J.; DEITEL, H. M. Java: como programar. 8. ed. São Paulo: Pearson Education do Brasil, 2010. 1144 p.
- SIERRA, K. Use a cabeça!: Java. 2. ed. Rio de Janeiro: Alta Books, 2010. 484 p. (Use a cabeça!).
- STELLMAN, A.; GREENE, J. Use a cabeça!: C#. 2. ed. Rio de Janeiro: Alta Books, 2011. 797 p. (Use a cabeça!).

- BIBLIOGRAFIA COMPLEMENTAR:

- DEITEL, H. M. et al. C#: como programar. São Paulo: Pearson Education do Brasil, 2003. 1153 p.
- LARMAN, C. Utilizando UML e padrões: uma introdução à análise e ao projeto orientados a objetos e ao desenvolvimento iterativo. 3. ed. Porto Alegre: Bookman, 2007. 695 p.
- MENDES, D. Rocha. Programação Java com ênfase em orientação a objetos. São Paulo: Novatec, 2009. 463 p.
- SANTOS, R. Introdução à programação orientada a objetos usando Java. 2 ed. Rio de Janeiro: Elsevier, 2013. 313 p.
- SHARP, J. Microsoft Visual C# 2010: Passo a passo. Porto Alegre: Bookman, 2011. 775 p.

Sobre a disciplina

- Critério de avaliação
 - Como vocês querem ser avaliados?



Definido pelos alunos em 09/02:

Provas individuais (n, de acordo com o conteúdo) 30%
Exercícios (n, de acordo com o conteúdo) totalizando 40%
Projeto Prático (1, contemplando todo o conteúdo) 30%



Sobre a disciplina Ferramentas 1s2018

- Java Oracle 8
- IntelliJ IDEA



IntelliJ IDEA

2017.1.4



Introdução a Plataforma Java



Como surgiu o Java?

- Em 1991 era grande o interesse da SUN no mercado de dispositivos eletrônicos inteligentes destinados ao consumidor final.
- Projeto Green, que resultou na criação de uma linguagem baseada em C/C++ chamada Oak (carvalho), nome de uma árvore da frente da janela do escritório da SUN.
- Mais tarde descobriu-se que existia uma linguagem de mesmo nome, quando a equipe da SUN visitou uma cafeteria local chamada JAVA, nome do café importado.
- O projeto Green atravessava dificuldades. Por sorte, em 1993 estoura a popularidade da Internet e os pesquisadores da SUN visualizam o potencial do JAVA para Web.



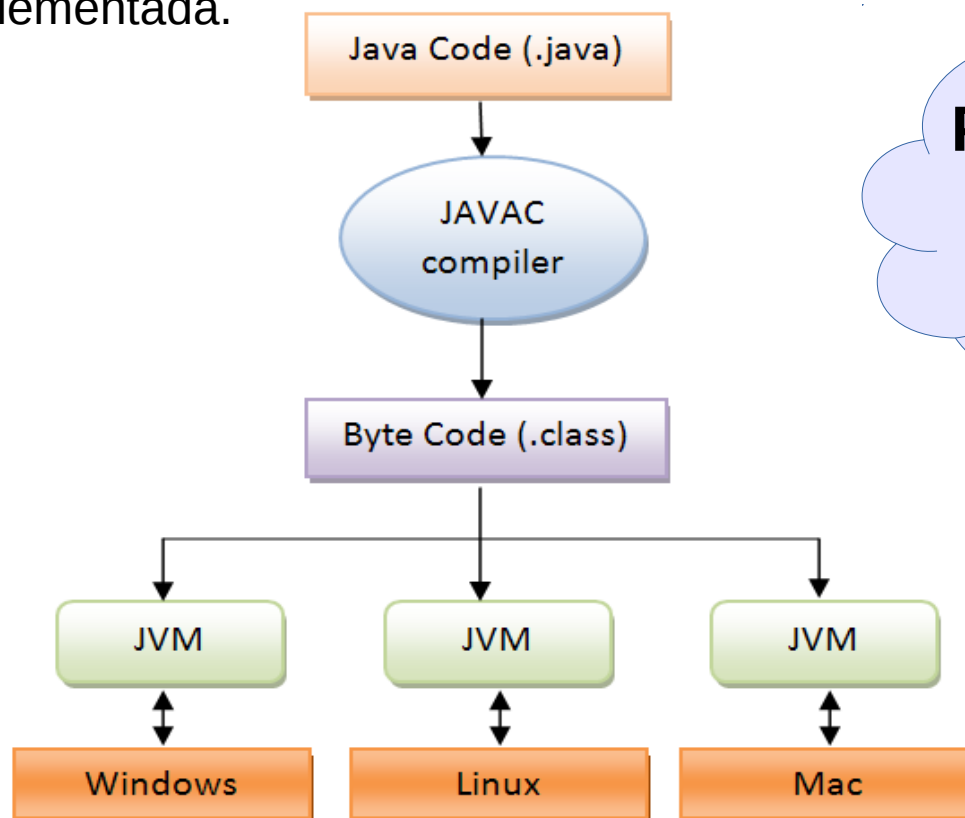


O que é Java?

- É uma linguagem de programação de alto nível cuja principal característica é ser **orientada a objetos**.
- Esta linguagem de programação é compilada para uma linguagem intermediária denominada **Java *bytecode***, que é constituída por instruções que podem ser executadas pela **Plataforma Java**.

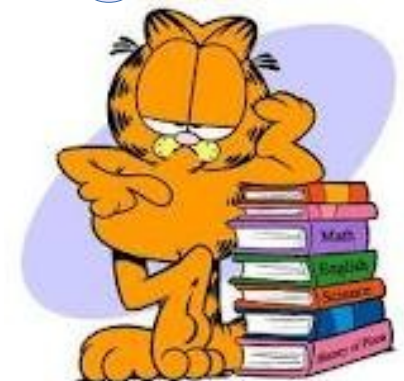
Write once run anywhere. O que isso significa?

- Escreva uma vez e execute em qualquer lugar!
 - Pela característica da plataforma Java, o mesmo código pode ser executado em qualquer hardware que possua uma JVM (Java Virtual Machine – Máquina Virtual Java) implementada.



Portabilidade...

Pesquise!





O que garante a portabilidade do Java?

- *A Java Virtual Machine (JVM);*
 - Pode-se afirmar que para qualquer dispositivo com uma JVM implementada, os programas Java são executados.

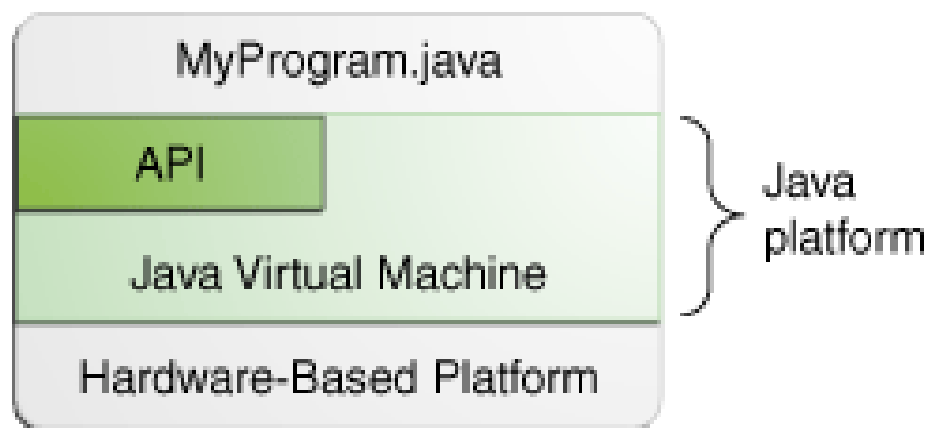


O que se entende por Plataforma?

- Existem vários sistemas operacionais (Linux, MS Windows, MacOS, Solaris, Android, etc) que podem ser executados em hardwares diferentes.
- Uma plataforma é o ambiente de hardware e/ou software onde uma aplicação é executada.
- Normalmente, a plataforma é descrita pela combinação de hardware e sistema operacional.

Plataforma Java (1)

- A plataforma Java é a plataforma onde os Java *bytecodes* são executados.
- Ela é diferente das plataformas usuais pelo fato de ser composta somente de software, que é executado em outra plataforma.





Plataforma Java (2)

- Significa que a plataforma Java, na verdade, interpreta os Java *bytecodes*.
- Cada instrução na forma de *bytecode* deve ser analisada pela JVM e interpretada por ela. Uma vez que isso ocorre, a JVM envia um comando para o hardware executar a instrução de fato.
- A plataforma Java foi desenvolvida para ser segura, possibilitar manipulação de exceções e coleta de lixo (*garbage collection*), que libera, automaticamente, a memória alocada dinamicamente.



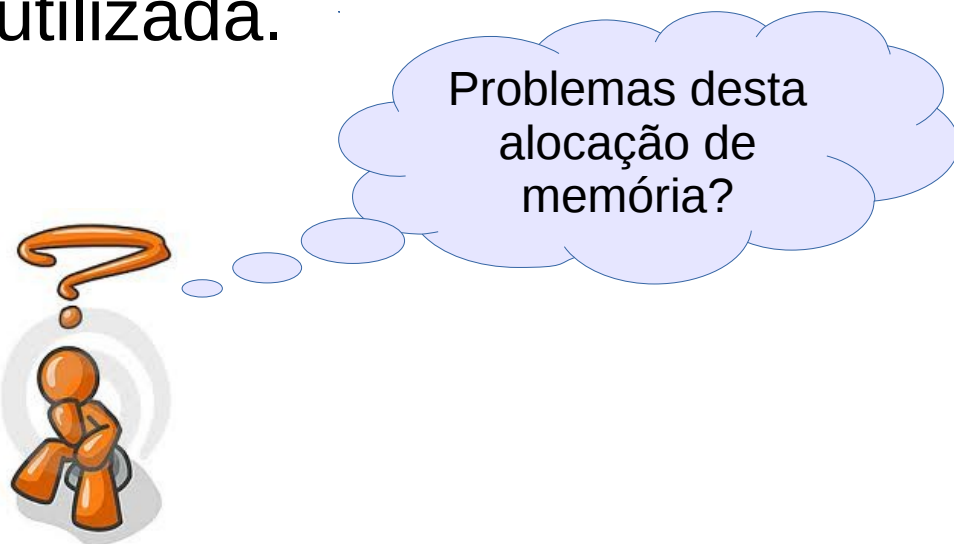
Plataforma Java (3)

- A plataforma Java é composta de duas partes:
 - **Máquina Virtual Java (JVM):** tem como objetivo executar os *bytecodes*, traduzindo-os para o código nativo. Ela pode ser implementada em hardware (processadores dedicados) ou software (JDK);
 - **Interface para Programação de Aplicações (API):** oferece um conjunto de pacotes de classes (*packages*) com funcionalidades semelhantes às bibliotecas de C/C++.

Plataforma Java (4)

Coletor de Lixo

- A maior parte das linguagens possuem alguma instrução que permite ao programador requisitar memória dinamicamente, ou seja, durante a execução do programa.
- Da mesma forma, existe um comando que permite ao programador liberar essa memória quando ela não for mais utilizada.

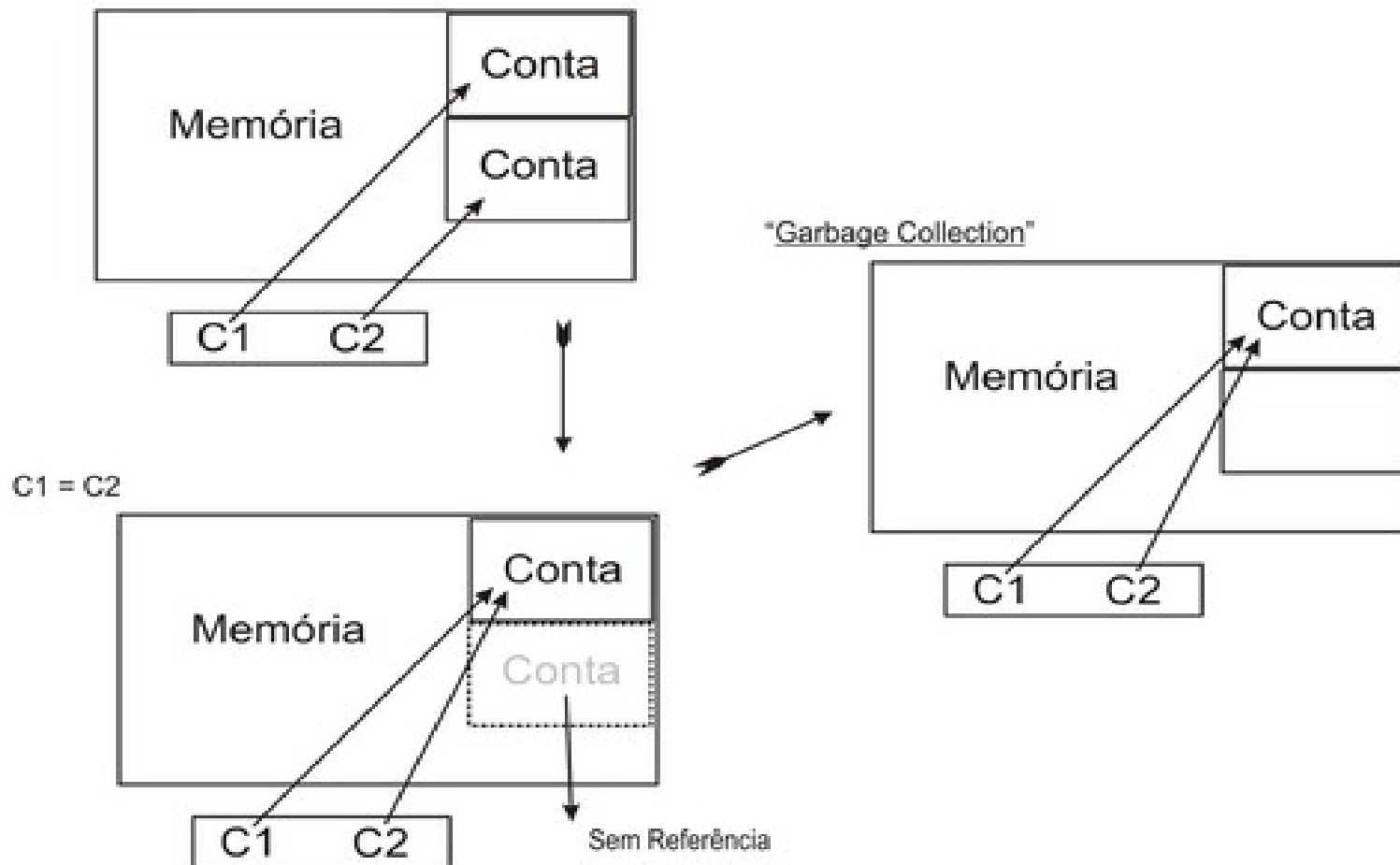




Plataforma Java (5)

Coletor de Lixo

```
Conta c1 = new ContaCorrente();  
Conta c2 = new ContaCorrente();
```





Plataforma Java (6)

Coletor de Lixo

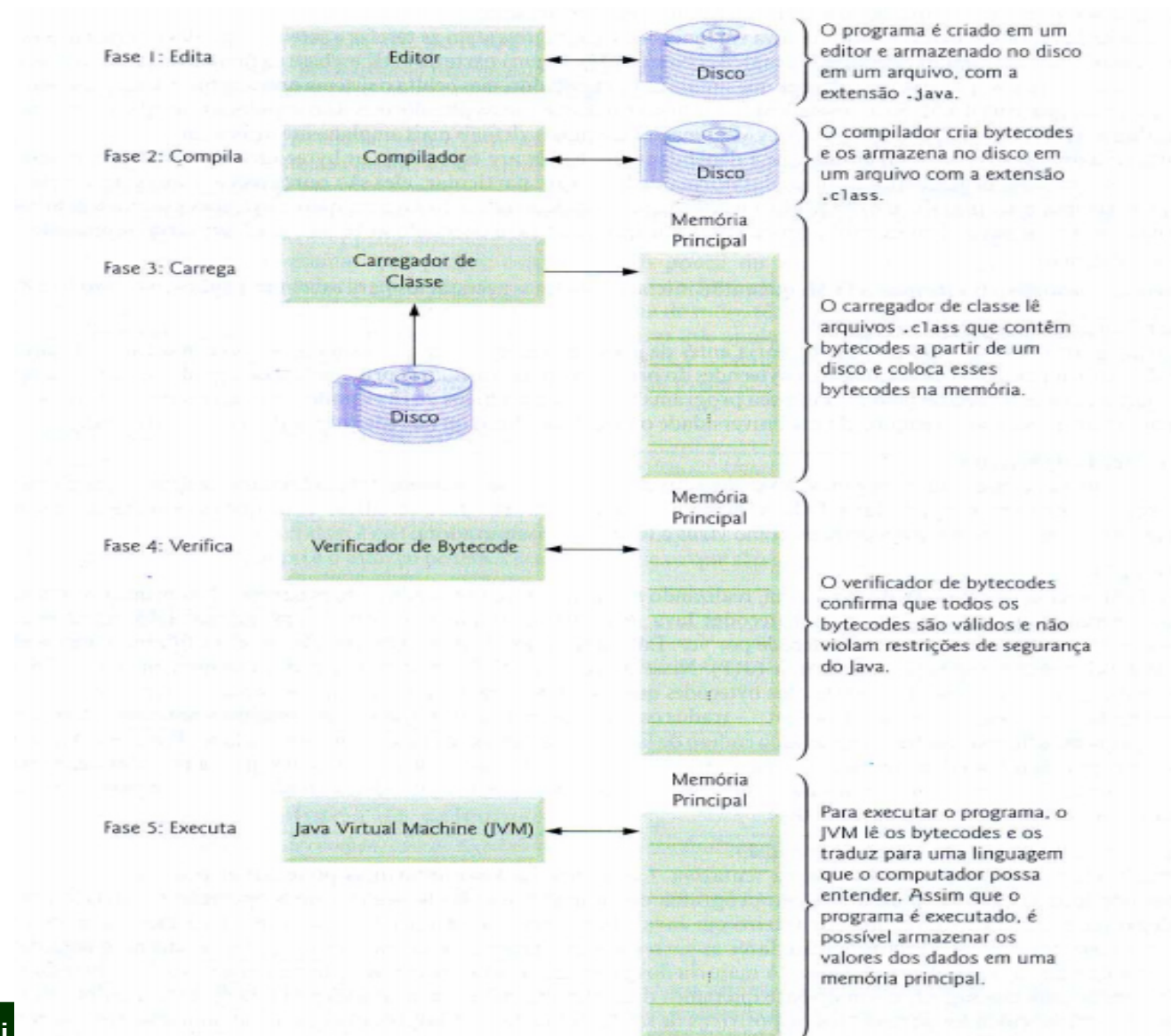
- O Coletor de Lixo faz parte da **Plataforma Java** e não da linguagem Java.
- Errado comparar este aspecto entre Java e C++.



Programação em Java



Ambiente de Desenvolvimento Java





Exemplo 1

- Digitar o texto abaixo em qualquer editor de texto:

```
public class Exemplo1{  
    public static void main(String args[ ]){  
        System.out.println("Caio, Java é melhor que Python...");  
    }  
}
```

- Salvar o arquivo como Exemplo1.java
- Compilar: **javac Exemplo1.java**
- Executar: **java Exemplo1**



Tipos de Dados

TIPO	FAIXA DE VALORES	TAMANHO
byte	-128 a 127	1 byte
char	0 a 65.535	2 bytes
short	-32.768 a 32.767	2 bytes
int	-2.147.483.648 a 1.147.483.648	4 bytes
long	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807	8 bytes
float	$-3,4 \times 10^{-38}$ a $3,4 \times 10^{38}$	4 bytes
double	$-1,7 \times 10^{-308}$ a $1,7 \times 10^{308}$	8 bytes



Saída de Dados

- `System.out.print();`
- `System.out.println();`
- `System.out.printf();`



Entrada de Dados: Classe Scanner (1)

```
import java.util.Scanner;

public class Exemplo2{

    public static void main(String args[]){
        //Instanciar um objeto para leitura
        Scanner input = new Scanner(System.in);

        int ano;

        System.out.printf("Ano de nascimento: ");
        ano = input.nextInt();

        System.out.printf("Idade: %d \n", 2018-ano);
    }
}
```



Entrada de Dados: Classe Scanner (2)

- Principais métodos de leitura:
 - `nextInt();`
 - `nextDouble();`
 - `nextFloat();`
 - `nextLine();`



Entrada de Dados: Classe Scanner (3)

```
import java.util.Scanner;

public class Exemplo3{

    public static void main(String args[]){
        Scanner input = new Scanner(System.in);
        char sexo;
        System.out.printf("Informe sexo (M, F ou I): ");
        sexo = input.nextLine().charAt(0);
        System.out.printf("Sexo digitado: %c \n", sexo);
    }
}
```



Estrutura Condicional

```
if( <condição> )
```

```
{
```

```
}
```

```
else
```

```
{
```

```
}
```



Estrutura de Seleção

```
switch( <variável>){  
    case valor: ...;  
    default ...;  
}
```



Estruturas de Repetição

- while(<condição de parada>){ }
- do ... while (<condição de parada>);
- for(<inícios>;<condição>; <incrementos>){ }



Exercícios

- Exercícios de Fixação 1 e 2.



Introdução à POO



O que é Programação Orientada a Objetos?

- É um paradigma de programação baseado no conceito de classes e objetos.
- As classes são elementos em que dados e procedimentos podem ser agrupados, segundo sua função para um determinado sistema; essas classes são codificadas em formatos de arquivos.
- Quando uma dessas classes é utilizada como um tipo de dado para a criação de uma variável, esta é chamada de objeto.



O que é classe? (1)

- A classe é definida como uma estrutura de dados que contém atributos e métodos.
- Ao se criar uma classe, o objetivo é agrupar métodos e atributos que estejam relacionados entre si.
- Uma classe é composta de partes e estas devem representar alguma funcionalidade segundo o objetivo da classe.



O que é classe? (2)

- As partes de uma classe devem representar funcionalidades para atender o objetivo da classe.
- Exemplo, uma classe Cliente:
 - Quais os dados relacionados a um cliente? (nome, endereço, cidade, estado, etc)
 - Quais outras informações desse cliente devem ser armazenadas?
 - Quais atitudes um cliente pode tomar ou quais atitudes podem ser tomadas diante de um cliente?



O que é classe? (3)

- Concluindo, a classe é o código que declara atributos e métodos, em que cada um destes possa fazer parte da representação de um mesmo objetivo.
- O objetivo da classe é representar de forma adequada uma entidade dentro de um sistema.



O que é objeto? (1)

- Classe é somente a codificação na forma de arquivo texto, um objeto é uma instância de uma classe.
- É uma porção de memória reservada para armazenar os dados e os métodos declarados na classe.
- **Um objeto é a instância de uma classe na memória.**

O que é objeto? (2)

- A classe é o código-fonte escrito em um arquivo texto, enquanto o objeto é uma parte de uma aplicação durante o processo de execução.



= Objeto



= Classe

O que é uma mensagem?

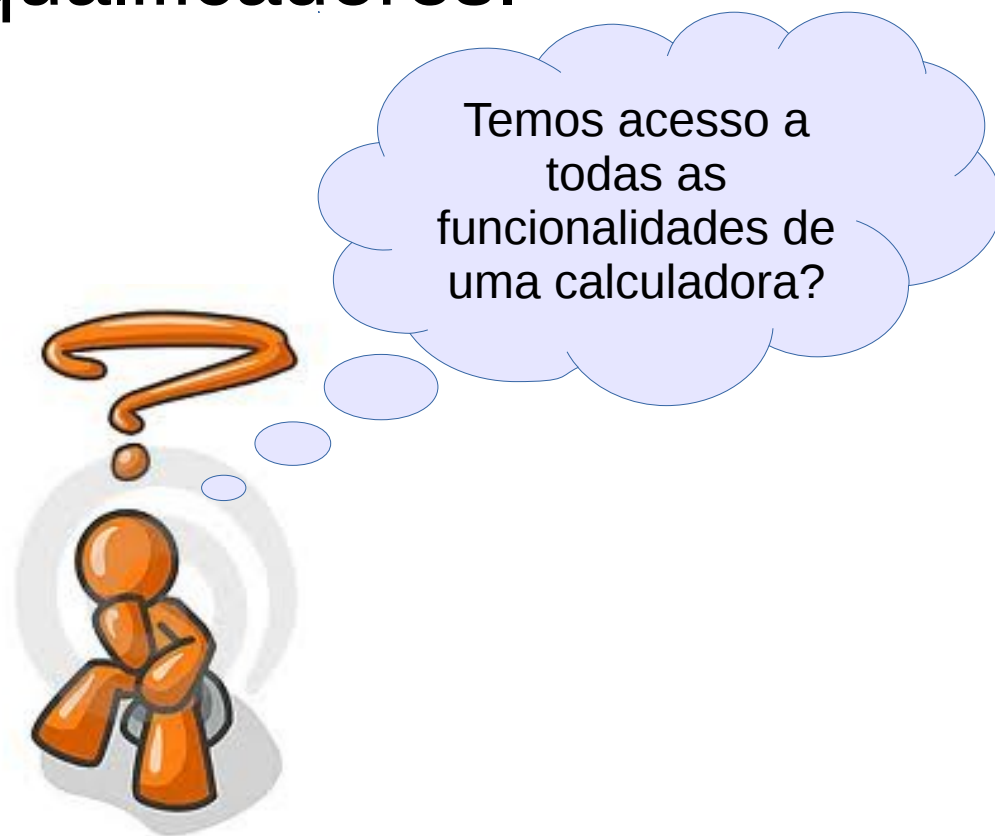
- A mensagem é definida como o ato de chamar ou requisitar a execução de um método.

```
...  
Scanner entrada = new Scanner(System.in).  
int valor;  
valor = entrada.nextInt();  
...
```

Esta é uma mensagem enviada ao método `nextInt()` do objeto `entrada` que está implementado na classe `Scanner`.

O que é Encapsulamento?

- É a capacidade de restringir o acesso a elementos de um objeto utilizando qualificadores.









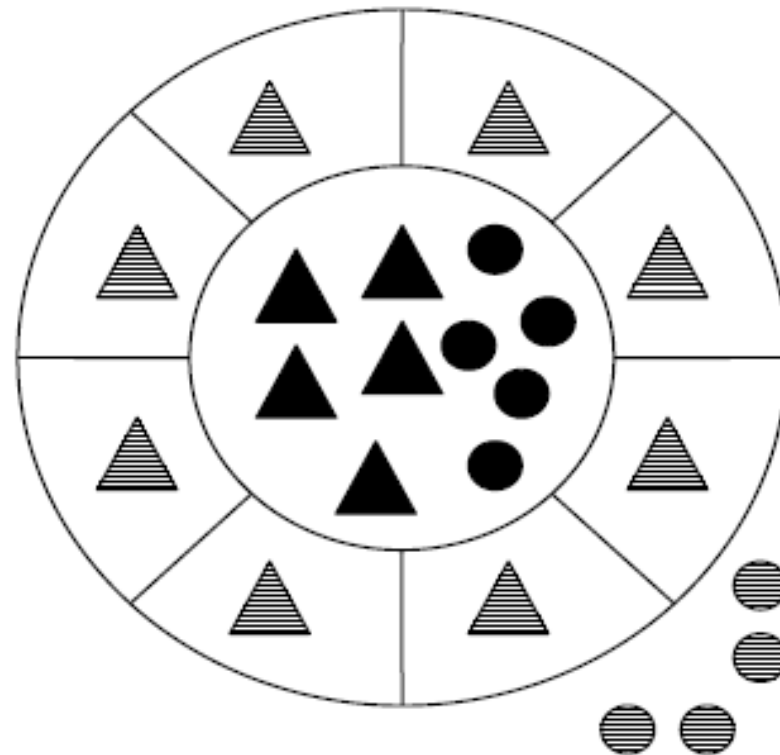
Qualificadores de Acesso

- **private:** o método ou atributo pode ser acessado somente dentro da própria classe;
- **public:** o método ou atributo pode ser acessado externamente por outro código;
- **protected:** o método ou atributo pode ser acessado pela própria classe ou por classes-filhas (herança);
- **package:** o método ou atributo pode ser acessado pela própria classe ou classes que participem do mesmo pacote.

Encapsulamento

CLASSE

-  métodos públicos
-  métodos privados
-  dados privados
-  dados públicos
(não recomendável)





O que são Construtores?

- Construtores são métodos especiais chamados no processo de instanciação de um objeto no sistema.
- A execução destes métodos garante a inicialização dos identificadores de forma correta.
- Um método construtor tem o mesmo nome da classe.



Exemplo

- Para exemplificar a construção de uma classe, iremos construir uma classe que represente uma data.
- Sabemos que uma data é representada por três valores inteiros (dia, mês e ano) que são os atributos da classe.
- A seguir a definição da classe MinhaData com estes atributos.



MinhaData

```
public class MinhaData {  
    private int dia;  
    private int mes;  
    private int ano;  
  
    public MinhaData(int oDia, int oMes, int oAno){  
        dia = oDia;  
        mes = oMes;  
        ano = oAno;  
    }  
  
    public String dataBrazil(){  
        String txt;  
        txt = dia + "/" + mes + "/" + ano;  
        return txt;  
    }  
  
    public String dataUS(){  
        String txt;  
        txt = ano + "-" + mes + "-" + dia;  
        return txt;  
    }  
}
```

A primeira observação é que toda classe é pública, caso contrário a JVM não terá acesso à classe.

Por questão de segurança, todos os atributos estão bloqueados para acesso de outros códigos. Apenas os métodos da própria classe podem acessar e/ou modificar os dados.

O método construtor é encarregado de passar os dados ao objeto. É importante observar a ordem dos argumentos que são passados para o construtor. Observem que não estamos fazendo qualquer verificação nestes dados.

Implementação de dois métodos que retornam uma String com a data.



MinhaData

```
public class MinhaData {  
    private int dia;  
    private int mes;  
    private int ano;  
  
    public MinhaData(int oDia, int oMes, int oAno){  
        dia = oDia;  
        mes = oMes;  
        ano = oAno;  
    }  
  
    public String dataBrazil(){  
        String txt;  
        txt = dia + "/" + mes + "/" + ano;  
        return txt;  
    }  
  
    public String dataUS(){  
        String txt;  
        txt = ano + "-" + mes + "-" + dia;  
        return txt;  
    }  
}
```

A variável txt é declarada em dois métodos. Seria ela um atributo da classe MinhaData?





O Main

```
public class Main {  
    public static void main(String args[]){  
        MinhaData hoje;  
  
        hoje = new MinhaData(23, 02, 2018);  
  
        System.out.println("Hoje no Brasil.: " + hoje.dataBrazil());  
        System.out.println("Hoje nos EUA...: " + hoje.dataUS());  
    }  
}
```

A classe Main hospeda o método main(), o qual inicia nosso sistema.

Foi instanciado o objeto hoje a partir da classe MinhaData. No processo de instanciar o objeto, foi enviada uma mensagem ao método construtor da classe com os argumentos exigidos.

Após a instancia do objeto, os métodos que exibem a data foram chamados.



Atividade

- A classe `MinhaData` está muito simples, implemente as seguintes melhorias:
 - Não permitir que uma data inválida seja instanciada. Caso tente-se instanciar uma data inválida instancie 01/01/1900.
 - Implemente um construtor que aceite várias ordens de argumentos para instanciar uma data. Por exemplo, o mesmo construtor deve receber os seguintes dados: (1, 2, 1900) ou (1900, 2, 1) e instanciar a data 01/02/1900.
 - Implementar um método que retorne a data por extenso (Brazil).
 - Implementar um método que retorne a quantidade de dias entre a data instanciada e uma data passada como argumento.