



Tecnologia em Análise e Desenvolvimento de Sistemas

Programação Orientada a Objetos



1º/2018





Índice

- Detalhes da disciplina – [Slide 3](#)
- Introdução a Plataforma Java – [Slide 9](#)
- Programação em Java – [Slide 21](#)
- Introdução à POO – [Slide 33](#)
- Métodos de acesso e modificadores – [Slide 52](#)
- Sobrecarga – [Slide 62](#)
- This – [Slide 72](#)
- Organização de pacotes – [Slide 76](#)
- Membros estáticos – [Slide 81](#)
- [Arrays](#)



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 interativo. 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?



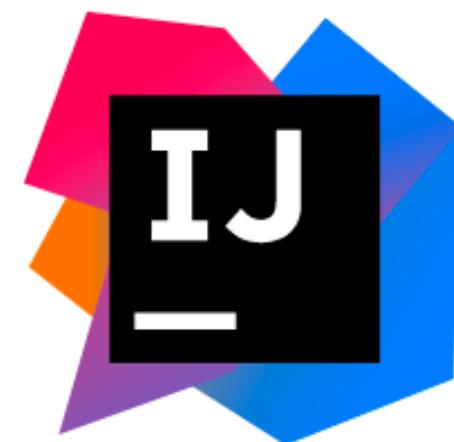
Definido pelos alunos em 09/02:

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.



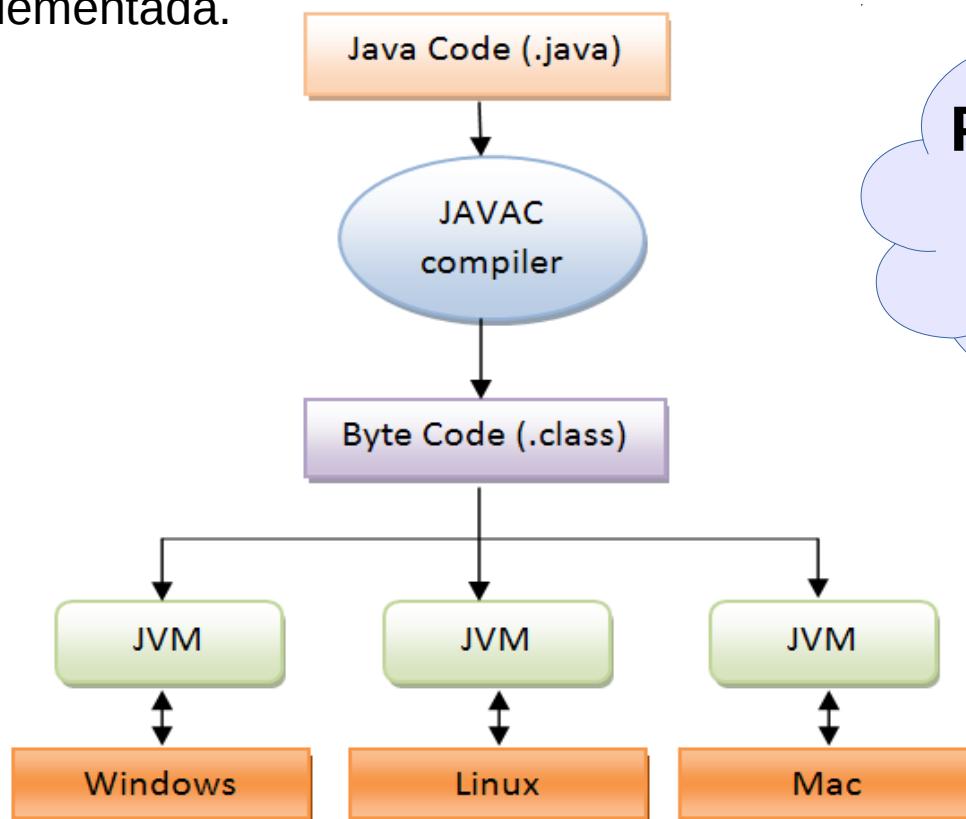


O que é Java?

- 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**.

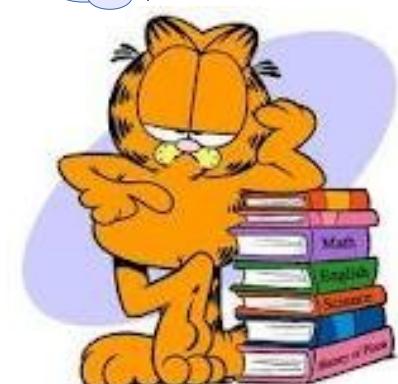
Write once run anywhere. O que isso significa?

- 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.



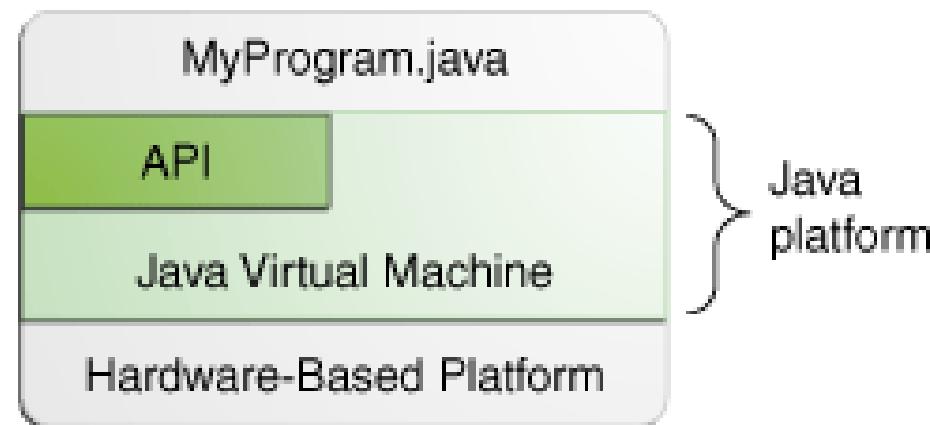
O que se entende por Plataforma?

- 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)

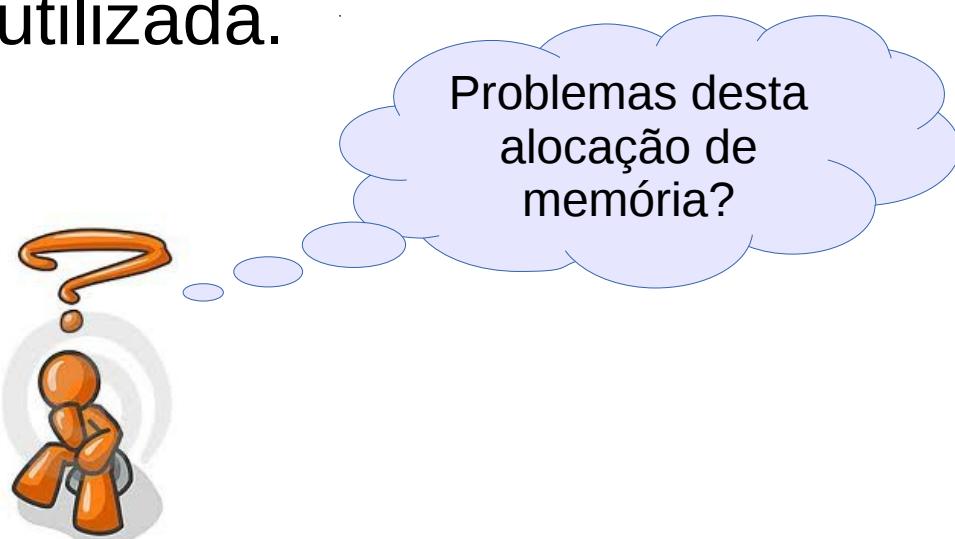
- A plataforma Java é composta de duas partes:
 - **Máquina Virtual Java (JVM)**: tem como objetivo executar os *bytecodes*, traduzindo-os para o código nativo. Ela pode ser implementada em hardware (processadores dedicados) ou software (JDK);
 - **Interface para Programação de Aplicações (API)**: oferece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



Plataforma Java (4)

Coletor de Lixo

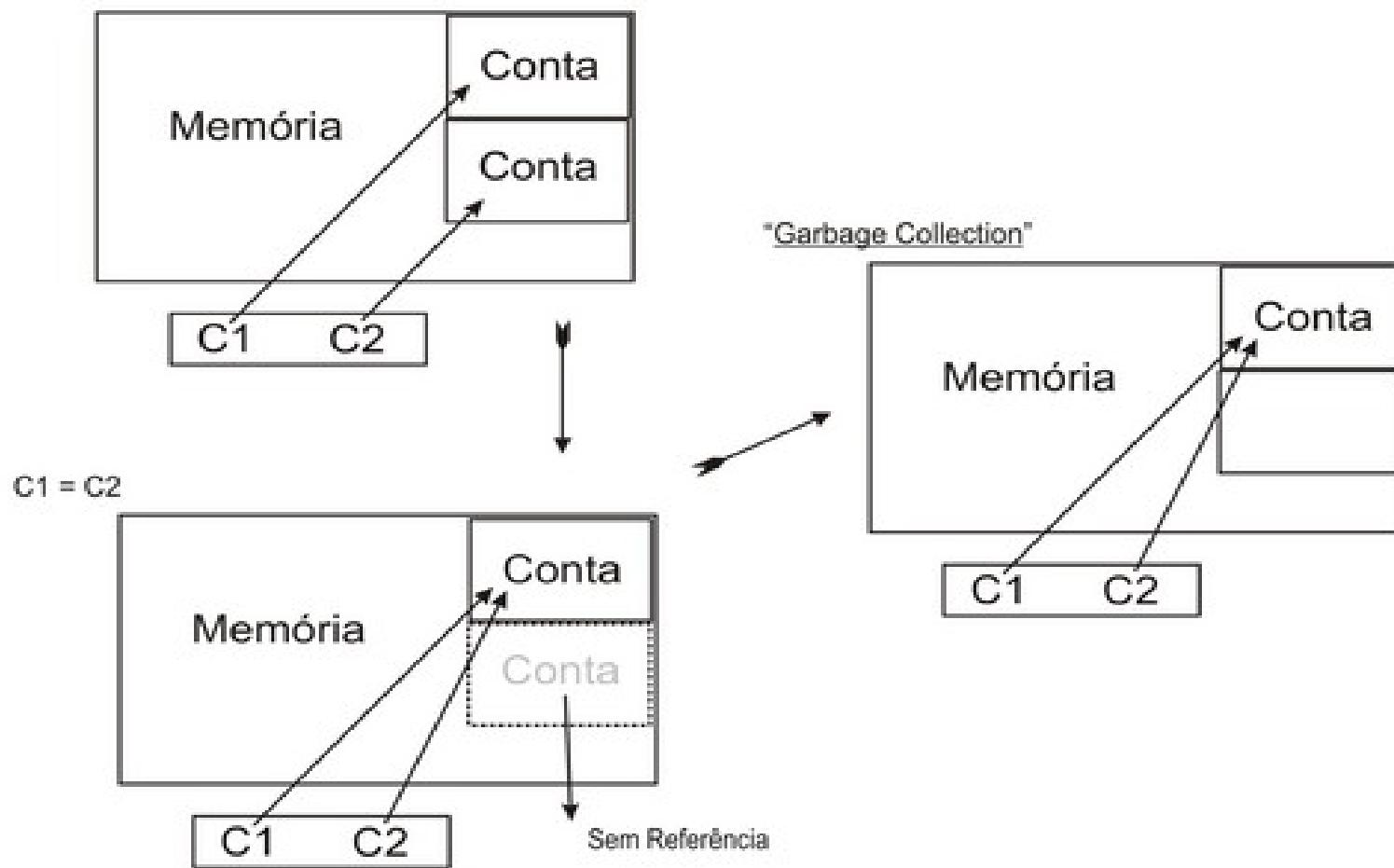
- A maior parte das linguagens possuem alguma instrução que permite ao programador requisitar memória dinamicamente, ou seja, durante a execução do programa.
- Da mesma forma, existe um comando que permite ao programador liberar essa memória quando ela não for mais utilizada.



Plataforma Java (5)

Coletor de Lixo

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





Plataforma Java (6)

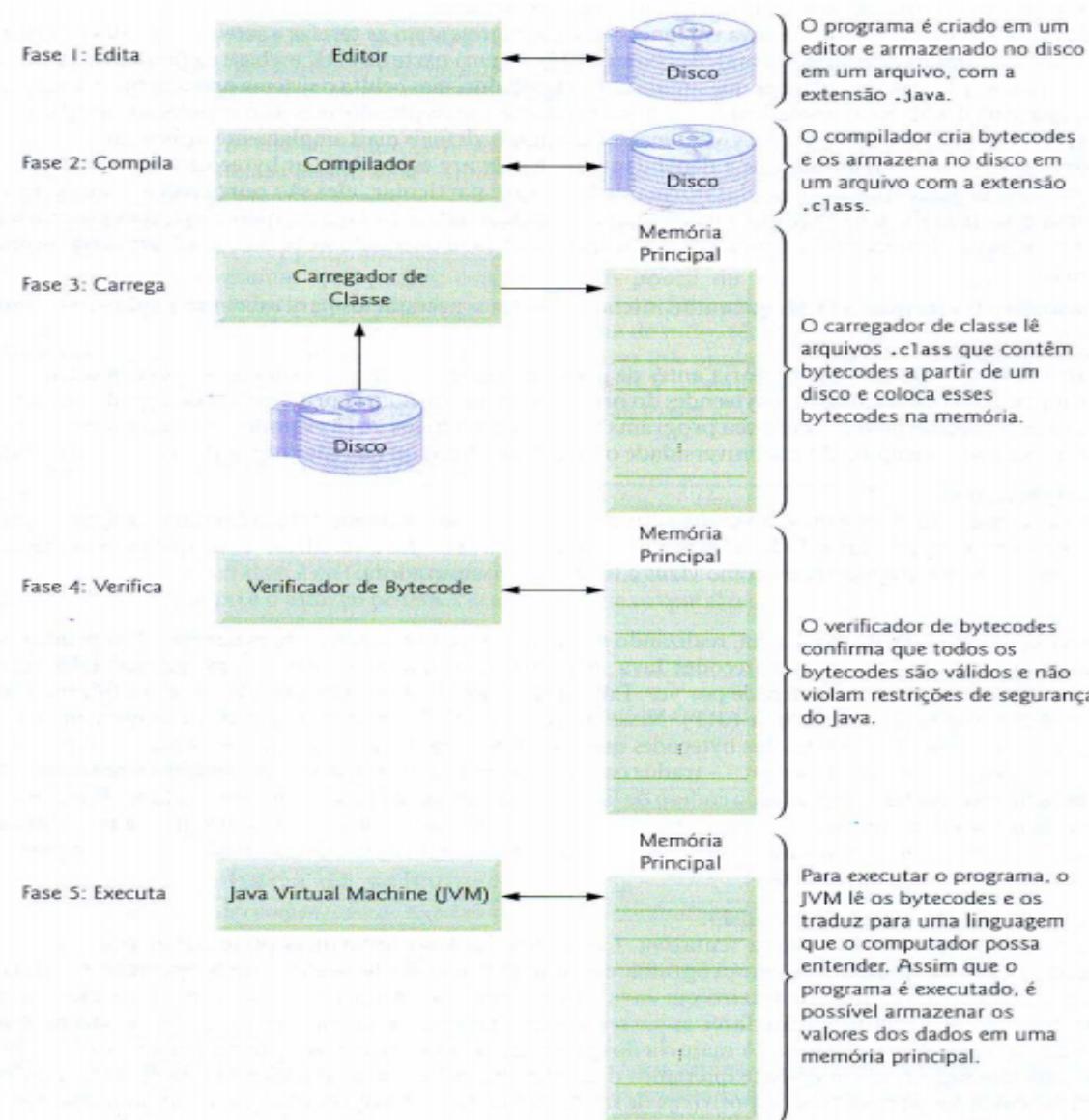
Coletor de Lixo

- 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.

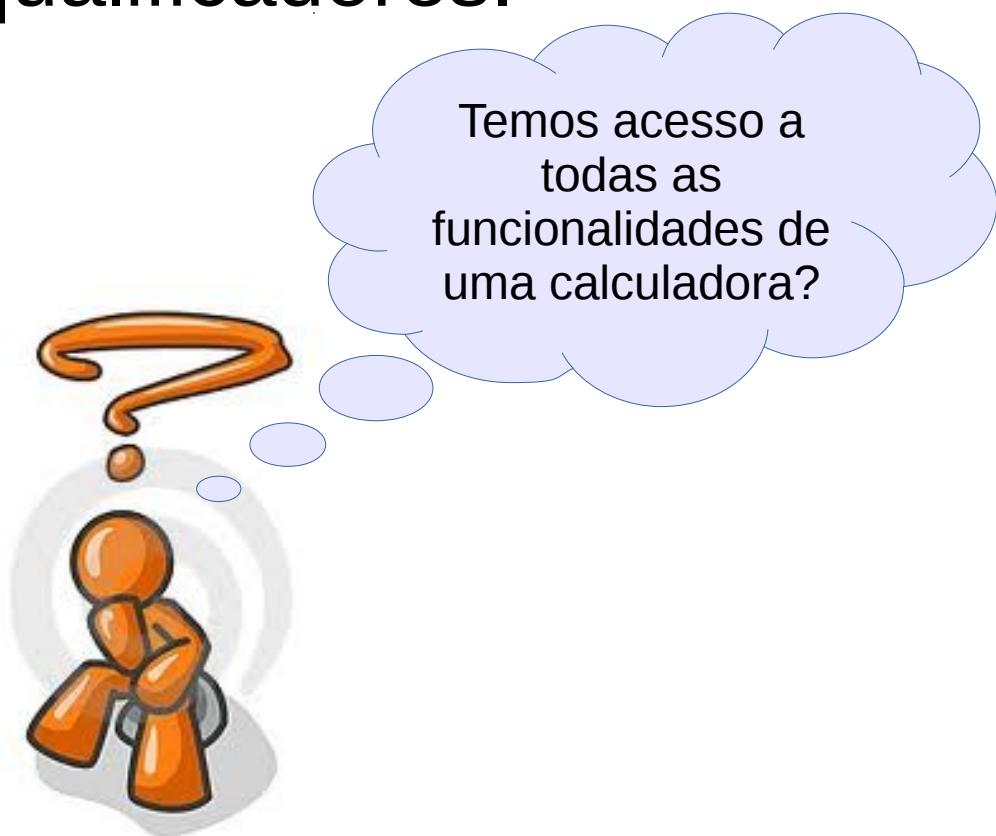
```
...
Scanner entrada = new Scanner(System.in).
int valor;
valor = entrada.nextInt();
...
```

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

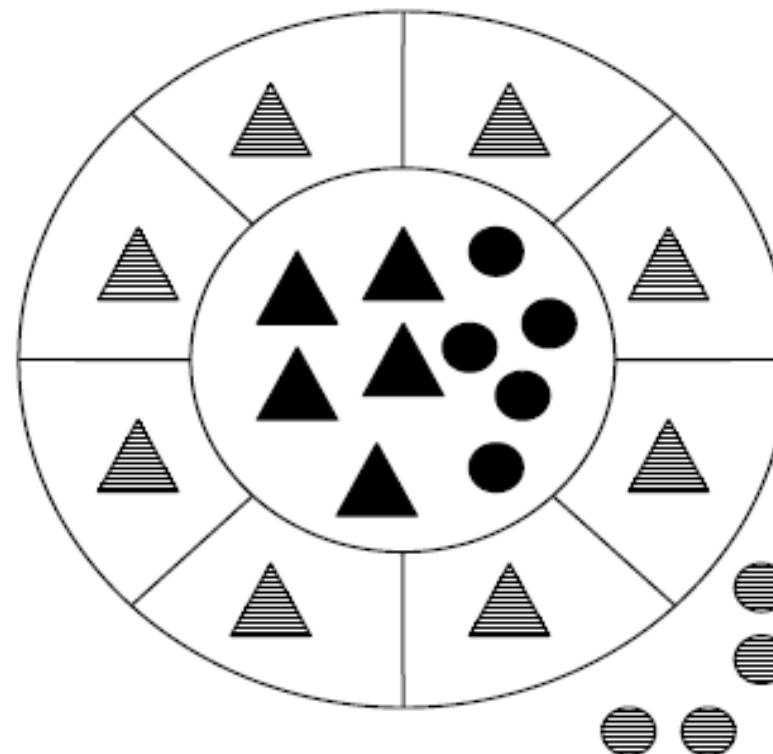
- **private**: o método ou atributo pode ser acessado somente dentro da própria classe;
- **public**: o método ou atributo pode ser acessado externamente por outro código;
- **protected**: o método ou atributo pode ser acessado pela própria classe ou por classes-filhas (herança);
- **package**: o método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



Momento da Revisão





Objetivo da aula

- Estudar os conceitos de POO:
 - Métodos de acesso e modificadores;
 - Sobrecarga;
 - Organização de pacotes.
- Metodologia:
 - Desenvolvimento de exemplos para contextualização do problema e introdução de novos conceitos.



Métodos de Acesso e Modificadores



Exemplo 5

- Vamos implementar um programa que leia nome, salário e valor do reajuste salarial.
- Os dados devem ser inseridos em um objeto.
- O programa deve apresentar o salário reajustado.



Classe Funcionario

```
public class Funcionario {  
  
    private String nome;  
    private double salario;  
    private int reajuste;  
  
    public Funcionario(String argNome, double argSalario, int argReajuste) {  
        nome = argNome;  
        salario = argSalario;  
        reajuste = argReajuste;  
    }  
  
    public double salarioReajustado(){  
        salario *= 1 + (reajuste/100.0);  
        return salario;  
    }  
}
```

Métodos de acesso, falar iremos.



O que há de errado com esse encapsulamento dessa classe?



Os atributos do objeto são private e por isso inacessíveis. Como recuperar os dados depois de instanciar um objeto?



Métodos de acesso e modificadores

- **getXxx()**
 - Para recuperarmos o valor de um atributo, que pode ser recuperado, utilizamos os métodos gets().
 - Nem todos os atributos possuem get() ou método get() público.
- **setXxx()**
 - Ao contrário do get() o método set() altera o valor do atributo.
 - Esse método é responsável pela validação e/ou formatação dos valores que serão inseridos nos atributos de nosso objeto.



Classe Funcionario

```
public class Funcionario {  
    private String nome; private double salario; private int reajuste;  
    public Funcionario(String argNome, double argSalario, int argReajuste) { ... }  
    public double salarioReajustado(){ ... }  
  
    public void setNome(String argNome) {  
        nome = argNome.toUpperCase();  
    }  
    public void setSalario(double argSalario) {  
        salario = argSalario >= 0? argSalario : 0;  
    }  
    public void setReajuste(int argReajuste) {  
        if(argReajuste >= 0)  
            reajuste = argReajuste;  
        else  
            reajuste = 0;  
    }  
    public String getNome() { return nome; }  
  
    public double getSalario() { return salario; }  
  
    public int getReajuste() { return reajuste; }  
}
```

Os métodos de acesso podem ser utilizados fora da classe e pelos próprios métodos da classe.

Por exemplo, seria inconsistente se ao criarmos um objeto o nome fosse minúsculo e ao alterar o nome ele fosse maiúsculo. Mas não precisamos repetir código :-D





Classe Funcionario

```
public class Funcionario {  
  
    private String nome;  
    private double salario;  
    private int reajuste;  
  
    public Funcionario(String argNome, double argSalario, int argReajuste) {  
        setNome(argNome);  
        setSalario(argSalario);  
        setReajuste(argReajuste);  
    }  
  
    public double salarioReajustado(){  
        setSalario(getSalario() * (1 + (getReajuste()/100.0)));  
        return getSalario();  
    }  
  
    //Gets e Sets  
}
```

Como os métodos sets() configuram e validam os argumentos antes de atribuírem aos atributos, podemos utilizá-los, inclusive, no método construtor.

Esse método que retorna o salário reajustado está uma graça!





Classe Funcionario

```
funcionario = new Funcionario(nome, sal, reaj);  
  
System.out.println("Salario reajustado de " + nome + ": " +  
    funcionario.salarioReajustado());  
  
System.out.println("Salario reajustado de " + nome + ": " +  
    funcionario.salarioReajustado());  
  
System.out.println("Salario reajustado de " + nome + ": " +  
    funcionario.salarioReajustado());
```

```
Nome:  
Michel  
Salario:  
1000  
Reajuste:  
10  
Salario reajustado de Michel: 1100.0  
Salario reajustado de Michel: 1210.0  
Salario reajustado de Michel: 1331.0
```

```
Process finished with exit code 0
```

Sempre que invoca-se o método para recuperar o salário reajustado o salário é reajustado novamente.

Pense, reajustar o salário é o mesmo que recuperar o salário do objeto Funcionário?





Classe Funcionario

```
public class Funcionario {  
  
    private String nome;  
    private double salario;  
  
    public Funcionario(String argNome, double argSalario) {  
        setNome(argNome);  
        setSalario(argSalario);  
    }  
  
    public void reajustarSalario(int reajuste){  
        if(reajuste > 0)  
            salario += salario * reajuste/100;  
    }  
  
    //Gets e Sets  
}
```

Reajustar o salário é uma ação que ocorre de tempo em tempos com o objeto, não uma ação corriqueira.

Além disso, o reajuste mudará (quase) sempre que o reajuste for aplicado, assim, não faz sentido que o reajuste seja um atributo do objeto funcionário.





Classe Funcionario

```
funcionario = new Funcionario(nome, sal);
funcionario.reajustarSalario(reaj);
System.out.println("Salario reajustado de " + nome + ": " +
    funcionario.getSalario());
System.out.println("Salario reajustado de " + nome + ": " +
    funcionario.getSalario());
System.out.println("Salario reajustado de " + nome + ": " +
    funcionario.getSalario());
```

```
Nome:
Michel
Salario:
1000
Reajuste:
10
Salario reajustado de Michel: 1100.0
Salario reajustado de Michel: 1100.0
Salario reajustado de Michel: 1100.0

Process finished with exit code 0
```

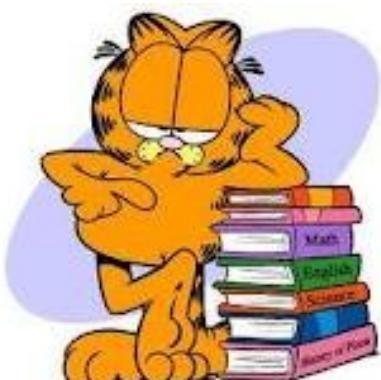
Agora faz sentido!





Material adicional

- Leitura Obrigatória:
 - Unidade 1 - Programação orientada a objetos / organizador Rafael Félix. - São Paulo: Pearson, 2016
- Videoaulas
 - <https://www.youtube.com/watch?v=g2x9oyBFSc0>
 - https://www.youtube.com/watch?v=6i-_R5cAcEc





Sobrecarga



Melhorando o Exemplo 5

- Minha empresa tem 10 funcionários.



Método main()

```
public static void main(String args[]){
    Scanner scanner;
    Funcionario funcionario;
    String nome;
    double sal;
    int reaj, i;
    scanner = new Scanner(System.in);

    for(i=0; i<10; i++){
        System.out.println("Funcionário " + (i+1));
        System.out.println("Nome: ");
        nome = scanner.nextLine();
        System.out.println("Salario: ");
        sal = scanner.nextDouble();
        System.out.println("Reajuste: ");
        reaj = scanner.nextInt();
        funcionario = new Funcionario(nome, sal);
        funcionario.reajustarSalario(reaj);
        System.out.println("Salario reajustado de " + nome + ": " +
                           funcionario.getSalario());
    }
}
```

Assim.



Método main()

```
public static void main(String args[]){
    Scanner scanner;
    Funcionario funcionario;
    String nome;
    double sal;
    int reaj, i;
    scanner = new Scanner(System.in);

    funcionario = new Funcionario();

    for(i=0; i<10; i++){
        System.out.println("Funcionário " + (i+1));
        System.out.println("Nome: ");
        nome = scanner.nextLine();
        System.out.println("Salario: ");
        sal = scanner.nextDouble();
        System.out.println("Reajuste: ");
        reaj = scanner.nextInt();
        funcionario.setNome(nome);
        funcionario.setSalario(sal);
        funcionario.reajustarSalario(reaj);
        System.out.println("Salario reajustado de " + nome + ": " +
                           funcionario.getSalario());
    }
}
```

Ou assim?

Qual a diferença?



Construtor,
diferente está!



De sobrecarga
falaremos.



Sobrecarga (overloading)

```
public class ExemploSobrecarga{
    public static void main(String args[]){
        int n1, n2, n3;
        String str1, str2, str3;

        n1 = 5;
        n2 = 10;
        str1 = "Python é ";
        str2 = "recurso alternativo de engenharia avançada.';

        n3 = n1 + n2;
        str3 = str1 + str2;

        System.out.println(n3);
        System.out.println(str3);
        System.out.println(str1 + n2);
    }
}
```

A screenshot of a terminal window titled 'ednilsonrossi@Dell-Inspiron-5557 ~/02-POOS3/POOS3_2018s1/Exemplos/Exemplo6'. The window shows the command 'java ExemploSobrecarga' being run, followed by the output of the program which prints '15', 'Python é recurso alternativo de engenharia avançada.', and 'Python é 10'.

```
ednilsonrossi@Dell-Inspiron-5557 ~/02-POOS3/POOS3_2018s1/Exemplos/Exemplo6
Arquivo Editar Ver Pesquisar Terminal Ajuda
ednilsonrossi@Dell-Inspiron-5557 ~/02-POOS3/POOS3_2018s1/Exemplos/Exemplo6 $ java ExemploSobrecarga
15
Python é recurso alternativo de engenharia avançada.
Python é 10
ednilsonrossi@Dell-Inspiron-5557 ~/02-POOS3/POOS3_2018s1/Exemplos/Exemplo6 $
```

Sobrecarga (overloading)

```
public class Funcionario {  
  
    private String nome;  
    private double salario;  
  
    public Funcionario() {  
    }  
  
    public Funcionario(String argNome, double argSalario) {  
        setNome(argNome);  
        setSalario(argSalario);  
    }  
}
```





Sobrecarga (overloading)

- Na programação orientada a objetos, um método aplicado a um objeto é selecionado para execução através da sua assinatura e da verificação a qual classe o objeto pertence.
- Através do mecanismo de sobrecarga (overloading), dois métodos de uma mesma classe podem ter o mesmo nome, desde que suas listas de argumentos sejam diferentes, constituindo assim uma assinatura diferente.
- Tal situação não gera conflito pois o compilador é capaz de detectar qual método deve ser escolhido a partir da análise dos tipos de argumentos do método.



Sobrecarga (overloading)

- Por fim, sobrecarregar um método (construtor ou não) é permitir que se utilize o mesmo nome de chamada para operar sobre uma variedade de tipos de dados.
- Podemos ter vários métodos com o mesmo nome em uma mesma classe, porém os tipos dos argumentos devem ser diferentes.



Sobrecarga (overloading)

- Em uma classe qualquer podemos ter dois métodos `setValor()`:
 - `public void setValor(int varA){ }`
 - `public void setValor(float varB){ }`
- Observe que o tipo dos argumentos é diferente.
- Diferente deste exemplo:
 - `public void setValor(int varA){ }`
 - `public void setValor(int varB){ }`

Neste segundo exemplo há um erro, pois o nome dos argumentos é diferente mas o tipo é o mesmo.





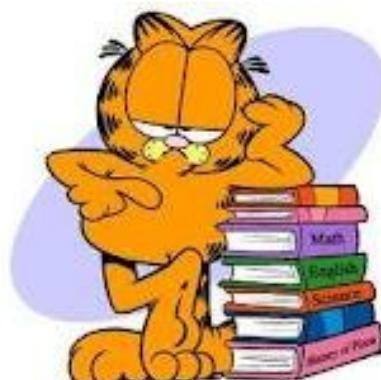
Material adicional

- **Leitura Obrigatória**

- Capítulo 6.12 → Deitel, P.; Deitel, H. Java como programar. 10 ed. São Paulo: Pearson Education do Brasil, 2017.

- **Videoaulas**

- https://www.youtube.com/watch?v=ZpssJov_5_A
 - Aula que trata sobre sobrecarga
- <https://www.youtube.com/watch?v=YvTiyjeXJZU>
 - Aula que trata sobre sobrecarga
- <https://www.devmedia.com.br/sobrecarga-e-sobreposicao-de-metodos-em-orientacao-a-objetos/33066>
 - Artigo que trata sobre sobrecarga e reescrita de métodos. Reescrita será assunto futuro!
- https://www.youtube.com/watch?v=uq4O__CGPdo
 - Aula que trata sobre sobrecarga de construtor





this



this

- Quando um método é chamado, ele recebe automaticamente um argumento implícito, que é a referência ao objeto chamador.
- Esta referência se chama **this**.



```
public class Funcionario {  
    private String nome;  
    private double salario;  
  
    public Funcionario() {}  
  
    public Funcionario(String nome, double salario) {  
        setNome(nome);  
        setSalario(salario);  
    }  
  
    public void reajustarSalario(int reajuste){  
        if(reajuste > 0)  
            this.salario += this.salario * reajuste/100;  
    }  
  
    public void setNome(String nome) {  
        this.nome = nome.toUpperCase();  
    }  
  
    public void setSalario(double salario) {  
        this.salario = salario >= 0? salario : 0;  
    }  
  
    public String getNome() { return this.nome; }  
  
    public double getSalario() {  
        return this.salario;  
    }  
}
```

Nesta implementação sempre que nos referimos aos atributos da classe utilizamos o **this** antes do atributo.

Não precisamos ficar inventando nomes para os argumentos dos métodos.

No futuro outras utilidades :-D

Como funciona o escopo das variáveis e métodos?





Material adicional

- Videoaula
 - <https://www.youtube.com/watch?v=RLzR--Pwvcs>
 - <https://www.youtube.com/watch?v=f6zbLCwq71w>

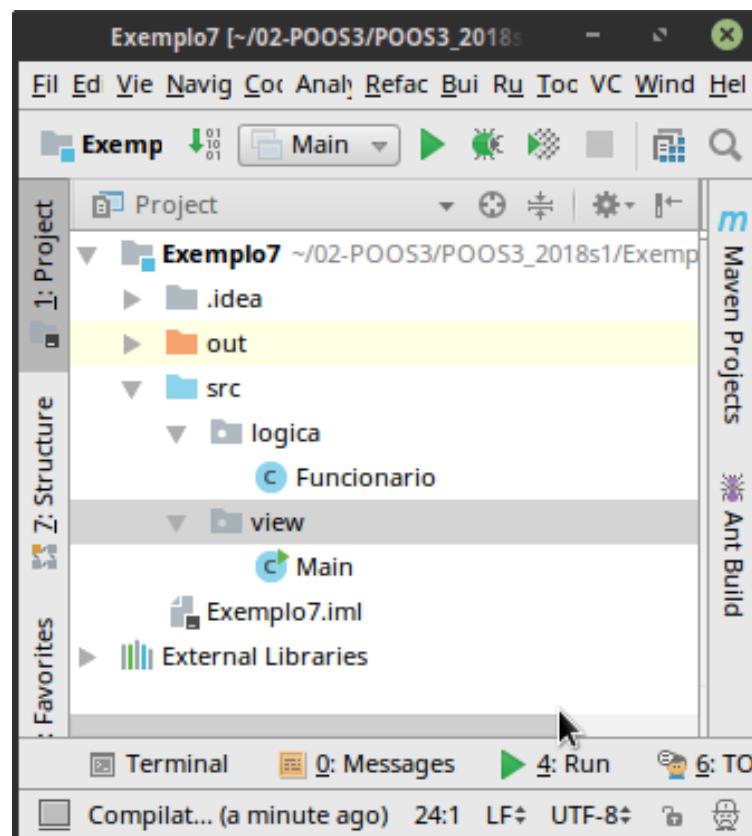




Organização pacotes



Organização de pacotes



No exemplo anterior implementamos duas classes: Funcionario e Main, ambas dentro de SRC.

Como nossos projetos aumentarão de tamanho em breve, vamos adotar o hábito de organizar nossa implementação em pacotes.

Nessa implementação dividiu-se as classes em dois **pacote**, um com a lógica do software e outra com a visualização dele.

Vamos analisar o exemplo 7 e entender um pouco mais sobre packages.





packages

```
package logica;  
  
public class Funcionario { ... }
```

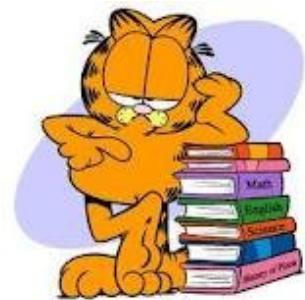
```
package view;  
  
import logica.Funcionario;  
import javax.swing.*;  
  
public class Main { ... }
```

Java está totalmente organizado em pacotes (packages). Observe que a classe Main **importa** o pacote javax.swing.* para poder utilizar recursos de interface gráfica com o usuário. Além disso, por estarem em pacotes diferentes, a classe Main também importa do nosso pacote logica a classe Funcionario.





Material adicional



- Apostila Caelum:
 - <https://www.caelum.com.br/apostila-java-orientacao-objetos/pacotes-organizando-suas-classes-e-bibliotecas/#import>
- Videoaula
 - <https://www.youtube.com/watch?v=aRQHjfYBpM8>
 - <https://www.youtube.com/watch?v=jlfPvg2s-dg>
 - <https://www.youtube.com/watch?v=u1Nd4UIGJel>
- Entrada e saída por JOptionPane
 - <https://pt.scribd.com/document/32741678/Entrada-e-Saida-de-dados-por-JOptionPane>



Trabalhando

- Exercícios de Fixação
 - 3, 4 e 5
- Exercícios Avaliativos
 - 1, 2 e 3





Membros estáticos



Memória

caio

caio.nome = "Caio"

frodo

frodo.nome = "Gabriel"

Cada um dos objetos possui seus atributos e métodos.

Invocar caio.getNome() executa um método enquanto frodo.getNome() executa outro método, ou seja, cada instância é independente da outra. Dados e métodos não são compartilhado.



```
public class Estudante {  
  
    private String nome;  
  
    public Estudante(String nome) {  
        this.nome = nome;  
    }  
  
    public String getNome() {  
        return nome;  
    }  
  
    public void setNome(String nome) {  
        this.nome = nome;  
    }  
  
    public static void main(String[] args) {  
        Estudante caio;  
        Estudante frodo;  
  
        caio = new Estudante("Caio");  
        frodo = new Estudante("Gabriel");  
  
        System.out.println(caio.getNome());  
        System.out.println(frodo.getNome());  
    }  
}
```

```

public class Aluno {

    private String nome;
    private static int quantidade=0;

    public Aluno(String nome) {
        this.nome = nome;
        quantidade++;
    }

    public String getNome() {
        return nome;
    }

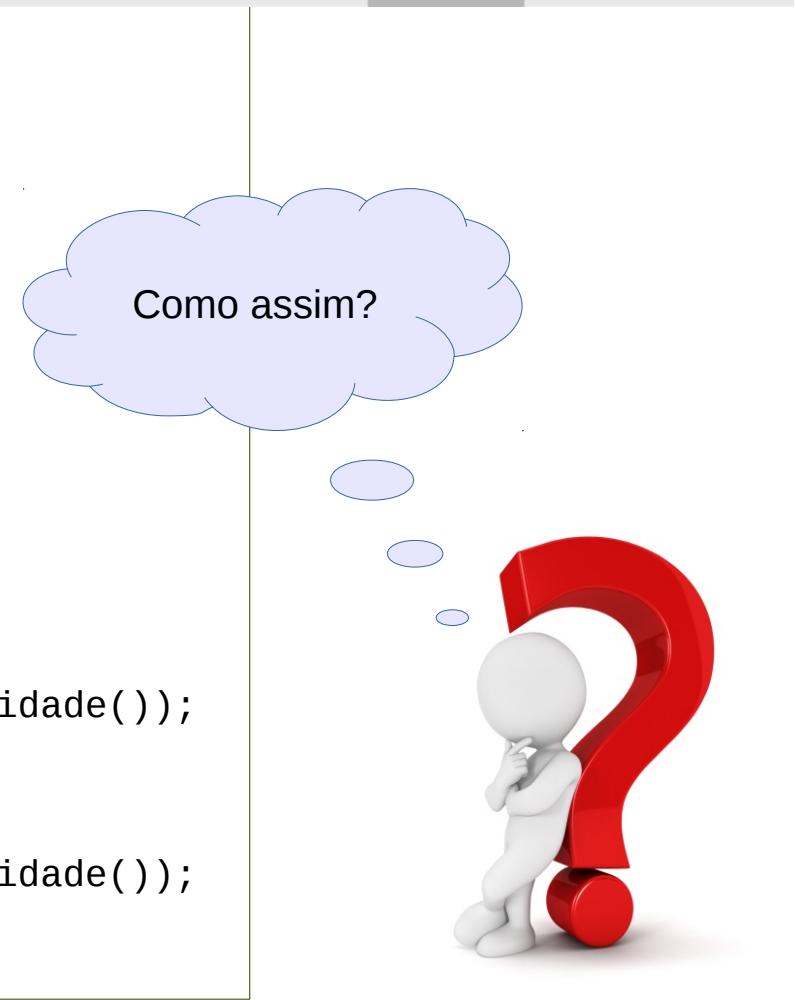
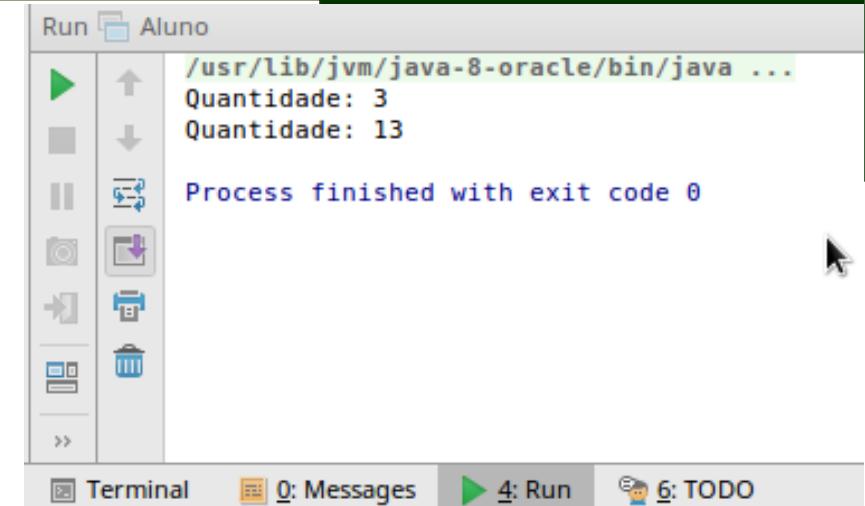
    public void setNome(String nome) {
        this.nome = nome;
    }

    public static int getQuantidade() {
        return quantidade;
    }

    public static void main(String[] args) {
        Aluno a1 = new Aluno("José");
        Aluno a2 = new Aluno("João");
        Aluno a3 = new Aluno("Caio");
        Aluno aN;

        System.out.println("Quantidade: " + Aluno.getQuantidade());
        for(int i=0; i<10; i++){
            aN = new Aluno("Gasparzinho");
        }
        System.out.println("Quantidade: " + Aluno.getQuantidade());
    }
}

```



```

public class Aluno {

    private String nome;
    private static int quantidade=0;

    public Aluno(String nome) {
        this.nome = nome;
        quantidade++;
    }

    public String getNome() {
        return nome;
    }

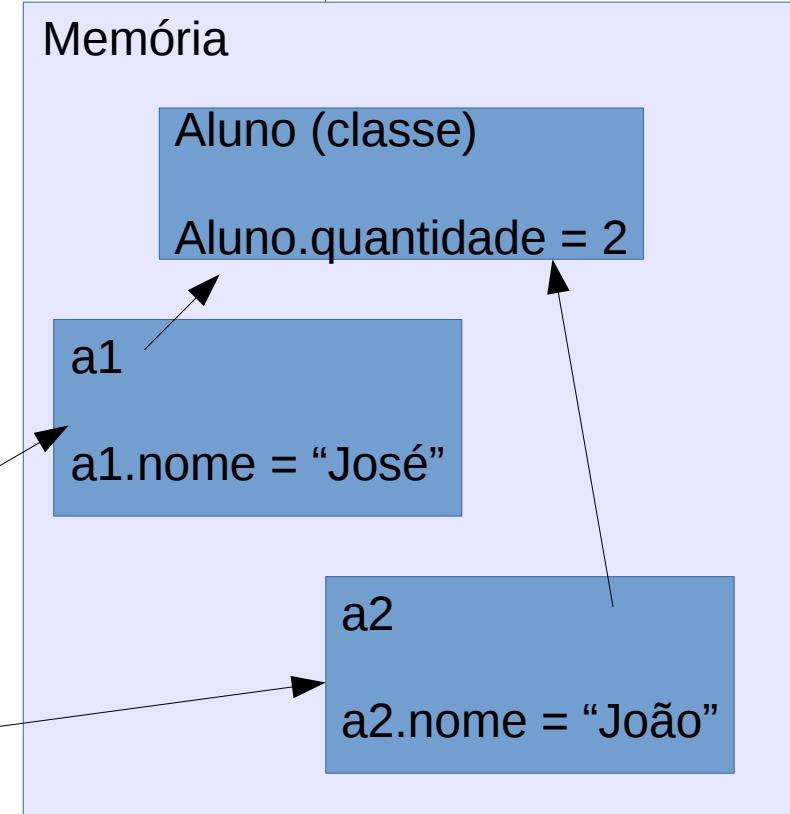
    public void setNome(String nome) {
        this.nome = nome;
    }

    public static int getQuantidade() {
        return quantidade;
    }

    public static void main(String[] args) {
        Aluno a1 = new Aluno("José");
        Aluno a2 = new Aluno("João");
        Aluno a3 = new Aluno("Caio");
        Aluno aN;

        System.out.println("Quantidade: " + Aluno.getQuantidade());
        for(int i=0; i<10; i++){
            aN = new Aluno("Gasparzinho");
        }
        System.out.println("Quantidade: " + Aluno.getQuantidade());
    }
}

```





Membros estáticos

- Java permite a criação de membros de classe que são comuns efetivamente a todos os objetos da classe.
- Só existe uma cópia de um atributo **estático** que pode ser usada por todos objetos da classe (variável global)

Cuidado com o uso de membros estáticos.

Variáveis globais continuam desencorajadas.





Quando usar membros estáticos?

- Um bom uso dos membros estáticos é a declaração de constantes.
- Ao criar a classe Calculadora, definiu-se as operações como constantes estáticas, isso permite que todo o sistema utilize essas constantes.

```
public class Calculadora {  
    public static final char OPERADOR_SOMA = '+';  
    public static final char OPERADOR_SUBTRACAO = '-';  
    public static final char OPERADOR_MULTIPLICACAO = '*';  
    public static final char OPERADOR_DIVISAO = '/';  
  
    private int memory;  
  
    public int calcular(int operando, char operador){  
        switch (operador){  
            case OPERADOR_SOMA:  
                memory = soma(operando);  
                break;  
            case OPERADOR_SUBTRACAO:  
                memory = subtrai(operando);  
                break;  
            case OPERADOR_MULTIPLICACAO:  
                memory = multiplica(operando);  
                break;  
            case OPERADOR_DIVISAO:  
                memory = divisao(operando);  
        }  
        return memory;  
    } ... }
```



() - constantes

Volte aos
membros
estáticos!



Constantes são declaradas com o operador **final** antes do tipo de dado. Isso indica que o valor daquela variável só é atribuído na declaração.

Também pode-se implementar métodos com o operador final. Isso define que o método não poderá ser rescrito. Assunto para um breve futuro.

```
public class Calculadora {  
    public static final char OPERADOR_SOMA = '+';  
    public static final char OPERADOR_SUBTRACAO = '-';  
    public static final char OPERADOR_MULTIPLICACAO = '*';  
    public static final char OPERADOR_DIVISAO = '/';
```



Constantes são variáveis que não variam!



Quando usar membros estáticos?

- Métodos acessados sem a instância do objeto.

```
public static Aluno lerAluno(){  
    Aluno estudante;  
    String nome;  
    int prontuario;  
    double n1, n2, n3, n4;  
    nome = JOptionPane.showInputDialog("Nome do aluno: ");  
    prontuario = Integer.parseInt(JOptionPane.showInputDialog("Prontuário: "));  
    n1 = Double.parseDouble(JOptionPane.showInputDialog("Prova 1:"));  
    n2 = Double.parseDouble(JOptionPane.showInputDialog("Prova 2:"));  
    n3 = Double.parseDouble(JOptionPane.showInputDialog("Exercícios:"));  
    n4 = Double.parseDouble(JOptionPane.showInputDialog("Projeto:"));  
    estudante = new Aluno(nome, prontuario, n1, n2, n3, n4);  
    return estudante;  
}
```

Quais métodos estáticos foram invocados e implementados nesse método?





Material adicional



- **Leitura Obrigatória**

- Winder, R.; Roberts, G. Desenvolvendo Software em Java. 3 ed. Rio de Janeiro: LTC, 2009.
 - Capítulo 6 completo.

- **Videoaula**

- https://www.youtube.com/watch?v=Dz_w4YpFL80
 - <https://www.youtube.com/watch?v=UMkftZ2hKxQ>
 - <https://www.youtube.com/watch?v=-dQ0yDTHcS0>



Array



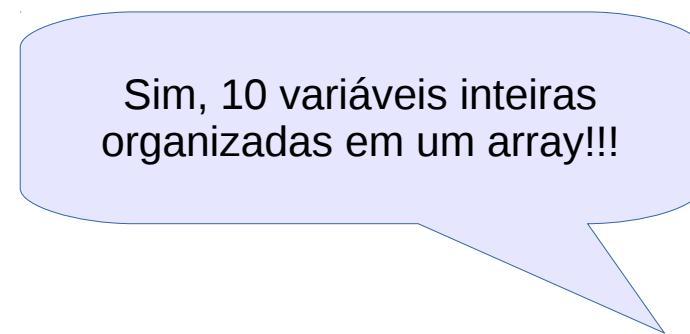
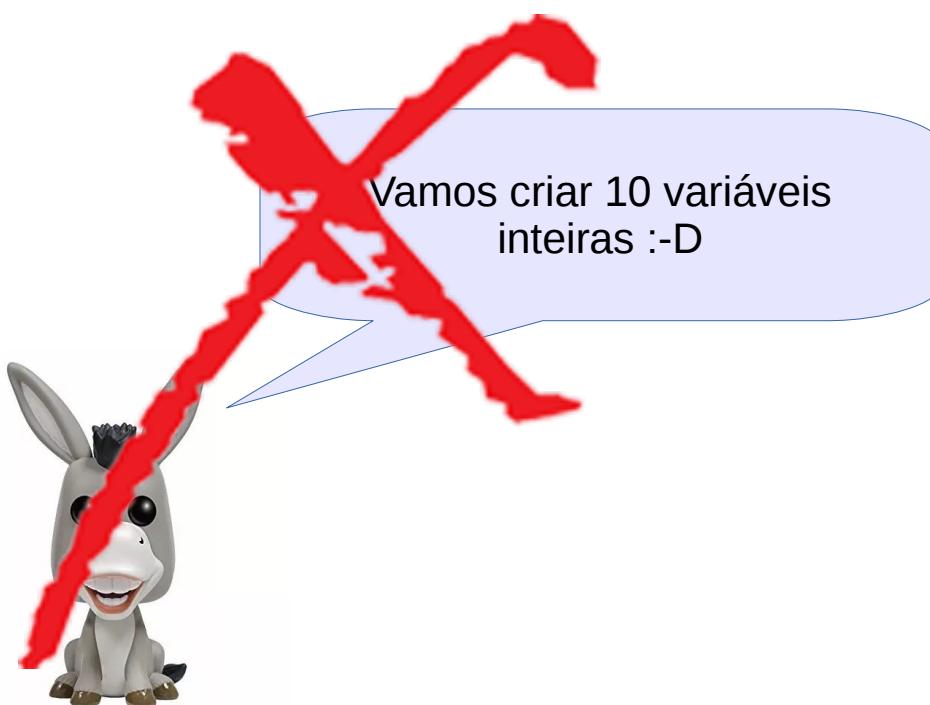
Arrays

- Um **array** é um conjunto de variáveis do mesmo tipo, referenciado por um nome comum e acessados por meio de sua posição (índice) .



Exemplo

- Implementar um programa que leia 10 idades e após a leitura informe, para cada idade lida, se a mesma está acima, abaixo ou na média.



```
package view;
import java.util.Scanner;

public class Pessoa {
    public static final int MAXIMO = 10;
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        int idades[] = new int[Pessoa.MAXIMO];
        int i;
        double media;

        System.out.println("Informe 10 idades: ");
        media = 0;
        for(i=0; i<Pessoa.MAXIMO; i++){
            idades[i] = input.nextInt();
            media += idades[i];
        }
        media /= Pessoa.MAXIMO;
        for(i=0; i<Pessoa.MAXIMO; i++){
            if(idades[i] > media)
                System.out.println("Idade " + idades[i] + " está acima da média");
            else
                if (idades[i] < media)
                    System.out.println("Idade " + idades[i] + " está abaixo da média");
                else
                    System.out.println("Idade " + idades[i] + " está exatamente na média");
        }
    }
}
```

Para declararmos um array definimos o tipo de dado armazenado e a quantidade de posições.

Podemos ter array de qualquer tipo de dado, inclusive arrays de objetos.

Para termos acesso a cada uma das variáveis declaradas dentro do array utilizamos o índice deste array.

Orientado a Objetos, programa não está.





```
package model;

public class Pessoa {
    private int idade;

    public Pessoa(int idade) {
        setIdade(idade);
    }

    public int getIdade() {
        return idade;
    }

    public void setIdade(int idade) {
        this.idade = idade >= 0 ? idade : 0 ;
    }
}
```

```

public class Main {
    public static final int MAXIMO = 10;

    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        Pessoa pessoas[] = new Pessoa[Main.MAXIMO];
        Pessoa p;
        int i;
        double media=0;

        System.out.println("Informe 10 idades: ");
        for(i=0; i<MAXIMO; i++){
            //p = new Pessoa(input.nextInt());
            //pessoas[i] = p;
            pessoas[i] = new Pessoa(input.nextInt());

            media += pessoas[i].getIdade();
        }
        media /= MAXIMO;
        for(i=0; i < MAXIMO; i++){
            if(pessoas[i].getIdade() > media)
                System.out.println("Idade " + pessoas[i].getIdade() + " está acima da
                                  média");
            else
                if (pessoas[i].getIdade() < media)
                    System.out.println("Idade " + pessoas[i].getIdade() + " está abaixo da
                                      média");
                else
                    System.out.println("Idade " + pessoas[i].getIdade() + " está
                                      exatamente na média");
        }
    }
}

```

Como pretendo armazenar objetos define-se a classe como tipo de dados do array.

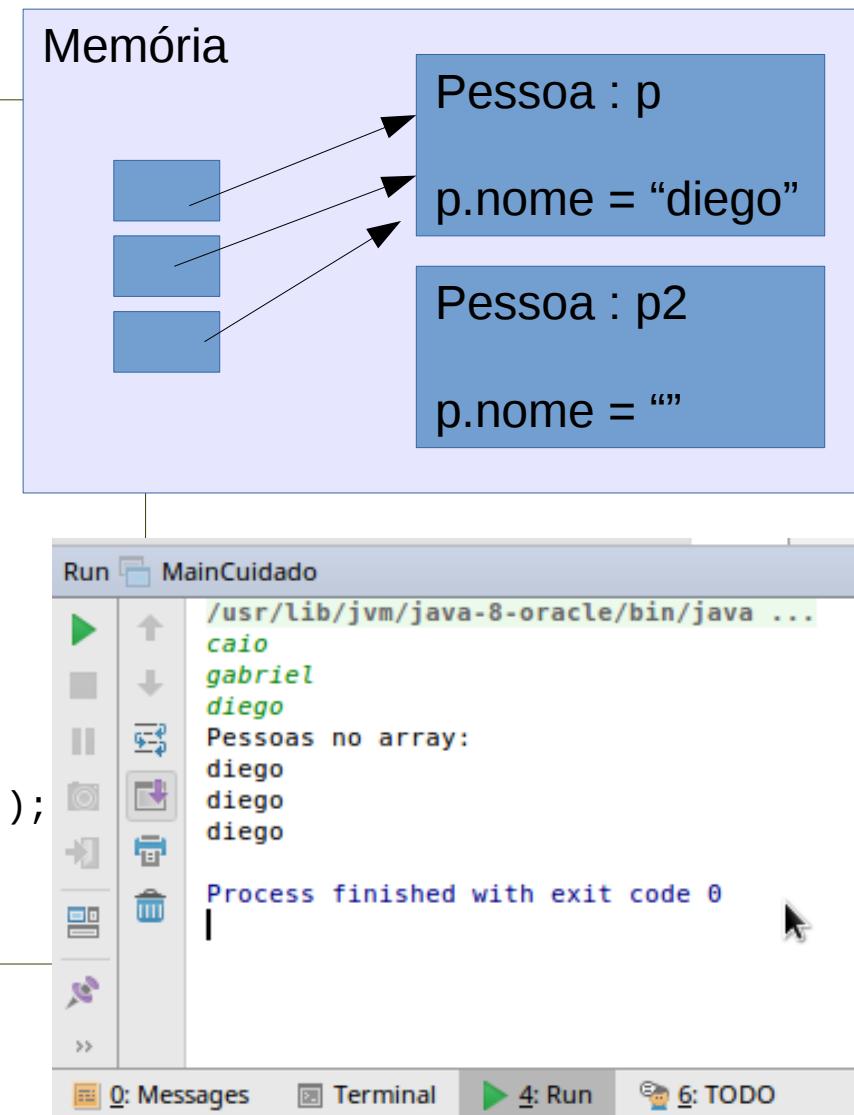
Diferente de um tipo primitivo, é necessário que um novo objeto seja instanciado para a posição do array. Pode-se instanciar um objeto e atribuí-lo ao array ou mesmo instanciar o objeto dentro do array.

Como cada posição do array possui um objeto, é possível chamar os métodos do objeto que está no array.



Atenção

```
public class MainCuidado {  
    public static final int MAXIMO = 10;  
  
    public static void main(String[] args) {  
        Scanner input = new Scanner(System.in);  
        Pessoa pessoas[] = new Pessoa[MAXIMO];  
        Pessoa p, p2;  
        int i;  
        p = new Pessoa();  
        p2 = new Pessoa();  
        for(i=0; i<MAXIMO; i++){  
            p.setNome(input.nextLine());  
            pessoas[i] = p;  
        }  
        System.out.println("Pessoas no array:");  
        for(i=0;i<MAXIMO;i++){  
            System.out.println(pessoas[i].getNome());  
        }  
    }  
}
```





Exemplo 10

- Pode-se utilizar array dentro de uma classe.
- Analisemos um exemplo onde uma loja que possui até 4 vendedores quer saber qual o vendedor que teve maior valor em vendas acumuladas no período de um ano.



```
public class Vendedor {  
    public static final int MESES = 12;  
    private String nome;  
    private double vendasMes[];  
  
    public Vendedor(String nome) {  
        this.nome = nome;  
        vendasMes = new double[MESES];  
    }  
  
    public void setVendasMes(double valor, int mes) {  
        vendasMes[mes] = valor;  
    }  
  
    public double getVendasAno(){  
        double soma=0;  
        for(int i=0; i<vendasMes.length; i++){  
            soma += vendasMes[i];  
        }  
        return soma;  
    }  
  
    public double getVendasPeriodo(int mesInicio, int mesFinal){  
        double soma=0;  
        for(; mesInicio <= mesFinal; mesInicio++){  
            soma += vendasMes[mesInicio];  
        }  
        return soma;  
    }  
    ...  
}
```

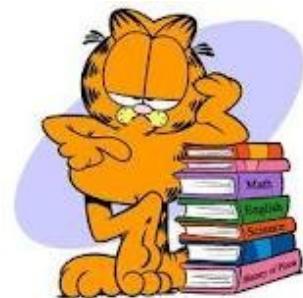
```
public class Loja {  
    private final int TOTAL_VENDEDORES = 4;  
    private String razaoSocial;  
    private Vendedor[] vendedores;  
    private int vendedoresCadastrados;  
  
    public Loja(String razaoSocial) {  
        this.razaoSocial = razaoSocial;  
        vendedores = new Vendedor[TOTAL_VENDEDORES];  
        vendedoresCadastrados = 0;  
    }  
  
    public boolean insereVendedor(Vendedor vendedor){  
        boolean deuCerto = false;  
        if(vendedoresCadastrados < TOTAL_VENDEDORES){  
            vendedores[vendedoresCadastrados] = vendedor;  
            vendedoresCadastrados++;  
            deuCerto = true;  
        }  
        return deuCerto;  
    }  
  
    public Vendedor getMelhorVendedor(){  
        Vendedor melhor=null;  
  
        if(vendedoresCadastrados > 0){  
            melhor = vendedores[0];  
            for(int i=1; i<vendedoresCadastrados; i++){  
                if(vendedores[i].getVendasAno() > melhor.getVendasAno()){  
                    melhor = vendedores[i];  
                }  
            }  
        }  
        return melhor;  
    }  
}
```



```
public class Main {  
  
    public static Vendedor lerVendedor(){  
        String nome;  
        double valor;  
        Vendedor v;  
  
        nome = JOptionPane.showInputDialog("Nome do vendedor: ");  
        v = new Vendedor(nome);  
  
        for(int mes=0; mes<12; mes++){  
            valor = Double.parseDouble(JOptionPane.showInputDialog("Vendas do mês " +  
                (mes+1)));  
            v.setVendasMes(valor, mes);  
        }  
        return v;  
    }  
  
    public static void main(String[] args) {  
  
        Loja loja = new Loja("Loja do Python");  
        loja.insereVendedor(lerVendedor());  
        loja.insereVendedor(lerVendedor());  
        loja.insereVendedor(lerVendedor());  
        loja.insereVendedor(lerVendedor());  
        JOptionPane.showMessageDialog(null, "Melhor vendedor: " +  
            loja.getMelhorVendedor().getNome());  
    }  
}
```



Material Adicional



- Leitura Obrigatória
 - Winder, R.; Roberts, G. Desenvolvendo Software em Java. 3 ed. Rio de Janeiro: LTC, 2009.
 - Capítulos 4.1 e 4.2
- Videoaulas
 - <https://www.youtube.com/watch?v=Kc2RXH7dW-M>
 - <https://www.youtube.com/watch?v=HxRb5KLofcl>
 - <https://www.youtube.com/watch?v=UN6ZJBSTR0M>
 - <https://www.youtube.com/watch?v=Es42H9OrDyl>



Trabalhando



- **Exercícios de Fixação**
 - 6 e 7
- **Exercícios Avaliativos**
 - 4
- **Leitura Obrigatória para próxima aula**
 - Winder, R.; Roberts, G. Desenvolvendo Software em Java. 3 ed. Rio de Janeiro: LTC, 2009.
 - Capítulo 7.3
 - Deitel, P.; Deitel, H. Java: Como programar. 10 ed. São Paulo: Pearson, 2017.
 - Capítulo 9



Herança



Herança

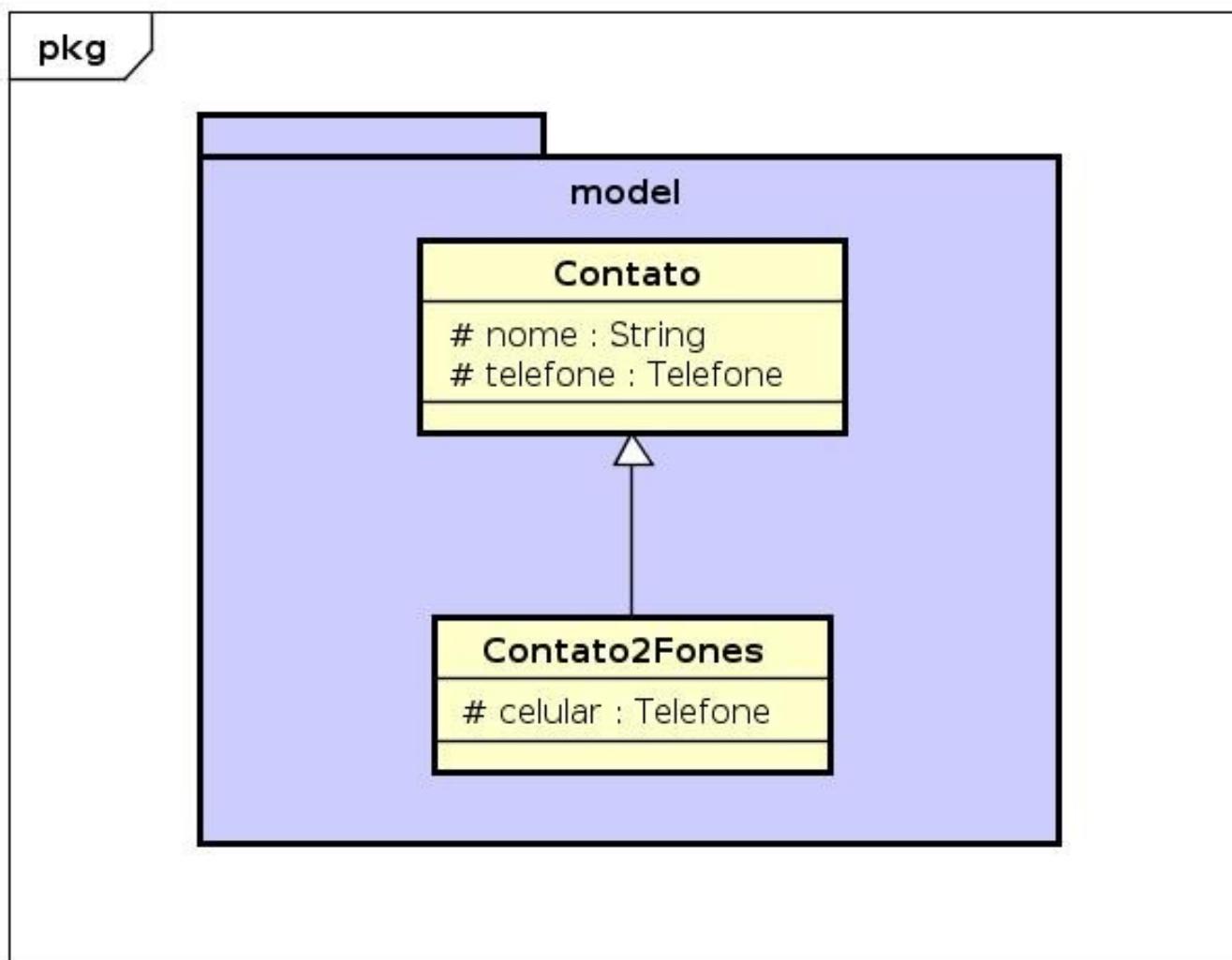
- Herança é um dos três princípios básicos da programação orientada a objetos, porque permite a criação e classificação hierárquica.
 - Com seu uso, pode-se criar uma classe geral que define características comuns a um conjunto de itens relacionados.
- A herança torna possível a reutilização de funcionalidades previamente definidas em uma classe.
- A finalidade é que a **subclasse** inclua o comportamento da **superclasse** e adicione mais funcionalidades.



Como assim?

- Considerando a implementação de uma classe Contato:
 - Essa classe já está em produção, ou seja, clientes estão utilizando o sistema e objetos dessa classe são instanciados.
 - Outro requisito pode surgir no sistema, como por exemplo, a inclusão de um segundo número de telefone.
- Com a herança pode-se ampliar (estender) as características e funcionalidades de uma classe já existente.
 - Também é possível alterar o comportamento de alguns métodos implementados na classe original (superclasse / classe pai).

Contato e Contato2Fones





Contato.java

```
package model;

/**
 * Classe que representa um contato.
 *
 * @author ednilsonrossi
 */

public class Contato {

    protected String nome;
    protected Telefone telefone;

    public Contato(String nome, int ddd, int prefixo, int numero) {
        this.nome = nome.toUpperCase();
        telefone = new Telefone(ddd, prefixo, numero);
    }

    @Override
    public String toString() {
        return "Contato: " + nome + " \t Telefone: " + telefone.toString();
    }
    //Gets e Sets
}
```

A classe Contato define dois atributos, o nome do contato e um telefone.

Toda classe criada já é uma herança da classe Object, por isso existem alguns comportamentos comuns a todos objetos. Assim, reescrevemos o método `toString()` que foi definido na superclasse, ou seja, mudamos o comportamento do método.



Contato2Fones.java



```
public class Contato2Fones extends Contato {  
    public static final int TIPO_FIXO = 1;  
    public static final int TIPO_CELULAR = 2;  
  
    protected TelefoneCelular celular;  
  
    public Contato2Fones(String nome, int ddd, int prefixo, int numero,  
        int dddCelular, int prefixoCelular, int numeroCelular) {  
        super(nome, ddd, prefixo, numero);  
        this.celular = new TelefoneCelular(dddCelular, prefixoCelular, numeroCelular);  
    }  
  
    public Contato2Fones(String nome, int ddd, int prefixo, int numero) {  
        super(nome, ddd, prefixo, numero);  
    }  
  
    public Contato2Fones(String nome, int ddd, int prefixo, int numero, int tipo) {  
        super(nome, ddd, prefixo, numero);  
        switch (tipo){  
            case TIPO_FIXO:  
                celular = (TelefoneCelular) doInvado();  
                break;  
            case TIPO_CELULAR:  
                this.telefone = doInvado();  
                this.celular = new TelefoneCelular(ddd, prefixo, numero);  
                break;  
            default:  
                this.celular = (TelefoneCelular) doInvado();  
                this.telefone = doInvado();  
        }  
    }  
  
    @Override  
    public String toString() {  
        String pai = super.toString();  
        pai = pai + "\t Celular: " + celular.toString();  
        return pai;  
    }  
}
```

Contato2Fones amplia a classe Contato, para formalizarmos essa ampliação dizemos que a Contato2Fones **estende (extends)** a classe Contato. Aqui temos que Contato é superclasse (pai) de Contato2Fones.
Além disso, Contato2Fones é um Contato.

O **super** é a indicação da superclasse. No caso, super() é a chamada do método construtor da superclasse. Sempre devemos construir a superclasse na subclasses.

Novamente reescreveu-se o método `toString()`. Agora o comportamento foi alterado para o cumprir um novo papel.

Importante observar que, se o método não for reescrito o método da classe pai será invocado.



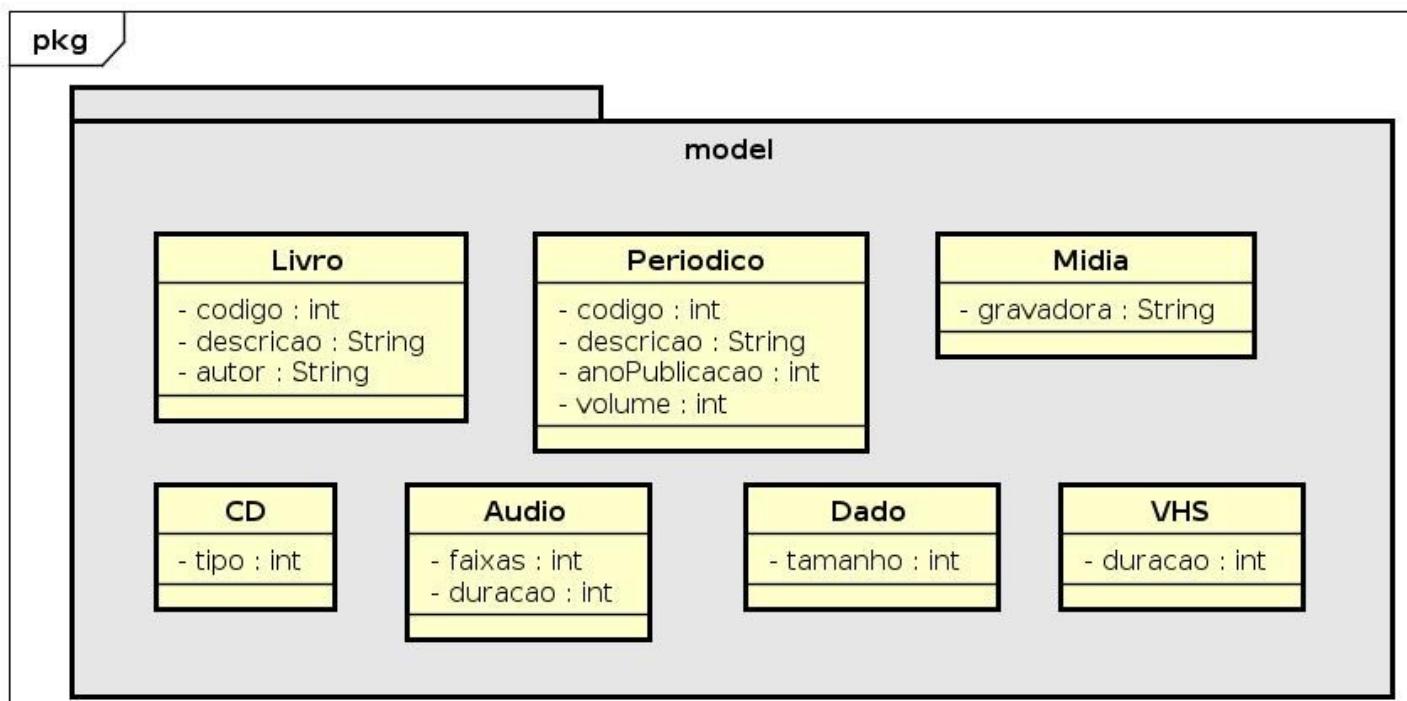
Problema

- Uma biblioteca possui vários itens em seu acervo, são eles:
 - Livro;
 - Periódico;
 - Mídia:
 - CD:
 - Áudio
 - Dados
 - VHS



Nossas classes

Modelo adequado, não está!

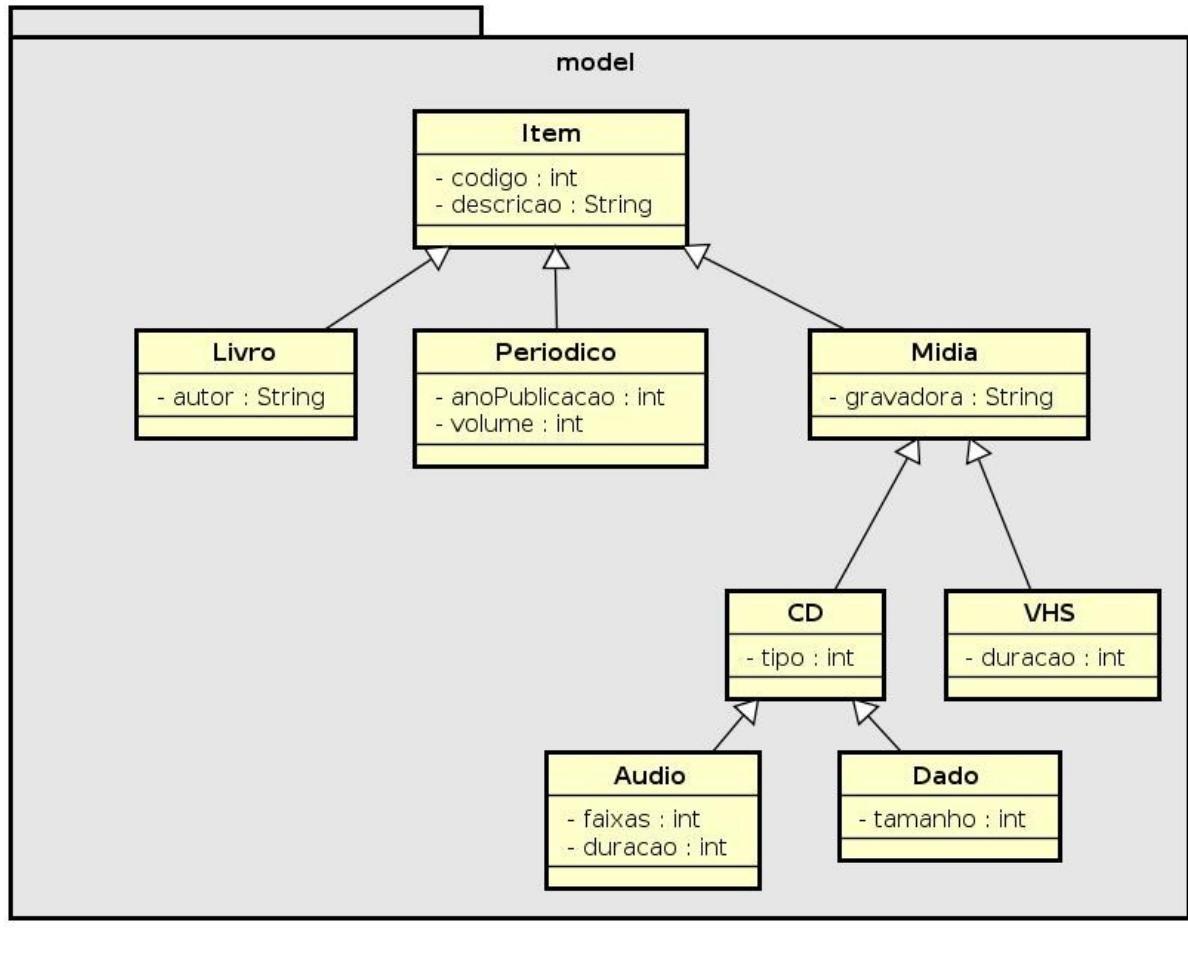


É possível observar a repetição de código, atributos, entre classes. Isso não é adequado. Para resolvemos isso vamos criar uma classe genérica e usar herança.





pkg



powered by Astah

Organizou-se o modelo de forma que a classe genérica Item represente todos os itens da biblioteca.

Perceba que não existe mais a repetição de atributos.

Além disso existem outras vantagens que serão vistas daqui a pouco.

Agora estamos prontos para dominar o código!



```
package model;

public class Item {

    private int id;
    private String descricao;

    public Item(int id, String descricao) {
        setId(id);
        setDescricao(descricao);
    }

    @Override
    public String toString(){
        StringBuffer sb = new StringBuffer();
        sb.append("ID: ");
        sb.append(id);
        sb.append("\t");
        sb.append("Descrição: ");
        sb.append(descricao);
        return sb.toString();
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id >= 0? id : 0;
    }

    public String getDescricao() {
        return descricao;
    }

    public void setDescricao(String descricao) {
        this.descricao = descricao.toUpperCase();
    }
}
```

Exemplo11



```
package model;

public class Livro extends Item {

    private String autor;

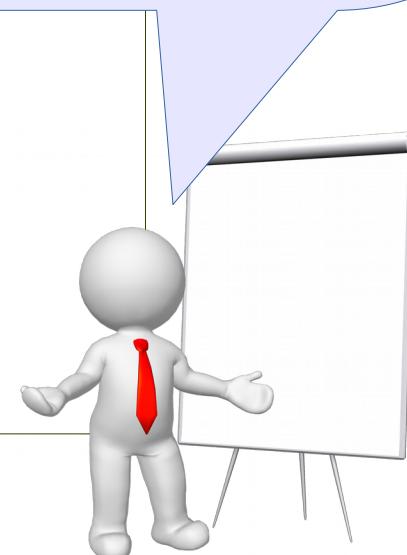
    public Livro(int id, String descricao, String autor) {
        super(id, descricao);
        this.autor = autor;
    }

    public String getAutor() {
        return autor;
    }

    public void setAutor(String autor) {
        this.autor = autor.toUpperCase();
    }

    @Override
    public String toString() {
        StringBuffer sb = new StringBuffer();
        sb.append(super.toString());
        sb.append("\n");
        sb.append("Autor do livro: ");
        sb.append(autor);
        return sb.toString();
    }
}
```

Observe que Livro é um Item e por isso possui todas as características de Item e as de Livro.



```
package model;

public class Periodico extends Item {

    private int anoPublicacao;
    private int volume;

    public Periodico(int id, String descricao, int anoPublicacao, int volume) {
        super(id, descricao);
        setAnoPublicacao(anoPublicacao);
        setVolume(volume);
    }

    public int getAnoPublicacao() {
        return anoPublicacao;
    }

    public void setAnoPublicacao(int anoPublicacao) {
        this.anoPublicacao = anoPublicacao > 1900 ? anoPublicacao:1900;
    }

    public int getVolume() {
        return volume;
    }

    public void setVolume(int volume) {
        this.volume = volume>0?volume:1;
    }

    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();
        sb.append(super.toString());
        sb.append("\nAno de publicação: ");
        sb.append(anoPublicacao);
        sb.append("\tVolume: ");
        sb.append(volume);
        return sb.toString();
    }
}
```



```
package model;

/**
 * Classe que representa uma Mídia
 *
 * @see model.Item
 * @author ednilsonrossi
 */
public class Midia extends Item {
    private String gravadora;

    public Midia(int id, String descricao, String gravadora) {
        super(id, descricao);
        this.gravadora = gravadora.toUpperCase();
    }

    @Override
    public String toString(){
        StringBuilder sb = new StringBuilder();
        sb.append(super.toString());
        sb.append("\nGravadora: ");
        sb.append(gravadora);
        return sb.toString();
    }

    public String getGravadora() {
        return gravadora;
    }

    /**
     * Configura a gravadora. Armazenado como maiúsculo
     * @param gravadora
     */
    public void setGravadora(String gravadora) {
        this.gravadora = gravadora.toUpperCase();
    }
}
```



CD é uma Midia que por sua vez é um Item, assim, CD possui todas as características de Item e Midia.

Isso é Hierarquia de classes.



```
package model;

public class CD extends Midia{
    public static final int TIPO_CDROM = 1;
    public static final int TIPO_DVD = 2;

    private int tipo;

    public CD(int id, String descricao, String gravadora, int tipo) {
        super(id, descricao, gravadora);
        this.tipo = tipo;
    }

    public int getTipo() {
        return tipo;
    }

    public void setTipo(int tipo) {
        if(tipo == TIPO_CDROM || tipo == TIPO_DVD)
            this.tipo = tipo;
        else
            this.tipo = TIPO_CDROM;
    }

    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();
        sb.append(super.toString());
        if(tipo == TIPO_CDROM)
            sb.append("\nTipo: CDROM");
        else
            sb.append("\nTipo: DVD");
        return sb.toString();
    }
}
```

```
package model;

public class Audio extends CD {

    private int faixas;
    private int duracao;

    public Audio(int id, String descricao, String gravadora, int tipo, int faixas, int duracao) {
        super(id, descricao, gravadora, tipo);
        setFaixas(faixas);
        setDuracao(duracao);
    }

    public int getFaixas() {
        return faixas;
    }

    public void setFaixas(int faixas) {
        this.faixas = faixas>=1?faixas:1;
    }

    public int getDuracao() {
        return duracao;
    }

    public void setDuracao(int duracao) {
        this.duracao = duracao>=1?duracao:1;
    }

    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();
        sb.append(super.toString());
        sb.append("\n Faixas: ");
        sb.append(faixas);
        sb.append("\tDuração: ");
        sb.append(duracao);
        sb.append(" minutos");
        return sb.toString();
    }
}
```



```
package model;

public class Dado extends CD {
    private int tamanho;

    public Dado(int id, String descricao, String gravadora, int tipo, int tamanho) {
        super(id, descricao, gravadora, tipo);
        setTamanho(tamanho);
    }

    public int getTamanho() {
        return tamanho;
    }

    public void setTamanho(int tamanho) {
        this.tamanho = tamanho>=1?tamanho:1;
    }

    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();
        sb.append(super.toString());
        sb.append("\nTamanho: ");
        sb.append(tamanho);
        sb.append(" MB");
        return sb.toString();
    }
}
```



```
package model;

/**
 * Classe que representa um VHS
 */
public class VHS extends Midia{
    private int duracao;

    public VHS(int id, String descricao, String gravadora, int duracao) {
        super(id, descricao, gravadora);
        setDuracao(duracao);
    }

    /**
     * Informa a duração em minutos.
     * @return
     */
    public int getDuracao() {
        return duracao;
    }

    /**
     * Configura a duração em minutos. Mínimo 1 minuto.
     * @param duracao
     */
    public void setDuracao(int duracao) {
        this.duracao = duracao>=1?duracao:1;
    }

    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();
        sb.append(super.toString());
        sb.append("\nDuração: ");
        sb.append(duracao);
        sb.append(" minutos.");
        return sb.toString();
    }
}
```

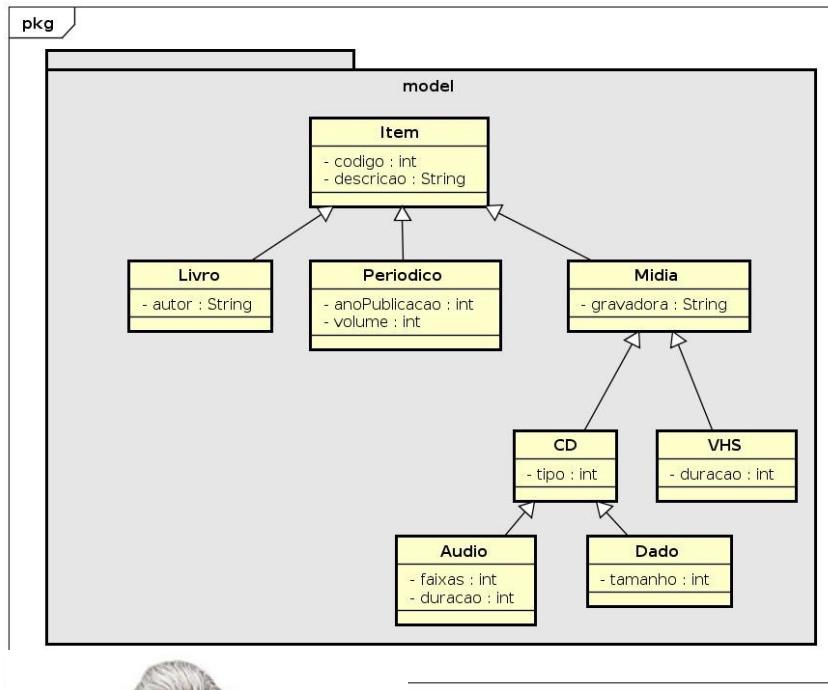
Legal esse negócio de herança.

Ficou alguma dúvida?





Problema - Biblioteca



Para resolver o problema dessa natureza, faz-se necessário, por assim dizer, um estudo mais aprofundado de outra importante característica da Orientação a Objetos, o Polimorfismo

Como esse negócio de herança pode me ajudar a resolver o problema de cadastrar todos esses itens?

Terei que instanciar os objetos da sua respectiva classe, acho que só gerou mais trabalho e o professor me enrolou!





Polimorfismo

- Existem duas formas de polimorfismo:
 - De métodos
 - Falaremos mais adiante
 - De compatibilidade de tipos
 - Aplicaremos no problema da biblioteca



Aplicação de polimorfismo no problema da biblioteca

- Com o polimorfismo é possível armazenar um objeto em outro.
 - O polimorfismo de objetos permite que um objeto seja tratado como se fosse outro.
- No exemplo da Biblioteca será criado um **array** de Itens (classe mais genérica) para armazenar todos os objetos que forem filhos de Item.
- Nesse array poderemos armazenar Livros, Periódicos, CD de Audio, CD de Dados, VHS.

```

public class Biblioteca {
    private static final int MAXIMO = 100;
    private Item[] itens;
    private int maximoItens;
    private int itensCadastrados;

    public Biblioteca(int maximoItens) {
        setMaximoItens(maximoItens);
        this.itensCadastrados = 0;
        itens = new Item[maximoItens];
    }

    public int getMaximoItens() { return maximoItens; }
    private void setMaximoItens(int maximoItens) { this.maximoItens = maximoItens>=1?maximoItens:1; }
    public int getItensCadastrados() { return itensCadastrados; }

    private boolean novoItem(Item item){
        boolean deuCerto = false;
        if(!isFull() && item != null){
            itens[itensCadastrados] = item;
            itensCadastrados += 1;
            deuCerto = true;
        }
        return deuCerto;
    }

    public boolean isFull(){ return itensCadastrados == maximoItens; }
    public boolean cadastralivro(Livro livro){ return novoItem(livro); }
    public boolean cadastraperiodico(Periodico periodico){ return novoItem(periodico); }
    public boolean cadastrauldoo(Audio audio){ return novoItem(audio); }
    public boolean cadastradado(Dado dado){ return novoItem(dado); }
    public boolean cadastravhs(VHS vhs){ return novoItem(vhs); }

    public Item getItemAt(int posicao){
        Item item = null;
        if(posicao >= 0 && posicao < itensCadastrados){
            item = itens[posicao];
        }
        return item;
    }
}

```

O método que faz o cadastro recebe qualquer subclasse de Item e insere no **array** de Itens.

Os métodos de cadastro apenas definem a “interface” para que o usuário insira objetos na biblioteca.

Cada objeto assume a “forma” de um Item.



O MAIN

```
package view;  
  
import model.*;  
  
public class Main {  
  
    public static void main(String[] args) {  
  
        Biblioteca library = new Biblioteca(5);  
  
        library.cadastraLivro(new Livro(5, "Java como programar", "Deitel & Deitel"));  
        library.cadastraVHS(new VHS(10, "De volta a lagoa azul.", "globoplay", 120));  
        library.cadastraDado(new Dado(15, "Dados do último SISU", "INEP", Dado.TIPO_CDROM, 720));  
        library.cadastraLivro(new Livro(20, "Desenvolvendo SW em Java", "Russel"));  
        library.cadastraAudio(new Audio(25, "Batidão", "Som Livre", Audio.TIPO_DVD, 10, 185));  
  
        int i=0;  
        Item objeto;  
        do{  
            objeto = library.getItemAt(i);  
            if(objeto != null){  
                System.out.println(objeto.toString());  
                System.out.println("-----");  
            }  
            i++;  
        }while (objeto != null);  
    }  
}
```

Observe que na biblioteca foram inseridos os mais variados objetos. Todos subclasses de Item.



Executado ...

```
ID: 5      Descrição: JAVA COMO PROGRAMAR
Autor do livro: Deitel & Deitel
-----
ID: 10     Descrição: DE VOLTA A LAGOA AZUL.
Gravadora: GLOBO
Duração: 120 minutos.
-----
ID: 15     Descrição: DADOS DO ÚLTIMO SISU
Gravadora: INEP
Tipo: CDROM
Tamanho: 720 MB
-----
ID: 20     Descrição: DESENVOLVENDO SW EM JAVA
Autor do livro: Russel
-----
ID: 25     Descrição: BATIDÃO
Gravadora: SOM LIVRE
Tipo: DVD
Faixas: 10    Duração: 185 minutos
-----
```



O Cliente ...

- Cliente quer saber quantos livros estão cadastrados na biblioteca.
 - Nem todos os itens do array são da classe Livro.
 - O que fazer?
 - Contar!!!



O operado **instanceof** compara se um objeto é uma instância daquela classe ou não.

ID: 5 Descrição: JAVA COMO PROGRAMAR
Autor do livro: Deitel & Deitel

ID: 10 Descrição: DE VOLTA A LAGOA AZUL.
Gravadora: GLOBO
Duração: 120 minutos.

ID: 15 Descrição: DADOS DO ÚLTIMO SISU
Gravadora: INEP
Tipo: CDROM
Tamanho: 720 MB

ID: 20 Descrição: DESENVOLVENDO SW EM JAVA
Autor do livro: Russel

ID: 25 Descrição: BATIDÃO
Gravadora: SOM LIVRE
Tipo: DVD
Faixas: 10 Duração: 185 minutos

Total de livros: 2

Classe Biblioteca

```
public int quantidadeLivros(){  
    int i, conta;  
    for(i=0, conta=0; i<itensCadastrados; i++){  
        if(itens[i] instanceof Livro){  
            conta++;  
        }  
    }  
    return conta;  
}
```

Método main()

```
System.out.println("Total de livros: " + library.quantidadeLivros());
```



Polimorfismo

- Também é a troca de mensagens entre métodos de uma subclasse com sua superclasse e assim sucessivamente.



Qual método `toString()` é chamado?

Lembrando que estamos
implementando a classe `Item` que
tem o método `toString()`.

Na classe Item

```
public String getDescritivo(){  
    StringBuilder sb = new StringBuilder();  
    sb.append("Extrato do Objeto\n");  
    sb.append("-----\n");  
    sb.append("Objeto da classe: ");  
    sb.append(getClass().getName());  
    sb.append("\n\t");  
  
    sb.append(toString());  
  
    sb.append("\n-----\n");  
    return sb.toString();  
}
```

Método main()

```
int i=0;  
Item objeto;  
do{  
    objeto = library.getItemAt(i);  
    if(objeto != null){  
        System.out.println(objeto.getDescritivo());  
    }  
    i++;  
}while (objeto != null);
```

Extrato do Objeto

Objeto da classe: model.Livro

ID: 5 Descrição: JAVA COMO PROGRAMAR
Autor do livro: Deitel & Deitel

Extrato do Objeto

Objeto da classe: model.VHS

ID: 10 Descrição: DE VOLTA A LAGOA AZUL.
Gravadora: GLOBO
Duração: 120 minutos.

Extrato do Objeto

Objeto da classe: model.Dado

ID: 15 Descrição: DADOS DO ÚLTIMO SISU
Gravadora: INEP
Tipo: CDROM
Tamanho: 720 MB

Extrato do Objeto

Objeto da classe: model.Livro

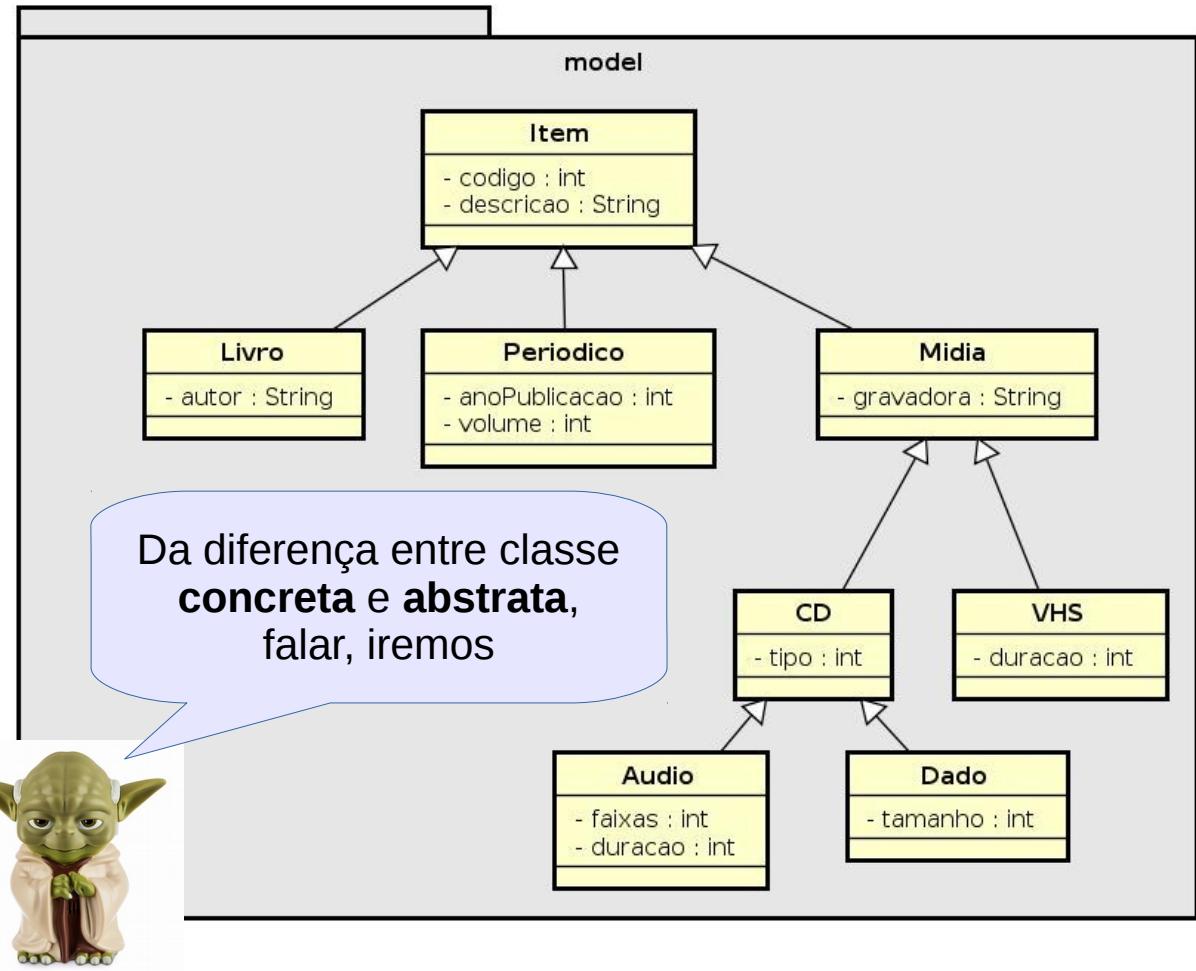
ID: 20 Descrição: DESENVOLVENDO SW EM JAVA
Autor do livro: Russel

Extrato do Objeto

Objeto da classe: model.Audio

ID: 25 Descrição: BATIDÃO
Gravadora: SOM LIVRE
Tipo: DVD
Faixas: 10 Duração: 185 minutos

pkg



Legal esse negócio de **herança** e **polimorfismo!**

Mas por que, na Biblioteca, só existem métodos para cadastrar Livro, Periódico, Áudio, Dado e VHS?

Por que não cadastraria Mídia, CD e ITEM?



```
public boolean cadastraLivro(Livro livro){      return novoItem(livro);      }
public boolean cadastraPeriodico(Periodico periodico){      return novoItem(periodico);      }
public boolean cadastraAudio(Audio audio){      return novoItem(audio);      }
public boolean cadastraDado(Dado dado){      return novoItem(dado);      }
public boolean cadastraVHS(VHS vhs){      return novoItem(vhs);      }
```



Classe Concreta x Classe Abstrata

- Uma classe concreta permite a instância de um objeto daquela classe.
 - Na biblioteca temos instância de algumas classes concretas:

```
library.cadastraLivro(new Livro(5, "Java como programar", "Deitel & Deitel"));  
library.cadastraVHS(new VHS(10, "De volta a lagoa azul.", "globo", 120));  
library.cadastraDado(new Dado(15, "Dados do último SISU", "INEP", Dado.TIPO_CDROM, 720));  
library.cadastraLivro(new Livro(20, "Desenvolvendo SW em Java", "Russel"));  
library.cadastraAudio(new Audio(25, "Batidão", "Som Livre", Audio.TIPO_DVD, 10, 185));
```

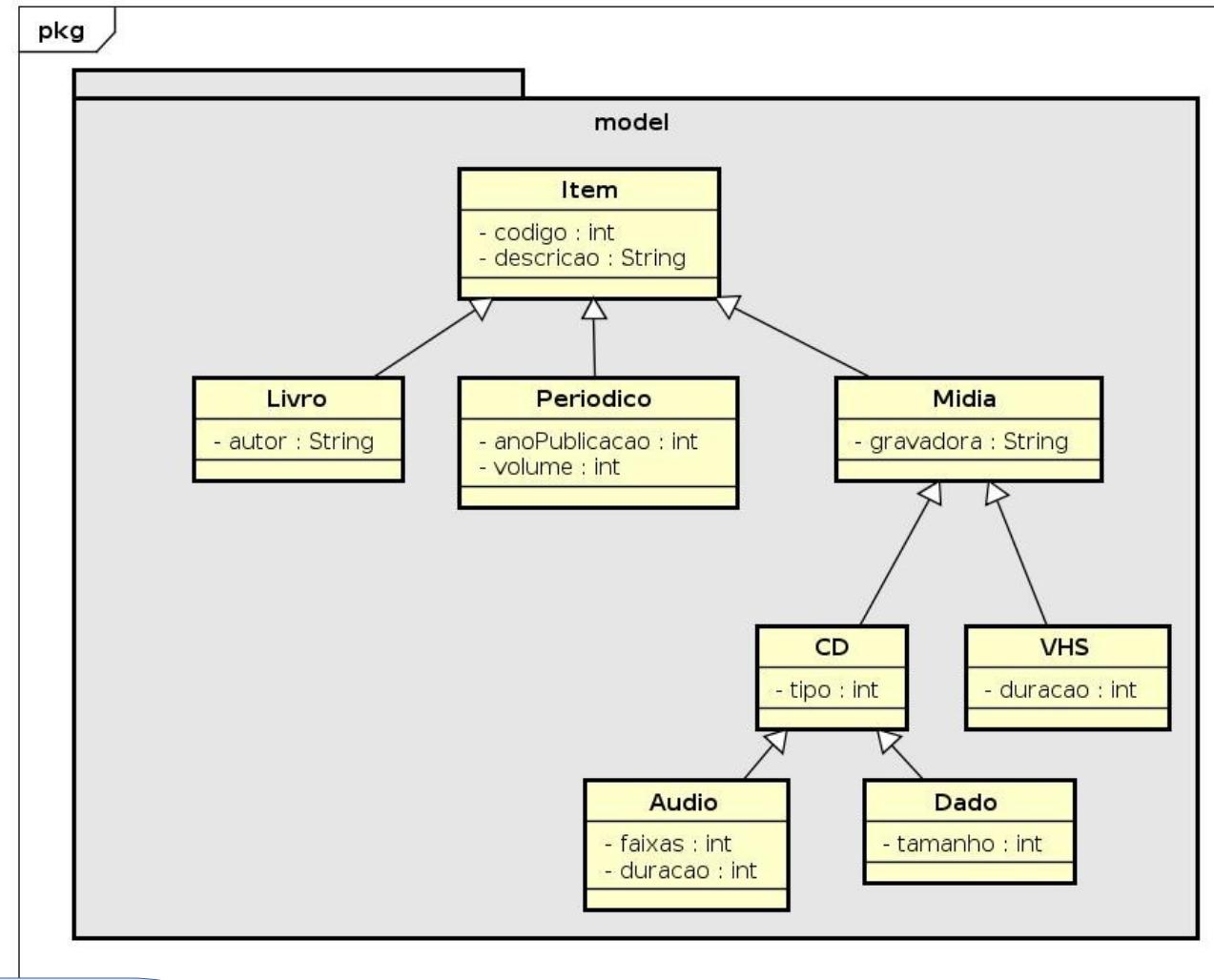
- Classe abstrata não permite que objetos sejam instanciados a partir dela, apenas de suas subclasses. Assim, uma Classe Abstrata é um modelo a ser seguido pelas subclasses.
 - Uma classe abstrata não precisa ser implementada totalmente, mas pode “forçar” que suas subclasses implementem comportamentos específicos.



Observe que nunca iremos instanciar um Item, mas os filhos de Item. Isso nos indica que Item pode ser uma **Classe Abstrata**, assim como Mídia e CD.

Não faz sentido para nossa Biblioteca termos um CD que não seja um CD de Audio ou um CD de Dados.

Como transformar nossas classes em classes abstratas?



powered by Astah





```
public abstract class Item { ... }  
  
public abstract class Midia extends Item { ... }  
  
public abstract class CD extends Midia{ ... }
```

Basta inserir a palavra “**abstract**” antes da classe para tornar nossa classe abstrata.



Legal professor, ficou
“bunitinho”.

Mas o que mudou?

Vá no método main() e instancie
um objeto da classe Item, usando o
construtor de Item.

Item obj = new Item(1, “Bla”);



É possível instanciar uma classe abstrata desde que ela seja implementada por você, em tempo de execução. Veremos isso no futuro.

Por hora, temos que classes abstratas são modelos para classes concretas. Mas não para por aqui. Temos que falar de métodos abstratos.

Uma classe abstrata pode possuir métodos (assinaturas) abstratos e fazer com que os filhos o implementem. Vejamos.



```
public abstract class CD extends Midia{
    public static final int TIPO_CDROM = 1;
    public static final int TIPO_DVD = 2;

    private int tipo;

    /**
     * Método abstrato que imprime as informações da capa do CD.
     *
     * @return String com Descrição e Gravadora.
     */
    public abstract String imprimeCapa();

    public CD(int id, String descricao, String gravadora, int tipo) {
        super(id, descricao, gravadora);
        this.tipo = tipo;
    } ... }

public class Dado extends CD {
    private int tamanho;

    public Dado(int id, String descricao, String gravadora) {
        super(id, descricao, gravadora, tipo);
        setTamanho(tamanho);
    }

    @Override
    public String imprimeCapa() {
        StringBuilder sb = new StringBuilder();
        sb.append("Título: ");
        sb.append(getDescricao());
        sb.append("\nGravadora: ");
        sb.append(getGravadora());
        sb.append("\nTamanho: ");
        sb.append(tamanho);
        sb.append(" MB");
        return sb.toString();
    }
}
```

O método abstrato `imprimeCapa()` não possui implementação em `CD`, contudo, as subclasses de `CD` são **obrigadas** a implementar esse método.

```
public class Audio extends CD {

    private int faixas;
    private int duracao;

    @Override
    public String imprimeCapa() {
        StringBuilder sb = new StringBuilder();
        sb.append("Título: ");
        sb.append(getDescricao());
        sb.append("\nGravadora: ");
        sb.append(getGravadora());
        sb.append("\nFaixas: ");
        sb.append(faixas);
        return sb.toString();
    }
}
```



Trabalhando

- **Exercício de Fixação**
 - 8 e 9
- **Exercícios Avaliativos**
 - *São considerados Atividade Prática Supervisionada
 - 5, 6, 7*, 8* e 9*
- **Próxima aula (06/04)**
 - Avaliação individual e sem consulta.

