




# Java WebDeveloper - Noite Seg, Qua, Sex - Aula 01 - Noite



## Tema da aula:

Orientação a Objetos, Programas para baixar, Projeto local, Importação de projeto, Atalhos, Modelagem UML, Classe, Java Beans, Objeto, MVC, Encapsulamento, Calculo media, Switch, Lançamento de erro, Lista, adicionar

—  Prof Edson Belém - [profedsonbelem@gmail.com](mailto:profedsonbelem@gmail.com) (<mailto:profedsonbelem@gmail.com>) 🕒 Segunda, Mar 09, 2020



## Dados dos professores:

- [profedsonbelem@gmail.com](mailto:profedsonbelem@gmail.com) (<mailto:profedsonbelem@gmail.com>) / face 98199-0108 / zap
- [lucianamedeiros.coti@gmail.com](mailto:lucianamedeiros.coti@gmail.com) (<mailto:lucianamedeiros.coti@gmail.com>) 98201-2525
- [alexandreaugusto1901@live.com](mailto:alexandreaugusto1901@live.com) (<mailto:alexandreaugusto1901@live.com>) 98806-9613



## Acompanhe o Blog:

[www.blogedsonbelem.com.br](http://www.blogedsonbelem.com.br) (<http://www.blogedsonbelem.com.br>)



## Indicação de livros:

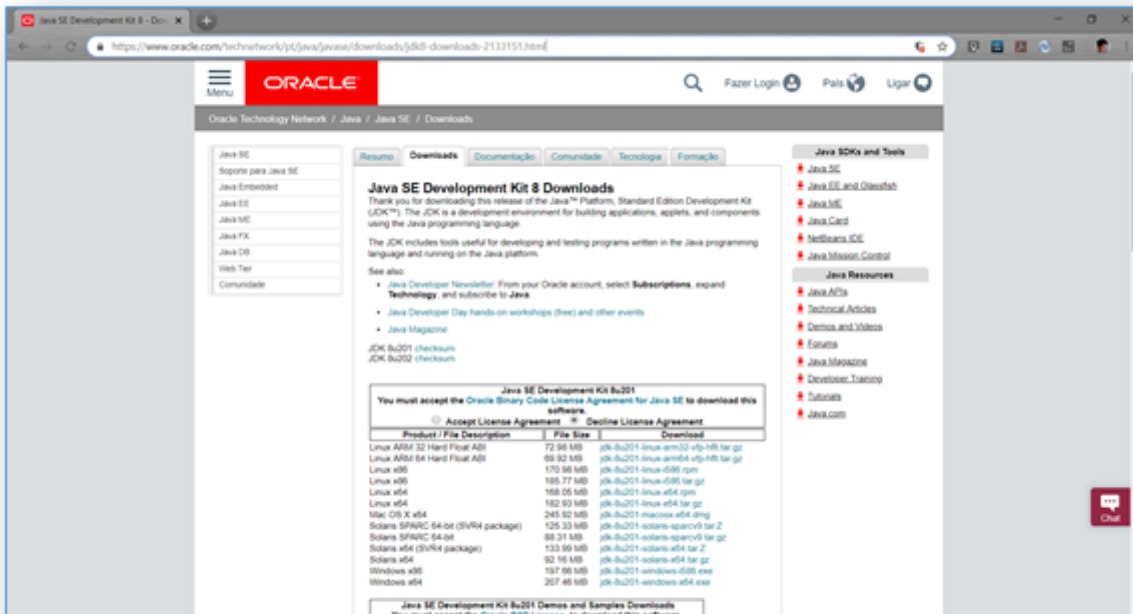
|              |  |  |
|--------------|--|--|
| Kathy Sierra |  |  |
|              |  |  |



## Programas para Baixar:

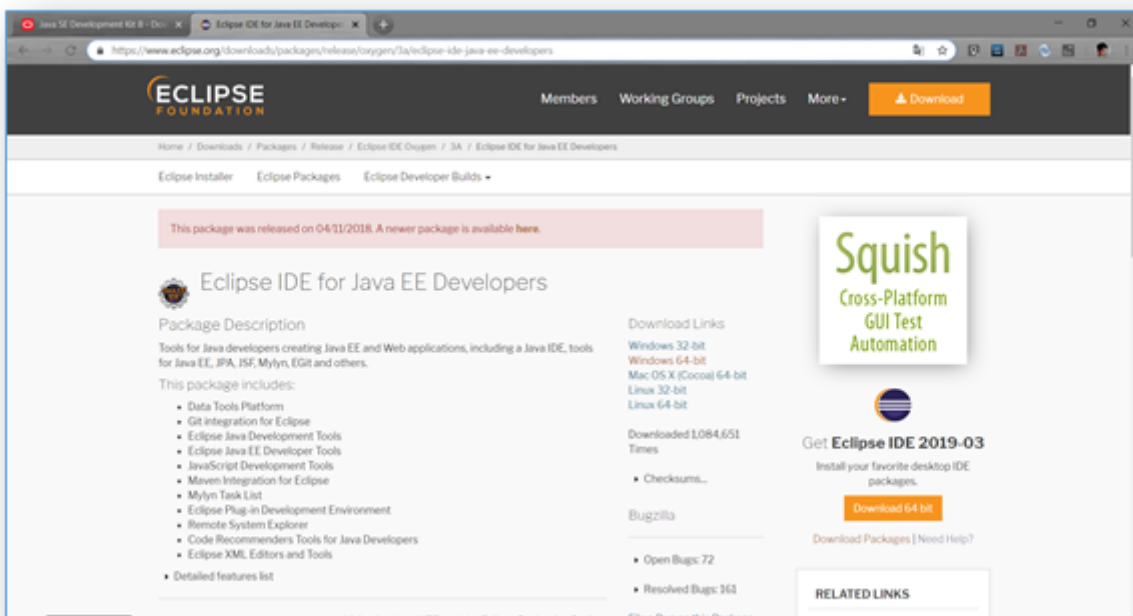
## JDK 8

👉 <https://www.oracle.com/technetwork/pt/java/javase/downloads/jdk8-downloads-2133151.html> (<https://www.oracle.com/technetwork/pt/java/javase/downloads/jdk8-downloads-2133151.html>)



## Eclipse Oxygen (essa é a versão mais estável)

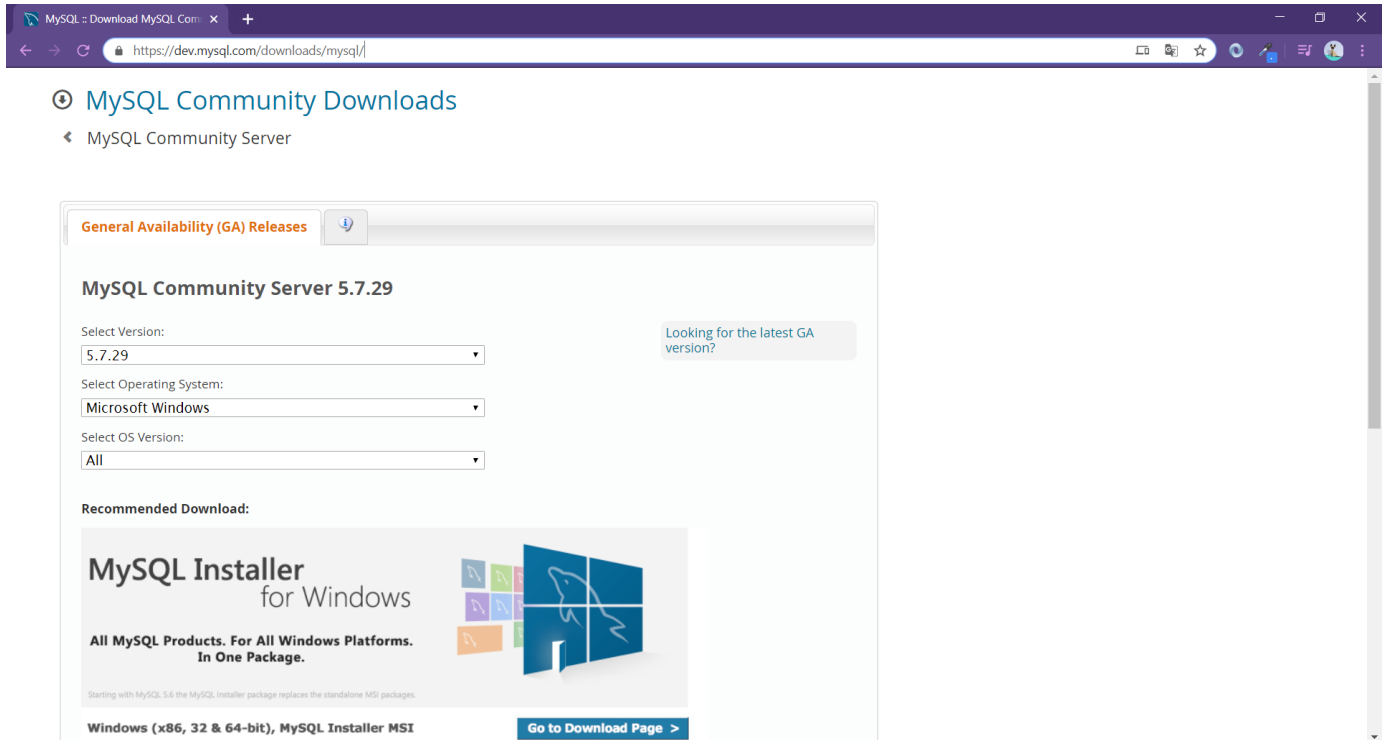
👉 <https://www.eclipse.org/downloads/packages/release/oxygen/3a/eclipse-ide-java-ee-developers> (<https://www.eclipse.org/downloads/packages/release/oxygen/3a/eclipse-ide-java-ee-developers>)



## MySQL

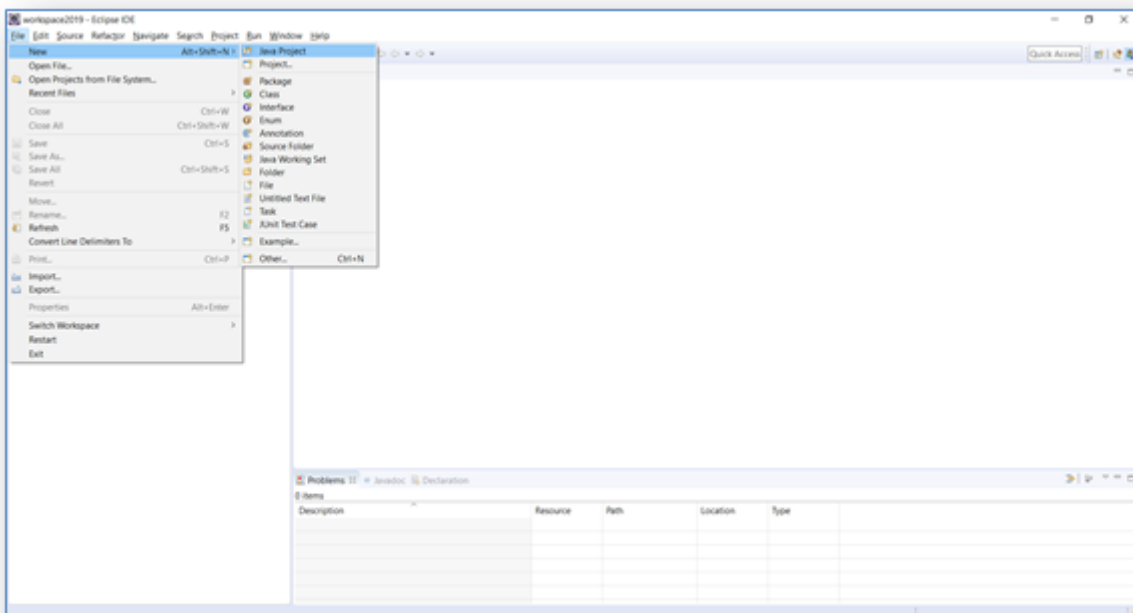
👉 <https://dev.mysql.com/downloads/mysql/> (<https://dev.mysql.com/downloads/mysql/>)

Baixar a versão: **5.7.29** que utilizamos nas aulas.

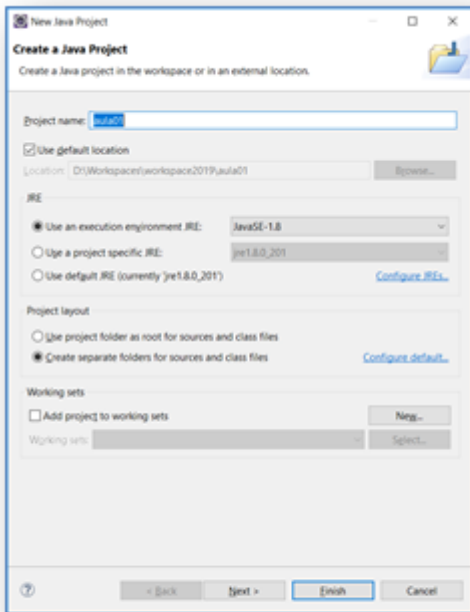


## Criando um projeto local

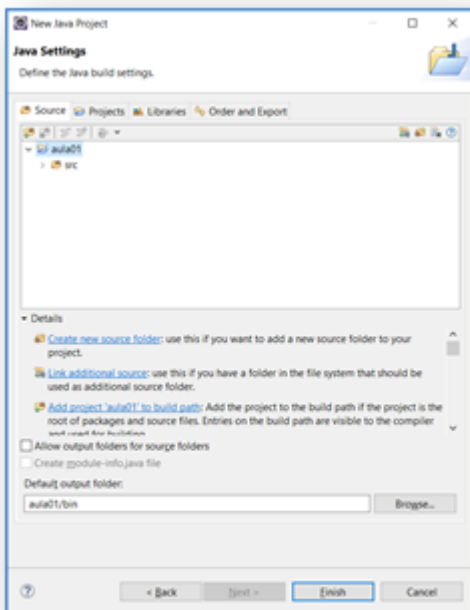
Clicar no menu ➡ File New ➡ Java Project



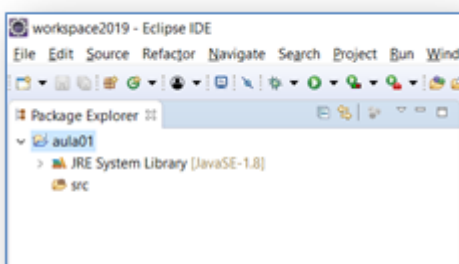
Digitar o nome do projeto ➡ next



Clicar em Finish

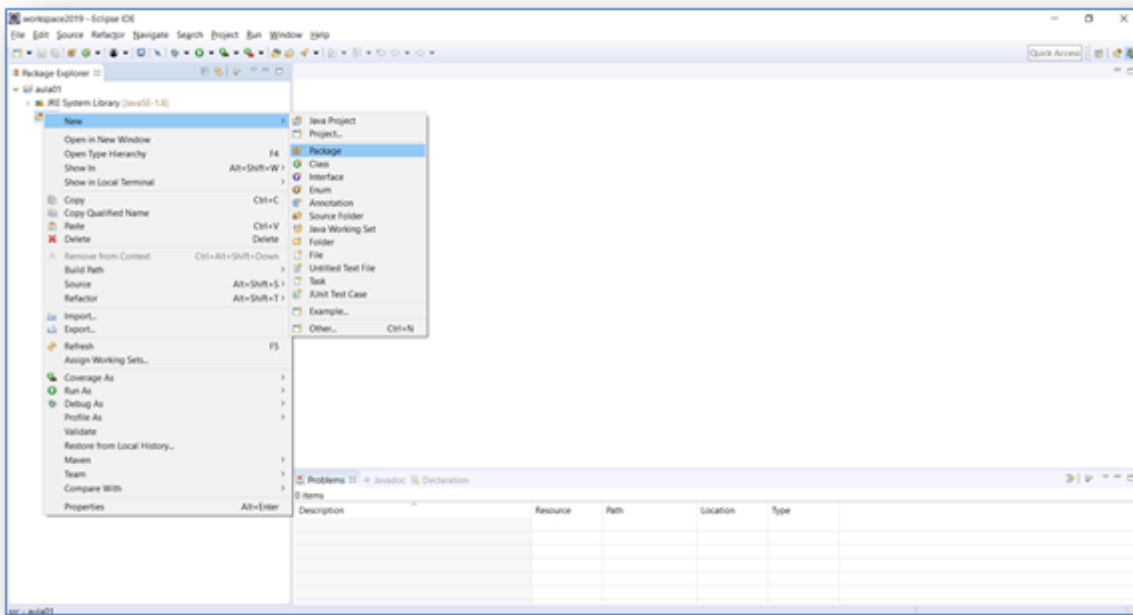


Abrir a estrutura do projeto clicando na seta

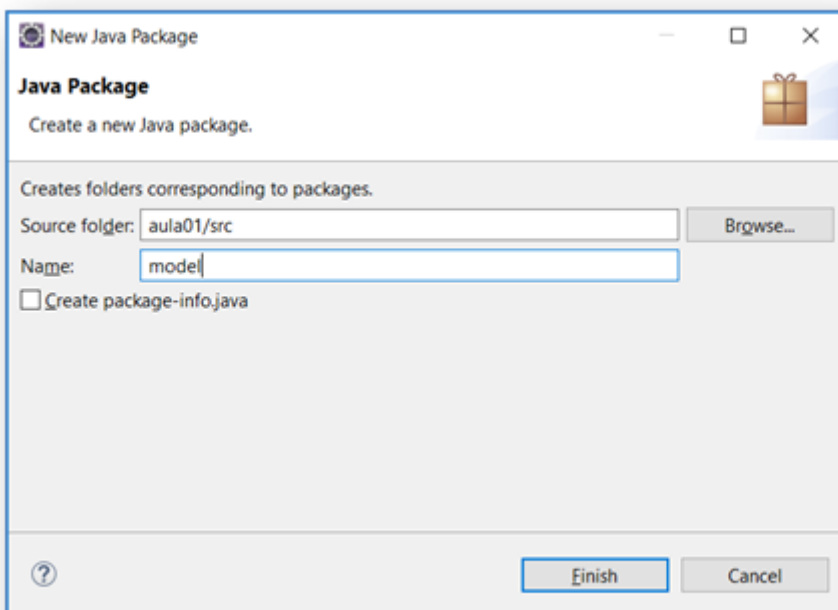


## Para criar o pacote ou diretorio

Clicar em src com o botão direito → new → package

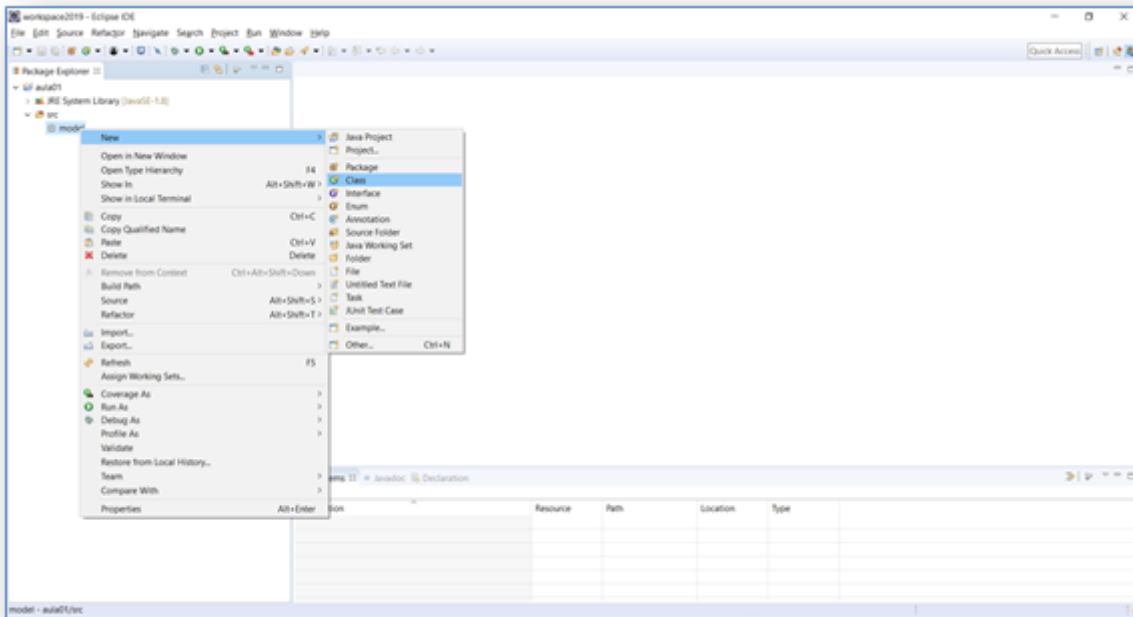


Digitar o nome do pacote → finish

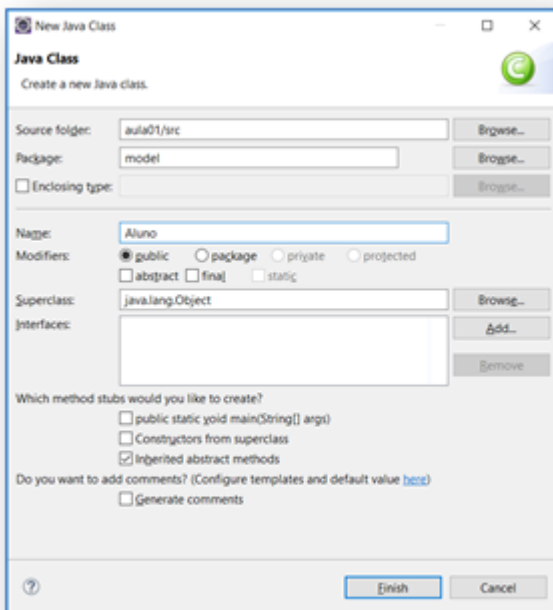


## Para criar a classe

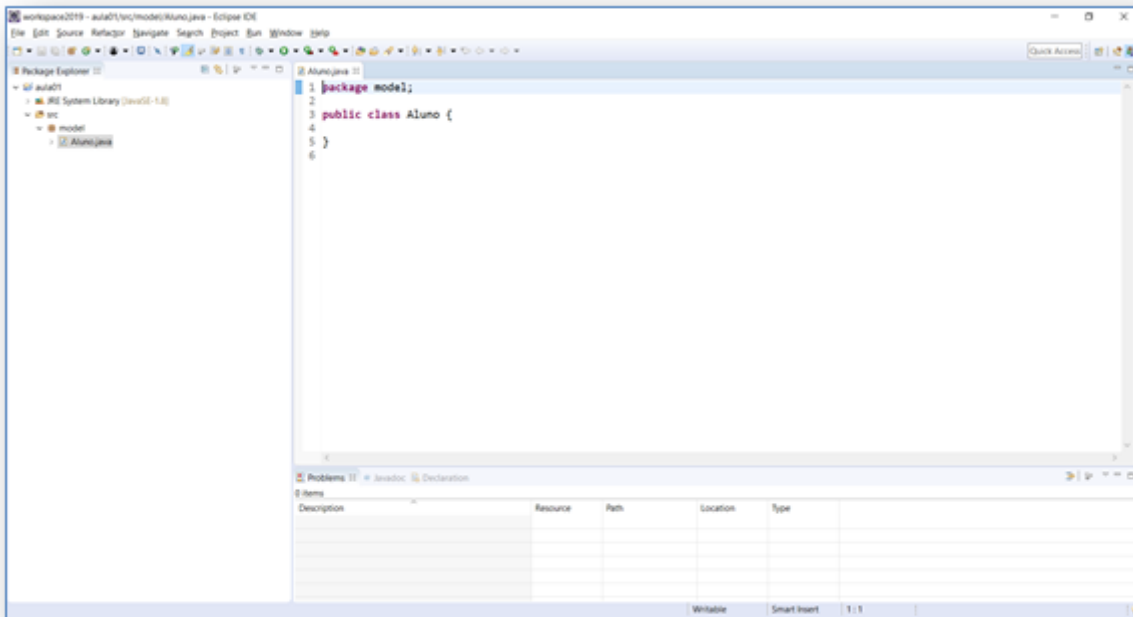
Clicar no pacote com o botão direito → new → class



Digitar o nome da classe (sempre com a letra inicial maiuscula) → FINISH

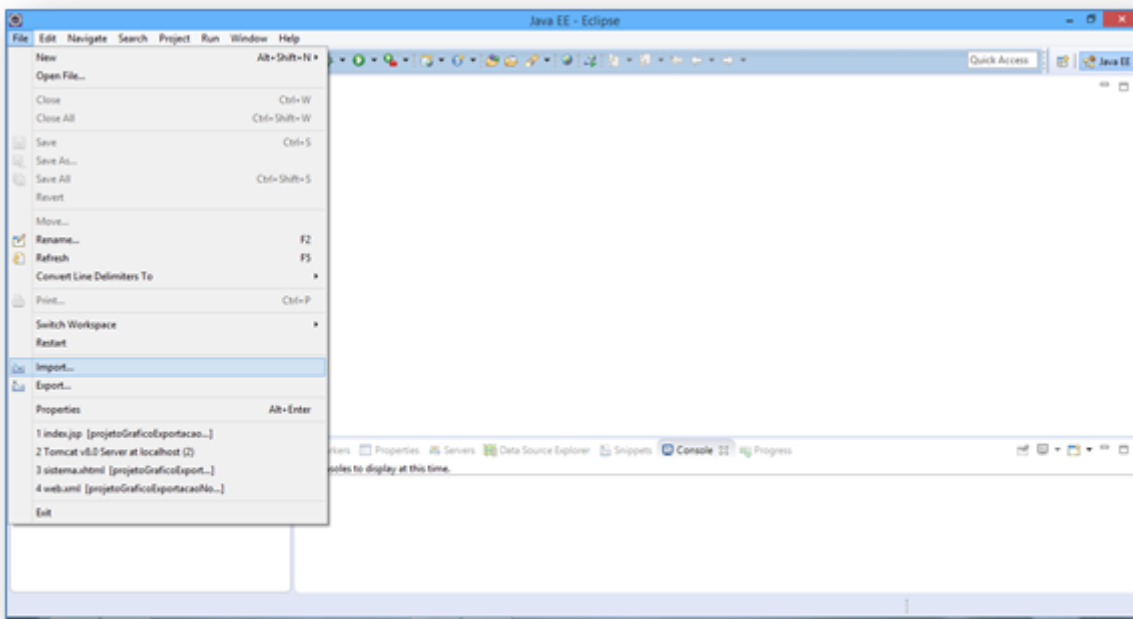


## Classe criada



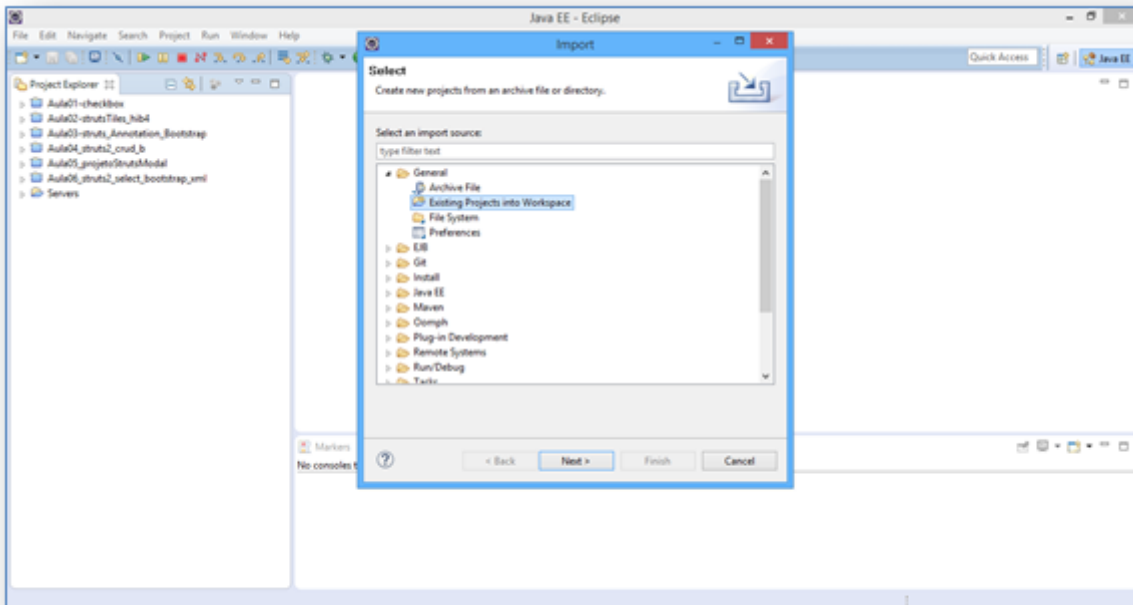
## ⚠ Importando um Projeto Local

Abrir o Eclipse. Clicar em File ➡ Import.

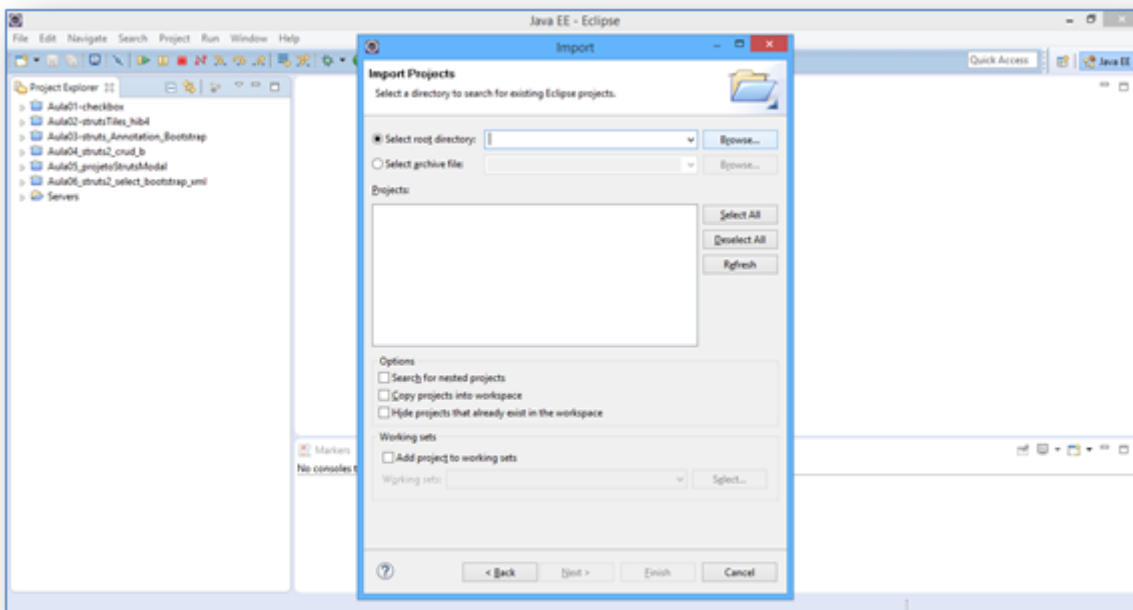




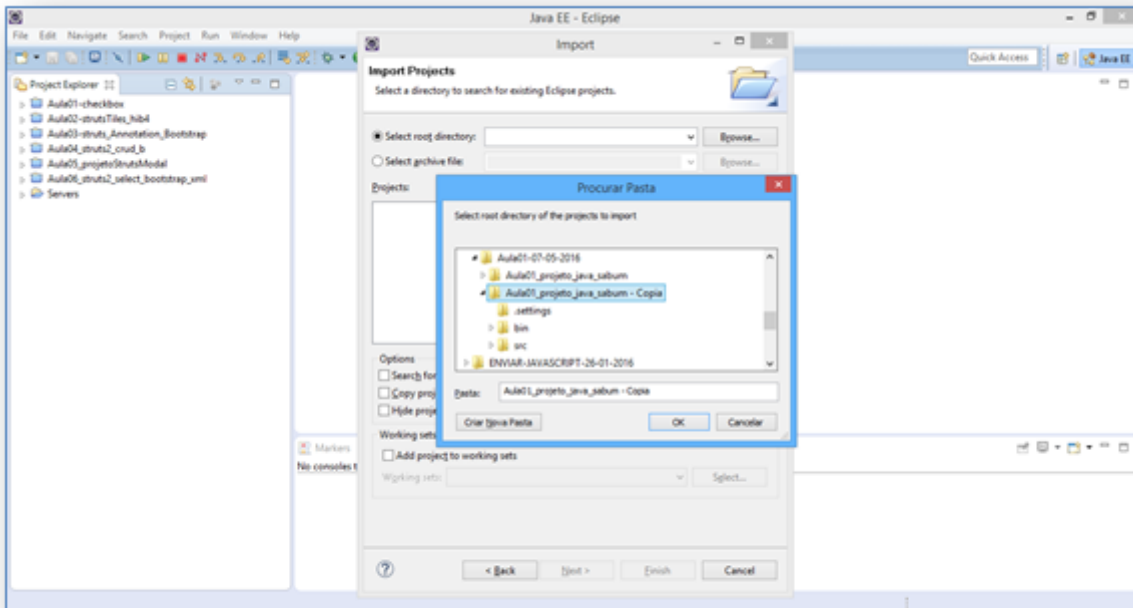
Clicar em general ➡ Existing Projects into Workspace ➡ Next.



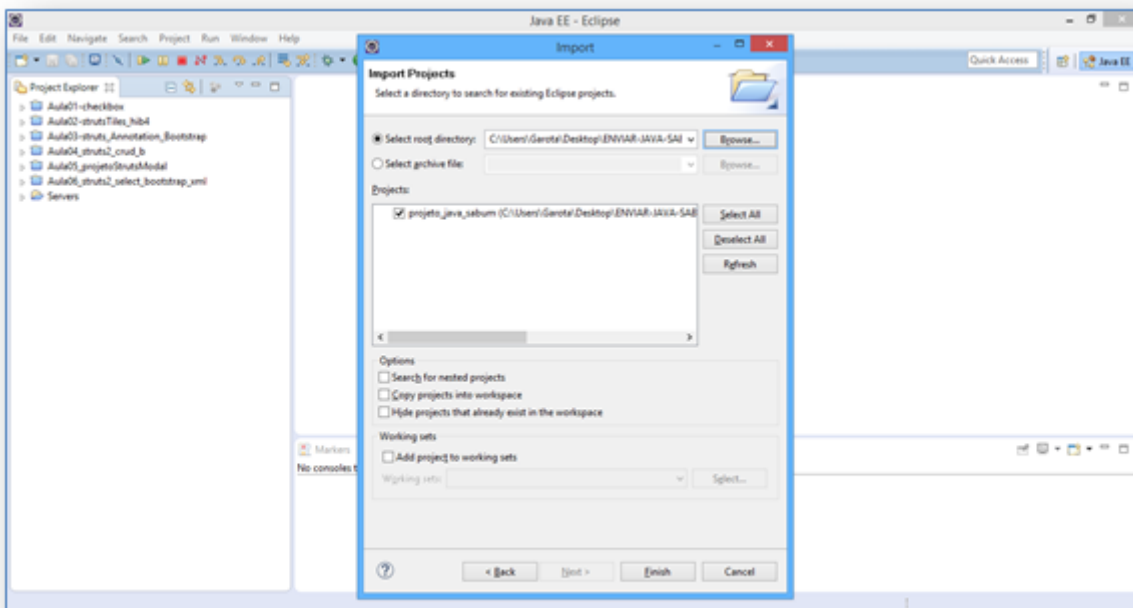
Clicar em Browse e apontar onde está o diretório do projeto.



Clicar na pasta ➡ OK.



Finish.



## Atalhos do Eclipse



## PRODUTIVIDADE COM ECLIPSE IDE

| TECLAS           | O que faz                                      |
|------------------|--|
| CTRL + S         | Salvar a classe                                |
| CTRL + SHIFT + S | Salvar todas as classes                        |
| CTRL + ESPAÇO    | Autocompleta os comandos                       |
| ALT + S          | Atalho para getters and setters e construtores |
| CTRL + SHIFT + O | Faz algumas das importações                    |
| CTRL + W         | Fecha a aba                                    |
| CTRL + SHIFT + W | Fecha todas as abas                            |
| CTRL + D         | Exclui a linha                                 |
| F11              | Rodar o main                                   |
| CTRL + PG UP     | Pula as abas das classes para frente           |
| CTRL + PG DOWN   | Pula as abas das classes para trás             |
| CTRL + SHIFT + F | Formata as o texto                             |
| CTRL + N         | cria novo arquivo                              |
| CTRL + SHIFT + L | Listar todos os atalhos                        |

## Orientação a objetos

A Orientação a Objetos é uma maneira alternativa de pensar os problemas de sistemas de informação utilizando modelos organizados a partir de conceitos do mundo real. O artefato base é o “objeto” capaz de combinar estrutura e comportamento em uma única “entidade”. Tudo o que podemos ver no mundo real é considerado um objeto com atributos e comportamentos definidos. Na qualidade de método de modelagem, é tida como a melhor estratégia para se eliminar a dificuldade recorrente no processo de modelar o mundo real do domínio do problema em um conjunto de componentes de software que seja o mais fiel na sua representação deste domínio.

**Programação Orientada a Objetos** (também conhecida pela sua sigla POO) é um modelo de análise, projeto e programação de software baseado na composição e interação entre diversas unidades chamadas de ‘objetos’. A POO é um dos 4 principais paradigmas de programação (as outras são programação imperativa, funcional e lógica). Os objetos são operados com o conceito de ‘this’ (isso) ou ‘self’ (si), de forma que seus métodos (muitas vezes) modifiquem os dados da própria instância. Os programas são arquitetados através de objetos que interagem entre si. Dentre as várias abordagens da POO, as baseadas em classes são as mais comuns: objetos são instâncias de classes, o que em geral também define o tipo do objeto. Cada classe determina o comportamento (definido nos métodos) e estados possíveis (atributos) de seus objetos, assim como o relacionamento com outros objetos. A alternativa mais usual ao uso de classes é o uso de protótipos. Neste caso, objetos são cópias de outros objetos, não instâncias de classes. Javascript e Lua são exemplos de linguagens cuja POO é realizada por protótipos. A diferença prática mais evidente é que na POO baseada em protótipos apenas a herança simples é implementada pela cópia do objeto. Assim, na POO, implementa-se um conjunto de classes passíveis de serem instanciadas como objetos, e.g. Python e C++ (ou objetos protótipos que são copiados e alterados, e.g. JavaScript e VimL).

Em alguns contextos, o termo modelagem orientada ao objeto (MOO) é preferível ao termo POO. De fato, o paradigma “orientado ao objeto” tem origem nos estudos da cognição e influenciou a inteligência artificial e a linguística, dada a relevância para a abstração de conceitos do mundo real. A MOO é considerada a melhor estratégia para diminuir o “gap semântico” (o hiato entre o mundo real e a representação dele), e facilita a comunicação das partes interessadas no modelo ou software (e.g. o modelador e o usuário final) na medida em que conceitos, terminologia, símbolos, grafismo e estratégias, são, potencialmente, mais óbvios, intuitivos, naturais e exatos.

Muitas das linguagens de programação mais utilizadas atualmente (talvez a maioria) são multi-paradigma com suporte à POO. C++, C#, VB.NET (<http://VB.NET>), Java, Object Pascal, Objective-C, Python, SuperCollider, Ruby e Smalltalk são exemplos de linguagens de programação orientadas a objetos. ActionScript, ColdFusion, Javascript, PHP (a partir da versão 4.0), Perl (a partir da versão 5), Visual Basic (a partir da versão 4), VimL (ou Vim script) são exemplos de linguagens de programação com suporte a orientação a objetos. Vivace é um exemplo de linguagem sem suporte à POO.

Os **atributos e métodos** podem ser referentes a uma classe (e todas as suas instâncias) ou a uma única instância. O vínculo dos atributos aos métodos, de forma a manter uma interface bem definida para operação sobre os dados, e a evitar corrupção dos dados, é chamado de encapsulamento. O encapsulamento foi responsável pelo conceito de 'ocultamento de dados', central para a POO. O encapsulamento pode ser realizado através de convenções (em Python, underscores demarcam métodos e atributos protegidos e privados), ou via recursos da linguagem (em Java ou C++, um método privado só é acessado pela própria classe). Encapsulamento incentiva o desacoplamento.

Quando um objeto contém outros objetos, diz-se que há composição de objetos. A composição de objetos é usada para representar uma relação 'tem um', usualmente uma meronímia. Já a herança (quase sempre suportada pelas linguagens que utilizam classes) apresenta relações 'é um' (i.e. 'é um tipo de'), ou seja, relações de hiperonímia cujo resultado final é a árvore taxonômica. Na herança, tipicamente todos os atributos e métodos da classe pai/mãe estão também disponíveis na classe filha, embora seja comum que algumas características sejam substituídas. Assim, a herança permite reuso facilitado de características e muitas vezes reflete relações do mundo real de forma intuitiva. Ambas a 'composição de objetos' e a 'herança' constituem hierarquias entre as classes e objetos na POO.

## Linguagem Java

Java é a base de praticamente todos os tipos de aplicativos em rede, e é o padrão global para desenvolvimento e fornecimento de aplicativos para celular, jogos, conteúdo on-line e software corporativo. Com mais de 9 milhões de desenvolvedores em todo o mundo, o Java permite desenvolver e implantar aplicativos e serviços incríveis de maneira eficiente. Com ferramentas abrangentes, um ecossistema sólido e um desempenho eficiente, o Java oferece a portabilidade de aplicativos mesmo entre os ambientes computacionais mais diferentes.

## Modificadores de acesso do Java

Em Java, podemos criar atributos ou métodos com os seguintes modificadores de acesso:

- **public** - Acesso total, qualquer classe tem acesso ao método (não indicado para usar em atributos de entidade).
- **protected** - Acesso permitido por herança e por classes do mesmo pacote.
- **default / friendly** - Acesso permitido somente a classes do mesmo pacote, este acesso é definido quando não declaramos nenhum modificador.
- **private** - Acesso permitido somente dentro da própria classe. (indicado para atributos de classes de entidade).

## Modelagem de dados em Java com UML

---

O Objetivo deste artigo é fornecer uma introdução à modelagem de Classes em Java utilizando diagramas UML. Neste exemplo iremos abordar o primeiro tipo de relacionamento aprendido em Orientação a Objetos: SER (É-UM) utilizado entre Classes e também entre Interfaces. Para tal iremos partir do principio, ou seja, o tipo mais "puro" de objeto Java, denominado JavaBean

## Classe JavaBean

Classe Java de Entidade (Modelagem)

Características:

- Atributos privados.
- Construtor default (vazio)
- Sobrecarga de Construtor (Entrada de dados)
- Encapsulamento
- Sobrescrita dos métodos de Object
- toString
- equals
- hashCode
- Serialização (Opcional)

```
1  package heranca;
2
3  public class Cliente {
4
5      private Integer idCliente;
6      private String nome;
7
8      public Cliente() {
9      }
10
11     public Cliente(Integer idCliente, String nome) {
12         super();
13         this.idCliente = idCliente;
14         this.nome = nome;
15     }
16
17     @Override
18     public String toString() {
19         return idCliente + ", " + nome;
20     }
21
22     public Integer getIdCliente() {
23         return idCliente;
24     }
25     public void setIdCliente(Integer idCliente) {
26         this.idCliente = idCliente;
27     }
28     public String getNome() {
29         return nome;
30     }
31     public void setNome(String nome) {
32         this.nome = nome;
33     }
34 }
```

| Cliente  |
|--|
| - idCliente : Integer<br>- nome : String   |
| + Cliente()<br>+ Cliente(idCliente : Integer, nome : String)<br>+ toString() : String<br>+ getIdCliente() : Integer<br>+ setIdCliente(idCliente : Integer) : void<br>+ getNome() : String<br>+ setNome(nome : String) : void |

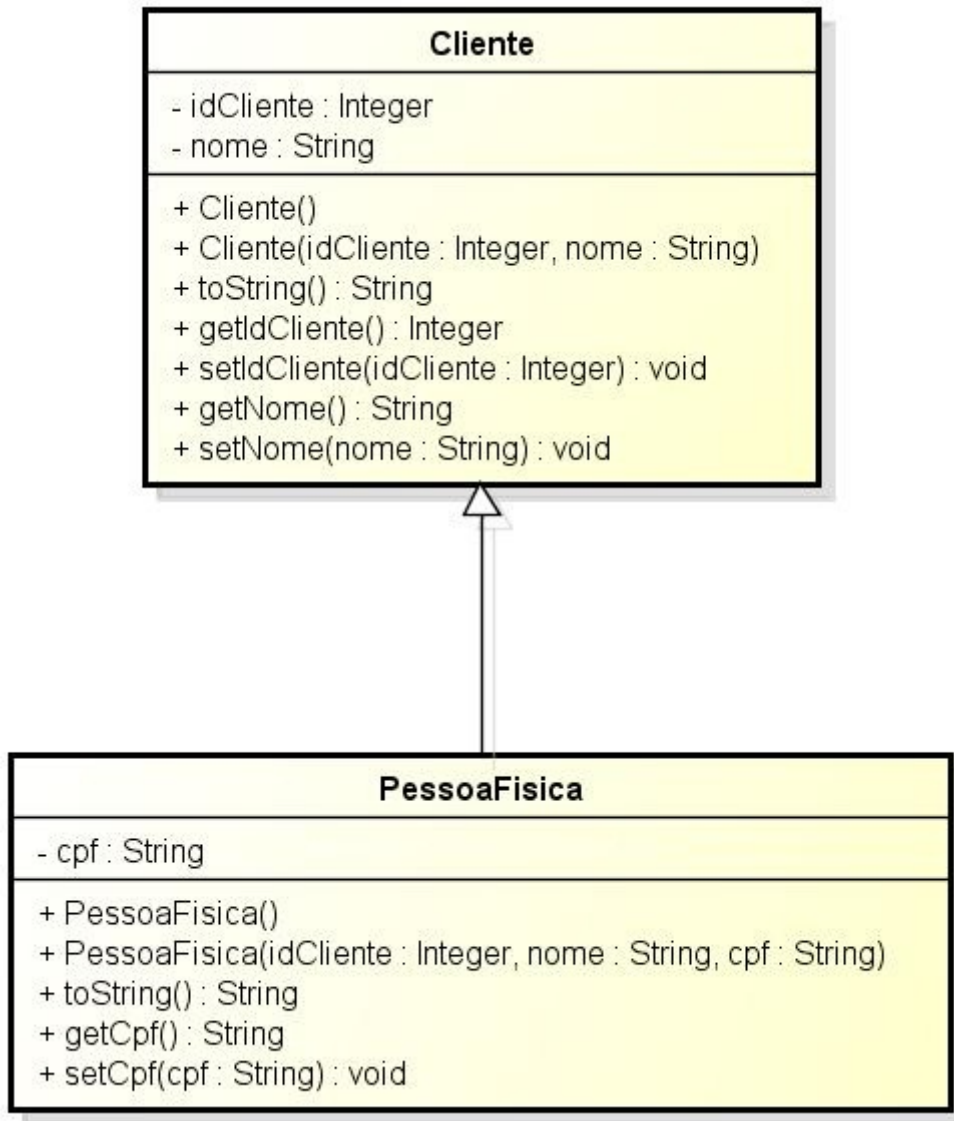
Visibilidades:

- -private Acesso somente dentro da Classe.

- ~default Acesso somente por classes do mesmo pacote.
- #protected Acesso por pacote e por herança.
- +public Acesso total.

```
1  package heranca;
2
3  public class PessoaFisica extends Cliente {
4
5      private String cpf;
6
7      public PessoaFisica() {
8      }
9
10     public PessoaFisica(Integer idCliente, String nome, String cpf) {
11         super(idCliente, nome);
12         this.cpf = cpf;
13     }
14
15     @Override
16     public String toString() {
17         return super.toString() + ", " + cpf;
18     }
19
20     public String getCpf() {
21         return cpf;
22     }
23     public void setCpf(String cpf) {
24         this.cpf = cpf;
25     }
26 }
```





## Implementação (É-UM)

Componente de programação OO totalmente abstrato. Tem como características: Padronização.

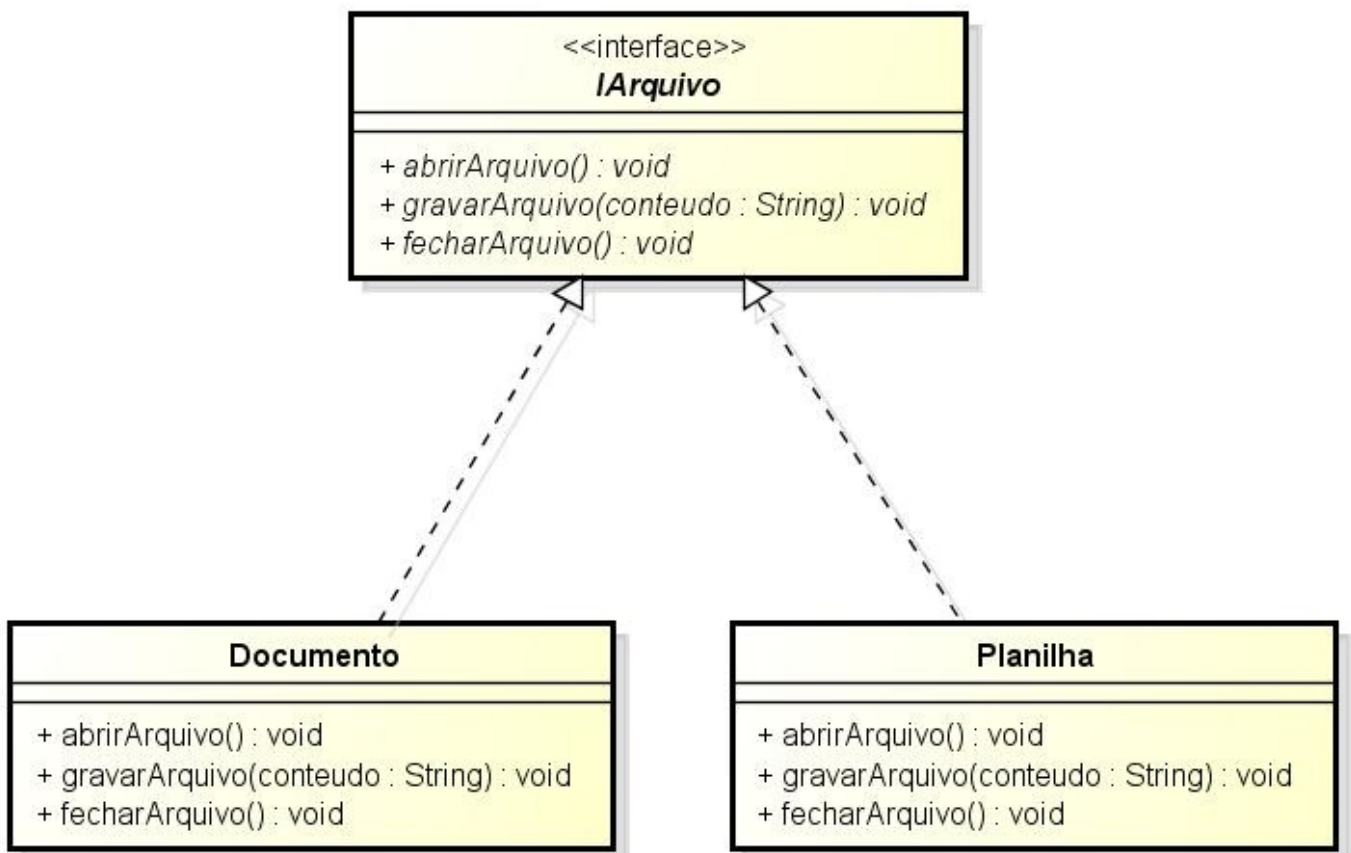
- Interfaces Não possuem construtores.
- Seus atributos são final (constantes).
- Métodos são públicos e abstratos (não tem corpo).
- Quando uma classe implementa uma interface, ela é obrigada a fornecer corpo para os métodos da interface (desde que não seja uma classe abstrata).

```

1 package relacionamento;
2
3 public interface IArquivo {
4
5     void abrirArquivo();
6     void gravarArquivo(String conteudo);
7     void fecharArquivo();
8 }
  
```

Quando uma Classe não abstrata (comum) herda (implementa) uma interface, esta é obrigada a fornecer corpo para os metodos da interface. Por exemplo:

```
1  package relacionamento;
2
3  public class Documento implements IArquivo {
4
5      @Override
6      public void abrirArquivo() {
7          // TODO Auto-generated method stub
8      }
9
10     @Override
11     public void gravarArquivo(String conteudo) {
12         // TODO Auto-generated method stub
13     }
14
15     @Override
16     public void fecharArquivo() {
17         // TODO Auto-generated method stub
18     }
19 }
```



UML, qualquer nome em *itálico*, representa algo abstrato ou interface. No próximo artigo iremos estudar o segundo tipo de relacionamento entre Classes denominado TER (Todo/Parte) e suas variações como Agregação, Composição, Dependência e suas Multiplicidades.

```
1 package entity2;
2
3 public abstract class Automovel {
4
5     private Integer idAutomovel;
6     private String nome;
7
8     public Automovel() {
9     }
10
11     public Automovel(Integer idAutomovel, String nome) {
12         this.idAutomovel = idAutomovel;
13         this.nome = nome;
14     }
15
16     @Override
17     public String toString() {
18         return idAutomovel + ", " + nome;
19     }
20
21     public Integer getIdAutomovel() {
22         return idAutomovel;
23     }
24     public void setIdAutomovel(Integer idAutomovel) {
25         this.idAutomovel = idAutomovel;
26     }
27     public String getNome() {
28         return nome;
29     }
30     public void setNome(String nome) {
31         this.nome = nome;
32     }
33
34     // Métodos abstratos
35     public abstract void setFabricante(String fabricante);
36     public abstract String getFabricante();
37 }
```

## Classes comuns herdando da Classe Abstrata

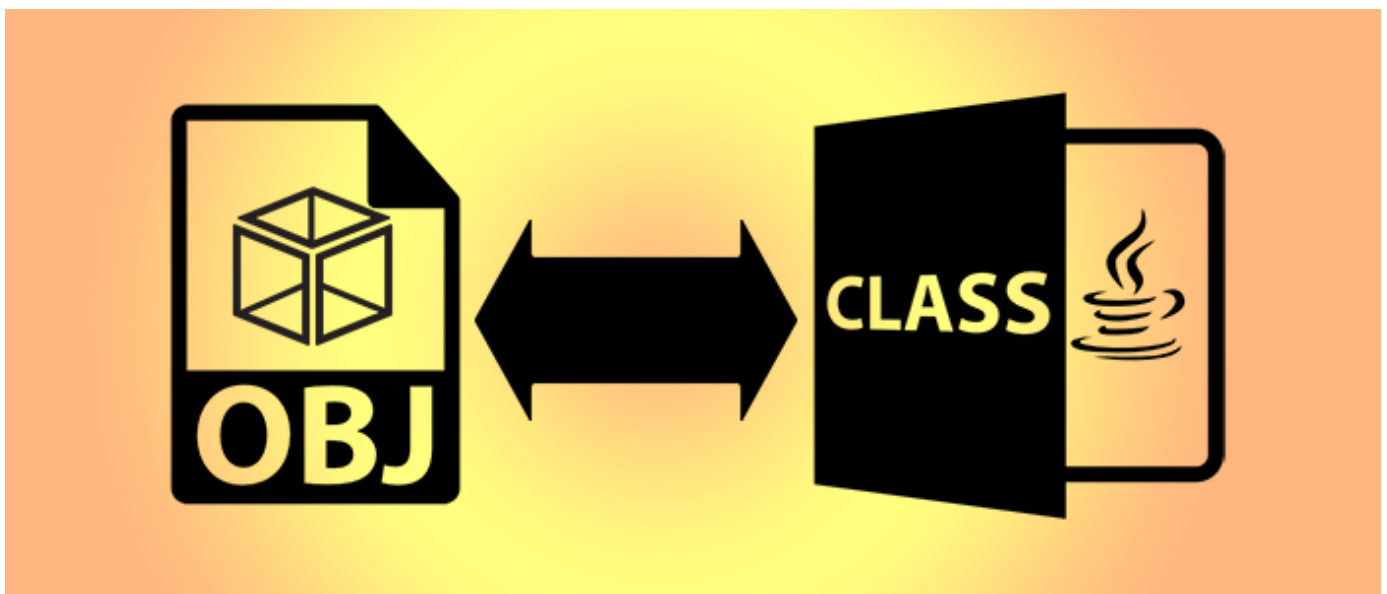
```
1  package entity2;
2
3  public class CarroEsportivo extends Automovel {
4
5      private Integer ano;
6      private String fabricante;
7
8      public CarroEsportivo() {
9
10     }
11
12     public CarroEsportivo(Integer idAutomovel, String nome,
13 Integer ano, String fabricante) {
14         super(idAutomovel, nome);
15         this.ano = ano;
16         this.fabricante = fabricante;
17     }
18
19     @Override
20     public String toString() {
21         return super.toString() + ", " + ano + ", " + fabricante;
22     }
23
24     public Integer getAno() {
25         return ano;
26     }
27     public void setAno(Integer ano) {
28         this.ano = ano;
29     }
30
31     @Override
32     public String getFabricante() {
33         return fabricante;
34     }
35
36     @Override
37     public void setFabricante(String fabricante) {
38         this.fabricante = fabricante;
39     }
40 }
```

```
1  package entity2;
2
3  public class CarroExecutivo extends Automovel {
4
5      private String modelo;
6      private String fabricante;
7
8      public CarroExecutivo() {
9
10     }
11
12     public CarroExecutivo(Integer idAutomovel, String nome,
13 String modelo, String fabricante) {
14         super(idAutomovel, nome);
15         this.modelo = modelo;
16         this.fabricante = fabricante;
17     }
18
19     @Override
20     public String toString() {
21         return super.toString() + ", " + modelo + ", " + fabricante;
22     }
23
24     public String getModelo() {
25         return modelo;
26     }
27     public void setModelo(String modelo) {
28         this.modelo = modelo;
29     }
30
31     @Override
32     public String getFabricante() {
33         return fabricante;
34     }
35
36     @Override
37     public void setFabricante(String fabricante) {
38         this.fabricante = fabricante;
39     }
40 }
```

```
1 package main;
2
3 import entity2.Automovel;
4 import entity2.CarroEsportivo;
5 import entity2.CarroExecutivo;
6
7 public class Main {
8
9     public static void main(String[] args) {
10
11         Automovel a1 = new CarroEsportivo(1, "Ferrari", 2012, "Ferrari Italia");
12         Automovel a2 = new CarroExecutivo(2, "C4", "Sedan", "Citroen");
13
14         System.out.println(a1);
15         System.out.println(a2);
16     }
17 }
```

No exemplo acima podemos dizer que CarroEsportivo É Automovel e CarroExecutivo É Automovel, portanto, a herança da Classe abstrata configura o uso de Polimorfismo. Note que Transformamos o objeto Automovel em CarroEsportivo e CarroExecutivo

## O que é uma classe



Uma classe define estado e comportamento de um Objeto geralmente implementando métodos e atributos (nomes utilizados na maioria das linguagens modernas). Os atributos, também chamados de campos (do inglês fields), indicam as possíveis informações armazenadas por um objeto de uma classe, representando o estado de cada objeto. Os métodos são procedimentos que formam os comportamentos e serviços oferecidos por objetos de uma classe.

Uma classe é uma estrutura que abstrai um conjunto de objetos com características similares. Uma classe define o comportamento de seus objetos - através de métodos - e os estados possíveis destes objetos - através de atributos. Em outras palavras, uma classe descreve os serviços oferecidos por seus objetos e quais informações eles podem armazenar. Classes não são diretamente suportadas em todas as linguagens, e são necessárias para que uma linguagem seja orientada a objetos. Uma classe representa um conjunto de objetos com características afins. Uma classe define o comportamento dos objetos através de seus métodos, e quais estados ele é capaz de manter através de seus atributos.

## Classe Java Beans

---

JavaBeans são componentes de software escritos na linguagem de programação Java. Segundo a especificação da Sun Microsystems os JavaBeans são "componentes reutilizáveis de software que podem ser manipulados visualmente com a ajuda de uma ferramenta de desenvolvimento".

Um bean também pode ser definido como uma classe Java que expõe propriedades, seguindo uma convenção de nomenclatura simples para os métodos getter e setter.

Praticamente são classes escritas de acordo com uma convenção em particular. São usados para encapsular muitos objetos em um único objeto (o bean), assim eles podem ser transmitidos como um único objeto em vez de vários objetos individuais.

São características de uma Classe JavaBean:

- Atributos privados
- Construtores
  - Vazio (sem argumentos)
  - Com entrada de argumetos (Sobrecarga)
- Métodos set e get (encapsulamento)
- Sobrescrita dos métodos da Classe Object
  - toString
  - equals
  - hashCode

## Objeto

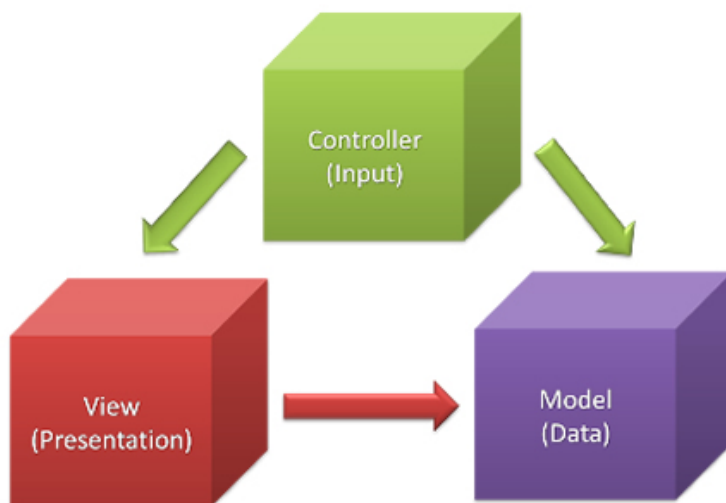
---

É uma instância de uma Classe. Armazenamento de estados através de seus atributos e reação a mensagens enviadas por outros objetos.

|          |          |                          |
|----------|----------|--------------------------|
| Cliente  | c1 =     | new Cliente();           |
| [Classe] | [Objeto] | [Construtor → Instância] |

Um objeto, na vida real, é qualquer coisa a qual pudermos dar um nome. Um objeto, em programação orientada a objetos, é uma instância (ou seja, um exemplar) de uma classe. A Wikilivros é um exemplo de Wiki, ou, a Wikilivros é uma instância de Wiki. Isto poderia ser representado em um programa orientado a objetos com uma classe chamada Wiki e um objeto do tipo Wiki chamado Wikilivros. Um objeto é capaz de armazenar estados através de seus atributos e reagir a mensagens enviadas a ele, assim como se relacionar e enviar mensagens a outros objetos. **Atributos** são características de um objeto. Basicamente a estrutura de dados que vai representar a classe. Exemplos: um objeto da classe "Funcionário" teria como atributos "nome", "endereço", "telefone", "CPF", etc. O conjunto de valores dos atributos de um determinado objeto é chamado de estado.

## Padrão MVC - Model, View e Controller



## Padrão MVC (Model, View e Controller)

Esse padrão tem a finalidade de dividir o projeto em três camadas:

- 1- Camada Modelo: A camada Modelo tem as classes de dados que define o domínio da aplicação, por exemplo, classe aluno, professor, turma se sua aplicação for de controle de turma.
- 2- Camada View: A camada View corresponde a qualquer entrada de dados da aplicação, por exemplo, leitura de dados pelo console, paginas jsp, xhtml entre outras.
- 3- Camada Control: A camada control corresponde à regra de negocio da aplicação e centralizando as informações vindas da camada view, passando para camada de modelo e voltando para camada de view.

### Camada Model



```
1  package modelo;
2
3  public class Pessoa {
4
5      private Integer idPessoa;
6      private String nome;
7      private Integer idade;
8
9      public Pessoa() {
10     }
11
12     public Pessoa(Integer idPessoa, String nome, Integer idade) {
13         super();
14         this.idPessoa = idPessoa;
15         this.nome = nome;
16         this.idade = idade;
17     }
18
19     @Override
20     public String toString() {
21         return "Pessoa [idPessoa=" + idPessoa + ", nome=" + nome +
22         ", idade=" + idade + "]";
23     }
24
25     public Integer getIdPessoa() {
26         return idPessoa;
27     }
28     public void setIdPessoa(Integer idPessoa) {
29         this.idPessoa = idPessoa;
30     }
31     public String getNome() {
32         return nome;
33     }
34     public void setNome(String nome) {
35         this.nome = nome;
36     }
37     public Integer getIdade() {
38         return idade;
39     }
40     public void setIdade(Integer idade) {
41         this.idade = idade;
42     }
43 }
```

## Camada Controle

```
1  package controle;  
2  
3  import java.util.regex.Matcher;  
4  import java.util.regex.Pattern;  
5  
6  import modelo.Pessoa;  
7  
8  public class ValidaPessoa {  
9  
10     public void validaCodigo(Pessoa p)throws Exception{  
11         if (p.getIdPessoa()<=0){  
12             throw new Exception  
13             ("Codigo Menor ou igual a zero invalido ...");  
14         }  
15     }  
16  
17     public void validaIdade(Pessoa p)throws Exception{  
18         if (p.getIdade()<18){  
19             throw new Exception  
20             ("So pode ser cadastrado maior de idade");  
21         }  
22     }  
23  
24     public void validaNome(Pessoa p)throws Exception{  
25         Pattern pa = Pattern.compile("[A-Z a-z]{3,35}");  
26         Matcher ma = pa.matcher(p.getNome());  
27         if (! ma.matches()){  
28             throw new Exception  
29             ("Nome somente com Letras faixa (3,35)");  
30             }//se for falso lança o erro  
31     }  
32 }
```

## Camada View

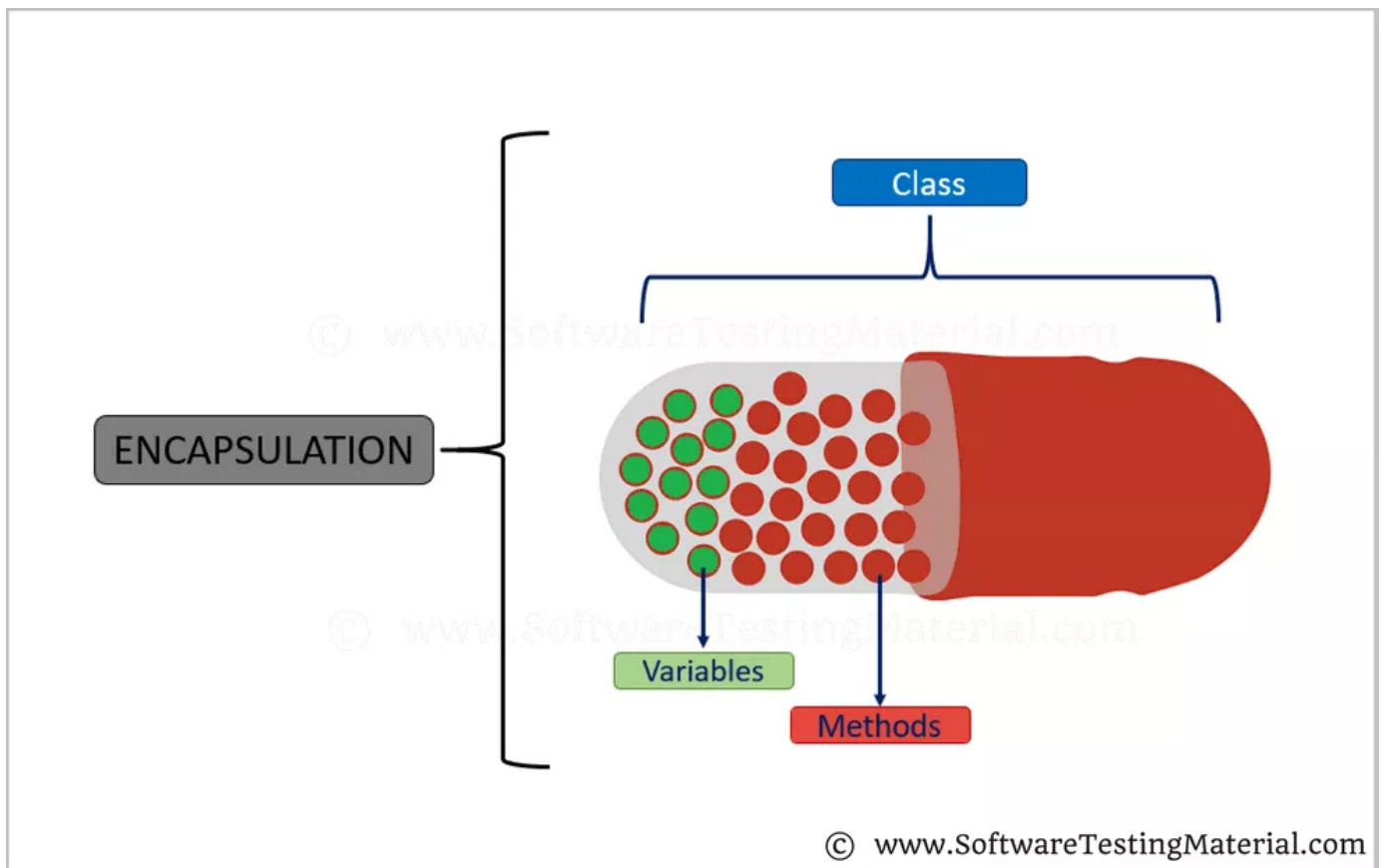
```
1  package view;
2
3  import java.util.Scanner;
4
5  public class InputPessoa {
6
7      public Integer lerCodigo() {
8          try {
9              System.out.println("Digite oCodigo :");
10             Scanner in = new Scanner(System.in);
11             return in.nextInt();
12         } catch (Exception ex) {
13             System.out.println("Digite numero inteiro ...");
14             return lerCodigo();
15         }
16     }
17
18     public Integer lerIdade() {
19         try {
20             System.out.println("Digite a Idade :");
21             Scanner in = new Scanner(System.in);
22             return in.nextInt();
23         } catch (Exception ex) {
24             System.out.println("Digite idade inteira ...");
25             return lerIdade();
26         }
27     }
28
29     public String lerNome() {
30         System.out.println("Digite o Nome :");
31         Scanner in = new Scanner(System.in);
32         return in.nextLine();
33     }
34 }
```

## Execução do Programa no Main

```
1  package main;
2
3  import modelo.Pessoa;
4  import view.InputPessoa;
5  import controle.ValidaPessoa;
6
7  public class Main {
8
9      public static void main(String[] args) {
10          Pessoa p = new Pessoa();
11          InputPessoa ip = new InputPessoa();
12          ValidaPessoa vp = new ValidaPessoa();
13
14          try{
15              p.setIdPessoa(ip.lerCodigo());
16              p.setNome(ip.lerNome());
17              p.setIdade(ip.lerIdade());
18              vp.validaCodigo(p);
19              vp.validaNome(p);
20              vp.validaIdade(p);
21
22              System.out.println("Dados" + p);
23          }catch(Exception ex){
24              System.out.println("Error" + ex.getMessage());
25          }
26      }
27  }
```

## Encapsulamento

---



Em linguagens orientadas a objetos, é possível encapsular o estado de um objeto. Em termos práticos, isso se realiza limitando o acesso a atributos de uma classe exclusivamente através de seus métodos. Para isso, as linguagens orientadas a objeto oferecem limitadores de acesso para cada membro de uma classe.

Tipicamente os limitadores de acesso são:

- **público (public)** - o membro pode ser acessado por qualquer classe. Os membros públicos de uma classe definem sua interface
- **protegido (protected)** - o membro pode ser acessado apenas pela própria classe e suas sub-classes
- **privado (private)** - o membro pode ser acessado apenas pela própria classe

Cada linguagem de programação pode possuir limitadores de acesso próprios. Por exemplo, em Java, o nível de acesso padrão de um membro permite que qualquer classe de seu pacote (package) possa ser acessado. Em C#, o limitador de acesso interno (internal) permite que o membro seja acessado por qualquer classe do Assembly (isto é, da biblioteca ou executável).

É um conceito da orientação a objetos que significa como pode ser feito a entrada e saída de dados de uma classe. Na Linguagem existem duas formas para isso. A primeira seria pelos métodos gets ( Saída de Dados ) e sets ( Entrada de Dados ) e a outra seria pelo construtor cheio ( Entrada de Dados ) e o toString ( Saída de Dados ). E os atributos da classe sendo private.

| Aluno   |
|---|
| <ul style="list-style-type: none"><li>- idAluno : Integer</li><li>- nomeAluno : String</li><li>- nota1 : Double</li><li>- nota2 : Double</li></ul>  |
| <ul style="list-style-type: none"><li>+ setIdAluno(idAluno : Integer) : void</li><li>+ getIdAluno() : Integer</li><li>+ setNomeAluno(nomeAluno : String) : void</li><li>+ getNomeAluno() : String</li><li>+ setNota1(nota1 : Double) : void</li><li>+ getNota1() : Double</li><li>+ setNota2(nota2 : Double) : void</li><li>+ getNota2() : Double</li><li>+ Aluno(idAluno : Integer, param1 : String, param2 : Double, param3 : Double) : void</li><li>+ toString() : String</li><li>+ Aluno() : void</li></ul> |

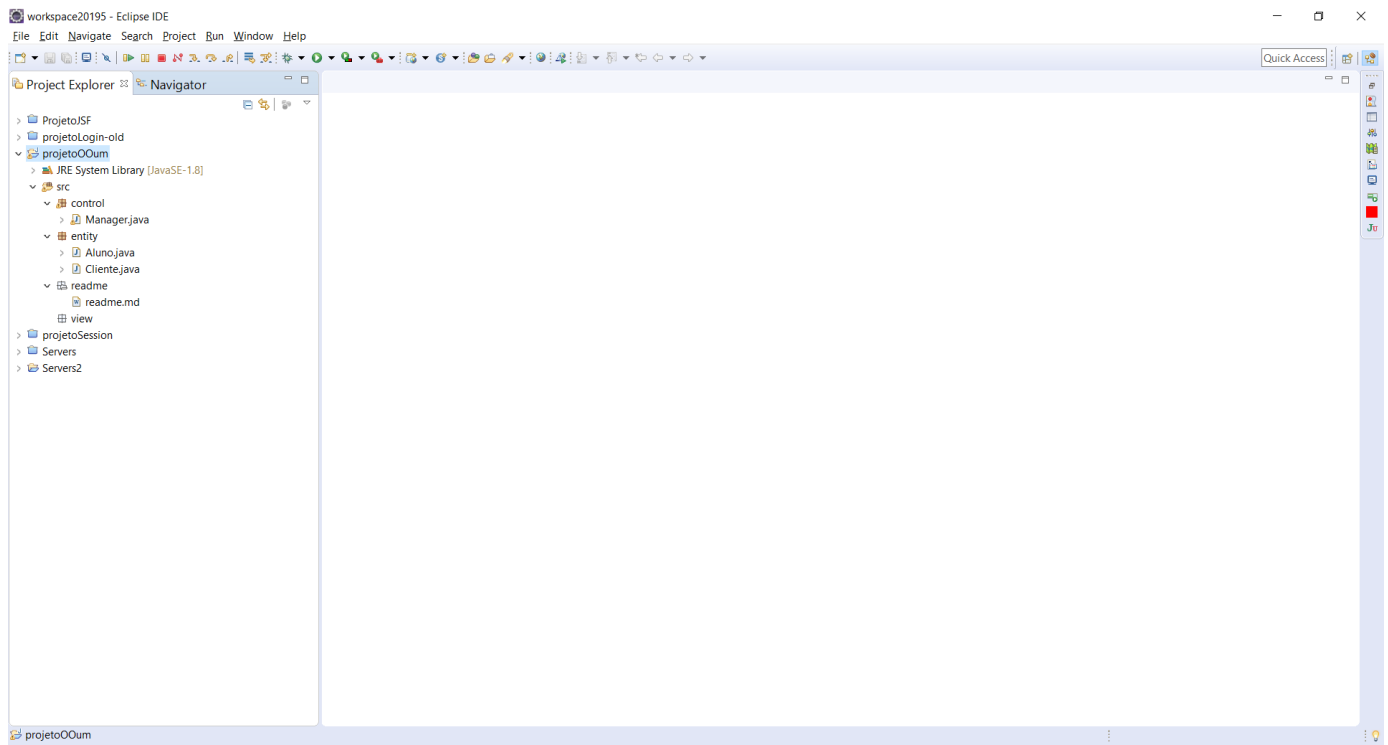
```
1  package entity;
2
3  public class Aluno {
4
5      private Integer idAluno;
6      private String nomeAluno;
7      private Double nota1;
8      private Double nota2;
9
10     public Aluno() {
11     }
12
13     public Aluno(Integer idAluno, String nomeAluno,
14 Double nota1, Double nota2) {
15         super();
16         this.idAluno = idAluno;
17         this.nomeAluno = nomeAluno;
18         this.nota1 = nota1;
19         this.nota2 = nota2;
20     }
21
22     @Override
23     public String toString() {
24         return "Aluno [idAluno=" + idAluno + ", nomeAluno=" +
25 nomeAluno + ", nota1=" + nota1 + ", nota2=" + nota2 + "]\n";
26     }
27
28     public Integer getIdAluno() {
29         return idAluno;
30     }
31     public void setIdAluno(Integer idAluno) {
32         this.idAluno = idAluno;
33     }
34     public String getNomeAluno() {
35         return nomeAluno;
36     }
37     public void setNomeAluno(String nomeAluno) {
38         this.nomeAluno = nomeAluno;
39     }
40     public Double getNota1() {
41         return nota1;
42     }
43     public void setNota1(Double nota1) {
44         this.nota1 = nota1;
45     }
46     public Double getNota2() {
47         return nota2;
48     }
49     public void setNota2(Double nota2) {
50         this.nota2 = nota2;
51     }
52 }
```

```
1  package main;
2
3  import entity.Aluno;
4
5  public class Main {
6
7      public static void main(String[] args) {
8
9          //0 encapsulamento utilizando set e get
10
11          Aluno a1 = new Aluno();
12          a1.setIdAluno(1);
13          a1.setNomeAluno("luiz");
14          a1.setNota1(7.);
15          a1.setNota2(8.);
16
17          System.out.println
18          ("Nome: " + a1.getNomeAluno() +
19           "Nota1: " + a1.getNota1() +
20           "Nota2: " + a1.getNota2()
21          );
22
23          //-----
24          // 0 encapsulamento utilizando Construtor cheio e
25          // toString ( representado pelo objeto )
26          Aluno a2 = new Aluno(2,"Joao",8.,6.);
27          System.out.println("Dados: " + a2);
28      }
29  }
```

## Estrutura do projeto depois de finalizado:

---





`</>` **readme.md** (<http://readme.md>)

```

1  ## Suas Anotações
2  ## requisitos:
3  ## Java vm (java 8)
4  ## IDE (Eclipse)
5
6  ## MVC (Dados / Controller/ Visão)
7
8      private int numero; //primitivo (c++) Java OO (sempre Objeto)
9
10 # Cliente --> Classe Cliente.java
11     // dados Cliente () ...
12     // JaIntegerva nao é vertical ...
13     // classe no maximo 1000 linhas
14     //Node (Vertical )
15     //Cada classe em seu Local ....
16     // modificador de acesso
17     //tipo
18     //nome ...
19     //java (Wrappers --> Integer, Double, Long, String)
20     //qualificadores : private, , proctected, public
21     //private : é na Classe (somente eu visualizo)
22     //em geral : 90% dos atributos sao fechados ...
23     // Integer ->{Byte 8, Short 16, Integer 32 D, Long 64}
24     // Float ->{Float 32, Double 64 D}
25     // NUmero grande demais --> BigDecimal ... (Tera)
26     //String = StringBuffer iu StringBuilder
27     //Boolean (true ou false)
28     //Date (Data)
29     //nativos ou primimitos (byte, short, int, long, float, double}
30
31 package entity;
32
33 public class Cliente {
34
35     private Integer idCliente;
36     private String  nome;
37     private String  email;
38     private String  plano;
39     private Double  valorPlano;
40     //para cada atributo eu tenho
41     //um método chamado getter e set entrada e saída ...
42     //encapsuladores ...
43     //setName (void) _ acao
44     //getNome() _ busca e um (devolve)...
45     //IDE
46     //alt + s (gerar get e set)
47
48     //function
49     public Integer getIdCliente() {
50         return idCliente;
51     }
52     //procedure
53     public void setIdCliente(Integer idCliente) {
54         this.idCliente = idCliente;

```

```
55     }
56     public String getNome() {
57         return nome;
58     }
59     public void setNome(String nome) {
60         this.nome = nome;
61     }
62     public String getEmail() {
63         return email;
64     }
65     public void setEmail(String email) {
66         this.email = email;
67     }
68     public String getPlano() {
69         return plano;
70     }
71     public void setPlano(String plano) {
72         this.plano = plano;
73     }
74     public Double getValorPlano() {
75         return valorPlano;
76     }
77     public void setValorPlano(Double valorPlano) {
78         this.valorPlano = valorPlano;
79     }
80 }
81
82
83 package entity;
84
85 public class Cliente {
86
87     //atributos
88     private Integer idCliente;
89     private String nome;
90     private String email;
91     private String plano;
92     private Double valorPlano;
93     private Boolean ativo;
94
95     public void gerarValorPlano() {
96         switch(this.plano) {
97             case "PLANOUM": this.valorPlano= 250.;
98                             break;
99             case "PLANODOIS": this.valorPlano= 400.;
100                                break;
101             case "PLANOTRES": this.valorPlano= 600.;
102                                break;
103             default: throw new IllegalArgumentException("Nao existe Plano");
104         }
105     }
106
107     public Integer getIdCliente() {
108         return idCliente;
109     }
```

```
110     public void setIdCliente(Integer idCliente) {
111         this.idCliente = idCliente;
112     }
113     public String getNome() {
114         return nome;
115     }
116     public void setNome(String nome) {
117         this.nome = nome;
118     }
119     public String getEmail() {
120         return email;
121     }
122     public void setEmail(String email) {
123         this.email = email;
124     }
125     public String getPlano() {
126         return plano;
127     }
128     public void setPlano(String plano) {
129         this.plano = plano;
130     }
131     public Double getValorPlano() {
132         return valorPlano;
133     }
134     public void setValorPlano(Double valorPlano) {
135         this.valorPlano = valorPlano;
136     }
137     public Boolean getAtivo() {
138         return ativo;
139     }
140     public void setAtivo(Boolean ativo) {
141         this.ativo = ativo;
142     }
143
144     //tem uma area d execucao (chamado main)
145     //public static void main(String args[])
146
147     //main + ctrl + espaco e enter
148
149     public static void main(String[] args) {
150         //Classe (Representacao)
151         //Objeto (instancia para Objeto(
152         //Cliente c (instancia)
153         //Cliente c = new Cliente();
154         //Objeto c (new espaco de mem) Construtor
155         //new no Construtor
156         //Cliente c =new ESpaco ( Cliente())
157         Cliente c = new Cliente();
158         c.setIdCliente(100);
159         c.setNome("marco");
160         c.setEmail("marco@gmail.com");
161         c.setPlano("PLANOUMs");
162
163         c.gerarValorPlano();
164
```

```
165         //syso (imprimir) syso (ctrl espaco e enter)
166         System.out.println(c.getNome() + ", " + c.getValorPlano());
167     }
168 }
```

</> **Aluno.java**

```
1 package entity;
2
3 public class Aluno {
4
5     private Integer id;
6     private String nome;
7     private String disciplina;
8     private Double nota1;
9     private Double nota2;
10    private Double media = 0.; //calcular (já possui o valor de zero)
11    private Boolean ativo;
12
13    // saída ()
14    // alt + s generate (getter and setter)
15    // Um unico valor de Saida ...
16
17    // Saida de todos
18    // alt + s generate getter and setter
19    public void gerarMedia() {
20        this.media = (this.nota1 + this.nota2) / 2;
21    }
22
23    // escolha de inicializador (construtor)
24
25    public Aluno() {
26    }
27
28    // toString (SAIDA DE TODOS)
29    // CONSTRUTOR ENTRADA DE TODOS
30    public Aluno(Integer id, String nome, String disciplina,
31    Double nota1, Double nota2, Boolean ativo) {
32        super();
33        this.id = id;
34        this.nome = nome;
35        this.disciplina = disciplina;
36        this.nota1 = nota1;
37        this.nota2 = nota2;
38        this.ativo = ativo;
39    }
40
41    @Override
42    public String toString() {
43        return "Aluno [id=" + id + ", nome=" + nome +
44        ", disciplina=" + disciplina + ", nota1=" + nota1 + ", nota2="
45        + nota2 + ", media=" + media + ", ativo=" + ativo + "];"
46    }
47
48    public Integer getId() {
49        return id;
50    }
51
52    public void setId(Integer id) {
53        this.id = id;
54    }
```

```
55
56     public String getNome() {
57         return nome;
58     }
59
60     public void setNome(String nome) {
61         this.nome = nome;
62     }
63
64     public String getDisciplina() {
65         return disciplina;
66     }
67
68     public void setDisciplina(String disciplina) {
69         this.disciplina = disciplina;
70     }
71
72     public Double getNota1() {
73         return nota1;
74     }
75
76     public void setNota1(Double nota1) {
77         this.nota1 = nota1;
78     }
79
80     public Double getNota2() {
81         return nota2;
82     }
83
84     public void setNota2(Double nota2) {
85         this.nota2 = nota2;
86     }
87
88     public Double getMedia() {
89         return media;
90     }
91
92     public void setMedia(Double media) {
93         this.media = media;
94     }
95
96     public Boolean getAtivo() {
97         return ativo;
98     }
99
100    public void setAtivo(Boolean ativo) {
101        this.ativo = ativo;
102    }
103
104    // MAIN + CTRL + ESPACO
105    public static void main(String[] args) {
106        Aluno a = new Aluno();
107        a.setNome("belem");
108        a.setDisciplina("java");
109        a.setNota1(10.);
```

```

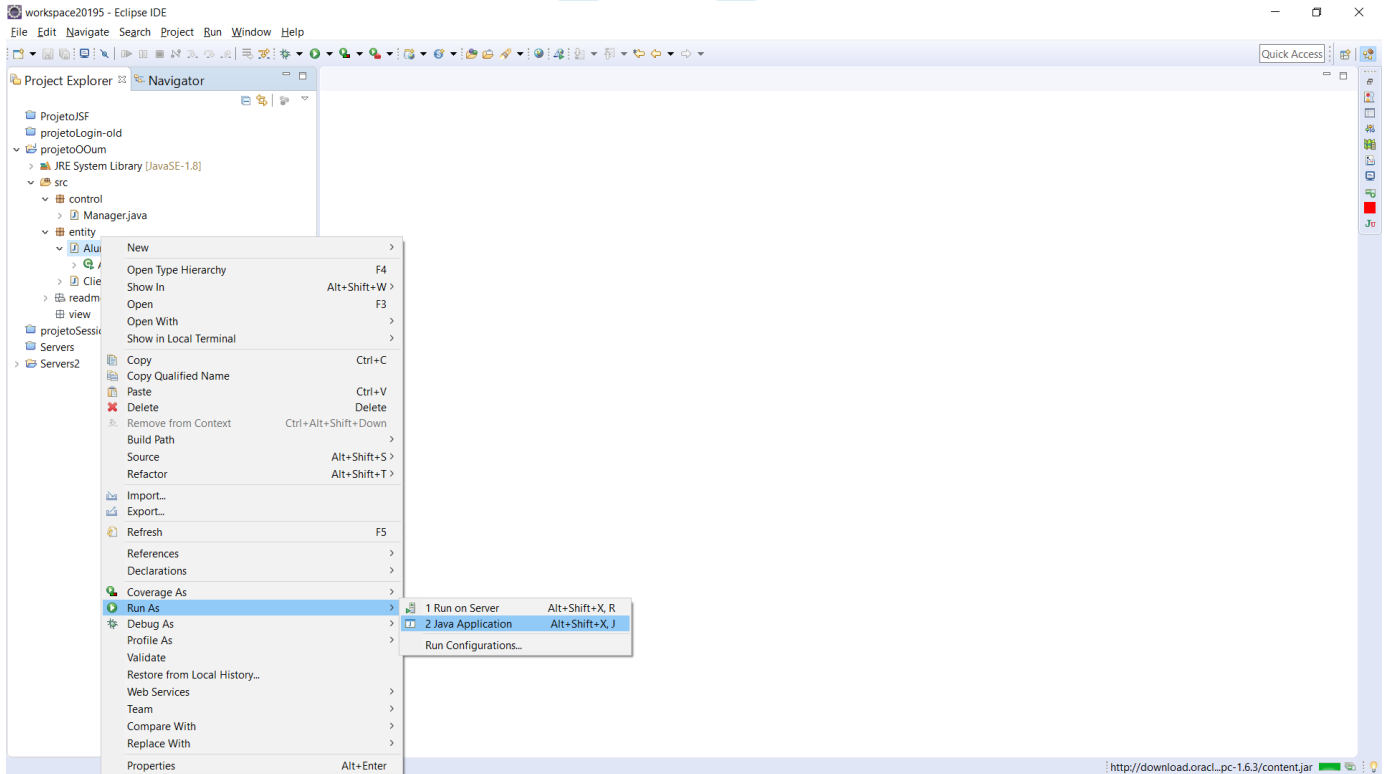
110         a.setNota2(9d);
111         a.gerarMedia();
112         System.out.println("Nome: " + a.getNome() + " - Media: "
113             + a.getMedia());
114         System.out.println(a);
115         // Objeto a = xoxean .... toString ...
116     }
117 }

```

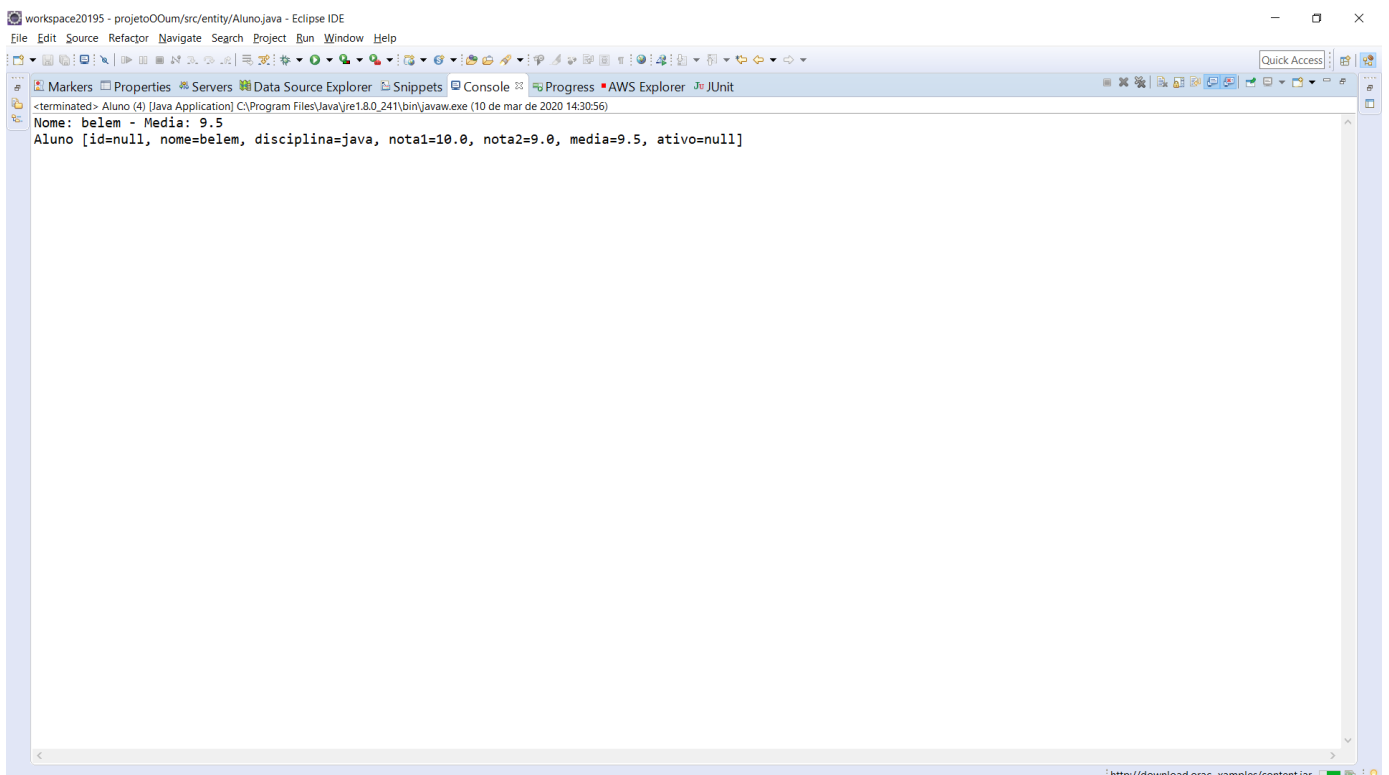


Para rodar a classe:

Clique na classe com o botão direito → run as → java application



👁 Resultado no console:






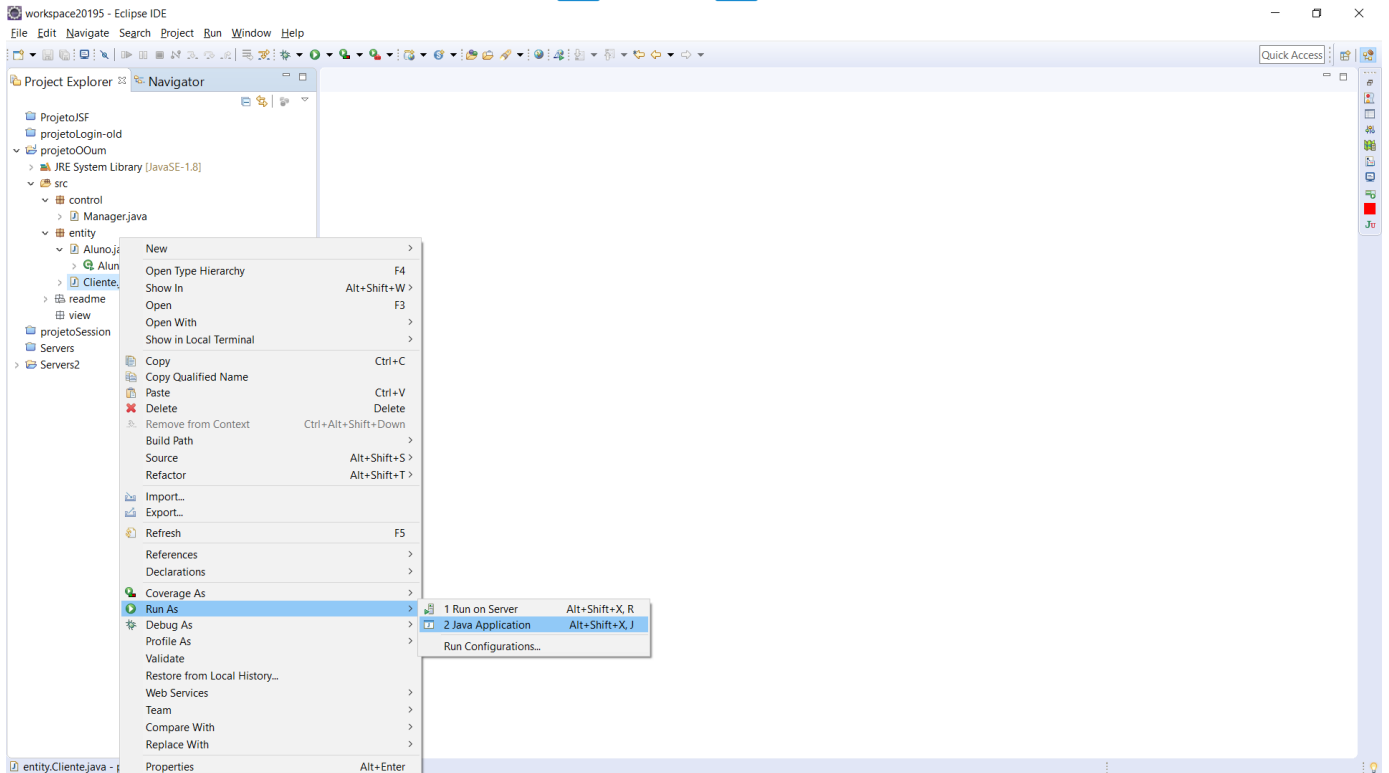
## </> Cliente.java


```
1  package entity;
2
3  public class Cliente {
4
5      // atributos
6      private Integer idCliente;
7      private String nome;
8      private String email;
9      private String plano;
10     private Double valorPlano;
11     private Boolean ativo;
12
13     public void gerarValorPlano() {
14         switch (this.plano) {
15             case "PLANOUM":
16                 this.valorPlano = 250.;
17                 break;
18             case "PLANODOIS":
19                 this.valorPlano = 400.;
20                 break;
21             case "PLANOTRES":
22                 this.valorPlano = 600.;
23                 break;
24             default:
25                 throw new IllegalArgumentException
26                     ("Nao existe Plano");
27             }
28     }
29
30     public Integer getIdCliente() {
31         return idCliente;
32     }
33
34     public void setIdCliente(Integer idCliente) {
35         this.idCliente = idCliente;
36     }
37
38     public String getNome() {
39         return nome;
40     }
41
42     public void setNome(String nome) {
43         this.nome = nome;
44     }
45
46     public String getEmail() {
47         return email;
48     }
49
50     public void setEmail(String email) {
51         this.email = email;
52     }
53
54     public String getPlano() {
```

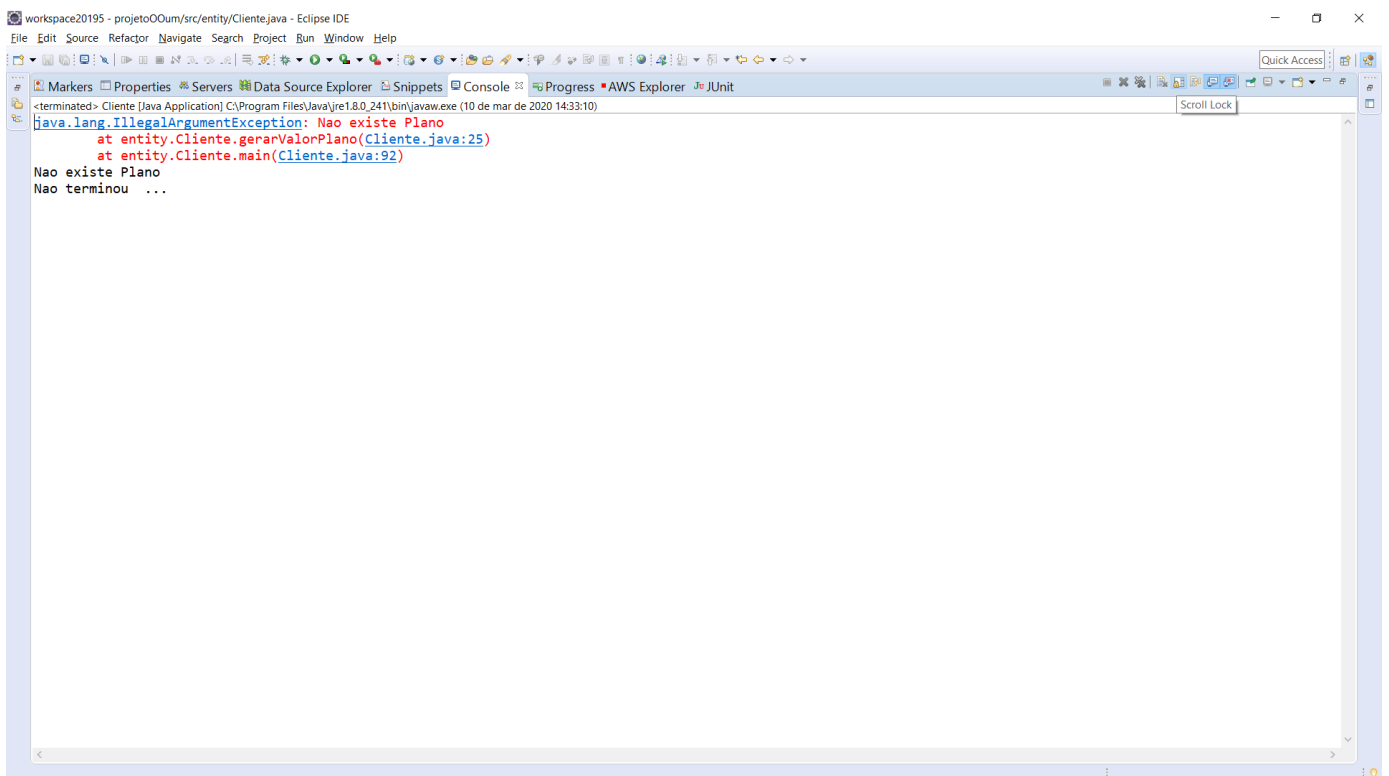
```
55         return plano;
56     }
57
58     public void setPlano(String plano) {
59         this.plano = plano;
60     }
61
62     public Double getValorPlano() {
63         return valorPlano;
64     }
65
66     public void setValorPlano(Double valorPlano) {
67         this.valorPlano = valorPlano;
68     }
69
70     public Boolean getAtivo() {
71         return ativo;
72     }
73
74     public void setAtivo(Boolean ativo) {
75         this.ativo = ativo;
76     }
77
78     // tem uma area d execucao (chamado main)
79     // public static void main(String args[])
80     // main + ctrl + espaco e enter
81
82     public static void main(String[] args) {
83         try {
84             // tentar
85             Cliente c = new Cliente();
86             c.setIdCliente(100);
87             c.setNome("marco");
88             c.setEmail("marco@gmail.com");
89             c.setPlano("PLANOUMs");
90             c.gerarValorPlano();
91             System.out.println(c.getNome() + ", " +
92             c.getValorPlano());
93         } catch (Exception e) {
94             // desvio ...
95             e.printStackTrace(); // detalhe do erro (Vermelho)
96             // e.getMessage() //imprimindo o erro
97             System.out.println(e.getMessage());
98         // Nao existe o plano
99             // esta trazendo a message ...
100             // syso (ctrl + espaco)
101         }
102         System.out.println("Nao terminou ...");
103     }
104 }
```

 Para rodar a classe:

Clique na classe com o botão direito  run as  java application



 Resultado no console:



**</> Manager.java**

```
1 package control;  
2  
3 import java.util.ArrayList;  
4 import java.util.List;  
5  
6 import entity.Aluno;  
7  
8 //Classe Diretoria ... (Gerente) Classe de Controler  
9 public class Manager {  
10     private Aluno aluno; // acostumar com a ideia  
11     private List<Aluno> alunos; // dinamico .... (um milhao) ...  
12  
13     // espaco de mem no construtor  
14     // inicializando no construtor ...  
15     public Manager() {  
16         this.aluno = new Aluno();  
17         // espaco de mem (para um)  
18         this.alunos = new ArrayList<Aluno>();  
19         // espaco de mem (para varios)  
20     }  
21  
22     public void gerarUnidade() {  
23         this.aluno = new Aluno(10, "lu", "java", 7., 10., true);  
24     }  
25  
26     public void gerarGrupo() {  
27         // add ==> para lista  
28         this.alunos.add(new Aluno(100, "mercador", "java", 5., 3., true));  
29         this.alunos.add(new Aluno(101, "filipe", "angular", 2., 1., true));  
30     }  
31  
32     public Aluno getAluno() {  
33         return aluno;  
34     }  
35  
36     public void setAluno(Aluno aluno) {  
37         this.aluno = aluno;  
38     }  
39  
40     public List<Aluno> getAlunos() {  
41         return alunos;  
42     }  
43  
44     public void setAlunos(List<Aluno> alunos) {  
45         this.alunos = alunos;  
46     }  
47  
48     // main + CTRL + ESPACO  
49     public static void main(String[] args) {  
50         // chamar ...  
51         Manager m = new Manager();  
52         m.gerarUnidade();  
53         m.gerarGrupo();  
54     }
```

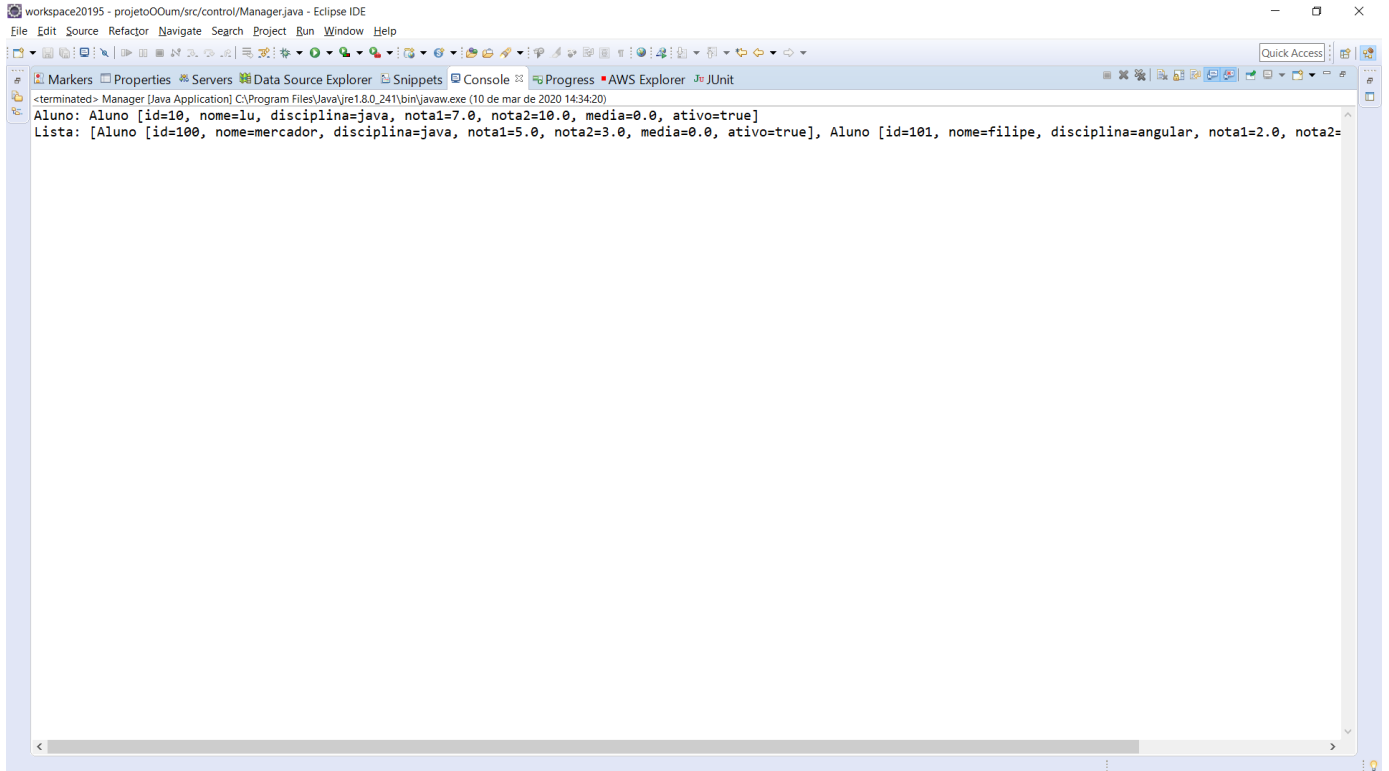
```
55 // syso + CTRL + ESPACO
56 System.out.println("Aluno: " + m.getAluno());
57 System.out.println("Lista: " + m.getAlunos());
58 }
59 }
```



Para rodar a classe:

Clique na classe com o botão direito → run as → java application

### 👁 Resultado no console:



— Coti Informática <https://www.cotiinformatica.com.br> (<https://www.cotiinformatica.com.br>)