**Curso: Programação Cliente Servidor**

**Aula 1 – Orientação a Objeto - OO**

1. Revisão OO (Orientação a Objeto)

É um paradigma de análise, projeto e programação de software baseada em composição e interação entre unidades de software chamadas Objetos. É uma forma de ver um problema, modelar uma solução e com isso desenvolver um sistema.

1. Abstração

É a capacidade de do ser humano concentrar-se apenas nos aspectos essenciais de um problema. Ignorando algumas outras que não são importantes.

No contexto da orientação a objetos, uma classe é uma abstração de conceitos do mundo real, sem que precisamos fazer uso de todas as características do conteúdo real, apenas usamos como modelo, as características que são fundamentais para o nosso sistema.

1. Classe

É uma representação genérica de um conjunto de indivíduo idênticos.

Teoricamente, uma classe é um modelo para a criação de **objetos**, então todos os objetos serão criados baseados a partir dessa classe, que possuirão os mesmos comportamentos(**classes**) e armazenarão as informações (**atributos**).

1. Atributos

Digamos que é quando se refere ao objeto, como características. Por exemplo, o **Aluno** Pedro cursa Análise e Desenvolvimento de Sistemas na Universidade Sr. Madruga e tem 20 anos de idade.

Nesse casso, a classe poderia se chamar **ALUNO**, que poderia ser formada pelos atributos **NOME DO ALUNO**, **CURSO**, **UNIVERSIDADE** e **IDADE**.

Poderíamos instanciar a classe aluno, fazendo uso de seus atributos, quantas vezes precisarmos para cadastrarmos novos aluno.

Exemplo:

|  |
| --- |
| **ALUNO** |
| **NOME**: Pedro **CURSO**: Análise e Desenvolvimento de Sistemas **IDADE**: 20 **Universidade**: Sr. Madruga |

Para os **atributos**, atribuímos **valores**, os atributos são abstratos, ou seja, a característica que abstraímos para formar um **objeto** **ALUNO**, já os valores podemos identifica-los (Pedro, Análise e Des. De Sistamas, 20, Sr. Madruga).

Obs.: O estado dos atributos, podem variar.

1. Métodos

São as **ações** que um Objeto pode fazer.

Por exemplo, a classe ALUNO pode fazer:

Estudar  
Ler  
Acordar  
Andar

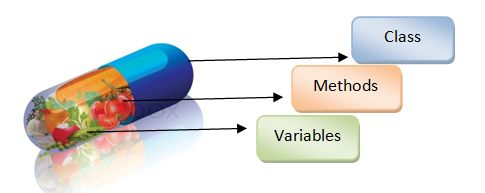
Podemos atribuir PARÂMETROS aos métodos, que são as informações que ele precisa para realizar uma ação e retornar valores, que é o produto da ação realizada.

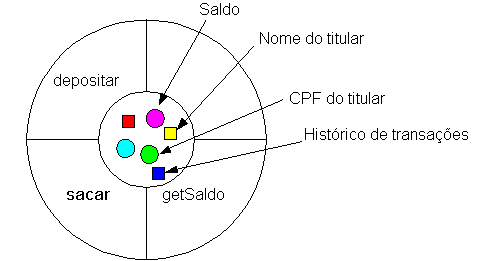
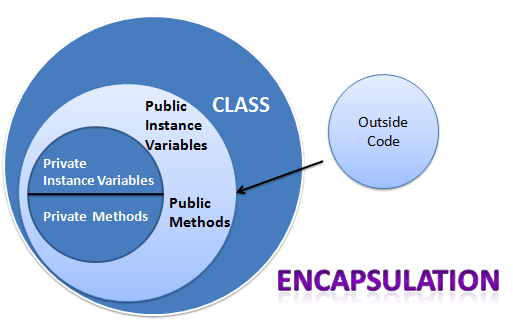
1. Encapsulamento

Os **dados ficam encapsulados** **no interior dos métodos** (ações) de um **objeto**, desta forma não saberemos o que há dentro dos métodos quando o estivermos manipulando. Impede acesso direto ao **Objeto.**

Exemplo:

Classe: TV  
Atributos: Cor, Polegadas, Modelo, Marca  
Métodos: Liga, Desliga, AumentaVolume, DiminuiVolume  
  
Cada método, possui a instrução que não poderá ser acessada pelo usuário enquanto estiver manipulando, mas podemos ver o que tem nos atributos como a cor da TV, a Polegada, o Modelo e a Marca, e podem ter seus valores alterados durante a manipulação.





1. Sobrecarga

Técnica que permite usar **vários métodos** que **tenham o mesmo identificador, variando somente a sua assinatura**.

1. Herança

Recurso em OO que permite que ***classes compartilhem atributos e métodos***, através de herança. (**SuperClasse** -> **SubClasse**

A classe **Felino**, poderia estender uma classe **Animal** e consequentemente a classe animal herdaria seus atributos e estados.

Exemplo 1:

Um Cross Fox, nada mais é que um Fox com mais recursos. Mas ambos possuem os mesmos atributos comuns entre si. (4 ou 2 portas, 5 pneus, Vidros, Teto, Motor, Etc). Podemos dizer que um Cross Fox(classe-filha) é Filho de um Fox(super-classe ou classe-pai).

Exemplo 2:

CLASSE-PAI: **FELINO**ATRIBUTOS**: TIPO(char), IDADE(inteiro), NOME(char), TAMANHO(char), ORIGEM (char)**MÉTODOS: **MIA**, **ANDA**, **CAÇA**

CLASSE-FILHA: **ANINAL** (INSTANCIA A CLASSE-PAI FELINO)  
HERDA ATRIBUTO E RECEBE VALORES: **TIPO=ONÇA, IDADE=8, NOME=MINGAU, TAMANHO=4M, ORIGEM=ÁFRICA**HERDA AÇÕES: **MIA, ANDA, CAÇA**

A conclusão seria que o **Animal** que **Mia**, **Anda** e **Caça**, é um **Felino**.

Conceito Visual:

ANIMAL

HERBIVORO

CARNÍVORO

ONIVORO

Coelho

Leão

Hiena

Homem

Class A

Class B

Class C

Podemos observar que no conjunto a cima, a **Class A** não herda nada, caracterizando-se como a Super-Classe ou Classe-Pai, a **Class B** herda todos os atributos e métodos da **Class A**, e a **Class C** herda todos os atributos e métodos das **Class A** e **Class B**, caracterizando-se com classes-filha,

No fluxograma ao lado, usando a mesma teoria, conseguimos ver que A *classe-pai* é **Animal** e as demais como Classe-Filha.

A classe mais baixa, herda as classes mais acima, por exemplo. As classes inferiores são todas Animais, pois herdam da classe-pai, e alguns herdam atributos e métodos da classe-filha acima que classifica alguns animais como características das classes herdadas do segundo nível.

1. Sobrescrita

Acontece quando precisamos **reescrever um método herdado pela classe-pai**. Nesse caso, o novo *método deve ter a mesma assinatura da classe pai*.

- Possuir o mesmo nome: **FELINO**  
- Retornar o mesmo tipo de dados: **Nome Atributo**(**char**), **Nome Atributo**(**inteiro**), **Nome Atributo**(**char**), **Nome Atributo**(**char**), **Nome Atributo**(**char**).  
- Recebe a mesma lista de parâmetros: **MÉTODOS(?)**.

1. Polimorfismo

Seria uma forma de **tratar as classes e sub-classes**, ***através de sua super-classe***.

Exemplo: Uma turma, onde o aluno é o 5º da segunda fileira. Para falar com esse aluno, a forma direta seria chamando-o pelo seu nome. Mas para proceder dessa forma, teríamos que saber o nome de cada aluno, ou seja, tratar nominalmente. Então a forma como se refere ao aluno, mesmo não sabendo seu nome é que faz a diferença (apontado para o aluno), mesmo aluno sendo um termo genérico.

Então, Aluno caberia para todos, mas o aluno em específico saberia que se trata dele pela forma como nos referimos a ele.

Mas a forma de fazer a chamada, serviria para todos os alunos. Daí o termo **Poli = Diversa + Morfismo = Forma.**

**Conceito**: Então, Polimorfismo permite que **referências de tipos** de *classes mais abstratas* *representem o comportamento das classes concretas que referenciam*.

É a capacidade de **Objetos instanciados de classes diferentes** usar um **mesmo método**, mas cada qual da sua maneira.

A superclasse define o comportamento esperado (interface pública) de todas as subclasses.

Programa na interface pública.

|  |
| --- |
| **Forma** |
| -cor: String |
| **+getArea():double** +toString():String |

Superclasse fornece as implementações reais

|  |  |  |
| --- | --- | --- |
| **Retangulo** |  | **Triangulo** |
| -altura:int -largura:int | -base:int -largura:int |
| **+getArea():double** +toString():String | **+getArea():double** +toString():String |

1. Classes e Métodos Abstratos

* **Classe Abstrata**: É toda classe que **não possui a capacidade de gerar objetos**, servindo apenas como **modelo para uma classe concreta**.
* **Métodos Abstratos**: Devem ser **implementados na classe concreta** que **estenda a classe na qual foi declarado na qual pode ter estendida**. Um método abstrato, somente podem ser definidos em classes abstratas.
* **Representa entidades e conceitos abstratos**
* **Não possui instancias**.
* **Template para classes derivadas**.
* **Normalmente possui métodos abstratos**.

É uma classe que não será objeto, ou seja, é uma classe que foi feita pra ser estendida.

Exemplo:

- Massa de uma Pizza  
- O que compramos é a pizza. A Massa serve de base para a pizza.

|  |
| --- |
| ***Animal*** |
| +habitoAlimentar: Setring = “”  +reproducao: String = “”  +pesoMedio: Double = 0.0 |
| +detelheAnimal(): void |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Peixe** |  | **Cavalo** |  | **Aguia** |
|  |  | - raca: String = “” - corPelagem: String = “” |  |  |
|  |  |  |  |  |

Peixe, Cavalo e Águia não são instancias de Animal, elas apenas herdam os atributos e métodos da classe animal que apenas serve como base e não pode ser instanciada como objeto.

1. **Classe Final**

É uma classe que não pode ser herdada.

1. **Método Final**

Seguindo a mesma lógica, o método final é um *método concreto que não podem ser sobrescritos*.

1. **Atributos Finais**

São atributos que se comportarão como constantes, ou seja, não poderão ter seus valores alterados.

1. **Interfaces**

São classes puramente abstratas, quando possuem atributos estes devem ser declarados como finais.

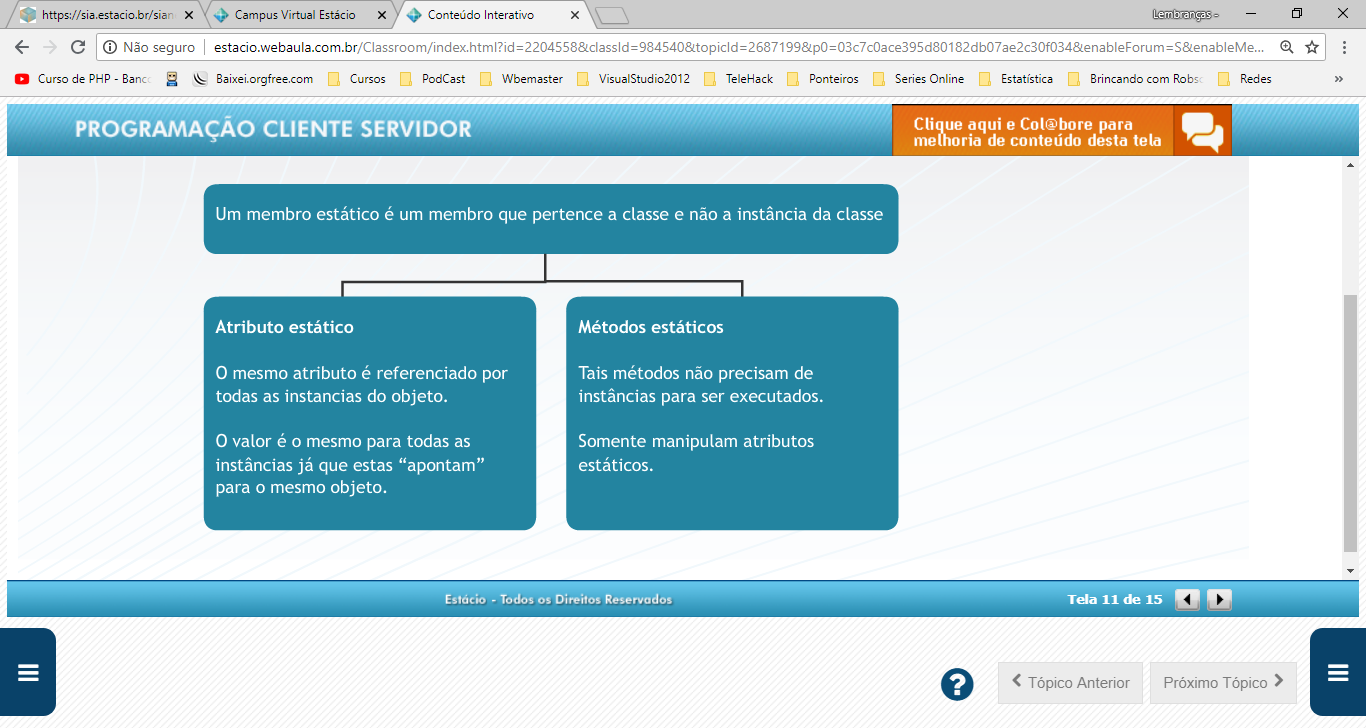
São utilizadas para ***definir padrões para especificação do comportamento de classes***.

1. Visibilidade de Membros de Classes (atributos)

* **Pública**: Podem ser *acessadas por qualquer objeto*.
* **Privada**: Podem ser *acessadas somente dentro da classe onde foi declarada*.
* **Protegida**: Podem ser *acessadas por qualquer objeto dentro do pacote no qual foi declarada* ou *fora desse pacote pelas classes que estendam a essa classe*.
* **Padrão**(**pacote**): Podem ser *acessadas somente por objetos que estejam no mesmo pacote em que a classe foi declarada*.

1. Membros de classe estáticos

Além dos *modificadores de visibilidade*, os *membros de uma classe* podem ainda ser *definidos como estáticos*.



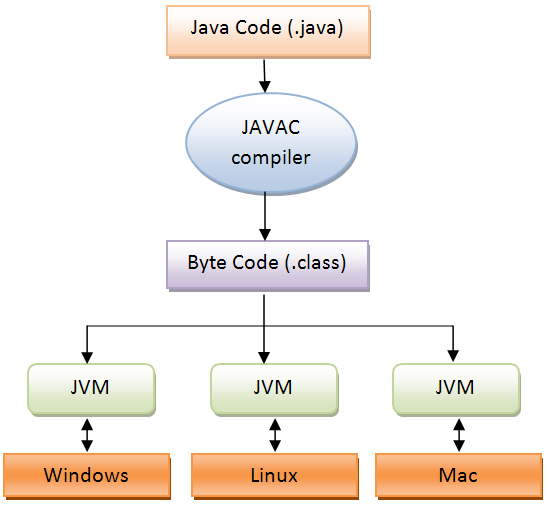
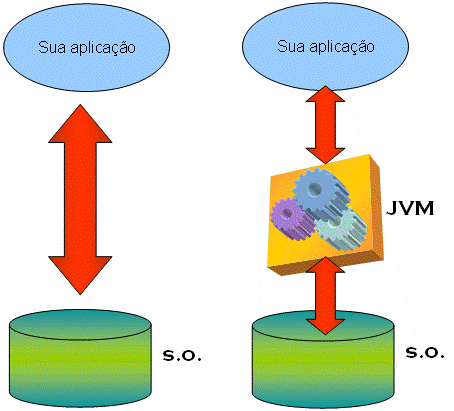
1. A Linguagem JAVA

“Write once, run anywhere” = Escreva uma vez e rode em qualquer lugar.

A ideia é escrever um código único e não precisar ter seus códigos alterados pelo simples fato de mudarmos de plataforma (sistema operacional).

1. **JVM = Java Virtual Machine**

É um programa que carrega e executa os aplicativos Java, convertendo os **bytecodes** em código **executável de máquina**.

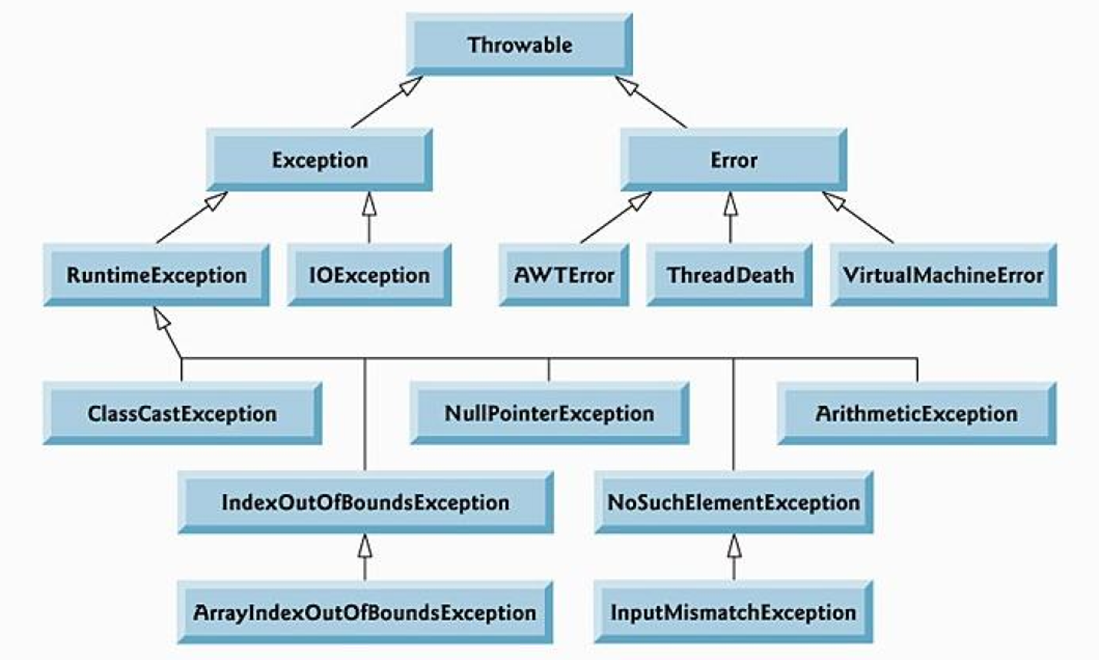


Qualquer sistema JAVA, roda em qualquer **S**istema **O**peracional com JVM específico.

1. Exceptions em JAVA

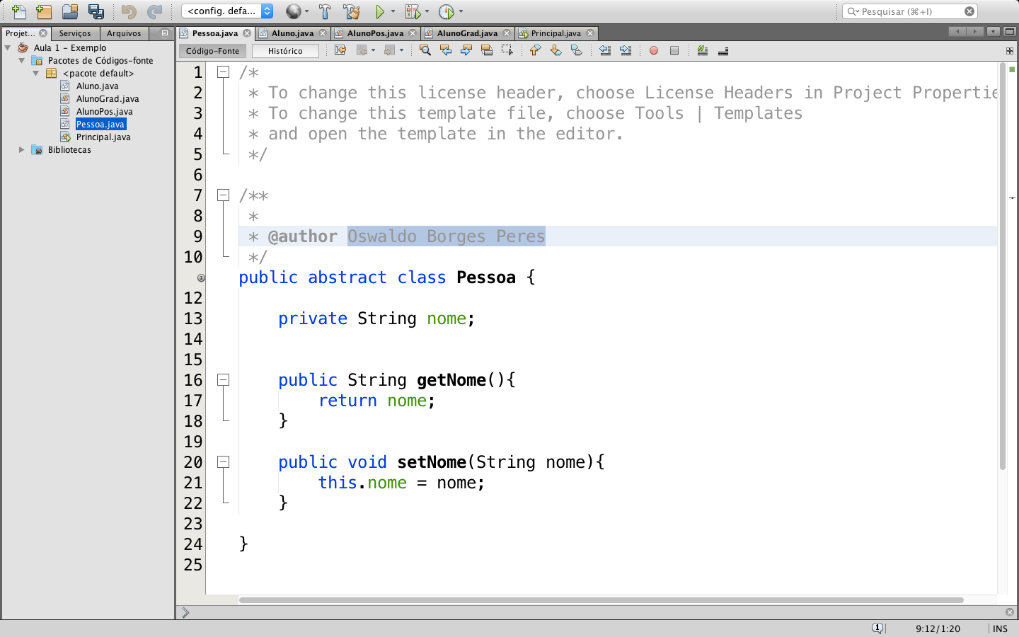
Existem dois tipos:

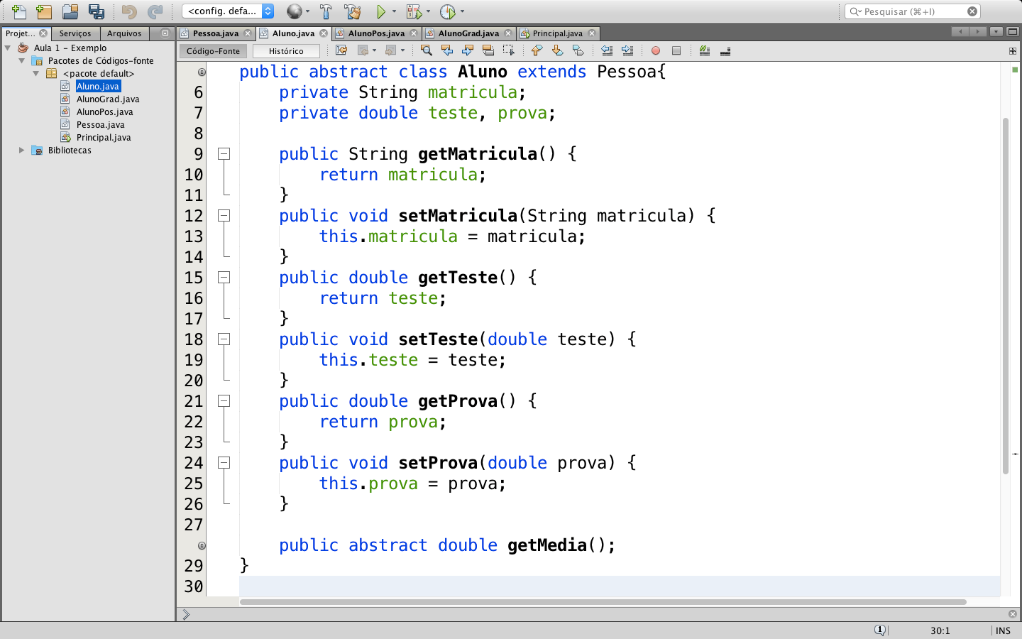
* Verificadas
* Não verificadas

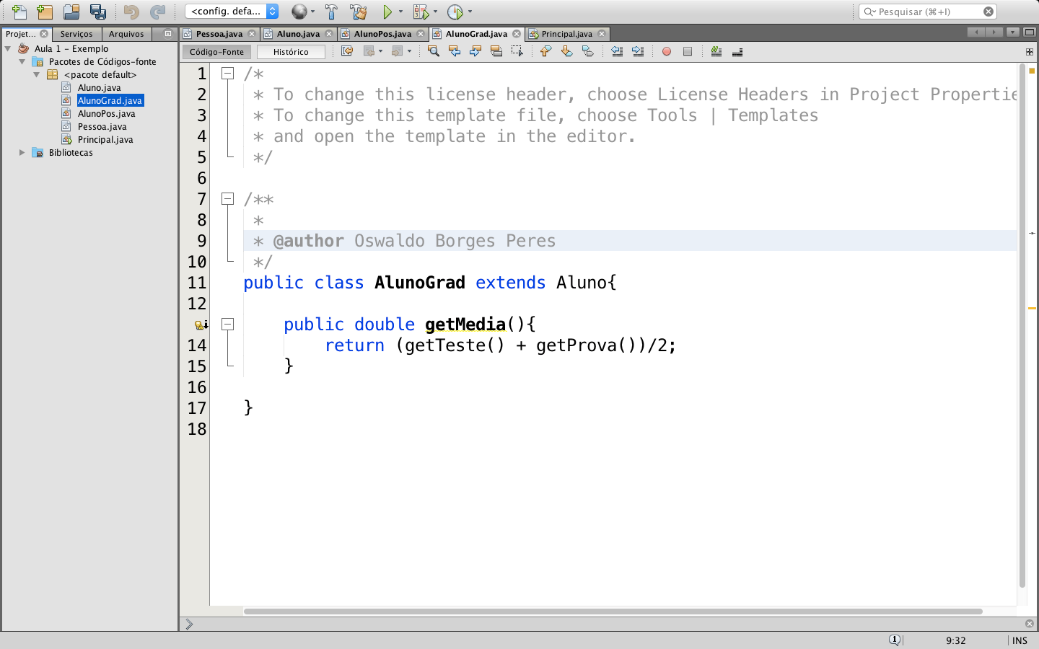


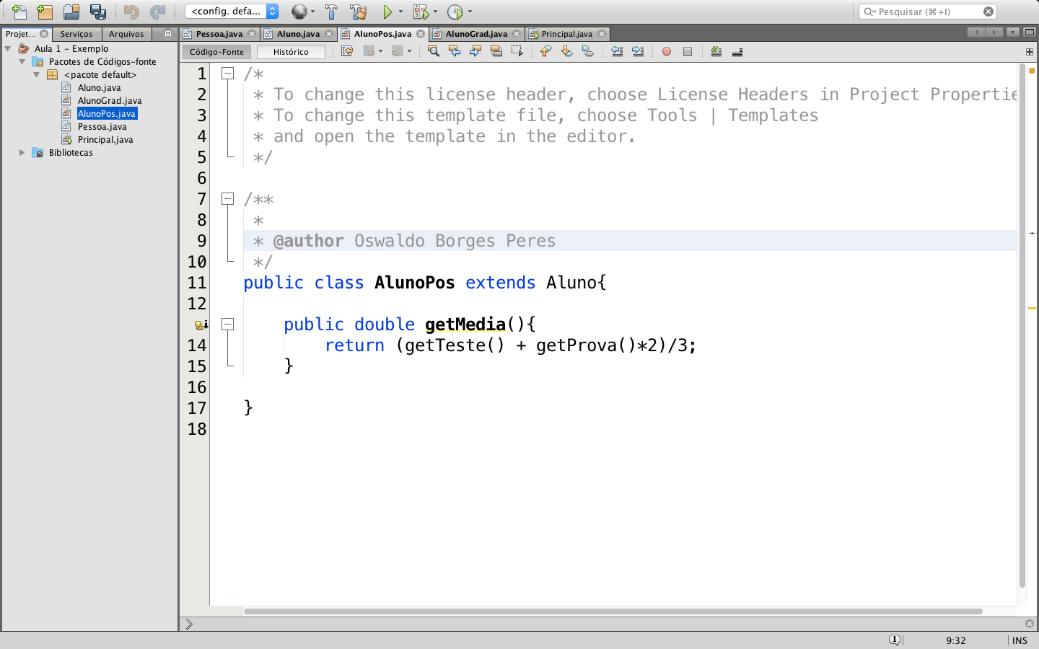
Obs.: Exemplos de Exception: Divisão com 0, posso controlar com IF, mas um erro de acesso a banco não permite fazer tratamento (Não verificadas) outros, podemos fazer via código.

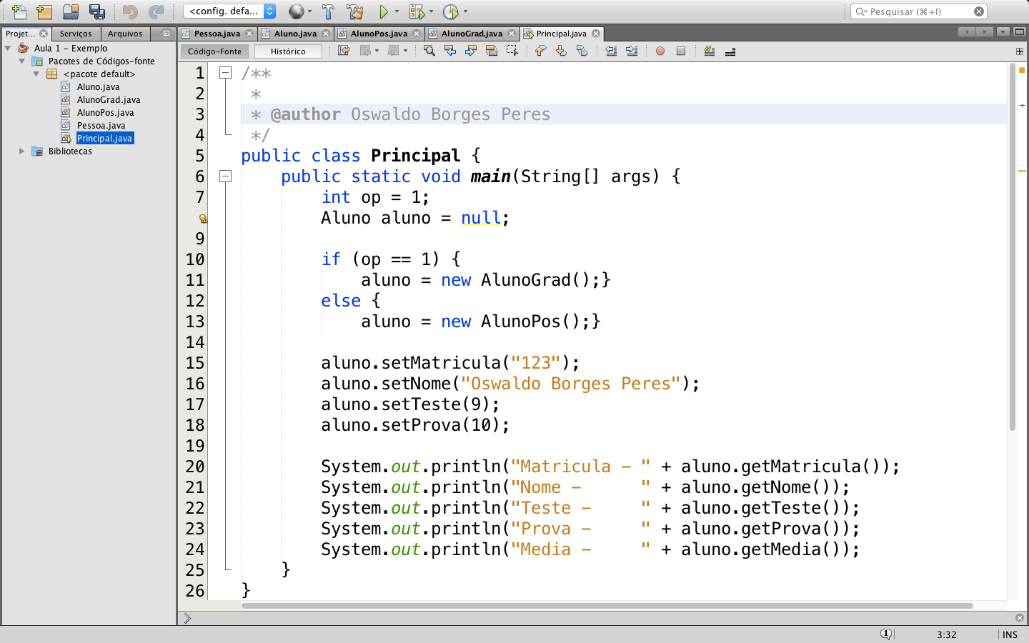
Exemplo:



1









**Bibliografia:**

Java: como programar - 8ª edição

Deitel, Paul J.; Deitel, Harvey M.

**Aula 2 –**