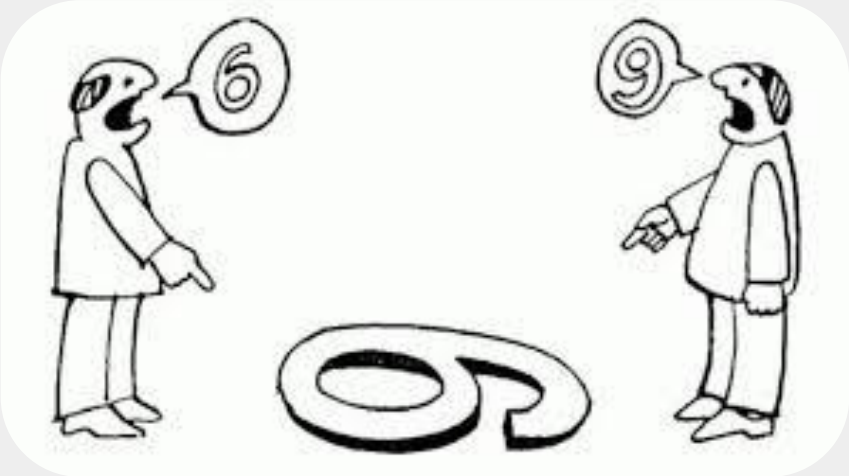
Programação Orientada a Objetos

Aula: Introdução a Classes e Objetos.

Prof. Anderson Augusto Bosing

O que é um Paradigma?



O que é um Paradigma?

Um exemplo que serve como modelo ou padrão.

O que são Paradigmas da Programação?

O que são Paradigmas da Programação?

Um paradigma é um estilo de programação, um modelo, uma metodologia.
Não se trata de uma linguagem, mas a forma como você soluciona problemas usando uma determinada linguagem de programação.

Paradigmas da Programação
Procedural

Consiste em um modelo de programação que funciona como uma espécie de lista de instruções que são executadas em forma de passo a passo.

Ex:

- C
- Pascal

Paradigmas da Programação
POO

Este paradigma é o que mais reflete os problemas atuais. Um programa OO consistem em objetos que enviam mensagens uns para os outros. Estes objetos no programa correspondem diretamente a objetos atuais, tais como pessoas, máquinas, departamentos, documentos e assim por diante.

Ex:

Java

C#

POO

Mundo Real

Projeto POO

Casa

Carro

Cachorro

Introdução à Tecnologia de Objetos

Hoje, como a demanda por software novo e mais poderoso está aumentando, construir softwares de maneira rápida, correta e econômica continua a ser um objetivo indefinido.

Objetos ou, mais precisamente, as classes de onde os objetos são essencialmente componentes reutilizáveis de software.
Há objetos data, objetos data/hora, objetos áudio, objetos vídeo, objetos automóvel, objetos pessoas etc.

Quase qualquer substantivo pode ser razoavelmente representado como um objeto de software em termos dos atributos (por exemplo, nome, cor e tamanho) e comportamentos (por exemplo, calcular, mover e comunicar).

Introdução à Tecnologia de Objetos

Quando programamos estamos modelando aspectos do ‘mundo real’ utilizando uma linguagem de programação. Assim sempre temos um aspecto do ‘mundo real’, a representação deste aspecto no ‘mundo computacional' (em tempo de execução) e a descrição deste aspecto no ‘mundo linguístico' (em uma linguagem de programação).

O automóvel como um objeto

Vamos supor que você queira dirigir um carro e fazê-lo andar mais rápido pisando no pedal acelerador. O que deve acontecer antes que você possa fazer isso?

O automóvel como um objeto

Vamos supor que você queira dirigir um carro e fazê-lo andar mais rápido pisando no pedal acelerador. O que deve acontecer antes que você possa fazer isso?

Alguém precisa projetá-lo.

Criar uma especificação que vai servir como base para fabricação dos carros que vão ser utilizados nas ruas.

Considerando um software para um banco.

O que toda conta corrente tem que é importante para nós?

Considerando um software para um banco.

O que toda conta corrente tem que é importante para nós?

- Número da conta
- Nome do titular da conta
- Saldo
- Limite

Considerando um software para um banco.

O que toda conta corrente faz que é importante para nós?

O que pedimos à conta corrente?

Considerando um software para um banco.

O que toda conta corrente faz que é importante para nós?

O que pedimos à conta corrente?

- **Saca uma quantidade x;**
- **Deposita uma quantidade x;**
- **Consultar o saldo atual;**
- **Imprimir extrato.**
- **Transfere uma quantidade x para uma outra conta y;**

Considerando um software para um banco.

De acordo com o que levantamos então temos uma especificação de uma conta. Sendo ela:

Atributos de Uma conta

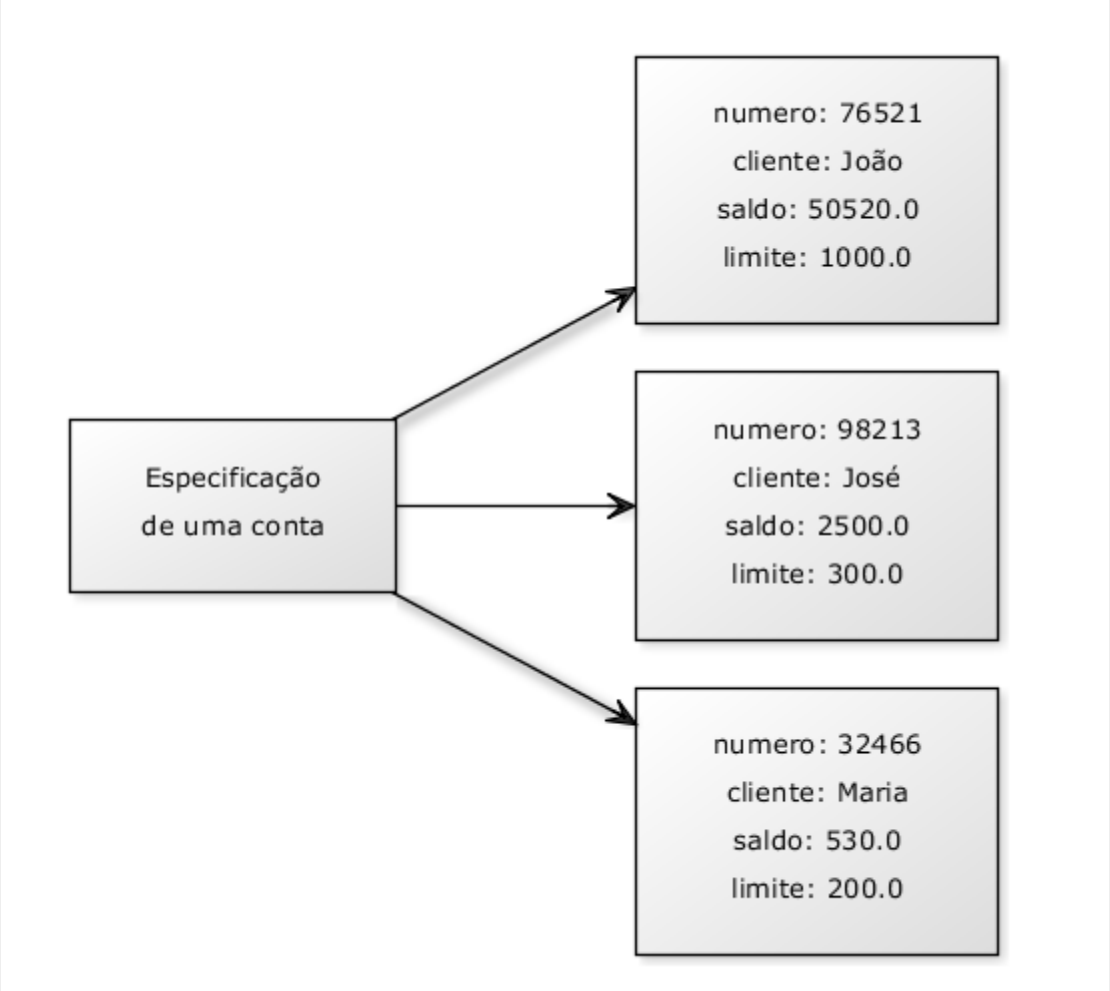
- **Número da conta**
- **Nome do titular da conta**
- **Saldo**
- **Limite**

Métodos de uma Conta

- **Saca uma quantidade x;**
- **Deposita uma quantidade x;**
- **Consultar o saldo atual;**
- **Imprimir extrato.**
- **Transfere uma quantidade x para uma outra conta y;**

Mas o que temos ainda é o projeto dela, antes de a utilizarmos precisamos construir uma conta.

Considerando um software para um banco.



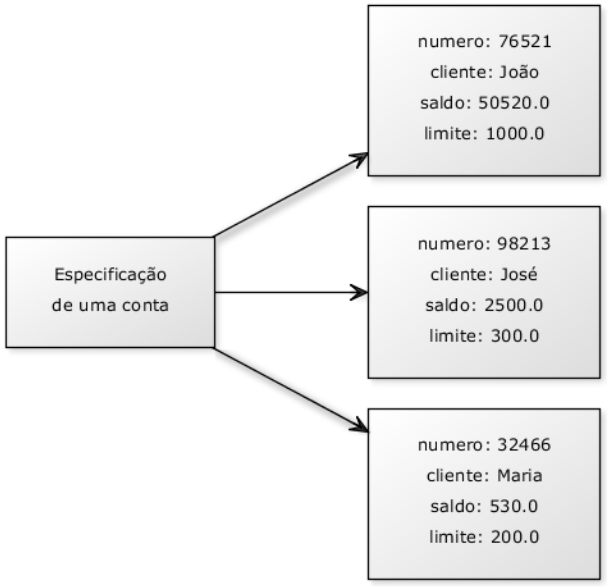
Considerando um software para um banco.

Na figura anterior podemos observar que ao lado esquerdo temos a especificação de uma conta, mas essa especificação é uma conta ?

Nós depositamos e sacamos dinheiro desse papel?

Não. Utilizamos a especificação da Conta para poder criar instâncias que realmente são contas, nas quais podemos realizar as operações que criamos.

Apesar de declararmos que toda conta tem um saldo, um número e uma agência no pedaço de, são nas instâncias desse projeto em que realmente há espaço para armazenar esses valores.

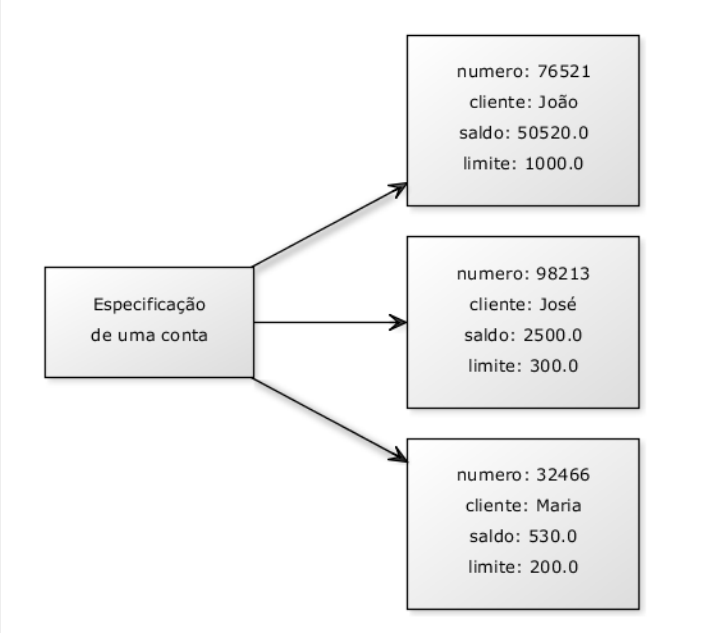


Considerando um software para um banco.

Ao projeto da conta, isto é, à especificação da conta, damos o nome de classe. Ao que podemos construir a partir desse projeto que são as contas de verdade, damos o nome de objetos.

As características da conta damos o nome de atributos(Numero, Saldo, Cliente).

E aos comportamentos desta conta damos o nome de métodos(Sacar, depositar, imprimir extrato).



Um outro exemplo: uma receita de bolo

A pergunta é certeira: você come uma receita de bolo? Não.

Precisamos instanciá-la e fazer um objeto bolo a partir dessa especificação (a classe) para utilizá-la.

Podemos criar centenas de bolos com base nessa classe (a receita, no caso). Eles podem ser bem semelhantes, alguns até idênticos, mas são objetos diferentes.

Um outro exemplo: planta de uma casa

A planta de uma casa é uma casa? Definitivamente, não.

Não podemos morar dentro da planta de uma casa nem podemos abrir sua porta ou pintar suas paredes.

Precisamos, antes, construir instâncias a partir dessa planta. Essas instâncias, sim, podemos pintar, decorar ou morar dentro.

Definições Técnicas

Classe

Assim como os objetos do mundo real, uma classe agrupa os objetos pelos seus comportamentos e atributos comuns.

Uma classe define os atributos e comportamentos comuns compartilhados por uma tipo de objeto. Os objetos de certo tipo ou classificação compartilham os mesmos comportamentos e atributos.

Definições Técnicas

Objeto

Um objeto é uma instância de uma classe.

Atributo

Atributos são as características de uma classe visíveis externamente. A cor de um carro e o seu modelo são exemplos de atributos.

Método

Métodos definem as habilidades dos objetos

Vamos codificar.

- Criar uma classe câmera.

- Criar uma classe gato.

NetBeans

Vamos Praticar?

Lista 1 - Exercicios Orientação a Objetos com Java.

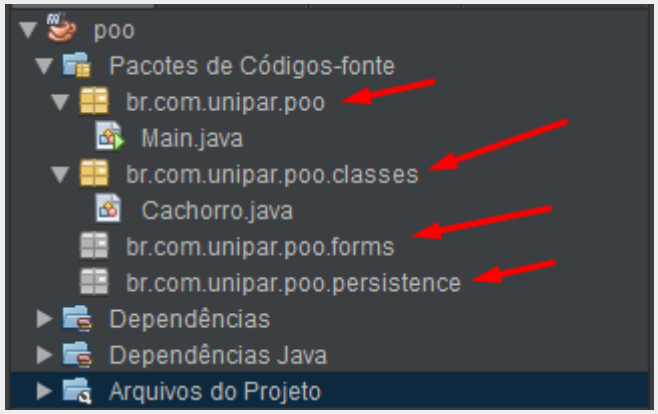
Programação Orientada a Objetos

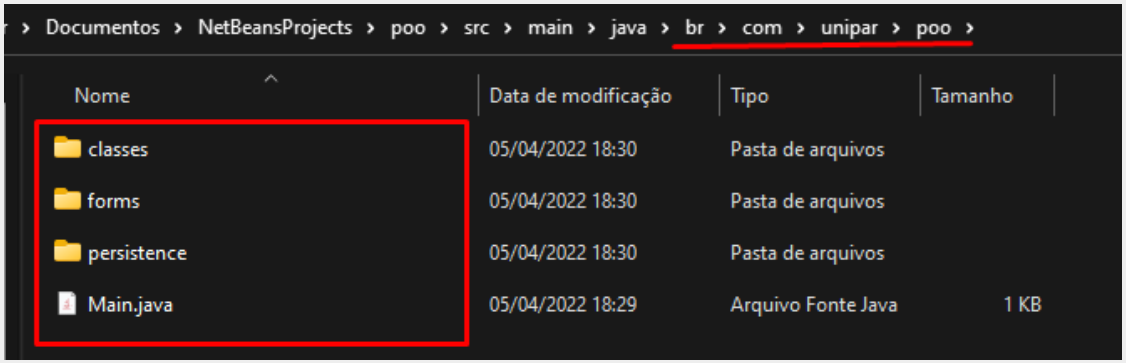
Aula: Modificadores de acesso,
Métodos Acessores.

Prof. Anderson Augusto Bosing

Pacotes

Pacotes java são utilizados para organizar as classes da sua aplicação. Um programa pode, facilmente, ter mais de centenas de classes. Então é muito importante que todos os seus componentes fiquem organizados. Podemos pensar nos pacotes como uma pasta do seu sistema de arquivos.





Modificadores de Acesso a Nível de Classe - public

Modificador public torna uma classe visível:

Para qualquer outra classe e em qualquer pacote.

```
 */
public class Cachorro {

}
```

Modificadores de Acesso a Nível de Classe - default

Modificador vazio ou default torna uma classe visível:

Torna uma classe visível apenas para classes do mesmo pacote.

```
 */
class Cachorro {

}
```

Modificadores de Acesso a Nível de Atributos e Métodos(Membros) - public

public torna um membro acessível:

Em qualquer lugar e a qualquer outra classe que possa visualizar a classe que contém o membro.

```
public class Cachorro {

    public String nome;

}
```

```
public class Main {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {

        Cachorro dog = new Cachorro();
        dog.nome = "bob";


    }

}
```

Modificadores de Acesso a Nível de Atributos e Métodos(Membros) - protected

protected torna um membro acessível às classes:

Do mesmo pacote.

Os membros herdados não são acessíveis a outras classes fora do pacote em que foram declarados.

```
*/
public class Cachorro {

    protected String nome;

}
```

```
public static void main(String[] args) {

    Cachorro dog = new Cachorro();
    dog.nome = "bob";

}
```

```
*/
public sta
    Cachor
    dog.nome = "bob";
```

nome has protected access in Cachorro

(Alt-Enter mostra dicas)

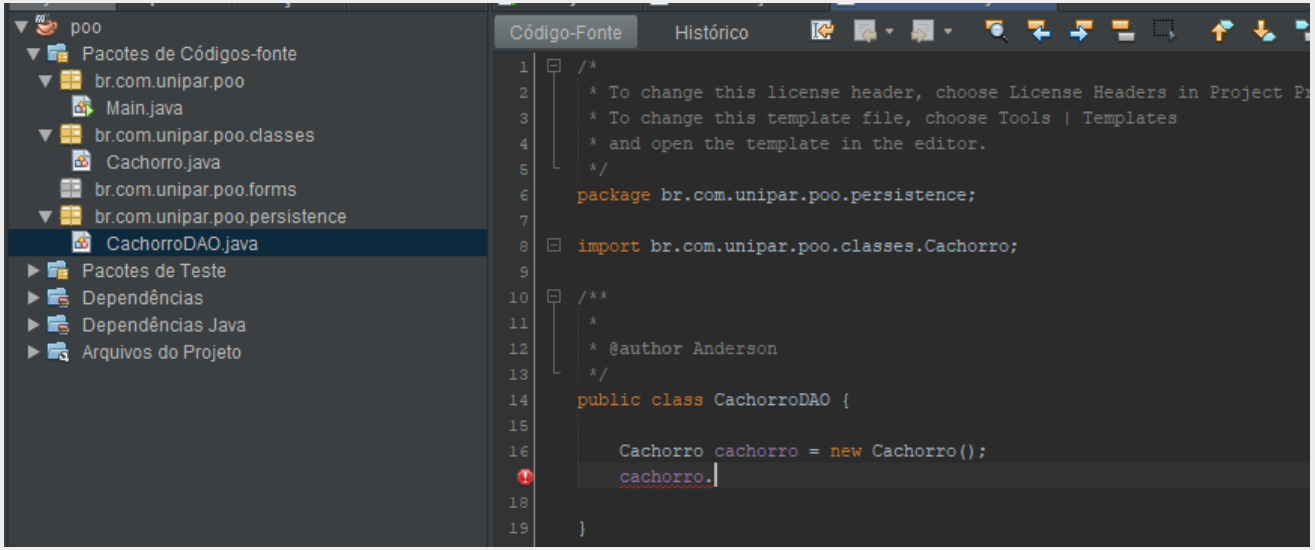
Modificadores de Acesso a Nível de Atributos e Métodos(Membros) - default

default (sem modificador explícito) torna um membro acessível apenas para classes do mesmo pacote.

```
 */
public class Cachorro {

    String nome;

}
```



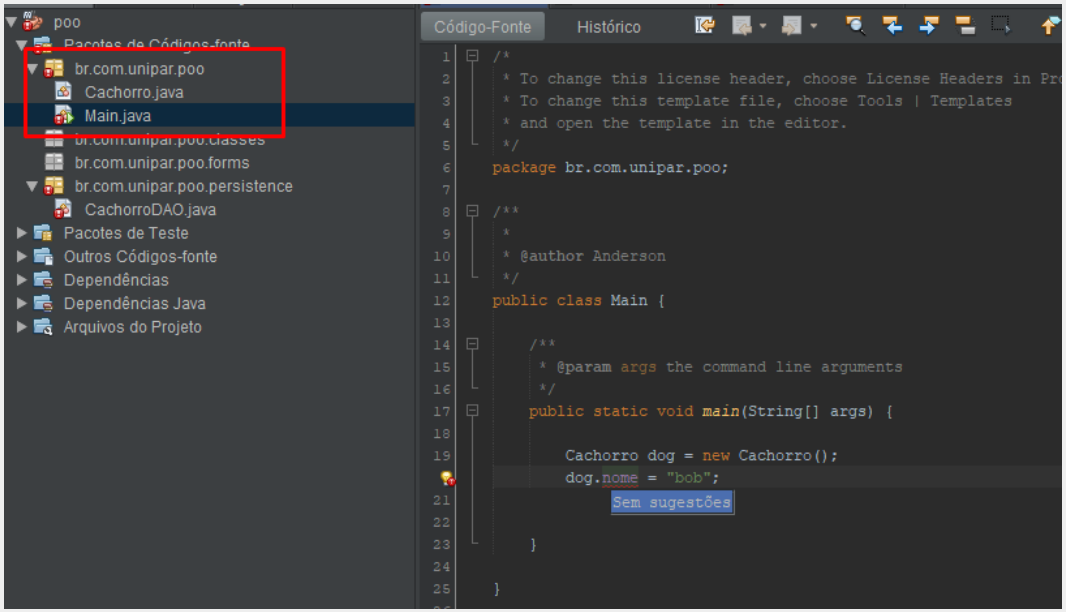
Modificadores de Acesso a Nível de Atributos e Métodos(Membros) - private

private torna um membro acessível apenas para a classe que o contém.

```
public class Cachorro {

    private String nome;

}
```



Objetos não devem alterar os estados dos outros ou seus atributos.

Exemplos:

Pessoa e um Carro.

Pessoa e um Porta.

Professor, mas se um objeto não pode alterar diretamente os atributos e o estado de outro objeto, como fazemos nosso código abaixo?

```
*/
public class Cachorro {

    String nome;

}
```

```
public class Main {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {

        Cachorro dog = new Cachorro();
        dog.nome = "bob";


    }

}
```

Professor, mas se um objeto não pode alterar diretamente os atributos e o estado de outro objeto, como fazemos nosso código abaixo?

Através dos métodos acessores.

Métodos Acessores(Get e Set)

Também conhecidos como getter e setters, são métodos utilizados para que seja possível acessar e modificar os valores de variáveis com modificador de acesso private.

Getters

O propósito de um getter é obter o valor de uma variável declarada como private e permitir sua leitura a partir de outra classe.

Setters

Um setter é um método que permite modificar o valor de um atributo da classe que não seja acessível diretamente por ser privado.

Getters e Setters

```java
 */
public class Cachorro {
   //Atributo privado
    private String nome;

    //No caso do setter o retorno do metodo é void ou seja sem retorno pois
    //usamos o metodo para setar o valor a um atributo privado
    //Acessor do metodo + Tipo de Retorno + nomeMetodo(Tipo e Parametro de Entrada)
    public void setNome(String nome) {
        this.nome = nome;//this identifica o contexto do que está sendo setado
                         //nesse caso o nosso contexto é a própria classe
                         //fazendo com que this.nome seja diferente da variavel de entrada nome
    }

    //No caso do getter como apenas queremos buscar o valor do atributo privado não temos
    //parametros de entrada no metodo mas temos o tipo de retorno
    //nesse caso como o atributo se trata de uma string
    //o tipo de retorno é uma string
    //Acessor do metodo + Tipo de Retorno + nomeMetodo()
    public String getNome() {
        return nome; //return é um comando reservado que identifica no java qual será o retorno do metodo
    }

}
```

Vamos codificar.

- Criar uma classe câmera.

- Criar uma classe gato.

NetBeans

Vamos Praticar?

- **Lista 2 - Exercicios Orientação a Objetos com Java.docx**

Programação Orientada a Objetos

Aula: Encapsulamento, Construtores e Destrutores.

Prof. Anderson Augusto Bosing

Encapsulamento

Classes (e seus objetos) encapsulam, isto é, contêm seus atributos e métodos. Os atributos e métodos de uma classe (e de seu objeto) estão intimamente relacionados.

Os objetos podem se comunicar entre si, mas eles em geral não sabem como outros objetos são implementados — os detalhes de implementação permanecem ocultos dentro dos próprios objetos. (Deitel, 2016).

Encapsulamento

Encapsular é tornar o código dentro de uma classe acessível ou inacessível para objetos fora da classe. A lógica que suporta este comportamento é que cada classe deve ter um significado e por si só deve descrever o comportamento de um objeto.

A palavra chave this

Palavra reservada do Java que referencia atributos e métodos da própria classe.

Objetos e Mais Objetos

É comum que sejam criadas classes para representar objetos do mundo real, e tão comum quanto isso são que os atributos das classes também seja identificados como outras classes.

Vamos a um exemplo.

Se nós temos uma classe que representa um carro, esse carro tem uma marca que tem seus próprios atributos. E essa marca consequentemente pode possuir um endereço que também possui seus próprios atributos.

Mas e professor, como isso seria codificado?

Objetos e Mais Objetos

Vamos a outro exemplo.

Um banco possui muitas agencias e estas agencias possuem muitos correntistas, sendo contas correntes ou contas poupanças. Cada correntista possui um endereço para onde deve ser enviado o cartão.

Mas e professor, como isso seria codificado?

Por que Utilizar o Encapsulamento?

Organização de código.

Quando criamos classes, cada uma delas possui uma função específica. Ou seja, elas descrevem um objeto específico, sendo assim, classes coesas não realizam várias funções.

Manutenção de código.

- **Depois de pronto, todo código, principalmente os mais extensos, são propensos a sofrer manutenções.**
- **Com o encapsulamento, isso passa a ser mais fácil, uma vez que, com a devida proteção de acesso aos dados, a pessoa programadora achará mais rápido algum ponto onde o código precisa ser melhorado.**

Por que utilizar o Encapsulamento?

Reuso de Código.

Com o encapsulamento, o programa terá mais chances de ter o código reaproveitado em outros projetos, poupando bastante tempo da equipe de desenvolvimento.

Simplificação da codificação.

O encapsulamento transforma a implementação de alguns códigos em uma espécie de caixa preta. Na prática, isso significa que as classes externas não precisam acessar alguns dados de forma direta. Assim, o desenvolvimento dos sistemas passa a ficar simplificado e acelerado.

Construtores

Construtores são os métodos responsáveis por criar uma instancia da classe em memória, ou seja instanciar um objeto em memória.

O método construtor é muito semelhante a um método comum, porém ele se difere dos demais métodos por alguns pontos bem específicos e importantes para a linguagem Java.

Construtores

Em primeiro lugar o método construtor deve possuir o mesmo nome da classe, sendo assim, é o único método por padrão Java que será nomeado com a primeira letra em maiúscula.

Outro ponto importante é que um construtor nunca terá um tipo de retorno, poderá ser do tipo private, public, protected ou default, mas nunca terá um tipo de retorno nem que ele seja do tipo void.

Construtores
Classe sem construtor explicito

```java
public class Animal {

    private double peso;
    private String grupo;

    public double getPeso() {
        return peso;
    }

    public void setPeso(double peso) {
        this.peso = peso;
    }

    public String getGrupo() {
        return grupo;
    }

    public void setGrupo(String grupo) {
        this.grupo = grupo;
    }

}
```

```java
public class Main {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {

        Animal animal = new Animal();

    }

}
```

Construtores
Classe com construtor explicito

```java
public class Animal {

    public Animal() {
    }

    private double peso;
    private String grupo;

    public double getPeso() {
        return peso;
    }

    public void setPeso(double peso) {
        this.peso = peso;
    }

    public String getGrupo() {
        return grupo;
    }

    public void setGrupo(String grupo) {
        this.grupo = grupo;
    }

}
```

```java
public class Main {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {

        Animal animal = new Animal();

    }

}
```

Construtores
Classe com construtor explicito e Parâmetros de entrada

```java
public class Animal {

    public Animal(double peso, String grupo) {
        this.peso = peso;
        this.grupo = grupo;
    }

    private double peso;
    private String grupo;

    public double getPeso() {
        return peso;
    }

    public void setPeso(double peso) {
        this.peso = peso;
    }

    public String getGrupo() {
        return grupo;
    }

    public void setGrupo(String grupo) {
        this.grupo = grupo;
    }

}
```

```java
public class Main {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {

        Animal animal = new Animal(2, "Hunter");

    }

}
```

Exercícios

Destrutores

Antes de começarmos a falar sobre destrutores, primeiro eu preciso dizer que o Java não possui métodos destrutores.

O que é Garbage Collector?

O garbage collection é o processo pelo qual os programas Java executam o gerenciamento automático de memória. Os programas Java compilam para bytecode que pode ser executado em um Java Virtual Machine (JVM).

Quando os programas Java são executados na JVM, os objetos são criados no heap, que é uma parte da memória dedicada ao programa. Eventualmente, alguns objetos não serão mais necessários. O garbage collector localiza esses objetos não utilizados e os exclui para liberar memória.

O mais próximo dos Destrutores

O método finalize, é um método protected definido na classe Object, e que pode ser redefinido pelo programador em sua classe. Ele é executado automaticamente quando a área do objeto da classe que a contem estiver para ser liberada pelo coletor.

O mais próximo dos Destrutores

```java
public class Animal {

    public Animal(double peso, String grupo) {
        this.peso = peso;
        this.grupo = grupo;
    }

    private double peso;
    private String grupo;

    public double getPeso() {
        return peso;
    }

    public void setPeso(double peso) {
        this.peso = peso;
    }

    public String getGrupo() {
        return grupo;
    }

    public void setGrupo(String grupo) {
        this.grupo = grupo;
    }

    @Override
    protected void finalize() throws Throwable {
        System.out.println("Finalizar");
        super.finalize();
    }

}
```

```java
public static void main(String[] args) {

    Animal animal = new Animal(2, "Hunter");
    animal = null;
    System.gc();

}
```

```
Building Teste 1.0
--------------------------------[ jar ]---------------------------------

--- exec-maven-plugin:1.2.1:exec (default-cli) @ Teste ---
Finalizar
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
```

Sobrecarga de Métodos

A sobrecarga de métodos (overload) consiste basicamente em criar variações de um mesmo método, ou seja, a criação de dois ou mais métodos com nomes totalmente iguais em uma classe mas que possuam assinaturas diferentes podendo ter retornos diferentes, parâmetros de entradas diferentes.

```
*/
public class Calculadora {

    public int soma(int num1, int num2) {
        return num1 + num2;
    }

    public int soma(int num1, int num2, int num3) {
        return num1 + num2 + num3;
    }

    public String soma(double num1, double num2) {
        return String.valueOf(num1 + num2);
    }

}
```

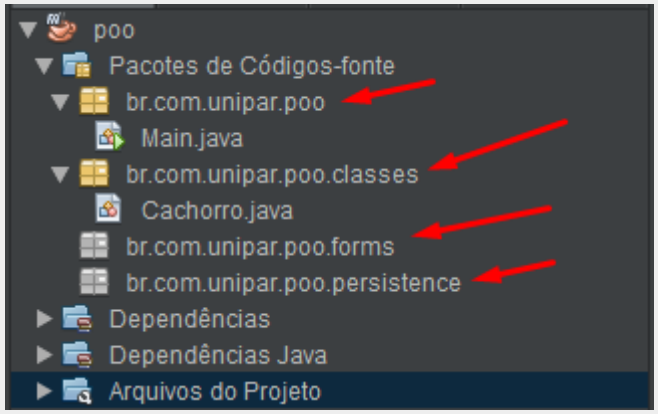
Programação Orientada a Objetos

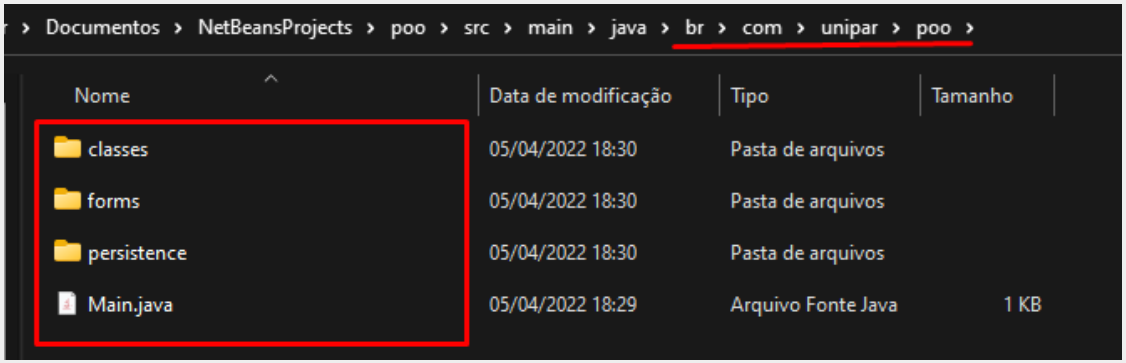
Aula: Packages e Arraylist.

Prof. Anderson Augusto Bosing

Pacotes(Packages)

Pacotes java são utilizados para organizar as classes da sua aplicação. Um programa pode, facilmente, ter mais de centenas de classes. Então é muito importante que todos os seus componentes fiquem organizados. Podemos pensar nos pacotes como uma pasta do seu sistema de arquivos.





Categorias de pacotes em Java

Existem dois tipos de pacotes em java:

- **Pacote definido pelo usuário (criar seu próprio pacote)**

- **Pacotes integrados(built in) que são pacotes da interface de programação de aplicativos Java que são os pacotes da API Java, por exemplo. como swing, util, net, io, AWT, lang, javax, etc.**

Pacotes integrados(built in)

Pactos Básicos:

java.lang- classes fundamentais do java(Boolean, Double, Float, Long, Math, String).

java.util – classes utilitárias do java(List, Arraylist, Arrays, Date, Timer, Timezone).

java.io – classes de para entrada e saída(File, FileInputStream, FileOutPutStream)

java.net - Classes para uso de comunicação de rede(TCP/IP, HTTP, PROXY, SOCKET).

java.sql - Classes para utilização e acesso de conexões JDBC responsaveis pelo acesso ao banco de dados(Connection, Driver, ResultSet, PreparedStatement).

java.awt - Classes para utilização e criação de interfaces desktop em java(Button, Checkbox, Dialog, Event, Font, Frame, Label, Menu, TextField, TextArea).

Pacotes integrados(built in)

Pactos Básicos:

java.text - Classes para formatação e transformação de texto(SimpleDateFormat, NumberFormat, DecimalFormat).

java.math - Classes para lidar com numeros precisos(Arredondar, randomizar um numero, abs, max, min).

java.time - Classes para utilização em dados que envolvem hora ou tempo.(Year, LocalTime, ZoneId, DayOfWeek, Month).

java.util.zip - Classes para lidar com arquivos ZIP's.

javax.swing - Classes para utilização e criação de interfaces desktop em java(Button, Checkbox, Dialog, Event, Font, Frame, Label, Menu, TextField, TextArea).

Pacotes integrados(built in)

https://docs.oracle.com/javase/8/docs/api/

Pacotes Definidos Pelo Usuário

Vamos praticar, pensando que criaremos um novo jogo FPS, quais classes seriam criadas ?

Como podemos organizar essas classes em pacotes?

Usuario, Pistola, Submetralhadora

Rifle, Armas Pesada

Utilitário, Mapa

Personagem, Fila

Janela, BotaoJogar

MenuPrincipal, MenuArma

MenuUsuario, MenuTab, MenuKick

Pacotes Definidos Pelo Usuário

Vamos praticar, pensando que criaremos uma nova netflix, quais classes seriam criadas ?

Como podemos organizar essas classes em pacotes?

Filme

Serie

Anime

Novela

Documentario

Janela

Principal

Miniatura

Video

BotaoPlay

BotaoPause

Motivos para Usar Packages

- Separação e Organização das classes.
- Definição das responsabilidades das classes.
- Evitar Conflitos de nomes em caso de classes com nomes idênticos.

Convenções em Pacotes

- **SEMPRE em Lower case(letra minúscula)**
- **Domínio da aplicação de forma reversa + Nome da Aplicação**
- **Exemplos:**

  br.unipar.nomeaplicacao

  br.com.pratidonaduzzi.nomeaplicacao

ArrayList

O Java ArrayList, é, basicamente, um array dinâmico que encontramos no java.util — um pacote que contém uma grande variedade de recursos, como estruturas de coleções, modelos de data e hora, recursos para internacionalização, entre muitos outras facilidades para o desenvolvimento de aplicações em Java.

Esses arrays redimensionáveis são muito úteis quando utilizados para implementações em que precisamos manipular listas.

ArrayList

A dinamicidade do recurso possibilita ao desenvolvedor a criação de coleções — arrays, classes e objetos — sem precisar se preocupar com o redimensionamento dos vetores. Caso algum haja necessidade de uma posição adicional em um array, o ArrayList realiza a operação de maneira autônoma.

Quais as principais características da classe Arraylist Java?

Entre as interfaces e classes das estruturas disponibilizados para uso durante o desenvolvimento de aplicações em Java, certamente, o ArrayList é uma das principais delas. Sua característica de construção dinâmica de arrays possibilita manipulá-las com o uso de métodos para adicionar ou retirar objetos.

Quais as diferenças entre array e arraylist em java?

Quando utilizamos o array em Java, estamos trabalhando com tamanhos fixos de coleções. Para inserirmos algum elemento naquele grupo de objetos, precisamos criar outro que contemple aquele novo tamanho, com determinado número de posições a mais — isso não acontece com o ArrayList.

Quais as diferenças entre array e arraylist em java?

É muito comum a confusão de que arrays e ArrayList são a mesma coisa, mas saiba que eles são muito diferentes. O ArrayList não é um array padrão, ele apenas utiliza essa funcionalidade para realizar o armazenamento dos objetos contidos nas listas manipuladas por ele. Nem ao menos os atributos desse array, que o ArrayList utiliza, é possível ser acessado durante o processamento.

Quais as diferenças entre array e arraylist em java?

Como a classe ArrayList é um agrupamento dinâmico de objetos, isso quer dizer que podemos adicionar ou retirar elementos de, por exemplo, uma lista, sem que seja necessário criar uma nova, mantendo a original com um número diferente dos objetos ali contidos.

Quais as diferenças entre array e arraylist em java?

Desde o momento em que criamos um array, seu tamanho não pode ser mudado:

String [] meuStringArray = new String[3];

Repare que, nesse caso, independentemente de termos objetos ou não dentro de nosso novo array, sempre teremos para ele três posições disponíveis, nunca mais do que isso.

Quais as diferenças entre array e arraylist em java?

O que é bem diferente para o ArrayList.

Aqui, trazemos um exemplo prático da aplicação:

```
List lista = new ArrayList();
lista.add("Pessoa 1");
lista.add("Pessoa 2");
lista.add("Pessoa 3");
```

Isso porque a classe ArrayList é independente do número de objetos contidos nela. Podemos aumentar ou diminuir seu tamanho, apenas inserindo ou retirando mais objetos.

Quais os principais métodos usados com a classe Arraylist?

- new ArrayList(): cria um novo ArrayList. Por padrão, essa classe tem a capacidade inicial de 10 elementos;
- add(item): é o método utilizado para adicionar novos elementos ao ArrayList. Os elementos são colocados no fim da lista, por padrão;
- remove(posição): remove um item de determinada posição na lista criada;
- set(item, lista): usamos para definir um elemento em determinado index;
- get(item): retorna o objeto ligado a determinado índice;
- iterator(): responsável por iterar um elemento na sequência adequada do vetor;
- clear(): limpa todos os elementos contidos na lista.

Vamos Praticar?

- Vamos praticar, vamos criar uma agenda de contatos com endereço e telefones para cada contato.

Vamos Praticar?

- Vamos praticar, vamos criar uma agenda de consultas para uma clinica médica.

Vamos Praticar?

- Vamos praticar, vamos criar um sistema para o cinema de toledo. Onde devem ser gerenciadas as cadeiras, salas de cinema e a ocupação delas pelos clientes.

Vamos Praticar?

- Lista 3 - Exercícios Orientação a Objetos com Java.docx

Programação Orientada a Objetos

Aula: Passagem de parâmetros, Polimorfismo e Herança.

Prof. Anderson Augusto Bosing

Herança

Herança é uma forma de reutilização de software, onde uma nova classe é criada absorvendo atributos e métodos de uma classe existente.

Esta nova classe pode ter características mais específicas ou modificadas em comparação com a classe antiga/absorvida.

Com a herança, o tempo de desenvolvimento de um software é reduzido e a depuração é facilitada.

A classe nova é chamada de SUBCLASSE, já a classe antiga, que é absorvida pela nova, é chamada de SUPERCLASSE.

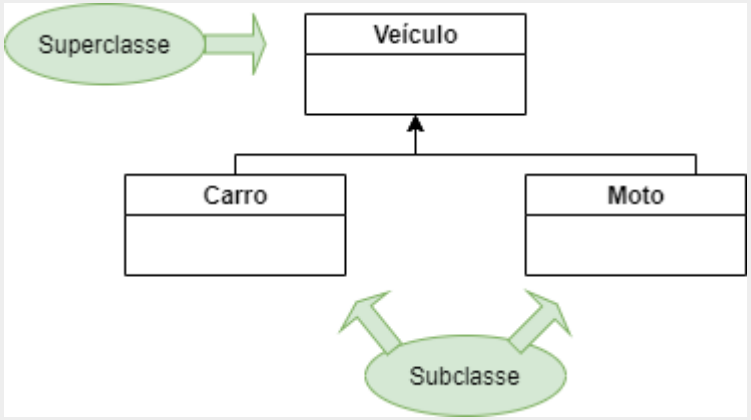
Herança

A herança pode se dar em vários níveis, formando uma hierarquia.

A classe imediatamente superior é uma SUPERCLASSE direta.

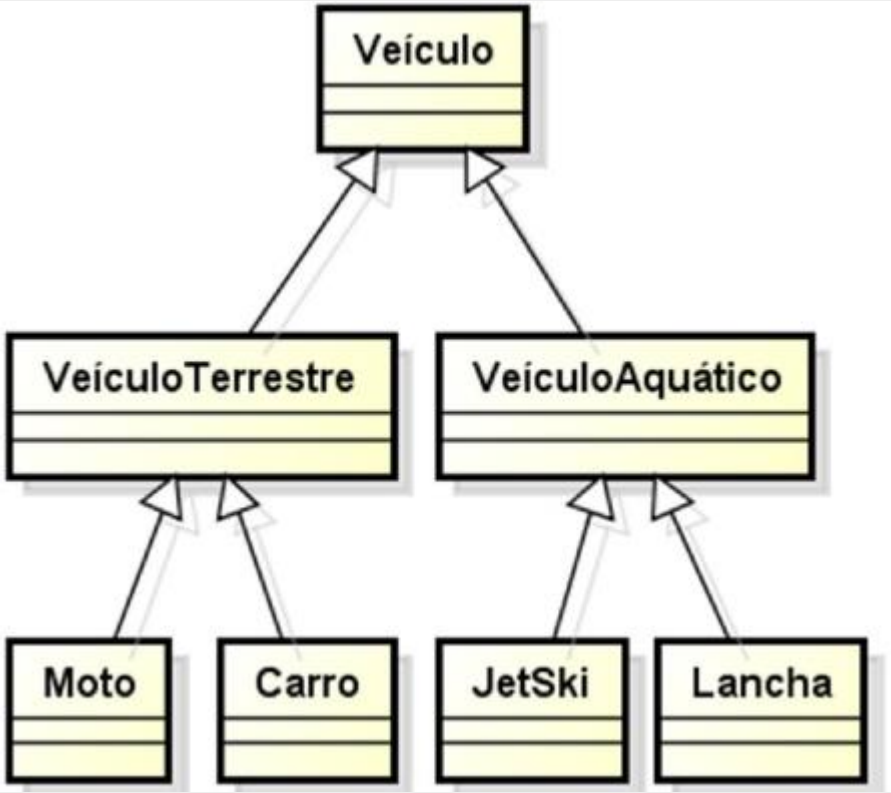
Uma classe que não seja imediatamente superior é uma SUPERCLASSE indireta.

Herança
Exemplo



A classe veiculo é uma SUPERCLASSE direta das classes carro e moto que são subclasses de veiculo.

Herança
Exemplo



Alguns apontamentos:

1-A Moto é um VeículoTerrestre, mas também é um Veículo.

2-A Lancha é um Veículo, porém não é VeículoTerrestre.

3-Moto e JetSki são Veículos, porém um é VeículoTerrestre e o outro é VeículoAquático.

4-VeículoAquático e VeículoTerrestre compartilham dados da classe veículo.

Herança

Então, para criarmos uma herança em nosso sistema, utilizamos a palavra reservada **extends**.

Herança
Exemplo

```java
/**
 *
 * @author Anderson
 */
public class Veiculo {

    public String marca;
    public String modelo;

    public Veiculo() {
    }

    public Veiculo(String modelo, String marca) {
        this.modelo = modelo;
        this.marca = marca;
    }
}
```

```java
 *
 * @author Anderson
 */
public class Carro extends Veiculo {

    public int porta;

    // Chamada implicita para o construtor de Veiculo
    public Carro() {
    }

    // Chamada explicita para o construtor de Veiculo
    public Carro(String marca, String modelo, int porta) {
        super(modelo, marca);
        this.porta = porta;
    }

}
```

Herança

Para compreender melhor o código acima, é importante conhecer as referências this e super.

Referência this: Referência à membros do objeto corrente.
Referência super: Referência à membros da superclasse, sendo também utilizado para chamar o construtor: Subclasse chama o construtor da superclasse; Primeira instrução no construtor da subclasse.

Herança
Exemplo

Pessoa.java | Medico.java | Enfermeira.java | Paciente.java

```java
import java.util.Date;

public class Pessoa {
    int idCadastro;
    String nome;
    String cpf;
    Date dataNascimento;

    public Pessoa(int idCadastro, String nome, String cpf, Date dataNascimento) {
        this.idCadastro = idCadastro;
        this.nome = nome;
        this.cpf = cpf;
        this.dataNascimento = dataNascimento;
    }
```

Pessoa.java | Medico.java | Enfermeira.java | Paciente.java

```java
import java.util.Date;

public class Enfermeira extends Pessoa {
    public Enfermeira(int idCadastro, String nome, String cpf, Date dataNascimento) {
        super(idCadastro, nome, cpf, dataNascimento);
    }
    private String coren;
    private String siglaEstadoCoren;
    private char setorEnfermaria;
}
```

Pessoa.java | Medico.java | Enfermeira.java | Paciente.java

```java
import java.util.Date;

public class Medico extends Pessoa {

    public Medico(int idCadastro, String nome, String cpf, Date dataNascimento) {
        super(idCadastro, nome, cpf, dataNascimento);
    }
    private String crm;
    private String siglaEstadoCRM;
    private String especialidade;
```

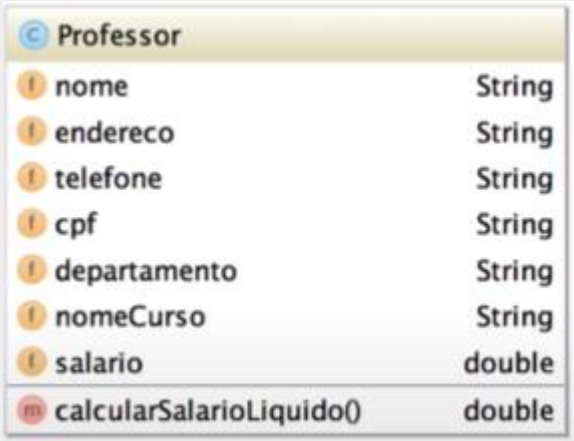
Pessoa.java | Medico.java | Enfermeira.java | Paciente.java

```java
import java.util.Date;

public class Paciente extends Pessoa {
    public Paciente(int idCadastro, String nome, String cpf, Date dataNascimento) {
        super(idCadastro, nome, cpf, dataNascimento);
    }
    private boolean internado;
    private char setor;
    private short leito;
    private String diagnostico;
}
```

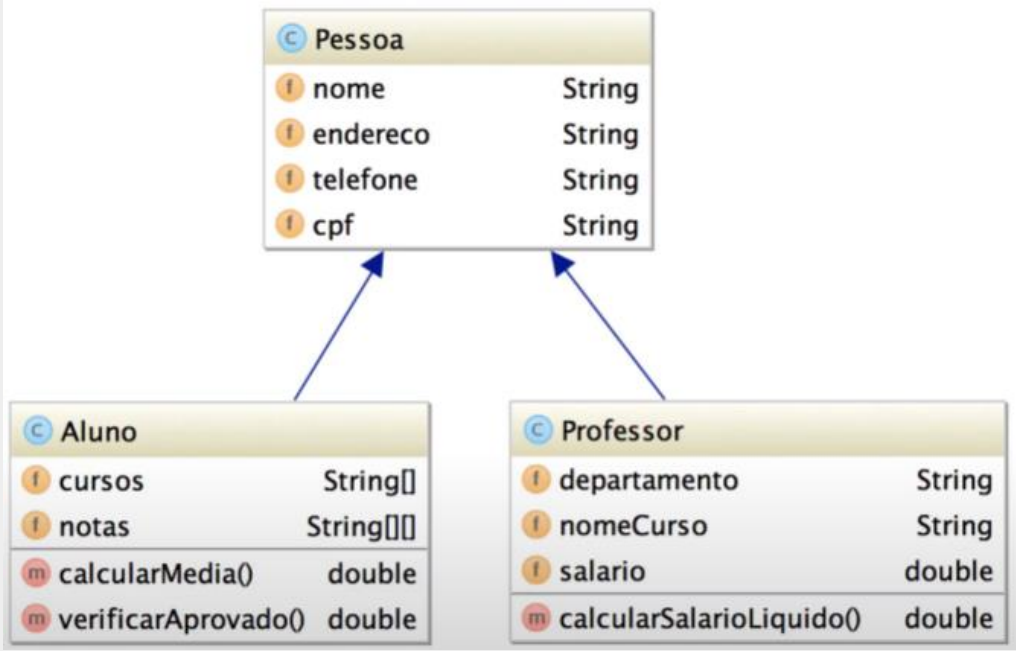
Vamos Praticar?

Criaremos a Classe Professor e Aluno.



Vamos Praticar?

Criaremos a Classe Professor, Aluno e Pessoa.



Herança - Importante

- Uma classe só pode herdar de uma outra classe (herança simples).
- Caso não seja declarada herança, a classe herda da classe Object (Ela define o método toString(), que retorna a representação em String do objeto, qualquer subclasse pode sobrescrever o método toString() para retornar o que ela deseja.)

Veja os demais métodos da classe Object em:

https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html

Sobrescrita de Métodos

Importante também falarmos sobre sobrescrita de métodos, A sobrescrita (ou override) está diretamente relacionada à orientação a objetos, mais especificamente com a herança. Com a sobrescrita, conseguimos especializar os métodos herdados das superclasses, alterando o seu comportamento nas subclasses por um mais específico.

A sobrescrita de métodos consiste basicamente em criar um novo método na classe filha contendo a mesma assinatura e mesmo tipo de retorno do método sobrescrito.

Os métodos podem ser sobrescritos, o que é diferente de sobrecarga por terem a mesma assinatura e o tipo de retorno.

Chegou a sua vez?

- **Lista 4 - Exercicios Orientação a Objetos com Java.docx**