

FIA/P GRADUAÇÃO

ENTERPRISE APPLICATION DEVELOPMENT

Prof. Me. Thiago T. I. Yamamoto

#01 - APRESENTAÇÃO

#01 - AGENDA

- Apresentação do Professor
- Objetivos da disciplina
- Conteúdo programático
- Metodologia e Avaliações



THIAGO T. I. YAMAMOTO



Mestre em Ciências, Gestão e Informática em Saúde pela Universidade Federal de São Paulo. Pós-graduado em Engenharia de Sistemas. Bacharel em Ciências da Computação pela Universidade Estadual Paulista - Unesp/Bauru. Mais de 10 anos de experiência na área de TI, como **desenvolvedor de sistemas nas empresas**: Autbank, UOL e Ericsson Telecomunicações. **Professor em curso graduação na FIAP** a mais de 10 anos, ministra várias disciplinas de desenvolvimento de sistemas. Certificado ITIL V3, PSM I e OCJA.

O mais importante: apaixonado por ensinar.



ENTERPRISE APPLICATION DEVELOPMENT

- Preparar o profissional para o **Mercado de Trabalho**;
- Trabalhar com frameworks **utilizados** no mercado;
- Desenvolver *back end* de aplicações **Java EE**;
- Apresentar a **Plataforma .NET** e implementar aplicações web com **ASP.NET Core** e **Entity Framework**;



1º Semestre

- ORM – JPA/Hibernate;



2º Semestre

- Plataforma .NET
 - ASP.NET Core;
 - Entity Framework Core;



ASP.NET



- **Lógica de programação**
 - Conhecer os operadores relacionais, estruturas de seleção, repetição, trabalhar com vetores e etc.;
- **Java e Orientação a Objetos**
 - Sintaxe da linguagem, construtores, métodos, atributos, tratamento de exceções, collections e etc.;
 - Os pilares da orientação a objetos: abstração, herança, polimorfismo e encapsulamento;
- **Banco de dados e SQL**
 - Tabelas, Colunas e Relacionamentos;
 - SQL, select, order by, count, group by e etc..



METODOLOGIA E AVALIAÇÕES

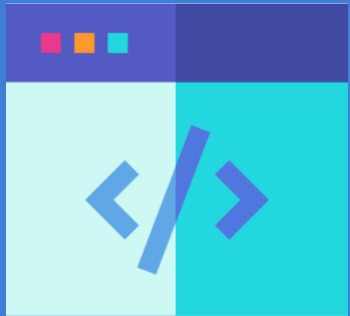
- Metodologia
 - Aulas “Hands On”;
 - Projetos no Github;





REVISÃO

JAVA



- Classe;
- Objeto;
- Herança;
- Encapsulamento;
- Atributos e Métodos;
- Construtores;
- Enums;
- Interfaces;
- O que mais?

- Palavras reservadas;
- Classes:
 - Qual a diferença entre classe e objeto?
- Métodos:
 - Diferença de sobrecarga e sobrescrita?
- Atributos:
 - Quais os tipos e valores padrões?
- Construtores:
 - Precisa sempre ter o construtor “cheio” e vazio?



- Operadores matemáticos:

+ , - , * , / , %

- Operadores relacionais:

==, !=, >=, <=, >, <, !, &&, ||

- Instrução if e else;

- Operador ternário:

– (condicao) ? seVerdadeiro : seFalso;



- FOR

```
for (int i=0; i<10; i++) { }
```

- WHILE

```
while(x == 0) { }
```

- DO WHILE

```
do { } while(x <0);
```

- FOR EACH

```
for (String item : lista) { }
```



- LISTS

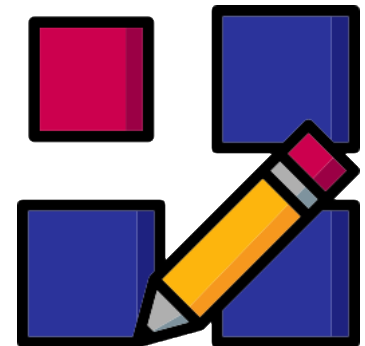
```
List<String> lista = new ArrayList<String>();
```

- SETS

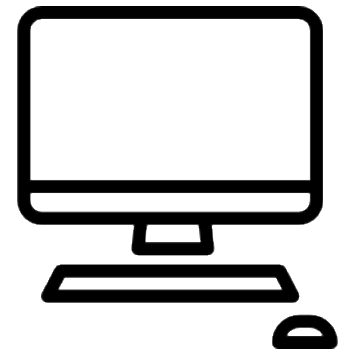
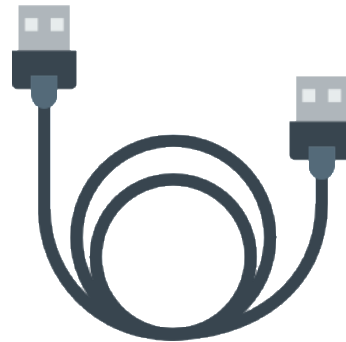
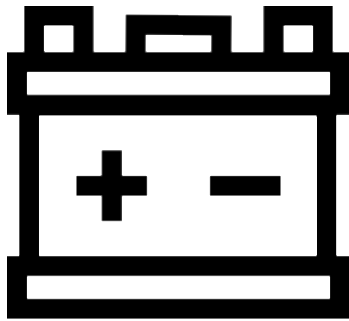
```
Set<Integer> set = new HashSet<Integer>();
```

- MAPS

```
Map<Integer,String> map = new HashMap<>();
```



- Uma **interface** é um conjunto nomeado de **comportamentos** para o qual um implementador precisa fornecer o código;
- Define as **assinaturas** dos métodos;




```
public interface ClienteDAO {  
    void cadastrar(Cliente cliente);  
    List<Cliente> listar();  
}
```

Interface define dois métodos.

Duas classes implementam a interface, uma para utilizar o banco Oracle e outro para o MySQL.

```
public class ClienteDAOMySQL implements ClienteDAO {  
  
    @Override  
    public void cadastrar(Cliente cliente) { //... }  
  
    @Override  
    public List<Cliente> listar() { //... }  
}
```

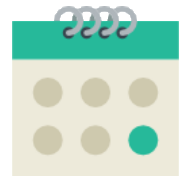
```
public class ClienteDAOOracle implements ClienteDAO {  
  
    @Override  
    public void cadastrar(Cliente cliente) { //... }  
  
    @Override  
    public List<Cliente> listar() { //... }  
}
```

- DATE

- Classe que armazena o tempo, porém a maioria dos métodos estão marcados como **deprecated**;

- CALENDAR

- Classe **abstrata** para trabalhar com Data no Java:
- `Calendar hoje = Calendar.getInstance();`
- `Calendar data = new GregorianCalendar(ano,mes,dia);`



- SimpleDateFormat

```
Calendar data = Calendar.getInstance();  
  
SimpleDateFormat format = new  
    SimpleDateFormat("dd/MM/yyyy");  
  
format.format(data.getTime());
```



Java 8 possui uma nova API para datas:

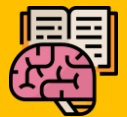
- **LocalDate** - data, sem horas;
 - `LocalDate hoje = LocalDate.now();`
 - `LocalDate data = LocalDate.of(ano, mes, dia);`
- **LocalTime** - horas, sem data;
 - `LocalTime time = LocalTime.now();`
 - `LocalTime horas = LocalTime.of(horas, minutos);`
- **LocalDateTime** - data e horas;
 - `LocalDateTime dateTime = LocalDateTime.now();`
 - `LocalDateTime dataHora = LocalDateTime.of(ano, mes, dia, hora, minutos);`

Formatação de datas:

```
LocalDate hoje = LocalDate.now();  
  
DateTimeFormatter formatador =  
    DateTimeFormatter.ofPattern("dd/MM/yyyy");  
  
hoje.format(formatador);
```

Artigo sobre API de Datas do Java 8:

<http://blog.caelum.com.br/conheca-a-nova-api-de-datas-do-java-8/>



- Define um conjunto de constantes, valores que não podem ser modificados.

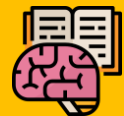
```
public enum Dias {
```

```
    SEGUNDA, TERCA, QUARTA, QUINTA, SEXTA,  
    SABADO, DOMINGO;
```

```
}
```

Artigo sobre enums:

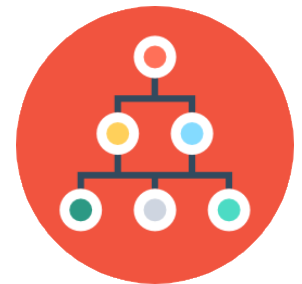
<https://www.devmedia.com.br/tipos-enum-no-java/25729>





ORIENTAÇÃO À OBJETOS

- **Abstração**
 - Representação do objeto, características e ações;
- **Encapsulamento**
 - Restringir o acesso às propriedades e métodos;
- **Herança**
 - Reutilização de código;
- **Polimorfismo**
 - Modificação do comportamento de um método herdado;



Copyright © 2013 – 2023

Prof. Me. Thiago T. I. Yamamoto

Todos direitos reservados. Reprodução ou divulgação total ou parcial deste documento é expressamente proibido sem o consentimento formal, por escrito, do Professor (autor).