FIAP GRADUAÇÃO

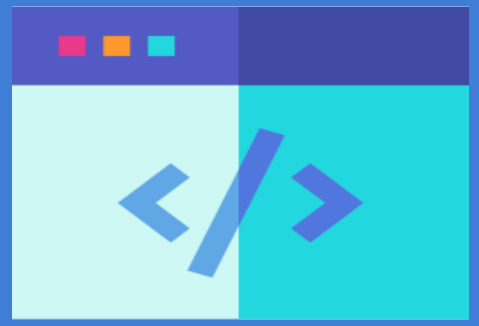
JAVA ADVANCED

PRÉ-REQUISITOS

- **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..

REVISÃO

JAVA



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

JAVA - FUNDAMENTOS

- **Palavras reservadas;**
- **Classes;**
- **Métodos;**
- **Atributos;**
- **Construtores.**

JAVA - OPERADORES

- **Operadores matemáticos:**

  + , - , * , / , %

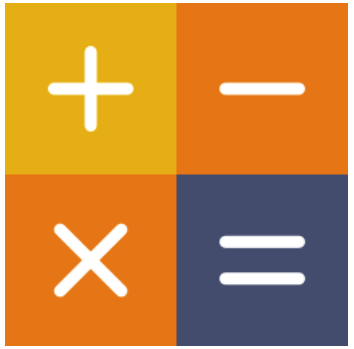
- **Operadores relacionais:**

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

- **Instrução if e else;**

- **Operador ternário:**

  – (condicao) ? seVerdadeiro : seFalso;



JAVA - LOOPS

- **FOR**

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

- **WHILE**

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

- **DO WHILE**

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

- **FOR EACH**

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

JAVA - COLLECTIONS

- **LISTS**

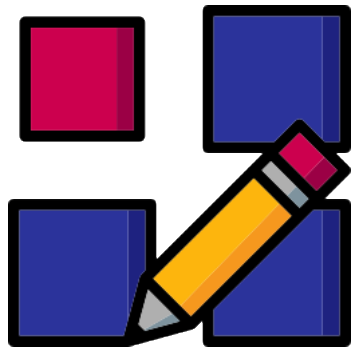
```
List<String> lista = new ArrayList<String>();
```

- **SETS**

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

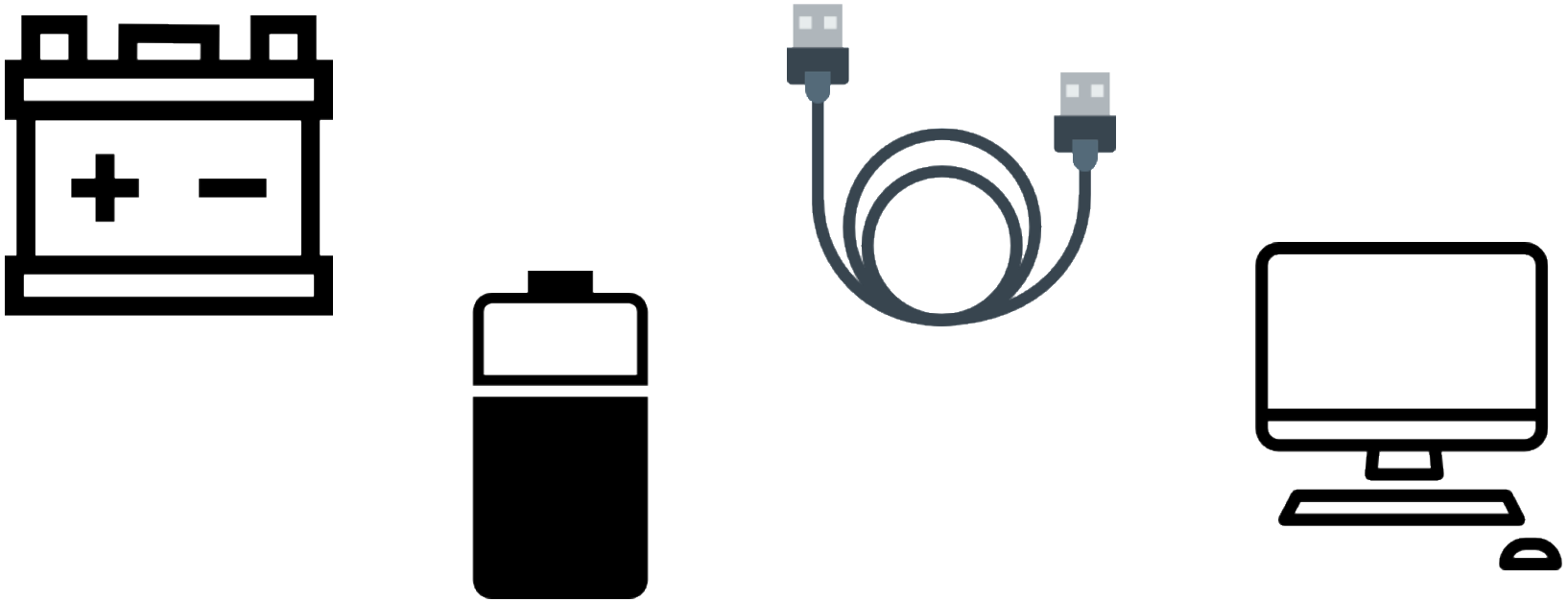
- **MAPS**

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



INTERFACES

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



INTERFACES

```
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() { //... }

}
```

DATAS

- **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);
```

DATAS - FORMATAÇÃO

- **SimpleDateFormat**

```
Calendar data = Calendar.getInstance();

SimpleDateFormat format = new
        SimpleDateFormat("dd/MM/yyyy");

format.format(data.getTime());
```

DATAS – JAVA 8

Java 8 possui uma 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;

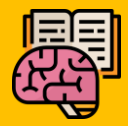
```
LocalDateTime dateTime = LocalDateTime.now();
LocalDateTime dataHora = LocalDateTime.of(ano, mes, dia, hora, minutos);
```

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/



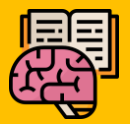
ENUM

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

```
public enum Dias {

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

}
```

Artigo sobre enums:

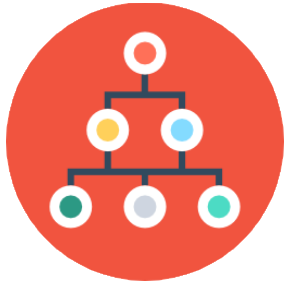
https://www.devmedia.com.br/tipos-enum-no-java/25729



ORIENTAÇÃO À OBJETOS

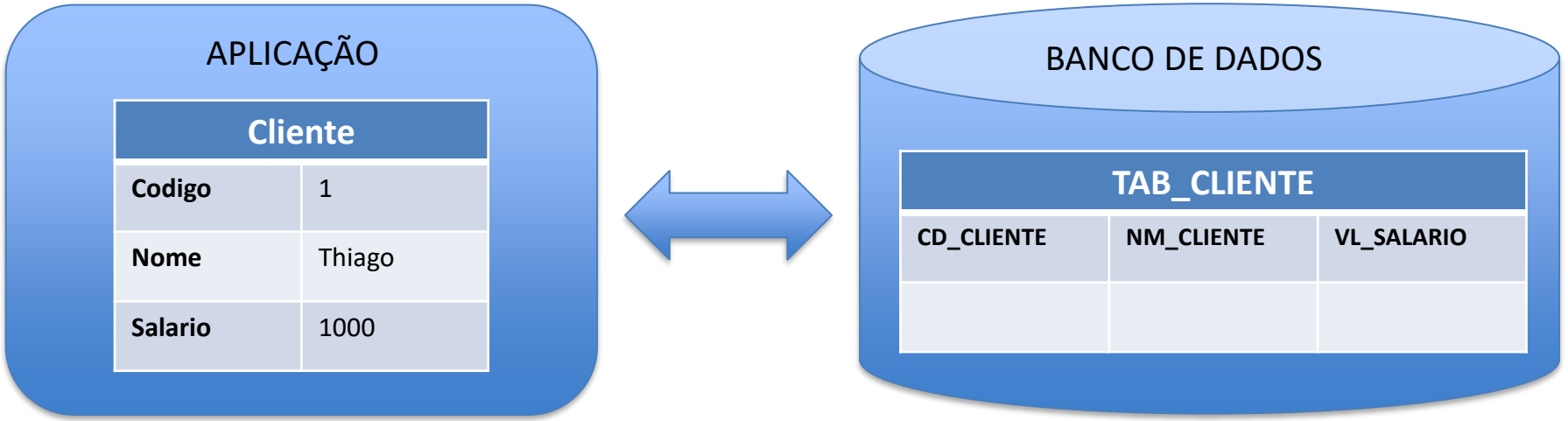
POO

- **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;



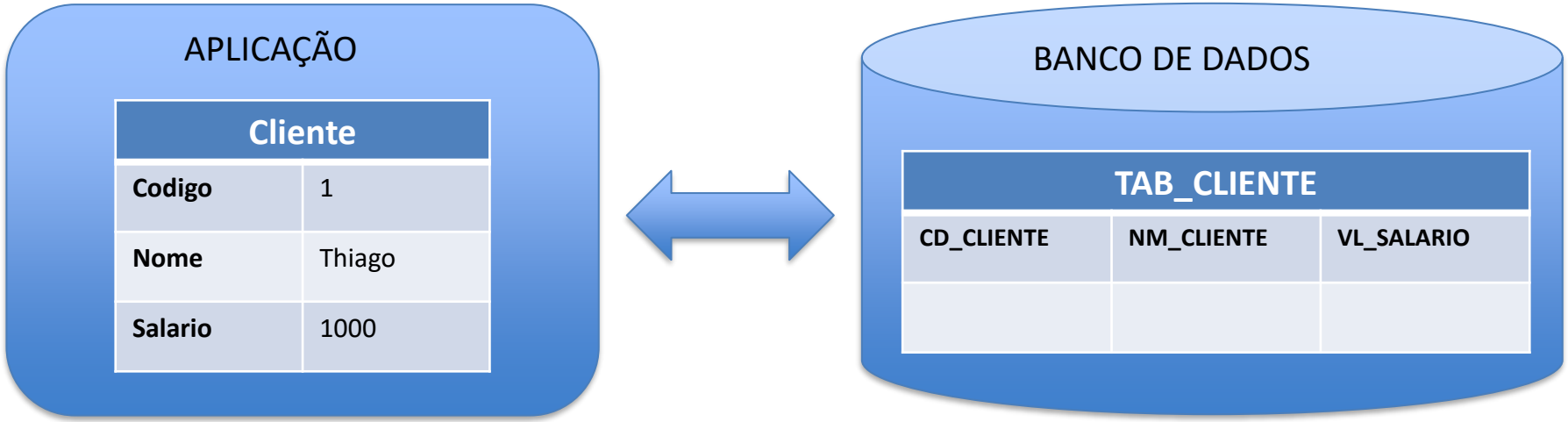
MAPEAMENTO OBJETO-RELACIONAL

PERSISTÊNCIA EM UMA APLICAÇÃO




```
String sql = “INSERT INTO TAB_CLIENTE (CD_CLIENTE, NM_CLIENTE, VL_SALARIO) VALUES (?, ?, ?)”;

PreparedStatement ps = conn.prepareStatement(sql);

ps.setInt(1, cliente.getCodigo());

ps.setString(2, cliente.getNome());

ps.setFloat(3, cliente.getSalario());

//...
```

MAPEAMENTO OBJETO-RELACIONAL



Mapeamento:
Cliente > TAB_CLIENTE
codigo > CD_CLIENTE
nome > NM_CLIENTE
salario > VL_SALARIO

- É possível **mapear a classe Cliente** para representar a tabela **TAB_CLIENTE** do banco de dados através de **anotações** ou arquivo xml;

ANOTAÇÕES JAVA

ANOTAÇÕES JAVA

- São textos inseridos **diretamente no código fonte,** que **expressam informações complementares** sobre uma classe, seus métodos, atributos e etc..;
- Tais informações podem ser acessadas via **API Reflection,** por elementos fora do código fonte, por exemplo, **APIs** de **persistência**;
- Disponíveis no **Java 5** (JSR-175);
- Pode-se **criar** novas anotações a qualquer momento, sendo um processo bastante simples;
- Alternativa aos **descritores de deployment (XML);**

ANOTAÇÕES JAVA

- Objetos são instanciados a partir de **classes anotadas**;
- Processador **reconhece as anotações** encapsuladas nos objetos;
- Os **resultados** são produzidos a partir das informações contidas nessas anotações.

ANOTAÇÕES JAVA - SINTAXE

- Podem ser inseridos antes da declaração de **pacotes**, **classes**, **interfaces**, **métodos** ou **atributos**;
- Iniciam com um **“@”;**
- Uma anotação tem efeito sobre o **próximo elemento** na sequência de sua declaração;
- **Mais de uma anotação** pode ser aplicada a um mesmo elemento do código (classe, método, propriedade, etc...);
- Podem ter parâmetros na forma (param1=“valor”, param2=“valor”, ...);

```
@Override
@SuppressWarnings("all")
public String toString() {
  return “Marcel”;
}
```

ANOTAÇÕES JAVA NATIVAS

Algumas anotações são **nativas**, isto é, já vem com o **JDK**:

- **@Override** - Indica que o método anotado sobrescreve um método da superclasse;
- **@Deprecated** - Indica que um método está obsoleto e não deve ser utilizado;
- **@SuppressWarnings(tipoAlerta)** - Desativa os alertas onde *tipoAlerta* pode ser “all”, “ cast ”, “null”, etc...;

ANOTAÇÕES JAVA - CRIAÇÃO

- Utiliza a palavra **@interface;**
- **Métodos** definem os parâmetros aceitos pela anotação;
- Parâmetros possuem tipos de dados restritos (**String, Class, Enumeration, Annotation e Arrays desses tipos**);
- Parâmetros **podem** ter **valores default**;

Exemplo Anotação:

```
public @interface Mensagem {
  String texto() default "vazio";
}
```

Exemplo Utilização:

```
@Mensagem(texto="Alo Classe")
public class Teste {
  @Mensagem(texto="Alo Metodo")
  public void teste() { }
}
```

ANOTAÇÕES JAVA – META ANOTAÇÕES

Anotações para criar anotações:

- **@Retention -** Indica por quanto tempo a anotação será mantida:
  - **RetentionPolicy.SOURCE** - Nível código fonte;
  - **RetentionPolicy.CLASS** - Nível compilador;
  - **RetentionPolicy.RUNTIME** - Nível JVM;
- **@Target -** Indica o escopo da anotação:
  - **ElementType.PACKAGE** - Pacote;
  - **ElementType.TYPE** - Classe ou Interface;
  - **ElementType.CONSTRUCTOR** - Método Construtor;
  - **ElementType.FIELD** - Atributo;
  - **ElementType.METHOD** - Método;
  - **ElementType.PARAMETER** - Parâmetro de método;

ANOTAÇÕES JAVA - EXEMPLO

Exemplo Anotação:

```
@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.TYPE, ElementType.METHOD})
public @interface Mensagem {
    String texto() default "vazio";
}
```

Exemplo Utilização:

```
@Mensagem(texto="Alo Classe")
public class Teste {
    @Mensagem(texto="Alo Metodo")
    public void teste() { }
}
```

ACESSANDO AS ANOTAÇÕES – API REFLECTION

- **API Reflection** é utilizado por vários **Frameworks** para recuperar informações de um objeto como **anotações, métodos** e **atributos**, em tempo de execução;

- Recuperar anotação de **classe**:

```
Mensagem m = obj.getClass().getAnnotation(Mensagem.class);
```

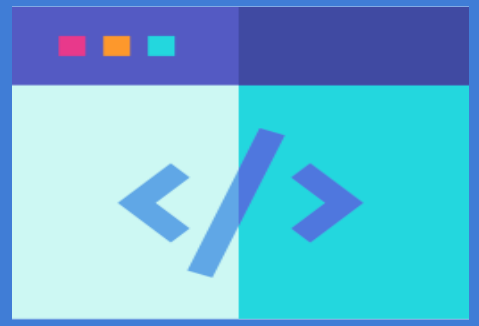
- Recuperar anotação de **método**:

```
Method[] metodos = obj.getClass().getDeclaredMethods();
for (int i = 0; i < metodos.length; i++)
        System.out.println(metodos[i].getAnnotation(Mensagem.class));
```

- Recuperar anotação de **propriedades**:

```
Field[] att = obj.getClass().getDeclaredFields();
for (int i = 0; i < att.length; i++)
        System.out.println(att[i].getAnnotation(Mensagem.class));
```

CODAR!



Escreva uma **classe** que tenha um **método** capaz de receber como parâmetro um objeto e gerar código SQL automaticamente capaz de **selecionar todos os registros de uma tabela**.

Crie uma anotação **@Tabela** que possua um parâmetro **nome** indicando o nome da tabela na qual a classe será mapeada.

Via **API Reflection** gerar automaticamente o código SQL necessário.

Exemplo:

```
@Tabela(nome="TAB_ALUNO")
public class Aluno { }
```

Irá gerar o **SQL** (impresso no console do eclipse):

```
SELECT * FROM TAB_ALUNO
```

Agradecimentos: Prof. Me Gustavo Torres Custódio | Prof. Me. Thiago T. I. Yamamoto

“Se a vida não ficar mais fácil, trate de ficar mais forte.”