# Linguagem de Programação III

# JPA

# JPA

O que é?

O que significa?

Por quê eu preciso saber?

Onde isso altera a minha vida?

# JPA

## O que é?

# JPA

## O que é?

Não é uma tecnologia nova;

É a coleção da solução de algumas tecnologias (Hibernate, EJB, TopLink, JDO)

Especificação padronizada.

# JPA

## O que significa?

# JPA

## O que significa?

**J**ava **P**ersistence **A**PI

# JPA

## Por quê eu preciso saber?

# JPA

## O que significa?

Atualmente, BANCO DE DADOS é uma parte fundamental na maioria dos aplicativos.

# JPA

## Onde isso altera a minha vida?

# JPA

## Onde isso altera a minha vida?

- programar várias classes para controlar as QUERY do BANCO DE DADOS;
- Controle, Manutenção, implementação

```
public class MysqlConnect{
	public static void main(String[] args) {
		System.out.println("MySQL Connect Example.");
		Connection conn = null;
		String url = "jdbc:mysql://localhost:3306/";
		String dbName = "jdbctutorial";
		String driver = "com.mysql.jdbc.Driver";
		String userName = "root";
		String password = "root";
		try {
			Class.forName(driver).newInstance();
			conn = DriverManager.getConnection(url
+dbName,userName,password);
			System.out.println("Connected to the database");
			conn.close();
			System.out.println("Disconnected from database");
		} catch (Exception e) {
			e.printStackTrace();
		}
	}
}
```

# JPA

## **Onde isso altera a minha vida?**

+ tratar as tabelas dos bancos de dados como objetos.

+ Utilizar: herança, polimorfismo, encapsulamento, abstração,…

+ “Esconder” a complexidade.

```
public class JPASetFirstResult {

	public static void main(String[] args) {

		EntityManagerFactory emf=Persistence.createEntityManagerFactory("jpa");
		EntityManager em=emf.createEntityManager();
		try{
			EntityTransaction entr=em.getTransaction();
			entr.begin();
			Query query=em.createQuery("SELECT st FROM Student st");
			query.setFirstResult(2);
			List stuList=query.getResultList();
			Iterator stuIterator=stuList.iterator();
			while(stuIterator.hasNext()){
				Student st=(Student)stuIterator.next();
				System.out.print("sname:"+st.getSname());
				System.out.print("sroll:"+st.getSroll());
				System.out.print("scourse:"+st.getSname());
				System.out.println();
			}
			entr.commit();
		} finally {
		em.close();
		}
	}
}
```

# JPA

É um mapeamento objeto-relacional (ORM) que permite aos aplicativos gerenciar dados entre objetos Java e um banco de dados relacional de uma maneira transparente ao desenvolvedor.

# JPA

Um conjunto de classes Java (**entidades**) espelham o modelo de dados.

A aplicação pode acessar as entidades, como se estivesse acessando diretamente o banco de dados.

# JPA

Alguns benefícios:

JPA possui uma linguagem  própria, similar ao SQL, para queries estática e dinâmicas.

Utilizando Java Persistence Query Language (JPQL), a aplicação continua portável para diferentes SGBD.

# JPA

Mais benefícios:

Evita escrever códigos JDBC/SQL;

JPA apresenta serviços de *data caching* e otimização em performance, transparentes ao desenvolvedor.

# Mas o que é persistência?

# Mas o que é persistência?

Habilidade de preservar os dados do usuário depois que o software/ programa foi fechado/finalizado.

# Mas o que é persistência?

Persistência refere-se ao ato de armazenar **automaticamente** os dados contidos em **objetos Java** em um **banco de dados relacional**.

# Mas o que é persistência?

Em Java, algumas formas de persistência são:

- escrevendo diretamente em arquivos (texto ou binário);
- Banco de dados relacional com JDBC;
- Banco de dados Orientado a Objetos;
- etc.

# Paradigma ORM

# Banco de dados relacional com JDBC

Comunicação através de queries SQL:

- Criar e alterar tabelas;
- Inserir, atualizar e deletar um dado;
- Etc.

Conexão através de um driver;

Propenso a erros;

# Alteração de paradigma

Programas “simples” podem ser escritos baseando-se em lógicas de acesso ao JDBC;

Programas “mais complexos” que apresentam modelos de domínio

- Classes que representam objetos do domínio do problema;
- Utilização de conceitos de OO;
- Lógica de negócio funciona com objetos.

# Paradigma ORM

Mapeamento Objeto Relacional (Object Relational Mapping), ORM, O/RM, O/R mapping, etc…

Uma alternativa para o problema;

# Paradigma ORM

Mapeamento objeto relacional é a persistência automatizada de objetos em um aplicativo Java para as tabelas em um banco de dados relacional, usando metadata que descreve o mapeamento entre os objetos e o banco de dados.

# Paradigma ORM

API para execução de operações CRUD;

Linguagem ou API para a construção de consultas(*queries)* que referenciam uma classe e suas propriedades.

# Por quê usar ORM?

Produtividade;

Performance;

Manutenção;

Independência de SGBD.

# Frameworks ORM

Não são “fáceis” de aprender;

Para utilizar é necessário conhecer SQL e banco de dados relacional;

Problemas em relação aos frameworks geralmente são complexos;

# Entidades
# (Entity)

# Entidades (entity)

Uma entidade representa uma tabela no banco de dados relacional;

Cada instância da entidade, corresponde a uma linha da tabela.

# Entidades (entity)

O estado de persistência é representado através dos **campos** e **propriedades** de persistência.

```java
import javax.persistence.*;

@Entity
@Table(name="usuarios")
public class Usuario{
	@Id
	@GeneratedValue(strategy=GenerationType.IDENTITY)
	private int id;
	private String nome;

	public Usuario(){}

	public Usuario(int id){
		this.id = id;
	}
	public int getId(){
		return id;
	}

	public void setId(int id){
		this.id = id;
	}

	public String getNome(){
		return nome;
	}

	public void setNome(String nome){
		this.nome = nome;
	}
}
```

```java
import javax.persistence.*;

@Entity
@Table(name="usuarios")
public class Usuario{
	@Id
	@GeneratedValue(strategy=GenerationType.IDENTITY)
	private int id;
	private String nome;

	public Usuario(){}

	public Usuario(int id){
		this.id = id;
	}
	public int getId(){
		return id;
	}

	public void setId(int id){
		this.id = id;
	}

	public String getNome(){
		return nome;
	}

	public void setNome(String nome){
		this.nome = nome;
	}
}
```

Annotations
Para informar que é uma classe entidade.

```java
import javax.persistence.*;

@Entity
@Table(name="usuarios")
public class Usuario{
	@Id
	@GeneratedValue(strategy=GenerationType.IDENTITY)
	private int id;
	private String nome;

	public Usuario(){}

	public Usuario(int id){
		this.id = id;
	}
	public int getId(){
		return id;
	}

	public void setId(int id){
		this.id = id;
	}

	public String getNome(){
		return nome;
	}

	public void setNome(String nome){
		this.nome = nome;
	}
}
```

Para informar a tabela que está sendo mapeada. Se não existisse (name="usuarios"), o mapeamento seria de uma tabela Usuário.

# Entidades (entity)

Para que uma classe Java possa ser usada como uma entidade JPA, ela deve obedecer a algumas regras:

a) A classe deve ser anotada com a anotação **javax.persistence.Entity**;

b) A classe deve ter um construtor sem argumento public ou protected. Outros construtores podem ser adicionados, contanto que um sem argumento seja fornecido.

c) A classe não deve ser declarada como final. Nenhum dos métodos ou variáveis (persistentes) de instâncias devem ser declarados como final.

# Entidades (entity)

Para que uma classe Java possa ser usada como uma entidade JPA, ela deve obedecer a algumas regras:

d) Se uma instância de uma entidade for passada por valor como um objeto isolado, por exemplo, através da interface remota de um bean de sessão, a classe deverá implementar a interface Serializable.

e) Entidades podem extender classes entidades e não-entidades. E classes não-entidades podem extender classes entidades.

f) Variáveis persistentes de instâncias devem ser declaradas private, protected ou de pacote e devem ser acessadas somente pelos métodos da classe de entidade. Clientes de uma classe entidade devem acessar seu estado somente por meio de métodos get e set.

# Entidades (entity)

## Campos e propriedades

Os estado de persistência de uma entidade pode ser acessada através das variáveis das instâncias das entidades ou através de propriedades JavaBeans.

# Entidades (entity)

## Campos e propriedades

Os campos e propriedades devem seguir os tipos da linguagem Java:

- Tipo primitivo;
- Java.lang.String;
- Tipos serializável:
    - wrappers;
    - java.math.BigInteger, java.math.BigDecimal;
    - java.util.Date, java.util.Calendar;
    - java.sql.Date, java.sql.Time, java.sql.TimeStamp;
- Outras entidades e/ou coleção de entidades;

# Managing Entities

As entidades são gerenciadas pelas Entity Manager. São representadas pela instâncias de **javax.persistence.EntityManager.**

A instância EntityManager é associado aos contexto de persistência. O contexto de persistência define o escopo onde uma instância de entidade são criadas, persistidas e removidas.

# Contexto de Persistência

O contexto de persistência é um conjunto de instâncias de entidades gerenciadas que existem em armazenamento de dados.

A interface EntityManager define os métodos que serão usados na interação com o contexto de persistência.

# Exemplo


cliente
id INT
nome VARCHAR(45)
email VARCHAR(45)
telefone VARCHAR(45)
endereco VARCHAR(45)
estado VARCHAR(45)
Indexes
PRIMARY