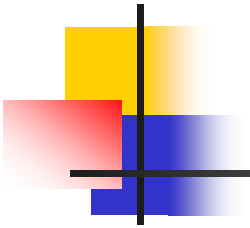
Framework Hibernate/JPA

SSC 124/621 – Análise e Projeto Orientados a Objetos

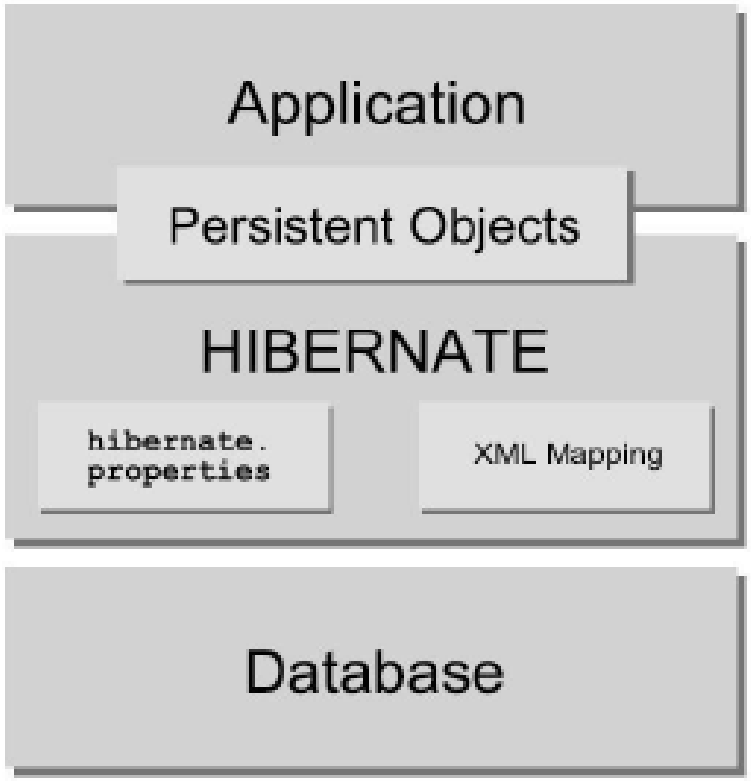
Sofia Costa

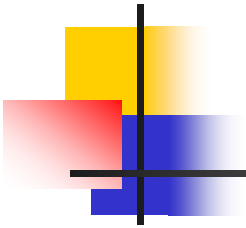


Hibernate

- É um Framework do tipo caixa-branca para persistência de dados.
- É uma ferramenta de mapeamento objeto/relacional (ORM) para Java.
  - O mapeamento é feito mediante uso de arquivos (XML) ou anotações em código Java
- Abstrai por total o banco de dados.

Hibernate - Arquitetura

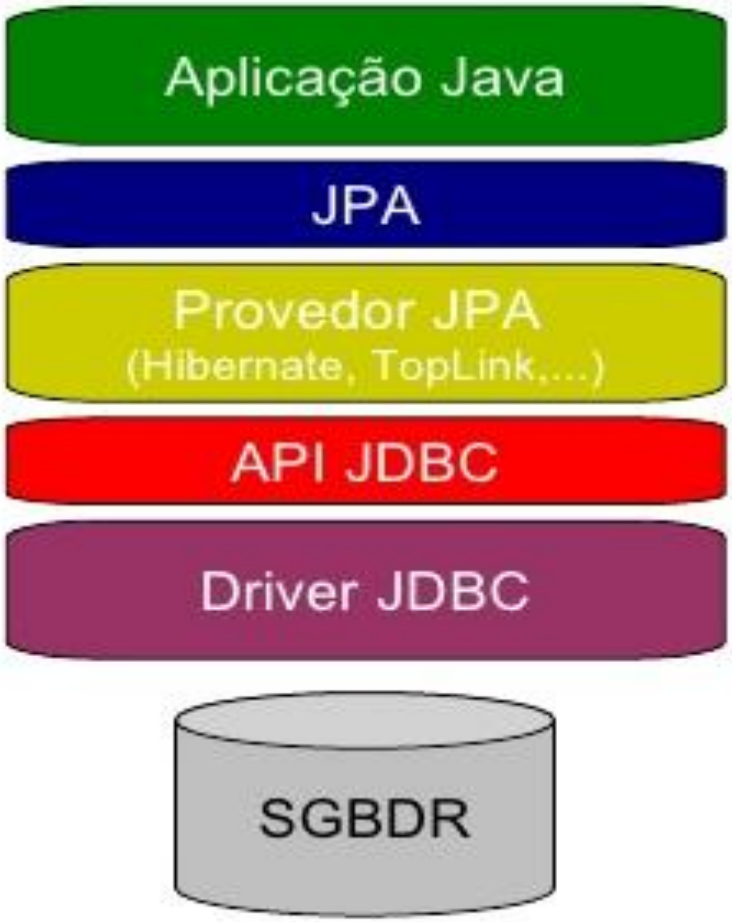


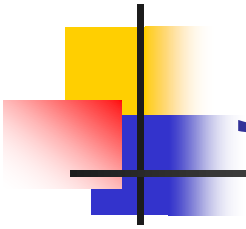


Hibernate

- Inspirou o desenvolvimento da especificação **Java Persistence API (JPA)**
  - Descreve uma interface comum para frameworks ORM
  - Possui diferentes implementações: Hibernate (Jboss), EclipseLink (Eclipse), OpenJPA (Apache)
  - Permite trocar o BD ou o framework de persistência sem alterar o código

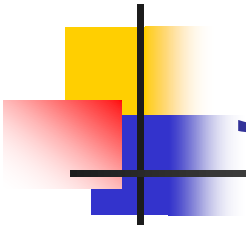
Arquitetura com JPA





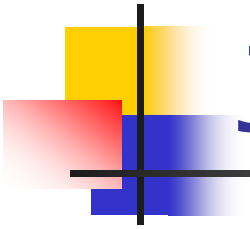
JPA - Características

- Os objetos são **POJOS** (*Plain Old Java Object)*, significando que os objetos possuem design simples que não dependem da herança de interfaces ou classes de frameworks externos. Qualquer objeto com um construtor *default* pode ser feito persistente sem nenhuma alteração em uma linha de código.
- As consultas podem ser realizadas através da **JPQL** (Java Persistence Query Language), uma linguagem de consulta que é derivada do EJB QL e transformada depois para SQL.



JPA - Características

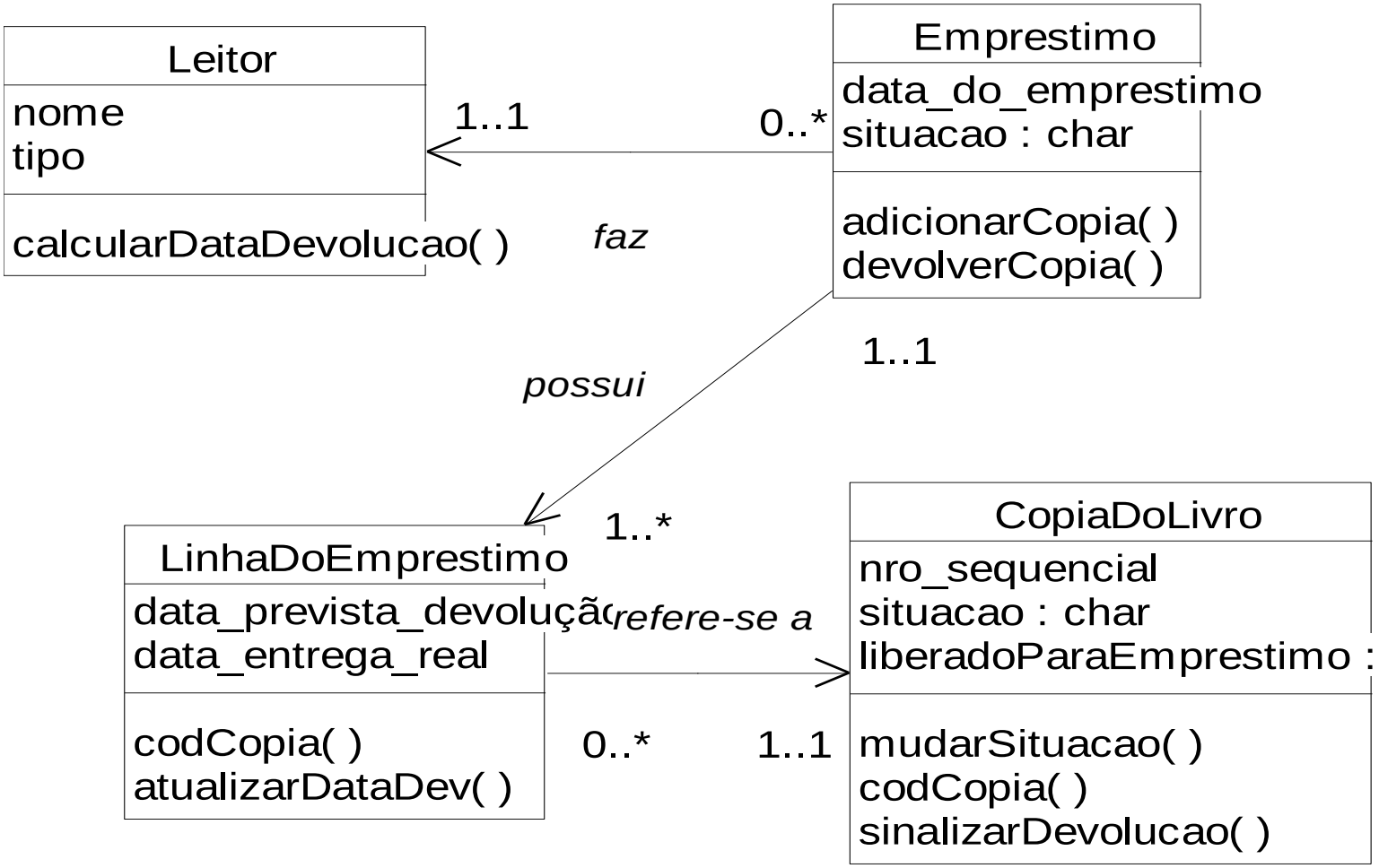
- As configurações são feitas através de anotações, XML ou uma combinação das duas.
- Integração e teste: JPA trabalha fora do servidor de aplicação. Isso permite que a JPA possa ser utilizada sem a existência de um servidor de aplicação. Dessa forma, testes unitários podem ser executados mais facilmente.



JPA

- Cinco passos para utilização do JPA:
  1. Configurar o projeto (JARs do Hibernate, JPA e do JDBC)
  2. Criar o banco de dados onde os objetos vão persistir
  3. Criar a classe (com anotações) cujos objetos vão ser persistidos
  4. Acrescentar as configurações da classe no XML
  5. Criar o objeto cujo estado vai ser persistido.

Exemplo: Empréstimo



Preparação do Modelo

- Classe POJO

```
3  public class Leitor {
4
5      private Integer id;
6      private String nome;
7      private TipoLeitor tipo;
8      private String login;
9      private String senha;
10
11
12     public Integer getId() {
13         return id;
14     }
15
16     public void setId(Integer id) {
17         this.id = id;
18     }
19
20     public String getNome() {
21         return nome;
22     }
```

Mapeamento com JPA

```
3  import javax.persistence.Entity;
10
11 @Entity
12 @SequenceGenerator(name = "SEQ_LEITOR", sequenceName = "SEQ_LEITOR", initialValue = 1)
13 public class Leitor {
14
15     @Id
16     @GeneratedValue(strategy = GenerationType.AUTO, generator = "SEQ_LEITOR")
17     private Integer id;
18     private String nome;
19
20     @Enumerated(EnumType.STRING)
21     private TipoLeitor tipo;
22     private String login;
23     private String senha;
24
25     public Integer getId() {
26         return id;
27     }
28
29     public void setId(Integer id) {
30         this.id = id;
31     }
```

Indica que os objetos dessa classe devem ser persistidos

Geração da chave primária para o MySQL

Declara o atributo que representa a chave primária (obrigatório)

Indica que o banco deve atribuir o valor da chave (opcional)

Indica o tipo do Enumerado a ser mapeado para o banco de dados

Configuração no XML

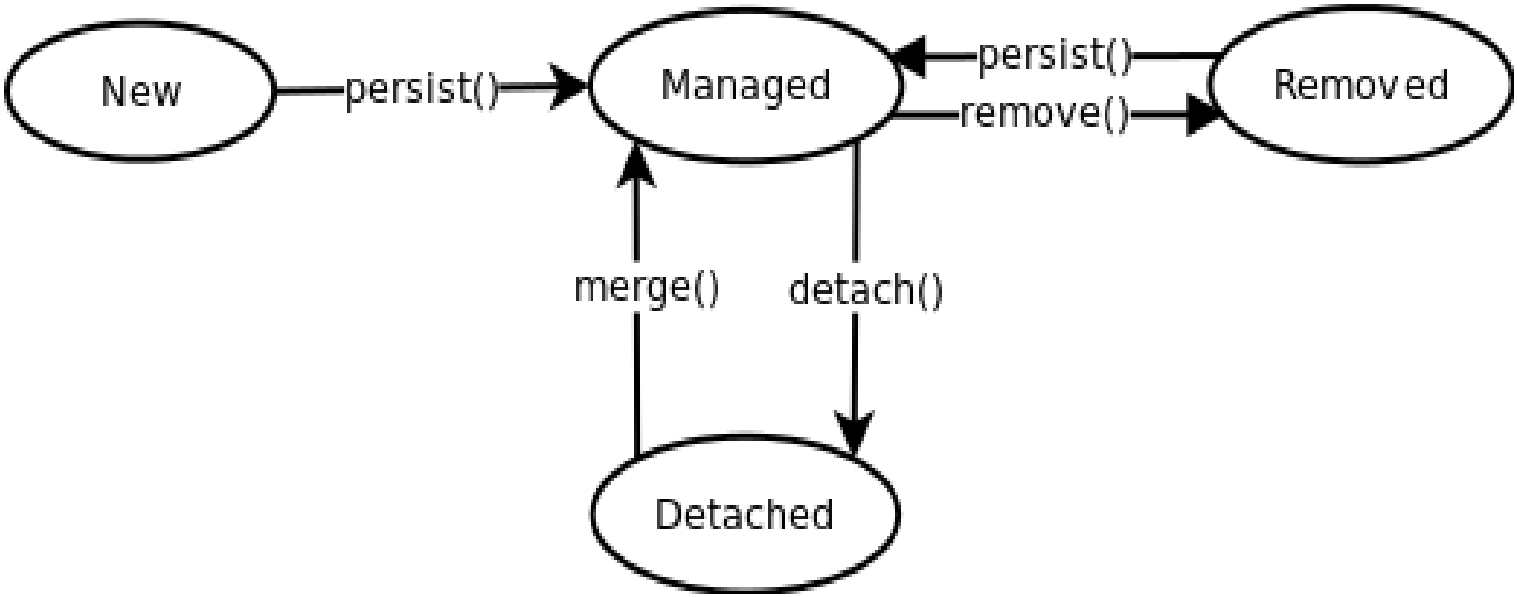
```
1  <persistence xmlns="http://java.sun.com/xml/ns/persistence"
2      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3      xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
4  http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
5      version="2.0">
6
7      <persistence-unit name="biblioteca">
8          <provider>org.hibernate.ejb.HibernatePersistence</provid
9          <class>br.com.caelum.financas.modelo.Leitor</class>
10         <class>br.com.caelum.financas.modelo.Emprestimo</class>
11         <properties>
12             <property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver" />
13             <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost/biblioteca" />
14             <property name="javax.persistence.jdbc.user" value="root" />
15             <property name="                                    ue="" />
16
17             <property name="                                    ate.dialect.MySQL5InnoDBDialect" />
18             <property name="                                    e" />
19             <property name="hibernate.show_sql" value="true" />
20             <property name="hibernate.format_sql" value="true" />
21         </properties>
22     </persistence-unit>
23
24     <persistence-unit name="bib-postgres">
25         <provider>org.hibernate.ejb.HibernatePersistence</provider>
26         <class>br.com.caelum.financas.modelo.Leitor</class>
27         <properties>
28             <property name="javax.persistence.jdbc.driver" value="org.postgresql.Driver" />
```

Indica a base de dados no banco de dados

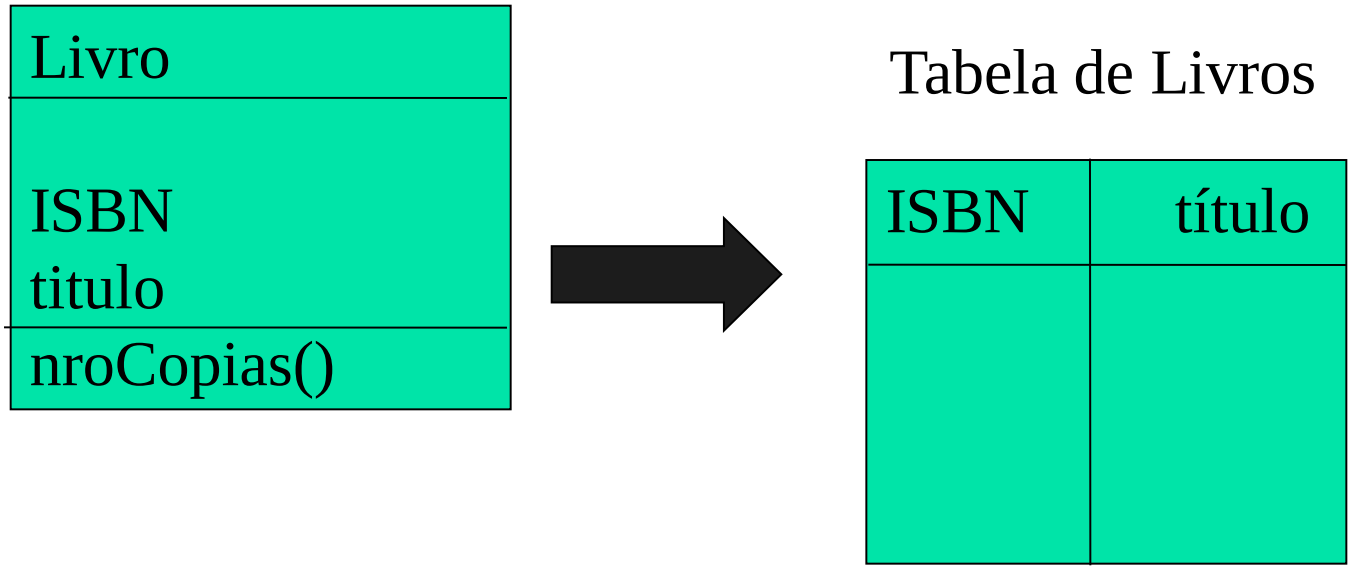
Indica a base de dados no banco de dados

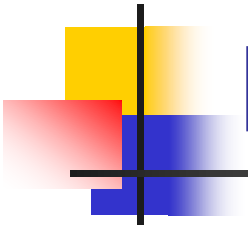
Indica o endereço da base de dados

Estados do JPA



O Padrão Representar Objetos como Tabelas





Problemas

- Para objetos apenas com atributos simples, o mapeamento é direto. Porém, objetos complexos possuem atributos que referenciam outros objetos (às vezes complexos por sua vez).

Como representar relacionamentos em tabelas

- Representação de objetos complexos, que possuem conexões com outros objetos, e não simplesmente atributos primitivos simples.

Como representar relacionamentos com JPA

- Associação um-para-um
  - Deve colocar uma chave estrangeira que seja um OID em uma ou em ambas as tabelas ou
  - Crie uma tabela associativa que registra os OIDs de cada objeto no relacionamento

```
10 @Entity
11 @SequenceGenerator(name = "SEQ_USUARIO", sequenceName = "SEQ_USUARIO", initialValue = 1)
12 public class Usuario {
13
14     @Id
15     @GeneratedValue(strategy = GenerationType.AUTO, generator = "SEQ_USUARIO")
16     private Integer id;
17     private String nome;
18     private String celular;
19
20     @OneToOne
21     private Cartao cartao;
```

Como representar relacionamentos com JPA

- Associação um-para-muitos
  - Deve criar uma tabela associativa que registra os OIDs de cada objeto no relacionamento

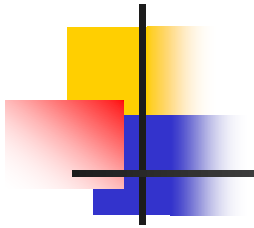
```
15 @Entity
16 @SequenceGenerator(name = "SEQ_LEITOR", sequenceName = "SEQ_LEITOR", initialValue = 1)
17 public class Leitor {
18
19     @Id
20     @GeneratedValue(strategy = GenerationType.AUTO, generator = "SEQ_LEITOR")
21     private Integer id;
22     private String nome;
23
24     @Enumerated(EnumType.STRING)
25     private TipoLeitor tipo;
26     private String login;
27     private String senha;
28
29     @OneToMany(mappedBy="leitor")
30     private List<Emprestimo> emprestimos;
```

Como representar relacionamentos com JPA

- Associação um-para-muitos

```
14 @Entity
15 @SequenceGenerator(name = "SEQ_LINEMP", sequenceName = "SEQ_LINEMP", initialValue = 1)
16 public class LinhaEmprestimo {
17
18⊖        @Id
19         @GeneratedValue(strategy = GenerationType.AUTO, generator = "SEQ_LINEMP")
20         private Integer id;
21
22⊖        @Temporal(TemporalType.DATE)
23         private Calendar data_prevista;
24⊖        @Temporal(TemporalType.DATE)
25         private Calendar data_entrega;
26
27⊖        @ManyToOne
28         private Emprestimo emprestimo;
29
30⊖        @ManyToOne
31         private CopiaLivro livro;
32
```

Como representar relacionamentos em tabelas



- Associação muitos-para-muitos
  - Deve criar uma tabela associativa que registra os OIDs de cada objeto no relacionamento

```
@ManyToMany @JoinTable(name="table1_has_table2")
```

JPA - Consultas

Trabalhando com Parâmetros no JPQL

- Positional Parameter Notation

```
20        Leitor leitor = new Leitor();
21        leitor.setId(6);
22
23        Query query = manager.createQuery("select e from Emprestimo e where e.leitor.id = ?1 "
24                + " and e.situacao = ?2");
25
26        query.setParameter(1, leitor.getId());
27        query.setParameter(2, TipoSituacao.ATIVO);
28
29        List<Emprestimo> emprestimos = query.getResultList();
```

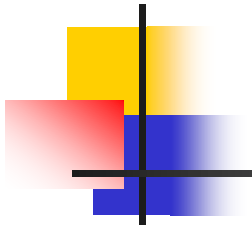
JPA - Consultas

Trabalhando com Parâmetros no JPQL

- Named Parameter Notation

```
20      Leitor leitor = new Leitor();
21      leitor.setId(6);
22
23      Query query = manager.createQuery("select e from Emprestimo e where e.leitor.id = :pLeitor "
24              + " and e.situacao = :pSituacao");
25
26      query.setParameter("pLeitor", leitor.getId());
27      query.setParameter("pSituacao", TipoSituacao.ATIVO);
28
29      List<Emprestimo> emprestimos = query.getResultList();
```

JPA - Consultas



Melhorando a busca

- TypedQuery
- Organizando as consultas com DAOs (*Data Access Object*)
- Alternativa: @NamedQuery