UFABC	Universidade Federal do ABC Disciplina: Programação para Web Turma A2 – noturno 2Q-2018 Atividade 3

Atividade 3: Hibernate

1. Criando um novo Projeto

Vamos utilizar o projeto SistemaBancario da aula passada como base para ilustrar a utilização do Hibernate, e modificar o acesso ao banco de dados.

Crie um novo projeto chamado **SistemaBancarioHibernate** e baixe as *libs* necessárias para o hibernate no repositório do Tidia (**libs → libs.rar**).

Para incluir os pacotes, clique com o botão direito em cima do projeto e, em seguida, properties. Selecione a opção Java Build Path e, na aba libraries, clique em Add External JARs. Inclua todos os JARs do arquivo descompactado.

2. Criando uma entidade

Copie o pacote **br.com.bb.sistemabancario.modelo** do projeto da aula passada e modifique a classe **ContaCorrente** incluindo as seguintes anotações do JPA:
(Ou, crie um novo pacote e nova classe com os nomes acima).

```
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name="contacorrente")
public class ContaCorrente {

	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private Long id;
	@Column(name="numero", nullable=false)
	private String numero;
	@Column(name="agencia", nullable=false)
	private String agencia;
	@Column(name="descricao", nullable=false)
	private String descricao;
	@Column(name="ativa", nullable=false)
	private boolean ativa;
	@Column(name="variacao", nullable=false)
	private int variacao;

	public Long getId() {
		return id;
	}
	public void setId(Long id) {
		this.id = id;
	}
	public String getNumero() {
		return numero;
```

```
	}
	public void setNumero(String numero) {
		this.numero = numero;
	}
	public String getAgencia() {
		return agencia;
	}
	public void setAgencia(String agencia) {
		this.agencia = agencia;
	}
	public String getDescricao() {
		return descricao;
	}
	public void setDescricao(String descricao) {
		this.descricao = descricao;
	}
	public boolean isAtiva() {
		return ativa;
	}
	public void setAtiva(boolean ativa) {
		this.ativa = ativa;
	}
	public int getVariacao() {
		return variacao;
	}
	public void setVariacao(int variacao) {
		this.variacao = variacao;
	}

}
```

A anotação `@Entity` indica que a classe **ContaCorrente** é uma entidade (que será persisitda pelo Hibernate). Toda classe (POJO – *Plain Old Java Object)* que representa uma entidade precisa ter essa anotação.
A anotação `@Table` permite definir a tabela em que a entidade será persistida. É opcional, se não estiver definida, o nome da tabela é o mesmo nome da classe da entidade.
A anotação `@Id` indica que o campo **id** é a chave primária da tabela e `@GeneratedValue` significa que o **id** será gerado pelo JPA, de forma sequencial.
A anotação `@Column` é usada para mapear uma coluna da tabela a um campo da classe. É opcional, se não estiver definida, por padrão o nome do campo é o mesmo nome da coluna. Outros possíveis atributos: **nullable**, **length**, **unique**, etc.
Observe que os imports são do pacote **javax.persistence.**
Veja maiores detalhes sobre as anotações em: https://docs.oracle.com/javaee/7/api/, pacote **javax.persistence** (*Annotation Types*).

3. Cofigurando o Hibernate/JPA (persistence.xml)

Para configurar o Hibernate/JPA é preciso criar um arquivo chamado **persistence.xml.** Este arquivo contém informações sobre sobre a implementação JPA (provedor), as classes (entidades) que serão mapeadas, além de propriedades do banco de dados (driver, URL de conexão, usuário e senha) e propriedades do Hibernate.

Crie uma nova pasta em **src** chamada **META-INF** e dentro desta, crie um arquivo XML (dê o nome de **persistence.xml**) e inclua o conteúdo abaixo:

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd"
	version="2.1">
	<persistence-unit name="sistemabancario">
		<!-- implementacao do JPA (provedor) -->
		<provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
		<!-- entidade mapeada -->
		<class>br.com.bb.sistemabancario.modelo.ContaCorrente</class>
		<properties>
			<!-- conexao -->
			<property name="javax.persistence.jdbc.driver"
				value="org.postgresql.Driver" />
			<property name="javax.persistence.jdbc.url"
				value="jdbc:postgresql://localhost/SistemaBancarioHibernate"
/>
			<property name="javax.persistence.jdbc.user"
				value="postgres" />
			<property name="javax.persistence.jdbc.password"
				value="postgres" />
			<property name="hibernate.dialect"
				value="org.hibernate.dialect.PostgreSQLDialect" />

			<!-- conexao H2 -->
			<!-- <property name="javax.persistence.jdbc.driver"
value="org.h2.Driver"
				/> <property name="javax.persistence.jdbc.url"
value="jdbc:h2:tcp://localhost/~/sistemabancario"
				/> <property name="javax.persistence.jdbc.user"
value="admin" /> <property
				name="javax.persistence.jdbc.password" value="admin" />
<property name="hibernate.dialect"
				value="org.hibernate.dialect.H2Dialect"/> -->

			<!-- imprime as queries SQL no console -->
			<property name="hibernate.show_sql" value="true" />
			<property name="hibernate.format_sql" value="true" />
			<!-- gera as tabelas se necessario -->
			<property name="hibernate.hbm2ddl.auto" value="update" />
		</properties>
	</persistence-unit>
</persistence>
```

Obs.: Se estiver utilizando o H2, comente no código acima as propriedades da conexão do Postgres e decomente a parte do H2.

4. Gerando tabelas no banco de dados

Primeiramente, crie um novo banco de dados chamado **SistemaBancarioHibernate** no banco de dados que você utiliza, Postgres ou H2.

Baseado nas anotações (da entidade) e no arquivo **persistence.xml**, o Hibernate é capaz de gerar automaticamente a tabela no banco de dados. Para isso, crie um pacote **br.com.bb.sistemabancario.dao** e crie uma nova classe: **GeradorTabelas.java**, com o seguinte conteúdo:

```
import javax.persistence.EntityManagerFactory;
```

```
import javax.persistence.Persistence;

public class GeradorTabelas {
	public static void main(String[] args) {
		EntityManagerFactory factory =
Persistence.createEntityManagerFactory("sistemabancario");
		System.out.println("Tabela gerada!");
		factory.close();
	}
}
```

No código acima, a classe **EntityManagerFactory** abre a conexão com o banco de dados, buscando as propriedades definidas no arquivo persistence.xml. No método *createEntityManagerFactory* é passado como parâmetro o nome da unidade de persistência (que foi definida no arquivo **persistence.xml**) e se tabela não existir esta é criada.

Execute o programa: clique direito sobre o nome da classe e selecione ***Run as → Java Application***.

Entre no programa do banco de dados e verifique se a tabela **ContaCorrente** foi gerada.

5. Acessando o banco de dados

Para acessar o banco de dados e realizar operações utilizando o Hibernate/JPA, precisamos alterar os métodos da classe **ContaCorrenteDAO** (visto na aula passada).
Para salvar um objeto **ContaCorrente** no banco de dados, utilizaremos o seguinte código no método **insere**:

```
    public void insere(ContaCorrente cc) {
		EntityManagerFactory factory =
Persistence.createEntityManagerFactory("sistemabancario");
		EntityManager manager = factory.createEntityManager();
		try {
			manager.getTransaction().begin();
			manager.persist(cc);
			manager.getTransaction().commit();
		} finally {
			if (manager.getTransaction().isActive())
				manager.getTransaction().rollback();
		}
		manager.close();
	}
```

A classe **EntityManager** oferece métodos para executar as operações (inserção, alteração, remoção e consulta) no banco de dados e uma instância é obtida através do objeto **EntityManagerFactory.**

Para salvar um objeto **ContaCorrente** no banco de dados, basta chamar o método **persist** do **EntityManager** passando o objeto. Operações que modificam o conteúdo do banco de dados devem ser executados dentro de uma transação. Uma transação é gerenciada pelo objeto **EntityTransaction**, através do **EntityManager.**

6. Operações adicionais

Implemente as outras operações: remoção, alteração e listagem de registros na classe **ContaCorrenteDAO** e teste os métodos na classe **CriaContaCorrente**. Veja os exemplos no slides da aula teórica. Faça uma verificação do BD após cada operação.

Crie um método para listar todas as contas correntes utilizando o trecho de código abaixo:

```
	public List<ContaCorrente> getLista() {

		EntityManagerFactory factory = 
Persistence.createEntityManagerFactory("sistemabancario");
		EntityManager manager = factory.createEntityManager();
		@SuppressWarnings("unchecked")
		List<ContaCorrente> contas = manager.createQuery("select c from 
ContaCorrente c").getResultList();

		manager.close();
		return contas;
	}
```

Para a busca, utilizamos uma consulta JPQL (JPA Query Language), que é uma linguagem que possui uma sintaxe semelhante à linguagem SQL. A principal diferença é que a JPQL trabalha com classes e objetos, enquanto que a SQL trabalha com tabelas, registros e linhas do banco de dados relacional.

Observe que a alteração e remoção devem ser feitas pelo **id** da conta corrente e, portanto, é necessário conhecê-lo de antemão. Para obter o **id**, pode-se inicialmente fazer uma busca para recuperar o registro do BD utilizando o número da conta corrente. Ou seja, basta criar um método em **ContaCorrenteDAO** para buscar um registro pelo número da conta corrente:

```
public ContaCorrente buscaPeloNumero(String numero) {…}
```

Implemente outras duas consultas no banco de dados: 1) busca por número de conta e agência e 2) busca por descrição (utilize o LIKE).

Por fim, modifique o método **insere** (inserção de conta corrente no banco de dados) para manipular tanto com a inserção de um objeto conta corrente quanto como sua alteração. Em outras palavras, os comandos **persist** e **merge** estarão no mesmo método (modifique o nome do método para **save**).