# Lab 5 – JPA e Hibernate

Neste laboratório iremos mudar a forma com que os dados são persistidos. Trocaremos o JDBC pelo JPA/Hibernate. Modificaremos o Projeto *Banco-Modelo* fazendo o mapeamento objeto relacional e modificaremos também o projeto *Banco-Service* fazendo com que este faça todas as operações necessárias utilizando JPA/Hibernate.

### *Exercícios*

**Exercício 1**: Adicionar JPA e Hibernate a aplicação.

**Exercício 2**: Realizar o mapeamento objeto relacional.

**Exercício 3**: Configurar Persistence.xml e realizar primeiro teste utilizando JPA/Hibernate.

**Exercício 4**: Criar um Dao Genérico utilizando EntityManager.

**Exercício 5**: Utilize o CDI para Injetar as classes DAO no Service.

**Exercício 6**: Modifique os outros projetos de modo que utilizem o CDI.

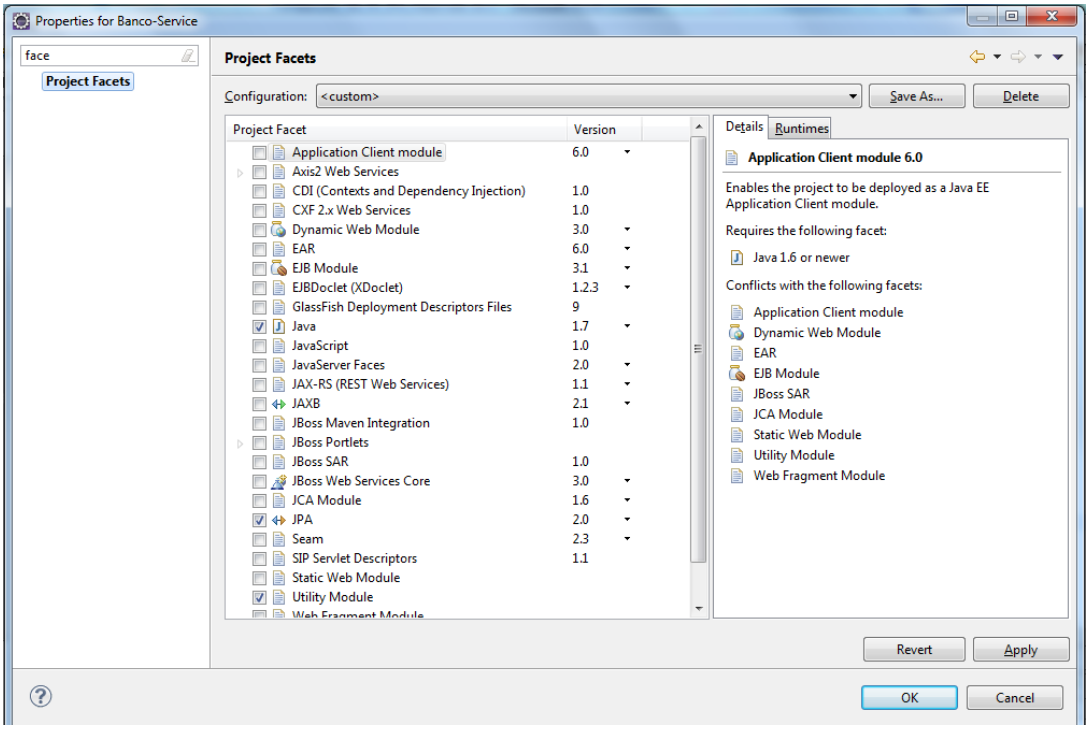
## **Exercício 1 –**Adicionar JPA e Hibernate a aplicação

1. Para adicionar o JPA e o Hibernate, adicione as seguintes dependências no **pom.xml** do projeto *Banco-Modelo.*

```
<!-- Núcleo do Hibernate -->
<dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>4.2.6.Final</version>
</dependency>

<!-- Implementação de EntityManager da JPA -->
<dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-entitymanager</artifactId>
        <version>4.2.6.Final</version>
</dependency>

<dependency>
        <groupId>javax.validation</groupId>
        <artifactId>validation-api</artifactId>
        <version>1.0.0.GA</version>
</dependency>

<dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-log4j12</artifactId>
        <version>1.6.1</version>
</dependency>
```

2. Agora vamos adicionar a configuração do JPA no projeto. Clique com o **botão direito no projeto Banco-Service > Properties.**

3. Agora Selecione a opção Project Facets e configure de acordo com a imagem abaixo.




# **Exercício 2 –**Realizar mapeamento objeto relacional

1. Modifique todas suas classes de negócio (Modelo) de acordo com o código abaixo:

   Classe **Banco**:

```
@Entity
public class Banco extends EntidadeBanco {

	/** Atributo serialVersionUID. */
	private static final long serialVersionUID = -7355477486605864951L;

	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	@Column(name = "id_banco")
	private Long identificador;

	private int numero;

	private String nome;

	@OneToMany(fetch = FetchType.LAZY, mappedBy = "banco", cascade = CascadeType.ALL)
	private Collection<Agencia> agencias;

}
```

   Classe **Agencia**:

```
@Entity
public class Agencia extends EntidadeBanco {

	/** Atributo serialVersionUID. */
	private static final long serialVersionUID = 4090546250667070313L;

	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	@Column(name = "id_agencia")
	private Long identificador;

	private String nome;

	@ManyToOne
	@JoinColumn(name = "id_banco")
	private Banco banco;

	@OneToMany(fetch = FetchType.LAZY, mappedBy = "agencia", cascade = CascadeType.ALL)
	private Collection<Conta> contas;

}
```

Classe **Pessoa**:

```
@Entity
@Inheritance(strategy = InheritanceType.JOINED )
public class Pessoa extends EntidadeBanco {

	private static final long serialVersionUID = -8152860402961368470L;

	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	@Column(name = "id_pessoa")
	private Long identificador;

	private String nome;

	private String telefone;

	private String endereco;

}
```

Classe **Cliente**:

```
@Entity
@PrimaryKeyJoinColumn(name = "id_pessoa")
public class Cliente extends Pessoa implements Comparable<Cliente> {

	private static final long serialVersionUID = 1130736339966591348L;

	private String cpf;

	private String rg;

	@Enumerated(EnumType.STRING)
	private EnumStatus statusCliente;


}
```

Classe **Conta**:

```
@Entity
public class Conta extends EntidadeBanco {

	/** Atributo serialVersionUID. */
	private static final long serialVersionUID = -1922189501184969082L;

	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	@Column(name = "id_conta")
	private Long identificador;

	private Integer numero;

	private Double saldo;

	@Temporal(TemporalType.TIMESTAMP)
	private Date dataAbertura;

	@Transient
	private Collection<Transacao> transacoes;

	@ManyToOne
	@JoinColumn(name = "id_titular")
	private Cliente titular;

	@ManyToOne
	@JoinColumn(name = "id_agencia")
	private Agencia agencia;

	@Enumerated(EnumType.STRING)
	private EnumTipoConta tipoConta;

}
```

Classe **Transacao**:

```
@Entity
public class Transacao extends EntidadeBanco implements Comparable<Transacao> {

	/** Atributo serialVersionUID. */
	private static final long serialVersionUID = -2085220597982994996L;

	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	@Column(name = "id_transacao")
	private Long identificador;

	@Temporal(TemporalType.TIMESTAMP)
	private Date data;

	@ManyToOne
	@JoinColumn(name = "id_conta_debito")
	private Conta contaDebito;

	@ManyToOne
	@JoinColumn(name = "id_conta_credito")
	private Conta contaCredito;

	private double valor;

	private String descricao;

	@Enumerated(EnumType.STRING)
	private EnumTipoTransacao tipoTransacao;

}
```

Crie um enum chamado **EnumStatusCliente** em *Banco-Modelo*:

```
public enum EnumStatusCliente {

	ATIVO("ativo"), INATIVO("inativo");

	private String valor;


	private EnumStatus (String valor) {

		this.valor = valor;
	}

//gere os métodos getter e setter

}
```

Esse enum será usado para realizar uma exclusão lógica do cliente, ou seja, ele apenas será inativado do sistema mas seus dados ainda serão persistidos.

## **Exercício 3 –**Configurar Persistence.xml e realizar primeiro teste utilizando JPA/Hibernate.

1. Modifique o arquivo **META-INF/Persistence.xml** de *Banco-service* de modo que mapeie as classes mapeadas, e configure o acesso ao banco de dados postgres.

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0"
xmlns="http://java.sun.com/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
<persistence-unit name="Banco-ServicePU">

	<provider>org.hibernate.ejb.HibernatePersistence</provider>

	<!-- Mapeamento das classes -->
	<class>threeway.projeto.modelo.Agencia</class>
	<class>threeway.projeto.modelo.Banco</class>
	<class>threeway.projeto.modelo.Cliente</class>
	<class>threeway.projeto.modelo.Conta</class>
	<class>threeway.projeto.modelo.Pessoa</class>
	<class>threeway.projeto.modelo.Transacao</class>

	<!-- Acesso ao banco -->
	<properties>
		<property name="javax.persistence.jdbc.driver" value="org.postgresql.Driver" />
		<property name="javax.persistence.jdbc.url" value="jdbc:postgresql://localhost:5432/threewayJPA" />
		<property name="javax.persistence.jdbc.user" value="postgres" />
		<property name="javax.persistence.jdbc.password" value="123456" />
		<property name="hibernate.hbm2ddl.auto" value="create" />
		<property name="hibernate.show_sql" value="true" />
		<property name="hibernate.format_sql" value="true" />
		<property name="hibernate.dialect" value="org.hibernate.dialect.PostgreSQLDialect" />
	</properties>
</persistence-unit>
</persistence>
```

2. Crie a classe **JPAUtil.java** dentro do pacote threeway/projeto/service/DaoJPA. Essa classe irá criar o Entity Manager.

```
public class JPAUtil {

	private EntityManagerFactory factory;
	static {
		factory = Persistence.createEntityManagerFactory("Banco-ServicePU");
	}


	public static EntityManager getEntityManager() {

		return factory.createEntityManager();
	}
}
```

3. Crie a classe **CriarTabelas.java** e modifique de forma que crie as tabelas do banco e insira um objeto.

```
package threeway.framework.lab05;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

import threeway.projeto.modelo.Banco;

public class CriarTabelas {

	public static void main(String[] args) {

		Banco b = new Banco();

		b.setNome("Teste JPA");

		EntityManager em = JPAUtil.getEntityManager();

		em.getTransaction().begin();

		em.persist(b);

		em.getTransaction().commit();
```

```
                em.close();
        }
}
```

## Exercício 4 –Criar um Dao Genérico utilizando EntityManager.

1. Crie a classe **DaoImpl.java** dentro do pacote threeway/projeto/service/DaoJPA de acordo com o código abaixo:

```
import javax.persistence.EntityManager;

import threeway.projeto.modelo.EntidadeBanco;
import threeway.projeto.service.Dao.Dao;

public abstract class DaoImpl<E extends EntidadeBanco> implements Dao<E>, Serializable{

        /** Atributo serialVersionUID. */
        private static final long serialVersionUID = -6511394566644536118L;

        @Override
        public void alterar(E entidade) {
                this.getSession().update(entidade);
                this.getSession().flush();
        }

        @Override
        public void salvar(E entidade) {
                this.getSession().save(entidade);
                this.getSession().flush();
        }

        @Override
        public void remover(E entidade) {
                //this.getSession().delete(entidade);
                //this.getSession().flush();
        }

        public abstract EntityManager getEntityManager();

        public Session getSession() {
                return (Session) getEntityManager().getDelegate();
        }
}
```

Esses são os métodos para manipularmos as entidades no banco de dados.

Deixaremos o método *remover(E entidade)* comentado para que utilizemos uma exclusão lógica do cliente (perceba que adicionamos um novo atributo em Cliente, *statusCliente,* que será ATIVO ou INATIVO). Isso irá permitir que não percamos dados de possíveis transações que ele tenha feito com outros clientes que ainda estejam ativos. Caso queira tentar excluir o cliente do banco de dados, será gerado um erro dizendo que esse cliente possui relacionamento com chave estrangeira de outra tabela, impedindo a exclusão.

Edite o método *excluir* na classe **ClienteService.java** de acordo com o código abaixo:

```
public void excluir(Cliente cliente) {

        cliente.setStatusCliente(EnumStatus.INATIVO);

        this.getDao().alterar(cliente);

}
```

2. Agora cria as Classes **AgenciaDaoImpl, BancoDaoImpl, ClienteDaoImpl, ContaDaoImpl e TransacaoDaoImpl** que irão herdar a classe **DaoImpl.java**. Modifique as classes conforme o código abaixo:

Classe **AgenciaDaoImpl**:

```
public class AgenciaDaoImpl extends DaoImpl<Agencia> {

	@Inject
	private EntityManager manager;

	@Override
	public Agencia obter(Serializable identificador) {
		Agencia resultado = null;
		resultado = (Agencia) this.getSession().get(Agencia.class, identificador);
		return resultado;
	}
	@SuppressWarnings("unchecked")
	@Override
	public Collection<Agencia> listar() {
		Criteria criteria = this.getSession().createCriteria(Agencia.class);
		return criteria.list();
	}
	@Override
	public EntityManager getEntityManager() {
		return this.manager;
	}
}
```

Classe **BancoDaoImpl**:

```
public class BancoDaoImpl extends DaoImpl<Banco> {
	@Inject
	private EntityManager manager;

	@Override
	public Banco obter(Serializable identificador) {
		Banco resultado = null;
		resultado = (Banco) this.getSession().get(Banco.class, identificador);
		return resultado;
	}
	@SuppressWarnings("unchecked")
	@Override
	public Collection<Banco> listar() {
		Criteria criteria = this.getSession().createCriteria(Banco.class);
		return criteria.list();
	}

	@Override
	public EntityManager getEntityManager() {
	return this.manager;
	}
}
```

Classe **ClienteDaoImpl**:

```
public class ClienteDaoImpl extends DaoImpl<Cliente> {
	@Inject
	private EntityManager manager;
	@Override
	public Cliente obter(Serializable identificador) {
		Cliente resultado = null;
		resultado = (Cliente) this.getSession().get(Cliente.class, identificador);
		return resultado;
	}
	@SuppressWarnings("unchecked")
	@Override
	public Collection<Cliente> listar() {
		Criteria criteria = this.getSession().createCriteria(Cliente.class);

		criteria.add(Restrictions.eq("statusCliente", EnumStatusCliente.ATIVO ));

		return criteria.list();
	}
```

```
	@Override
	public EntityManager getEntityManager() {
		return this.manager;
	}
	public Cliente buscarClientePorCPF(String cpf) {
		Criteria criteria = this.getSession().createCriteria(Cliente.class);
		criteria.add(Restrictions.eq("cpf", cpf));
		criteria.setResultTransformer(Criteria.DISTINCT_ROOT_ENTITY);
		return (Cliente) criteria.uniqueResult();
	}
}
```

Classe **ContaDaoImpl**:

```
public class ContaDaoImpl extends DaoImpl<Conta> {
	@Inject
	private EntityManager manager;
	@Override
	public Conta obter(Serializable identificador) {
		Conta resultado = null;
		resultado = (Conta) this.getSession().get(Conta.class, identificador);
		return resultado;
	}
	@SuppressWarnings("unchecked")
	@Override
	public Collection<Conta> listar() {
		Criteria criteria = this.getSession().createCriteria(Conta.class);
		return criteria.list();
	}
	@Override
	public EntityManager getEntityManager() {
		return this.manager;
	}
}
```

Classe **TransacaoDaoImpl**:

```
public class TransacaoDaoImpl extends DaoImpl<Transacao> {
	@Inject
	private EntityManager manager;
	@Override
	public Transacao obter(Serializable identificador) {
		Transacao resultado = null;
		resultado = (Transacao) this.getSession().get(Transacao.class, identificador);
		return resultado;
	}
	@SuppressWarnings("unchecked")
	@Override
	public Collection<Transacao> listar() {
		Criteria criteria = this.getSession().createCriteria(Transacao.class);
		return criteria.list();
	}
	@Override
	public EntityManager getEntityManager() {
		return this.manager;
	}
	@SuppressWarnings("unchecked")
	public Collection<Transacao> listarTransacoesPorConta(Long identificador) {
		Criteria criteria = this.getSession().createCriteria(Transacao.class);
		criteria.createAlias("contaDebito", "contaDebito", JoinType.LEFT_OUTER_JOIN );
		criteria.createAlias("contaCredito", "contaCredito", JoinType.LEFT_OUTER_JOIN );
		criteria.add(Restrictions.or(Restrictions.eq("contaCredito.identificador", identificador),
				Restrictions.eq("contaDebito.identificador", identificador)));
		return criteria.list();
	}
}
```

As classes acima fazem a busca no banco utilizando criteria.

3. Modifique a classe **JPAUtil.java** de modo que ela seja o produtor de EntityManager.

```
@ApplicationScoped
public class JPAUtil {

private EntityManagerFactory factory;

public JPAUtil() {

        this.factory = Persistence.createEntityManagerFactory("Banco-ServicePU");
}


@Produces
@RequestScoped
public EntityManager createEntityManager() {

        return factory.createEntityManager();
}

public void closeEntityManager(@Disposes EntityManager manager) {

        manager.close();
}
}
```

Feito isso, a classe **CriarTabelas.java** deverá apenas criar o banco, já que agora JPAUtil irá produzir o EntityManager.

4. Mude todas as classes de Service para que ao invés de chamar o Dao JDBC, seja injetado com CDI o DAOImpl da respectiva classe através da anotação *@Inject*.

Classe **ClienteService**:

```
public class ClienteService implements Serializable{

private static final long serialVersionUID = 6082782208927451337L;

@Inject
private ClienteDaoImpl dao;

}
```

Classe **ContaService**:

```
public class ContaService implements Serializable{

/** Atributo serialVersionUID. */
private static final long serialVersionUID = -1817485252683568346L;

@Inject
private ContaDaoImpl dao;


}
```

Classe **TransacaoServce**:

```
public class TransacaoService implements Serializable{

/** Atributo serialVersionUID. */
private static final long serialVersionUID = 1901649352964084473L;

@Inject
private TransacaoDaoImpl dao;

}
```

5. Modifique as classes **AgenciaService.java e BancoService.java** de acordo com o código abaixo:

## a. AgenciaService.java

```
public class AgenciaService implements Serializable {

/** Atributo serialVersionUID. */
private static final long serialVersionUID = 2646320509948935999L;

@Inject
private AgenciaDaoImpl dao;

private static Agencia agenciaSistema;

public static final Agencia agenciaSistema() {

		return agenciaSistema;
}

@PostConstruct
public void inicializaAgencia() {

		List<Agencia> lista = new ArrayList<Agencia>(dao.listar());

		if (lista != null && lista.size() > 0) {

			agenciaSistema = lista.get(0);

		} else {

			Banco bancoSistema = BancoService.bancoSistema();

			Agencia agenciaSistema = new Agencia("3way NetWorks!");

			agenciaSistema.setBanco(bancoSistema);

			dao.salvar(agenciaSistema);

			inicializaAgencia();

		}

		}
}
```

## b. BancoService.java

```
public class BancoService implements Serializable {

/** Atributo serialVersionUID. */
private static final long serialVersionUID = 8507462341844071183L;

@Inject
private BancoDaoImpl dao;

private static Banco bancoSistema;

public static final Banco bancoSistema() {

		return bancoSistema;
}

@PostConstruct
public void inicializaBanco() {

		List<Banco> lista = new ArrayList<Banco>(dao.listar());

		if (lista != null && lista.size() > 0) {

			bancoSistema = lista.get(0);

		} else {

			Banco bancoSistema = new Banco(1);

			bancoSistema.setNome("Banco Java Brasil");

			dao.salvar(bancoSistema);

			inicializaBanco();

		}

}
}
```

As classes que acabamos de modificar inicializam um banco e uma agência.

6. Vamos criar um Interceptor CDI das transações do banco de dados. Crie a interface **Transactional.java** dentro do pacote threeway.projeto.service.DaoJPA.transactions.

```
import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

import javax.interceptor.InterceptorBinding;

@InterceptorBinding
@Retention(RetentionPolicy.RUNTIME)
@Target({ ElementType.TYPE, ElementType.METHOD })
public @interface Transactional {

}
```

Aqui estamos criando um tipo vinculador de interceptador para especificar em quais beans estamos interessados. Nesse caso, os que receberão a anotação *@Transactional.*

7. Agora crie a Classe que fará a interceptação das transações chamada de **TransactionInterceptor.java**. Esse será o nosso interceptador.

```
package threeway.projeto.service.DaoJPA.transactions;

import java.io.Serializable;

import javax.inject.Inject;
import javax.interceptor.AroundInvoke;
import javax.interceptor.Interceptor;
import javax.interceptor.InvocationContext;
import javax.persistence.EntityManager;
import javax.persistence.EntityTransaction;

@Interceptor
@Transactional
public class TransactionInterceptor implements Serializable {

	private static final long serialVersionUID = 1L;

	@Inject
	private EntityManager manager;

	@AroundInvoke
	public Object invoke(InvocationContext context) throws Exception {

		EntityTransaction trx = manager.getTransaction();
		boolean criador = false;
		try {
			if (!trx.isActive()) {
				// truque para fazer rollback no que já passou
				// (senão, um futuro commit, confirmaria até mesmo
				// operações sem transação)
				trx.begin();
				trx.rollback();
				// agora sim inicia a transação
				trx.begin();
				criador = true;
			}
			return context.proceed();
		} catch (Exception e) {
			if (trx != null && criador) {
				trx.rollback();
			}
			throw e;
		} finally {
			if (trx != null && trx.isActive() && criador) {
				trx.commit();
			}
		}
	}
}
```

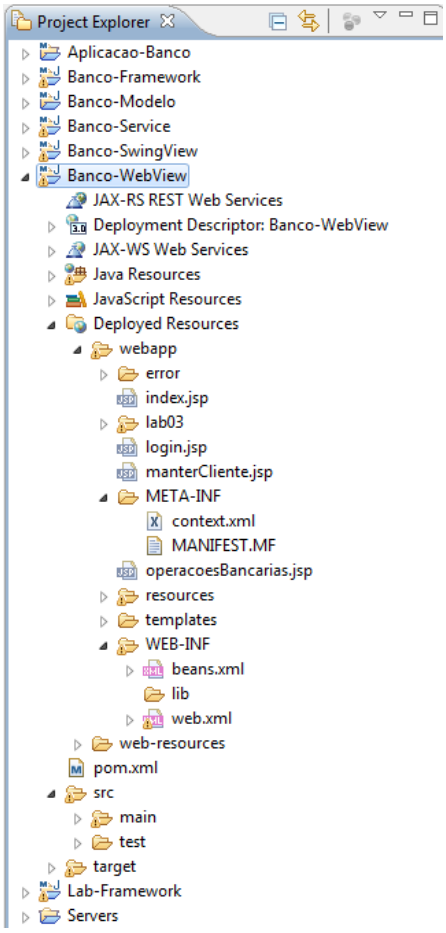
A anotação *@AroundInvoke* indica que o método do interceptador sobrepõe métodos de negócio. Lembrando que somente um método de um interceptador pode receber essa anotação.

8. Agora mapeie esse interceptor no arquivo **beans.xml** do projeto *Banco-Service,* com isso seu interceptador será ativado.

```
<?xml version="1.0"?>
<beans xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://jboss.org/schema/cdi/beans_1_0.xsd">
<interceptors>
        <class>threeway.projeto.service.DaoJPA.transactions.TransactionInterceptor</class>
</interceptors>
</beans>
```

## Exercício 6 –Modifique os outros projetos de modo que utilizem o CDI.

1. Com o projeto *Banco-WebView* adicionado e convertido para projeto Maven. Seu workspace ficará algo parecido com a imagem abaixo.




Arquivo --- pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<parent>
        <artifactId>Aplicacao-Banco</artifactId>
        <groupId>threeway.projeto</groupId>
        <version>0.0.1-SNAPSHOT</version>
        <relativePath>..</relativePath>
</parent>
<artifactId>Banco-WebView</artifactId>
<packaging>war</packaging>
<groupId>threeway.projeto.webView</groupId>

<dependencies>
        <dependency>
```

```
                <groupId>jstl</groupId>
                <artifactId>jstl</artifactId>
                <version>1.2</version>
            </dependency>
            <dependency>
                <groupId>com.sun.faces</groupId>
                <artifactId>jsf-api</artifactId>
                <version>2.2.1</version>
            </dependency>
            <dependency>
                <groupId>com.sun.faces</groupId>
                <artifactId>jsf-impl</artifactId>
                <version>2.2.1</version>
            </dependency>
            <dependency>
                <groupId>threeway.projeto.service</groupId>
                <artifactId>Banco-Service</artifactId>
                <version>0.0.1-SNAPSHOT</version>
            </dependency>
    </dependencies>
    </project>
```

2. Adicione o arquivo **beans.xml** no diretorio WEB-INF e o arquivo **context.xml** no diretorio META-INF:

Arquivo --- context.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Context>
    <Resource name="BeanManager" auth="Container"
        type="javax.enterprise.inject.spi.BeanManager" factory="org.jboss.weld.resources.ManagerObjectFactory" />
</Context>
```

Arquivo --- beans.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://docs.jboss.org/cdi/beans_1_0.xsd">
</beans>
```

3. Modifique a classes **ManterCliente.java** conforme abaixo:

```
public class ManterCliente extends AppController {

    @Inject
    private ClienteService service;

    ...

}
```

4. Modifique a classes **OperacoesBancarias.java** conforme abaixo:

```
@WebServlet("/OperacoesBancarias")
public class OperacoesBancarias extends AppController {
    private static final long serialVersionUID = 1L;

    private Conta conta;

    @Inject
    private ContaService contaService;

    @Inject
    private TransacaoService transacaoService;

    ...

}
```

5. Modifique a classe **Login.java** de acordo com o código abaixo:

```
@WebServlet("/Login")
public class Login extends AppController {
private static final long serialVersionUID = 1L;

@Inject
	private AgenciaService agenciaService;

	...

}
```

1. Modifique o arquivo **web.xml** adicionando a seguinte configuração:

```
<listener>
	<listener-class>org.jboss.weld.environment.servlet.Listener</listener-class>
</listener>
```