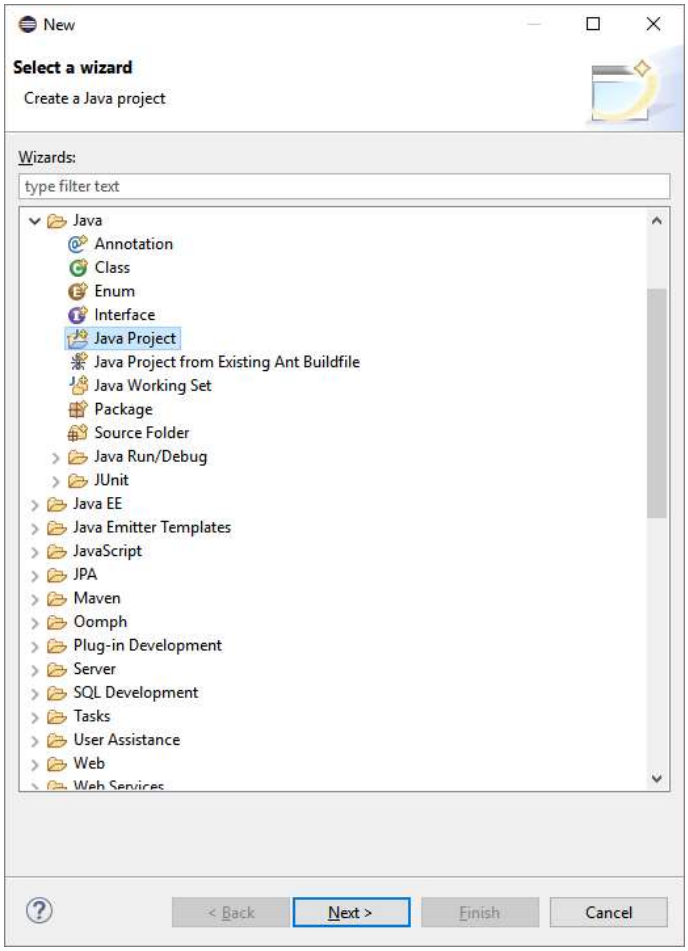
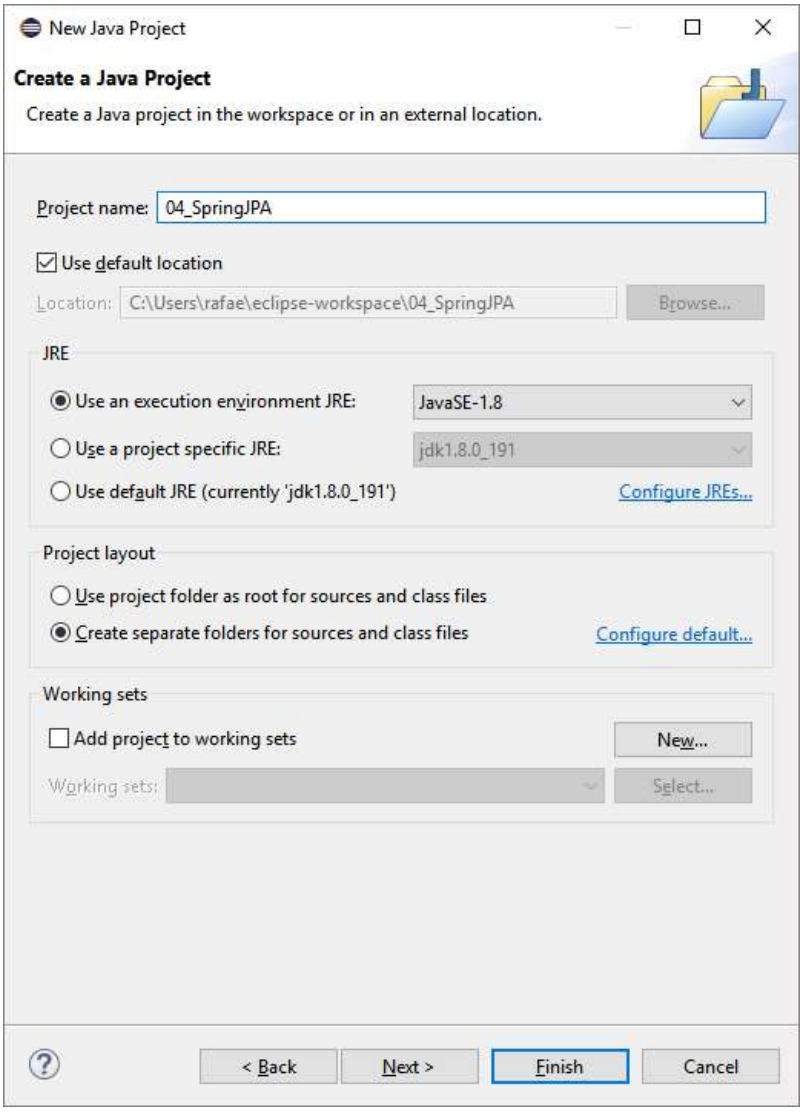
Roteiro para desenvolvimento da aula sobre Configuração do Spring Data JPA

Neste roteiro configuraremos uma aplicação baseada em Spring Data JPA. Os passos são apresentados a seguir:

1. Criar um projeto **Java Project** chamado **04_SpringJPA.**



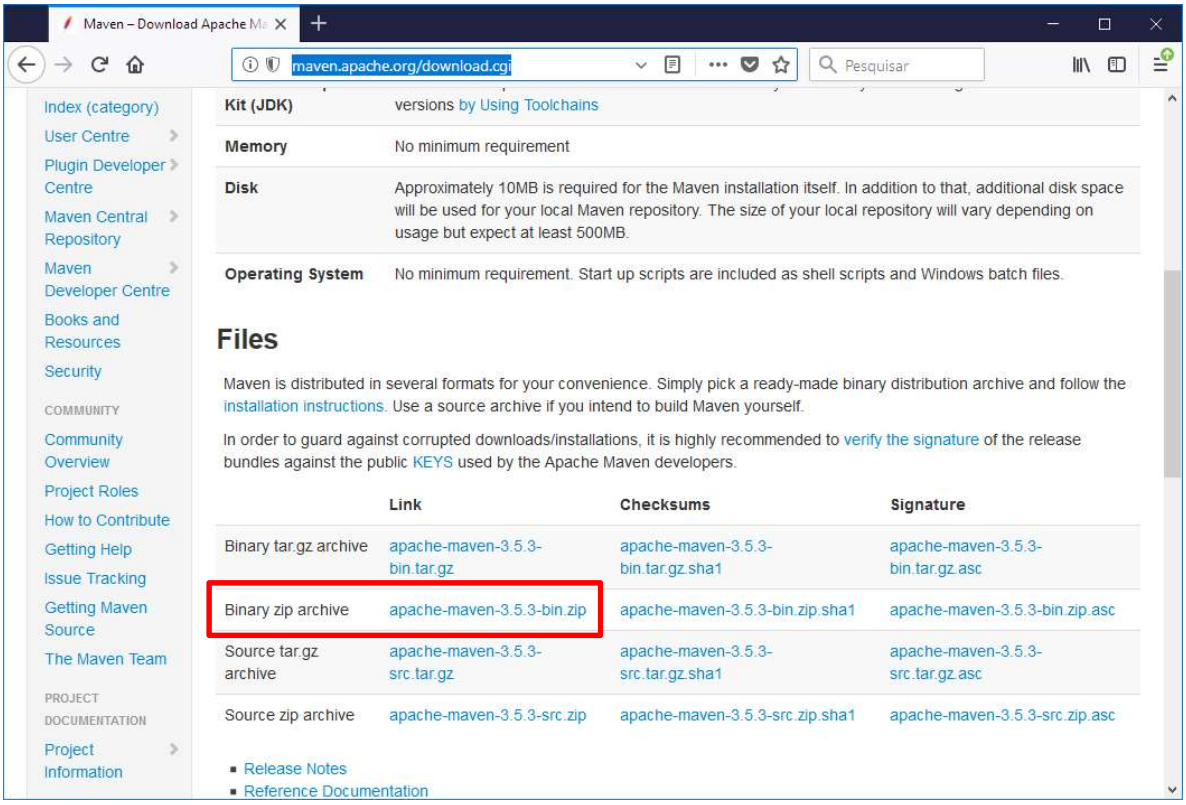


2. Clique em **Finish**.

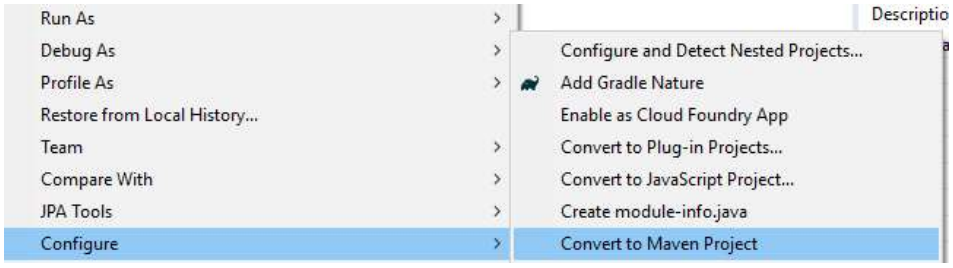
Usando o Maven:

3. Certificar-se de que o maven esteja instalado e incluído no path do seu sistema. O maven pode ser obtido no link: http://maven.apache.org/download.cgi

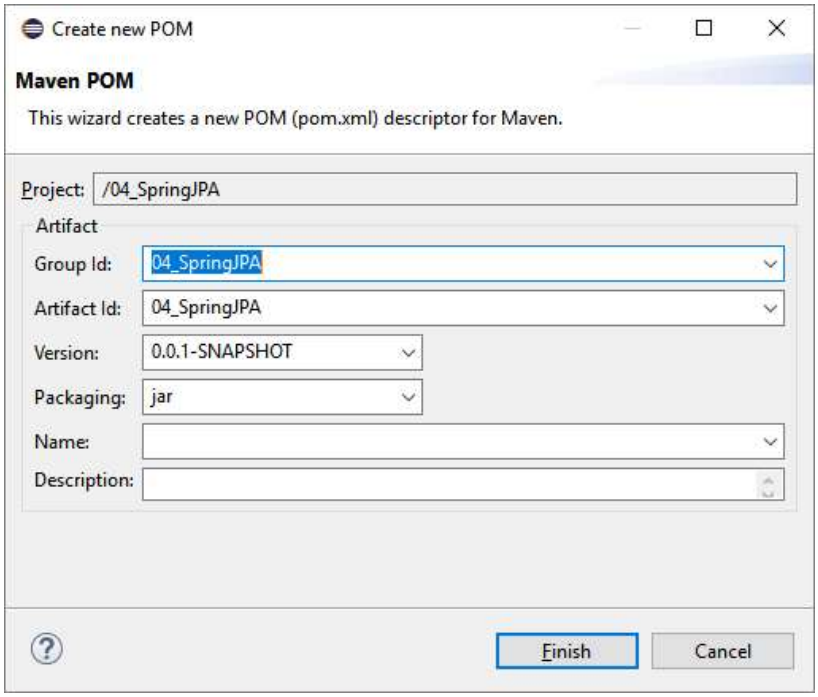
4. Fazer o download e descompactar o arquivo mostrado:



5. Clicar com o botão direito do mouse no projeto, selecione **Configure** e, em seguida, **Convert to Maven Project**.



6. Configurar as opções como mostrado abaixo.



7. Este procedimento criará o arquivo **pom.xml**, onde incluímos as dependências.

8. Acessar o site: mvnrepository.com e buscar as dependências: **hibernate-core** e **mysql-connector-java** (ficar atento às versões, pois tem grande possibilidade de incompatibilidades).

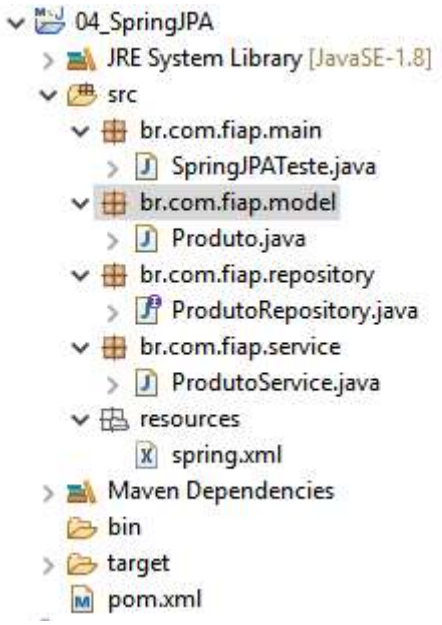
9. Incluir as dependências como mostrado a seguir.

```
<dependencies>
  <!-- Spring -->
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>4.0.0.RELEASE</version>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-orm</artifactId>
    <version>4.0.0.RELEASE</version>
  </dependency>

  <!-- Spring-Data-JPA -->
  <dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-jpa</artifactId>
    <version>1.7.1.RELEASE</version>
  </dependency>

  <!-- Hibernate -->
  <dependency>
```

```
      <groupId>org.hibernate</groupId>
      <artifactId>hibernate-core</artifactId>
      <version>5.2.10.Final</version>
    </dependency>
    <dependency>
      <groupId>org.hibernate</groupId>
      <artifactId>hibernate-entitymanager</artifactId>
      <version>5.2.10.Final</version>
    </dependency>

    <!-- MySQL -->
    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>5.1.44</version>
    </dependency>
  </dependencies>
```

10. Salve o projeto, que o maven irá atualizar as dependências.

11. A estrutura do projeto será semelhante a esta:



12. Criar a entidade **Produto**, incluindo as anotações JPA adequadas para o mapeamento com o banco de dados:

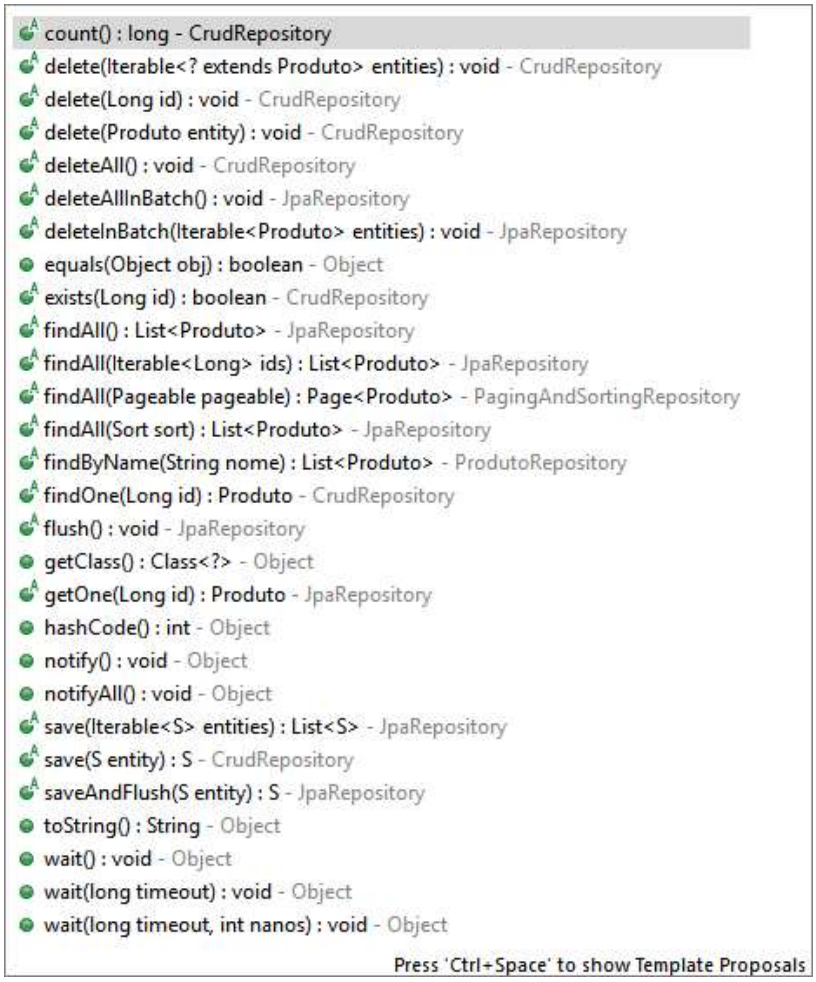
```
package br.com.fiap.model;

import javax.persistence.Entity;
import javax.persistence.Id;

@Entity
public class Produto {
```

```
    @Id
    public Integer id;
    public String nome;

    public Produto() {
  }

  public Produto(Integer id, String nome) {
    this.id = id;
    this.nome = nome;
  }

  @Override
    public String toString() {
        return "Produto (id: " + id + ", nome: " + nome + ")";
    }
}
```

13. Agora criaremos a interface **ProdutoRepository**, que conterá o código do repositório propriamente dito:

```
package br.com.fiap.repository;

import java.util.List;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;

import br.com.fiap.model.Produto;

@Repository
public interface ProdutoRepository extends JpaRepository<Produto, Long> {

    /** Query via JPA **/
    @Query("select p from Produto p where p.nome = :nome")
    public List<Produto> findByName(@Param("nome") String nome);
}
```

14. Repare que a estrutura dos repositórios automaticamente cria o CRUD e extensões, como pode ser verificado ao observar os métodos que constam na classe **ProdutoRepository**:



15. Agora criaremos o **ProdutoService**, que conterá a lógica de negócio da aplicação. Nesse caso, será apenas um *wrapper* para chamar o repositório:

```
package br.com.fiap.service;

import java.util.Collection;
import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import org.springframework.transaction.annotation.Transactional;

import br.com.fiap.model.Produto;
import br.com.fiap.repository.ProdutoRepository;

@Component
public class ProdutoService {
    @Autowired
    private ProdutoRepository produtoRepository;
```

```
    @Transactional
    public void add(Produto produto) {
        produtoRepository.save(produto);
    }

    @Transactional(readOnly=true)
    public List<Produto> findAll() {
        return produtoRepository.findAll();
    }

    @Transactional
    public void addAll(Collection<Produto> produtos) {
        for (Produto produto : produtos) {
            produtoRepository.save(produto);
        }
    }

    @Transactional(readOnly=true)
    public List<Produto> findByName(String nome) {
        return produtoRepository.findByName(nome);
    }
}
```

16. Crie o arquivo **spring.xml** dentro da pasta **resources** com o seguinte conteúdo e configurações de persistência:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:p="http://www.springframework.org/schema/p"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:jpa="http://www.springframework.org/schema/data/jpa"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-3.0.xsd
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx.xsd
        http://www.springframework.org/schema/data/jpa
        http://www.springframework.org/schema/data/jpa/spring-jpa.xsd
        ">

    <context:component-scan base-package="br.com.fiap" />
    <jpa:repositories base-package="br.com.fiap.repository" />
    <context:annotation-config />

    <bean id="dataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSource">
```

```
        <property name="driverClassName" value="com.mysql.jdbc.Driver" />
        <property name="url" value="jdbc:mysql://localhost:3306/dbprodutos"
/>
        <property name="username" value="root" />
        <property name="password" value="hahaha" />
    </bean>

    <bean id="entityManagerFactory"
class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean"
            p:packagesToScan="br.com.fiap.model"
            p:dataSource-ref="dataSource"
            >
        <property name="jpaVendorAdapter">
            <bean
class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter">
                <property name="generateDdl" value="true" />
                <property name="showSql" value="false" />
                <property name="databasePlatform"
value="org.hibernate.dialect.MySQL5InnoDBDialect"/>
            </bean>
        </property>
    </bean>

    <tx:annotation-driven transaction-manager="transactionManager" />
    <bean id="transactionManager"
class="org.springframework.orm.jpa.JpaTransactionManager">
        <property name="entityManagerFactory" ref="entityManagerFactory" />
    </bean>
</beans>
```

17. Para testar a aplicação, utilize e adapte a classe **SpringJPATeste** abaixo:

```
package br.com.fiap.main;

import java.util.Arrays;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import br.com.fiap.model.Produto;
import br.com.fiap.service.ProdutoService;

public class SpringJPATeste {
    public static void main(String[] args) {
        ClassPathXmlApplicationContext ctx = new
ClassPathXmlApplicationContext("resources/spring.xml");
        ProdutoService produtoService = ctx.getBean(ProdutoService.class);

        produtoService.add(new Produto(1, "Laranja"));
        produtoService.add(new Produto(2, "Limao"));
```

```
        System.out.println(produtoService.findAll());

        produtoService.addAll(Arrays.asList(
                new Produto(3, "Pera"),
                new Produto(4, "Morango"),
                new Produto(5, "Maracuja")
                ));

        System.out.println(produtoService.findAll());

        System.out.println(produtoService.findByName("Maracuja"));

        ctx.close();
    }
}
```

18. Caso a aplicação não execute corretamente por conta de erro de caching, será necessário desativar o caching de sessão SSL do MySQL, isso pode ser realizado executando-se a seguinte *query* no MySQL:

```
ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY 'hahaha';
```

Bom trabalho a todos!

Prof. Rafael Matsuyama