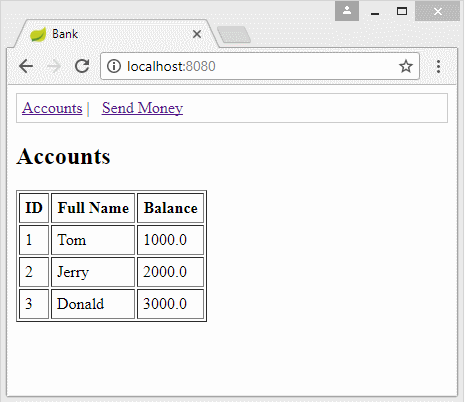
**Лабораторна робота № 26 «Застосування технологій JPA/Hibernate»**

Лабораторна робота заснована на:

* Spring Boot 2.x
* Spring JDBC
* Eclipse 4.7

У цій роботі треба створити проект Spring Boot і працювати з базою даних (Oracle, MySQL, SQL Server, Postgres.) за допомогою Hibernate & Spring Transaction . Питання, які будуть обговорені в цій роботі, включають:

1. Оголосити необхідні бібліотеки для роботи з базою даних.
2. Конфігурувати  Spring Boot, щоб підключитися до бази даних.
3. Маніпулювати з базою даних за допомогою сеансу в Hibernate.
4. Використовувати  Spring Transaction  і пояснити принцип роботи  Spring Transaction .



2- Підготувати базу даних

MySQL / SQL Server

-- Create table

**create** **table** BANK\_ACCOUNT

(

ID BIGINT **not** **null**,

FULL\_NAME VARCHAR(128) **not** **null**,

BALANCE **DOUBLE** **not** **null**

) ;

--

**alter** **table** BANK\_ACCOUNT

**add** **constraint** BANK\_ACCOUNT\_PK **primary** key (ID);

**Insert** **into** Bank\_Account(ID, Full\_Name, Balance) **values** (1, 'Tom', 1000);

**Insert** **into** Bank\_Account(ID, Full\_Name, Balance) **values** (2, 'Jerry', 2000);

**Insert** **into** Bank\_Account(ID, Full\_Name, Balance) **values** (3, 'Donald', 3000);

**commit**;

SQL Server

-- Create table

**create** **table** BANK\_ACCOUNT

(

ID BIGINT **not** **null**,

FULL\_NAME VARCHAR(128) **not** **null**,

BALANCE DOUBLE PRECISION **not** **null**

) ;

--

**alter** **table** BANK\_ACCOUNT

**add** **constraint** BANK\_ACCOUNT\_PK **primary** key (ID);

**Insert** **into** Bank\_Account(ID, Full\_Name, Balance) **values** (1, 'Tom', 1000);

**Insert** **into** Bank\_Account(ID, Full\_Name, Balance) **values** (2, 'Jerry', 2000);

**Insert** **into** Bank\_Account(ID, Full\_Name, Balance) **values** (3, 'Donald', 3000);

Oracle

-- Create table

**create** **table** BANK\_ACCOUNT

(

ID NUMBER(19) **not** **null**,

FULL\_NAME VARCHAR2(128) **not** **null**,

BALANCE NUMBER **not** **null**

) ;

--

**alter** **table** BANK\_ACCOUNT

**add** **constraint** BANK\_ACCOUNT\_PK **primary** key (ID);

**Insert** **into** Bank\_Account(ID, Full\_Name, Balance) **values** (1, 'Tom', 1000);

**Insert** **into** Bank\_Account(ID, Full\_Name, Balance) **values** (2, 'Jerry', 2000);

**Insert** **into** Bank\_Account(ID, Full\_Name, Balance) **values** (3, 'Donald', 3000);

**commit**;

PostGres

**Create** **table** Bank\_Account (

ID Bigint **not** **null**,

Full\_Name Varchar(128) **not** **null**,

Balance real **not** **null**,

**CONSTRAINT** Bank\_Account\_pk **PRIMARY** KEY (ID)

);

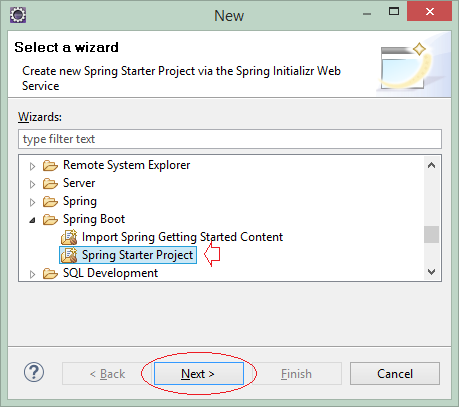
**Insert** **into** Bank\_Account(ID, Full\_Name, Balance) **values** (1, 'Tom', 1000);

**Insert** **into** Bank\_Account(ID, Full\_Name, Balance) **values** (2, 'Jerry', 2000);

**Insert** **into** Bank\_Account(ID, Full\_Name, Balance) **values** (3, 'Donald', 3000);

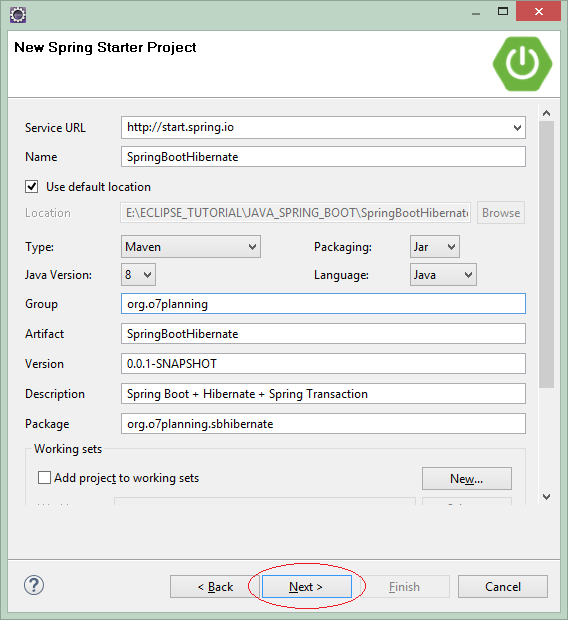
3- Створити проект Spring Boot

На  Eclipse створити проект  Spring Boot .



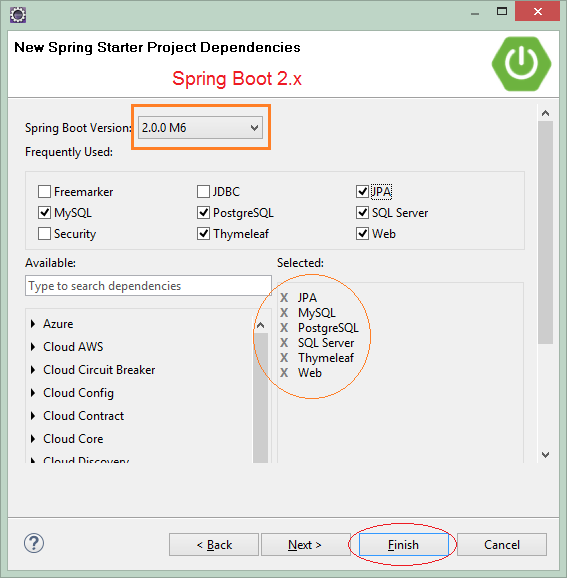
Ввести:

* Назва : SpringBootHibernate
* Група : org.o7planning
* Артефакт : SpringBootHibernate
* Опис : Spring Boot + Hibernate + Spring Transaction
* Пакунок : org.o7planning.sbhibernate



Вибрати технології та бібліотеки, які будуть використовуватися:

* JPA
* MySQL
* PostgrsSQL
* SQL Server
* Інтернет
* Thymeleaf



4- Конфігурація pom.xml

Якщо ви працюєте з базою даних  Oracle , вам потрібно об'явити наступні бібліотеки на  pom.xml :

\* Оракул \*

<**dependencies**>

.....

<**dependency**>

<**groupId**>com.oracle</**groupId**>

<**artifactId**>ojdbc6</**artifactId**>

<**version**>11.2.0.3</**version**>

</**dependency**>

.....

</**dependencies**>

<**repositories**>

....

<!-- Repository for ORACLE JDBC Driver -->

<**repository**>

<**id**>codelds</**id**>

<**url**>https://code.lds.org/nexus/content/groups/main-repo</**url**>

</**repository**>

.....

</**repositories**>

Якщо ви підключаєтеся до бази даних  SQL Service , ви можете використовувати одну з 2 бібліотек  JTDS або  Mssql-Jdbc :

\* SQL Server \*

<**dependencies**>

.....

<**dependency**>

<**groupId**>com.microsoft.sqlserver</**groupId**>

<**artifactId**>mssql-jdbc</**artifactId**>

<**scope**>runtime</**scope**>

</**dependency**>

<**dependency**>

<**groupId**>net.sourceforge.jtds</**groupId**>

<**artifactId**>jtds</**artifactId**>

<**scope**>runtime</**scope**>

</**dependency**>

.....

</**dependencies**>

Повний вміст файлу  pom.xml :

pom.xml

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

<**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**>

<**groupId**>org.o7planning</**groupId**>

<**artifactId**>SpringBootHibernate</**artifactId**>

<**version**>0.0.1-SNAPSHOT</**version**>

<**packaging**>jar</**packaging**>

<**name**>SpringBootHibernate</**name**>

<**description**>Spring Boot + Hibernate + Spring Transaction</**description**>

<**parent**>

<**groupId**>org.springframework.boot</**groupId**>

<**artifactId**>spring-boot-starter-parent</**artifactId**>

<**version**>2.0.0.RELEASE</**version**>

<**relativePath** /> <!-- lookup parent from repository -->

</**parent**>

<**properties**>

<**project.build.sourceEncoding**>UTF-8</**project.build.sourceEncoding**>

<**project.reporting.outputEncoding**>UTF-8</**project.reporting.outputEncoding**>

<**java.version**>1.8</**java.version**>

</**properties**>

<**dependencies**>

<**dependency**>

<**groupId**>org.springframework.boot</**groupId**>

<**artifactId**>spring-boot-starter-data-jpa</**artifactId**>

</**dependency**>

<**dependency**>

<**groupId**>org.springframework.boot</**groupId**>

<**artifactId**>spring-boot-starter-thymeleaf</**artifactId**>

</**dependency**>

<**dependency**>

<**groupId**>org.springframework.boot</**groupId**>

<**artifactId**>spring-boot-starter-web</**artifactId**>

</**dependency**>

<**dependency**>

<**groupId**>mysql</**groupId**>

<**artifactId**>mysql-connector-java</**artifactId**>

<**scope**>runtime</**scope**>

</**dependency**>

<**dependency**>

<**groupId**>org.postgresql</**groupId**>

<**artifactId**>postgresql</**artifactId**>

<**scope**>runtime</**scope**>

</**dependency**>

<!-- SQL Server - Mssql-Jdbc driver -->

<**dependency**>

<**groupId**>com.microsoft.sqlserver</**groupId**>

<**artifactId**>mssql-jdbc</**artifactId**>

<**scope**>runtime</**scope**>

</**dependency**>

<!-- SQL Server - JTDS driver -->

<**dependency**>

<**groupId**>net.sourceforge.jtds</**groupId**>

<**artifactId**>jtds</**artifactId**>

<**scope**>runtime</**scope**>

</**dependency**>

<**dependency**>

<**groupId**>com.oracle</**groupId**>

<**artifactId**>ojdbc6</**artifactId**>

<**version**>11.2.0.3</**version**>

</**dependency**>

<!-- https://mvnrepository.com/artifact/org.threeten/threetenbp -->

<**dependency**>

<**groupId**>org.threeten</**groupId**>

<**artifactId**>threetenbp</**artifactId**>

<**version**>1.3.6</**version**>

</**dependency**>

<**dependency**>

<**groupId**>org.springframework.boot</**groupId**>

<**artifactId**>spring-boot-starter-test</**artifactId**>

<**scope**>test</**scope**>

</**dependency**>

<**dependency**>

<**groupId**>org.springframework.security</**groupId**>

<**artifactId**>spring-security-test</**artifactId**>

<**scope**>test</**scope**>

</**dependency**>

</**dependencies**>

<**repositories**>

<!-- Repository for ORACLE JDBC Driver -->

<**repository**>

<**id**>codelds</**id**>

<**url**>https://code.lds.org/nexus/content/groups/main-repo</**url**>

</**repository**>

</**repositories**>

<**build**>

<**plugins**>

<**plugin**>

<**groupId**>org.springframework.boot</**groupId**>

<**artifactId**>spring-boot-maven-plugin</**artifactId**>

</**plugin**>

</**plugins**>

</**build**>

</**project**>

5- Конфігурація Hibernate

Щоб  Spring міг підключитися до бази даних, вам потрібно налаштувати необхідні параметри у файлі  application.properties .

application.properties (MySQL)

# ===============================

# DATABASE

# ===============================

spring.datasource.driver-class-name=com.mysql.jdbc.Driver

spring.datasource.url=jdbc:mysql://localhost:3306/mydatabase

spring.datasource.username=root

spring.datasource.password=12345

# ===============================

# JPA / HIBERNATE

# ===============================

spring.jpa.show-sql=true

spring.jpa.hibernate.ddl-auto=none

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect

spring.jpa.properties.hibernate.current\_session\_context\_class=org.springframework.orm.hibernate5.SpringSessionContext

application.properites (Sql Server + Mssql-Jdbc)

# ===============================

# DATABASE

# ===============================

spring.datasource.driver-class-name=com.microsoft.sqlserver.jdbc.SQLServerDriver

spring.datasource.url=jdbc:sqlserver://tran-vmware-pc\\SQLEXPRESS:1433;databaseName=bank

spring.datasource.username=sa

spring.datasource.password=12345

# ===============================

# JPA / HIBERNATE

# ===============================

spring.jpa.show-sql=true

spring.jpa.hibernate.ddl-auto=none

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.SQLServerDialect

spring.jpa.properties.hibernate.current\_session\_context\_class=org.springframework.orm.hibernate5.SpringSessionContext

application.properites (Sql Server + JTDS)

# ===============================

# DATABASE

# ===============================

spring.datasource.driver-class-name=net.sourceforge.jtds.jdbc.Driver

spring.datasource.url=jdbc:jtds:sqlserver://tran-vmware-pc:1433/bank;instance=SQLEXPRESS

spring.datasource.username=sa

spring.datasource.password=12345

# ===============================

# JPA / HIBERNATE

# ===============================

spring.jpa.show-sql=true

spring.jpa.hibernate.ddl-auto=none

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.SQLServerDialect

spring.jpa.properties.hibernate.current\_session\_context\_class=org.springframework.orm.hibernate5.SpringSessionContext

application.properties (Oracle)

# ===============================

# DATABASE

# ===============================

spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver

spring.datasource.url=jdbc:oracle:thin:@tran-vmware-pc:1521:db12c

spring.datasource.username=bank

spring.datasource.password=12345

# ===============================

# JPA / HIBERNATE

# ===============================

spring.jpa.show-sql=true

spring.jpa.hibernate.ddl-auto=none

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.Oracle10gDialect

spring.jpa.properties.hibernate.current\_session\_context\_class=org.springframework.orm.hibernate5.SpringSessionContext

application.properties (PostGres)

# ===============================

# DATABASE

# ===============================

spring.datasource.driver-class-name=org.postgresql.Driver

spring.datasource.url=jdbc:postgresql://tran-vmware-pc:5432/bank

spring.datasource.username=postgres

spring.datasource.password=12345

# ===============================

# JPA / HIBERNATE

# ===============================

spring.jpa.show-sql=true

spring.jpa.hibernate.ddl-auto=none

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQL82Dialect

spring.jpa.properties.hibernate.current\_session\_context\_class=org.springframework.orm.hibernate5.SpringSessionContext

# Fix Postgres JPA Error:

# Method org.postgresql.jdbc.PgConnection.createClob() is not yet implemented.

spring.jpa.properties.hibernate.temp.use\_jdbc\_metadata\_defaults=false

Spring Boot  за замовчуванням автоматично конфігурує  JPA , і створює  Spring BEAN, пов’язаний з  JPA , ці автоматичні конфігурації  Spring Boot включають:

1. DataSourceAutoConfiguration
2. DataSourceTransactionManagerAutoConfiguration
3. HibernateJpaAutoConfiguration

Ціль у цьому додатку використовувати  Hibernate , тому нам потрібно відключити автоматичні конфігурації, згадані вище, у  Spring Boot.

\*\* SpringBootHibernateApplication \*\*

**package** org.o7planning.sbhibernate;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.EnableAutoConfiguration;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

**import** org.springframework.boot.autoconfigure.jdbc.DataSourceAutoConfiguration;

**import** org.springframework.boot.autoconfigure.jdbc.DataSourceTransactionManagerAutoConfiguration;

**import** org.springframework.boot.autoconfigure.orm.jpa.HibernateJpaAutoConfiguration;

@SpringBootApplication

@EnableAutoConfiguration(exclude = { //

DataSourceAutoConfiguration.class, //

DataSourceTransactionManagerAutoConfiguration.class, //

HibernateJpaAutoConfiguration.class })

**public** **class** **SpringBootHibernateApplication** {

**public** **static** **void** **main**(String[] args) {

SpringApplication.run(SpringBootHibernateApplication.class, args);

}

.......

}

Потом налаштувати необхідний  Spring BEAN для  Hibernate .

SpringBootHibernateApplication.java

**package** org.o7planning.sbhibernate;

**import** java.util.Properties;

**import** javax.sql.DataSource;

**import** org.hibernate.SessionFactory;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.EnableAutoConfiguration;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

**import** org.springframework.boot.autoconfigure.jdbc.DataSourceAutoConfiguration;

**import** org.springframework.boot.autoconfigure.jdbc.DataSourceTransactionManagerAutoConfiguration;

**import** org.springframework.boot.autoconfigure.orm.jpa.HibernateJpaAutoConfiguration;

**import** org.springframework.context.annotation.Bean;

**import** org.springframework.core.env.Environment;

**import** org.springframework.jdbc.datasource.DriverManagerDataSource;

**import** org.springframework.orm.hibernate5.HibernateTransactionManager;

**import** org.springframework.orm.hibernate5.LocalSessionFactoryBean;

@SpringBootApplication

@EnableAutoConfiguration(exclude = { //

DataSourceAutoConfiguration.class, //

DataSourceTransactionManagerAutoConfiguration.class, //

HibernateJpaAutoConfiguration.class })

**public** **class** **SpringBootHibernateApplication** {

@Autowired

**private** Environment env;

**public** **static** **void** **main**(String[] args) {

SpringApplication.run(SpringBootHibernateApplication.class, args);

}

@Bean(name = "dataSource")

**public** DataSource **getDataSource**() {

DriverManagerDataSource dataSource = **new** **DriverManagerDataSource**();

// See: application.properties

dataSource.setDriverClassName(env.getProperty("spring.datasource.driver-class-name"));

dataSource.setUrl(env.getProperty("spring.datasource.url"));

dataSource.setUsername(env.getProperty("spring.datasource.username"));

dataSource.setPassword(env.getProperty("spring.datasource.password"));

System.out.println("## getDataSource: " + dataSource);

**return** dataSource;

}

@Autowired

@Bean(name = "sessionFactory")

**public** SessionFactory **getSessionFactory**(DataSource dataSource) **throws** Exception {

Properties properties = **new** **Properties**();

// See: application.properties

properties.put("hibernate.dialect", env.getProperty("spring.jpa.properties.hibernate.dialect"));

properties.put("hibernate.show\_sql", env.getProperty("spring.jpa.show-sql"));

properties.put("current\_session\_context\_class", //

env.getProperty("spring.jpa.properties.hibernate.current\_session\_context\_class"));

// Fix Postgres JPA Error:

// Method org.postgresql.jdbc.PgConnection.createClob() is not yet implemented.

// properties.put("hibernate.temp.use\_jdbc\_metadata\_defaults",false);

LocalSessionFactoryBean factoryBean = **new** **LocalSessionFactoryBean**();

// Package contain entity classes

factoryBean.setPackagesToScan(**new** **String**[] { "" });

factoryBean.setDataSource(dataSource);

factoryBean.setHibernateProperties(properties);

factoryBean.afterPropertiesSet();

//

SessionFactory sf = factoryBean.getObject();

System.out.println("## getSessionFactory: " + sf);

**return** sf;

}

@Autowired

@Bean(name = "transactionManager")

**public** HibernateTransactionManager **getTransactionManager**(SessionFactory sessionFactory) {

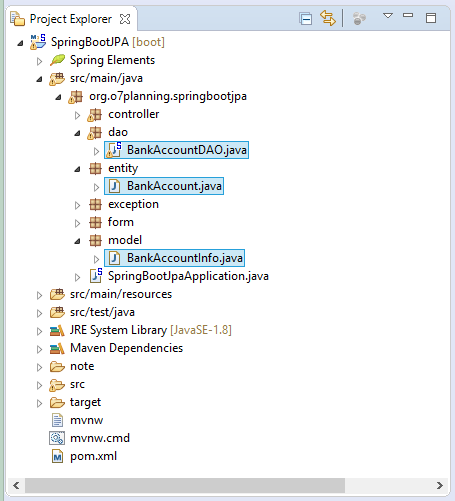
HibernateTransactionManager transactionManager = **new** **HibernateTransactionManager**(sessionFactory);

**return** transactionManager;

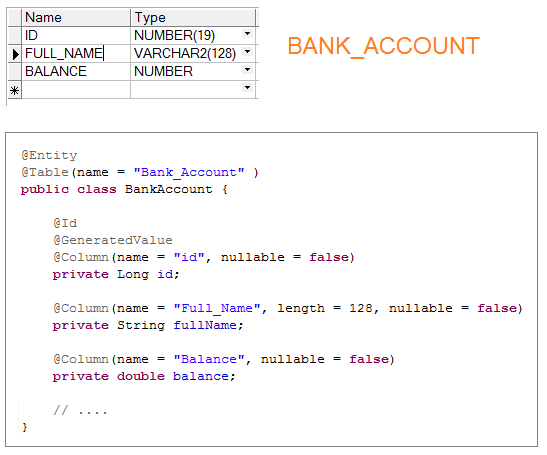
}

}

6- Сутність, Модель, Форма, DAO



У JPA (або  Hibernate ) Entity є представником класу (відповідної) таблиці в базі даних. Поля (поле) в цьому класі будуть відповідати стовбцям у таблиці.  
Ми створили клас  BankAccount,  щоб представити таблицю  BANK\_ACCOUNT в базі даних. Анотація JPA  буде використана для анотацій на полях (поле), щоб описати співставлення (відображення) між полями та стовбцями таблиці. Ці співставлення є 1-1, кожне поле відповідає одній таблиці в таблиці.



BankAccount.java

**package** org.o7planning.sbhibernate.entity;

**import** javax.persistence.Column;

**import** javax.persistence.Entity;

**import** javax.persistence.GeneratedValue;

**import** javax.persistence.Id;

**import** javax.persistence.Table;

@Entity

@Table(name = "Bank\_Account")

**public** **class** **BankAccount** {

@Id

@GeneratedValue

@Column(name = "id", nullable = false)

**private** Long id;

@Column(name = "Full\_Name", length = 128, nullable = false)

**private** String fullName;

@Column(name = "Balance", nullable = false)

**private** double balance;

**public** Long **getId**() {

**return** id;

}

**public** **void** **setId**(Long id) {

this.id = id;

}

**public** String **getFullName**() {

**return** fullName;

}

**public** **void** **setFullName**(String fullName) {

this.fullName = fullName;

}

**public** double **getBalance**() {

**return** balance;

}

**public** **void** **setBalance**(double balance) {

this.balance = balance;

}

}

У той час як клас  Entity  містить дані одного запису (запису) таблиці, а клас  Model  містить дані одного запису запиту (Join з однієї або більше таблиць). Ви використовуєте клас  Model,  коли вас цікавлять кілька столбців однієї або більше таблиць.

BankAccountInfo.java

**package** org.o7planning.sbhibernate.model;

**public** **class** **BankAccountInfo** {

**private** Long id;

**private** String fullName;

**private** double balance;

**public** **BankAccountInfo**() {

}

// Used in Hibernate query.

**public** **BankAccountInfo**(Long id, String fullName, double balance) {

this.id = id;

this.fullName = fullName;

this.balance = balance;

}

**public** Long **getId**() {

**return** id;

}

**public** **void** **setId**(Long id) {

this.id = id;

}

**public** String **getFullName**() {

**return** fullName;

}

**public** **void** **setFullName**(String fullName) {

this.fullName = fullName;

}

**public** double **getBalance**() {

**return** balance;

}

**public** **void** **setBalance**(double balance) {

this.balance = balance;

}

}

BankAccountDAO.java

**package** org.o7planning.sbhibernate.dao;

**import** java.util.List;

**import** org.hibernate.Session;

**import** org.hibernate.SessionFactory;

**import** org.hibernate.query.Query;

**import** org.o7planning.sbhibernate.entity.BankAccount;

**import** org.o7planning.sbhibernate.exception.BankTransactionException;

**import** org.o7planning.sbhibernate.model.BankAccountInfo;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.stereotype.Repository;

**import** org.springframework.transaction.annotation.Propagation;

**import** org.springframework.transaction.annotation.Transactional;

@Repository

@Transactional

**public** **class** **BankAccountDAO** {

@Autowired

**private** SessionFactory sessionFactory;

**public** **BankAccountDAO**() {

}

**public** BankAccount **findById**(Long id) {

Session session = this.sessionFactory.getCurrentSession();

**return** session.get(BankAccount.class, id);

}

**public** List<BankAccountInfo> **listBankAccountInfo**() {

String sql = "Select new " + BankAccountInfo.class.getName() //

+ "(e.id,e.fullName,e.balance) " //

+ " from " + BankAccount.class.getName() + " e ";

Session session = this.sessionFactory.getCurrentSession();

Query<BankAccountInfo> query = session.createQuery(sql, BankAccountInfo.class);

**return** query.getResultList();

}

// MANDATORY: Transaction must be created before.

@Transactional(propagation = Propagation.MANDATORY)

**public** **void** **addAmount**(Long id, double amount) **throws** BankTransactionException {

BankAccount account = this.findById(id);

**if** (account == null) {

**throw** **new** **BankTransactionException**("Account not found " + id);

}

double newBalance = account.getBalance() + amount;

**if** (account.getBalance() + amount < 0) {

**throw** **new** **BankTransactionException**(

"The money in the account '" + id + "' is not enough (" + account.getBalance() + ")");

}

account.setBalance(newBalance);

}

// Do not catch BankTransactionException in this method.

@Transactional(propagation = Propagation.REQUIRES\_NEW, rollbackFor = BankTransactionException.class)

**public** **void** **sendMoney**(Long fromAccountId, Long toAccountId, double amount) **throws** BankTransactionException {

addAmount(toAccountId, amount);

addAmount(fromAccountId, -amount);

}

}

BankTransactionException.java

**package** org.o7planning.sbhibernate.exception;

**public** **class** **BankTransactionException** **extends** **Exception** {

**private** **static** **final** long serialVersionUID = -3128681006635769411L;

**public** **BankTransactionException**(String message) {

super(message);

}

}

SendMoneyForm.java

**package** org.o7planning.sbhibernate.form;

**public** **class** **SendMoneyForm** {

**private** Long fromAccountId;

**private** Long toAccountId;

**private** Double amount;

**public** **SendMoneyForm**() {

}

**public** **SendMoneyForm**(Long fromAccountId, Long toAccountId, Double amount) {

this.fromAccountId = fromAccountId;

this.toAccountId = toAccountId;

this.amount = amount;

}

**public** Long **getFromAccountId**() {

**return** fromAccountId;

}

**public** **void** **setFromAccountId**(Long fromAccountId) {

this.fromAccountId = fromAccountId;

}

**public** Long **getToAccountId**() {

**return** toAccountId;

}

**public** **void** **setToAccountId**(Long toAccountId) {

this.toAccountId = toAccountId;

}

**public** Double **getAmount**() {

**return** amount;

}

**public** **void** **setAmount**(Double amount) {

this.amount = amount;

}

}

@Transactional(propagation = Propagation.REQUIRES\_NEW,

rollbackFor = BankTransactionException.class)

**public** **void** **sendMoney**(Long fromAccountId, Long toAccountId,

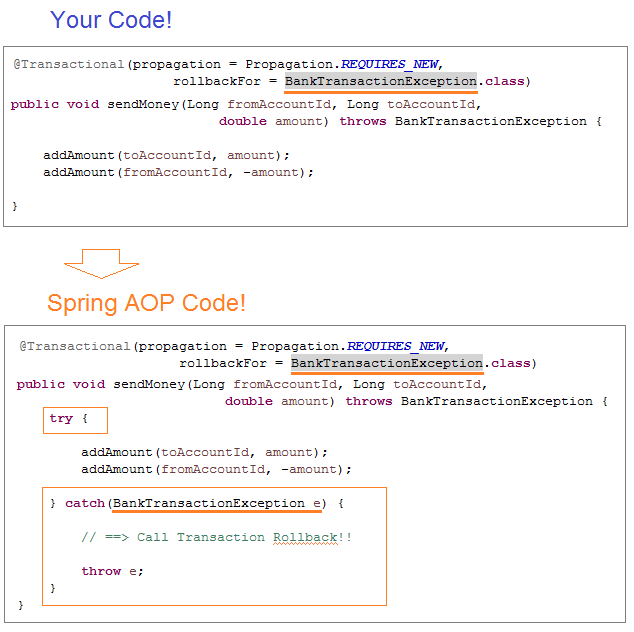
double amount) **throws** BankTransactionException {

addAmount(toAccountId, amount);

addAmount(fromAccountId, -amount);

}

Spring Transaction застосовує  Spring AOP для вашого методу, він схожий на дію зміни коду методу, доданої в код і викликає  транзакції відкоту  , коли це відбувається, потім виключає (повторне виключення) виключення за метод. Все схоже на зображення нижче:



7- Контролер

MainController.java

**package** org.o7planning.sbhibernate.controller;

**import** java.util.List;

**import** org.o7planning.sbhibernate.dao.BankAccountDAO;

**import** org.o7planning.sbhibernate.exception.BankTransactionException;

**import** org.o7planning.sbhibernate.form.SendMoneyForm;

**import** org.o7planning.sbhibernate.model.BankAccountInfo;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.stereotype.Controller;

**import** org.springframework.ui.Model;

**import** org.springframework.web.bind.annotation.RequestMapping;

**import** org.springframework.web.bind.annotation.RequestMethod;

@Controller

**public** **class** **MainController** {

@Autowired

**private** BankAccountDAO bankAccountDAO;

@RequestMapping(value = "/", method = RequestMethod.GET)

**public** String **showBankAccounts**(Model model) {

List<BankAccountInfo> list = bankAccountDAO.listBankAccountInfo();

model.addAttribute("accountInfos", list);

**return** "accountsPage";

}

@RequestMapping(value = "/sendMoney", method = RequestMethod.GET)

**public** String **viewSendMoneyPage**(Model model) {

SendMoneyForm form = **new** **SendMoneyForm**(1L, 2L, 700d);

model.addAttribute("sendMoneyForm", form);

**return** "sendMoneyPage";

}

@RequestMapping(value = "/sendMoney", method = RequestMethod.POST)

**public** String **processSendMoney**(Model model, SendMoneyForm sendMoneyForm) {

System.out.println("Send Money::" + sendMoneyForm.getAmount());

**try** {

bankAccountDAO.sendMoney(sendMoneyForm.getFromAccountId(), //

sendMoneyForm.getToAccountId(), //

sendMoneyForm.getAmount());

} **catch** (BankTransactionException e) {

model.addAttribute("errorMessage", "Error: " + e.getMessage());

**return** "/sendMoneyPage";

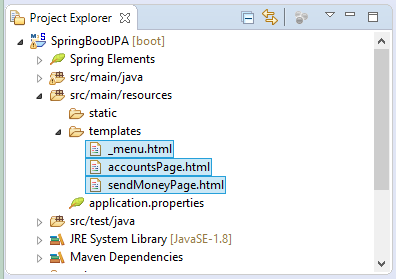
}

**return** "redirect:/";

}

}

8- Шаблон листя чебрецю



\_menu.html

<**div** xmlns:th="http://www.thymeleaf.org" style="border: 1px solid #ccc;padding:5px;margin-bottom:20px;">

<**a** th:href="@{/}">Accounts</**a**>

| &nbsp;

<**a** th:href="@{/sendMoney}">Send Money</**a**>

</**div**>

accountsPage.html

<!DOCTYPE **HTML**>

<**html** xmlns:th="http://www.thymeleaf.org">

<**head**>

<**title**>Bank</**title**>

<**style**>

**th**, **td** {

**padding**: 5px;

}

</**style**>

</**head**>

<**body**>

<!-- Include \_menu.html -->

<**th:block** th:include="/\_menu"></**th:block**>

<**h2**>Accounts</**h2**>

<**table** border="1">

<**tr**>

<**th**>ID</**th**>

<**th**>Full Name</**th**>

<**th**>Balance</**th**>

</**tr**>

<**tr** th:each="accountInfo : ${accountInfos}">

<**td** th:utext="${accountInfo.id}">..</**td**>

<**td** th:utext="${accountInfo.fullName}">..</**td**>

<**td** th:utext="${accountInfo.balance}">..</**td**>

</**tr**>

</**table**>

</**body**>

</**html**>

sendMoneyPage.html

<!DOCTYPE **HTML**>

<**html** xmlns:th="http://www.thymeleaf.org">

<**head**>

<**title**>Bank</**title**>

</**head**>

<**body**>

<!-- Include \_menu.html -->

<**th:block** th:include="/\_menu"></**th:block**>

<**h2**>Send Money</**h2**>

<**ul**>

<**li**>1 - Tom</**li**>

<**li**>2 - Jerry</**li**>

<**li**>3 - Donald</**li**>

</**ul**>

<**div** th:if="${errorMessage!=null}"

style="color:red;font-style:italic" th:utext="${errorMessage}">..</**div**>

<**form** th:action="@{/sendMoney}" th:object="${sendMoneyForm}" method="POST">

<**table**>

<**tr**>

<**td**>From Bank Account Id</**td**>

<**td**><**input** type="text" th:field="\*{fromAccountId}"/></**td**>

</**tr**>

<**tr**>

<**td**>To Bank Account Id</**td**>

<**td**><**input** type="text" th:field="\*{toAccountId}"/></**td**>

</**tr**>

<**tr**>

<**td**>Amount</**td**>

<**td**><**input** type="text" th:field="\*{amount}" /></**td**>

</**tr**>

<**tr**>

<**td**>&nbsp;</**td**>

<**td**><**input** type="submit" value="Send"/></**td**>

</**tr**>

</**table**>

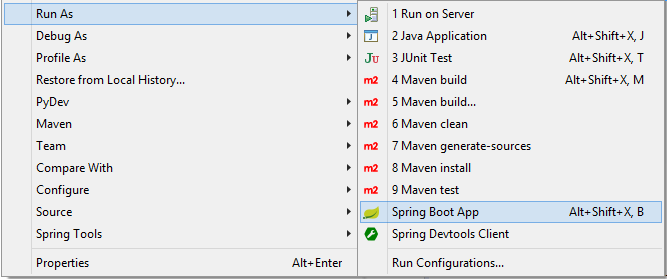
</**form**>

</**body**>

</**html**>

9- Запуск додатків

У  Eclipse запустіть ваше додаток.



* [http://localhost:8080](http://localhost:8080/)

