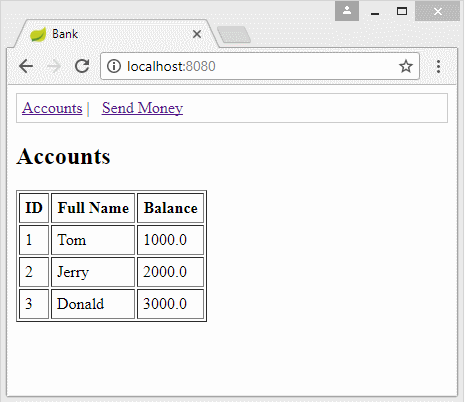
**Лабораторна робота № 25 «Робота в Spring Boot з базами даних. Налаштування Spring JDBC та Spring Transaction»**

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 Spring JDBC і Spring Transaction . Питання, які будуть обговорені в цій статті, 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- Приготувати базу даних

MySQL

-- 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)

);

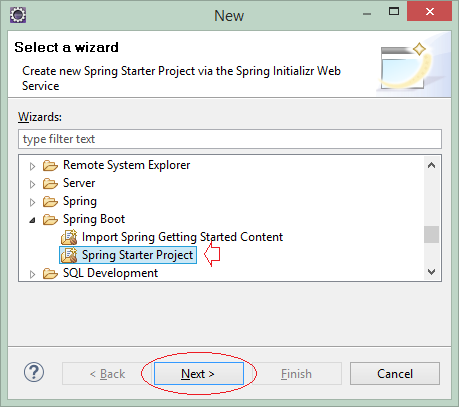
**Insert** **into** Bank\_Account(ID, Full\_Name, Balance) **values** (1, 'Tom', 1000);

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

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

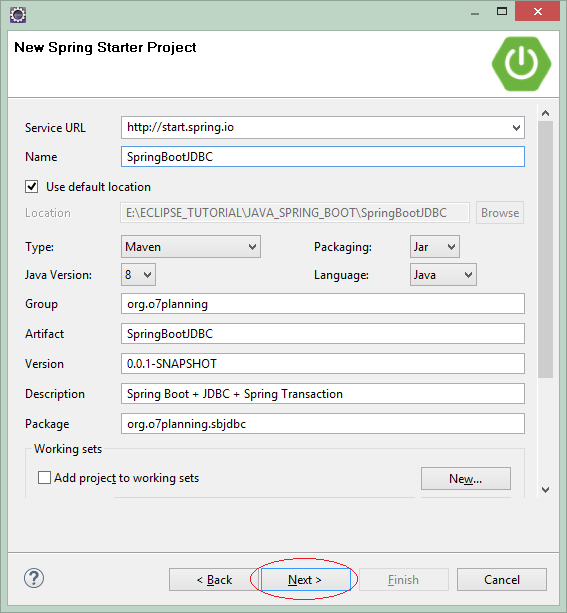
3- Створити проект Spring Boot

На Eclipse створити проект Spring Boot .



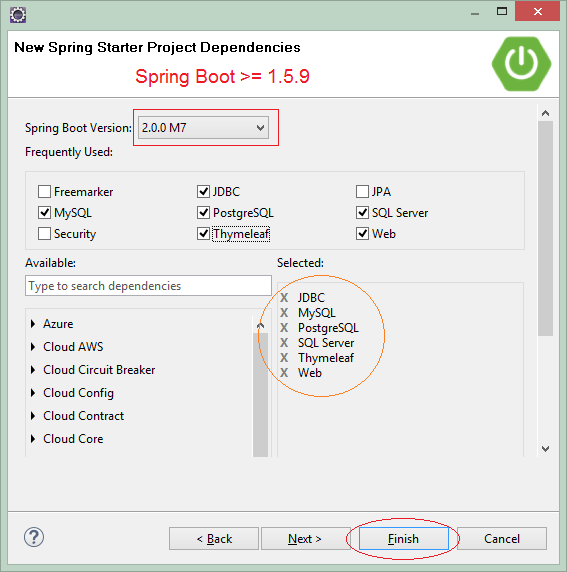
Ввести:

* Назва : SpringBootJDBC
* Група : org.o7planning
* Артефакт : SpringBootJDBC
* Опис : Spring Boot + Spring JDBC + Spring Transaction
* Пакет : org.o7planning.sbjdbc



Вибрати технології та бібліотеки, які будуть використовуватися:

* JDBC
* MySQL
* PostgrsSQL
* SQL Server
* Інтернет
* Thymeleaf



4 - Конфігурація pom.xml

Якщо ви працюєте з базою даних Oracle , вам потрібно оголосити наступні бібліотеки у pom.xml :

\* Oracle \*

<**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**>SpringBootJDBC</**artifactId**>

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

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

<**name**>SpringBootJDBC</**name**>

<**description**>Spring Boot + JDBC + 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-jdbc</**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**>

<!-- Oracle Driver -->

<**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**>

</**dependencies**>

<**build**>

<**plugins**>

<**plugin**>

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

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

</**plugin**>

</**plugins**>

</**build**>

<**repositories**>

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

<**repository**>

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

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

</**repository**>

</**repositories**>

</**project**>

5 - Конфігурація DataSource

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

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=testdb

spring.datasource.username=sa

spring.datasource.password=12345

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/testdb;instance=SQLEXPRESS

spring.datasource.username=sa

spring.datasource.password=12345

application.properties (Oracle)

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

# DATABASE

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

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

spring.datasource.url=jdbc:oracle:thin:@localhost:1521:db12c

spring.datasource.username=Test

spring.datasource.password=12345

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

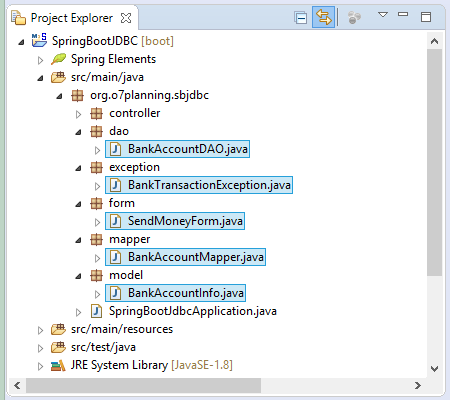
spring.datasource.password=12345

# Fix Postgres JPA Error:

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

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

6- Модель, Mapper, Form, DAO



У Spring , один клас містить дані 1-ої записи, отримані від команди запиту (Query statement), називається класом model. Клас BankAccountInfo є класом моделі.

BankAccountInfo.java

**package** org.o7planning.sbjdbc.model;

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

**private** Long id;

**private** String fullName;

**private** double balance;

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

super();

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;

}

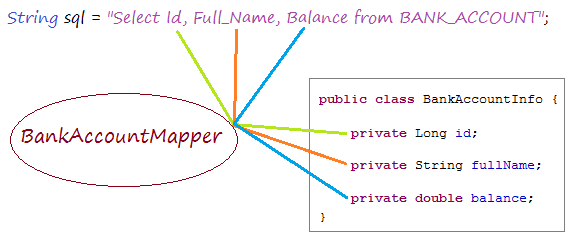
**public** **void** **setBalance**(double balance) {

this.balance = balance;

}

}

Один клас використовується для компонування (відображення) відповідно 1-1 між 1 столбцом у команді запиту та 1 полем (полем) у моделі класу називається класом Mapper . BankAccountMapper є таким класом.



* [*Spring JDBC*](https://betacode.net/10301/spring-jdbc)

BankAccountMapper.java

**package** org.o7planning.sbjdbc.mapper;

**import** java.sql.ResultSet;

**import** java.sql.SQLException;

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

**import** org.springframework.jdbc.core.RowMapper;

**public** **class** **BankAccountMapper** **implements** **RowMapper**<BankAccountInfo> {

**public** **static** **final** String BASE\_SQL //

= "Select ba.Id, ba.Full\_Name, ba.Balance From Bank\_Account ba ";

@Override

**public** BankAccountInfo **mapRow**(ResultSet rs, int rowNum) **throws** SQLException {

Long id = rs.getLong("Id");

String fullName = rs.getString("Full\_Name");

double balance = rs.getDouble("Balance");

**return** **new** **BankAccountInfo**(id, fullName, balance);

}

}

BankAccountDAO.java

**package** org.o7planning.sbjdbc.dao;

**import** java.util.List;

**import** javax.sql.DataSource;

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

**import** org.o7planning.sbjdbc.mapper.BankAccountMapper;

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

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

**import** org.springframework.dao.EmptyResultDataAccessException;

**import** org.springframework.jdbc.core.support.JdbcDaoSupport;

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

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

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

@Repository

@Transactional

**public** **class** **BankAccountDAO** **extends** **JdbcDaoSupport** {

@Autowired

**public** **BankAccountDAO**(DataSource dataSource) {

this.setDataSource(dataSource);

}

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

// Select ba.Id, ba.Full\_Name, ba.Balance From Bank\_Account ba

String sql = BankAccountMapper.BASE\_SQL;

Object[] params = **new** **Object**[] {};

BankAccountMapper mapper = **new** **BankAccountMapper**();

List<BankAccountInfo> list = this.getJdbcTemplate().query(sql, params, mapper);

**return** list;

}

**public** BankAccountInfo **findBankAccount**(Long id) {

// Select ba.Id, ba.Full\_Name, ba.Balance From Bank\_Account ba

// Where ba.Id = ?

String sql = BankAccountMapper.BASE\_SQL + " where ba.Id = ? ";

Object[] params = **new** **Object**[] { id };

BankAccountMapper mapper = **new** **BankAccountMapper**();

**try** {

BankAccountInfo bankAccount = this.getJdbcTemplate().queryForObject(sql, params, mapper);

**return** bankAccount;

} **catch** (EmptyResultDataAccessException e) {

**return** null;

}

}

// MANDATORY: Transaction must be created before.

@Transactional(propagation = Propagation.MANDATORY)

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

BankAccountInfo accountInfo = this.findBankAccount(id);

**if** (accountInfo == null) {

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

}

double newBalance = accountInfo.getBalance() + amount;

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

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

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

}

accountInfo.setBalance(newBalance);

// Update to DB

String sqlUpdate = "Update Bank\_Account set Balance = ? where Id = ?";

this.getJdbcTemplate().update(sqlUpdate, accountInfo.getBalance(), accountInfo.getId());

}

// 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.sbjdbc.exception;

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

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

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

super(message);

}

}

SendMoneyForm.java

**package** org.o7planning.sbjdbc.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;

}

}

У цьому прикладі симулюється транзакцію банку. Аккаунт А відправляє аккаунт На суму 700$. Таким чином будуть створені 2 дії в базі даних:

1. Додавання 700$ на рахунок B.
2. Вичитання 700$ з рахунку А.

Якщо перша дія успішна (Додається 700$ на рахунок B), то друга дія неуспішна з певної причини. В цьому випадку банк буде в побуті. Тому необхідно контролювати транзакції (Transaction), щоб гарантувати, якщо буде неуспішна дія, дані будуть повернуті в початкову стадію (Перед транзакцією). Транзакція вважається успішною, якщо всі дії успішні.

Використовуйте @Transactional(rollbackFor = BankTransactionException.class) для анотації (аннотації) на метод, щоб сказати "Spring Transaction" застосувати AOP для цього методу.

@Transactional(propagation = Propagation.REQUIRES\_NEW,

rollbackFor = BankTransactionException.class)

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

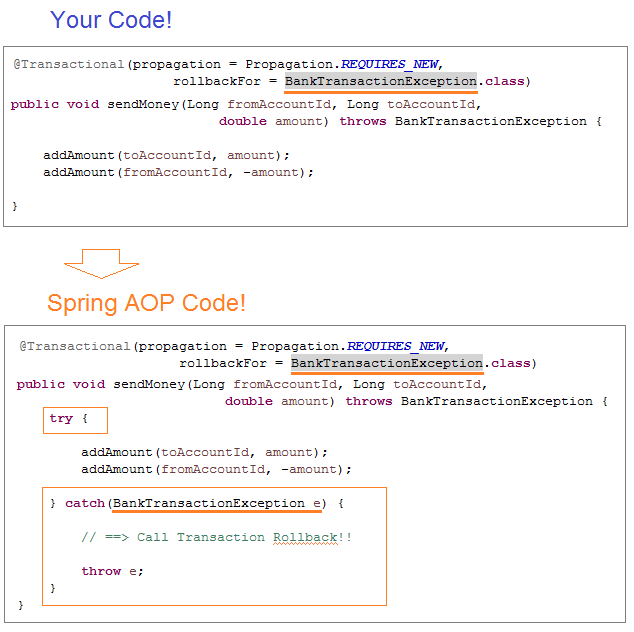
double amount) **throws** BankTransactionException {

addAmount(toAccountId, amount);

addAmount(fromAccountId, -amount);

}

Spring Transaction застосовує Spring AOP для вашого методу, він схожий на дію зміни коду методу, додаючи в код виключення, і викликає транзакцію відкоту , коли це відбувається, потім перебирає (повторно) виключає за допомогою методу. Все схоже на зображення нижче:



7- Контролер

MainController.java

**package** org.o7planning.sbjdbc.controller;

**import** java.util.List;

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

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

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

**import** org.o7planning.sbjdbc.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.getBankAccounts();

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";

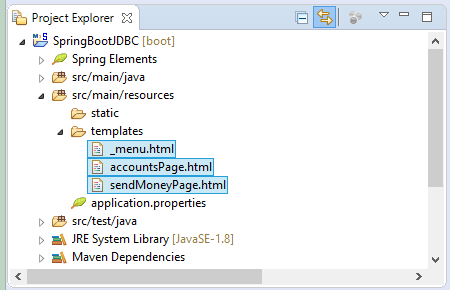
}

**return** "redirect:/";

}

}

8- Шаблон Thymeleaf



\_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 ваш додаток.

