Expense Tracker

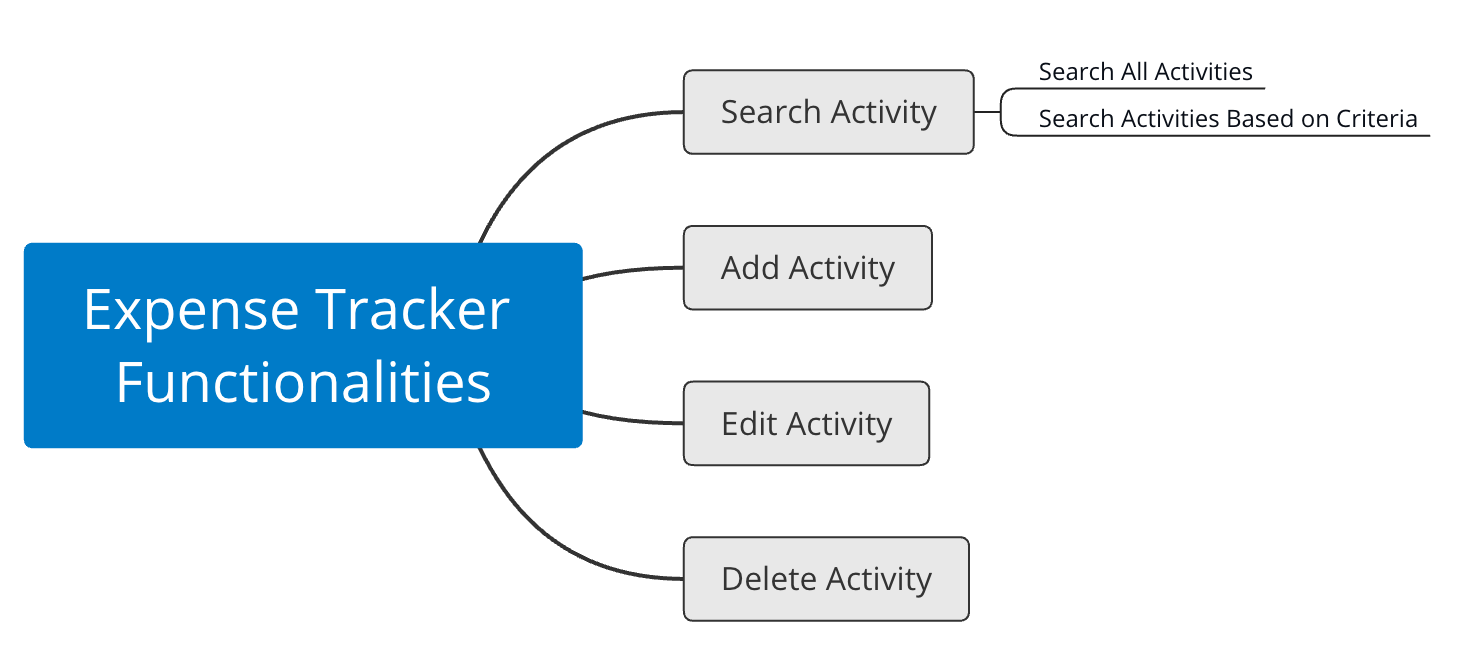
# Goal

* Learn Three-Layered software architecture.
* Learn Apache Commons DBUtils and accomplish CRUD (Create, Read, Update and Delete) operations in Java code.

# Requirements of Expense Tracker

Users: anyone who wants to keep track of his/her spending, earning etc in everyday life.

* The system (Expense Tracker) shall allow user to add new financial activity.
* The system shall allow user to edit an existing activity.
* The system shall allow user to view all existing activities.
* The system shall allow user to search certain activities based on start and end date.
* The system shall allow user to delete an activity.



An activity includes id, name, money, account, createtime and description. Refer to the SQL script of this project.

# Development Environment

* Eclipse
* JDK 11.0 or up
* MySQL 5.7.27
* MySQL Workbench
* Apache commons：
* commons-dbutils-1.7.jar：simplify the JDBC;
* commons-dbcp2-2.7.0.jar：connection pool provided by Apache called DBCP;
* commons-pool2-2.7.0.jar：JAR file needed by DBCP2;
* commons-logging-1.2.jar
* mysql-connector-java-8.0.18.jar：MySQL’s JDBC driver, Java needs this JAR to work with MySQL database. Every vendor provides a driver which implements the APIs specified by Java JDBC.
* Tool class: every project needs a package that stores some utility class. There is one utility class we need here: JDBCUtils.

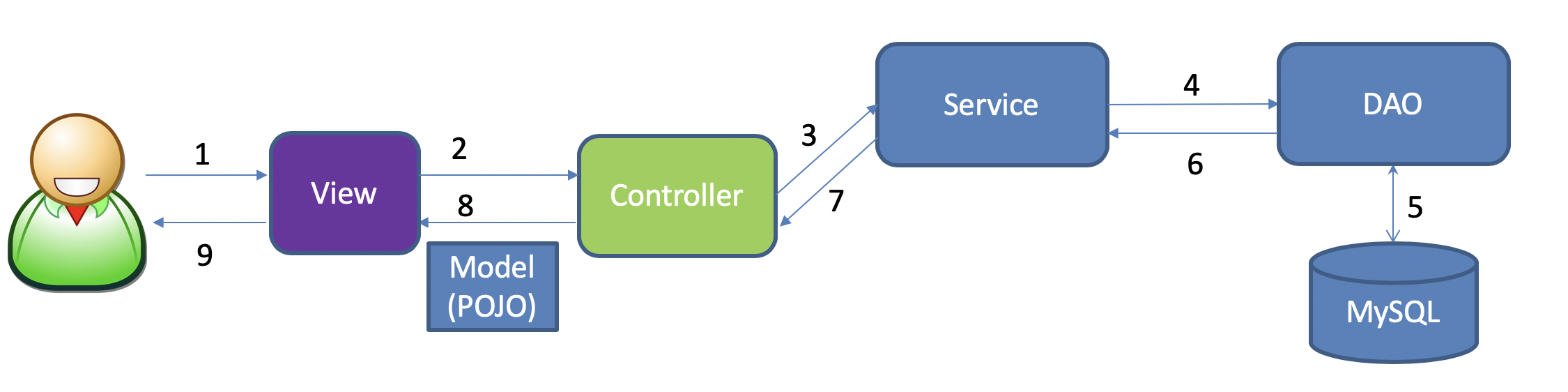
A screenshot of a social media post

Description automatically generated

# Three-Layered Architecture

Three-Layered Architecture is a software architecture that can be used to develop flexible and loosely coupled applications. This architecture results in separating the different aspects of the application (UI logic, business logic and data access), while providing a loose coupling between these elements.

* The **Presentation layer (including View and Controller)** is responsible for rendering the data and in general it generates HTML output that the client's browser can interpret, or in this project, console view.
* The **Logic layer (aka service layer)** is responsible for processing user requests and building an appropriate model and passes it to the view for rendering.
* The **Persistence layer (aka DAO, data access object)** provides CRUD to a data source



Presentation layer

Persistence layer

Logic layer

1. User makes request to View object (e.g. type some strings and press enter);
2. View object (e.g. html or console) passes the request to Controller object;
3. Controller object receives the request and selects the right Service object to handle the request;
4. Service object processes the request, if data is needed, service object passes the request to DAO (Data Access Object) to retrieve the data;
5. DAO is the object that interacts with MySQL, it issues SQL statements to MySQL and receives results from it;
6. DAO returns the results to Service object, Service object may further process the data before return to Controller;
7. Service object returns the result to Controller;
8. Controller prepares the data to make sure the data can be displayed by View object, and then sends to View object;
9. View object renders the results to the user.

# Create Database

In your MySQL Workbench

|  |
| --- |
| CREATE DATABASE expensetracker;  USE expensetracker;  CREATE TABLE activity (  id INT PRIMARY KEY AUTO\_INCREMENT,  name VARCHAR(200),  money DOUBLE,  account VARCHAR(100),/\*where the money comes from or goes to\*/  createtime DATE,  description VARCHAR(1000)  );  INSERT INTO activity(name,money,account,createtime,description) VALUES  ('Dining Spending',247,'ChaseBank Credit Card','2016-03-02','Dutch Burger'),  ('Salary Income',3000,'Cash','2016-03-15','September Salary'),  ('Books Spending',200,'Cash','2016-04-02','Calculus Book'),  ('Dining Spending',325,'Cash','2016-06-18','Eat at TCU Cafe'),  ('Stock Income',8000,'WellsFargo Saving Account','2016-10-28','Apple Stock is great!'),  ('Stock Income',5000,'WellsFargo Saving Account','2016-10-28','Oracle Stock is great!'),  ('Salary Income',3000,'ChaseBank Saving Account','2016-10-28','Oct Salary time!'),  ('Clothes Spending',5000,'Cash','2016-10-28','JCrew again'),  ('Other Spending',20,'Cash','2016-10-29','Lost 20 bucks in library. :('),  ('Gas Spending',1200,'ChaseBank Credit Card','2016-10-29','Gas is so expensive'),  ('Dining Spending',1000,'ChaseBank Debit Card','2016-10-29','Eat eat eat'),  ('Salary Income',3000,'ChaseBank Saving Account','2016-10-30','Nov salary'),  ('Flight Ticket Spending',500,'ChaseBank Debit Card','2016-10-30','Fly to LA'),  ('Salary Income',3000,'ChaseBank Saving Account','2016-10-30','December Salary'); |

# Eclipse Java Project Layout

A JavaBean is just a standard:

* All properties private (use getters/setters)
* A public no-argument constructor
* Implements Serializable.

1. Create a Java project with name ExpenseTracker.
2. Create 7 empty packages.
3. Create a folder called lib and copy the jars into it.

A screenshot of a cell phone

Description automatically generated

1. Add all the jars to build path.

A screenshot of a cell phone

Description automatically generated

1. Create class JDBCUtils under edu.tcu.cs.et.tools

|  |
| --- |
| **package** edu.tcu.cs.et.tools;  **import** java.io.IOException;  **import** java.io.InputStream;  **import** java.util.Properties;  **import** javax.sql.DataSource;  **import** org.apache.commons.dbcp2.BasicDataSource;  **public** **class** JDBCUtils {  **private** **static** BasicDataSource *dataSource* = **new** BasicDataSource();  **private** **static** String *driverClass*;  **private** **static** String *url*;  **private** **static** String *username*;  **private** **static** String *password*;  **private** **static** Integer *maxTotal*;  **private** **static** Integer *maxIdle*;  **private** **static** Integer *minIdle*;  **private** **static** Integer *initialSize*;  // Static block is used for initializing the static variables.  // This block gets executed when the class is loaded in the memory.  **static** {  **try** {  *readProperty*();  *dataSource*.setDriverClassName(*driverClass*);  *dataSource*.setUrl(*url*);  // If you are not using root to log on MySQL, you need to change here.  *dataSource*.setUsername(*username*);  // you need to change this to your DB password  *dataSource*.setPassword(*password*);  // The maximum number of active connections that can be allocated from this pool  // at the same time, or non-positive for no limit.  *dataSource*.setMaxTotal(*maxTotal*);  // The maximum number of connections that can remain idle in the pool, without  // extra ones being released, or negative for no limit.  *dataSource*.setMaxIdle(*maxIdle*);  // The minimum number of active connections that can remain idle in the pool,  // without extra ones being created, or 0 to create none.  *dataSource*.setMinIdle(*minIdle*);  // Sets the initial size of the connection pool.  *dataSource*.setInitialSize(*initialSize*);  } **catch** (Exception ex) {  **throw** **new** RuntimeException(ex + " Database connection failure");  }  }  // get a datasource  **public** **static** DataSource getDataSource() {  **return** *dataSource*;  }  **private** **static** **void** readProperty() **throws** IOException {  // Load a properties file from classpath  InputStream is = JDBCUtils.**class**.getClassLoader().getResourceAsStream("database.properties");  Properties prop = **new** Properties();  prop.load(is);  *driverClass* = prop.getProperty("driverClass");  *url* = prop.getProperty("url");  *username* = prop.getProperty("username");  *password* = prop.getProperty("password");  *maxTotal* = Integer.*parseInt*(prop.getProperty("maxTotal"));  *maxIdle* = Integer.*parseInt*(prop.getProperty("maxIdle"));  *minIdle* = Integer.*parseInt*(prop.getProperty("minIdle"));  *initialSize* = Integer.*parseInt*(prop.getProperty("initialSize"));  }  } |

1. Create MainApp class under edu.tcu.cs.et.app

|  |
| --- |
| **package** edu.tcu.cs.et.app;  /\*\*  \* Application starts here.  \* **@author** bwei  \*  \*/  **public** **class** MainApp {  **public** **static** **void** main(String[] args) {    }  } |

1. Under edu.tcu.cs.et.domain, create Java Bean Activity.

|  |
| --- |
| **package** edu.tcu.cs.et.domain;  /\*\*  \* Class Activity is called JavaBean, it corresponds to the table Activity in  \* Database. Each attribute in JavaBean corresponds to each field or column in  \* the table. There should also be setter and getter methods.  \*  \* **@author** bwei  \*  \*/  **public** **class** Activity {  **private** **int** id;  **private** String name;  **private** **double** money;  **private** String account;  **private** String createtime;  **private** String description;  **public** Activity(**int** id, String flname, **double** money, String account, String createtime, String description) {  **this**.id = id;  **this**.name = flname;  **this**.money = money;  **this**.account = account;  **this**.createtime = createtime;  **this**.description = description;  }  **public** Activity() {  }  @Override  **public** String toString() {  **return** "Activity [id=" + id + ", name=" + name + ", money=" + money + ", account=" + account + ", createtime=" + createtime + ", description=" + description + "]";  }  // getters and setters  **public** **int** getId() {  **return** id;  }  **public** **void** setId(**int** id) {  **this**.id = id;  }  **public** String getName() {  **return** name;  }  **public** **void** setName(String name) {  **this**.name = name;  }  **public** **double** getMoney() {  **return** money;  }  **public** **void** setMoney(**double** money) {  **this**.money = money;  }  **public** String getAccount() {  **return** account;  }  **public** **void** setAccount(String account) {  **this**.account = account;  }  **public** String getCreatetime() {  **return** createtime;  }  **public** **void** setCreatetime(String createtime) {  **this**.createtime = createtime;  }  **public** String getDescription() {  **return** description;  }  **public** **void** setDescription(String description) {  **this**.description = description;  }  } |

1. Under edu.tcu.cs.et.dao, create ActivityDAO class

|  |
| --- |
| package edu.tcu.cs.et.dao;  import org.apache.commons.dbutils.QueryRunner;  public class ActivityDao {  private QueryRunner qr = new QueryRunner(JDBCUtils.getDataSource());  } |

1. Under edu.tcu.cs.et.service, create ActivityService class

|  |
| --- |
| **package** edu.tcu.cs.et.service;  **import** edu.tcu.cs.et.dao.ActivityDao;  **public** **class** ActivityService {  **private** ActivityDao dao = **new** ActivityDao();    } |

1. Under edu.tcu.cs.et.controller, create ActivityController class

|  |
| --- |
| **package** edu.tcu.cs.et.controller;  **import** edu.tcu.cs.et.service.ActivityService;  **public** **class** ActivityController {  **private** ActivityService service = **new** ActivityService();    } |

1. Under edu.tcu.cs.et.view, create ActivityView class

|  |
| --- |
| **package** edu.tcu.cs.et.view;  **import** edu.tcu.cs.et.controller.ActivityController;  **public** **class** MainView {  //MainView holds an instance of ActivityController  **private** ActivityController controller = **new** ActivityController();  } |

1. Until now, every package has one class.

A screenshot of a cell phone

Description automatically generated

A close up of text on a white background

Description automatically generated

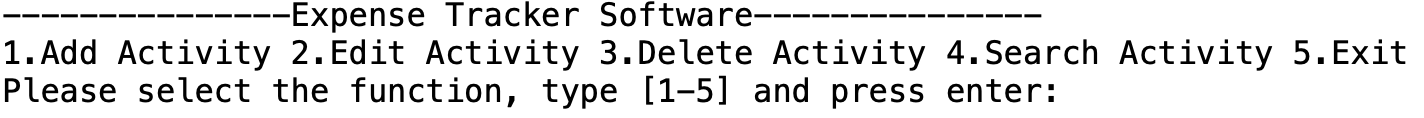
# Functionalities

Now, we implement the functionalities one by one. We start with “Display Main Menu,” then “Search All Activities,” then “Search Activities Based on Criteria,” then “Add Activity,” then “Edit Activity” and in the end, “Delete Activity.”

## Display Main Menu

Let’s work on MainView class first. Since this class is responsible for displaying and collecting inputs from user.

Below is the menu we would like to implement.

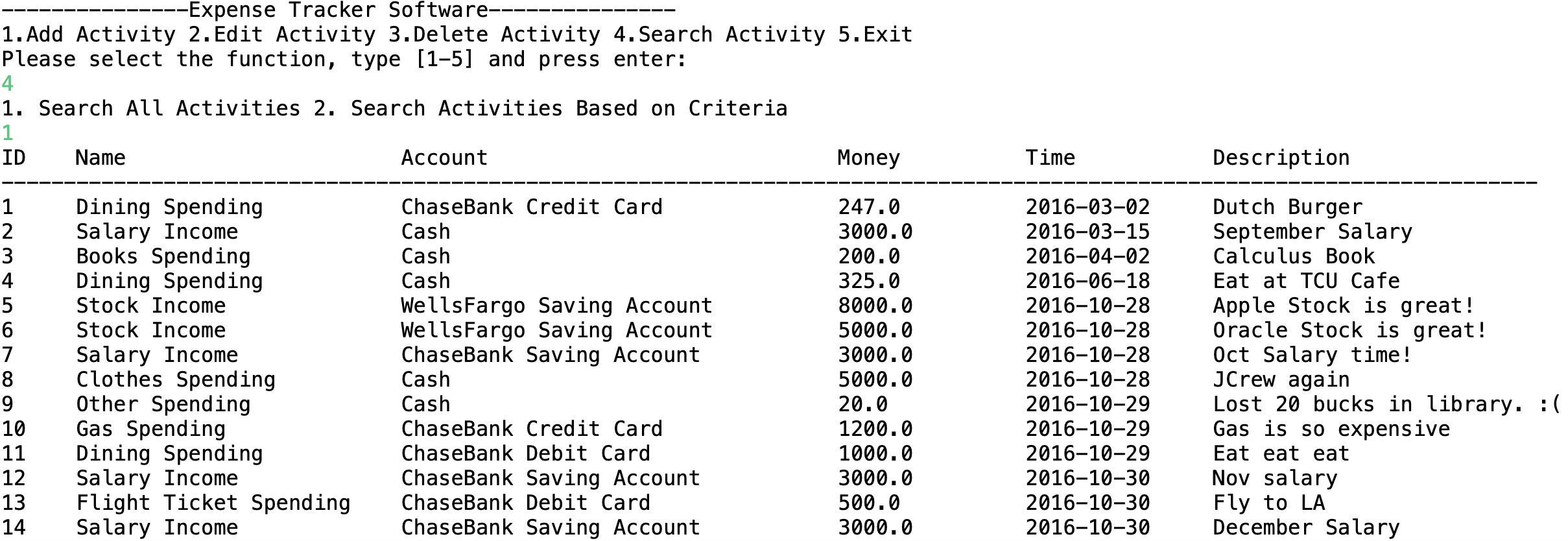


Steps:

1. Implement MainView run() method.
   1. Display the menu
   2. Receive user’s input
   3. Based on user’s input, invoke corresponding method.
2. Invoke MainView’s run() method from MainApp class’s main method.

## Search All Activities

Below is how a user invokes the Search All functionality:

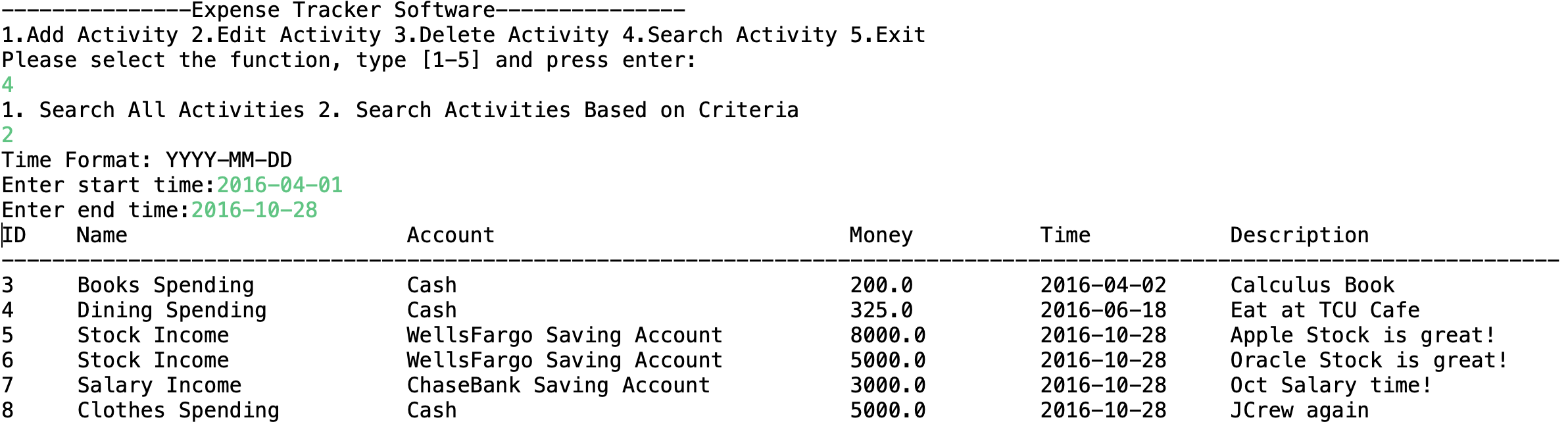


Steps:

1. Implement MainView class selectActivity method.
   1. Display two different search strategies, “1 Search All” and “2 Search Activities Based on Criteria”
   2. Invoke the corresponding method based on the user input (1 or 2)
2. Implement MainView class selectAll method
   1. Invoke ActivityController class selectAll method which returns Activities in a list.
   2. Invoke print method to print the list
3. Implement MainView class print method
   1. Print title: 
   2. Iterate the list of activities
4. Implement ActivityController selectAll method
   1. Invokes ActivityService class selectAll method
5. Implement ActivityService selectAll method
   1. Invokes ActivityDAP class selectAll method
6. Implement ActivityDAO selectAll method
   1. Use QueryRunner object’s query method to query database, returns a list of activities.

## Search Activities Based on Criteria

Below is how a user can invoke this functionality:

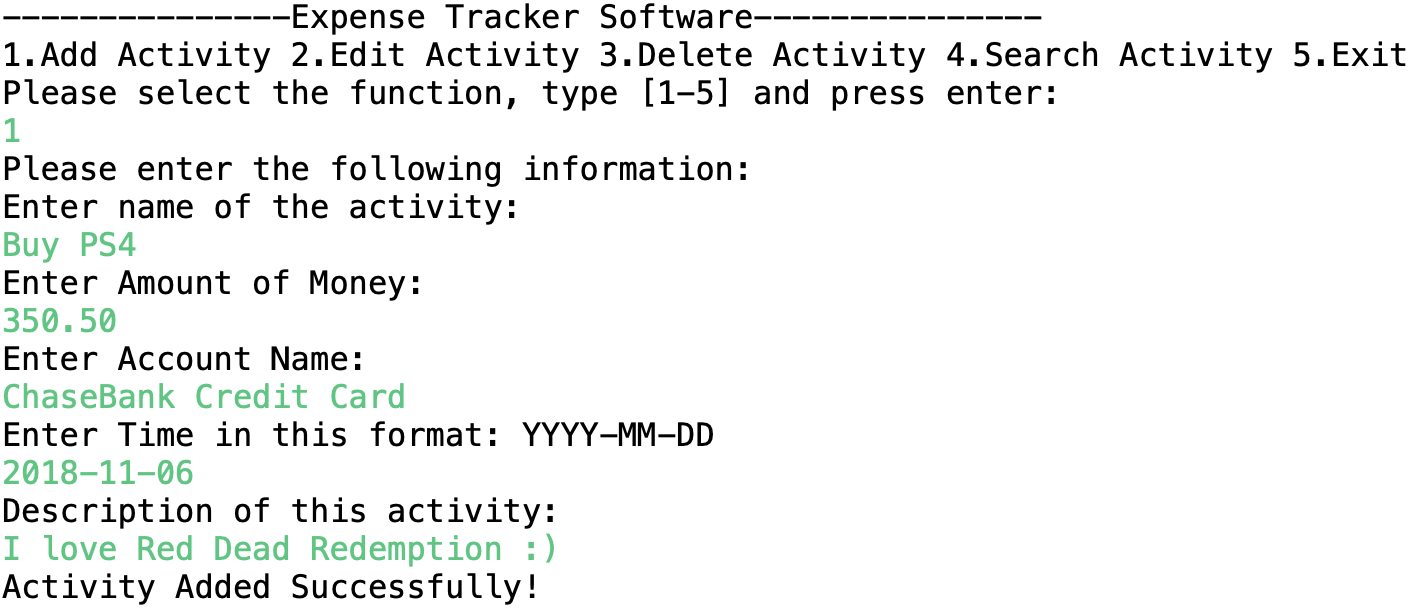


Steps:

1. Implement MainView class select method
   1. Collect start and end time from user
   2. Invoke ActivityController select method which returns a list of activities during that period
   3. Invoke print method to print the list of activities
2. Implement ActivityController select method
3. Implement ActivityService select method
4. Implement ActivityDAO select method

## Add Activity

Below is how a user can invoke this functionality:

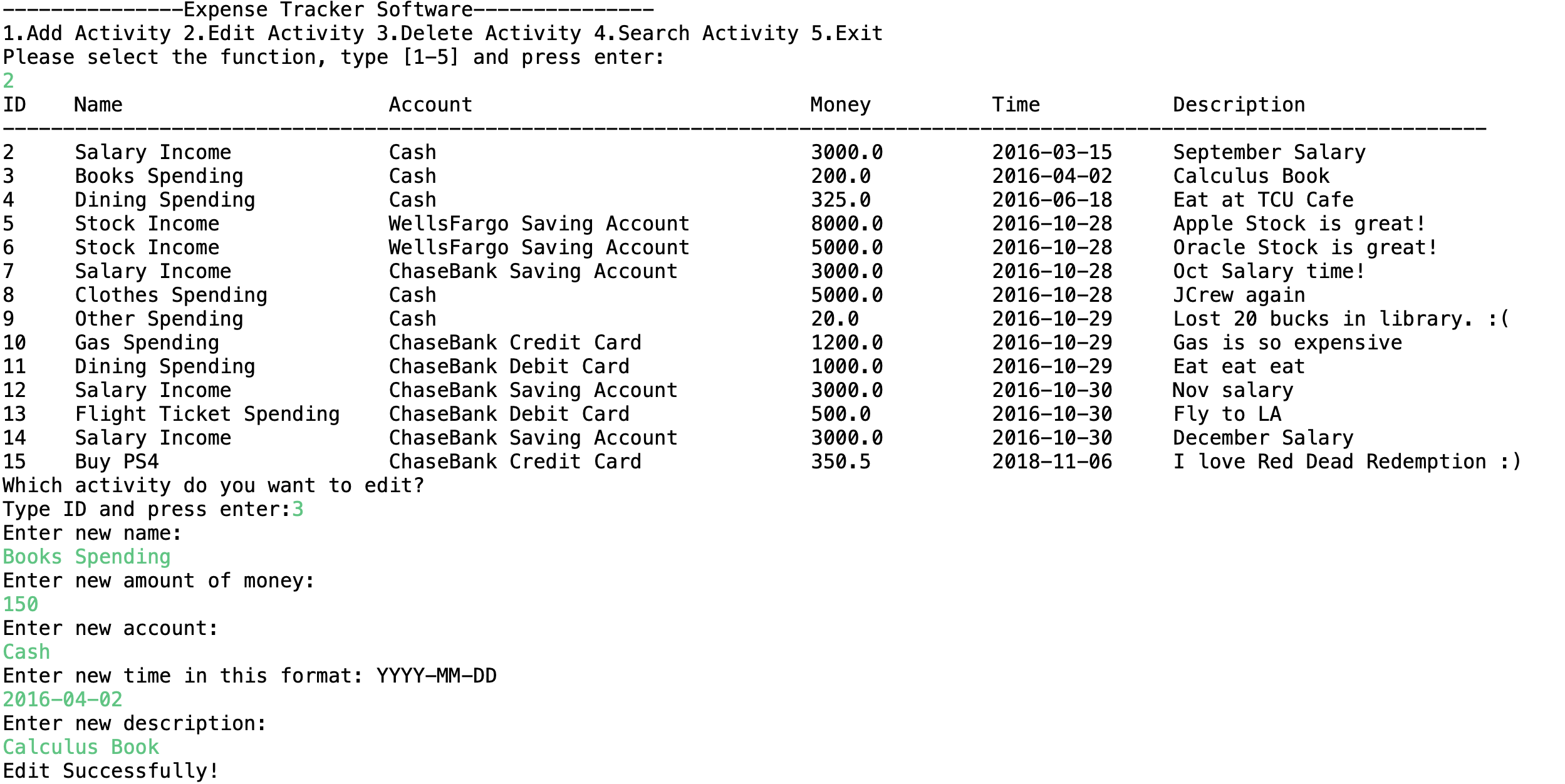


Steps:

1. Implement MainView class select method
   1. Collect name, money, account, time and description of the new activity
   2. Create an instance of Activity and put the above data in this instance.
   3. Invoke ActivityController class addActivity method and pass the Activity instance to it.
   4. Print success message to console.
2. Implement ActivityController addActivity method
   1. Invokes ActivityService addActivity
3. Implement ActivityService addActivity method
   1. Invokes ActivityDAO addActivity
4. Implement ActivityDAO addActivity method
   1. Invokes QueryRunner object’s method update to modify database.

## Edit Activity

Below is how a user can invoke this functionality:

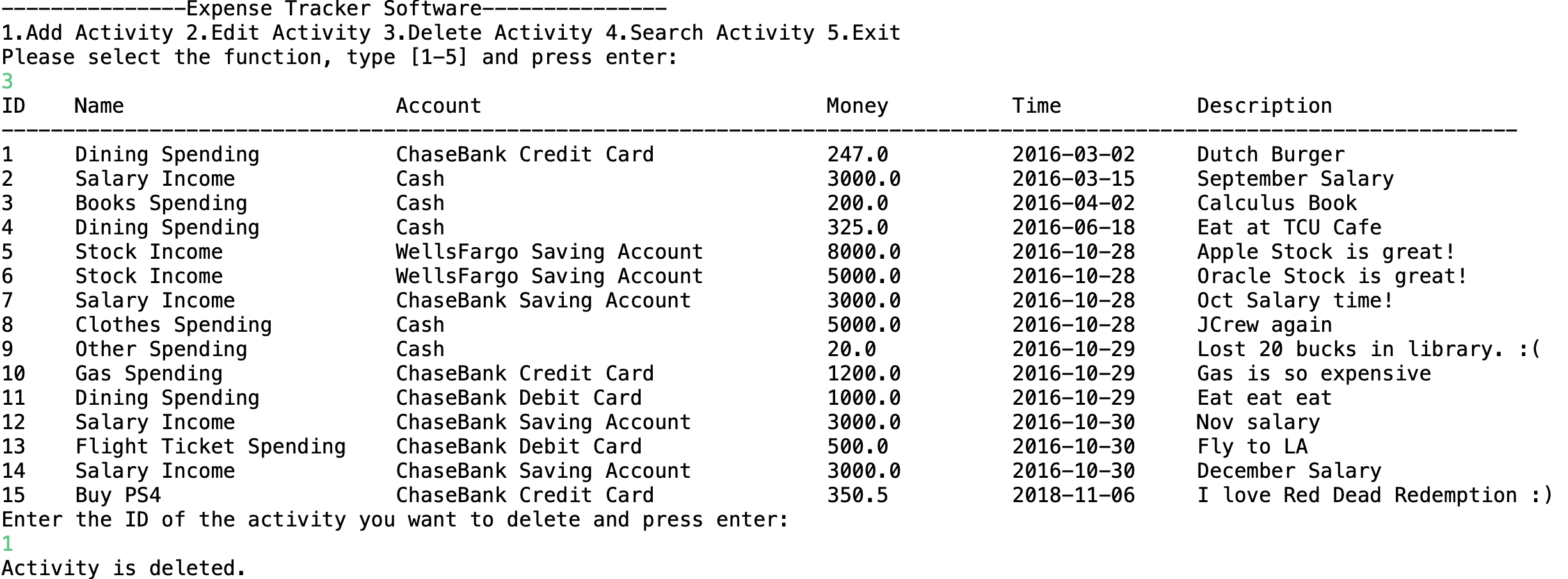


Steps:

1. Implement MainView class editActivity method
   1. Collect the id of activity to be edited
   2. Collect new information of this activity
   3. Invoke ActivityController class editActivity method
   4. Print edit successful message.
2. Implement ActivityController editActivity method
3. Implement ActivityService editActivity method
4. Implement ActivityDAO editActivity method

## Delete Activity

Below is how a user can invoke this functionality:



Steps:

1. Implement MainView class deleteActivity method
   1. Collect the id of the activity to be deleted
   2. Invoke ActivityController deleteActivity method
   3. Invoke print method
2. Implement ActivityController deleteActivity method
3. Implement ActivityService deleteActivity method
4. Implement ActivityDAO deleteActivity method