Manavendra Sen

OOP LAB 9 & 10

Exercise 1

```
import java.util.Scanner;
import java.io.*;

class InsufficientFundsException extends Exception {

 public InsufficientFundsException(String message) {
   super(message);
 }
}

class Bank {

 /** The array of BankAccount objects. */
 private BankAccount[] accounts;

 /** The first available account index. */
 private int firstAvailableAcc;

 /**
  * Creates a bank that can have up to numAccounts accounts.
  */
 public Bank(int numAccounts) {
   this.accounts = new BankAccount[numAccounts];
   this.firstAvailableAcc = 0;
 }

 /**
  * Adds the given BankAccount to the bank. If the bank is full an error
message
  * is printed and the bank is unchanged.
  *
  * @param account The account to add
```

```java
 */
public void add(BankAccount account) {
  if (firstAvailableAcc == accounts.length) {
    System.out.println("Bank is full. No account added.");
    return;
  }
  this.accounts[firstAvailableAcc] = account;
  firstAvailableAcc++;
}

/**
 * Returns the bank account with the given account number. If no such account
 * exists, null is returned.
 *
 * @param acctNumber The account number
 * @return The account
 */
public BankAccount find(int acctNumber) {
  for (int i = 0; i < firstAvailableAcc; i++) {
    if (accounts[i].getAccountNumber() == acctNumber) {
      return accounts[i];
    }
  }

  return null;
}

/**
 * Returns a string representation of the bank. The format is one account per
 * line.
 */
public String toString() {
  if (firstAvailableAcc == 0)
    return "NONE";

  String result = "";
```

```
    for (int i = 0; i < firstAvailableAcc; i++) {
      result += accounts[i].getAccountNumber() + " ";
      result += accounts[i].getBalance() + "\n";
      // Note that we don't make use of BankAccount's toString because
      // we don't want to have dollar signs ($) as part of the String
      // representation for the Bank.
    }

    return result;
  }
}

class BankAccount {

  private double balance;
  private int accountNumber;

  // ************
  // * TASK 2 *
  // ************
  /**
   * Constructor
   *
   * @param initialBalance the initial balance of the BankAccount
   * @param accountNumber  The account number to associate with this
BankAccount.
   *                       Must be a 5 digit integer.
   */
  public BankAccount(double initialBalance, int accountNumber) {

    if (initialBalance < 0) {
      throw new IllegalArgumentException("Accounts with a negative balance
cannot be created!");
    }
    // TODO TASK 2: add code here to throw an exception if the account
    // number isn't 5 digits.
```

```
    if (accountNumber < 10000 || accountNumber > 99999) {
      throw new IllegalArgumentException("Account Number must be of 5
digits!");
    }
    balance = initialBalance;
    this.accountNumber = accountNumber;

  }

  /**
   * Deposits money into the BankAccount
   *
   * @param amount the amount to deposit
   */
  public void deposit(double amount) {
    if (amount < 0) {
      throw new IllegalArgumentException("Don't deposit negative amounts!");
    }
    balance = balance + amount;
  }

  // *************
  // * TASK 3 *
  // *************
  /**
   * Withdraws money from the BankAccount
   *
   * @param amount the amount to withdraw
   */
  // TODO TASK 3: add code to throw our new exception if an overdraw is
attempted
  public void withdraw(double amount) throws InsufficientFundsException {
    if (amount < 0) {
      throw new IllegalArgumentException("Don't withdraw a negative amount!");
    }
    if (amount > this.balance)
```

```
    throw new InsufficientFundsException("Not sufficient balance in
account!!");
   balance = balance - amount;
 }

 /**
  * Gets the current balance of the BankAccount
  *
  * @return the current balance
  */
 public double getBalance() {
   return balance;
 }

 /**
  * Gets the account number of the BankAccount
  *
  * @return the account number
  */
 public int getAccountNumber() {
   return accountNumber;
 }

 /**
  * Returns a string representation this BankAccount in the following format,
  * e.g. 12345 $100.52 where 12345 is the account number, and 100.52 is the
  * balance.
  */
 public String toString() {
   return "" + accountNumber + " $" + balance;
 }
}

class BankApp {

 private static Scanner stdin = new Scanner(System.in);
```

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

  Bank bank = new Bank(100);

  // ************
  // * TASK 1 *
  // ************
  // TODO TASK 1: note that the account is initially null
  BankAccount account = null;
  int choice;
  double amount;
  int accountNumber;

  do {
    choice = getUserChoice();
    switch (choice) {
      case 1:
        amount = getAmount();
        accountNumber = getAccountNumber();
        try {
          account = new BankAccount(amount, accountNumber);
          bank.add(account);
          System.out.println("Account info: " + account + "\n");
        } catch (IllegalArgumentException exception) {
          System.out.println("\n*****ERROR*****: " + exception.getMessage() +
"\n");
        }
        break;

      // *****************
      // * TASKS 1 & 2 *
      // *****************
      // TODO TASK 1: Note the first catch clause
      // TODO TASK 2: Note the second catch clause
      case 2:
```

```
        amount = getAmount();
        try {
          account.deposit(amount);
          System.out.println("Account info: " + account + "\n");
        } catch (NullPointerException exception) {
          System.out.println("\n*****ERROR*****: " + "No account! First find
account" + " or create a new account\n");
        } catch (IllegalArgumentException exception) {
          System.out.println("\n*****ERROR*****: " + exception.getMessage() +
"\n");
        }
        break;

      // *****************
      // * TASKS 1 & 3 *
      // *****************
      // TODO TASK 1: add a similar NullPointerException catch clause below
      // TODO TASK 3: add a InsufficientFunds catch clause below
      case 3:
        amount = getAmount();
        try {
          account.withdraw(amount);
          System.out.println("Account info: " + account + "\n");
        } catch (NullPointerException exception) {
          System.out.println("\n*****ERROR*****: " + "No account! First find
account" + " or create a new account\n");
        } catch (IllegalArgumentException exception) {
          System.out.println("\n*****ERROR*****: " + exception.getMessage() +
"\n");
        } catch (InsufficientFundsException exception) {
          System.out.println("\n*****ERROR*****: " + exception.getMessage() +
"\n");
        }

        break;
```

```
      case 4:
        accountNumber = getAccountNumber();
        BankAccount found = bank.find(accountNumber);
        if (found != null) {
          account = found;
          System.out.println("Account info: " + account + "\n");
        } else {
          System.out.println("\n*****ERROR*****: Bank account " +
accountNumber + " not found!\n");
        }
        break;

      case 5:
        System.out.print("\n\nThe accounts: \n" + bank + "\n\n");
        break;
    }
  } while (choice != 0);
  System.out.println("\n\nGoodbye!");
}

private static int getUserChoice() {
  int choice;
  do {
    choice = -1;
    System.out.println("Menu Options:");
    System.out.println("0) Quit");
    System.out.println("1) Create new account");
    System.out.println("2) Deposit to current account");
    System.out.println("3) Withdraw from current account");
    System.out.println("4) Find account");
    System.out.println("5) Print all accounts");

    System.out.print("Enter your choice (0 - 5): ");
    try {
      choice = Integer.parseInt(stdin.nextLine());
    } catch (NumberFormatException exception) {
```

```
    }
    if (choice < 0 || choice > 5)
      System.out.println("Invalid choice");
  } while (choice < 0 || choice > 5);
  return choice;
}

private static double getAmount() {
  System.out.print("Enter the amount: $ ");
  double amount = -1;
  boolean valid = false;
  do {
    try {
      amount = Double.parseDouble(stdin.nextLine());
      valid = true;
    } catch (NumberFormatException exception) {
      System.out.println("Make sure you enter a valid double!");
    }
  } while (!valid);
  return amount;
}

private static int getAccountNumber() {
  System.out.print("Enter the account number: ");
  int amount = -1;
  boolean valid = false;
  do {
    try {
      amount = Integer.parseInt(stdin.nextLine());
      valid = true;
    } catch (NumberFormatException exception) {
      System.out.println("Make sure you enter a valid integer!");
    }
  } while (!valid);
  return amount;
}
```

```
}
```

```
mavn:~/ $ cd Java                                                    [0:10:29]
mavn:Java/ $ cd LAB9                                                 [0:10:58]
mavn:LAB9/ $ javac Bank.java                                         [0:16:24]
mavn:LAB9/ $ java BankApp                                            [0:16:32]
Menu Options:
0) Quit
1) Create new account
2) Deposit to current account
3) Withdraw from current account
4) Find account
5) Print all accounts
Enter your choice (0 - 5): 2
Enter the amount: $ 100

*****ERROR*****: No account! First find account or create a new account

Menu Options:
0) Quit
1) Create new account
2) Deposit to current account
3) Withdraw from current account
4) Find account
5) Print all accounts
Enter your choice (0 - 5): 1
Enter the amount: $ -10
Enter the account number: 11111

*****ERROR*****: Accounts with a negative balance cannot be created!

Menu Options:
0) Quit
1) Create new account
2) Deposit to current account
3) Withdraw from current account
4) Find account
5) Print all accounts
Enter your choice (0 - 5): 1
```

```
*****ERROR*****: Account Number must be of 5 digits!

Menu Options:
0) Quit
1) Create new account
2) Deposit to current account
3) Withdraw from current account
4) Find account
5) Print all accounts
Enter your choice (0 - 5): 1
Enter the amount: $ 10000
Enter the account number: 12345
Account info: 12345 $10000.0

Menu Options:
0) Quit
1) Create new account
2) Deposit to current account
3) Withdraw from current account
4) Find account
5) Print all accounts
Enter your choice (0 - 5): 3
Enter the amount: $ 1000
Account info: 12345 $9000.0

Menu Options:
0) Quit
1) Create new account
2) Deposit to current account
3) Withdraw from current account
4) Find account
5) Print all accounts
Enter your choice (0 - 5): 2
Enter the amount: $ 10000
Account info: 12345 $19000.0

Menu Options:
```

Exercise 2

```java
// Exeception Classes
class InvalidInitialTemperatureException extends Exception {
 private int temp;

 InvalidInitialTemperatureException(int temp) {
   this.temp = temp;
 }
```

```
 public String toString() {
   return "InvalidInitialTemperatureException : " + this.temp;
 }
}

/*************************************************************************
******/

class HighTemperatureException extends Exception {

 HighTemperatureException() {
 }

 public String toString() {
   return "\nHigh Temperature Exception : Cooling down\n";
 }
}

/*************************************************************************
******/

class LowTemperatureException extends Exception {
 LowTemperatureException() {
 }

 public String toString() {
   return "\nLow Temperature Exception : Heating\n";
 }
}

/*************************************************************************
******/

// Class Thermostat
class Thermostat {
```

```
 private int temperature;
 static final int LOWER_LIM = 50;
 static final int UPPER_LIM = 60;

 Thermostat(int initTemp) throws InvalidInitialTemperatureException {
   if ((initTemp >= LOWER_LIM) && (initTemp <= UPPER_LIM)) {
     this.temperature = initTemp;
     System.out.println("Thermostat Starting. With Initial Temprature:" +
temperature);
   } else {
     throw new InvalidInitialTemperatureException(initTemp);
   }
 }

 public void startThermostat() throws HighTemperatureException {
   System.out.println("*****************Thermostat
Started************************");
   while (true) {
     System.out.println(temperature);
     if (temperature != UPPER_LIM) {
       temperature++;
       try {
         Thread.sleep(1000);
       } catch (InterruptedException e) {
         System.out.println("Thread Interrupted");
       }
     } else {
       throw new HighTemperatureException();
     }

   }
   /*
    * This method increments (+1) and displays the temperature of the
Thermostat
    * after every 1000 ms. When the temperature reaches the UPPER_LIM it raises
    * HighTemperatureException.
```

```
    */

 }

 public void stopThermostat() throws LowTemperatureException {
   System.out.println("*****************Thermostat
Stopping************************");
   while (true) {
     System.out.println(temperature);
     if (temperature != LOWER_LIM) {
       temperature--;
       try {
         Thread.sleep(1000);
       } catch (InterruptedException e) {
         System.out.println("Thread Interrupted");
       }
     } else {
       throw new LowTemperatureException();
     }
   }
   /*
    * This method decrements (-1) and displays the temperature of the
Thermostat
    * after every 1000 ms. When the temperature reaches the LOWER_LIM it raises
    * LowTemperatureException.
    */

 }
}

/**********************************************************************
******/

class ThermostatDriver {
 public static void main(String[] args) throws
InvalidInitialTemperatureException {
```

```
   Thermostat t = new Thermostat(55);
   boolean flag = true;
   int z = 0;
   // Setting the initial temperature of the thermostat as 55.

   while (flag) {

     try {
       t.startThermostat();
     } catch (HighTemperatureException e) {
       System.out.println(e);
     }

     try {
       t.stopThermostat();
     } catch (LowTemperatureException ex) {
       System.out.println(ex);
     }
     z++;
     if (z == 2) {
       return;
     }
   }

 }// End of main()
} // End of ThermostatDriver
```

```
mavn:LAB9/ $ javac ThermoDriver.java                                    [0:27:31]
mavn:LAB9/ $ java ThermostatDriver                                      [0:28:58]
Thermostat Starting. With Initial Temprature:55
*****************Thermostat Started************************
55
56
57
58
59
60

High Temperature Exception : Cooling down

*****************Thermostat Stopping************************
60
59
58
57
56
55
54
53
52
51
50

Low Temperature Exception : Heating

*****************Thermostat Started************************
50
51
52
53
54
55
56
57
```

```
51
50

Low Temperature Exception : Heating

*****************Thermostat Started************************
50
51
52
53
54
55
56
57
58
59
60

High Temperature Exception : Cooling down

*****************Thermostat Stopping************************
60
59
58
57
56
55
54
53
52
51
50

Low Temperature Exception : Heating

mavn:LAB9/ $                                                    [0:29:36]
```