**HANDS-ON #5:** EXCEPTIONS AND FILE HANDLING

1. Objective

After completing this first lab tutorial, you can:

* Understand and implement exception handling in Java programs.
* Understand and implement file handling in Java programs

1. Literature
   1. Java exception handling
      1. There are three types of errors:
         * Syntax errors: This occurs when the rule of the language is violated and detected by the compiler.
         * Run-time errors: Occurs when the computer detects an operation that cannot be carried out ( e.g., division by zero, x/y is syntactically correct, but if y is zero at run-time a run-time error will occur).
         * Logic errors: This occurs when a program does not perform the intended task.
      2. Instead of deciding how to deal with an error, Java provides the exception mechanism:
         * Indicate an error (exception event) has occurred.
         * Let the user decide how to handle the problem in a separate section of code specific for that  
           purpose.
         * Crash the program if the error is not handled.
   2. Exception Indication
      1. Use built-in exception class

There are many useful predefined exception classes, for example:  
*- ArithmeticException  
- NullPointerException  
- IndexOutOfBoundsException  
- IllegalArgumentException  
- InputMismatchException*

Example 1.

**import** java.util.InputMismatchException;

**import** java.util.Scanner;

**public** **class** SampleException {

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

Scanner sc = **new** Scanner(System.***in***);

**boolean** isError = **false**;

**do** {

System.***out***.println("Enter an integer number");

**try** {

**int** num = sc.nextInt();

System.***out***.println("num = " + num);

isError = **false**;

} **catch** (InputMismatchException e) {

System.***err***.println("Incorrect input");

sc.nextLine(); // skip newline

isError = **true**;

}

} **while** (isError);

}

}

Example 2.

**public** **class** SampleException2 {

**public** **static** **double** factorial(**int** n) **throws** IllegalArgumentException {

**if** (n < 0) {

IllegalArgumentException obj = **new** IllegalArgumentException(n + " is invalid.");

**throw** obj;

}

**else** {

**double** output = 1;

**for** (**int** i = 2; i <= n; i++) {

output \*= i;

}

**return** output;

}

}

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

System.***out***.println("n = 5 --> " + *factorial*(5));

System.***out***.println("n = -1 --> " + *factorial*(-1));

// try {

// System.out.println("n = -1 --> " + factorial(-1));

// }

// catch (IllegalArgumentException err) {

// System.out.println(err);

// }

System.***out***.println("n = 6 --> " + *factorial*(6));

}

}

* + 1. Define new Exception class

New exception classes can be defined by deriving from class Exception. Then it can be used in throw statements and catch blocks.

Example 3.

**public** **class** MathException **extends** Exception {

**public** MathException() {

**super**();

}

**public** MathException(String s) {

**super**(s);

}

}

**public** **class** SampleException3 {

**public** **static** **double** factorial(**int** n) **throws** MathException {

**if** (n < 0) {

**throw** **new** MathException(n + " is invalid.");

}

**else** {

**double** output = 1;

**for** (**int** i = 2; i <= n; i++) {

output \*= i;

}

**return** output;

}

}

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

System.***out***.println("n = 5 --> " + *factorial*(5));

System.***out***.println("n = -1 --> " + *factorial*(-1));

// try {

// System.out.println("n = -1 --> " + factorial(-1));

// }

// catch (IllegalArgumentException err) {

// System.out.println(err);

// }

System.***out***.println("n = 6 --> " + *factorial*(6));

}

}

* 1. Java File Handling

File handling is an important part of any application.

Java has several methods for creating, reading, updating, and deleting files.

The File class from the java.io package, allows us to work with files.

To use the File class, create an object of the class, and specify the filename or directory name

import java.io.File; // Import the File class  
File myObj = new File("filename.txt"); // Specify the filename

The File class has many useful methods for creating and getting information about files.

|  |  |  |
| --- | --- | --- |
| **Method** | **Type** | **Description** |
| canRead() | Boolean | Tests whether the file is readable or not |
| canWrite() | Boolean | Tests whether the file is writable or not |
| createNewFile() | Boolean | Create an empty file |
| delete() | Boolean | Deletes a file |
| exists() | Boolean | Tests whether the file exists |
| getName() | String | Returns the name of the file |
| getAbsolutePath() | String | Returns the absolute pathname of the file |
| length() | Long | Returns the size of the file in bytes |
| list() | String[] | Returns an array of the files in the directory |
| mkdir() | Boolean | Creates a directory |

* + 1. Create a File

To create a file in Java, you can use the createNewFile() method. This method returns a boolean value: true if the file was successfully created, and false if the file already exists. Note that the method is enclosed in a try…catch block. This is necessary because it throws an IOException if an error occurs (if the file cannot be created for some reason):

Example 4.

**import** java.io.File;

**import** java.io.IOException;

**public** **class** CreateFile {

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

// **TODO** Auto-generated method stub

**try** {

File myObj = **new** File("filename.txt");

**if** (myObj.createNewFile()) {

System.***out***.println("File created: " + myObj.getName());

}

**else** {

System.***out***.println("File already exists.");

}

} **catch** (IOException e) {

System.***out***.println("An error occurred.");

e.printStackTrace();

}

}

}

* + 1. Write to a File

In the following example, we use the FileWriter class together with its write() method to write  
some text to the file we created in the example above. Note that when you are done writing to the file,  
you should close it with the close() method:

Example 5.

**import** java.io.FileWriter;

**import** java.io.IOException;

**public** **class** WriteToFile {

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

**try** {

FileWriter myWriter = **new** FileWriter("filename2.txt");

myWriter.write("Hello world! Hello TDTU!");

myWriter.close();

System.***out***.println("Successfully wrote to the file.");

} **catch** (IOException e) {

System.***out***.println("An error occurred.");

e.printStackTrace();

}

}

}

Note: Many available classes in the Java API can be used to write files in Java: FileWriter,  
BufferedWriter, FileOutputStream, etc.

* + 1. Read a File

Example 6.

**import** java.io.File;

**import** java.io.FileNotFoundException;

**import** java.io.FileReader;

**import** java.util.Scanner;

**public** **class** ReadFile {

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

// **TODO** Auto-generated method stub

**try** {

File myObj = **new** File("filename2.txt");

Scanner myReader = **new** Scanner(myObj);

**while** (myReader.hasNextLine()) {

String data = myReader.nextLine();

System.***out***.println(data);

}

myReader.close();

} **catch** (FileNotFoundException e) {

System.***out***.println("An error occurred.");

e.printStackTrace();

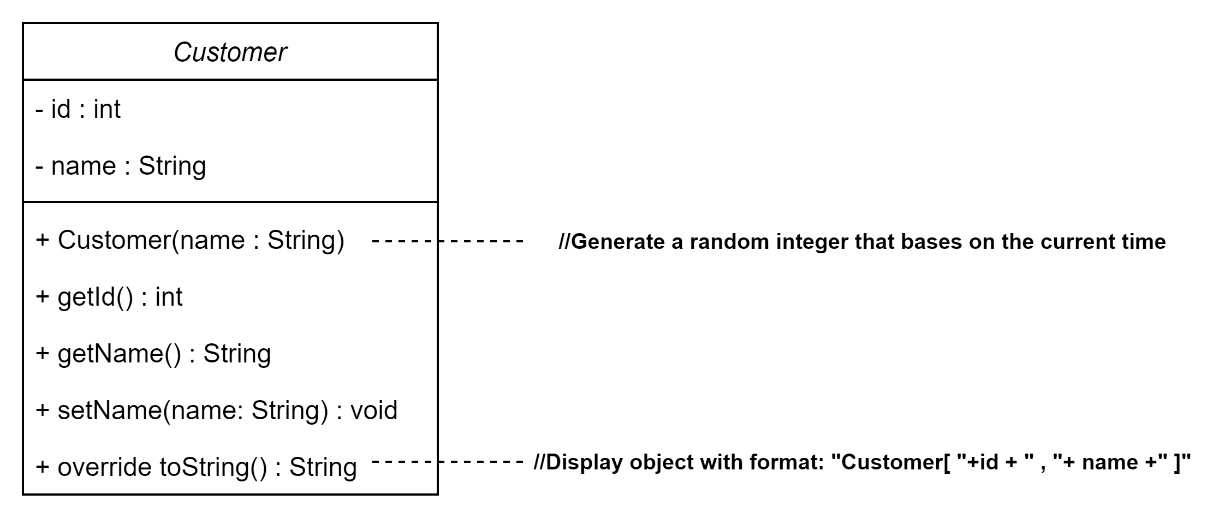
}

}

}

* + 1. Read and Write Object to a File with Serializable

Create class Customer with class diagram



To read/write object on file, the class must implement interface Serializable

**import** java.io.Serializable;

**import** java.util.Random;

**public** **class** Customer **implements** Serializable{

//Data member

**private** **int** id;

**private** String name;

//Constructors

**public** Customer(String name)

{

//Generate a random integer that bases on the current time

Random rand = **new** Random(System.currentTimeMillis());

**this**.id =Math.*abs*(rand.nextInt());

**this**.name = name;

}

//Getters && Setters

**public** **int** getId() {

**return** id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

//Override method toString for display object

@Override

**public** String toString() {

// **TODO** Auto-generated method stub

**return** "Customer[ "+id + " , "+ name +" ]";

}

}

Create Design Pattern Factory to Read and write objects to files using Serializable

**public** **class** SerializableFileFactory {

**public** **static** **boolean** writeFile(Vector<Customer> customers, String path)

{

**try** {

FileOutputStream fos = **new** FileOutputStream(path);

ObjectOutputStream oos = **new** ObjectOutputStream(fos);

oos.writeObject(customers);

oos.close();

fos.close();

**return** **true**;

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

**return** **false**;

}

**public** **static** Vector<Customer> readFile(String path)

{

Vector<Customer> customers= **new** Vector<Customer>();

**try** {

FileInputStream fis = **new** FileInputStream(path);

ObjectInputStream ois = **new** ObjectInputStream(fis);

customers =(Vector<Customer>) ois.readObject();

ois.close();

fis.close();

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

**return** customers;

}

}

Create class TestReadWriteObject

**public** **class** TestReadWriteObject {

**public** **static** Scanner *sc* = **new** Scanner(System.***in***);

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

// **TODO** Auto-generated method stub

Vector<Customer> listCustomers = **new** Vector<Customer>();

//Input n Customer

**int** n;

n = Integer.*parseInt*(*sc*.nextLine());

**for**(**int** i=0; i<n;i++)

{

String name;

name = *sc*.nextLine();

Customer c = **new** Customer(name);

listCustomers.add(c);

}

//Write listCustomers to file data.txt

SerializableFileFactory.*writeFile*(listCustomers, "data.txt");

//Read listCustomers from file data.txt

Vector<Customer> data = SerializableFileFactory.*readFile*("data.txt");

**for** (Customer c : data) {

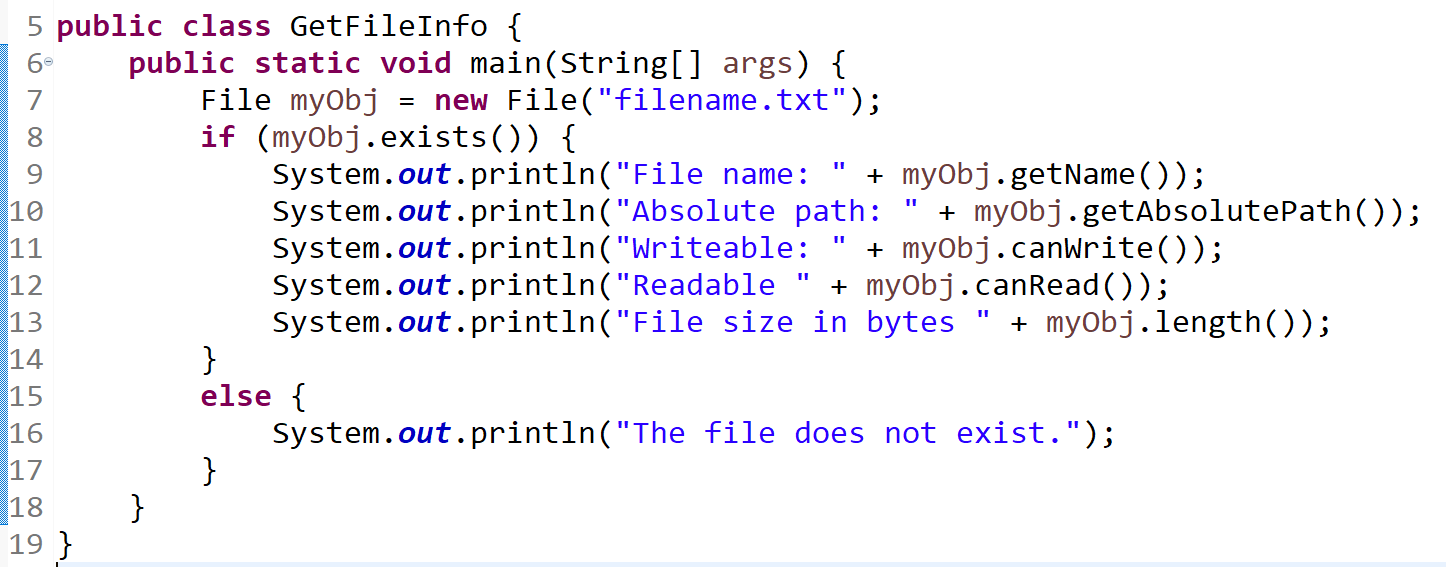
System.***out***.println(c);

}

}

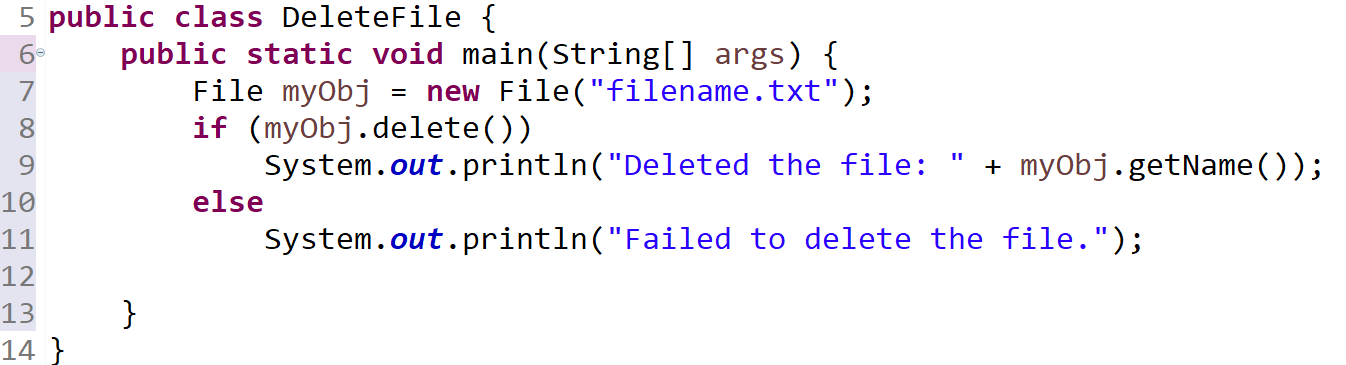
}

* + 1. Get File Information

To get more information about a file, use any of the File methods:

* + 1. Delete a File

To delete a file in Java, use the delete() method:

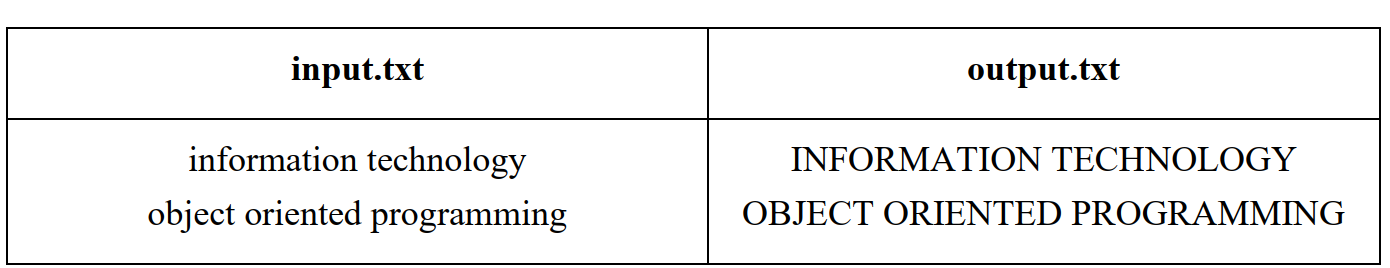


1. Hands-on
   1. Create class Calculator has 2 method below:

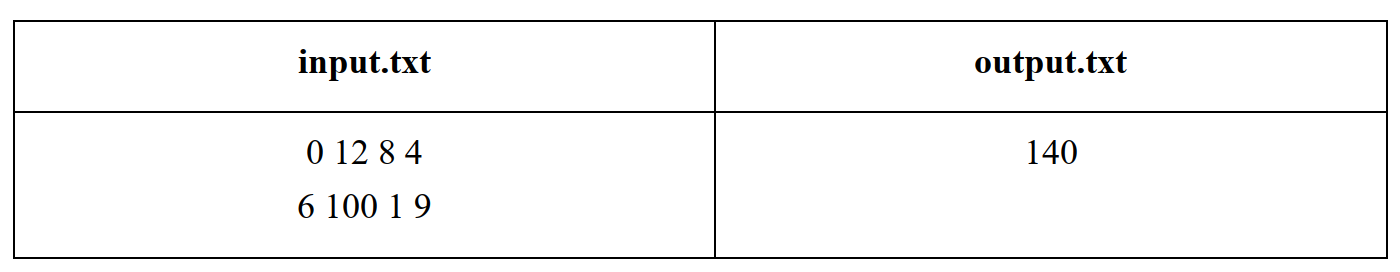
* public double divide(int a, int b)
* public int multiply(int a, int b)

**Implement exceptions:**

* If parameter b = 0, the method throws an exception ArithmeticException with the  
  message “divide by zero”.
* If the values of a and b are outside the range [-1000, 1000], the method throws an  
  exception NumberOutOfRangeException with the message “Number is outside the  
  computation”. NumberOutOfRangeException is a student-defined exception.
  1. Write a Java program:
* Read all contents from an input.txt file.
* Then uppercase all contents and write the results to the output.txt file.  
  For example:



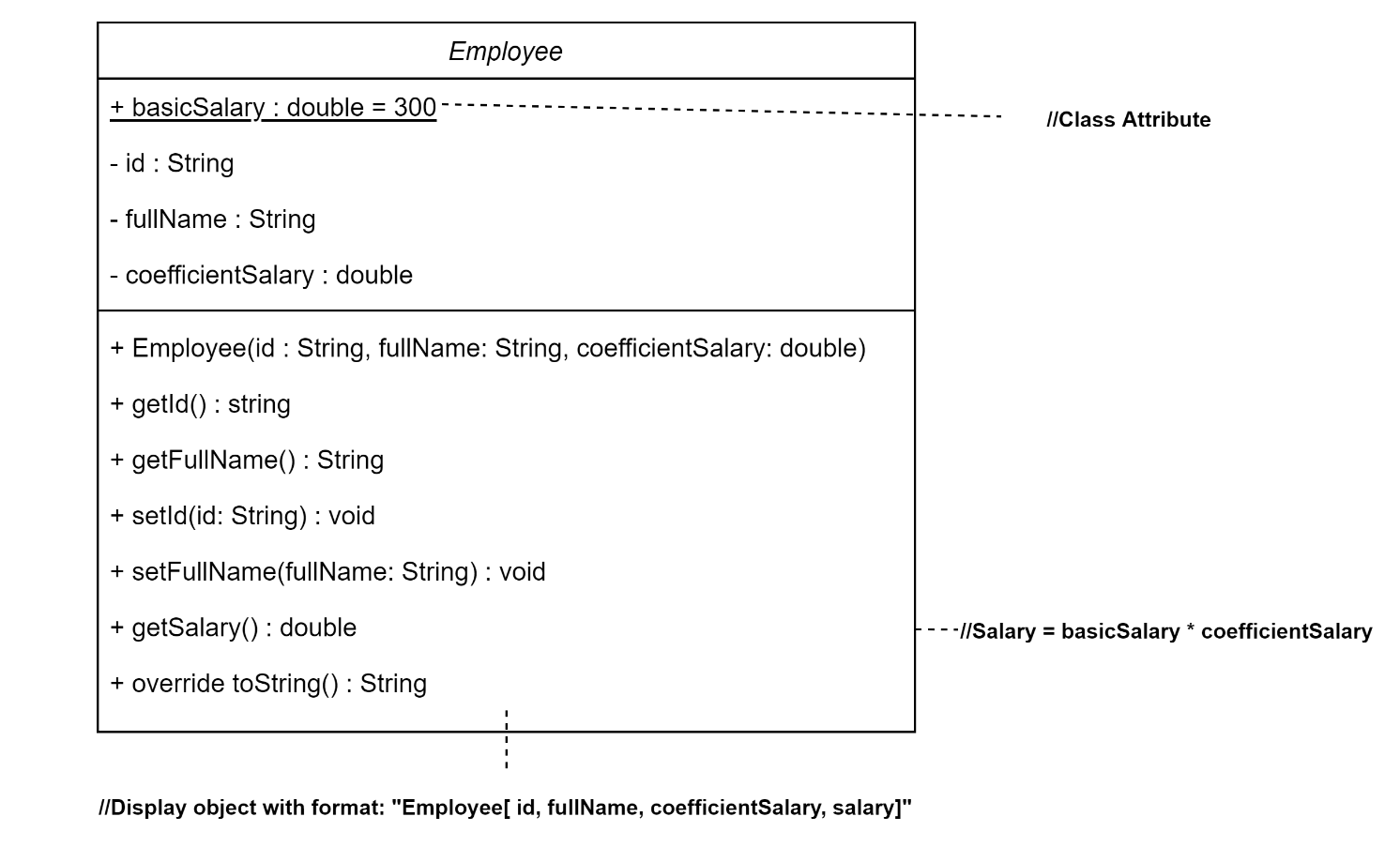
* 1. Write a Java program:
* Read all integers from an input.txt file.
* Then calculate the sum of them and write the result to the output.txt file.



* 1. Write a Java program:
* Get specific files by extensions from a given folder.
* Check if a file or directory specified by pathname exists or not.
* Check if given pathname is a directory or a file.
* Append text to an existing file.
* Find the longest word in a text file.
  1. Write a Java program to manage the salary of employees of ABC company, including the following information: employee code (string), employee's full name (string), salary coefficient (real number).

**Make the following requests:**

* Create a class **Employee** with class diagrams:



* Enter n (n>0) employees and write data.txt file with format:

Line 1: n

Line 2 to n+1 with format:

“Id/tfullName/tsalaryCoefficient”

* Read employee information from data.txt file and print payroll to the screen in the format:

Employee[ id, fullName, coefficientSalary, salary]

**Reference code:**

**public** **class** Employee {

//Data member

**public** **static** **double** *basicSalary* = 300;

**…**

//Constructors

**public** Employee(String id, String fullName, **double** salaryCoefficient) {

…

}

//Methods

**…**

}

**import** java.io.File;

**import** java.io.FileWriter;

**import** java.io.IOException;

**import** java.util.Scanner;

**import** java.util.Vector;

**public** **class** TestEmployee {

**public** **static** Scanner *sc* = **new** Scanner(System.***in***);

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

// **TODO** Auto-generated method stub

//Input list employees

Vector<Employee> list = **new** Vector<Employee>();

**int** n;

Employee.*basicSalary* = 500;

n = Integer.*parseInt*(*sc*.nextLine());

**for**(**int** i=0; i<n;i++)

{

String id, fullName;

**double** salaryCoefficient;

id = *sc*.nextLine();

fullName = *sc*.nextLine();

salaryCoefficient = Double.*parseDouble*(*sc*.nextLine());

Employee e = **new** Employee(id, fullName, salaryCoefficient);

list.add(e);

}

//Write list employees to file

**try** {

FileWriter fw = **new** FileWriter("data.txt");

fw.write(list.size());

….

fw.close();

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

//Read list employees from file display screen

….

}

}

* 1. Write a java program to manage a bank account (BankAccount), each customer can open a bank account including the following information: account number (accNum - integer ), customer's full name (name – string ), account balance (balance – real number). Customers can create an account with the account number and initial balance, customers can perform the following services: deposit an amount in the account (deposit), withdraw an amount from the account (withdraw) or transfer money from one account to another (transferMoney).

**Class Diagram**

********

**Request:**

* Create a BankAcount class with the necessary attributes (Attribute/Field/Data Member) and methods (Methods).\
* Write class TetstBankAccount according to the following requirements:
* Enter integer n and enter n account information including: account number, customer name and balance and write data.txt file. (with Serializable)
* Read the account list from data.txt file. (with Serializable)
* Make money transfer to the account number.
* Make a withdrawal from the account number.
* Transfer money from one account number to another.
* Print the list of accounts to the screen.