Lab 11

File Handling

Objective:

The purpose of lab is to make students understand ways of getting information in and out of your Java programs, using files.

Activity Outcomes:

This lab teaches you the following topics:

- Students will be able to retrieve create file.
- Students will be able to write and read data to and from a file.
- Students will learn about Object Serialization.

Instructor Note:

As pre-lab activity, read Chapter 17 from the text book “Introduction to Java Programming”, Y. Daniel Liang, Pearson, 2019.

1) Useful Concepts

To read an entire object from or write an entire object to a file, Java provides object serialization. A serialized object is represented as a sequence of bytes that includes the object's data and its type information. After a serialized object has been written into a file, it can be read from the file and deserialized to recreate the object in memory.

A class that implements the Serializable interface is said to be a serializable class. To use objects of a class with writeObject() and readObject() , that class must be serializable. But to make the class serializable, we change nothing in the class. All we do is add the phrase implements Serializable . This phrase tells the run-time system that it is OK to treat objects of the class in a particular way when doing file I/O.

Classes ObjectInputStream and ObjectOutputStream, which respectively implement the ObjectInput and ObjectOutput interfaces, enable entire objects to be read from or written to a stream.

To use serialization with files, initialize ObjectInputStream and ObjectOutputStream objects with FileInputStream and FileOutputStream objects

ObjectOutput interface method writeObject takes an Object as an argument and writes its information to an OutputStream.

A class that implements ObjectOuput (such as ObjectOutputStream) declares this method and ensures that the object being output implements Serializable.

ObjectInput interface method readObject reads and returns a reference to an Object from an InputStream. After an object has been read, its reference can be cast to the object's actual type.

```
. /**
Demonstrates binary file I/O of serializable class objects.
*/
                public class ObjectIODemo
                {
                public static void main(String[] args)
                {
                try
                {
                ObjectOutputStreamoutputStream =
                new ObjectOutputStream(new FileOutputStream("datafile"));
                SomeClass oneObject = new SomeClass(1, 'A');
                SomeClassan otherObject = new SomeClass(42, 'Z');
                outputStream.writeObject(oneObject);
```

```
outputStream.writeObject(anotherObject);
outputStream.close();
System.out.println("Data sent to file.");
}
catch(IOException e)
{
System.out.println("Problem with file output.");
}
System.out.println(
"Now let's reopen the file and display the data.");

try
{
ObjectInputStreaminputStream =
new ObjectInputStream(new FileInputStream("datafile")); Notice the type casts.
SomeClassreadOne = (SomeClass)inputStream.readObject( );
SomeClassreadTwo = (SomeClass)inputStream.readObject( );
System.out.println("The following were read from the file:");
System.out.println(readOne);
System.out.println(readTwo);
}
catch(FileNotFoundException e)
{
System.out.println("Cannot find datafile.");
}
catch(ClassNotFoundException e)
{
System.out.println("Problems with file input.");
}

catch(IOException e)
{
System.out.println("Problems with file input.");
}
System.out.println("End of program.");
}
}
```

2) Solved Lab Activities (Allocated Time 50 Min.)

Sr.No	*Allocated Time*	*Level of Complexity*	*CLO Mapping*
Activity 1	*25 mins*	*Medium*	*CLO-4*
Activity 2	*25 mins*	*High*	*CLO-4*

Activity 1:

The following example demonstrates writing of objects to a file.

Solution:

```
import java.io.*;

public  class Person implements Serializable
{
public String name = null;
public int age = 0;

public void setAge(int a) { age = a ;}

public String getName() {return name ;}

}
-----------------------------------------------------------------------
import java.io.*;
public class ObjectOutputStreamExample {

public void writeToFile() {

try
{
ObjectOutputStream objectOutputStream =
new ObjectOutputStream(new FileOutputStream("filename"));

Person p = new Person();
p.name = "Jakob Jenkov"; p.age = 40;
```

```
objectOutputStream.writeObject(p);
}
catch (FileNotFoundException ex)
{ ex.printStackTrace();
}
catch (IOException ex)
{ ex.printStackTrace();
} } }
```

Activity 2:

The following example demonstrates reading of all objects from a file.

```
import java.io.*;

public class ObjectInputStreamExample {

public void readFromFile()

{

try

{

ObjectInputStream objectInputStream = new ObjectInputStream(new
FileInputStream("filename"));

while (true)

{

Person personRead = (Person) objectInputStream.readObject();

System.out.println(personRead.name);

System.out.println(personRead.age);

}

}

catch (EOFException ex) { //This exception will be caught when EOF is
reached System.out.println("End of file reached.");
```

```
} catch (ClassNotFoundException ex) { ex.printStackTrace();
} catch (FileNotFoundException ex) { ex.printStackTrace();
} catch (IOException ex) { ex.printStackTrace();}}}
```

Output

Jakob Jenkov

40

3) Graded Lab Tasks(Allocated Time 2 Hr.)

Note: The instructor can design graded lab activities according to the level of difficulty and complexity of the solved lab activities. The lab tasks assigned by the instructor should be evaluated in the same lab.

Lab Task 1

Create a class Book that has name(String), publisher (String) and an author (Person). Write five objects of Book Class in a file named "BookStore".

Lab Task 2

Consider the Book class of Activity 1 and write a function that displays all Books present in file "BookStore".

Lab Task 3

Consider the Book class of Activity 1 and write a function that asks the user for the name of a Book and searches the record against that book in the file "BookStore".

Lab Task 4

Create an ATM System with Account as the Serializable class. Write ten objects of Account in a file. Now write functions for withdraw, deposit, transfer and balance inquiry.

Note:

a. Each function should ask for the account number on which specific operation should be made.

b. All changes in Account object should be effectively represented in the file.