NMK30703: Programming for Networks

Lab Ex 2:

Streams

Mohamed Elshaikh

Faculty of Electronics Engineering Technology – UniMAP (FTKEN-UniMAP)

In-Lab Exercises & Solutions

Exercise 1: Basic File Copy (Byte-by-Byte)

Write a Java program that copies `farrago.txt` to `outagain.txt` using `FileInputStream` and `FileOutputStream` without buffering.

Solution:

```java
import java.io.*;
public class FileCopyNoBuffer {
   public static void main(String[] args) {
      try (FileInputStream in = new FileInputStream("farrago.txt");
           FileOutputStream out = new FileOutputStream("outagain.txt")) {
         int byteRead;
         while ((byteRead = in.read()) != -1) {
            out.write(byteRead);
         }
         System.out.println("File copied byte-by-byte!");
      } catch (IOException ex) {
         ex.printStackTrace();
      }
   }
}
```

Exercise 2: Buffered File Copy (Block I/O)

Task:

Modify Exercise 1 to use a **4KB buffer** for faster copying.

Solution:

```java
import java.io.*;
public class FileCopyBuffered {
   public static void main(String[] args) {
      try (FileInputStream in = new FileInputStream("farrago.txt");
           FileOutputStream out = new FileOutputStream("outagain.txt")) {
         byte[] buffer = new byte[4096]; // 4KB buffer
         int bytesRead;
```

```
            while ((bytesRead = in.read(buffer)) != -1) {
                out.write(buffer, 0, bytesRead);
            }
            System.out.println("File copied with 4KB buffer!");
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
}
```

Key Points:

- Buffered I/O reduces system calls.
- `read(byte[])` reads chunks of data.

Exercise 3: Chaining Streams (Buffered + DataInputStream)

Task:

Chain `FileInputStream → BufferedInputStream → DataInputStream` to read primitive data types.

Solution:

```java
import java.io.*;
public class ChainedStreams {
    public static void main(String[] args) {
        try (DataInputStream in = new DataInputStream(
                new BufferedInputStream(
                    new FileInputStream("data.bin")))) {
            int intValue = in.readInt();
            double doubleValue = in.readDouble();
            System.out.println("Read: " + intValue + ", " + doubleValue);
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
}
```

Key Points:

- `DataInputStream` reads `int`, `double`, etc.
- Buffering improves performance.

Exercise 4: Character Streams (FileReader/FileWriter)

Task:

Copy `farrago.txt` using `FileReader` and `FileWriter` (character streams).

Solution:

```java
import java.io.*;
public class CharStreamCopy {
   public static void main(String[] args) {
      try (FileReader in = new FileReader("farrago.txt");
           FileWriter out = new FileWriter("outagain.txt")) {
         int charRead;
         while ((charRead = in.read()) != -1) {
            out.write(charRead);
         }
         System.out.println("File copied using char streams!");
      } catch (IOException ex) {
         ex.printStackTrace();
      }
   }
}
```

Key Points:

- Uses 16-bit Unicode characters.
- Default charset (may not handle all encodings).

Exercise 5: Buffered Character I/O (readLine())

Task:

Use `BufferedReader.readLine()` to print `farrago.txt` line-by-line.

Solution:

```java
import java.io.*;
public class ReadLines {
```

```
    public static void main(String[] args) {
        try (BufferedReader in = new BufferedReader(
                new FileReader("farrago.txt"))) {
            String line;
            while ((line = in.readLine()) != null) {
                System.out.println(line);
            }
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
}
```

Key Points:

- `readLine()` excludes line terminators.
- More efficient than char-by-char.

Exercise 6: Charset Encoding (UTF-8)

Task:

Write "Hello, 您好!" to a file using `OutputStreamWriter` with UTF-8 encoding.

Solution:

```java
import java.io.*;
import java.nio.charset.StandardCharsets;
public class CharsetDemo {
    public static void main(String[] args) {
        try (OutputStreamWriter out = new OutputStreamWriter(
                new FileOutputStream("utf8.txt"), StandardCharsets.UTF_8)) {
            out.write("Hello, 您好!");
            System.out.println("File written in UTF-8!");
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
}
```

Key Points:

- Explicit charset avoids platform dependency.
- Supports non-ASCII characters.

Exercise 7: Byte-to-Char Decoding (InputStreamReader)

Task:
Read `utf8.txt` using `InputStreamReader` with UTF-8 decoding.

Solution:

```java
import java.io.*;
import java.nio.charset.StandardCharsets;
public class DecodeFile {
   public static void main(String[] args) {
      try (InputStreamReader in = new InputStreamReader(
            new FileInputStream("utf8.txt"), StandardCharsets.UTF_8)) {
         int charRead;
         while ((charRead = in.read()) != -1) {
            System.out.print((char) charRead);
         }
      } catch (IOException ex) {
         ex.printStackTrace();
      }
   }
}
```

Key Points:

- Matches the encoding used in Exercise 6.
- Correctly decodes multi-byte characters.

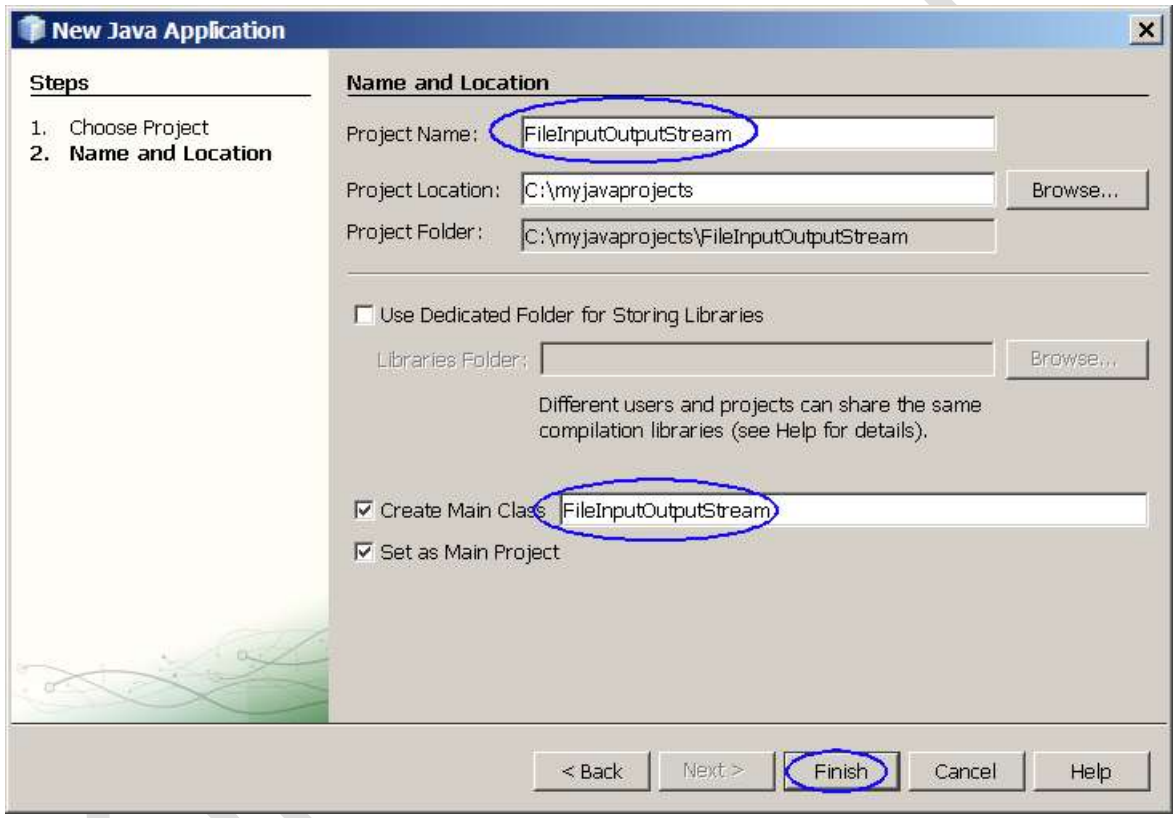
Takeaways:

- Prefer buffered I/O for performance.
- Use `try-with-resources` for automatic cleanup.
- Specify charset for international text.
- Chain streams to add functionality (buffering, parsing).

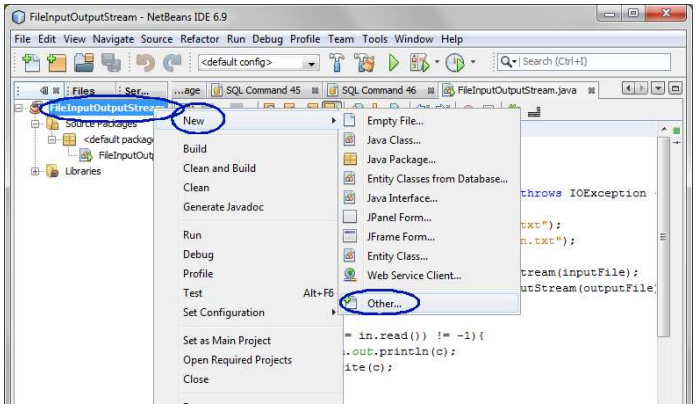
TASK: Byte Stream

Write an application that uses FileInputStream and FileOutputStream:

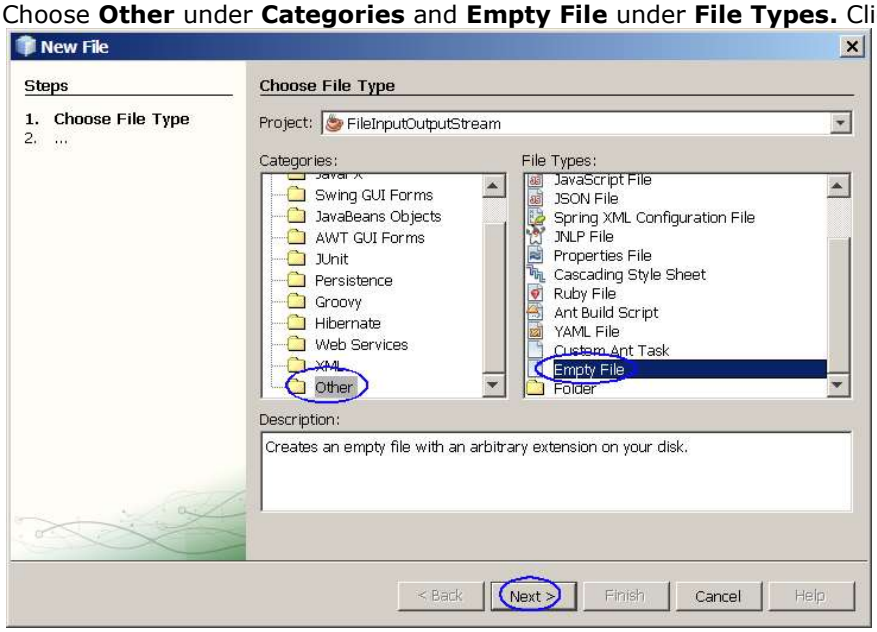
1. Create a new NetBeans project
    a. Select **File**->**New Project (Ctrl+Shift+N)**.
    b. Under **Choose Project** pane, select **Java** under **Categories** and **Java Application** under **Projects**.
    c. Click **Next**.
    d. Under **Name and Location** pane, for the **Project Name** field, type in **FileInputOutputStream** as project name. (Your Project Location may differ from the figure below).
    e. For **Create Main Class** field, type in **FileInputOutputStream**.
    f. Click **Finish**.



2. Provide **farrago.txt** as an input file. You may create this text file using any text editor program such as Notepad or using Netbeans IDE as follows:
    a. Right click **FileInputOutputStream** project and select **New->Other**.



b. Choose **Other** under **Categories** and **Empty File** under **File Types.** Click Next.



c. Observe that the **New Empty File** dialog box appears.

d. For the **File Name** field, type in **farrago.txt**. Click Finish.

e. Observe that the empty **farrago.txt** appears in the editor window. Write the contents below to the empty file:

> Subclasses of OutputStream use these methods to write data onto particular media.
>
> For instance, a FileOutputStream uses these methods to write data into a file.
>
> TelnetOutputStream uses these methods to write data onto a network connection.
>
> ByteArrayOutputStream uses these methods to write data into an expandable byte array.

3. Modify the IDE generated **FileInputOutputStream.java** as shown below.

```
   import java.io.*;

public class FileInputOutputStream {

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

        File inputFile = new File("farrago.txt");
        File outputFile = new File("outagain.txt");

        FileInputStream in = new FileInputStream(inputFile);
        FileOutputStream out = new FileOutputStream(outputFile);
        int c;

        while ((c = in.read()) != -1){
            System.out.println(c);
            out.write(c);
        }

        System.out.println("FileInputStream is used to read a file and
FileOutPutStream is used for writing.");

        in.close();
        out.close();
    }
}
```

4. Build and run the project and observe the result in the **Output** window. Explain why do you get such output?

5. Open the project directory and find the file **outagain.txt**. What is the content of the file?

6. **Re-read the codes that you have written previously in FileInputOutputStream.java and write a line of //comment at the end of each line to explain what that particular line of codes does to the program.**

7. Modify the codes so that it can display proper characters into the Output window.

8. Modify the codes to include a try-catch block.