Java Stream I/O

Most programs use data in one form or another, whether it is as input, output, or both. The sources of input and output can vary between a local file, a socket on the network, a database, variables in memory, or another program. Even the type of data can vary between objects, characters, multimedia, and others.

The Java Development Kit (JDK) provides APIs for reading and writing streams of data. These APIs have been part of the core JDK since version 1.0, but are often overshadowed by the more well-known APIs, such as JavaBeans, JFC, RMI, JDBC, and so on. However, input and output streams are the backbone of the JDK APIs, and understanding them is not only crucial, but can also make programming with them a lot of fun.

This hands-on lab takes you through the basics of using Java I/O stream.

Resources

- **Basic I/O tutorial** from java.sun.com

Exercises

Exercise 1: Byte stream

In this exercise, you will learn how to use FileInputStream class which is InputStream type and FileOutputStream class which is OutputStream type to read from and write to a file.

(1.1) Write an application that uses FileInputStream and FileOutputStream

0. Start NetBeans IDE if you have not done so yet.
1. Create a new NetBeans project

- Select **File**->**New Project (Ctrl+Shift+N)**.
- Observe that the **New Project** dialog box appears.
- Under **Choose Project** pane, select **Java** under **Categories** and **Java Application** under **Projects**.
- Click **Next**.

- Under **Name and Location** pane, for the **Project Name** field, type in **FileInputOutputStream** as project name.
- For **Create Main Class** field, type in **FileInputOutputStream**. (Figure-1.10 below)
- Click **Finish**.

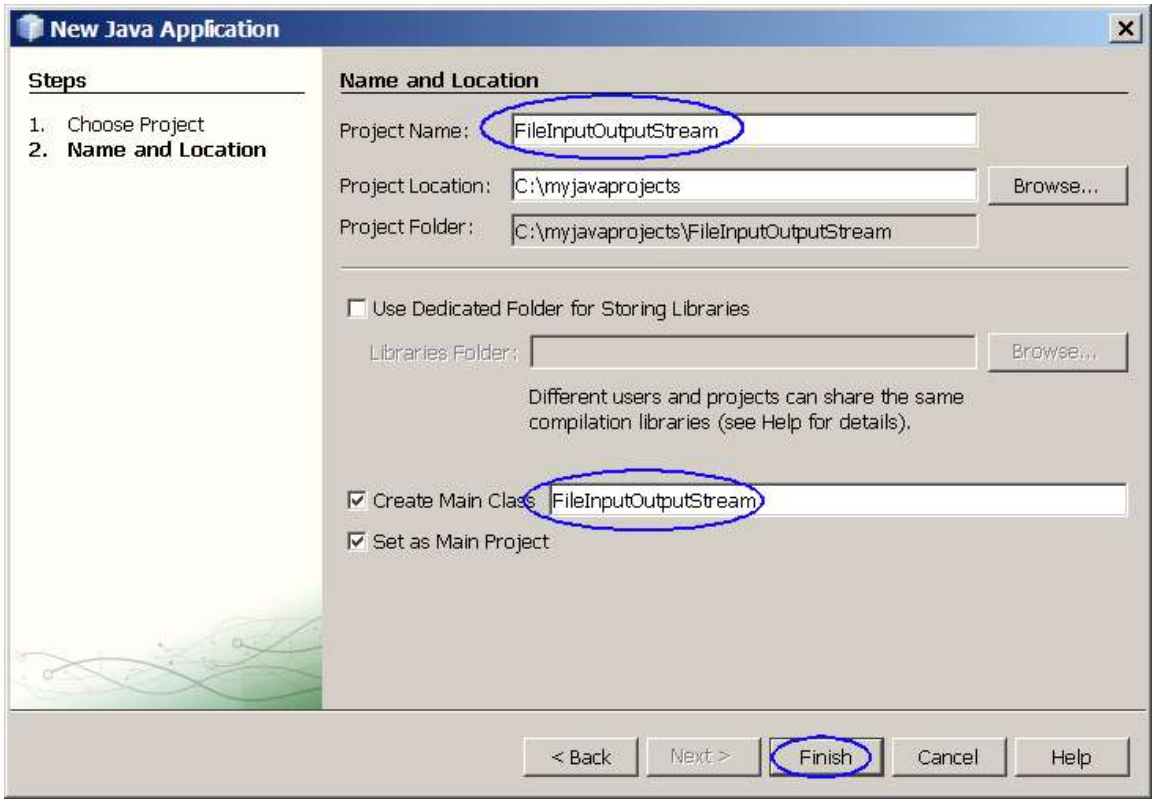


Figure-1.10: Create a new project

- Observe that **FileInputOutputStream** project appears and IDE generated **FileInputOutputStream.java** is displayed in the source editor window of NetBeans IDE.

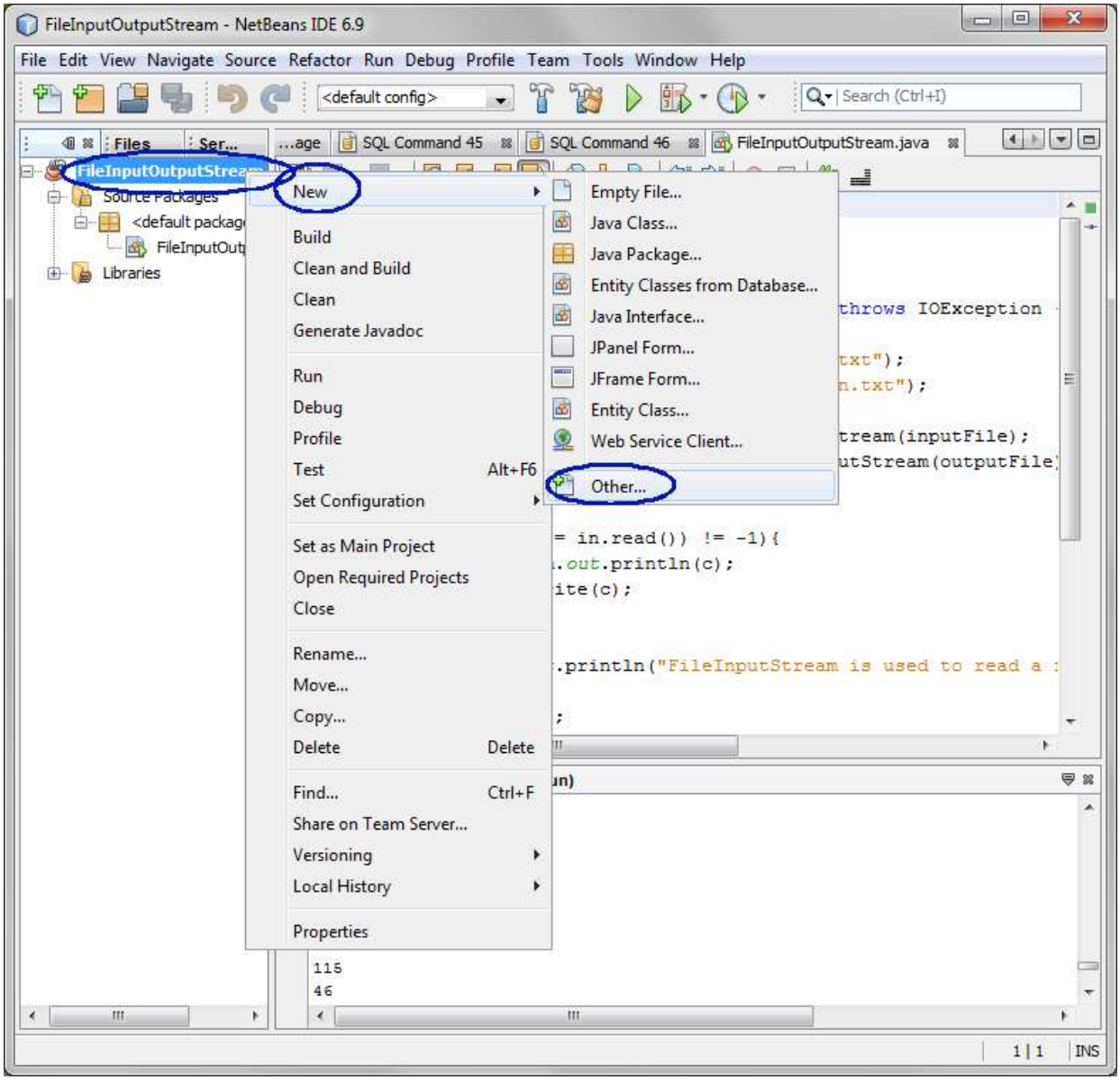
2. Modify the IDE generated **FileInputOutputStream.java** as shown in Code-1.11 below. Study the code by paying special attention to the bold fonted parts.

```
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();
   }
}
```

Code-1.11: FileInputOutputStream.java

3. Provide **farrago.txt** as an input file.

- Right click **FileInputOutputStream** project and select **New->Other**.



- Observe the **New File** dialog box appears.
- Choose **Other** under **Categories** and **Empty File** under **File Types.** (Figure-1.12 below)
- Click Next.

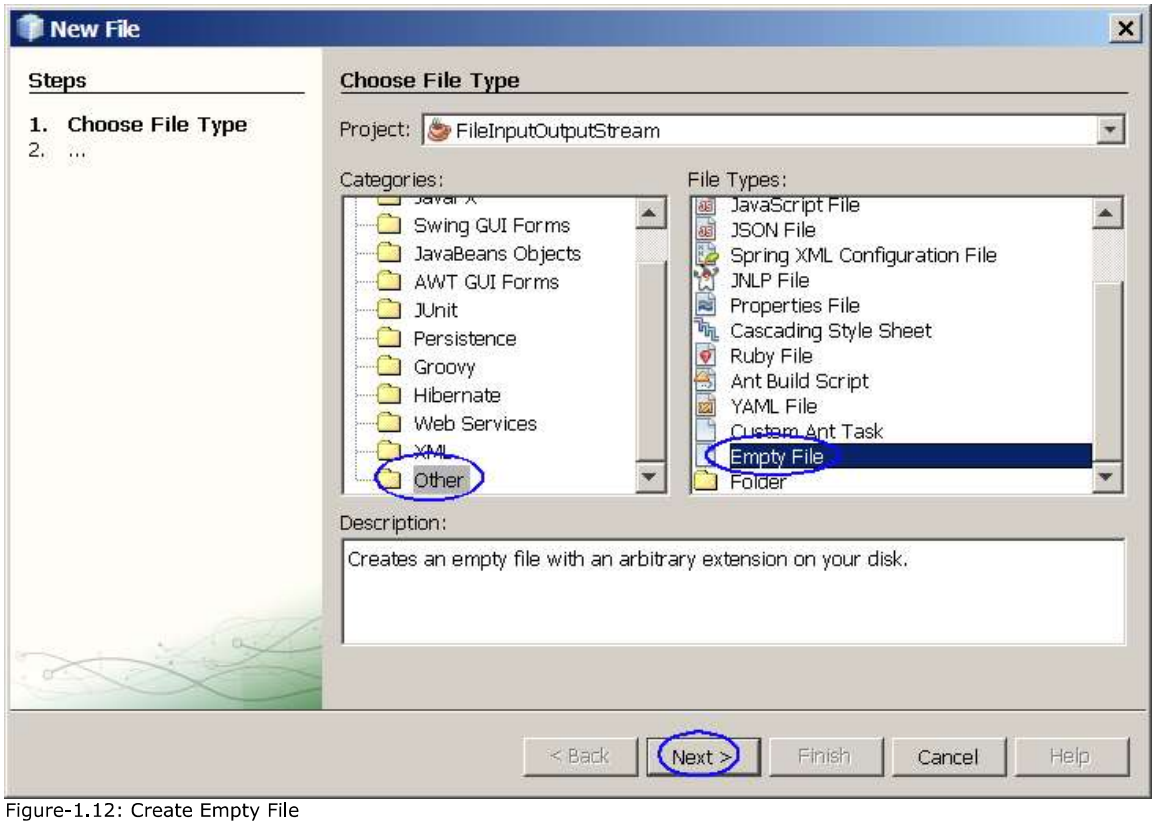
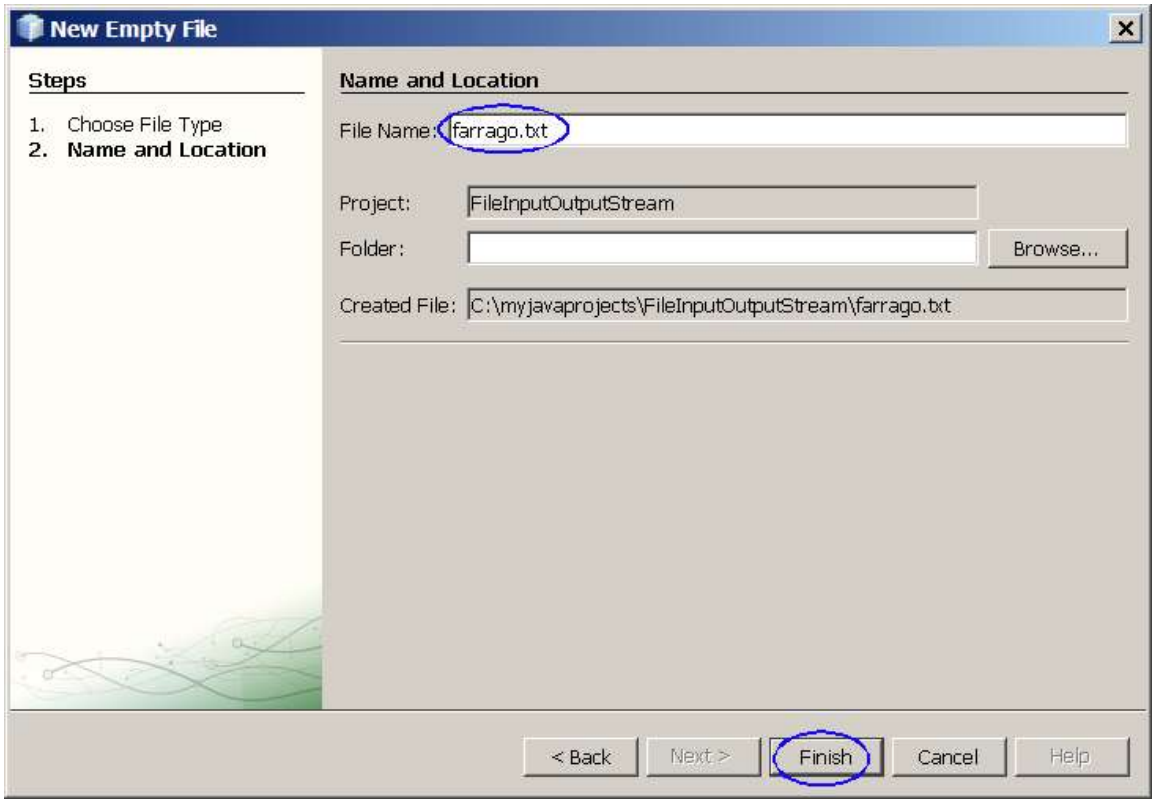


Figure-1.12: Create Empty File

- Observe that the **New Empty File** dialog box appears.
- For the **File Name** field, type in **farrago.txt**.
- Click Finsh.



- Observe that the empty **farrago.txt** appears in the editor window.
- Cut and paste the contents from the InputFile-1.14 below to the empty file.

```
So she went into the garden to cut a cabbage-leaf, to
make an apple-pie; and at the same time a great
she-bear, coming up the street, pops its head into the
shop. 'What! no soap?' So he died, and she very
imprudently married the barber; and there were
present the Picninnies, and the Joblillies, and the
```

```
Garyalies, and the grand Panjandrum himself, with the
little round button at top, and they all fell to playing
the game of catch as catch can, till the gun powder ran
out at the heels of their boots.

Samuel Foote 1720-1777
```

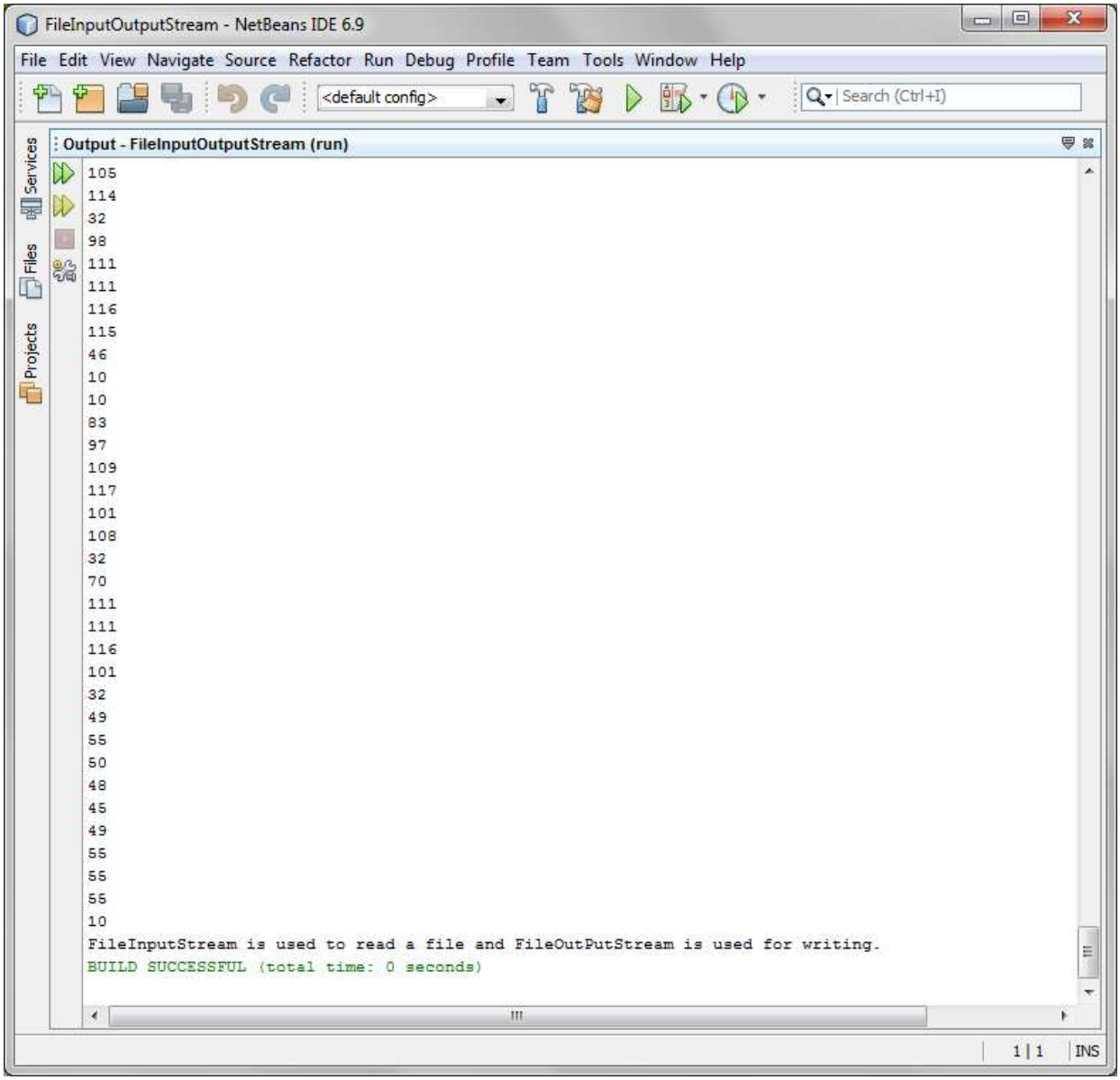
InputFile-1.14: farrago.txt

4. Build and run the project

- Right click **FileInputOutputStream** project and select **Run**.
- Observe the result in the **Output** window. (Figure-1.15 below)

```
...
55
55
55
10
FileInputStream is used to read a file and FileOutPutStream is used for writing.
```

Figure-1.15: Result of running FileInputOutputStream application



5. For your own exercise, modify **FileInputOutputStream.java** as following. Build and run the application.

- Read your own file, myowninputfile.txt, and write it to another file, myownoutputfile.txt.

return to top of the exercise

Summary

In this exercise, you learned how to use FileInputStream class which is InputStream type and FileOutputStream class which is OutputStream type to read from and write to a file.

return to the top

Exercise 2: Character Stream

In this exercise, you will learn how to use FileReaderclass which is Reader type and FileWriter class which is Writer type to read from and write to a file.

(2.1) Write an application that uses FileReader and FileWriter

1. Create a new NetBeans project

- Select **File->New Project (Ctrl+Shift+N).** The **New Project** dialog box appears.
- Under **Choose Project** pane, select **Java** under **Categories** and **Java Application** under **Projects.** Click **Next.**
- Under **Name and Location** pane, for the **Project Name** field, type in **FileReaderWriter** as project name.
- For **Create Main Class** field, type in **FileReaderWriter**.
- Click **Finish.**

- Observe that **FileReaderWriter** project appears and IDE generated **FileReaderWriter.java** is displayed in the source editor window of NetBeans IDE.

2. Modify the IDE generated **FileReaderWriter.java** as shown in Code-2.11 below. Study the code by paying special attention to the bold fonted parts.

```
import java.io.*;

public class FileReaderWriter {

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

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

      FileReader in = new FileReader(inputFile);
      FileWriter out = new FileWriter(outputFile);
      int c;

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

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

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

Code-2.11: FileReaderWriter.java

3. Provide **farrago.txt** as an input file.

```
So she went into the garden to cut a cabbage-leaf, to
make an apple-pie; and at the same time a great
she-bear, coming up the street, pops its head into the
shop. 'What! no soap?' So he died, and she very
imprudently married the barber; and there were
present the Picninnies, and the Joblillies, and the
Garyalies, and the grand Panjandrum himself, with the
little round button at top, and they all fell to playing
the game of catch as catch can, till the gun powder ran
out at the heels of their boots.

Samuel Foote 1720-1777
```

File-2.12: farrago.txt

4. Build and run the project

- Right click **FileReaderWriter** project and select **Run.**
- Observe the result in the **Output** window. (Figure-2.13 below)

```
...
55
55
```

```
55
10
FileReader is used to read a file and FileWriter is used for writing.
```

Figure-2.13: Result of running FileReaderWriter application

Solution: This exercise up to this point is provided as a ready-to-open-and-run NetBeans project as part of hands-on lab zip file. You can find it as **<LAB_UNZIPPED_DIRECTORY>/javase_iostream/samples/FileReaderWriter.** You can just open it and run it.

5. For your own exercise, modify **FileReaderWriter.java** as following. Build and run the application. (This is not a homework. No submission is required.)

- Read your own file, myowninputfile.txt, and write it to another file, myownoutputfile.txt.

return to top of the exercise

Summary

In this exercise, you learned how to use FileReaderclass which is Reader type and FileWriter class which is Writer type to read from and write to a file.

return to the top

Exercise 3: Buffered Stream

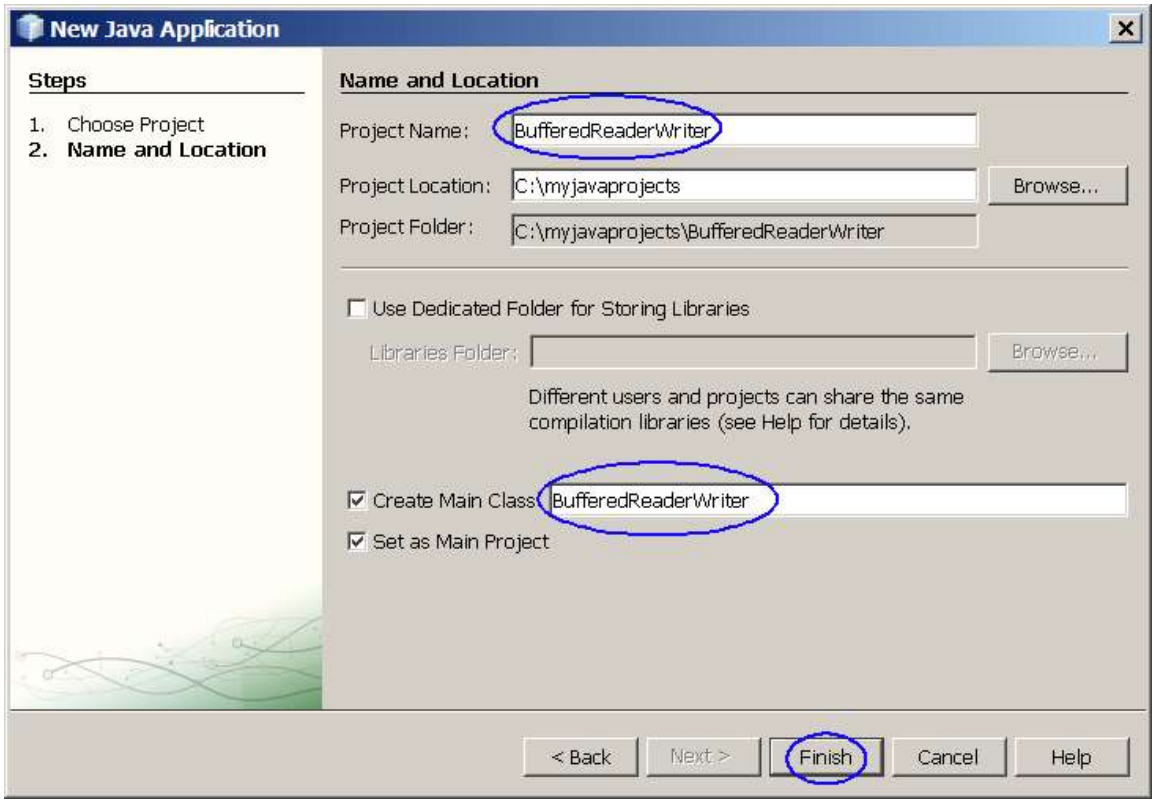
In this exercise, you will learn how to use BufferedReader and BufferedWriter classes for performing buffered I/O operations.

(3.1) Write an application that uses BufferedReader and BufferedWriter classes

1. Create a new NetBeans project

- Select **File->New Project (Ctrl+Shift+N).** The **New Project** dialog box appears.
- Under **Choose Project** pane, select **Java** under **Categories** and **Java Application** under **Projects.**
- Click **Next.**

- Under **Name and Location** pane, for the **Project Name** field, type in **BufferedReaderWriter** as project name.
- For **Create Main Class** field, type in **BufferedReaderWriter.**
- Click **Finish.**



- Observe that **BufferedReaderWriter** project appears and IDE generated **BufferedReaderWriter.java** is displayed in the source editor window of NetBeans IDE.

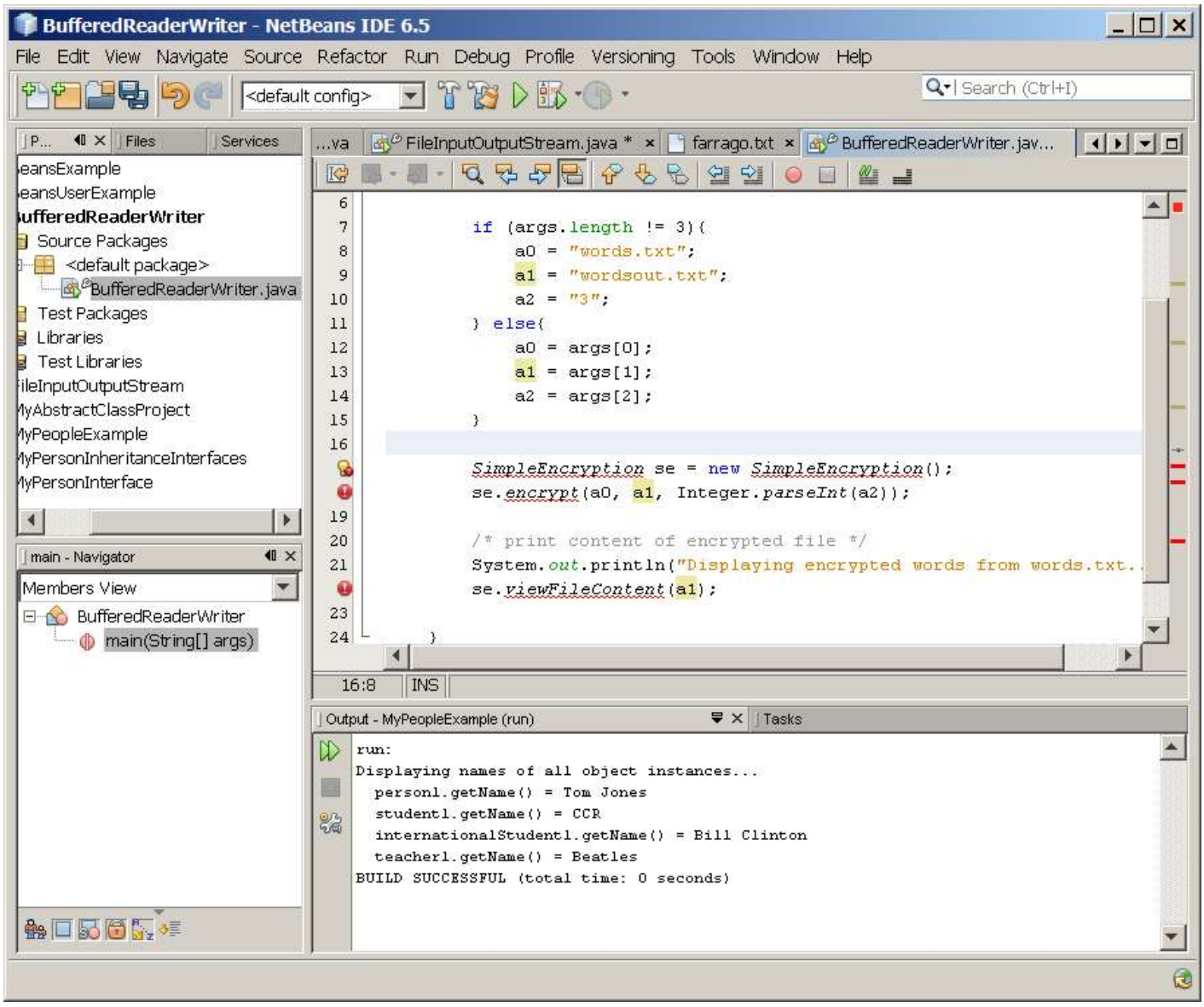
2. Modify the IDE generated **BufferedReaderWriter.java** as shown in Code-3.11 below. Study the code by paying special attention to the bold fonted parts.

```
public class BufferedReaderWriter {
```

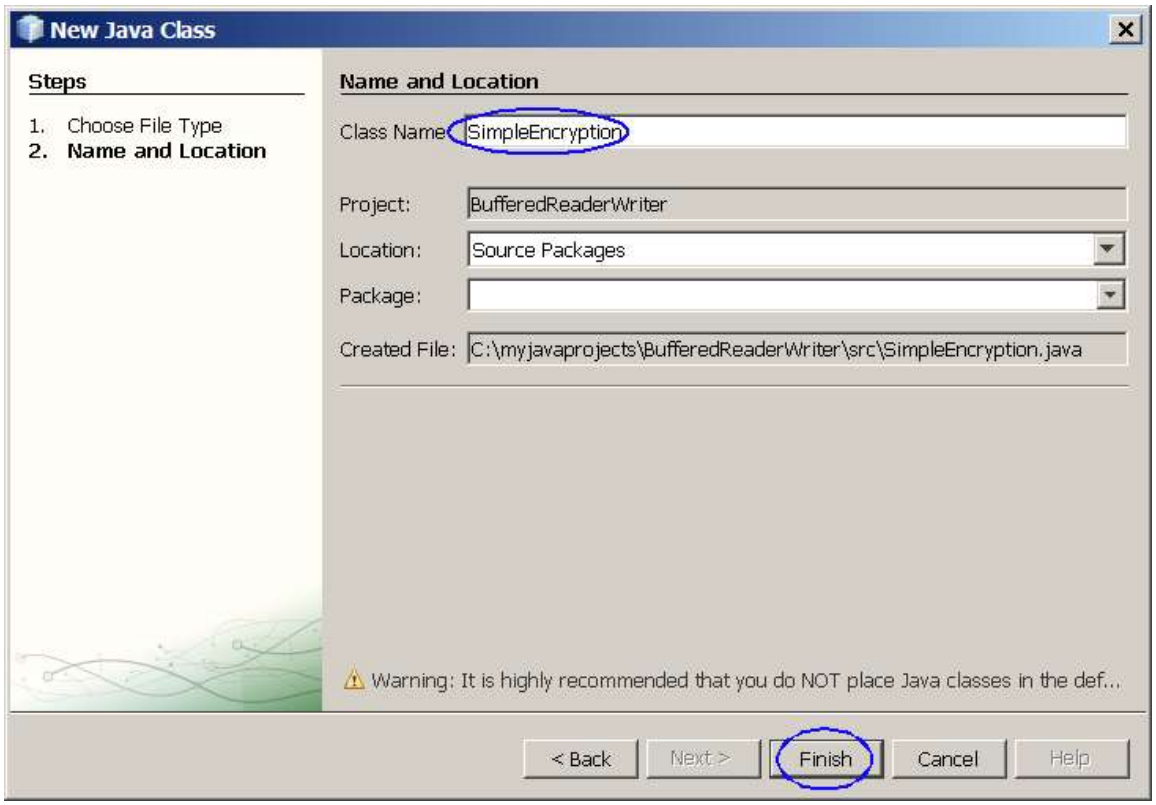
```
   public static void main(String args[]) {

      String a0, a1, a2;

      if (args.length != 3){
         a0 = "words.txt";
         a1 = "wordsout.txt";
         a2 = "3";
      } else{
         a0 = args[0];
         a1 = args[1];
         a2 = args[2];
      }

      SimpleEncryption se = new SimpleEncryption();
      se.encrypt(a0, a1, Integer.parseInt(a2));

      /* print content of encrypted file */
      System.out.println("Displaying encrypted words from words.txt...");
      se.viewFileContent(a1);

   }
}
```

Code-3.11: BufferedReaderWriter.java

- Observe that there are some compile errors. This is expected.



3. Write **SimpleEncryption.java** as shown in Code-3.12 below. Study the code by paying special attention to the bold fonted parts.



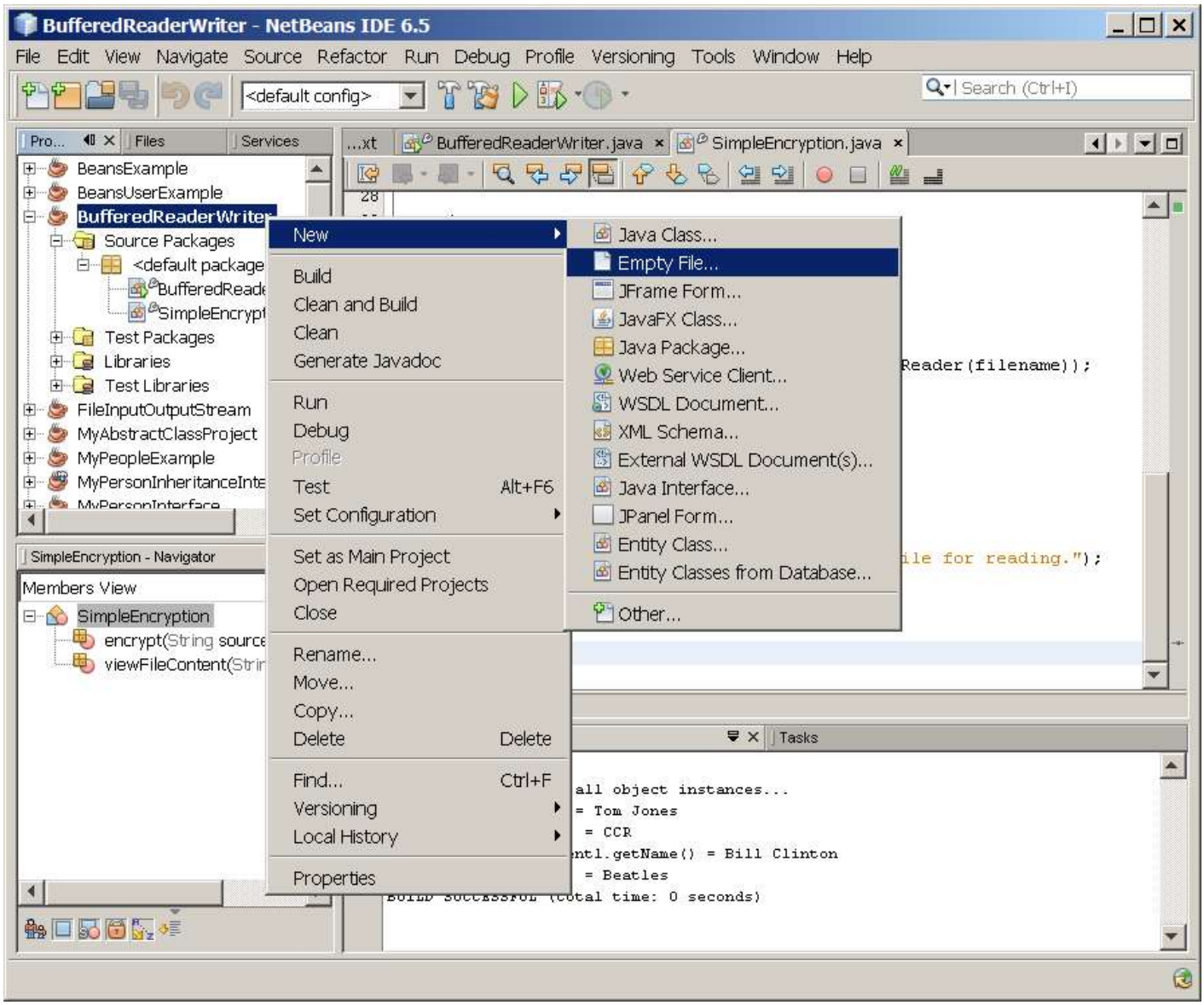
- Modify the code as shown below.

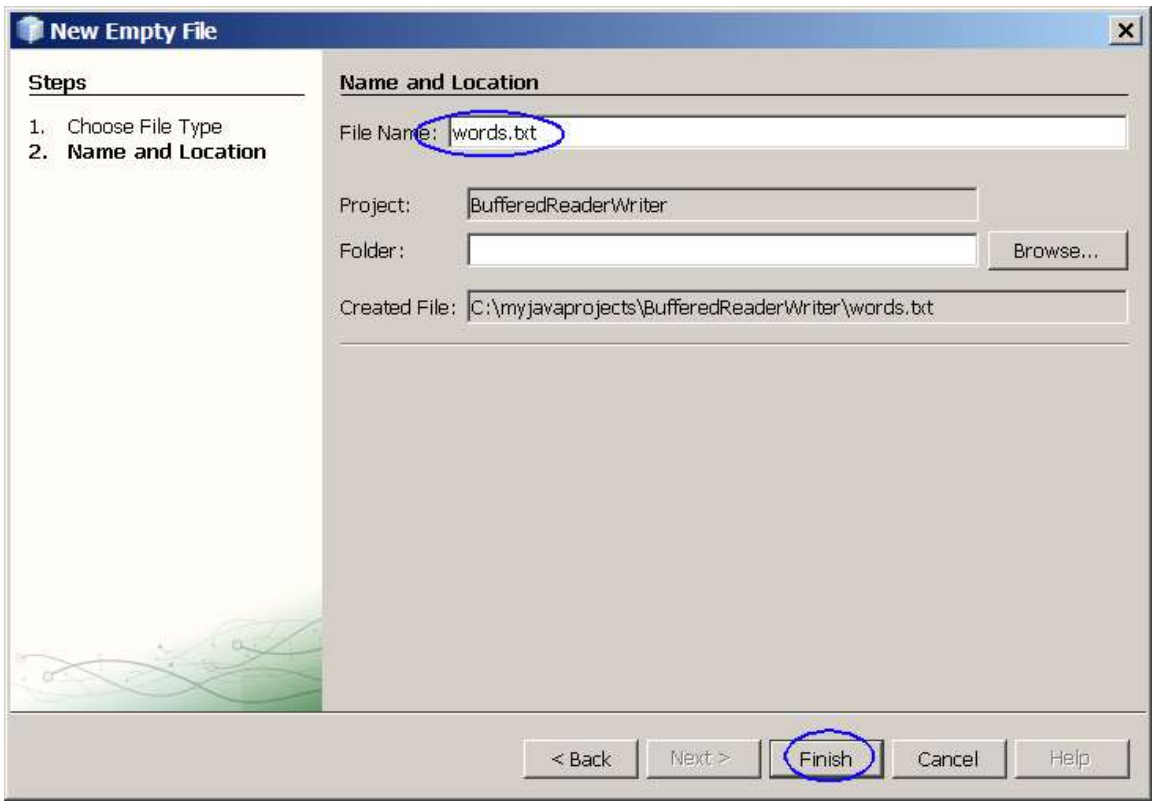
```
import java.io.*;

class SimpleEncryption {

  void encrypt(String source, String dest, int shiftSize) {

    BufferedReader reader;
    BufferedWriter writer;

    try {
      reader = new BufferedReader(new FileReader(source));
      writer = new BufferedWriter(new FileWriter(dest));
      String line = reader.readLine();
      int data;
      while (line != null) {
        for (int i = 0; i < line.length(); i++) {
          data =(int) line.charAt(i) + shiftSize;
          writer.write(data);
        }
        writer.write((int)'\n');
        line = reader.readLine();
      }
      reader.close();
      writer.close();
    } catch (IOException ie) {
      System.out.println("Failed encrypting the file content.");
    }

  }

  void viewFileContent(String filename) {

    BufferedReader reader;
    try {
      reader = new BufferedReader(new FileReader(filename));
      String line = reader.readLine();
      while (line != null) {
        System.out.println(line);
        line = reader.readLine();
      }
      reader.close();
    } catch (IOException ie) {
      System.out.println("Failed to open file for reading.");
    }
  }
```

```
}
```

Code-3.12: SimpleEncryption.java

4. Write **words.txt.** (File-3.13 below)



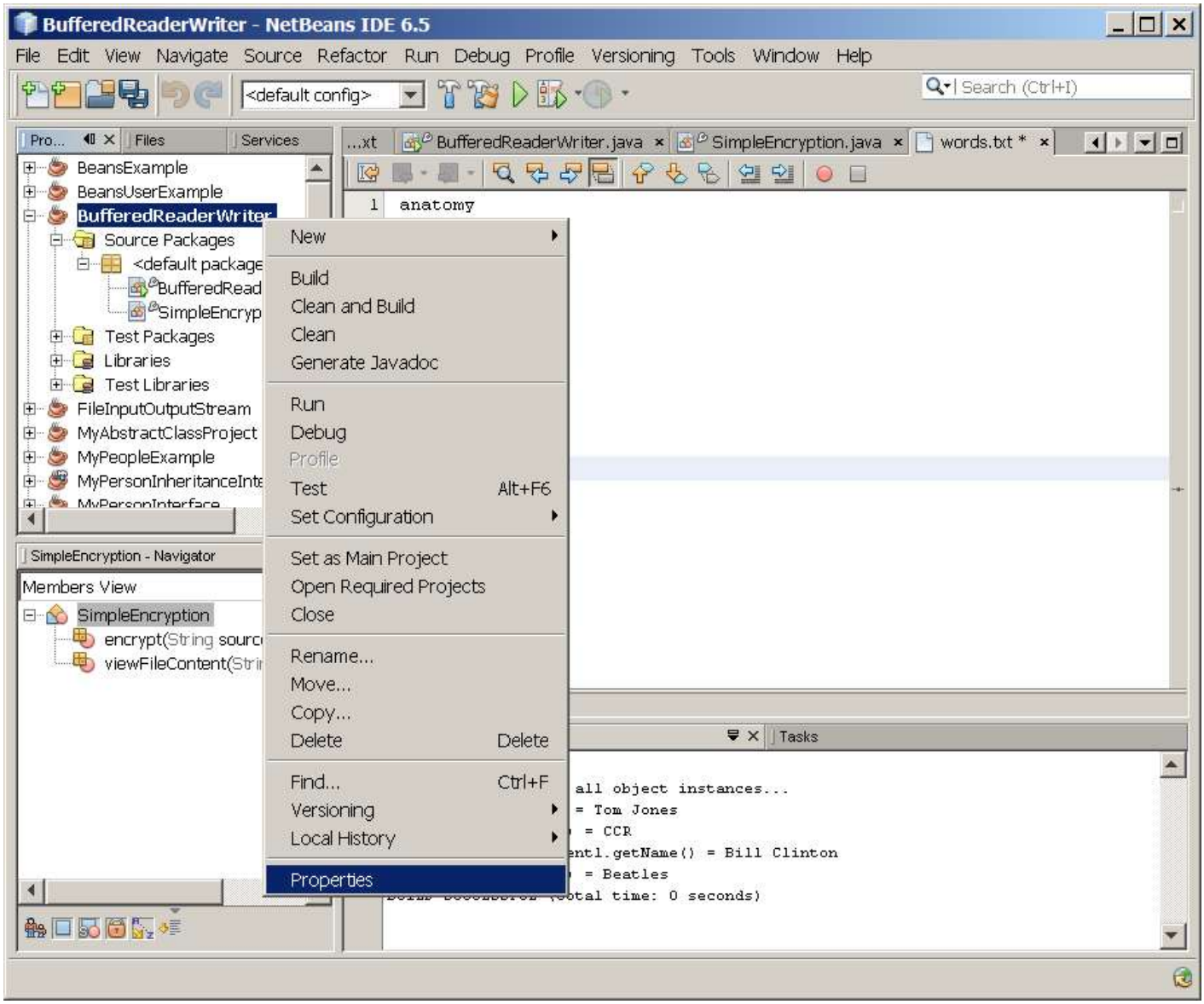



```
anatomy
animation
applet
application
argument
bolts
class
communicate
string
threads
tools
user
```

File-3.13: words.txt

5. Provide command line arguments.

- Right click **BufferedReaderWriter** project and select **Properties**.



- Observe **Project Properties** dialog box appears.
- Select Run and for the Arguments field, type in **words.txt wordsout.txt 3.** (Figure-3.14 below)

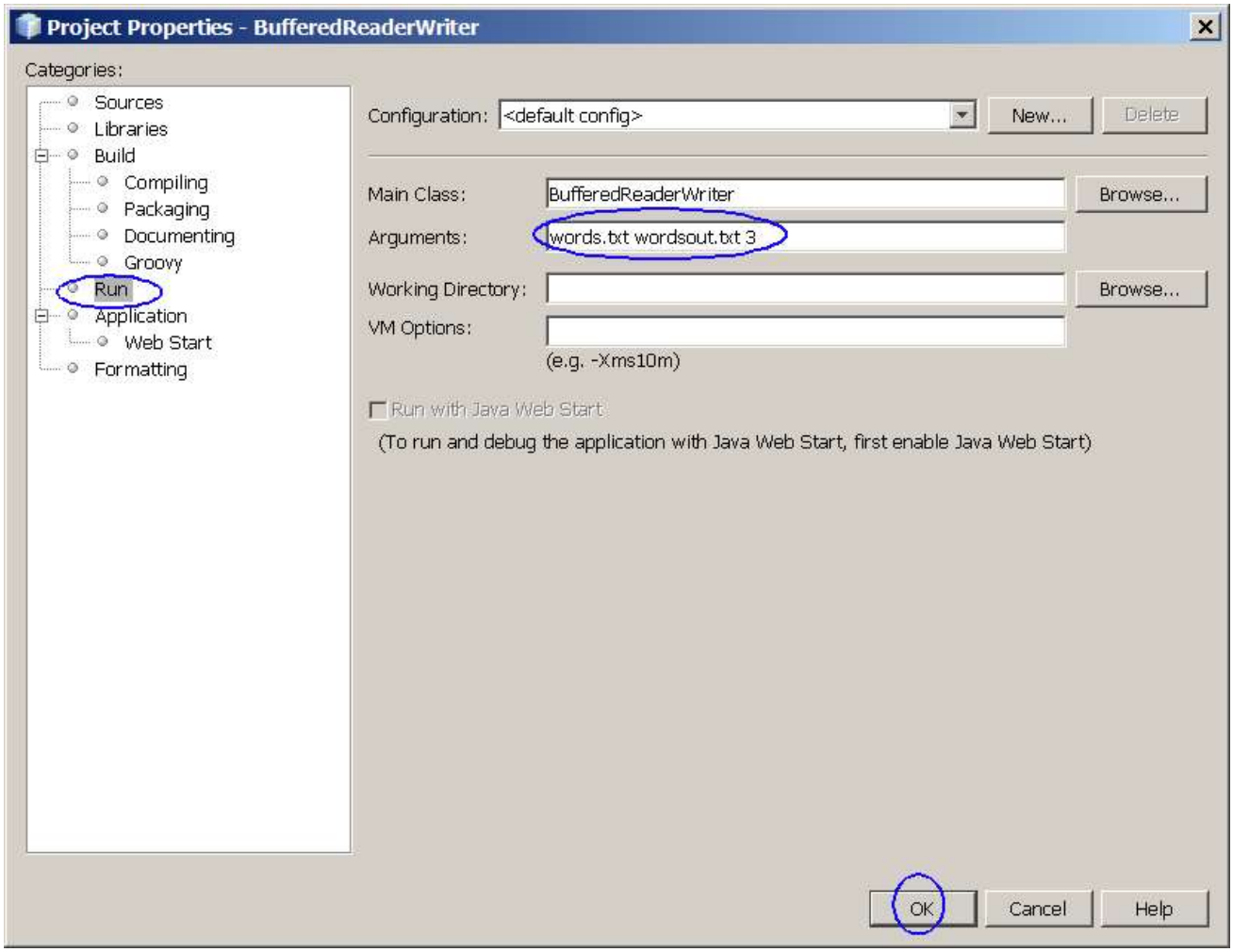


Figure-3.14: Provide command line arguments

6. Build and run the project

- Right click **BufferedReaderWriter** project and select **Run**.
- Observe the result in the **Output** window. (Figure-3.15 below)

```
Displaying encrypted words from words.txt...
dqdwrp|
dqlpdwlrq
dssohw
dssolfdwlrq
dujxphqw
erowv
fodvv
frppxqlfdwh
vwulqj
wkuhdgv
wrrov
xvhu
```

Figure-3.15: Result of running BufferedReaderWriter application

return to top of the exercise

Summary

In this exercise, you learned how to use BufferedReader and BufferedWriter classes for performing buffered I/O operations.

return to the top

Exercise 4: Data Stream

In this exercise, you are going to learn how to use DataInputStream and DataOutputStream classes for reading and writing primitive data types.

1. **DataInputStream and DataOutputStream**
2. **DataInput and DataOut**

(4.1) DataInputStream and DataOutputStream

1. Create a new NetBeans project

- Select **File->New Project (Ctrl+Shift+N).** The **New Project** dialog box appears.
- Under **Choose Project** pane, select **Java** under **Categories** and **Java Application** under **Projects.** Click **Next.**
- Under **Name and Location** pane, for the **Project Name** field, type in **DataInputOutputStream** as project name.
- For **Create Main Class** field, type in **DataInputOutputStream**.
- Click **Finish.**

- Observe that **DataInputOutputStream** project appears and IDE generated **DataInputOutputStream.java** is displayed in the source editor window of NetBeans IDE.

2. Modify the IDE generated **DataInputOutputStream.java** as shown in Code-4.11 below. Study the code especially the code fragments that are in bold-fonted font.

```
import java.io.*;

public class DataInputOutputStream {

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

      // write the data out
      DataOutputStream out = new DataOutputStream(new
            FileOutputStream("invoice"));

      double[] prices = { 19.99, 9.99, 15.99, 3.99, 4.99 };
      int[] units = { 12, 8, 13, 29, 50 };
      String[] descs = { "Java T-shirt",
      "Java Mug",
      "Duke Juggling Dolls",
      "Java Pin",
      "Java Key Chain" };

      for (int i = 0; i < prices.length; i ++) {
         out.writeDouble(prices[i]);
         out.writeChar('\t');
         out.writeInt(units[i]);
         out.writeChar('\t');
         out.writeChars(descs[i]);
         out.writeChar('\n');
      }
      out.close();

      // read it in again
      DataInputStream in = new DataInputStream(new
            FileInputStream("invoice"));

      double price;
      int unit;
      StringBuffer desc;
      double total = 0.0;

      String lineSepString = System.getProperty("line.separator");
      char lineSep = lineSepString.charAt(lineSepString.length()-1);

      try {
         while (true) {
            price = in.readDouble();
            in.readChar();       // throws out the tab
            unit = in.readInt();
            in.readChar();       // throws out the tab
            char chr;
            desc = new StringBuffer(20);
            while ((chr = in.readChar()) != lineSep)
               desc.append(chr);
            System.out.println("You've ordered " +
                  unit + " units of " +
                  desc + " at $" + price);
            total = total + unit * price;
         }
      } catch (EOFException e) {
      }
      System.out.println("For a TOTAL of: $" + total);
      in.close();
   }
}
```

Code-4.11: DataInputOutputStream.java

5. Build and run the project

- Right click **DataInputOutputStream** project and select **Run**.
- Observe the result in the **Output** window. (Figure-4.14 below)

```
You've ordered 12 units of Java T-shirt at $19.99
You've ordered 8 units of Java Mug at $9.99
You've ordered 13 units of Duke Juggling Dolls at $15.99
You've ordered 29 units of Java Pin at $3.99
You've ordered 50 units of Java Key Chain at $4.99
For a TOTAL of: $892.8800000000001
```

Figure-4.14: Result of running DataInputOutputStream application

return to top of the exercise

(4.2) DataInput and DataOutput

In this step, you are going to exercise a concept of a stream processor that receives an input stream and perferms a checksum operation against the data.

1. Create a new NetBeans project

- Select **File->New Project (Ctrl+Shift+N)**. The **New Project** dialog box appears.
- Under **Choose Project** pane, select **Java** under **Categories** and **Java Application** under **Projects**. Click **Next**.
- Under **Name and Location** pane, for the **Project Name** field, type in **CustomDataInputOutput** as project name.
- For **Create Main Class** field, type in **CustomDataInputOutput**.
- Click **Finish**.

- Observe that **CustomDataInputOutput** project appears and IDE generated **CustomDataInputOutput.java** is displayed in the source editor window of NetBeans IDE.

2. Modify the IDE generated **CustomDataInputOutput.java** as shown in Code-4.21 below. Study the code especially the code fragments that are in bold-fonted font.

```
import java.io.*;

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

      Adler32 inChecker = new Adler32();
      Adler32 outChecker = new Adler32();
      CheckedDataInput in = null;
      CheckedDataOutput out = null;

      try {
         in = new CheckedDataInput(
            new DataInputStream(
                  new FileInputStream("farrago.txt")),
            inChecker);
         out = new CheckedDataOutput(
             new DataOutputStream(
                   new FileOutputStream("outagain.txt")),
             outChecker);
      } catch (FileNotFoundException e) {
         System.err.println("CheckedIOTest: " + e);
         System.exit(-1);
      } catch (IOException e) {
         System.err.println("CheckedIOTest: " + e);
         System.exit(-1);
      }

      boolean EOF = false;

      while (!EOF) {
         try {
            int c = in.readByte();
            out.write(c);
         } catch (EOFException e) {
            EOF = true;
         }
      }

      System.out.println("Input stream check sum: " +
            in.getChecksum().getValue());
      System.out.println("Output stream check sum: " +
            out.getChecksum().getValue());
   }
}
```

Code-4.21: CustomDataInputOutput.java

3. Write **CheckedDataInput.java**. (Code-4.22 below)

```
import java.io.*;

public class CheckedDataInput {

   private Checksum cksum;
   private DataInput in;

   public CheckedDataInput(DataInput in, Checksum cksum) {
      this.cksum = cksum;
      this.in = in;
   }

   public byte readByte() throws IOException {
      byte b = in.readByte();
      cksum.update(b);
      return b;
   }

   public void readFully(byte[] b) throws IOException {
      in.readFully(b, 0, b.length);
      cksum.update(b, 0, b.length);
   }

   public void readFully(byte[] b, int off, int len) throws IOException {
      in.readFully(b, off, len);
      cksum.update(b, off, len);
   }

   public Checksum getChecksum() {
      return cksum;
   }
}
```

Code-4.22: CheckedDataInput.java

4. Write **CheckedDataOutput.java**. (Code-4.23 below)

```
import java.io.*;

public class CheckedDataOutput {

   private Checksum cksum;
   private DataOutput out;

   public CheckedDataOutput(DataOutput out, Checksum cksum) {
      this.cksum = cksum;
      this.out = out;
   }

   public void write(int b) throws IOException {
      out.write(b);
      cksum.update(b);
   }

   public void write(byte[] b) throws IOException {
      out.write(b, 0, b.length);
      cksum.update(b, 0, b.length);
   }

   public void write(byte[] b, int off, int len) throws IOException {
      out.write(b, off, len);
      cksum.update(b, off, len);
   }

   public Checksum getChecksum() {
      return cksum;
   }
}
```

Code-4.23: CheckedDataOutput.java

5. Write **Checksum.java**. (Code-4.24 below)

```
public interface Checksum {
   /**
    * Updates the current checksum with the specified byte.
    */
   public void update(int b);
```

```
    /**
     * Updates the current checksum with the specified array of bytes.
     */
    public void update(byte[] b, int off, int len);

    /**
     * Returns the current checksum value.
     */
    public long getValue();

    /**
     * Resets the checksum to its initial value.
     */
    public void reset();
}
```

Code-4.24: Checksum.java

6. Write **Adler32.java**. (Code-4.25 below)

```
public class Adler32 implements Checksum {
    private int value = 1;

    /*
     * BASE is the largest prime number smaller than 65536
     * NMAX is the largest n such that 255n(n+1)/2 + (n+1)(BASE-1) <= 2^32-1
     */
    private static final int BASE = 65521;
    private static final int NMAX = 5552;

    /**
     * Update current Adler-32 checksum given the specified byte.
     */
    public void update(int b) {
        int s1 = value & 0xffff;
        int s2 = (value >> 16) & 0xffff;
        s1 += b & 0xff;
        s2 += s1;
        value = ((s2 % BASE) << 16) | (s1 % BASE);
    }

    /**
     * Update current Adler-32 checksum given the specified byte array.
     */
    public void update(byte[] b, int off, int len) {
        int s1 = value & 0xffff;
        int s2 = (value >> 16) & 0xffff;

        while (len > 0) {
            int k = len < NMAX ? len : NMAX;
            len -= k;
            while (k-- > 0) {
                s1 += b[off++] & 0xff;
                s2 += s1;
            }
            s1 %= BASE;
            s2 %= BASE;
        }
        value = (s2 << 16) | s1;
    }

    /**
     * Reset Adler-32 checksum to initial value.
     */
    public void reset() {
        value = 1;
    }

    /**
     * Returns current checksum value.
     */
    public long getValue() {
        return (long)value & 0xffffffff;
    }
}
```

Code-4.25: Adler32.java

7. Provide **farrago.txt** as an input file.

```
So she went into the garden to cut a cabbage-leaf, to
make an apple-pie; and at the same time a great
she-bear, coming up the street, pops its head into the
```

```
shop. 'What! no soap?' So he died, and she very
imprudently married the barber; and there were
present the Picninnies, and the Joblillies, and the
Garyalies, and the grand Panjandrum himself, with the
little round button at top, and they all fell to playing
the game of catch as catch can, till the gun powder ran
out at the heels of their boots.

Samuel Foote 1720-1777
```

File-2.12: farrago.txt

8. Build and run the project

- Right click **CustomDataInputOutput** project and select **Run**.
- Observe the result in the **Output** window. (Figure-4.14 below)

```
Input stream check sum: 736868089
Output stream check sum: 736868089
```

Figure-4.14: Result

return to top of the exercise

Summary

In this exercise, you learned how to use DataInputStream and DataOutputStream classes for reading and writing primitive data types.

return to the top

Exercise 5: Object Stream

In this exercise, you are going to learn how to use ObjectInputStream and ObjectOutputStream classes for reading and writing objects.

(5.1) Write an application that uses ObjectInputStream and ObjectOutputStream classes

1. Create a new NetBeans project

- Select **File->New Project (Ctrl+Shift+N)**. The **New Project** dialog box appears.
- Under **Choose Project** pane, select **Java** under **Categories** and **Java Application** under **Projects**. Click **Next**.
- Under **Name and Location** pane, for the **Project Name** field, type in **ObjectInputOutputStream** as project name.
- For **Create Main Class** field, type in **ObjectInputOutputStream**.
- Click **Finish**.

- Observe that **ObjectInputOutputStream** project appears and IDE generated **ObjectInputOutputStream.java** is displayed in the source editor window of NetBeans IDE.

2. Modify the IDE generated **ObjectInputOutputStream.java** as shown in Code-5.11 below.

```
import java.io.*;

public class ObjectInputOutputStream {

   public static void main(String[] args) {

      Card3 card = new Card3(12, Card3.SPADES);
      System.out.println("Card to write is: " + card);

      try {
         FileOutputStream out = new FileOutputStream("card.out");
         ObjectOutputStream oos = new ObjectOutputStream(out);
         oos.writeObject(card);
         oos.flush();
      } catch (Exception e) {
         System.out.println("Problem serializing: " + e);
      }

      try {
         FileInputStream in = new FileInputStream("card.out");
         ObjectInputStream ois = new ObjectInputStream(in);
         card = (Card3)(ois.readObject());
      } catch (Exception e) {
         System.out.println("Problem serializing: " + e);
      }

      System.out.println("Card read is: " + card);
   }
}
```

Code-5.11: ObjectInputOutputStream.java

3. Write **Card3.java** as shown in Code-5.12 below. Study the code by paying special attention to the bold fonted parts.

```
import java.io.Serializable;

public class Card3 implements Serializable {
   private int suit = UNASSIGNED;
   private int number = UNASSIGNED;

   public final static int UNASSIGNED = -1;

   public final static int DIAMONDS = 1;
   public final static int CLUBS = 2;
   public final static int HEARTS = 3;
   public final static int SPADES = 4;

   public final static int ACE = 1;
   public final static int KING = 13;

   public Card3(int number, int suit) {
      if (isValidNumber(number)) {
         this.number = number;
      } else {
         //Error
      }

      if (isValidSuit(suit)) {
         this.suit = suit;
      } else {
         //Error
      }
   }

   public int getSuit() {
      return suit;
   }

   public int getNumber() {
      return number;
   }

   public static boolean isValidNumber(int number) {
      if (number >= ACE && number <= KING) {
         return true;
      } else {
         return false;
      }
   }
   public static boolean isValidSuit(int suit) {
      if (suit >= DIAMONDS && suit <= SPADES) {
         return true;
      } else {
         return false;
      }
   }

   public boolean equals(Object obj) {
      if (obj instanceof Card3) {
      Card3 card = (Card3)obj;
         if (card.getNumber() == this.number && card.getSuit() == this.suit) {
            return true;
         } else {
            return false;
         }
      } else {
         return false;
      }
   }
   public int hashCode() {
      return number * suit;
   }
   public String toString() {
      return numberToString(this.number) + " of "
           + suitToString(this.suit);
   }

   public static String numberToString(int number) {
      String result = "";
      switch (number) {
         case ACE: result =  "Ace"; break;
         case 2: result = "Two"; break;
```

```
            case 3: result = "Three"; break;
            case 4: result = "Four"; break;
            case 5: result = "Five"; break;
            case 6: result = "Six"; break;
            case 7: result = "Seven"; break;
            case 8: result = "Eight"; break;
            case 9: result = "Nine"; break;
            case 10: result = "Ten"; break;
            case 11: result = "Jack"; break;
            case 12: result = "Queen"; break;
            case KING: result = "King"; break;
            case UNASSIGNED: result = "Invalid Number"; break;
        }
        return result;
    }

    public static String suitToString(int suit) {
        String result = "";
        switch (suit) {
            case DIAMONDS: result = "Diamonds"; break;
            case CLUBS: result = "Clubs"; break;
            case HEARTS: result = "Hearts"; break;
            case SPADES: result = "Spades"; break;
            case UNASSIGNED: result = "Invalid Suit"; break;
        }
        return result;
    }
}
```

Code-5.12: Card3.java

4. Build and run the project

- Right click **ObjectInputOutputStream** project and select **Run**.
- Observe the result in the **Output** window. (Figure-5.15 below)

```
Card to write is: Queen of Spades
Card read is: Queen of Spades
```

Figure-5.15: Result

return to top of the exercise

Summary

In this exercise, you learned how to use ObjectInputStream and ObjectOutputStream classes for reading and writing objects.

return to the top

Exercise 6: Piped Stream

In this exercise, you will learn how to use PipedReader and PipedWriter classes.

(6.1) Pipe stream

1. Create a new NetBeans project

- Select **File->New Project (Ctrl+Shift+N)**. The **New Project** dialog box appears.
- Under **Choose Project** pane, select **Java** under **Categories** and **Java Application** under **Projects**. Click **Next**.
- Under **Name and Location** pane, for the **Project Name** field, type in **PipedReaderWriter** as project name.
- For **Create Main Class** field, type in **PipedReaderWriter**.
- Click **Finish**.

- Observe that **PipedReaderWriter** project appears and IDE generated **PipedReaderWriter.java** is displayed in the source editor window of NetBeans IDE.

2. Modify the IDE generated **PipedReaderWriter.java** as shown in Code-6.11 below. Study the code by paying special attention to the bold fonted parts.

```
import java.io.*;

public class PipedReaderWriter {

    public static Reader reverse(Reader source) throws IOException {

        BufferedReader in = new BufferedReader(source);

        PipedWriter pipeOut = new PipedWriter();
        PipedReader pipeIn = new PipedReader(pipeOut);
        PrintWriter out = new PrintWriter(pipeOut);
```

```
        new ReverseThread(out, in).start();

        return pipeIn;
    }

    public static Reader sort(Reader source) throws IOException {

        BufferedReader in = new BufferedReader(source);

        PipedWriter pipeOut = new PipedWriter();
        PipedReader pipeIn = new PipedReader(pipeOut);
        PrintWriter out = new PrintWriter(pipeOut);

        new SortThread(out, in).start();

        return pipeIn;
    }

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

        FileReader words = new FileReader("words.txt");

        // do the reversing and sorting
        Reader rhymedWords = reverse(sort(reverse(words)));

        // write new list to standard out
        BufferedReader in = new BufferedReader(rhymedWords);
        String input;

        while ((input = in.readLine()) != null)
            System.out.println(input);
        in.close();
    }

}
```

Code-6.11: PipedReaderWriter.java

3. Write **ReverseThread.java**.

```
import java.io.*;

public class ReverseThread extends Thread {
    private PrintWriter out = null;
    private BufferedReader in = null;

    public ReverseThread(PrintWriter out, BufferedReader in) {
        this.out = out;
        this.in = in;
    }

    public void run() {
        if (out != null && in != null) {
            try {
                String input;
                while ((input = in.readLine()) != null) {
                    out.println(reverseIt(input));
                    out.flush();
                }
                out.close();
            } catch (IOException e) {
                System.err.println("ReverseThread run: " + e);
            }
        }
    }

    private String reverseIt(String source) {
        int i, len = source.length();
        StringBuffer dest = new StringBuffer(len);

        for (i = (len - 1); i >= 0; i--)
            dest.append(source.charAt(i));
        return dest.toString();
    }
}
```

Code-6.12: ReserveThread.java

4. Write **SortThread.java**.

```
import java.io.*;
```

```
public class SortThread extends Thread {
   private PrintWriter out = null;
   private BufferedReader in = null;

   public SortThread(PrintWriter out, BufferedReader in) {
      this.out = out;
      this.in = in;
   }

   public void run() {
      int MAXWORDS = 50;

      if (out != null && in != null) {
         try {
            String[] listOfWords = new String[MAXWORDS];
            int numwords = 0;

            while ((listOfWords[numwords] = in.readLine()) != null)
               numwords++;
            quicksort(listOfWords, 0, numwords-1);
            for (int i = 0; i < numwords; i++)
               out.println(listOfWords[i]);
            out.close();
         } catch (IOException e) {
            System.err.println("SortThread run: " + e);
         }
      }
   }

   private static void quicksort(String[] a, int lo0, int hi0) {
      int lo = lo0;
      int hi = hi0;

      if (lo >= hi)
         return;

      String mid = a[(lo + hi) / 2];
      while (lo < hi) {
         while (lo<hi && a[lo].compareTo(mid) < 0)
            lo++;
         while (lo<hi && a[hi].compareTo(mid) > 0)
            hi--;
         if (lo < hi) {
            String T = a[lo];
            a[lo] = a[hi];
            a[hi] = T;
            lo++;
            hi--;
         }
      }
      if (hi < lo) {
         int T = hi;
         hi = lo;
         lo = T;
      }
      quicksort(a, lo0, lo);
      quicksort(a, lo == lo0 ? lo+1 : lo, hi0);
   }
}
```

Code-6.13: SortThread.java

5. Create **words.txt** file as a input file.

```
anatomy
animation
applet
application
argument
bolts
class
communicate
user
```

File-6.14: words.txt

6. Build and run the project

- Right click **PipedReaderWriter** project and select **Run**.
- Observe the result in the **Output** window. (Figure-6.15 below)

```
communicate
application
animation
```

```
user
class
bolts
applet
argument
anatomy
```

Figure-6.15: Result of running UnPipedReaderWriter application

return to top of the exercise

Summary

In this exercise, you learned how to use PipedReader and PipedWriter classes.

return to the top

Exercise 7: File and Directory Handling

In this exercise, you will learn how to use File class to handle file and directory related operations.

1. **File handling**
2. **Directory handling**
3. **Random access file handling**

(7.1) File handling

1. Create a new NetBeans project

- Select **File->New Project (Ctrl+Shift+N)**. The **New Project** dialog box appears.
- Under **Choose Project** pane, select **Java** under **Categories** and **Java Application** under **Projects**. Click **Next**.
- Under **Name and Location** pane, for the **Project Name** field, type in **FileInfo** as project name.
- For **Create Main Class** field, type in **FileInfo**.
- Click **Finish**.

- Observe that **FileInfo** project appears and IDE generated **FileInfo.java** is displayed in the source editor window of NetBeans IDE.

2. Modify the IDE generated **FileInfo.java** as shown in Code-6.11 below. Study the code by paying special attention to the bold fonted parts.

```
import java.io.File;
import java.io.IOException;

public class FileInfo {

   public static void main(String[] args) {

      // The first command line argument needs to be provided
      String fileName = args[0];
      File fn = new File(fileName);
      try {
         fn.createNewFile();
      } catch (IOException e) {

      }
      System.out.println("Name: " + fn.getName());

      // Check if the file exists using exists() method
      if (fn.exists()) {
         System.out.println(fileName + " does exist.");
      }

      if (fn.canRead()) {
         System.out.println(fileName  + " is readable.");
      }

      System.out.println(fileName + " is " + fn.length() + " bytes long.");
      System.out.println(fileName + " is last modifed at " +
            new java.util.Date(fn.lastModified())) ;

      if (fn.canWrite()) {
         System.out.println(fileName  + " is writable.");
      }
      else{
         System.out.println(fileName  + " is not writable.");
      }
   }
}
```

Code-7.11: FileInfo.java

3. Provide command line arguments.

- Right click **FileInfo** project and select **Properties**.
- Observe **Project Properties** dialog box appears.
- Select Run and for the Arguments field, type in **dummyname**. (Figure-3.13 below)

4. Build and run the project

- Right click **FileInfo** project and select **Run**.
- Observe the result in the **Output** window. (Figure-6.12 below)

```
Name: dummyname
dummyname does exist.
dummyname is readable.
dummyname is 0 bytes long.
dummyname is last modifed at Sat Feb 24 15:36:28 EST 2007
dummyname is writable.
```

Figure-7.12: Result of running UnFileInfo application

return to top of the exercise

(7.2) Directory handling

1. Create a new NetBeans project

- Select **File->New Project (Ctrl+Shift+N)**. The **New Project** dialog box appears.
- Under **Choose Project** pane, select **Java** under **Categories** and **Java Application** under **Projects**. Click **Next**.
- Under **Name and Location** pane, for the **Project Name** field, type in **DirectoryInfo** as project name.
- For **Create Main Class** field, type in **DirectoryInfo**.
- Click **Finish**.

- Observe that **DirectoryInfo** project appears and IDE generated **DirectoryInfo.java** is displayed in the source editor window of NetBeans IDE.

2. Modify the IDE generated **DirectoryInfo.java** as shown in Code-7.21 below. Study the code by paying special attention to the bold fonted parts.

```
import java.io.File;

public class DirectoryInfo {

   public static void main(String[] args) {

      // Create a directory
      System.out.println("Creating temp directory...");
      String fileName = "temp";
      File fn = new File(fileName);
      fn.mkdir();

      // Create sub directories under the temp directory
      File subdir1 = new File(fn, "subdir1");
      subdir1.mkdir();
      File subdir2 = new File(fn, "subdir2");
      subdir2.mkdir();

      // Check if it is a file or directory using isFile() method
      System.out.println(fileName + " is a " +
            (fn.isFile()? "file." :"directory."));

      if (fn.isDirectory()) {
         String content[] = fn.list();
         System.out.println("The content of this directory:");
         for (int i = 0; i < content.length; i++) {
            System.out.println(content[i]);
         }
      }

      // Delete a directory
      System.out.println(fileName +
            (fn.exists()? " exists": " does not exist"));
      System.out.println("Deleting temp directory...");
      fn.delete();

   }
}
```

Code-7.21: DirectoryInfo.java

3 Build and run the project

- Right click **DirectoryInfo** project and select **Run**.
- Observe the result in the **Output** window. (Figure-7.22 below)

```
Creating temp directory...
temp is a directory.
The content of this directory:
subdir1
subdir2
temp exists
Deleting temp directory...
```

Figure-7.22: Result

return to top of the exercise

(7.3) Random access file handling

1. Create a new NetBeans project

- Select **File->New Project (Ctrl+Shift+N)**. The **New Project** dialog box appears.
- Under **Choose Project** pane, select **Java** under **Categories** and **Java Application** under **Projects**. Click **Next**.
- Under **Name and Location** pane, for the **Project Name** field, type in **RandomAccessFileHandling** as project name.
- For **Create Main Class** field, type in **RandomAccessFileHandling**.
- Click **Finish**.

- Observe that **RandomAccessFileHandling** project appears and IDE generated **RandomAccessFileHandling.java** is displayed in the source editor window of NetBeans IDE.

2. Modify the IDE generated **RandomAccessFileHandling.java** as shown in Code-6.21 below. Study the code by paying special attention to the bold fonted parts.

```
import java.io.IOException;
import java.io.RandomAccessFile;

public class RandomAccessFileHandling {

   public static void main(String[] args) {
      try {
         RandomAccessFile raf = new RandomAccessFile("test.txt", "rw");
         raf.writeInt(10);
         raf.writeInt(43);
         raf.writeInt(88);
         raf.writeInt(455);

         // change the 3rd integer from 88 to 99
         raf.seek((3 - 1) * 4);
         raf.writeInt(99);
         raf.seek(0); // go to the first integer
         int i = raf.readInt();
         while (i != -1) {
            System.out.println(i);
            i = raf.readInt();
         }
         raf.close();
      } catch (IOException e) {
      }
   }
}
```

Code-7.31: RandomAccessFileHandling.java

3 Build and run the project

- Right click **RandomAccessFileHandling** project and select **Run**.
- Observe the result in the **Output** window. (Figure-6.22 below)

```
10
43
99
455
```

Figure-7.32: Result

return to top of the exercise

Summary

In this exercise, you have learned how to use File class to handle file and directory related operations.

return to the top

Exercise 8: Filter Streams

A FilterInputStream contains some other input stream, which it uses as its basic source of data, possibly transforming the data along the way or providing additional functionality. The class FilterInputStream itself simply overrides all methods of InputStream with versions that pass all requests to the contained input stream. Subclasses of FilterInputStream may further override some of these methods and may also provide additional methods and fields.

(8.1) Create custom stream class that extends Filter streams

1. Create a new NetBeans project

- Select **File->New Project (Ctrl+Shift+N)**. The **New Project** dialog box appears.
- Under **Choose Project** pane, select **Java** under **Categories** and **Java Application** under **Projects**. Click **Next**.
- Under **Name and Location** pane, for the **Project Name** field, type in **FilterInputOutputStream** as project name.
- For **Create Main Class** field, type in **FilterInputOutputStream**.
- Click **Finish**.

- Observe that **FilterInputOutputStream** project appears and IDE generated **FilterInputOutputStream.java** is displayed in the source editor window of NetBeans IDE.

2. Modify the IDE generated **FilterInputOutputStream.java** as shown in Code-8.11 below. Study the code by paying special attention to the bold fonted parts.

```
import java.io.*;

public class FilterInputOutputStream {

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

      Adler32 inChecker = new Adler32();
      Adler32 outChecker = new Adler32();
      CheckedInputStream in = null;
      CheckedOutputStream out = null;

      try {
         in = new CheckedInputStream(
            new FileInputStream("farrago.txt"),
            inChecker);
         out = new CheckedOutputStream(
             new FileOutputStream("outagain.txt"),
             outChecker);
      } catch (FileNotFoundException e) {
         System.err.println("CheckedIOTest: " + e);
         System.exit(-1);
      } catch (IOException e) {
         System.err.println("CheckedIOTest: " + e);
         System.exit(-1);
      }

      int c;

      while ((c = in.read()) != -1)
        out.write(c);

      System.out.println("Input stream check sum: " +
            inChecker.getValue());
      System.out.println("Output stream check sum: " +
            outChecker.getValue());

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

Code-8.11: FilterInputOutputStream.java

3. Write CheckedInputStream.java. (Code-8.12 below)

```
import java.io.FilterInputStream;
import java.io.InputStream;
import java.io.IOException;

// Custom InputStream
public class CheckedInputStream extends FilterInputStream {

   private Checksum cksum;

   public CheckedInputStream(InputStream in, Checksum cksum) {
      super(in);
```

```
        this.cksum = cksum;
    }

    public int read() throws IOException {
        int b = in.read();
        if (b != -1) {
            cksum.update(b);
        }
        return b;
    }

    public int read(byte[] b) throws IOException {
        int len;
        len = in.read(b, 0, b.length);
        if (len != -1) {
            cksum.update(b, 0, len);
        }
        return len;
    }

    public int read(byte[] b, int off, int len) throws IOException {
        len = in.read(b, off, len);
        if (len != -1) {
            cksum.update(b, off, len);
        }
        return len;
    }

    public Checksum getChecksum() {
        return cksum;
    }
}
```

Code-8.12: CheckedInputStream.java

4. Write CheckedOutputStream.java. (Code-8.13 below)

```
import java.io.*;

// Custom OutputStream
public class CheckedOutputStream extends FilterOutputStream {

    private Checksum cksum;

    public CheckedOutputStream(OutputStream out, Checksum cksum) {
        super(out);
        this.cksum = cksum;
    }

    public void write(int b) throws IOException {
        out.write(b);
        cksum.update(b);
    }

    public void write(byte[] b) throws IOException {
        out.write(b, 0, b.length);
        cksum.update(b, 0, b.length);
    }

    public void write(byte[] b, int off, int len) throws IOException {
        out.write(b, off, len);
        cksum.update(b, off, len);
    }

    public Checksum getChecksum() {
        return cksum;
    }
}
```

Code-8.13: CheckedOutputStream.java

5. Write **Checksum.java**. (Figure-8.14 below)

```
public interface Checksum {
    /**
     * Updates the current checksum with the specified byte.
     */
    public void update(int b);

    /**
     * Updates the current checksum with the specified array of bytes.
     */
    public void update(byte[] b, int off, int len);
```

```
    /**
     * Returns the current checksum value.
     */
    public long getValue();

    /**
     * Resets the checksum to its initial value.
     */
    public void reset();
}
```

Code-8.14: Checksum.java

6. Write **Adler32.java**. (Code-8.15 below)

```
public class Adler32 implements Checksum {

    private int value = 1;

    /*
     * BASE is the largest prime number smaller than 65536
     * NMAX is the largest n such that 255n(n+1)/2 + (n+1)(BASE-1) <= 2^32-1
     */
    private static final int BASE = 65521;
    private static final int NMAX = 5552;

    /**
     * Update current Adler-32 checksum given the specified byte.
     */
    public void update(int b) {
        int s1 = value & 0xffff;
        int s2 = (value >> 16) & 0xffff;
        s1 += b & 0xff;
        s2 += s1;
        value = ((s2 % BASE) << 16) | (s1 % BASE);
    }

    /**
     * Update current Adler-32 checksum given the specified byte array.
     */
    public void update(byte[] b, int off, int len) {
        int s1 = value & 0xffff;
        int s2 = (value >> 16) & 0xffff;

        while (len > 0) {
            int k = len < NMAX ? len : NMAX;
            len -= k;
            while (k-- > 0) {
                s1 += b[off++] & 0xff;
                s2 += s1;
            }
            s1 %= BASE;
            s2 %= BASE;
        }
        value = (s2 << 16) | s1;
    }

    /**
     * Reset Adler-32 checksum to initial value.
     */
    public void reset() {
        value = 1;
    }

    /**
     * Returns current checksum value.
     */
    public long getValue() {
        return (long)value & 0xffffffff;
    }
}
```

Code-8.15: Checksum.java

7. Provide **farrago.txt** as an input file.

```
So she went into the garden to cut a cabbage-leaf, to
make an apple-pie; and at the same time a great
she-bear, coming up the street, pops its head into the
shop. 'What! no soap?' So he died, and she very
imprudently married the barber; and there were
present the Picninnies, and the Joblillies, and the
```

```
Garyalies, and the grand Panjandrum himself, with the
little round button at top, and they all fell to playing
the game of catch as catch can, till the gun powder ran
out at the heels of their boots.

Samuel Foote 1720-1777
```

File-2.12: farrago.txt

8. Build and run the project

- Right click **FilterInputOutputStream** project and select **Run**.
- Observe the result in the **Output** window. (Figure-8.16 below)

```
Input stream check sum: 736868089
Output stream check sum: 736868089
```

Figure-8.16: Result

return to top of the exercise

Summary

In this exercise, you have learned how to use File class to handle file and directory related operations.

return to the top

Exercise 9: Scanner and Formatter

Scanner class provides a simple text scanner which can parse primitive types and strings using regular expressions. A Scanner breaks its input into tokens using a delimiter pattern, which by default matches whitespace. The resulting tokens may then be converted into values of different types using the various `next` methods. Scanner class is introduced from Java SE 5.

1. **Scanner**
2. **Formatter**

(9.1) Scanner class

1. Create a new NetBeans project

- Select **File->New Project (Ctrl+Shift+N)**. The **New Project** dialog box appears.
- Under **Choose Project** pane, select **Java** under **Categories** and **Java Application** under **Projects**. Click **Next**.
- Under **Name and Location** pane, for the **Project Name** field, type in **ScannerClass** as project name.
- For **Create Main Class** field, type in **ScannerClass**.
- Click **Finish**.

- Observe that **ScannerClass** project appears and IDE generated **ScannerClass.java** is displayed in the source editor window of NetBeans IDE.

2. Modify the IDE generated **ScannerClass.java** as shown in Code-6.11 below. Study the code by paying special attention to the bold fonted parts.

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

public class ScannerClass {

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

      Scanner s =
            new Scanner(new BufferedReader(new FileReader("usnumbers.txt")));
      s.useLocale(Locale.US);

      double sum = 0;

      while (s.hasNext()) {
         sum += s.nextDouble();
      }
      s.close();

       System.out.println("Sum of all numbers = " + sum);
   }
}
```

Code-9.11: ScannerClass.java

3. Provide **usnumbers.txt** as input data file.

```
8.5
32,767
```

```
3.14159
1,000,000.1
```

File-9.12: usnumbers.txt

4. Build and run the project

- Right click **ScannerClass** project and select **Run**.
- Observe the result in the **Output** window. (Figure-6.12 below)

```
Sum of all numbers = 1032778.74159
```

Figure-6.12: Result

return to top of the exercise

(9.2) Format class

1. Create a new NetBeans project

- Select **File->New Project (Ctrl+Shift+N)**. The **New Project** dialog box appears.
- Under **Choose Project** pane, select **Java** under **Categories** and **Java Application** under **Projects**. Click **Next**.
- Under **Name and Location** pane, for the **Project Name** field, type in **FormatClass** as project name.
- For **Create Main Class** field, type in **FormatClass**.
- Click **Finish**.

- Observe that **FormatClass** project appears and IDE generated **FormatClass.java** is displayed in the source editor window of NetBeans IDE.

2. Modify the IDE generated **FormatClass.java** as shown in Code-6.21 below. Study the code by paying special attention to the bold fonted parts.

```
public class FormatClass {
   public static void main(String[] args) {
      System.out.format("%f, %1$+020.10f %n", Math.PI);
   }
}
```

Code-6.21: FormatClass.java

3 Build and run the project

- Right click **FormatClass** project and select **Run**.
- Observe the result in the **Output** window. (Figure-6.22 below)

```
3.141593, +00000003.1415926536
```

Figure-6.22: Result

return to top of the exercise

Summary

In this exercise, you have learned how to use Scanner and Format classes.

return to the top

Homework

1. The homework is to either modify **FilterInputOutputStream** NetBeans project you've done in **Exercise 8 above** or create a new project as following. (You might want to create a new project by copying the **FilterInputOutputStream** project. You can name the homework project in any way you want but here I am going to call it **MyIOStreamProject**.)

- Write ChangeToUpperCaseInputStream class, which extends FilterInputStream.
- Write ChangeToUpperCaseOutputStream class, which extends FilterOutputStream.
- Use either ChangeToUpperCaseInputStream class or ChangeToUpperCaseOutputStream class to convert the characters read to upper case.