eg#2. Reading an array of characters

abc.txt
=======
  1000 characters are available

Scenario1:

```
       FileReader fr=new FileReader("abc.txt");
       char[] ch=new char[10];
       int noOfCharactersCopied=fr.read(ch);
```

Scenario2:

```
       FileReader fr=new FileReader("abc.txt");
       char[] ch=new char[10000];
       int noOfCharactersCopied=fr.read(ch);
```

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

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

                  File f=new File("abc.txt");

                  FileReader fr=new FileReader(f);
                  char ch[] = new char[(int)f.length()];

                  fr.read(ch);

                  String data=new String(ch);
                  System.out.println(data);

                  fr.close();
      }
}
```

Usage of FileWriter and FileReader is not recommended because of following reason

1. While writing data by FileWriter compulsory we should insert line separator(\n) manually which is a bigger headache to the
     programmer.

2. While reading data by FileReader we have to read character by character instead of line by line which is not convenient to the
     programmer.

Assume we need to search for a 10 digit mobile no present in a file called "mobile.txt"
  =>Since we can read only character just to search one mobile no 10 searching and to search 10,000 mobile no we need to read 1cr times,
       so performance is very low.

3. To overcome these limitations we should go for BufferedWriter and BufferedReader concepts.

BufferedWriter:

```
    By using BufferedWriter we can write the character data to the file.
    It can't communicate with the file directly, it can communicate only with
writer Object.

Constructor
   BufferedWriter bw=new BufferedWriter(Writer w);
   BufferedWriter bw=new BufferedWriter(Writer w,int buffersize);

Which of the following declarations are valid?
1. BufferedWriter bw=new BufferedWriter("cricket.txt"); //invalid
2. BufferedWriter bw=new BufferedWriter (new File("cricket.txt"));//invalid
3. BufferedWriter bw=new BufferedWriter (new FileWriter("cricket.txt")); //valid
4. BufferedWriter bw=new BufferedWriter(new BufferedWriter(new
FileWriter("crickter.txt")));//valid
                              // it indicates 2 levels of buffering.

Methods
========
1. write(int ch);
2. write(char[] ch);
3. write(String s);
4. flush();
5. close();
6. newline();
      Inserting a new line character to the file.

Note:
When compared with FileWriter which of the following capability(facility) is
available as method in BufferedWriter.
1. Writing data to the file.
2. Closing the writer.
3. Flush the writer.
4. Inserting newline character.(newLine())

Ans. 4

eg#1.
import java.io.*;
/*
      public void newLine() throws java.io.IOException;
*/
class Test
{
      public static void main(String[] args)throws Exception
      {
            FileWriter fw     = new FileWriter("abc.txt",true);
            BufferedWriter bw = new BufferedWriter(fw);

            bw.write(105);
            bw.write("Neuron");

            bw.newLine();

            char[] ch ={'P','W','S','K','I','L','L','S'};
            bw.write(ch);

            bw.newLine();

            bw.write("unicorn");
            bw.flush();//to make sure the operation is succesfull on file

            bw.close();//internally fw.close() call will happen

      }
      //JVM shutdown now
}
```

Note
1.bw.close()// recomended to use
2.fw.close()// not recomended to use
3.bw.close()// not recomended to use
  fw.close()
=> When ever we are closing BufferedWriter automatically underlying writer will be closed and we are not close explicitly.

BufferedReader:
     This is the most enhanced(better) Reader to read character data from the file.

Constructors:
```
      BufferedReader br=new BufferedReader(Reader r);
      BufferedReader br=new BufferedReader(Reader r,int buffersize);
```

Note
 => BufferedReader can not communicate directly with the File it should communicate via some Reader object.
 => The main advantage of BufferedReader over FileReader is we can read data line by line instead of character by character.

Methods:
1. int read();
2. int read(char[] ch);
3. String readLine();
           It attempts to read next line and return it , from the File. if the next line is not available then this method returns null.
4. void close();

eg#1.Read the data from the file called "abc.txt"4
```
/*
      public java.lang.String readLine() throws java.io.IOException;
*/
import java.io.*;
public class TestApp {
      public static void main(String[] args)throws IOException {

                  FileReader fr=new FileReader("abc.txt");
                  BufferedReader br=new BufferedReader(fr);
                  String line= br.readLine();
                  while(line!=null){
                        System.out.println(line);
                        line=br.readLine();
                  }
                  br.close();
      }
}
```

Note:

```
1.br.close()  // recomended to use

2.fw.close() // not recomended to use

3.br.close() // not recomended to use both
   fw.close()
```

=> Whenever we are closing BufferedReader automatically underlying FileReader will be closed it is not required to close explicitly.
=> Even this rule is applicable for BufferedWriter also.

PrintWriter:
=> This is the most enhanced Writer to write text data to the file.
=> By using FileWriter and BufferedWriter we can write only character data to the File but by using PrintWriter
      we can write any type of data to the File.

Constructors:

```
PrintWriter pw=new PrintWriter(String name);
PrintWriter pw=new PrintWriter(File f);
PrintWriter pw=new PrintWriter(Writer w);
```

PrintWriter can communicate either directly to the File or via some Writer object also.

Methods:

```
1. write(int ch);
2. write (char[] ch);
3. write(String s);
4. flush();
5. close();

6. print(char ch);
7. print (int i);
8. print (double d);
9. print (boolean b);
10.print (String s);
11.println(char ch);
12.println (int i);
13.println(double d);
14.println(boolean b);
15.println(String s);
```

eg#1.

```
import java.io.*;
/*
       public void print(xxxx type);
       public void println(xxxx type);
*/
class Test
{
      public static void main(String[] args)throws Exception
      {
            FileWriter  fw = new FileWriter("abc.txt");
            PrintWriter out = new PrintWriter(fw);

            out.write(100);//100 unicode value will be written to a file
            out.write('\n');

            out.println(100);//100 only will be written to the file

            out.println(true);

            out.println('c');
            out.println("DenisRitchie");

            out.flush();
            out.close();
      }
      //JVM shutdown now
}
```

What is the difference between write(100) and print(100)?
=> In the case of write(100) the corresponding character "d" will be added to the File but
=> In the case of print(100) "100" value will be added directly to the File.

Note 1:
1. The most enhanced Reader to read character data from the File is BufferedReader.
2. The most enhanced Writer to write character data to the File is PrintWriter.

Note 2:
1.In general we can use Readers and Writers to handle character data. Where as we can use   InputStreams and OutputStreams to handle
    binary data(like images, audio files, video files etc).
2. We can use OutputStream to write binary data to the File and we can use InputStream to read binary data from the File

```
            Character Data  => Reader and Writer
            Binary Data       =>  InputStream and OutputStream
```

Requirement => file1.txt ,file2.txt copy all the contents to file3.txt

```
import java.io.*;
class Test
{
      public static void main(String[] args)throws Exception
      {
            PrintWriter pw = new PrintWriter("file3.txt");

            //reading from first file and writing to file3.txt
            BufferedReader br= new BufferedReader(new FileReader("file1.txt"));
            String line = br.readLine();
            while (line!=null)
            {
                  pw.println(line);
                  line=br.readLine();
            }

            //reading from second file and writing to file3.txt
            br =new BufferedReader(new FileReader("file2.txt"));
            line = br.readLine();
            while (line!=null)
            {
                  pw.println(line);
                  line=br.readLine();
```

```
		}

		//To write all the data to file3.txt
		pw.flush();

		br.close();
		pw.close();

		System.out.println("Open file3.txt to see the result");

	}
	//JVM shutdown now
}
```

Requirement => file1.txt file2.txt copy one line from file1.txt and from file2.txt to file3.txt.

```
import java.io.*;
class Test
{
	public static void main(String[] args)throws Exception
	{
		PrintWriter pw = new PrintWriter("file3.txt");

		//reading from first file and writing to file3.txt
		BufferedReader br1= new BufferedReader(new FileReader("file1.txt"));
		String line1 = br1.readLine();

		BufferedReader br2 =new BufferedReader(new FileReader("file2.txt"));
		String line2 = br2.readLine();

		while (line1!=null || line2!=null)
		{
			if (line1!=null)
			{
				pw.println(line1);
				line1=br1.readLine();
			}

			if(line2!=null)
			{
				pw.println(line2);
				line2=br2.readLine();
			}

		}

		//To write all the data to file3.txt
		pw.flush();

		br1.close();
		br2.close();

		pw.close();

		System.out.println("Open file3.txt to see the result");

	}
	//JVM shutdown now
}
```

Requirement => Write a program to remove duplicates from the file

```
import java.io.*;
class Test
{
      public static void main(String[] args)throws Exception
      {
            BufferedReader br = new BufferedReader(new FileReader("input.txt"));
            PrintWriter    pw = new PrintWriter("output.txt");

            String target = br.readLine();
            while (target!=null)
            {

                  boolean isAvailable =false;

                  BufferedReader br1 =new BufferedReader(new
FileReader("output.txt"));
                  String line = br1.readLine();

                  //control comes out of while looop in smooth fashion without
break
                  while (line!=null)
                  {
                        //if matched control should come out with break
                        if (line.equals(target))
                        {
                                    isAvailable = true;
                                    break;
                        }
                        line=br1.readLine();
                  }

                  if (isAvailable==false){
                        pw.println(target);
                        pw.flush();
                  }

                  target = br.readLine();
            }
            br.close();
            pw.close();

      }
      //JVM shutdown now
}
```

Requirement => Write a program to perform extraction of mobile no only if there is no duplicates

```
import java.io.*;


class Test
{
      public static void main(String[] args)throws Exception
      {
            BufferedReader br = new BufferedReader(new FileReader("input.txt"));
            PrintWriter    pw = new PrintWriter("output.txt");

            String target = br.readLine();

            while (target!=null)
            {
                  boolean isAvailable =false;

                  BufferedReader br1 =new BufferedReader(new
FileReader("delete.txt"));
                  String line = br1.readLine();

                  //control comes out of while looop in smooth fashion without
break
                  while (line!=null)
                  {
                        //if matched control should come out with break
                        if (line.equals(target))
                        {
                                    isAvailable = true;
                                    break;
                        }
                        line=br1.readLine();
                  }

                  if (isAvailable==false){
                        pw.println(target);
                        pw.flush();
                  }

                  target = br.readLine();
            }
            br.close();
            pw.close();
      }
      //JVM shutdown now
}
```

Write a code to read the data from the file and identify which data is of larger in length(assuming the data is String)

```
import java.io.*;
class Test
{
      public static void main(String[] args)throws Exception
      {
            BufferedReader br = new BufferedReader(new FileReader("data.txt"));
            String data = br.readLine();

            int maxLength = 0;
            String result = "";
            while (data!=null)
            {
                  int resultLength=data.length();
                  if (maxLength<resultLength)
                  {
                        maxLength = resultLength;
                        result=data;
                  }
                  data= br.readLine();
            }
            System.out.println("The maxLength string data in a file is :: 
"+result);
            System.out.println("The maxLength string in a file is :: "+maxLength);
      }
      //JVM shutdown now
}
```