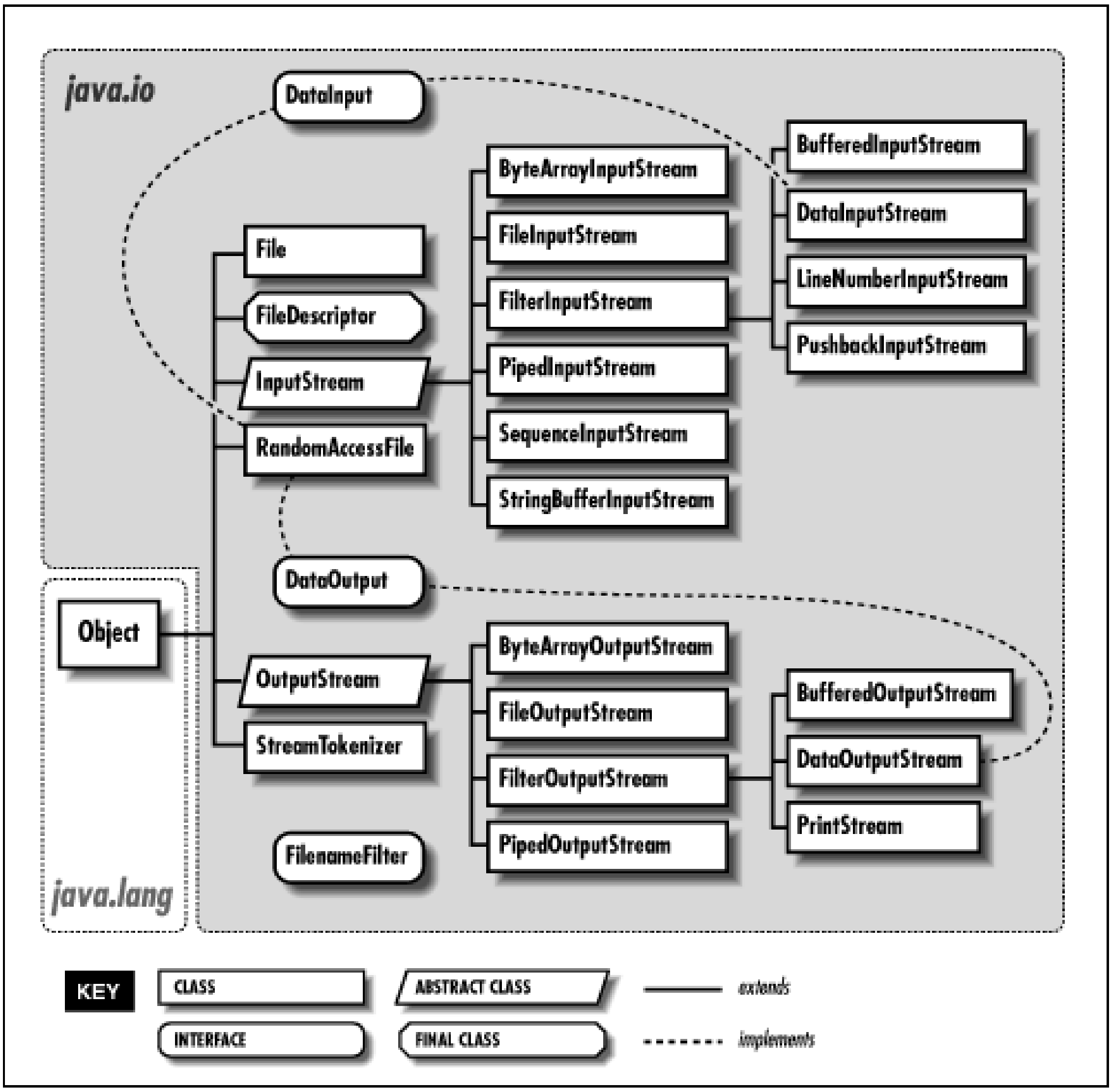
12. FILE HANDLING

Java provides a number of classes and methods that allow you to read and write files. In Java, all files are byte-oriented, and Java provides methods to read and write bytes from and to a file. However, Java allows you to wrap a byte-oriented file stream within a character-based object.

1. The stream classes:



Following **abstract classes** are used: Following classes are designed for byte streams

- InputStream
- OutputStream

Following classes are designed for character streams

- Reader
- Writer

2. Byte Stream classes:

These are used to handle byte streams

1. InputStream class:

This class implements Closeable interface. It has some following methods used by subclasses.

Name	Description
int available( )	Returns the number of bytes of input currently available for reading.
void close( )	Closes the input source. Further read attempts will generate an IOException.
int read( )	Returns an integer representation of the next available byte of input. –1 is returned when the end of the file is encountered.
int read(byte buffer[ ])	Attempts to read up to buffer.length bytes into buffer and returns the actual number of bytes that were successfully read. –1 is returned when the end of the file is encountered.
int read(byte buffer[ ], int offset, int no.of bytes)	Attempts to read file but start storing the bytes into buffer from the position specified in offset and reads number of bytes specified in variable no. Of bytes. It returns the actual number of bytes that were successfully read. If the array size is exceeding while storing then exception will occur.

Most of the methods throw **IOException** (checked exception)

2. OutputStream class:

It implements Closeable and Flushable interface. Following are some methods of OutputStream class which are used by subclasses.

Name	Description
void close( )	Closes the output stream. Further write attempts will generate an IOException.
void flush( )	Finalizes the output state so that any buffers are cleared. That is, it flushes the output buffers.
void write(int b)	Writes a single byte to an output stream. Note that the parameter is an int, which allows you to call write( ) with expressions without having to cast them back to byte.
void write(byte buffer[ ])	Writes a complete array of bytes to an output stream.
void write(byte buffer[ ], int offset, int numBytes)	Writes a subrange of numBytes bytes from the array buffer,beginning at buffer[offset].

3. Opening the file in read mode:

To open the file in read mode to read bytes from file. FileInputStream class is used.

FileInputStream class extends InputStream

Constructors of FileInputStream:

- **FileInputStream(String filepath) throws FileNotFoundException**
  This constructor takes the file path to open the file. If only file name is provided then file will be searched in current working directory
- **FileInputStream(File fileObj) throws FileNotFoundException**
  This constructor takes File class object to open the file specified in this object. File is class in java. It does not deals with streams. It directly deals with files and file system.

The constructors of FileInputStream throws checked exception known FileNotFoundException if the file is not present on specified path.

Example:
Suppose there is a file present, details.txt, which has following contents

Employee Name: Arvind Gupta Age: 30 Salary: 55000
Address: Sector 20, Airoli, Navi Mumbai PinCode: 400708

Following program reads the file one byte at a time. and return the bytes in terms of integer.

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
public class FileHandlingReadBytesDemo {

	public static void main(String[] args) throws IOException {
		InputStream readfile=null;
		int content=0;
		try {
			readfile=new FileInputStream("details.txt");

		System.out.println("No. of bytes available to read:"+readfile.available()+"\n");
			System.out.println("File contents:");
			while(content!=-1)
			{
				content=readfile.read();

				if(content==-1)
					break;
				System.out.print((char)content);
```

```
            }
            System.out.println("\nNo. of bytes available to read:"+readfile.available());

        } catch (FileNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        finally
        {
            readfile.close();
        }

    }
}
```

Output:

```
No. of bytes available to read:106
File contents:
Employee Name: Arvind Gupta Age: 30 Salary: 55000
Address: Sector 20, Airoli, Navi Mumbai PinCode: 400708

No. of bytes available to read:0
```

Following program reads file contents and store into byte array.

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;

public class FileHandlingByteArrayDemo {

    public static void main(String[] args) throws IOException {
        InputStream readfile=null;
        int content=0;
        byte store[]=new byte[50];
        try {

            readfile=new FileInputStream("details.txt");
            content=readfile.read(store);   // file contents store in byte array starting from 0 position but up-to capacity of array. Remaining will be discarded.
            System.out.println("File contents in array:");

            for(int i=0;i<store.length;i++)
            {
                System.out.print((char)store[i]);
            }

            System.out.println("Number of bytes read from the file: "+content);

        } catch (FileNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        finally
        {
            readfile.close();
        }

    }

}
```

Output:

```
File contents in array:

Employee Name: Arvind Gupta Age: 30 Salary: 55000
Number of bytes read from the file: 50
```

If you use following read method

content=readfile.read(store,5,40);   // 40 bytes get read from the file and stored from 5th position of array name store. Remaining will be discarded. If we display the byte array then following will be output.

```
    System.out.println("File contents in array:");

    for(int i=5;i<45;i++)
            {
                System.out.print((char)store[i]);
                }
```

Output will be:

```
File contents in array:
Employee Name: Arvind Gupta Age: 30
```

4. Opening the file in write or append mode.

The class FileOutputStream is used to open file in either write mode or append mode. This class extends OutputStream class.

Constructors of FileOutputStream:

- **FileOutputStream(String filePath)**

  This constructor takes the file path to open the file in write mode. If only file name is provided then file will be searched in current working directory
- **FileOutputStream(File fileObj)**

  This constructor takes the File object to open the file in write mode.
- **FileOutputStream(String filePath , boolean append)**

  This constructor takes the file path as well as boolean value as true if to open the file in append mode.

- **FileOutputStream(File fileObj , boolean append)**

  This constructor takes the File object as well as boolean value as true if to open the file in append mode.

If the file is not present at specified path then new file will get creatred. Still all above constructors throw FileNotFoundException

Example: to write byte array in file

```
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStream;

public class FileHandlingWriteBytesDemo {

        public static void main(String[] args) {
                OutputStream writefile=null;
                String contents="Coder Technologies, Vashi Railway Station Complex, Vashi";
                byte writearray[]=new byte[500];
                try {
                writefile=new FileOutputStream("WriteDemo.txt"); // open file in write mode
                        writearray=contents.getBytes(); // converts string to byte array
                        writefile.write(writearray);  // write byte array contents into file
                        writefile.write(writearray,4,50);// write byte array contents staring from
particular position

        } catch (FileNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

Example: to read one file contents and write to another file

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

public class FileHandlingReadWriteBytesDemo {

    public static void main(String[] args) throws IOException {
        OutputStream writefile=null;
        InputStream readfile=null;
        int content=0;

        try {

            readfile=new FileInputStream("details.txt"); // open the file in read mode
        writefile=new FileOutputStream("WriteDemo.txt"); // open file in write mode

            while(content!=-1)
            {
                content=readfile.read();  // read byte in integer
                    if(content==-1)
                    break;
                writefile.write(content); // write integer
            }

        } catch (FileNotFoundException e) {
            // TODO Auto-generated catch block
```

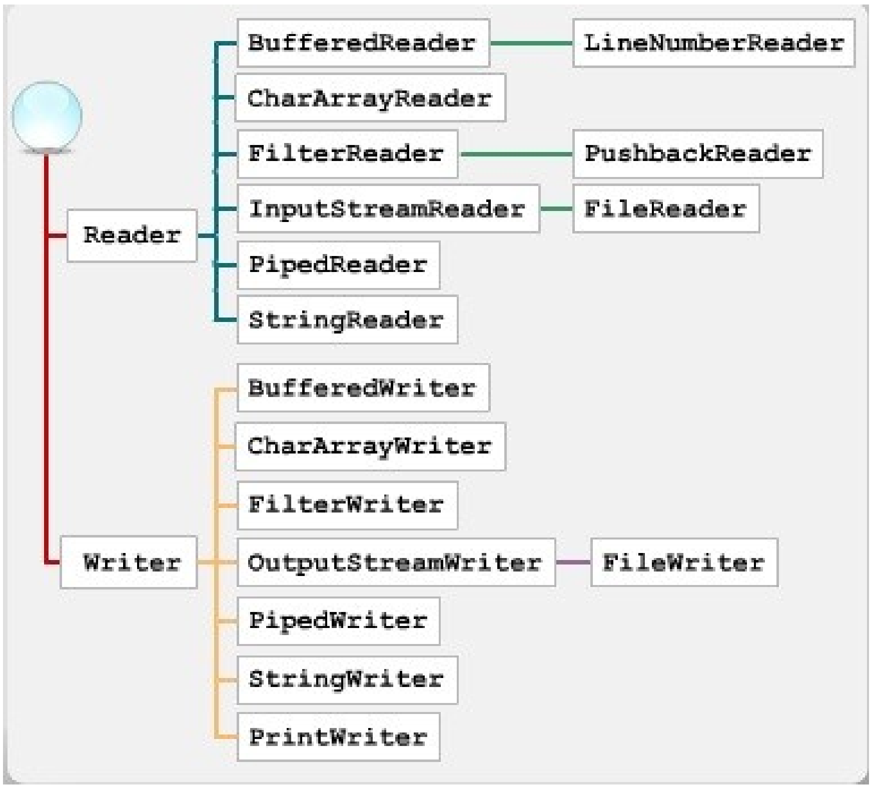
```
                    e.printStackTrace();
                }
                catch (IOException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
                finally
                {
                    readfile.close();
                    writefile.close();
                }

        }

}
```

3. Character Stream classes:

The classes under this are used to handle the character stream.

1. Reader class:

- abstract class
- implements Closeable and Readable interface



Some of the methods of Reader class which are used by its sub classes are:

Method	Descrption
abstract void close( )	Closes the input source. Further read attempts will generate an IOException
int read( )	Returns an integer representation of the next available character from the invoking input stream. –1 is returned when the end of the file is encountered
int read(char buffer[ ])	Attempts to read up to buffer.length characters into buffer and returns the actual number of characters that were successfully read. –1 is returned when the end of the file is encountered
abstract int read(char buffer[ ], int offset, int numChars)	Attempts to read up to numChars characters into buffer starting at buffer[offset], returning the number of characters successfully read. –1 is returned when the end of the file is encountered

Here most of the methods throw IOException

2. Writer Class:

- abstract class
- implements Closeable, Flushable and Appendable interface

Some of the methods of Writer class which are used by sub classes are:

Method	Descrption
abstract void close( )	Closes the output stream. Further write attempts will generate an IOException.
abstract void flush( )	Finalizes the output state so that any buffers are cleared. That is, it flushes the output buffers.
void write(int ch)	Writes a single character to the invoking output stream.
void write(char buffer[ ]) a	Writes a complete array of characters to the invoking output stream.
bstract void write(char buffer[ ], int offset, int numChars)	Writes a subrange of numChars characters from the array buffer, beginning at buffer[offset] to the invoking output stream.
void write(String str)	Writes str to the invoking output stream.
void write(String str, int offset, int numChars)	Writes a subrange of numChars characters from the string str, beginning at the specified offset

most methods throw IOException

3. Opening the file in read mode

The **FileReader** class is used to open the file in read mode.

Constructors of FileReader:

- **FileReader(String filePath)throws FileNotFoundException**
  opens the file in read mode from specified path
- **FileReader(File fileObj)throws FileNotFoundException**
  opens the file in read mode specified by File object

Example:

```
import java.io.FileNotFoundException;

import java.io.FileReader;
import java.io.IOException;
import java.io.InputStreamReader;
public class FileHandlingReadCharsDemo {

        public static void main(String[] args) throws IOException {
                // TODO Auto-generated method stub
        InputStreamReader readfile=null;
        int i=0;
        char store[]=new char[100];
        try {
                 readfile=new FileReader("demo.txt");
                 System.out.println("File contents:");
                while(i!=-1)
                {
                i=readfile.read();//reads one byte at a time and returns bytes in term of integer
                        if(i==-1)break;
                        System.out.print((char)i);
                }
                readfile.close();

                 readfile=new FileReader("demo.txt");

                 i=readfile.read(store);    // reads bytes and store in byte array until byte array
becomes full. returns number of bytes read from file
                String s=new String(store);
                 System.out.println("\n\nBuffer contents:\n"+s);
                 readfile.close();
                readfile=new FileReader("demo.txt");

                i=readfile.read(store, 3, 10); // reads bytes and store in byte array from given
position and number of bytes. returns number of bytes read from file
                System.out.println("\nBuffer contents:");
                for(int j=3;j<13;j++)
                {
                    System.out.print((char)store[j]);
                }


        } catch (FileNotFoundException e) {
                    // TODO Auto-generated catch block
                e.printStackTrace();
        } catch (IOException e) {
                    // TODO Auto-generated catch block
                e.printStackTrace();
        }

        finally
        {
                readfile.close();
        }
        }
}// end
```

Output:

```
1. For first read method
File contents:
Coder Technologies, Vashi Railway Station Complex, VashiCoder Technologies, Vashi Railway
Station Complex, Vashi

2. For second read method
Buffer contents:
Coder Technologies, Vashi Railway Station Complex, VashiCoder Technologies, Vashi Railway
Station Co

3. For third read demo
Buffer contents:
Coder Tech
```

3. Opening the file in write mode

12. FILE HANDLING

The **FileWriter** class is used to open the file in write or append mode

Constructors of FileWriter:

- **FileWriter(String filePath)**
  opens the file specified in path in write mode
- **FileWriter(File fileObj)**
  opens the file specifies by File object in write mode
- **FileWriter(String filePath, boolean append)**
  opens the file specified in path in append mode if append=true
- FileWriter(File fileObj, boolean append)

  opens the file specified by File object in append mode if append=true

all throws IOException and FileNotFoundException. For e.g., attempt to open the read-only file in write mode.

Example:

```
import java.io.FileWriter;
import java.io.IOException;
import java.io.OutputStreamWriter;

public class FileHandlingWriteCharsDemo {

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

        OutputStreamWriter writefile=null;

        try {
                writefile=new FileWriter("demo.txt");

                String contents="Coder Technologies, Vashi Railway Station Complex, Vashi";

                writefile.write(contents);              //writes whole string

                writefile.write(contents, 3, 10);       // write 10 characters starting from 3rd position

                char contentarray[]=contents.toCharArray();
                writefile.write(contentarray);   //write whole character array


                writefile.write(contentarray, 3, 10); // write 10 characters starting from 3rd position
        } catch (IOException e) {
                    // TODO Auto-generated catch block
            e.printStackTrace();
            }
        finally{
            writefile.close();
            }

        }

}
```

4. PrintWriter class

- It is used to write the contents into file, console or port.
- implements Appendable, Closeable, Flushable interfaces

Constructors:

PrintWriter(OutputStream outputStream)
PrintWriter(OutputStream outputStream, boolean flushOnNewline)
PrintWriter(Writer outputStream)
PrintWriter(Writer outputStream, boolean flushOnNewline)

Constructors to write output to file:

PrintWriter(File outputFile) throws FileNotFoundException
PrintWriter(File outputFile, String charSet)throws FileNotFoundException, UnsupportedEncodingException
PrintWriter(String outputFileName) throws FileNotFoundException
PrintWriter(String outputFileName, String charSet)throws FileNotFoundException, UnsupportedEncodingException

Methods to use:

print or println method: to write all type values
toString method: to write objects to file, call this method first

Example:

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.PrintWriter;

public class PrintWriterDemo {

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

12. FILE HANDLING

```
                File f=null;
                PrintWriter pw=null;

                String s="Coder technologies, Vashi";
                String s1="Squad Infotech, Nerul";
                String name="Rahul";

                int age=25;
                int luckynum=7;
                try {
                 f=new File("pwdemo.txt");
                     pw=new PrintWriter(f);
                     pw.println(s);                    //to print string
                     pw.flush();                       // to flush the buffer contents

                     pw.append(s1);                   // to append the character sequence
                     pw.flush();

                     pw.println();

                     pw.printf("Hello I am %s. My age is %d. My lucky number is %d. Good
bye...", name,age,luckynum); //to format the contents
                     pw.flush();

                }
                 catch (FileNotFoundException e) {
                     // TODO Auto-generated catch block
                     e.printStackTrace();
                }

                finally {
                     pw.close();                       // closing PrintWriter
                }
         }
}
```

Output:

```
file contents in pwdemo.txt:
Coder technologies, Vashi
Squad Infotech, Nerul
Hello I am Rahul. My age is 25. My lucky number is 7. Good bye...
```

5. Scanner class:

- can read data from keyboard, disk, file or another source that implements Readable interface.

Constructors:

Scanner(InputStream from)
Scanner(File from) throws FileNotFoundException
Scanner(Readable from)
Scanner(String from)

To read file:

FileReader fin=new FileReader("scandemo.txt")
Scanner sc=new Scanner(fin)

Example:

Suppose file contents in scandemo.txt:

Name: Rajanikant Sharma
Age: 60
Qualification: ME Computer Science. Work Experience: 10 years(Industrial), 20 years(teaching)

```
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.InputStreamReader;
import java.util.Scanner;

public class ScannerDemo {
        public static void main(String[] args) {
                Scanner sc,sc1=null;
                InputStreamReader fread=null;

                try {
                        fread=new FileReader("scandemo.txt");
                        sc=new Scanner(fread);

                        while(sc.hasNextLine())
                        {
                                String s=sc.nextLine();
                                sc1=new Scanner(s);
                                while(sc1.hasNext())
                                {
                                        if(sc1.hasNextInt())
                                        {
                                                System.out.println(sc1.nextInt());
                                        }
                                        else
                                        {
                                                sc1.next();
                                        }
                                }//end inner while
                        }//end outer while
                        } catch (FileNotFoundException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
        }
}
```

Output:

```
60
10
20
```

Example In smplified form:

```
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.InputStreamReader;
import java.util.Scanner;

public class ScannerDemo {
        public static void main(String[] args) {
                Scanner sc,sc1=null;
                InputStreamReader fread=null;
                try {

                        fread=new FileReader("scandemo.txt");
                        sc=new Scanner(fread);
                        while(sc.hasNextLine())
                        {
                                if(sc.hasNextInt())
                                {
                                        System.out.println(sc.nextInt());
                                }
                                else
                        if(sc.hasNext())
                            sc.next();

            }//end outer while
            } catch (FileNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

Example: to find out patterns from file.

```
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.InputStreamReader;
import java.util.Scanner;
public class ScannerPatternDemo {
    public static void main(String[] args) {
        Scanner sc,sc1=null;
        InputStreamReader fread=null;
        try {

            fread=new FileReader("scandemo.txt");
            sc=new Scanner(fread);
            while(sc.hasNextLine())
            {
                String pattern=sc.findInLine("Age:");   // find out given pattern
                    if(pattern!=null)
                    {
                        System.out.println(pattern+sc.nextInt());  //prints age
                    }
                    else
                    sc.nextLine();

            }//end outer while

            } catch (FileNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

6. Random Access File:

- not derived from InputStream or OutputStream
- implements DataInput and DataOutput interfaces, also implements Closeable interface
- we can position file pointer within the file

Constructors:

- RandomAccessFile(File fileObj , String access)throws FileNotFoundException
- RandomAccessFile(String filename, String access)throws FileNotFoundException

  access: white type of access is permitted: r, rw, rws, rwd

Methods:

- **void seek(long newPos) throws IOException :** used to set the current position of the file pointer within the file

  newPos: new position in bytes
- **void setLength(long len) throws IOException :** sets the length of the invoking file to that specified by len

ASSIGNMENTS

1. Implement the program of swapping the contents of two files using FileInputStream and FileOutputStream.
2. Implement the program of swapping the contents of two files using FileReader and FileWriter.
3. Implement program by using RandomAccessFile class.
4. Implement a program to read only integer values from file.
5. Implement a program that will write a formatted string into file.
6. Implement a program to append the contents of one file into another file.