7. Strings, Streams and Files

1) String class and StringBuffer Class
2) Stream classes
   a. Byte Stream classes
   b. Character Stream Classes
3) Using the File class
4) Creation of files
5) Reading/Writing characters and bytes
6) Handling primitive data types
7) Random Access files

1. String class and StringBuffer class

What is String in Java?

Generally, string is a sequence of characters. But in java, string is an object that represents a sequence of characters. String class is used to create string object. It is a predefined class in java.lang package used to handle the String.

How to create String object?

There are two ways to create String object:

1. By string literal
2. By new keyword

1) By String Literal

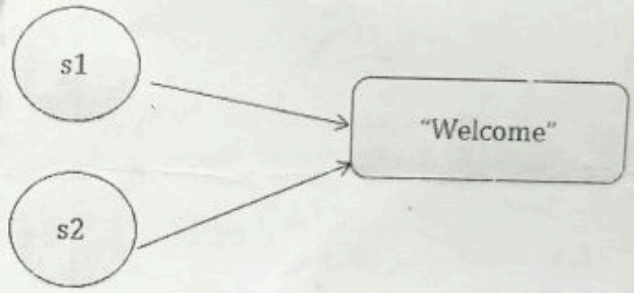
Java String literal is created by using double quotes. For Example:

String s="welcome";

Each time you create a string literal, the Java checks the string already exists or not it simply refers to that string. If string doesn't exist a new string instance is created.

For example:

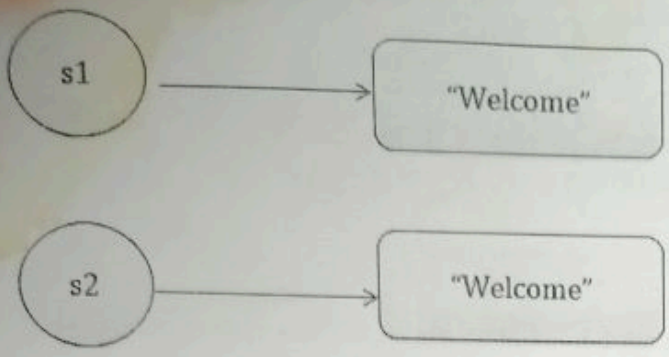
```
String s1="Welcome";
String s2="Welcome";
```



2) By new keyword

```
String s=new String("Welcome");
```

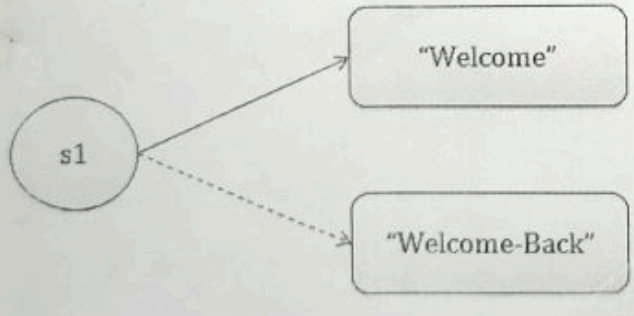
In such case, Java will create a new string object.



*** Immutability in String**

The java String is immutable. Immutability means once we declare a string object then we cant perform any changes in that object. If we try to perform any changes then with those changes a new object will be created.

E.g. s1.concat("back");



Methods of String class.

1. **int length()**

Core Java by - Mr. Kadam R.R

This method returns the length of the specified string

E.g.

```
String s="Welcome";
System.out.println(s.length());// 7
```

2. **String concat(String str)**

This method concatenates two strings.

E.g.

```
String s1="MGM";
String s2="College";
s1.concat(s2);
```

3. **String toLowerCase()**

This method is used to convert a string into lower case letters.

E.g.

```
String s1="Welcome";
System.out.println(s1.toLowerCase());
```

O/P welcome

4. **String toUpperCase()**

This method is used to convert a string into upper case letters.

E.g.

```
String s1="Welcome";
System.out.println(s1.toUpperCase());
```

O/P- WELCOME

5. **int indexOf(int ch)**

This method returns the index of specified character.

E.g.

```
System.out.println(s1.indexOf('l'));// 2
```

6. **String replace(char old, char new)**

This method replaces all the occurrences of old characters with new characters.

E.g.

```
System.out.println(s1.replace('e','a'));
```

O/P – Walcoma

7. **boolean isEmpty()**

It is used to check whether the string is empty or not. If it is empty it returns true otherwise falls.

E.g.

```
String s1="Welcome";
System.out.println(s2.isEmpty());// false
String s2="";
System.out.println(s2.isEmpty());// true
```

8. **boolean equals(Object)**

This method is used to compare two string objects. If two strings are equals it returns true otherwise false.

E.g.

```
String s1="Welcome";
String s2="MGM";
System.out.println(s1.equals(s2));// false
```

9. **boolean equalsIgnoreCase(String)**

```
String s1="Welcome";
String s2="WelcomE";
System.out.println(
s1.equalsIgnoreCase(s2));// true
```

What is StringBuffer class

It is a predefined class in java.lang package can be used to handle the String, whose object is mutable that means content can be modify.

How to create StringBuffer object

We can create StringBuffer object using following syntax.

1. **StringBuffer sb1=new StringBuffer():**

It creates an empty string buffer with the initial capacity of 16.

2. **StringBuffer sb2=new StringBuffer(String str):**

It creates a string buffer with the specified string.

called as *RandomAccessFile*. This class allows

Methods of StringBuffer class

1. **append(String s):**

   It is used to append the specified string with this string.

   E.g.

```
StringBuffer sb1=new
StringBuffer("MGM");
sb1.append(" College");
System.out.println(sb1);// MGM College
```

2. **insert(int offset, String s):**

   It is used to insert the specified string with this string at the specified position.

   E.g.

```
StringBuffer sb1=new
StringBuffer("MGM");
sb1.insert(1,"Welcome");
System.out.println(sb1);          //
```

   MWelcomeGM

3. **replace(int startIndex, int endIndex, String str):**

   It is used to replace the string from specified startIndex and endIndex.

   E.g.

```
StringBuffer sb1=new
        StringBuffer("MGM College");
sb1.replace(1,4,"ITM College");
System.out.println(sb1);
```

   O/P=> MITM CollegeCollege

4. **delete(int startIndex, int endIndex):**

   It is used to delete the string from specified startIndex and endIndex.

```
StringBuffer sb1=new
    StringBuffer("MGM College");
sb1.delete(1,4);
System.out.println(sb1);
```

   O/P- MCollege

5. **reverse():**

   It is used to reverse the string.

```
StringBuffer sb1=new
        StringBuffer("MGM
College");
sb1.reverse();
System.out.println(sb1);
```

   O/P- egelloC MGM

2. Stream classes

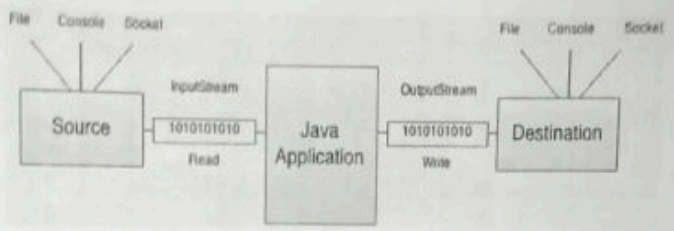
The data in the program can come from different sources such as keyboard, files, sockets (network connections) etc.

Java uses the concept of stream to make I/O operation fast. The java.io package contains all the classes required for input and output operations. We can perform file handling in java by java IO API.

What is Stream?

A stream is a sequence of data. In Java a stream is composed of bytes. It's called a stream because it's like a stream of water that continues to flow.

Whenever we need to make a data transfer we use streams. All the classes that support streams present in java.io.

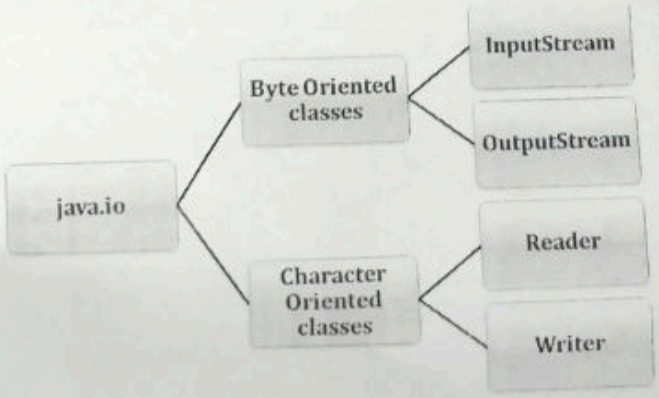


Before transferring data we need to represent it in its equivalent low level format i.e. either in the form of bytes or in the form of characters.

Stream classes

Depending upon the representation of data to be transferred the classes in the java.io package are classified into two types.

1. Byte oriented classes
2. Character Oriented classes



The byte oriented classes are used to read or write data in the form of bytes.

The character oriented classes are used to read or write the data in the form of characters or strings.

a. Byte Stream classes

The byte streams as name implies handle reading and writing the bytes. The java.io package provides two abstract classes i.e. InputStream and OutputStream which contains methods to read and write data in the form of bytes.

1. InputStream class

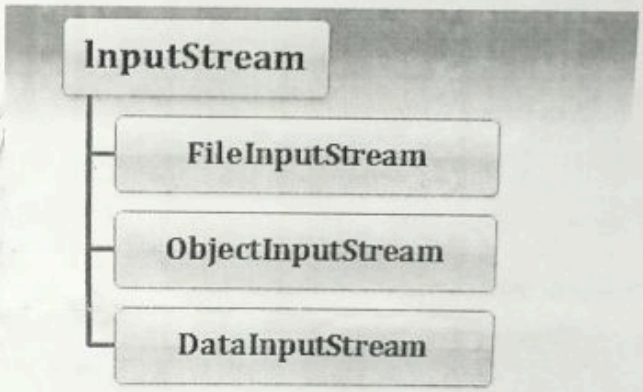
InputStream class is an abstract class. It is the super class of all classes representing an input stream of bytes.

The InputStream class provides the following methods.

Method	Description
1) public abstract int read()	It reads the next byte of data from the input stream. It returns -1 at the end of file.
2) public int available()	It returns number of bytes that can be read from the current input stream.
3) public void close()	It is used to close the current input stream.

Note: All the above methods throws an IOException. Therefore we must use these methods in the try block.

The InputStream class has several sub classes that handles the read or input operations as shown in following fig.



The *FileInputStream* class is used to read the byte data from the file.

The *ObjectInputStream* class is used to read the byte data from the networked system

The *DataInputStream* class is used to read the byte data from the console or command prompt.

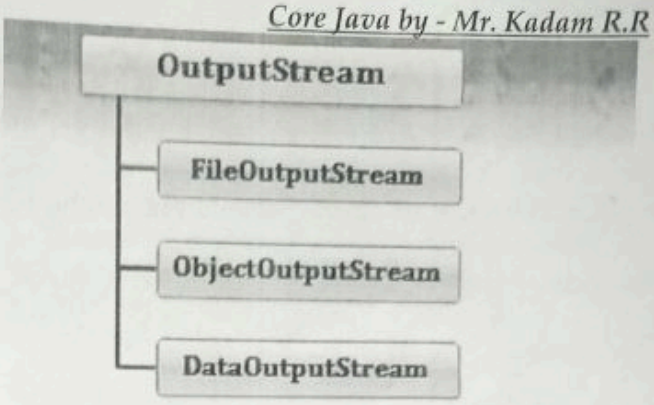
2. OutputStream class

The OutputStream is a base class with all the methods required to write or output the bytes data. The OutputStream class provides the following methods.

Method	Description
1) void write(int)	It is used to write a byte to the current output stream.
2) void write(byte[])	It is used to write an array of byte to the current output stream.
3) void flush()	It flushes the current output stream.
4) void close()	It is used to close the current output stream.

Note: All the above methods throws an IOException. Therefore we must use these methods in the try block.

The OutputStream class has several sub classes that handles the write or output operations as shown in following fig.



The *FileOutputStream* class is used to write the byte data into file.

The *ObjectOutputStream* class is used to write the byte data to another system in the network.

The *DataOutputStream* class is used to write the byte data to the console.

b. Character Stream classes

Character streams work with characters and strings. Character streams are defined by using two super abstract classes. These are:

1. **Reader stream class**
2. **Writer stream class**

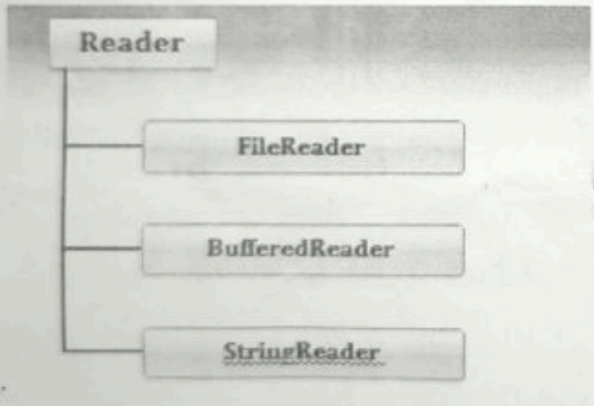
Both these classes are used to input and output characters streams.

1. **Reader stream class**

This is the base class of character oriented input streams. It provides methods to read a sequence of characters. These methods provide the character oriented input operations.

The reader class uses all the methods provided by InputStream class to read the character data.

The Reader class has several sub classes that handles the read or input operations as shown in following fig.



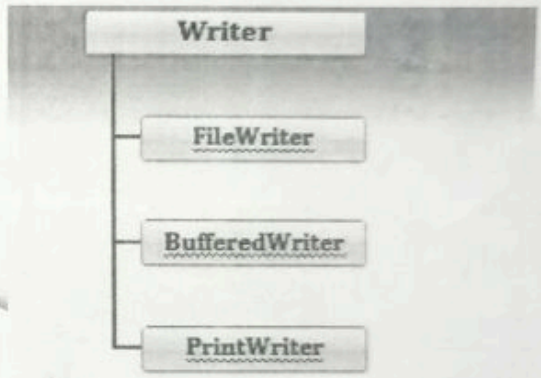
The ***FileReader class*** is used to read the data from the file in the form of character.

The ***BufferedReader class*** is used to read the data from the file in the form of character.

2. Writer stream class

The ***Writer*** is the base class of charact oriented output streams. It provides methods to write a sequence of characters. These methods provide the character oriented output operations.

The ***Writer*** class has several sub classes that handles the write operations as shown in following fig.



The ***FileWriter*** class is extended from Writer class. It is used to write the character data in the file.

The ***PrintWriter*** class is used to write the primitive data in the file like integers, float, boolean etc.

3. Using the File class

The java.io package includes a class known as ***File*** class. The File class provides the

led as

pport for creating files and directories. This class contains several methods for supporting the operations such as

- ✓ Creating a file
- ✓ Opening a file
- ✓ Closing a file
- ✓ Deleting a file
- ✓ Getting the name of a file
- ✓ Getting the size of the file
- ✓ Checking the file is exist or not
- ✓ Renaming a file etc.

When creating files and performing i/o operations on them, the program may generate i/o exceptions such as **FileNotFoundException**, **IOException** etc. Therefore we must have to handle such types of exceptions using try catch blocks.

4. Creation of files

To create a new file we can use File class present in java.io package. Following are the constructors of File class.

Syntax1:


```
File objectName=new File(String fileName);
```

In the above syntax the fileName indicates the name of the file with extension.

E.g1.

```
File f1=new File("abc.txt");
```

The above statement will not create any file. First it check, whether the file is already available or not if it is already available then f1 objects simply refers to that file. If it is not already available then f1 simply represent the name of the file and it will not create any file. To create a new file the File class provides a method called **createNewFile()** i.e **f1.createNewFile();**

Syntax2:

```
File objectName=new File(String subDir, String name);
```

The above syntax is used to create a new file in the specified sub directory

E.g.

```
File f1=new File("mgm" , "abc.txt");
```

Methods of File class

1. **boolean exists()-** This method is used to check whether the file or directory is already available or not.
2. **boolean createNewFile()-** This method is used to create a new file.
3. **boolean mkdir()-** This method is used to create a new directory.
4. **boolean delete()-** This method is used to delete the file.

5. **Reading/Writing characters**

The Reader and Writer classes implements streams that can handle characters. The two sub-classes used for handling characters in the files are FileReader and FileWriter.

a. Reading characters from file using FileReader class

The ***FileReader*** class is similar to the FileInputStream class. It is present in java.io package. The class FileReader reads characters from the file. It throws IOException, therefore must be used in try block.

Syntax:

```
FileReader fr=new FileReader(String fileName);
```

E.g.

```
FileReader fr=new FileReader("xyz.txt");
```

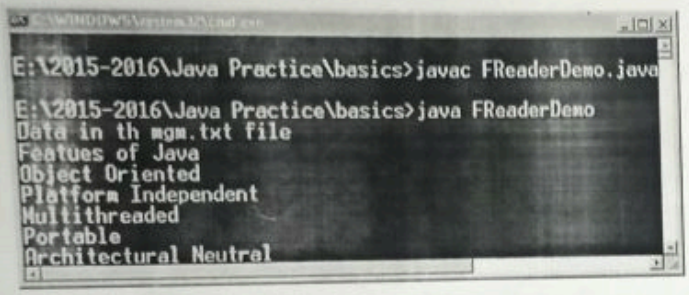
The above example reads the character data from the file xyz.txt. To read the data character by character the read() function is used.

/*W.a.p to read the character data from the file*/

```
import java.io.*;
class FReaderDemo
{
public static void main(String[] args)
{
try
{
File f1=new File("mgm.txt");
FileReader fr=new FileReader(f1);
System.out.println("Data in th mgm.txt file");
for(int i=0; i<f1.length(); i++)
{
System.out.print((char)fr.read());
}
}
```

```
atch(IOException e1)
{
System.out.println(e1);
}
}
}
```

O/P



b. Writing characters into the file using FileWriter class

The FileWriter class is present in java.io. package. This class is used to write the character data to the file. It throws an IOException, therefore must be used in try block.

Syntax:

```
1. FileWriter fr=new FileWriter (String fileName);
2. FileWriter fr=new FileWriter (String fName,boolean append);
```

The first syntax is meant for overwriting the contents of file. If we want to perform append then we have to use second syntax which takes two parameters first is filename and second is boolean append.

E.g.

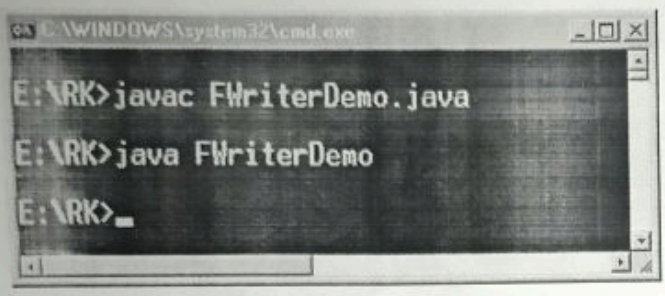
```
1. FileWriter fr=new FileWriter ("abc.txt");
2. FileWriter fr=new FileWriter ("abc.txt",true);
```

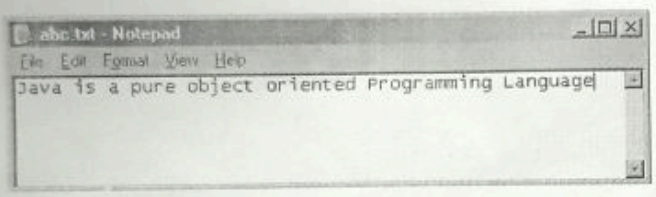
The second example appends the file abc.txt with previous contents.

/*W.a.p to write the character data into the file using FileWriter class */

```
import java.io.*;
class FWriterDemo
{
public static void main(String[] args)
{
try
{
FileWriter fw=new FileWriter("abc.txt");
String data="Java is a pure object oriented
                Programming Language";
fw.write(data);
fw.close();
}
catch(IOException e1)
```

```
{
System.out.println(e1);
}
}
}
```





6. Reading/Writing Bytes

The InputStream and OutputStream classes provide the methods to read and write the data in the form of bytes.

a. *Reading byte data from file using FileInputStream class*

The *FileInputStream* class extends from InputStream class. It is used to read the binary data from a file. It throws FileNotFoundException, therefore must be used in try block. To read the byte data from the file this class uses the **int read()** method.

Syntax:

```
FileInputStream f=new FileInputStream(String fileName);
```

E.g.

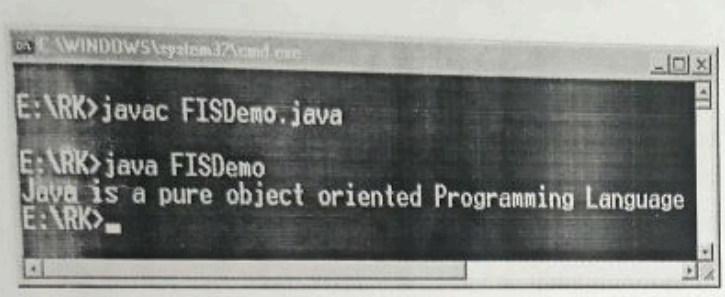
```
FileInputStream f=new FileInputStream("abc.txt");
```

/* Program to read the data from the file using FileInputStream class.*/

```
import java.io.*;
class FISDemo
{
public static void main(String[] args)
{
try
{
FileInputStream fistream=new
                    FileInputStream("abc.txt");
int size=fistream.available();
for(int i=0; i<size; i++)
{
System.out.print((char)fistream.read());
}
fistream.close();
}
catch(IOException e1)
```

```
System.out.println("Exception = "+e1);
}
}
}
```

O/P



b. *Reading byte data from console using DataInputStream class*

The *DataInputStream* is used to read the data from the command prompt or console. It extends from InputStream class. It throws IOException, therefore must be used in try block.

This class uses the readLine() method to read the data.

Syntax:

```
DataInputStream dis=new DataInputStream(System.in);
```

E.g.

```
DataInputStream dis=new DataInputStream(System.in);
String s1=dis.readLine();
```

/* Program to read the data from the console using DataInputStream class.*/

```
import java.io.*;
class DIStreamDemo
{
public static void main(String[] args)
{
try
{
DataInputStream dis=new
        DataInputStream(System.in);
System.out.println("Enter a String ");
String s1=dis.readLine();
System.out.print("You have entered = "+s1);
}
catch(IOException e1)
{
System.out.println("Exception = "+e1);
}
}
}
```

O/P

```
E:\RK>javac DIStreamDemo.java
Note: DIStreamDemo.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

E:\RK>java DIStreamDemo
Enter a String
Good Morning MGM
You have entered = Good Morning MGM
E:\RK>
```

c. *Writing byte data into the file using FileOutputStream class*

It is sub class of OutputStream class. It is present in java.io package; It is used to write the byte data in the file. It throws FileNotFoundException, therefore must be used in try block. This class uses the write() method to write the data.

Syntax:

```
FileOutputStream fs=new FileOutputStream(String fileName);
```

E.g.

```
FileOutputStream fostream=new FileOutputStream("xyz.txt");
```

// Program to write the data to the file using FileOutputStream class.

```
import java.io.*;
import java.util.*;
class FOStreamDemo
{
public static void main(String[] args)
{
try
{
FileOutputStream fostream=new
        FileOutputStream("xyz.txt");
DataInputStream dim=new
        DataInputStream(System.in);
System.out.println("Enter a String ");
String s1=dim.readLine();
byte b1[]=s1.getBytes();
fostream.write(b1);
}
catch(IOException e1)
{
System.out.println("Exception = "+e1);
}
}
}
```

O/P

```
E:\RK>javac FOStreamDemo.java
Note: FOStreamDemo.java uses or o
Note: Recompile with -Xlint:depre

E:\RK>java FOStreamDemo
Enter a String
Sachin is my favorite player
```

```
xyz.txt - Notepad
File Edit Format View Help
Sachin is my favorite player
```

7. Handling primitive data types

The basic input and output streams provide read/write methods. These methods can only be used for reading and writing bytes or characters. If we want to read or write the primitive data types such as integers, doubles etc. then we can use ***DataInputStream and FileInputStream*** classes.

The ***DataInputStream*** is a high level stream used to read primitive data types. When you read bytes, you have to use methods to convert them to characters to understand their meaning. But this class provides methods to directly work with primitive types. The following are the commonly used methods of this class.

Methods to read primitive values

Methods	Description
1. byte readByte()	This method is used to read byte value.
2. int readInt()	This method is used to read int value.
3. float readFloat()	This method is used to read float value.
4. boolean readBoolean()	This method is used to read boolean value.
5. char readChar()	This method is used to read char value.

Methods to write primitive values

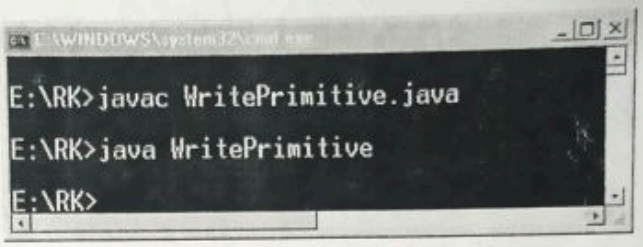
Methods	Description
1. void writeByte()	This method is used to write byte value.
2. void writeInt()	This method is used to write int value.
3. void writeFloat()	This method is used to write float value.
4. void	This method is used to

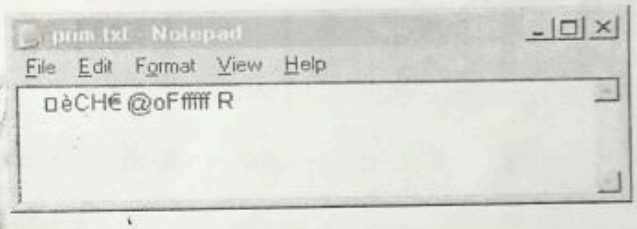
writeBoolean()	write boolean value.
5. void writeChar()	This method is used to write char value.

// W.a.p to write primitive data to the file

```
import java.io.*;
class WritePrimitive
{
public static void main(String[] args)
{
try
{
// Write primitive data to the prim.txt file
FileOutputStream fos=new
        FileOutputStream("prim.txt");
DataOutputStream dos=new
        DataOutputStream(fos);
dos.writeInt(1000);
dos.writeFloat(200.50f);
dos.writeDouble(250.20);
dos.writeChar('R');
dos.close();
}
catch(IOException e1)
{
System.out.println(e1);
}
}
}
```

O/P

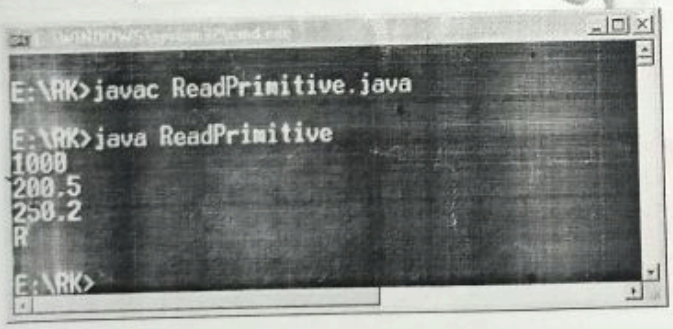




// W.a.p to read primitive data from the file

```
import java.io.*;
class ReadPrimitive
{
public static void main(String[] args)
{
try
{
// read primitive data from the prim.txt file
FileInputStream fis=new
        FileInputStream("prim.txt");
```

```
DataInputStream dis=new
DataInputStream(fis);
System.out.println(dis.readInt());
System.out.println(dis.readFloat());
System.out.println(dis.readDouble());
System.out.println(dis.readChar());
dis.close();
}
catch(IOException e1)
{
System.out.println(e1);
}
}
}
```



8. Random Access files

The general i/o stream classes used either for read only or write only operations and not for both purpose simultaneously. These files are read and written sequentially and therefore are known as *sequential files.* Bu using general i/o classes we cant read or write data randomly.

The java.io package provides a class called as ***RandomAccessFile.*** This class allows us to read and write data randomly in the file. That is, we can jump around the file. Such files are known as *random access files.*

The following is the syntax of creating the object of RandomAccesssFile class.

Syntax:

```
RandomAccessFile rf=new RandomAccessFile(String filename,
                                         String read/writemode);
```

E.g.

In the above syntax the RandomAccessFile class takes two arguments. The first is the filename and second is the mode of operation i.e. read or write. We can use one of the following mode of string.

"r" for reading only.

"rw" for reading and writing.

E.g.

```
RandomAccessFile rf=new RandomAccessFile("myfile.txt","rw");
```

This example crates a object with read and write mode operations.

The *RandomAccessFile* class has following methods.

1. **long getFilePointer()**

   This method returns the current position of file pointer.

2. **long length()**

   This method returns the length of the file.

3. **void seek(long position)**

   This method sets the current position of file pointer within the file. Files starts at position 0.

Method to read the data

1. **int read()**

   This method returns the next byte from the file.

Methods to write the data

2. **void write(int)**

   This method writes the data from the file.

Methods	Description
1. byte readByte()	This method is used to read byte value.
2. int readInt()	This method is used to read int value.
3. float readFloat()	This method is used to read float value.
4. boolean readBoolean()	This method is used to read boolean value.
5. char readChar()	This method is used to read char value.

Methods to write primitive values

Methods	Description
1. void writeByte()	This method is used to write byte value.
2. void writeInt()	This method is used to write int value.
3. void writeFloat()	This method is used to write float value.
4. void writeBoolean()	This method is used to write boolean value.