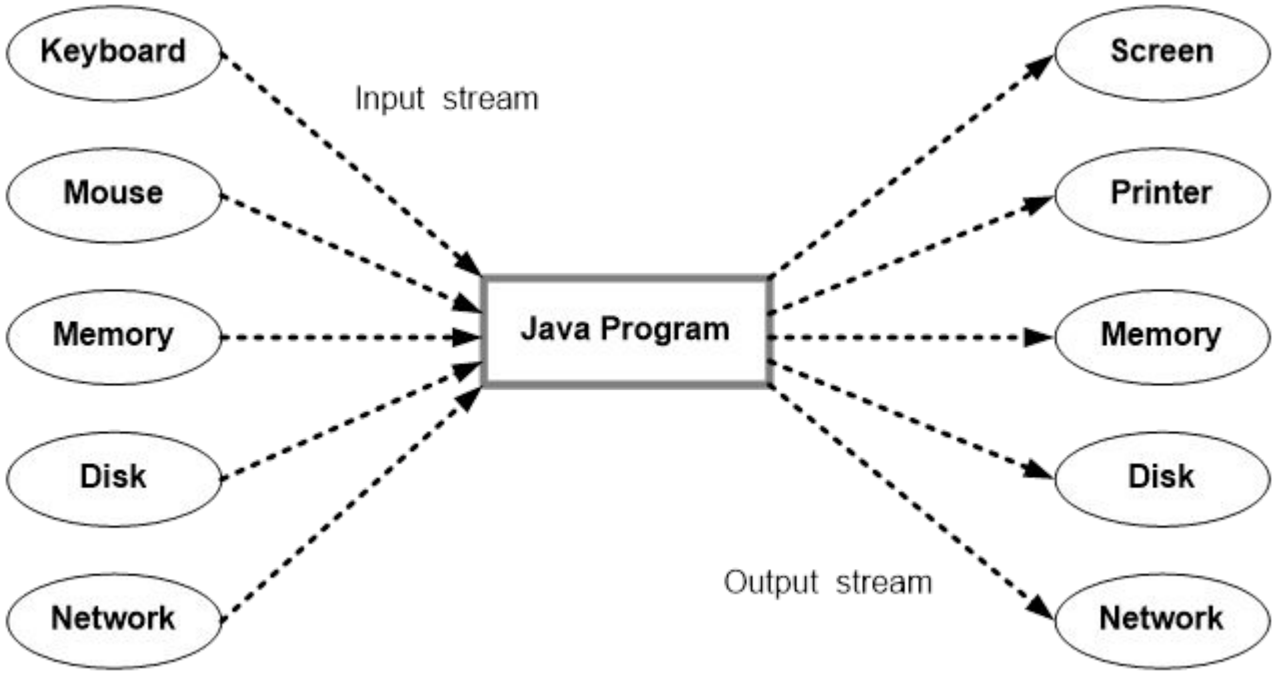
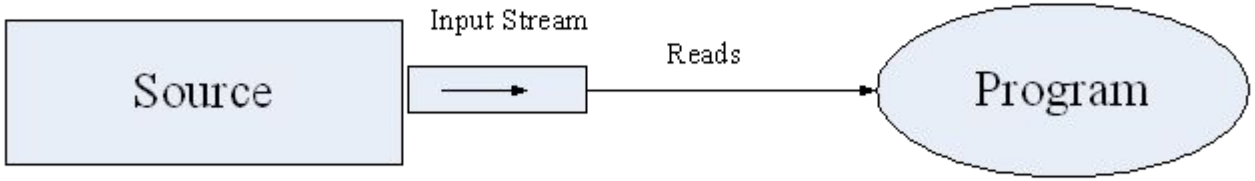
OOPs with Java

File Handling

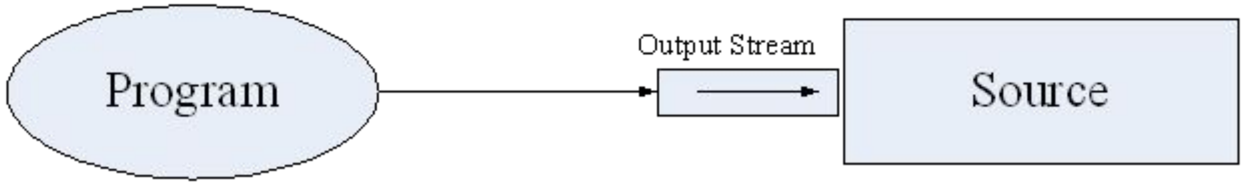
I/O in JAVA

- Java I/O (Input and Output) is used to process the input and produce the output.
- 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 I/O API.
-


Keyboard
Mouse
Memory
Disk
Network
Input stream
Java Program
Screen
Printer
Memory
Disk
Network
Output stream



(a) Reading data into a program



(b) Writing data to a destination

Stream

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

- In java, 3 streams are created for us automatically. All these streams are attached with console.

    1) System.out: standard output stream

    2) System.in: standard input stream

    3) System.err: standard error stream

- The java.io package contains a large number of stream classes to support the streams
- This package provides system input and output through data streams, serialization and the file system.
    - Java I/O (Input and Output) is used to process the input and produce the output.
    - Java uses the concept of a 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.

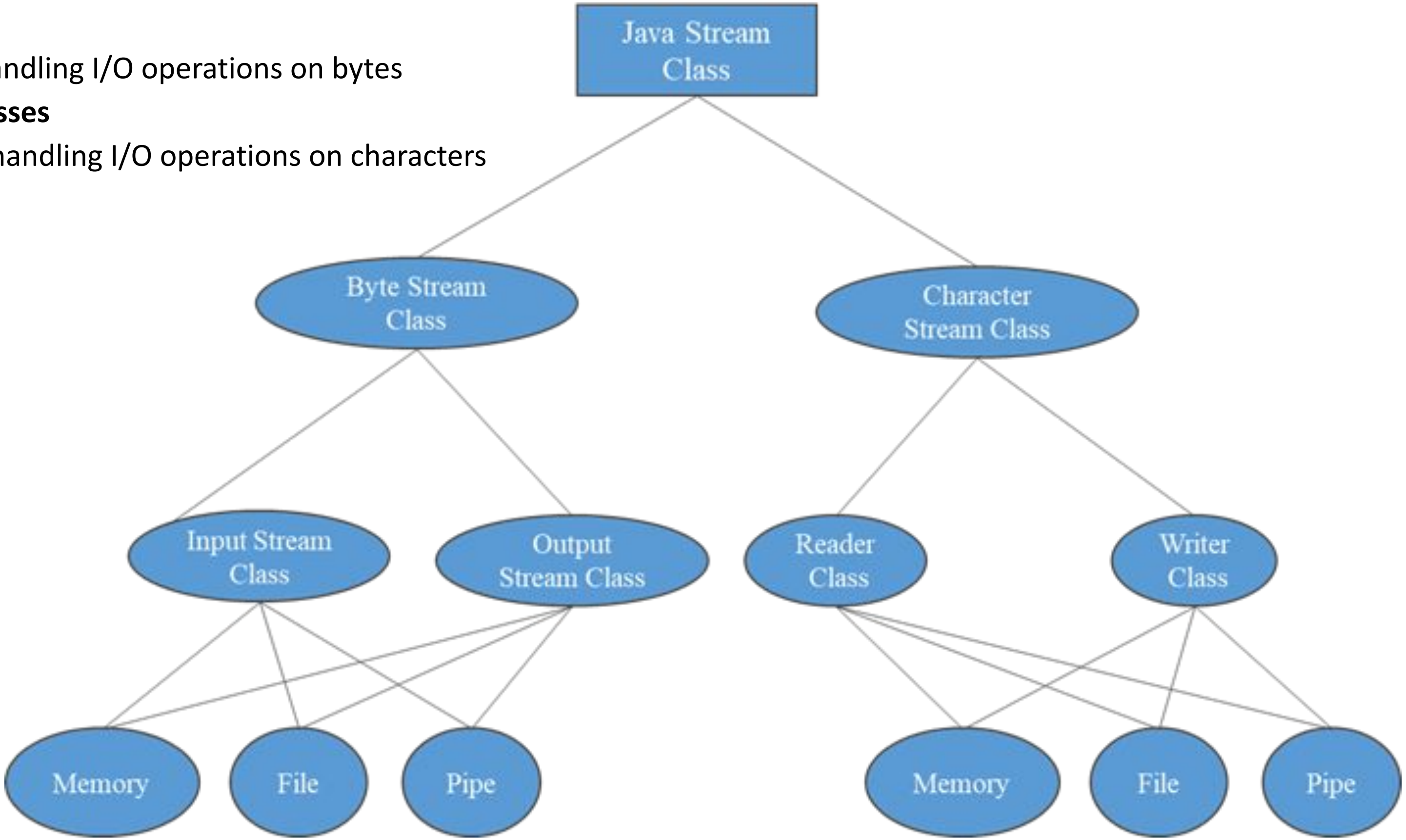
Interfaces in Package java.io

Interface	Description
Closeable	A Closeable is a source or destination of data that can be closed.
DataInput	The DataInput interface provides for reading bytes from a binary stream and reconstructing from them data in any of the Java primitive types.
DataOutput	The DataOutput interface provides for converting data from any of the Java primitive types to a series of bytes and writing these bytes to a binary stream.
Externalizable	Only the identity of the class of an Externalizable instance is written in the serialization stream and it is the responsibility of the class to save and restore the contents of its instances.
FileFilter	A filter for abstract pathnames.
FilenameFilter	Instances of classes that implement this interface are used to filter filenames.
Flushable	A Flushable is a destination of data that can be flushed.
ObjectInput	ObjectInput extends the DataInput interface to include the reading of objects.
ObjectInputValidation	Callback interface to allow validation of objects within a graph.
ObjectOutput	ObjectOutput extends the DataOutput interface to include writing of objects.
ObjectStreamConstants	Constants written into the Object Serialization Stream.
Serializable	Serializability of a class is enabled by the class implementing the java.io.Serializable interface.

I-O STREAM CLASSES IN JAVA

Java provides java.io package which contains a large number of stream classes to process all types of data:

- **Byte stream classes**
  - Support for handling I/O operations on bytes
- **Character stream classes**
  - Supports for handling I/O operations on characters



Methods in Character Stream (reader) classes

Method	Description
close()	Closes the stream and releases any system resources associated with it.
mark(int readAheadLimit)	Tells whether this stream supports the mark() operation.
markSupported()	Tells whether this stream supports the mark() operation.
read()	Reads a single character.
read(char[] cbuf)	Reads characters into an array.
read(char[] cbuf, int off, int len)	Reads characters into a portion of an array.
read(CharBuffer target)	Attempts to read characters into the specified character buffer.
ready()	Tells whether this stream is ready to be read.
reset()	Resets the stream.
skip(long n)	Skips characters.

Methods in Character Stream (Writer) classes

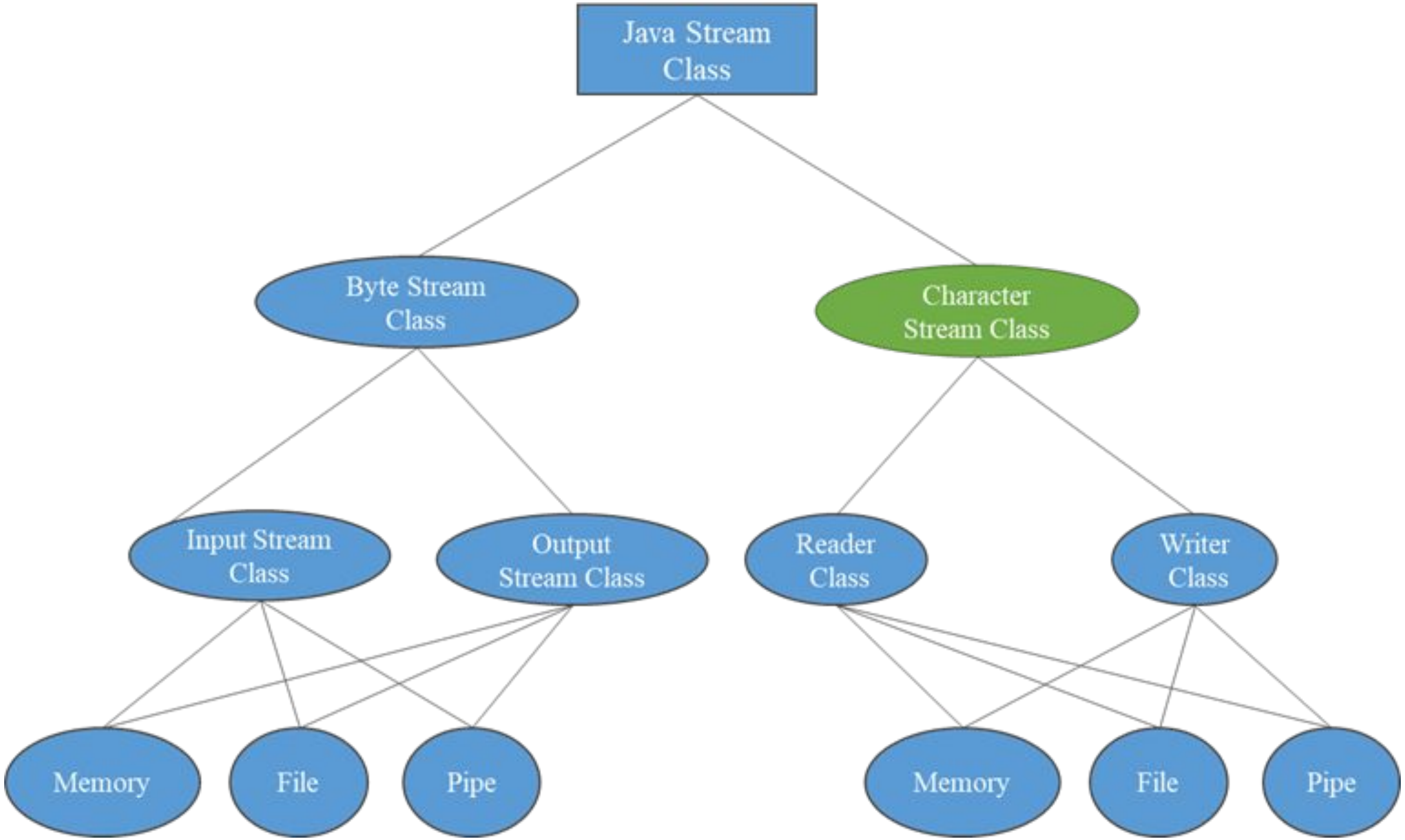
Method	**Description**
append(char c)	Appends the specified character to this writer.
append(CharSequence csq)	Appends the specified character sequence to this writer.
append(CharSequence csq, int start, int end)	Appends a subsequence of the specified character sequence to this writer.
close()	Closes the stream, flushing it first.
flush()	Flushes the stream.
write(char[] cbuf)	Writes an array of characters.
write(char[] cbuf, int off, int len)	Writes a portion of an array of characters.
write(int c)	Writes a single character.
write(String str)	Writes a string.
write(String str, int off, int len)	Writes a portion of a string.

IO WITH CHARACTER STREAMS

Character Stream Classes

- In Western locales, the local character set is usually an 8-bit superset of ASCII.
- The Java platform stores character values using Unicode conventions.
- Character stream I/O automatically translates this internal format to and from the local character set.

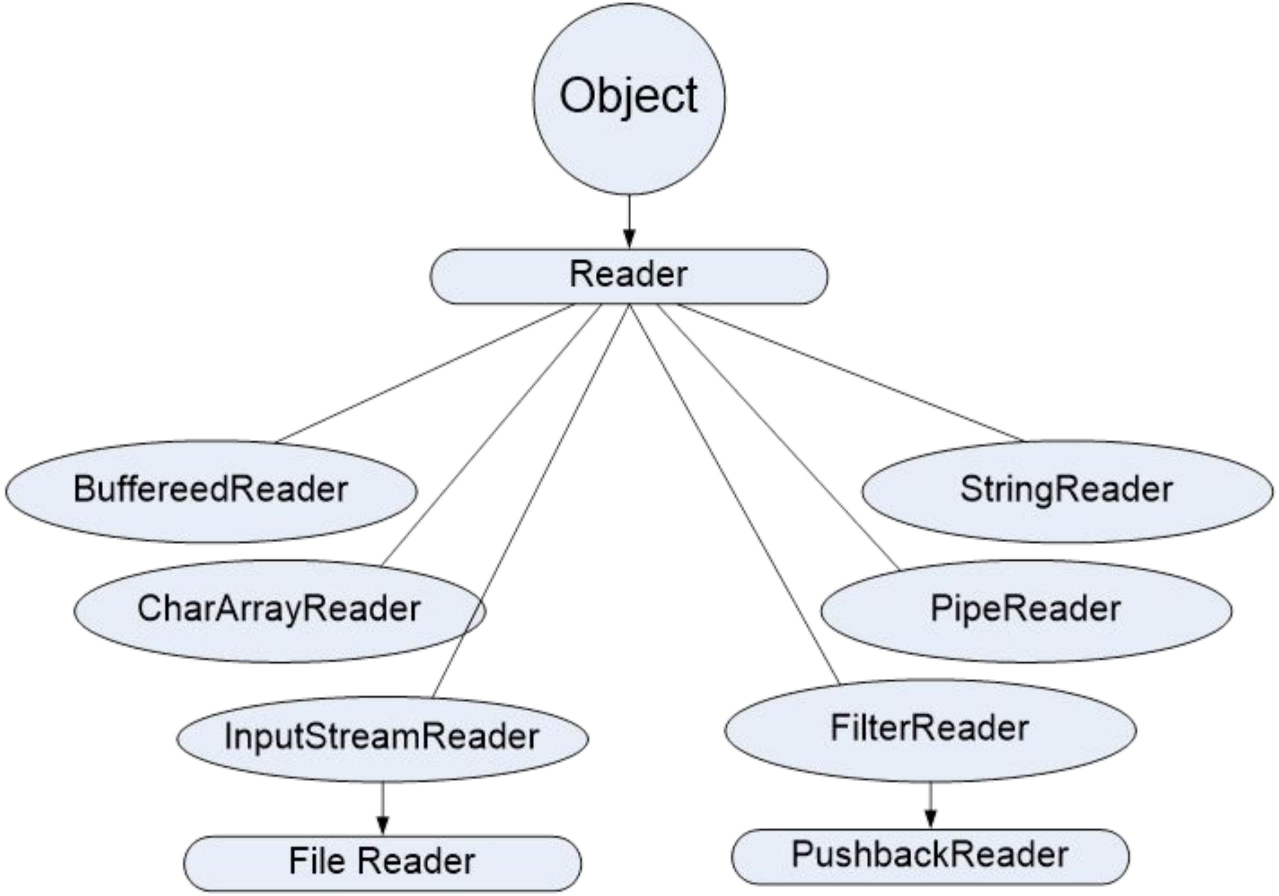
Taxonomy: Java stream classes



Character stream classes is used to read and write characters and supports a number of input-output related methods

- **Reader stream classes**
  - To read characters from files.
  - In many way, identical to InputStream classes.
- **Writer stream classes**
  - To write characters into files.
  - In many way, identical to OutputStream classes.

READER STREAM CLASSES



- Abstract class for reading character streams.

```
public abstract class Reader extends Object implements
Readable, Closeable
```

- The only methods that a sub class must implement are *read(char[], int, int)* and *close()*.
- Most sub classes, however, will override some of the methods defined here in order to provide higher efficiency, additional functionality, or both.

Different classes under the Reader class

Method	Description
BufferedReader	Reads text from a character-input stream, buffering characters so as to provide for the efficient reading of characters, arrays, and lines.
CharArrayReader	This class implements a character buffer that can be used as a character-input stream.
InputStreamReader	An InputStreamReader is a bridge from byte streams to character streams: It reads bytes and decodes them into characters using a specified **charset**.
PipedReader	Piped character-input streams.
FilterReader	Abstract class for reading filtered character streams.
PushbackReader	A character-stream reader that allows characters to be pushed back into the stream.
StringReader	A character stream whose source is a string.
LineNumberReader	A buffered character-input stream that keeps track of line numbers.
FileReader	Convenience class for reading character files.

Constructors and Fields in Reader class

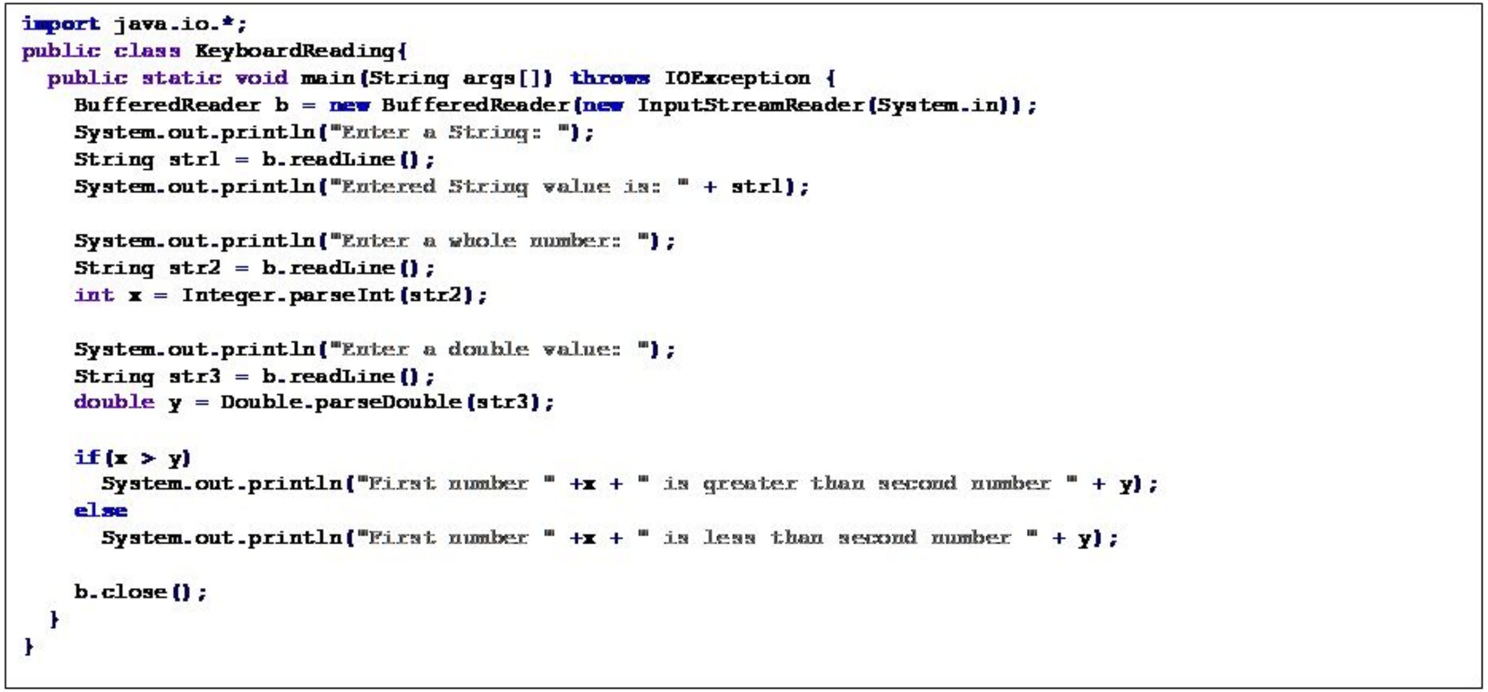
Fields	
Field	***Description***
protected Object **lock**	The object used to synchronize operations on this stream.

Constructor	**Description**
Reader()	Creates a new character-stream reader whose critical sections will synchronize on the reader itself.
Reader(Object lock)	Creates a new character-stream reader whose critical sections will synchronize on the given object.

Methods in Reader class

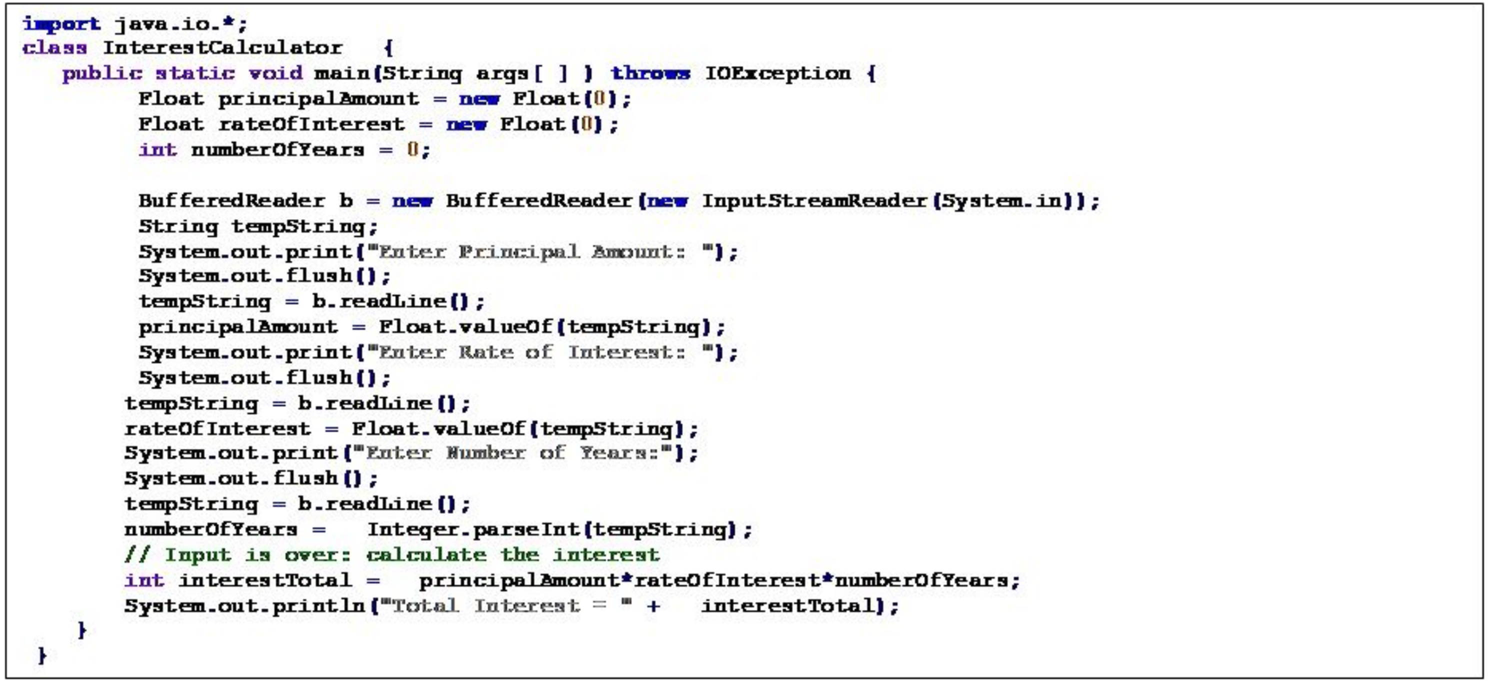
Method	Description
close()	Closes the stream and releases any system resources associated with it.
mark(int readAheadLimit)	Tells whether this stream supports the mark() operation.
markSupported()	Tells whether this stream supports the mark() operation.
read()	Reads a single character.
read(char[] cbuf)	Reads characters into an array.
read(char[] cbuf, int off, int len)	Reads characters into a portion of an array.
read(CharBuffer target)	Attempts to read characters into the specified character buffer.
ready()	Tells whether this stream is ready to be read.
reset()	Resets the stream.
skip(long n)	Skips characters.

Reading String, Integer and Double from Keyboard



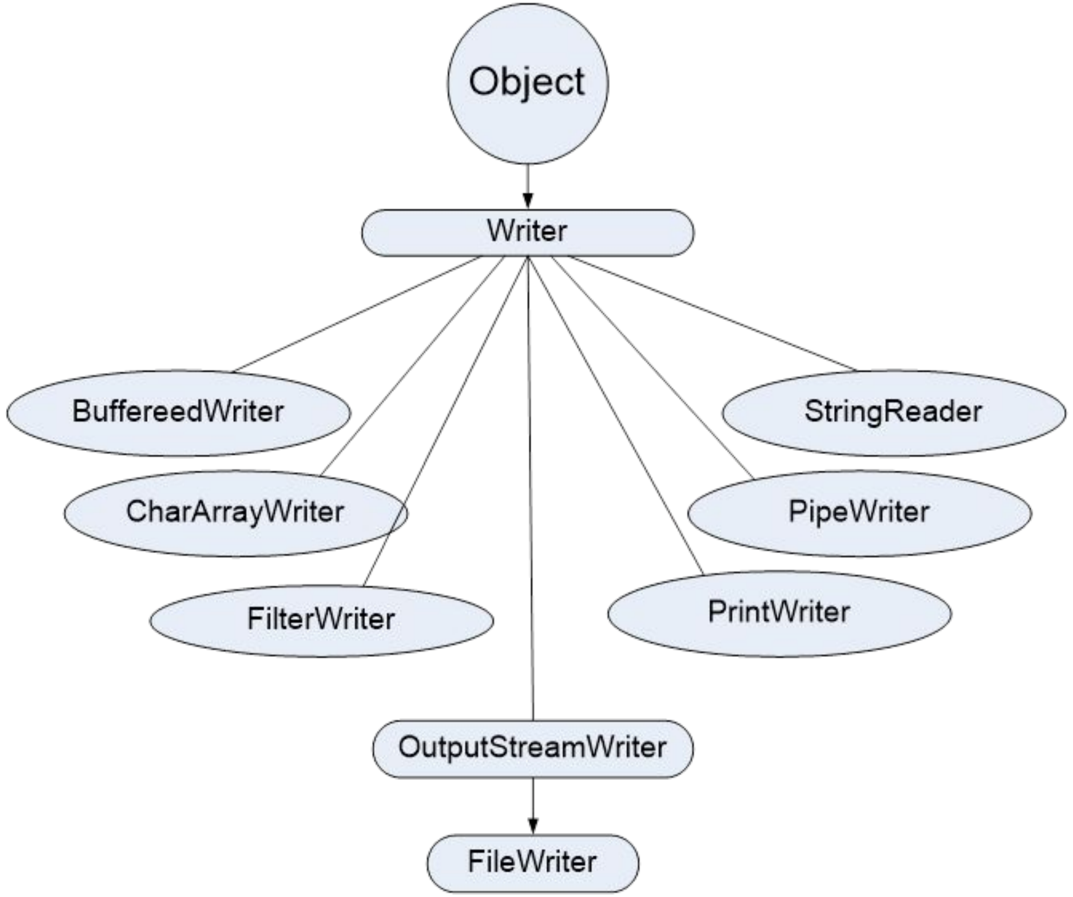
```java
import java.io.*;
public class KeyboardReading{
  public static void main(String args[]) throws IOException {
    BufferedReader b = new BufferedReader(new InputStreamReader(System.in));
    System.out.println("Enter a String: ");
    String str1 = b.readLine();
    System.out.println("Entered String value is: " + str1);

    System.out.println("Enter a whole number: ");
    String str2 = b.readLine();
    int x = Integer.parseInt(str2);

    System.out.println("Enter a double value: ");
    String str3 = b.readLine();
    double y = Double.parseDouble(str3);

    if(x > y)
      System.out.println("First number " +x + " is greater than second number " + y);
    else
      System.out.println("First number " +x + " is less than second number " + y);

    b.close();
  }
}
```

Calculator using BufferedReader



```
import java.io.*;
class InterestCalculator   {
   public static void main(String args[ ] ) throws IOException {
         Float principalAmount = new Float(0);
         Float rateOfInterest = new Float(0);
         int numberOfYears = 0;

         BufferedReader b = new BufferedReader(new InputStreamReader(System.in));
         String tempString;
         System.out.print("Enter Principal Amount: ");
         System.out.flush();
         tempString = b.readLine();
         principalAmount = Float.valueOf(tempString);
         System.out.print("Enter Rate of Interest: ");
         System.out.flush();
        tempString = b.readLine();
        rateOfInterest = Float.valueOf(tempString);
        System.out.print("Enter Number of Years:");
        System.out.flush();
        tempString = b.readLine();
        numberOfYears =   Integer.parseInt(tempString);
        // Input is over: calculate the interest
        int interestTotal =   principalAmount*rateOfInterest*numberOfYears;
        System.out.println("Total Interest = " +   interestTotal);
    }
 }
```

WRITER STREAM CLASSES



- Abstract class for writing character streams.
  - `public abstract class Writer extends Object implements Appendable, Closeable, Flushable`
- The only methods that a subclass must implement are *write(char[], int, int)*, *flush()*, and *close()*.
- Most subclasses, however, will override some of the methods defined here in order to provide higher efficiency, additional functionality, or both.

Sub classes of the Writer class

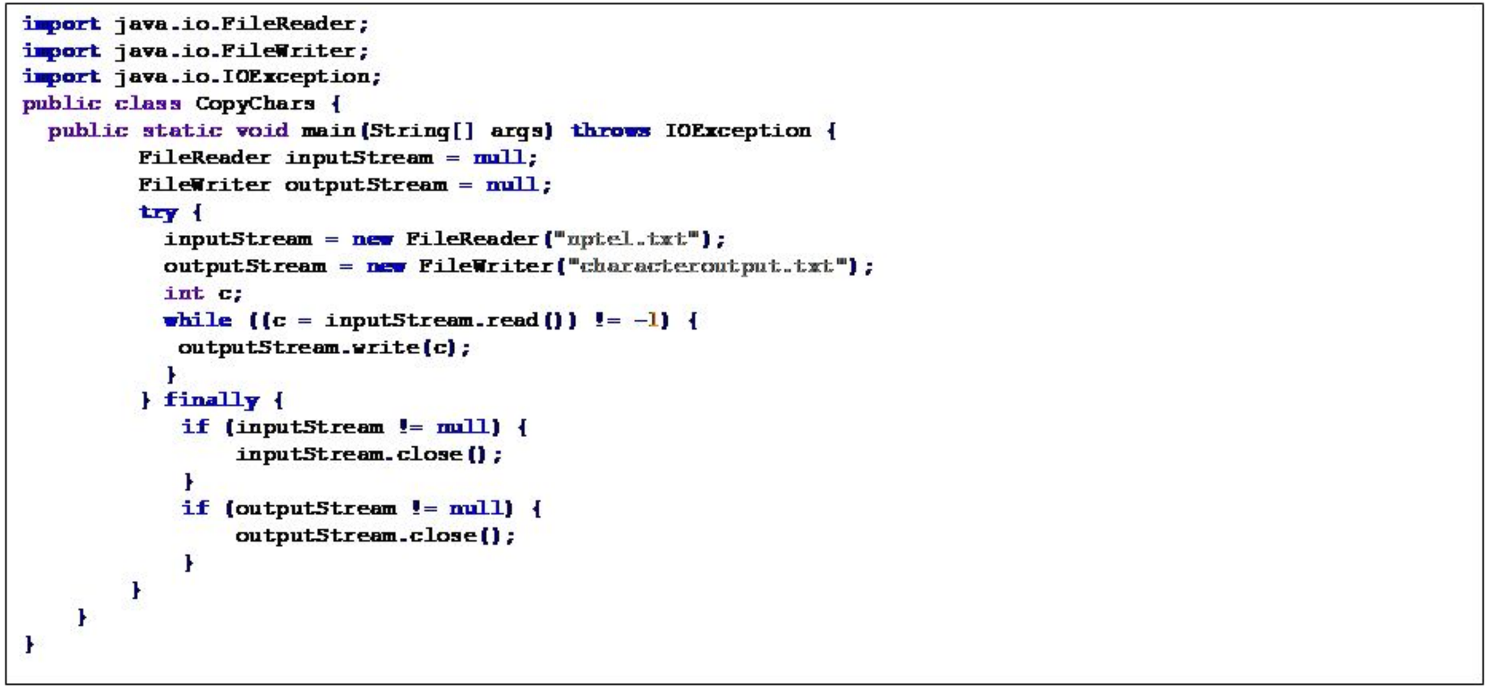
Class	Description
BufferedWriter	Writes text to a character-output stream, buffering characters so as to provide for the efficient writing of single characters, arrays, and strings.
CharArrayWriter	This class implements a character buffer that can be used as an Writer.
PipedWriter	Piped character-output streams.
StringWriter	A character stream that collects its output in a string buffer, which can then be used to construct a string.
FileWriter	Convenience class for writing character files.
FilterWriter	Abstract class for writing filtered character streams.
PrintWriter	Prints formatted representations of objects to a text-output stream.

Constructors and Fieldsin Writer class

Fields	
Field	***Description***
protected Object **lock**	The object used to synchronize operations on this stream.

Constructor	**Description**
Writer()	Creates a new character-stream writer whose critical sections will synchronize on the writer itself.
Writer(Object lock)	Creates a new character-stream writer whose critical sections will synchronize on the given object.

Copying files using FileReader and FileWriter



```java
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
public class CopyChars {
  public static void main(String[] args) throws IOException {
        FileReader inputStream = null;
        FileWriter outputStream = null;
        try {
          inputStream = new FileReader("nptel.txt");
          outputStream = new FileWriter("characteroutput.txt");
          int c;
          while ((c = inputStream.read()) != -1) {
           outputStream.write(c);
          }
        } finally {
            if (inputStream != null) {
                inputStream.close();
            }
            if (outputStream != null) {
                outputStream.close();
            }
        }
    }
}
```

CopChars versus CopyBytes

- **CopyChars** is very similar to **CopyBytes**. The most important difference is that **CopyChars** uses **FileReader** and **FileWriter** for input and output in place of **FileInputStream** and **FileOutputStream**.
- Both CopyBytes and CopyChars use an *int* variable to read to and write from.
- In CopyChars, the int variable holds a character value in its last 16 bits;
- In CopyBytes, the int variable holds a byte value in its last 8 bits.

Summary of Java Streams: List of important tasks and their classes

Task	*Character Stream Class*	*Byte Stream Class*
Performing input operations	Reader	InputStream
Buffering input	BufferedReader	BufferedInputStream
Keeping track of line numbers	LineNumberReader	LineNumberInputStream
Reading from an array	CharArrayReader	ByteArrayInputStream
Translating byte stream into a character stream	InputStreamReader	(none)
Reading from files	FileReader	FileInputStream
Filtering the input	FilterReader	FilterInputStream
Pushing back characters/bytes	PushbackReader	PushbackInputStream
Reading from a pipe	PipedReader	PipedInputStream
Reading from a string	StringReader	StringBufferInputStream

Task	***Character Stream Class***	***Byte Stream Class***
Reading primitive types	(none)	DataInputStream
Performing output operations	Writer	OutputStream
Buffering output	BufferedWriter	BufferedOutputStream
Writing to an array	CharArrayWriter	ByteArrayOutputStream
Filtering the output	FilterWriter	FilterOutputStream
Translating character stream into a byte stream	OutputStreamWriter	(none)
Writing to a file	FileWriter	FileOutputStream
Printing values and objects	PrintWriter	PrintStream
Writing to a pipe	PipedWriter	PipedOutputStream
Writing to a string	StringWriter	(none)
Writing primitive types	(none)	DataOutputStream

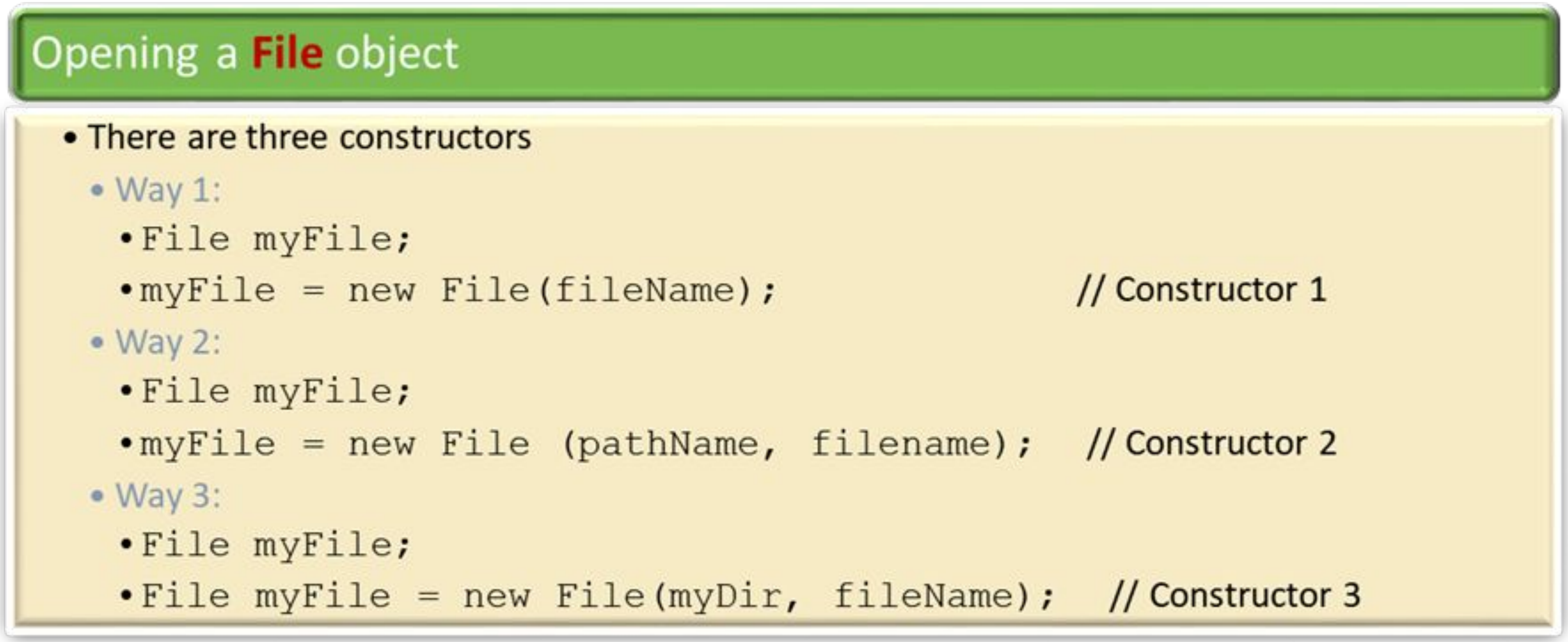
Java File I/O

Java provides java.io package which includes numerous class definitions and methods to manipulate file and flow of data (called File I/O streams)

There are four major classes:

- ***File***
- ***FileInputStream***
- ***FileOutputStream***
- ***RandomAccessFile***

Using class File



Opening a **File** object

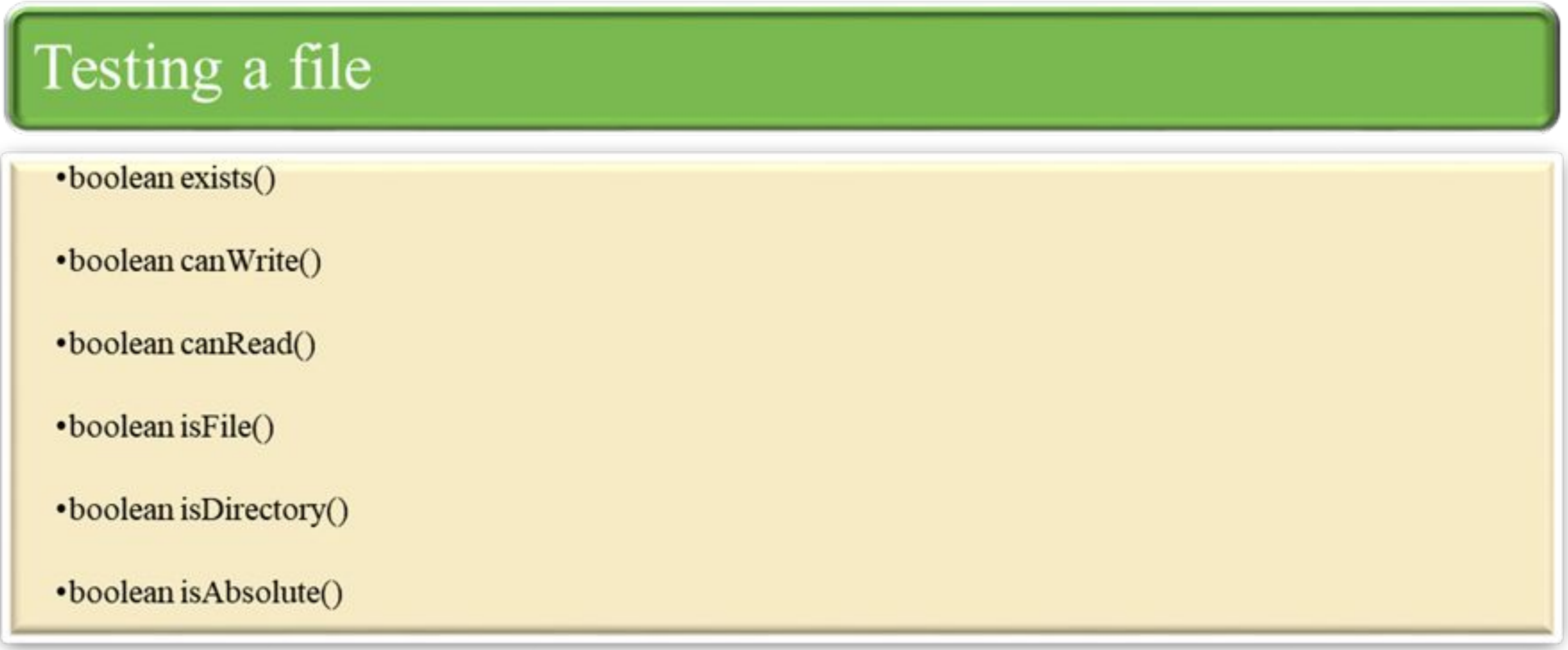
- There are three constructors
  - Way 1:
    - `File myFile;`
    - `myFile = new File(fileName);` // Constructor 1
  - Way 2:
    - `File myFile;`
    - `myFile = new File (pathName, filename);` // Constructor 2
  - Way 3:
    - `File myFile;`
    - `File myFile = new File(myDir, fileName);` // Constructor 3

Using class File

Dealing with file names

- String getName()
- String getPath()
- String getAbsolutePath()
- String getParent()
- boolean renameTo(File newFilename)

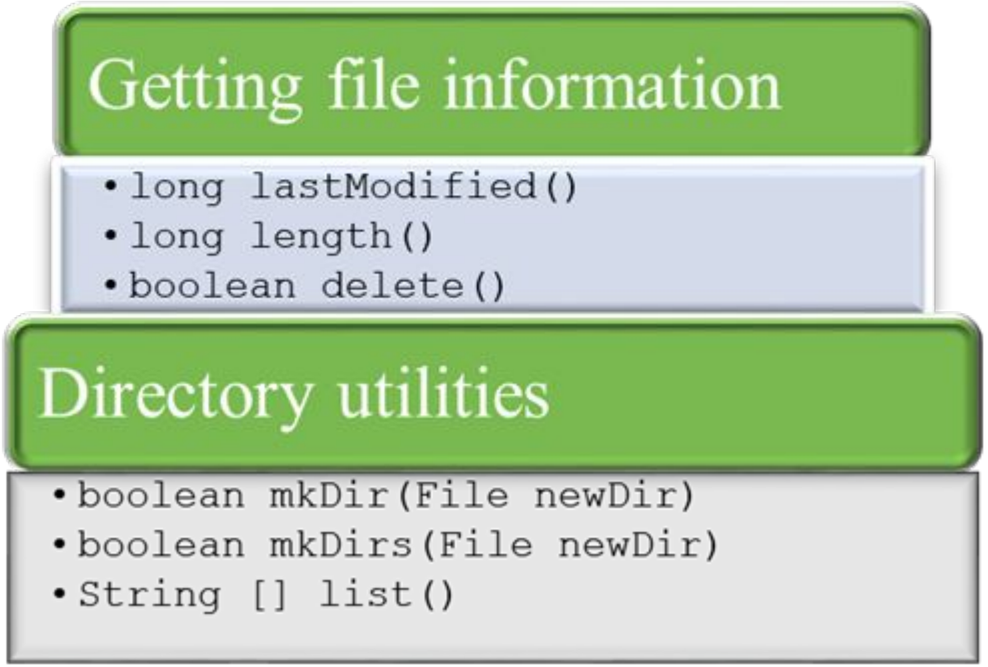
Using class File



Testing a file

- •boolean exists()
- •boolean canWrite()
- •boolean canRead()
- •boolean isFile()
- •boolean isDirectory()
- •boolean isAbsolute()

Using class File



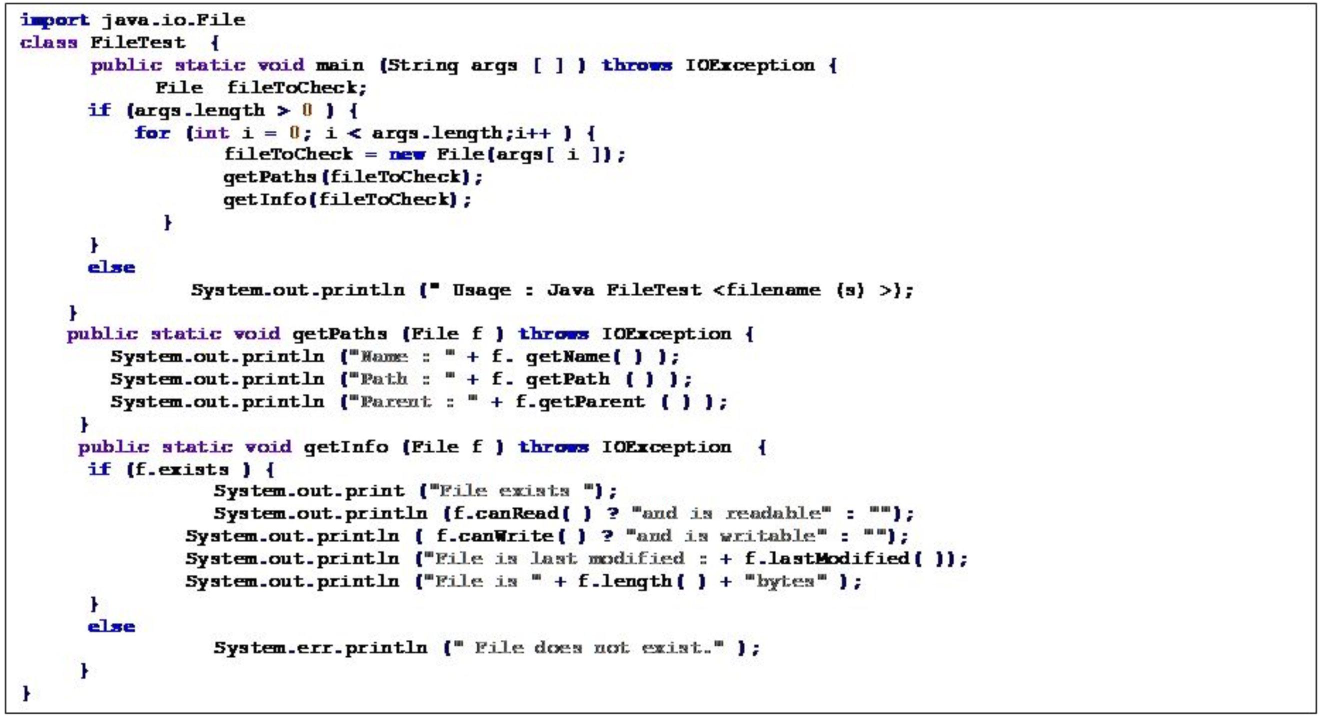
Getting file information

- long lastModified()
- long length()
- boolean delete()

Directory utilities

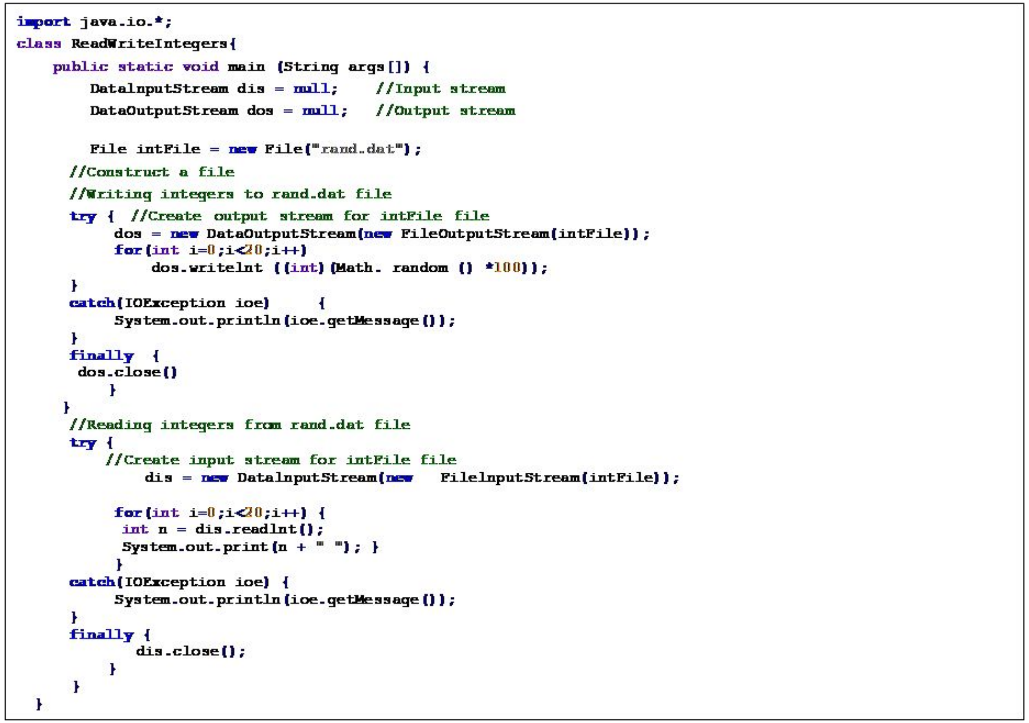
- boolean mkDir(File newDir)
- boolean mkDirs(File newDir)
- String [] list()

Knowing information about a File object



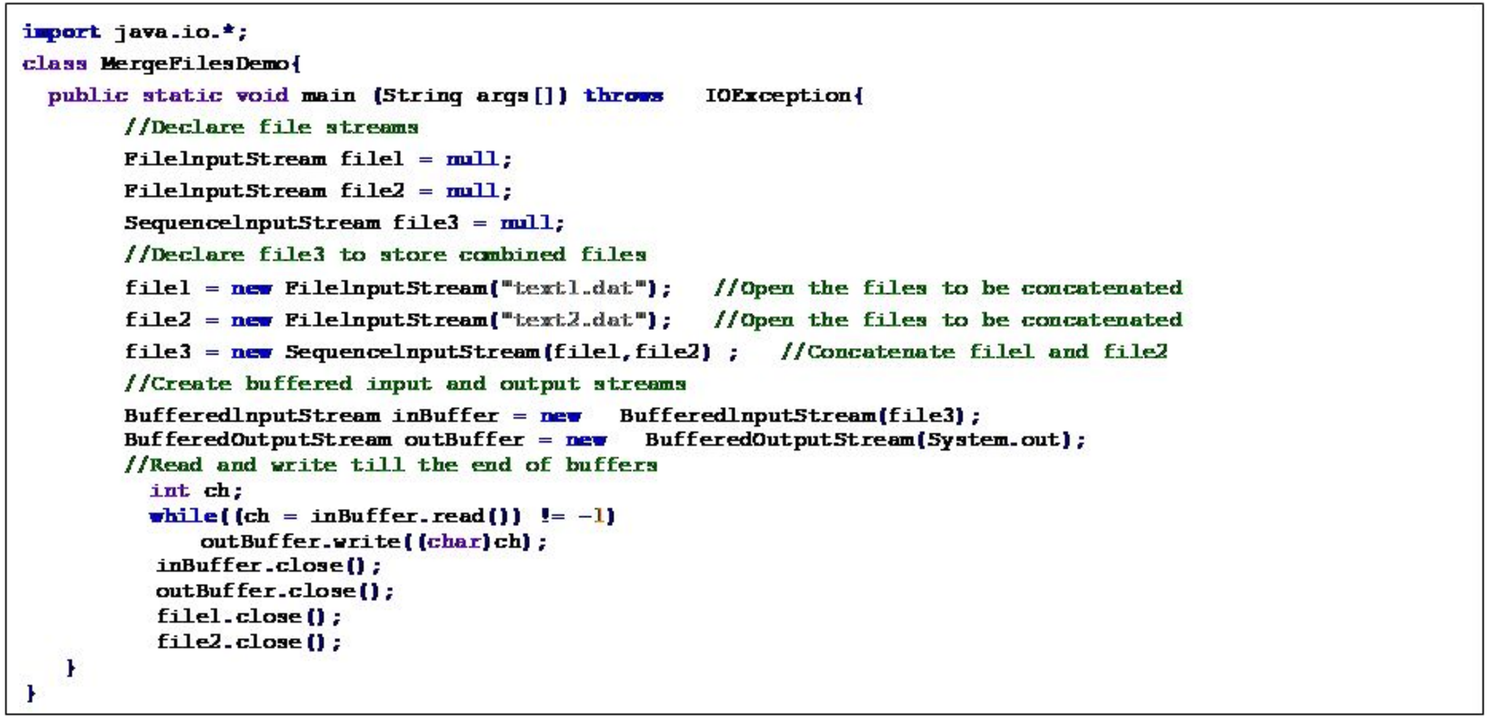
```
import java.io.File
class FileTest  {
      public static void main (String args [ ] ) throws IOException {
            File  fileToCheck;
      if (args.length > 0 ) {
          for (int i = 0; i < args.length;i++ ) {
                  fileToCheck = new File(args[ i ]);
                  getPaths (fileToCheck);
                  getInfo(fileToCheck);
             }
      }
      else
                System.out.println (" Usage : Java FileTest <filename (s) >);
    }
    public static void getPaths (File f ) throws IOException {
        System.out.println ("Name : " + f. getName( ) );
        System.out.println ("Path : " + f. getPath ( ) );
        System.out.println ("Parent : " + f.getParent ( ) );
     }
     public static void getInfo (File f ) throws IOException  {
      if (f.exists ) {
                 System.out.print ("File exists ");
                 System.out.println (f.canRead( ) ? "and is readable" : "");
               System.out.println ( f.canWrite( ) ? "and is writable" : "");
               System.out.println ("File is last modified : + f.lastModified( ));
               System.out.println ("File is " + f.length( ) + "bytes" );
      }
      else
                 System.err.println (" File does not exist." );
     }
}
```

Storing and reading data in the same file



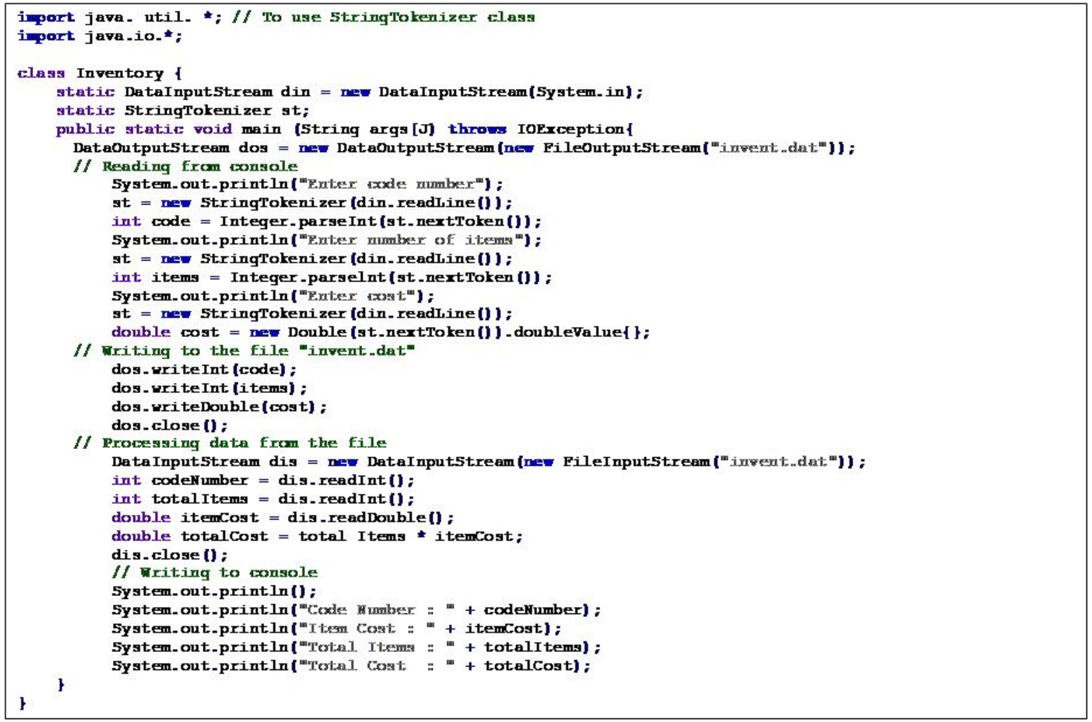
```
import java.io.*;
class ReadWriteIntegers{
    public static void main (String args[]) {
        DataInputStream dis = null;     //Input stream
        DataOutputStream dos = null;    //Output stream

        File intFile = new File("rand.dat");
      //Construct a file
      //Writing integers to rand.dat file
      try {  //Create output stream for intFile file
           dos = new DataOutputStream(new FileOutputStream(intFile));
           for(int i=0;i<20;i++)
                dos.writeInt ((int)(Math. random () *100));
      }
      catch(IOException ioe)      {
           System.out.println(ioe.getMessage());
      }
      finally  {
       dos.close()
          }
     }
      //Reading integers from rand.dat file
      try {
          //Create input stream for intFile file
               dis = new DataInputStream(new   FileInputStream(intFile));

           for(int i=0;i<20;i++) {
            int n = dis.readInt();
            System.out.print(n + " "); }
           }
      catch(IOException ioe) {
           System.out.println(ioe.getMessage());
      }
      finally {
              dis.close();
          }
      }
  }
```

Concatenation and buffering



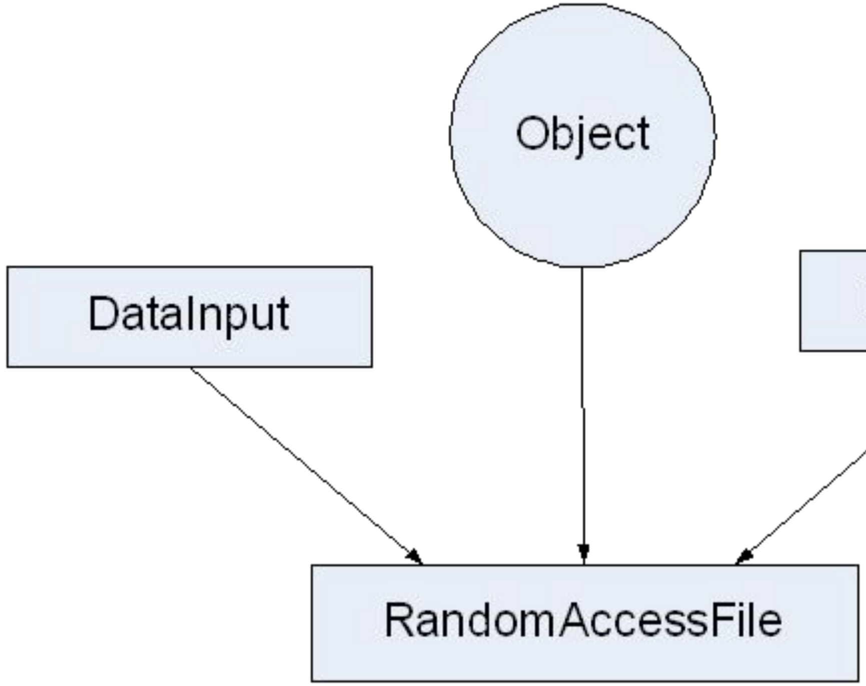
```
import java.io.*;
class MergeFilesDemo{
  public static void main (String args[]) throws   IOException{
        //Declare file streams
        FilelnputStream filel = null;
        FilelnputStream file2 = null;
        SequencelnputStream file3 = null;
        //Declare file3 to store combined files
        filel = new FilelnputStream("textl.dat");   //Open the files to be concatenated
        file2 = new FilelnputStream("text2.dat");   //Open the files to be concatenated
        file3 = new SequencelnputStream(filel,file2) ;   //Concatenate filel and file2
        //Create buffered input and output streams
        BufferedlnputStream inBuffer = new   BufferedlnputStream(file3);
        BufferedOutputStream outBuffer = new   BufferedOutputStream(System.out);
        //Read and write till the end of buffers
          int ch;
          while((ch = inBuffer.read()) != -1)
              outBuffer.write((char)ch);
          inBuffer.close();
          outBuffer.close();
          filel.close();
          file2.close();
   }
}
```

Interactive input and output



```
import java. util. *; // To use StringTokenizer class
import java.io.*;

class Inventory {
    static DataInputStream din = new DataInputStream(System.in);
    static StringTokenizer st;
    public static void main (String args[J) throws IOException{
      DataOutputStream dos = new DataOutputStream(new FileOutputStream("invent.dat"));
      // Reading from console
          System.out.println("Enter code number");
          st = new StringTokenizer(din.readLine());
          int code = Integer.parseInt(st.nextToken());
          System.out.println("Enter number of items");
          st = new StringTokenizer(din.readLine());
          int items = Integer.parselnt(st.nextToken());
          System.out.println("Enter cost");
          st = new StringTokenizer(din.readLine());
          double cost = new Double(st.nextToken()).doubleValue{};
      // Writing to the file "invent.dat"
          dos.writeInt(code);
          dos.writeInt(items);
          dos.writeDouble(cost);
          dos.close();
      // Processing data from the file
          DataInputStream dis = new DataInputStream(new FileInputStream("invent.dat"));
          int codeNumber = dis.readInt();
          int totalItems = dis.readInt();
          double itemCost = dis.readDouble();
          double totalCost = total Items * itemCost;
          dis.close();
          // Writing to console
          System.out.println();
          System.out.println("Code Number : " + codeNumber);
          System.out.println("Item Cost : " + itemCost);
          System.out.println("Total Items : " + totalItems);
          System.out.println("Total Cost  : " + totalCost);
    }
}
```

RANDOM ACCESS FILE



This class is used for reading and writing to random access file.

- A random access file behaves like a large array of bytes.
- There is a cursor implied to the array called file pointer, by moving the cursor we do the read write operations.
- If end-of-file is reached before the desired number of byte has been read than EOFException is thrown. It is a type of IOException.

Constructors of RandomAccessFile class

Constructor	**Description**
RandomAccessFile(File file, String mode)	Creates a random access file stream to read from, and optionally to write to, the file specified by the File argument.
RandomAccessFile(String name, String mode)	Creates a random access file stream to read from, and optionally to write to, a file with the specified name.

Methods of RandomAccessFile class

Method	Description
close()	It closes this random access file stream and releases any system resources associated with the stream.
getChannel()	It returns the unique FileChannel object associated with this file.
readInt()	It reads a signed 32-bit integer from this file.
readUTF()	It reads in a string from this file.
seek(long pos)	It sets the file-pointer offset, measured from the beginning of this file, at which the next read or write occurs.
writeDouble(double v)	It converts the double argument to a long using the doubleToLongBits method in class Double, and then writes that long value to the file as an eight-byte quantity, high byte first.
writeFloat(float v)	It converts the float argument to an int using the floatToIntBits method in class Float, and then writes that int value to the file as a four-byte quantity, high byte first.
write(int b)	It writes the specified byte to this file.
read()	It reads a byte of data from this file.
length()	It returns the length of this file.
seek(long pos)	It sets the file-pointer offset, measured from the beginning of this file, at which the next read or write occurs.

- As the name implies the class RandomAccessFile allows us to handle a file randomly in contrast to sequentially in InputStream or OutputStream classes.
- It allows to move file pointer randomly.
- Moreover, it allows read or write or read-write simultaneously.

READING FROM A RAF

Methods used to read RAF

Method	Description
read()	It reads a byte of data from this file.
length()	It returns the length of this file.
seek(long pos)	It sets the file-pointer offset, measured from the beginning of this file, at which the next read or write occurs.
close()	It closes this random access file stream and releases any system resources associated with the stream.
getChannel()	It returns the unique FileChannel object associated with this file.
readInt()	It reads a signed 32-bit integer from this file.
readUTF()	It reads in a string from this file.

Methods to write into RAF

Method	Description
write(int b)	It writes the specified byte to this file.
length()	It returns the length of this file.
seek(long pos)	It sets the file-pointer offset, measured from the beginning of this file, at which the next read or write occurs.
close()	It closes this random access file stream and releases any system resources associated with the stream.
writeDouble(double v)	It converts the double argument to a long using the doubleToLongBits method in class Double, and then writes that long value to the file as an eight-byte quantity, high byte first.
writeFloat(float v)	It converts the float argument to an int using the floatToIntBits method in class Float, and then writes that int value to the file as a four-byte quantity, high byte first.
write(int b)	It writes the specified byte to this file.

Java Packages

- A java package is a group of similar types of classes, interfaces and sub- packages.
- Package in java can be categorized in two form, built-in package and user-defined package.
- There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.
- Here, we will have the detailed learning of creating and using user-defined packages.
- Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

Advantage of Java Package

- Java package is used to categorize the classes and interfaces so that they can be easily maintained.
- Java package provides access protection.
- Java package removes naming collision.

Simple example of java package

The package keyword is used to create a package in java.

```
//save as Simple.java
package mypack;
public class Simple{
public static void main(String args[]){
System.out.println("Welcome to package");
}
}
```

How to compile java package

- If you are not using any IDE, you need to follow the syntax given below:
  - javac -d directory javafilename
- For example
  - javac -d . Simple.java
- The -d switch specifies the destination where to put the generated class file. You can use any directory name like /home (in case of Linux), d:/abc (in case of windows) etc. If you want to keep the package within the same directory, you can use . (dot).

How to access package from another package?

- There are three ways to access the package from outside the package.
  - import package.*;
  - import package.classname;
  - fully qualified name.

Using packagename.*

Using packagename.*

- If you use package.* then all the classes and interfaces of this package will be accessible but not subpackages.
- The import keyword is used to make the classes and interface of another package accessible to the current package.

```
//save by A.java
package pack;
public class A{
public void msg(){System.out.println("Hello");}
}
```

```
//save by B.java
package mypack;
import pack.*;

class B{
public static void main(String args[]){
A obj = new A();
obj.msg();
}
}
```

2) Using packagename.classname

2) Using packagename.classname

- If you import package.classname then only declared class of this package will be accessible.

```
//save by A.java

package pack;
public class A{
public void msg(){System.out.println("Hello");}
}
```

```
//save by B.java
package mypack;
import pack.A;

class B{
public static void main(String args[]){
A obj = new A();
obj.msg();
}
}
```

3) Using fully qualified name

If you use fully qualified name then only declared class of this package will be accessible. Now there is no need to import. But you need to use fully qualified name every time when you are accessing the class or interface.

It is generally used when two packages have same class name e.g. java.util and java.sql packages contain Date class.

```
//save by A.java
package pack;
public class A{
public void msg(){System.out.println("Hello");}
}

```

```
//save by B.java
package mypack;
class B{
public static void main(String args[]){
pack.A obj = new pack.A();//using fully qualified name
obj.msg();
}
}
```

Access Modifiers in java

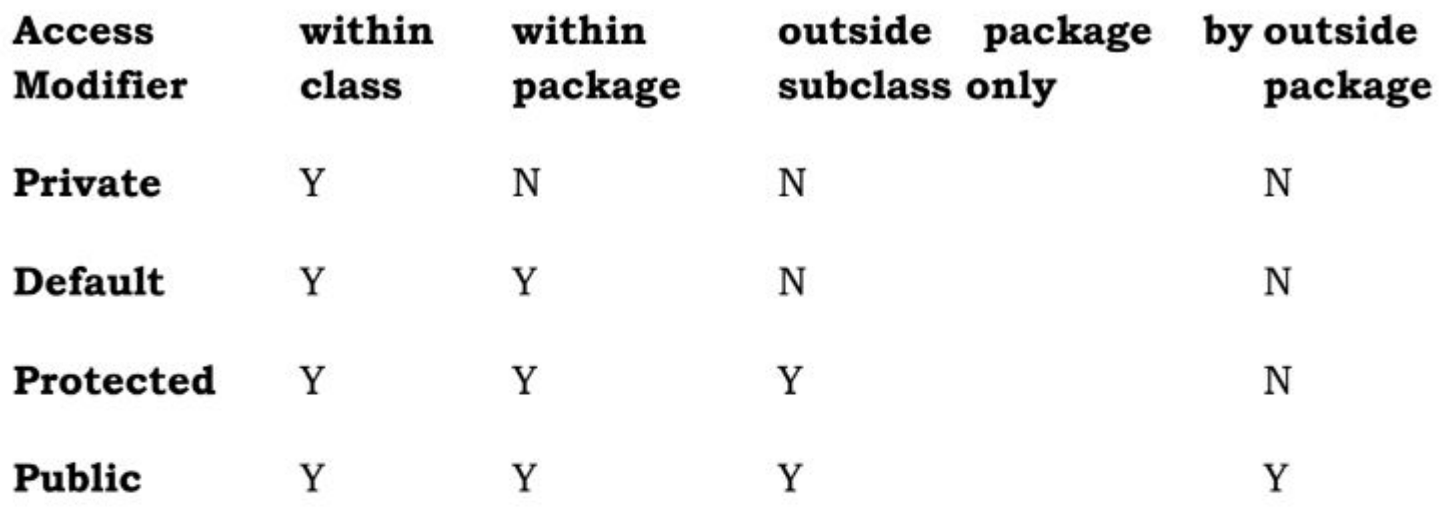
There are two types of modifiers in java: access modifiers and non-access modifiers.

The access modifiers in java specifies accessibility (scope) of a data member, method, constructor or class.

There are 4 types of java access modifiers:

1. Private
2. Default
3. Protected
4. public

Understanding all java access modifiers



Access Modifier	**within class**	**within package**	**outside package by subclass only**	**outside package**
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y