Ch – 3 Exception Handling, Threading & Streams

Concept of Exception Handling

- Actually, an exception is an object of an exception class. So when an exception arises, an object of that exception class is created and thrown. Then it is caught and processed.

- In Java, Exception Handling is done using five keywords : try, catch, throw, throws and finally.

- The general use of these keywords is explained here :

  - **Try:** Try is a block. The statements that can generate an exception are replaced in the try block.

Concept of Exception Handling

- **Catch :** If an exception is generated in the try block, then its object is created and thrown. This object is caught in the catch block and is handled.

- **Throw :** Normally, the exceptions are generated by some statement. But to manually create and throw an exception object, throw keyword is used.

- **Throws :** When a method can generate an exception, this method is specified by throws keyword.

- **Finally :** Finally is a block. The statements written in finally block are executed mandatorily whether the exception is caught or not.

Types of Exceptions

- ❑The super class of all Exception class is Throwable class. Error and exception are two subclass of throwable class.

- ❑The exception class has subclass that our program can catch. This class also used o create our own exception.

- ❑There are two subclass of Exception class : (1) RuntimeException and (2) IOException.

Use try and catch to handle exception

```
Class ExceptionEx{
  public static void main(String arg[]){
  int arr[]={23,43,45};
  try {
  System.out.println(“trying to access 26th
  element”);
  System.out.println(“arr[25]=”+arr[26]); // array
  index out of bound limits
  }
  catch(ArrayImdexOutOfBoundsException e){
  System.out.println(“Array index is out of
  bounds”);
} }
```

Nesting of try blocks and catch:

```
class nested_try_demo
{
    public static void main(String arsg[])
    {
        int a=0,b;
        try
        {
                a=10/args.length; // generate divide by zero if no
    command line argument is givenz
        System.out.println("A="+a);
        try
        {
                b=Integer.parseInt(args[1]);
                System.out.println("B="+b);
        a=12/0; // generate arithmetic Exception
        }
```

```
  catch(ArrayIndexOutOfBoundsException ae)
  {
      System.out.println("Array index out of limits");
  }
  }
  catch(ArithmeticException ae)
  {
      System.out.println("Divide by zero exception");
  }
    }
}
```

Throw keyword :

- We can throw an exception manually by throw keyword.
- Syntax : throw obj;
- Here the obj can be object of throwable class or any subclass of throwable class.
- Using throw keyword one can explicitly throw any exception.
- You can create object of any throwable class or subclass using new keyword or you can use the object specified in catch block.

Throw keyword :

1) **throw new ExceptionType();**
   - Here ExceptionType() is the constructor of the exceptionType class.
   - You can pass a String type argument describing the exception to this constructor.
   - This message is displayed when you print the exception or you can use getMessage() method that returns this description.
   - **Ex. throw new ArithmeticException("Divide by Zero");**

Throw keyword :

2) catch(ArithmeticException ae)

```
{
    throw ae;
}
```

- In first case the throw statement must be put in try block and caught in catch block.
- While in second case the exception is caught in catch block and it is thrown again.

Throw keyword :

- ❖ **Ex. WAP for exception handling using throw keyword.**

```
class throw_demo
{
    public static void main(String args[])
    {
            System.out.println(“Throwing
    exception explicitly”);
```

Throw keyword :

```
try
{
    throw new ArithmeticException(“Divide
    by zero”);
    }
    catch(ArithmeticException ae)
    {
      System.out.println(“Exception is:”+ae);
    }
}
```

Throw keyword :

WAP for exception handling to display that exception is caught again.

```
class throw_demo2
{
      static void Mtmethod()
      {
         throw new ClassCastException();
      }
      catch(ClassCastException cc)
      {
         System.out.println(“Exception is:”+cc);
         throw cc;
      }
}
```

Throw keyword :

```
    public static void main(String args[])
    {
            System.out.println(“Throwing Exception Excplicitly”);
    try
    {
            Mymethod();
    }
    catch(ClassCastException cc) // catching an exception
  thrown by Mymethod().
    {
            System.out.println(“Again caught:”+cc);
    }
  }
}
```

Throws keyword :

- Throws keyword is used to specify that a method can throw an exception that it does not handle.
- At a time, method definition you can do this by throws keyword.
- After throws keyword specify the exception class that it can throw.

Throws keyword :

- WAP for exception handling using throws keyword.

```
class test
{
    void testMethod() throws
    ClassNotFoundException
    {
      throw new ClassNotFoundException(“Class
    not found”);
    }
}
```

Throws keyword :

```
class throw_demo
{
        public static void main(String args[])
        {
                testclass t=new testclass();
                t.testMethod();
        }
        catch(ClassNotFoundException e)
        {
                System.out.println(“Exception :”+e);
        }
     }
}
```

Finally :

- Finally is a block which executes compulsorily. In some case, when an exception is occurred the program may omit some of the statements because its execution flow might be changed.

- Finally block is generally written after try-catch block and it is executed no matter an exception is caught by a catch block or not. The finally block is optional but a try block must have a catch or a finally block.

Creating our own exception class

- We create our own exception class by extending the Exception class which is the super class of exception class.

- The following example of Custom exception :

```
Import java.util.*;
Classs ownException extends Exception {
      boolean state;
      ownExcdeption(boolean state) {
      this.state=state; }
```

```
Public string toString() {
    return “ownException occurred with”+state;
}
Class customEx {
    static void test(boolean state) throwable
    ownException
    {
    if(state==false)
    throw.new ownException(state);
    else
    system.out.println(“no exception”);
}
Public static void main(String args[]) {
    boolean state;
    try {
      Random r=new Random();
      state=r.nextBoolean();
      test(state);
      }
      catch(ownException e)
      {
      system.out.println(”Exception”+e);}
      }
}
```

Basic introduction

- A thread is simply a light-weight process or you can say it is a sub process or child of a process.
- Multitasking can be achieved by two different ways: Process-based and Thread-based multitasking.
- In a process-based multitasking there are multiple programs running simultaneously. For example, you are running your program and at the same time you are listen to music. This is called process-based multitasking.

Basic introduction

- While a thread-based multitasking there are multiple threads running simultaneously.

- **For example,** in a web browser you are downloading a file from the server and at the same time you are also surfing the Internet. This is called thread-based multitasking.

Java Thread Model

- Java support multiple threads to run concurrently which is known as multithreading. This is far more advantageous than single thread system.

- In single threaded system a thread runs for indefinite time. While a thread is running, it may fore some event and wait for signal. So between two events, it stops for some time. This is called CPU cycle. Thus the time for these CPU cycle are wasted.

Java Thread Model

- But if we have multiple threads, then if a thread holds for some interval of time, other threads are running, so the wasted CPU cycle time can be eliminated.

Thread Life Cycle

- Every thread has a life cycle. As a thread is created, it may pass through the following states :
  - 1. Initial state
  - 2. Runnable state
  - 3. Running state
  - 4. Blocked state
  - 5. Dead state

Thread Life Cycle

- **1) Initial state :** When you create an object of thread, a thread is created and it is said to be in initial or newborn state.

- As this stage, a thread does nothing but it can go to either runnable state or dead state.

- **2) Runnable state :** Runnable state means the thread is ready to run but is not running. From the runnable state, a thread can go to running state.

Thread Life Cycle

- **3) Running state :** A thread is said to be in running state, when it is given the processor execute is process. At this stage, it runs its run() method and executes until release its control by yield() method.

- **4) Blocked state :** When a thread is not in the queue and is totally idle, it is said to be in blocked state. It can be sent into the blocked state by suspend(), wait() or sleep() method. A thread in the blocked state is not running but it is not dead because it can run again.

Thread Life Cycle

- **5) Dead state :** When a thread completes its execution i.e. its run() method, its life ends and is said to be dead. When a thread finishes its execution and it is called natural death.

Some basic thread methods

- ❖ **1] final String getName() :**
  - ❖ Returns the name of the thread.
- ❖ **2] final void setName() :**
  - ❖ Sets the name of the thread to the specified name.
- ❖ **3] static Thread currentThread() :**
  - ❖ Returns a reference to the invoking thread object.

Following Example shows these methods :

```
Class ThreadEx
{ public static void main(String args[])
   {
      Thread t=Thread.currentThread();
      System.out.println(“The thread is”+t);
      System.out.println(“Name of thread
        is”+t.getName());
      t.setName(“MyThread”);
      System.out.println(“Now the name of thread
        is”+t.hgetName());
   }
}
```

Creating Threads

- **There are two ways in java to create a thread :**
  - 1. By extending the Thread class.
  - 2. By implementing Runnable interface.

  - **[1] Extending Thread class :**

  - The Thread class is in the java.lang package which is the default package. The thread class has run() method. After extending the Thread class, you have to override the run() method.

Creating Threads

- **The Thread class has following constructors :**
  - Thread()
  - Thread(String name)
  - Thread(Runnable object)
  - Thread(Runnable object, String name)

Creating Threads

- **[2] Implementing Runnable Interface :**

- The second way of creating thread is implementing the Runnable interface. The runnable interface has only one method, which is the run() method. So to implement the Runnable interface you have to define the run() method.

- **Now to create a thread object, you have to use the following constructor :**
  - Thread(Runnable object, String name)

Multiple Threads

❖You can use more thread in single program then create multiple thread. Because java's strength also lies in multithreading. So in example we will create multiple threads :

```
Class MyThread implements Runnable
{ Thread t;
  String name;
  MyThread(String name)
  {this.name=name;
  t=new Thread(this.name);
  System.out.println("The thread created"+t);}
```

Multiple Threads

```
t.start(); }
Public void run()
{  for(int i=1;i<6;i++)
   { System.out.println(“Thread”+name”:”+i);
        try
        { Thread.sleep(400); }
        catch(InterruptedException e) {
        Syetem.out.println(“Thread”+name+”Comple
        ted”); } }
        System.out.println(“Thread”+name+”Cmplt”);
}  }
```

Multiple Threads

```
Class MultiThreadEx{
Public static void main(String args[]) {
   MyThread t=new MyThread(“First”);
   new MyThread(“Second”);
   try {
        Thread.sleep(2500);
        System.out.println(“main thread completed
        its ececution”); }
   catch(InterruptedException e) {
   System.out.println(“Main Thread intrrupted”);
}  } }
```

The isAlive() and join() methods

- Every time we can not know that after how many milliseconds the child threads will complete their execution. So java has two useful methods for it : isAlive() and join().

- The **isAlive()** method is used to know whether a thread is running or not. It returns true if the thread is still running else it returns false.

- **Its syntax is :**
  - Final boolean isAlive()

The isAlive() and join() methods

- The **join()** method is used to joins the thread which is running currently. Means it will not finish its execution until the thread on which it is called, completes its execution.

- **Its syntax is :**
  - Final void join() throws InterruptedException

Thread Priorities

- Each thread has its priority which is used for scheduling by the processor. These priorities its precedence over other threads.

- The thread priority can be obtained or set by the following methods :
  - Final void setPriority(int priority)

- The setPriority() method is used to set the priority of thread. The parameter priority is an int value from 1 to 10 . 1 is the lowest priority and 10 is the height priority.

Thread Priorities

- The getPriority() method returns the priority of the invoking thread object.
  - **Final void getPriority()**

  - The thread class has following int constants for thread priorities. These variable are static and final.

    - **NORM_PRIORITY :** The default priority is 5
    - **MIN_PRIORITY :** The minimum priority which is 1.
    - **MAX_PRIORITY :** The maximum priority which is 10.

Synchronization

- When more than one threads attempt to use a thread resource, a phenomenon occurs which is known as race condition. This race condition must be avoided because when two or more threads access a shared resources such as a file. This may lead to unexpected results. Therefore the synchronization is necessary.

- The synchronization ensures that only one thread will access a shared resource and after the completion of its execution the resource will be unlocked.The synchronization is done by the synchronized keyword.

Deadlock

- When two threads have circular dependency on objects a major problem arises. This type of error is known as ‘deadlock’.

- In deadlock, one thread is waiting for other thread and the other thread is waiting for first thread.

- Thus, both threads go in waiting(blocked) state and can not continue their execution.

Basic Introduction

- This chapter introduces the java.io package and its various classes and methods. These classes are used to handle the input and output related tasks. First of all we will see the file class and then we will discuss about streams.

1. The file class

- ❖The file class is used to get the information of a file, such as the length of file, its permissions, directory path etc. you can also create a directory, delete a file and directory using the file class.
- ❖The file class has many useful methods and constants that can be used to get such kind of information about a file.
- ❖**The file class has following constructors:**
  - ❖1) File(String path)

1. The file class

- File(String directoryPath, String filename)
- File(File obj, String filename)

- The first form has one parameter that is the path to a file or directory.

- The second form has two parameters. These are the path to a directory and the name of a file in that directory.

- The last form also has two parameters. These are a File object for a directory and the name of a file in that directory.

1. The file class

- The methods defined by the File class are given below :

- **1) String getName() :**
  - Returns the name of the file.

- **2) String getParent() :**
  - Returns the name of the parent directory of the file.

- **3) String getPath() :**
  - Returns the path of the file.

1. The file class

- ❖**4) Boolean canRead() :**
  - ❖Returns true if the file exists and can be read. Otherwise, returns false.

- ❖**5) Boolean canWrite() :**
  - ❖Returns true if the file exists and can be written. Otherwise, returns false.

- ❖**6) Boolean delete() :**
  - ❖Deletes the file. Returns true if the file is successfully deleted. Otherwise, returns false.

1. The file class

- **The file class has following two constants :**
  - **(1) sepratorChar(\) :** It is the seprator character that seprates the directory names.
  - **(2) pathSepratorChar(;) :** It is the seprator character that septrates the paths.

2. Streams

- ❖A stream is a part or a medium along which the data passes through streams from source to destination.

- ❖The source is called the input and the destination is called the output of the program.

- ❖The example of input stream can be a keyboard, a mouse, a file, a memory buffer etc. and the output can be a monitor screen, a pointer, a file etc.

2. Streams

- **There are two types of streams:**
  - (1) character Streams
  - (2) byte Streams

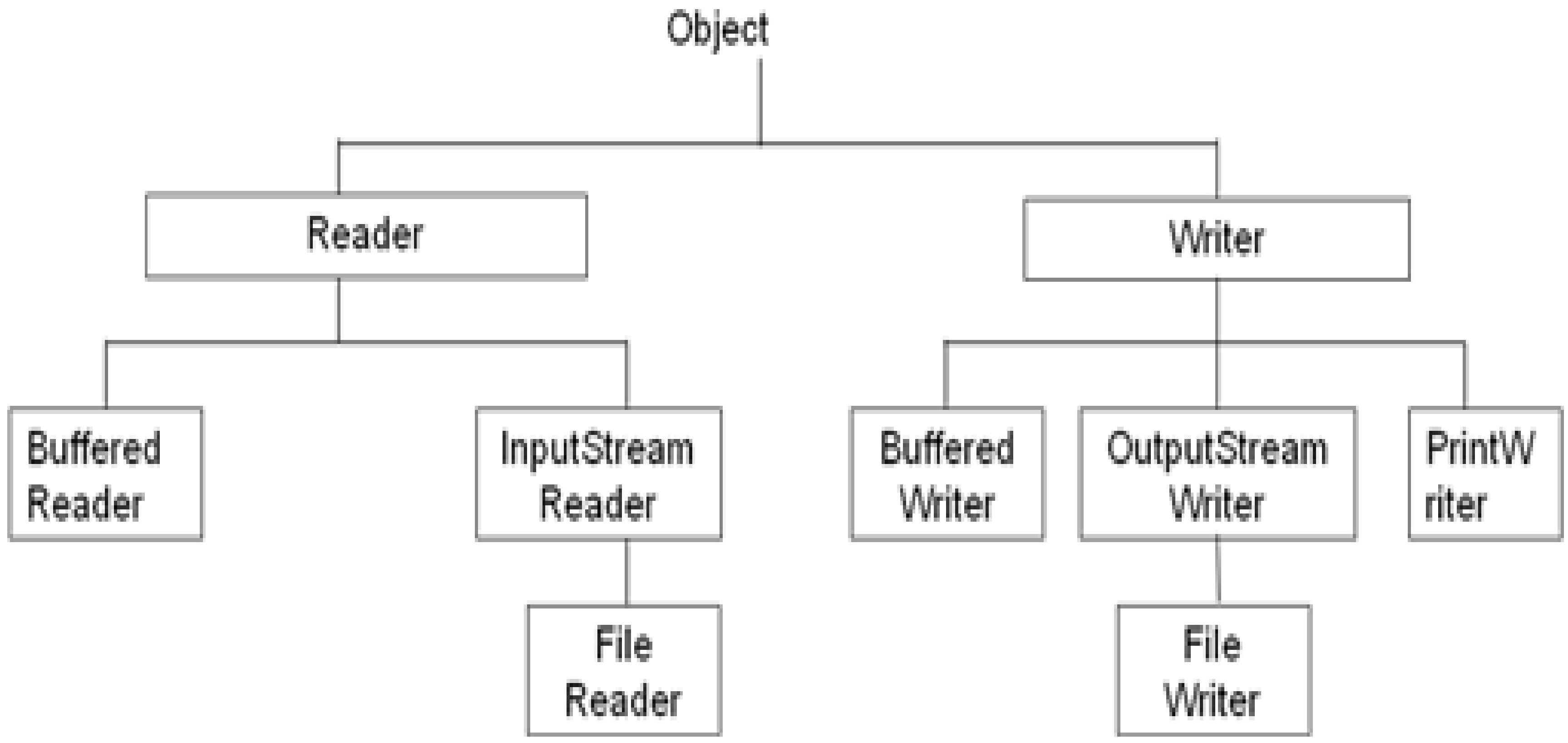
  - **(1) Character Streams :**
    - The character stream classes manipulate data as character. A character in java is of 16-bits. Thus the character stream classes can work with 16-bit Unicode characters. The main two classes of character stream are Reader and Writer.

2. Streams

- **(2) Byte Streams :**

  - The Byte stream classes manipulate data byte by byte. A byte in Java is 8-bits. Thus a byte stream object reads or writes data value in binary form. The main two classes of byte stream are InputStream and OutputStream.

1. **Character Streams Classes:**
   The character stream classes are Reader and Writer. The classes of character streams are classified as follow:



2.1 Reader

- **The reader class is an abstract class so we can not create its object. Its subclass are used to read characters from a file. The methods of Reader class are as follow :**

  - **1) Void close() :**Closes the input stream. Further read attempts generate an IOException. Must be implemented by a subclass.

  - **2) Int read() :** Reads a character from the stream. Waits until data is available.

- ❖**3) Boolean ready():** Returns true if the next read() will not wait.

- ❖**4) Void reset() :** Resets the input pointer to the previously set mark.

- ❖**5) Void mark(int numChars) :**Places a mark at the current point in the input stream that will remain valid until numChars characters are read.

- ❖**6) Int skip(long numChars) :** Skips over numChars bytes of input returning the number of characters actually skipped.

2.1.1 InputStreamReader Class

- This class is a subclass of the Reader class. It reads a byte from the input stream i.e. A file and converts it to the character. Its constructor are :
  - [1] InputStreamReader(InputSytream obj):
    - Here, obje is the object of the InputStream which is the input stream class of byte stream.
    - The InputStream is an abstract class, so we have to use any concrete subclass object as parameter.

2.1.2 FileReader

- ❖The file reader class is a subclass of InputStreamReader class and inputs characters from a file.

- ❖Its most common constructor are :

  - ❖FileReader(string filepath)
  - ❖FileReader(File object)

2.2 Writer

- **The writer streams are used to perform all output operations on files. The writer class is an abstract class which acts as a base class for all the other writer stream classes.**

- **Its constructor are :**
  - Writer()
  - Writer(Object obj)

- **Its Methods are :**
  - **1) Void close() :** Closes the output stream.

- ❖ **2) Void flush() :** Writes any buffered data to the physical device represented by that stream.

- ❖ **3) Void write(int c) :** Writes the lower 16 bits of c to the stream.

- ❖ **4) Void write(char buffer[]) :**Writes the characters in buffer to the stream.

- ❖ **5) Void write(char buffer[], int index, int size) :** Writes size characters form buffer starting at position index to the stream.

2.2.1 OutputStreamWriter class

- ❖**This class is a subclass of Writer class.**
- ❖**It converts a stream of characters to a stream of bytes.**
- ❖**Its constructors are like this:**
  - ❖**OutputStreamWriter(OutputStream os)**
    - ❖Here, os is the output stream and encoding is the name of a character encoding.

2.2.2 FileWriter class

- ❖The FileWriter class is used to write characters in a file.

- ❖**Its constructor are :**
  - ❖**1) FileWriter(String filepath) :** The path specifies the path of the file to   be writen.
  - ❖**2) FileWriter(File obj) : The obj is object of file class.**
  - ❖**3) FileWriter(String filepath, Boolean append) :** If the second constructor append is true, the file will be opened in append mode.

2.2.3 BufferWriter class

- ❖The BufferWriter class is a subclass of Writer. The BufferWriter class extends Writer and buffers output to a character stream.

- ❖**Its constructors are as follows:**

  - ❖**1) BufferedWriter(Writer w) :** The w is object of a subclass of Writer class.

  - ❖**2) BufferedWriter(Writer w, int bufSize) :** here you specify the size.

2.2.4 PrintWriter class

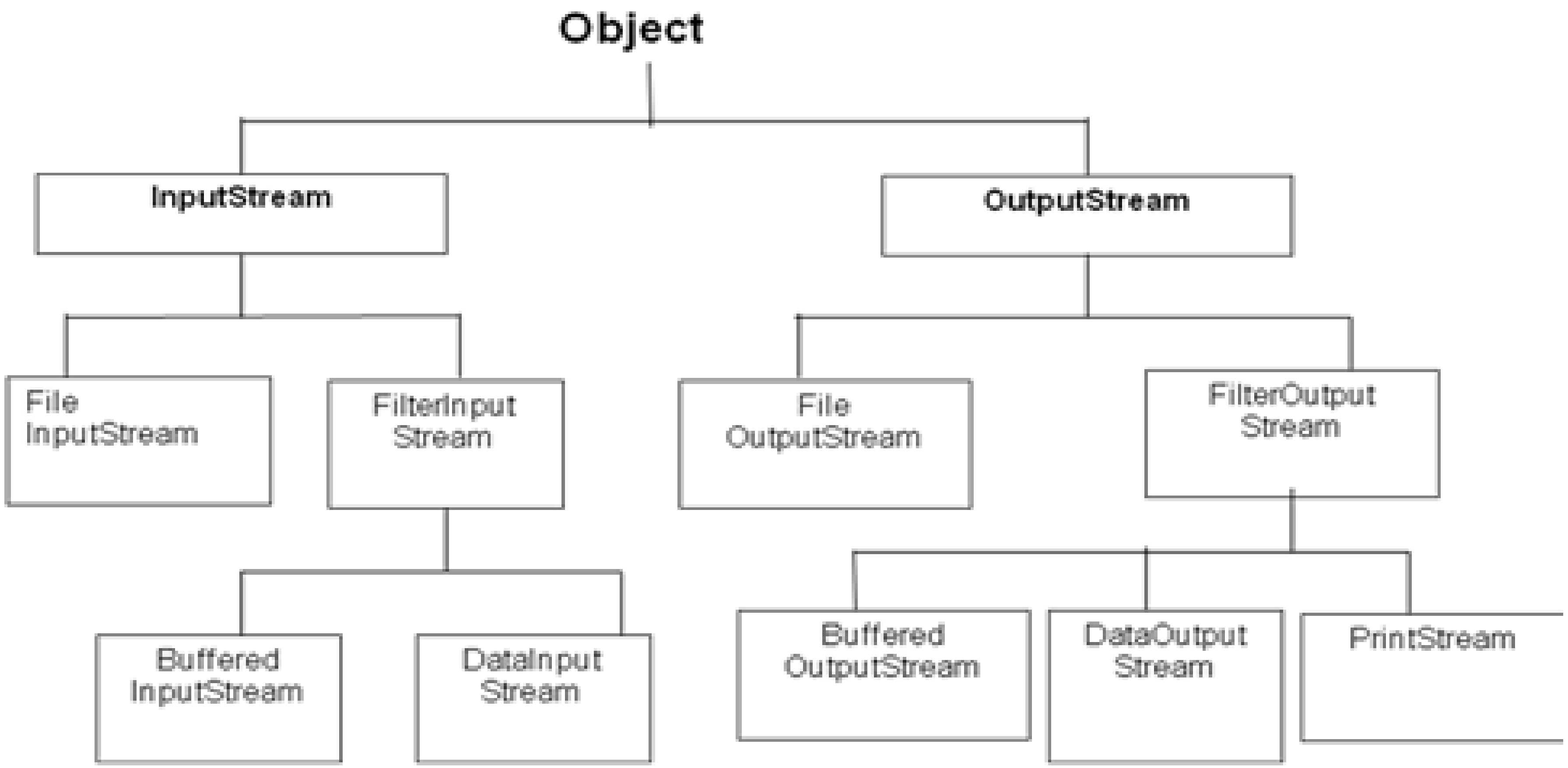
- ❖The printWriter class is used to display values of simple data type. It has method to print data such as print() and println(). In these methods java calls toString() method to convert data type to a simple data type.

- ❖ **Its constructor are :**
  - ❖**1) PrintWriter(Writer obj) :** The obj is an object of Writer class.

  - ❖**2) PrintWriter(OutputStream obj) :**The obj is an object of OutputStream’s subclass.

3 Byte Stream class

- The byte stream class are InputStream and OutStream. The classes of byte streams are classified as follows:



3.1 InputStream

- ❖The InputStream is an abstract class of byte stream.
- ❖**The methods provided by the InputStream class are:**
  - ❖**1) Int available() :** Returns the number of bytes currently available for reading.
  - ❖**2) Void close() :** Closes the input stream.
  - ❖**3) Void mark(int numBytes) :**Places a mark at the current point in the input stream. It remains valid until numBytes are read.

3.1 InputStream

- ❖**4) Int read() :** Reads one byte form the input stream.

- ❖**5) Void reset() :** Resets the input pointer to the previously set mark.

- ❖**6) Int skip(long numBytes) :** Skips numBytes of input. Returns the number of bytes actually skipped.

3.1.1 FileInputStream

- ❖The FileInputStream class is used to create an InputStream obj file that is used to read byte from a file.

- ❖**Its constructor are :**

  - ❖**1) FileInputStream(String filepath) :** The filepath is path of file to be read.

  - ❖**2) FileInputStream(File fileObj) :** The fileObj is object of file class.

3.1.2 FilterInputStream

- ❖The FilterInputStream is subclass of InputStream that offers some more functionality than other InputStream class.

- ❖**Its constructor are :**

    - ❖**FilterInputStream(InputStream obj) :** The obj is the object of InputStream subclass.

3.1.3 BufferedInputStream

- ❖The BufferedInputStream class is a subclass of FilterdInputStream and is used to read bytes from input stream using buffer.

- ❖**Its constructor are :**
  - ❖**1) BufferedInputStrem(InputStream obj) :** The obj is the object of InputStream subclass. The default buffer size is used.

  - ❖**2) BufferedInputStream(InputStream obj, int bufferSize) :** The buffer size can be specified by buffersize.

3.1.4 DataInputStream

- The DataInputStream class is a subclass of FilterInputStream class. This class allow reading of Java’s standard data type value.

- **Its constructor are :**
  - **1) DataInputStream(InputStream obj) :** Here obj is the object of an InputStream subclass.

- **Its methods are :**
  - **1) boolean readBoolean() :** It reads a boolean object of an InputStream subclass.

- ❖**2) byte readByte() :** It reads byte value from the input and it returns it.
- ❖**3) char readChar()**
- ❖**4) int readInt()**
- ❖**5) long readLong()**
- ❖**6) Short readShort()**
- ❖**7) float readFloat()**
- ❖**8) double readDouble()**

- ❖**9) String readUTF() :**It reads a string value from the input stream and returns it. The character are converted from UTF-8 to Unicode (16-bit) format.

3.1.5 ObjectInputStream

- ❖The ObjectInputStream is a subclass of InputStream and it implement the objectInterface interface. This class is used to read object from the input stream.

- ❖**Its constructor are :**
  - ❖**1) ObjectInputStream(InputStream obj) :** Here obj is the object of an InputStream.

- ❖**Its methods are :**
  - ❖**1)final Object readObject() :** This method is read object from the input stream. This method is final, so can’t be overriden.

4. ObjectOutputStream

- ❖The ObjectOutputStream class is discussed here to maintain continuty. It is a subclass of OutputStream and it implements the ObjectOutput interface.

- ❖**Its constructor are:**
  - ❖**ObjectOutputStream(OutputStream obj) :** Here, the obj is an object of any OutputStream.

- ❖**Its method are:**
  - ❖**Void writeObject(Object obj) :** It writes the object obj to the output stream.

4.1 OutputStream

- The OutputStream class is an abstract class that is used to output bytes in an output stream.

  - **The Methods of OutputStream class are listed below :**
    - **1) void write(int byte) :** It writes a single byte in the output stream.
    - **2) void write(byte buffer[]) :** It writes the contents of byte array buffer to the output stream.
    - 3) void close()
    - 4) void flush()

4.1.1 FileOutputStream

- ❖The FileOutputStream class is a subclass of the OutputStream class. Its used to write byte to a file.
  - ❖**The constructor are :**
    - ❖1) FileOutputStream(File obj)
    - ❖2) FileOutputStream(String path)
    - ❖3) FileOutputStream(String path,boolean append)

4.1.2 FilterOutputStream

- ❖The FilterOutputStream class is a subclass of OutputStream class. It offers some extended level of functionality with compared to the other output stream class. It is an abstract class and its three subclass are BuffedOutputStream, DataOutputStream and PrintStream classees.

  - ❖**The constructor are :**

    - ❖**1) FilterOutputStream(OutputStream obj) :** The obj is an object of an output stream class.

4.1.3 BufferedOutputStream

- ❖The BufferedOutputStream class is a subclass of FilterOutputStream and is used to output byte to a file using buffer.

  - ❖**The constructor are :**
    - ❖**1) BufferedOutputStream(OutputStream obj) :** The obj is an object of an subclass of OutputStream. The default buffer size is used.

    - ❖**2) BufferedOutputStream(OutputStream obj, int bufferSize) :** The size of buffer is specifed by the bufferSize parameter.

4.2.4 DataOutputStream

- The DataOutputStream class is a subclass of FilterOutputStream. It is used to write Java’s standard data type values. It implements DataOutput interface.

  - **The constructor are :**

    - **1) DataOutputStream (OutputStream obj) :** The obj is an object of a subclass of OutputStream class.

- **The methods are :**
  - 1) void write(int i)
  - 2) void writeChars(String str)
  - 3) void writeDouble(double d)
  - 4) void writeFloat(float f)
  - 5) void writeBoolean(boolean b)
  - 6) void writeShort(short s)
  - 7) void writeLong(long l)

4.2.5 PrintStream

- ❖ The PrintStream class is a subclass of FilterOutputStream class. It is used to print data as we do using print() and println() methods of System.out. This class also supports these methods.

  - ❖ **The constructor are :**
    - ❖ **1) PrintStream(OutputStream obj) :** The obj is an object of any OutputStream class.
    - ❖ **2) PrintStream(OutputStream obj, boolea flushOnNewLine) :** If the flushOnNewLine is true,the output stream is automatically gets flushed when a new line character.

5. Other java.io calsses

- **RandomAccessFile :**
  - The RandomAccessFile class is used to read or write data from a file randomaly. This class is not a subclass of InputStream or OutputStream but it is directly the subclass of Object class. This class implements DataInput and DataOutput interfaces.
  - Other java.io classes have sequential access to a file. But the RandomAccessFile can read or write from to any location of a file.

- **The constructor are :**
  - 1) RandomAccessFile(File obj,String mode)
  - 2) RandomAccessFile(String filenm,String mode)
- **The methods are :**
  - **1) void seek(long numOfBytes) :** It positions the file pointer after the numOfBytes bytes.
  - **2) int read() :** It reads and returns a byte from the input stream.

- ❖**3) long length() :** It reaturns the length of the file in byte.

- ❖**4) long setLength() :** It sets the length of file.

- ❖**5) int skipBytes(int numOfByte) :** It ignores the numOfBytes bytes. It numOfBytes is less or qual to zero, no bytes are skipped.

- ❖**6) void close() :** It close the file.

6. StreamTokenizer class

- ❖The StreamTokenizer class is used to separate token from an input stream. A token may be a word, a character or a number.

- ❖This can be used to make a compilers or parser or a similar program that needs to process token.

- ❖**A constructor for this class is as follows:**

  - ❖**StreamTokenizer(Reader obj) :** The obj is an object of Reader's subclass.

- **There are some constants of StreamTokenizer**
  - **1) ttype:** It is used to check token type. i.e. A word, a number etc.
  - **2) TT_EOF :** It inndicates end-of-file.
  - **3) TT_EOL :** It indicated end-of-line.
  - **4) TT_NUMBER :** If the token is a number, ttype equals TT_NUMBER.
  - **5) TT_WORD :** IF the token is a word, ttype equal to TT_WORD.

- ❖**There are some methods of StreamTokenizer :**
  - ❖**1) void eollsSignificant(boolean flag):** If the flag is true, the end-of-line is considered as a tken, else is cinsidered as a white space.
  - ❖**2) int lineno() :** It returns the current line number.
  - ❖**3) int nextToken() :** It returns the type of the next token.
  - ❖**4) String toString():** It Converts and the token into string and returns it.

- ❖**5) void resetSyntax():** If specifies that all characters should be considered as normal caharacter.

- ❖**6) void ondinaryChar(int ch):** If specifies that ch should be treated as normal caharacter.

- ❖**7) void wordChars(int start, int end) :** The characters whose ASCII values come between the range start and end are treated as word characters.