OBJECT ORIENTED TECHNIQUES USING JAVA(ACSE0302)

Unit: 4

Concurrency in Java and I/O Stream

Course Details
(B.Tech 3rd Sem /2nd Year)

Dr. Mohd Shahid
Department CSE

NIET Greater Noida
GET FUTURE READY
Estd.2001
AN AUTONOMOUS INSTITUTE

UNIT CONTENT

1. Threads: Introduction and Types
   - Creating Threads
   - Thread Life-Cycle
   - Thread Priorities
   - Daemon Thread
   - Runnable Class
   - Synchronizing Threads
2. I/O Stream: Introduction and Types
   - Common I/O Stream Operations
   - Interaction with I/O Streams Classes
3. Annotations: Introduction
   - Custom Annotations
   - Applying Annotations

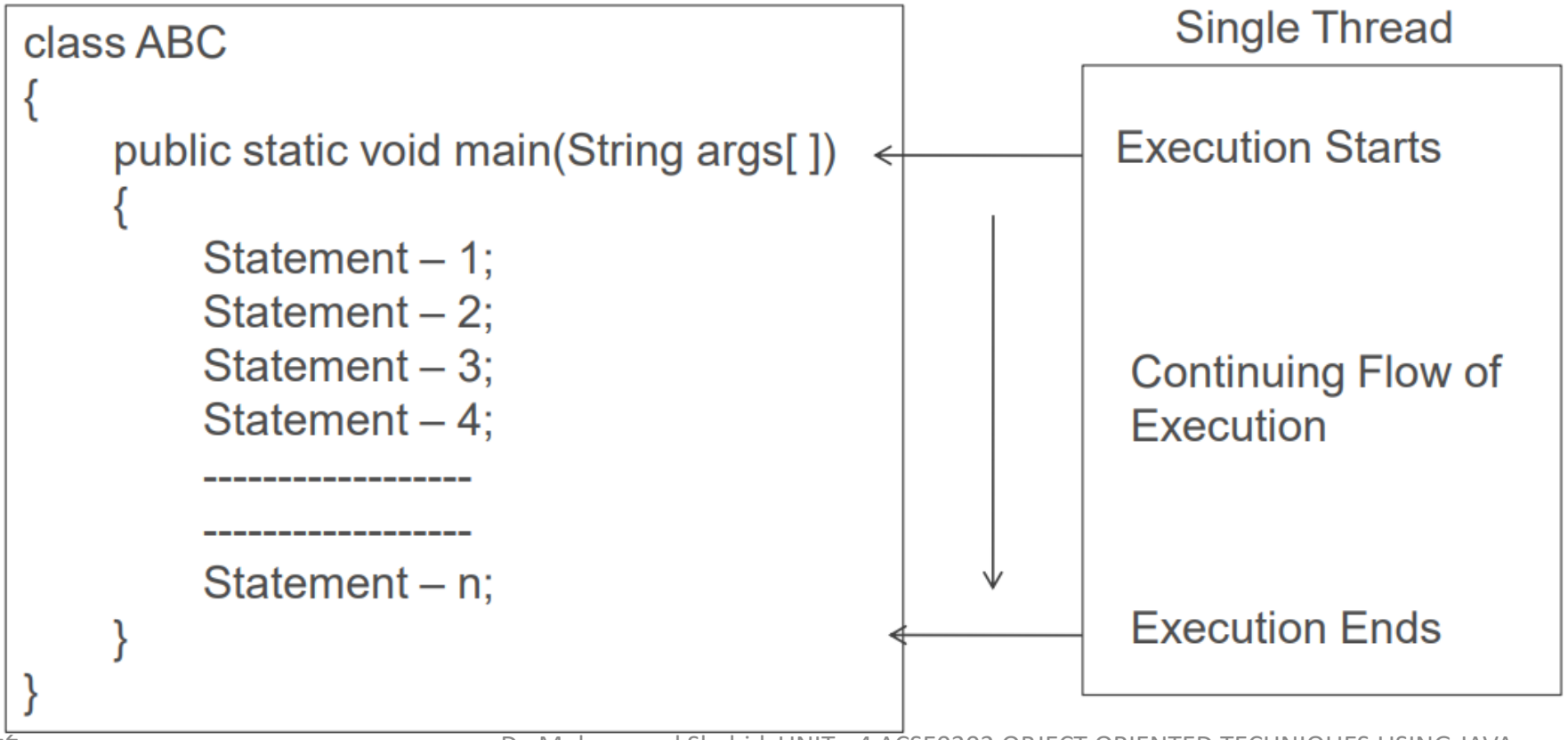
Threads: Introduction

- Threads
  - Threads allows a program to operate more efficiently by doing multiple things at the same time.
  - Threads can be used to perform complicated tasks in the background without interrupting the main program.
  - Java is a multi-threaded programming language which means we can develop multi-threaded program using Java.
  - A multi-threaded program contains two or more parts that can run concurrently, and each part can handle a different task at the same time making optimal use of the available resources specially when your computer has multiple CPUs.
  - Multitasking is when multiple processes share common processing resources such as a CPU.

Threads: Introduction

Threads:

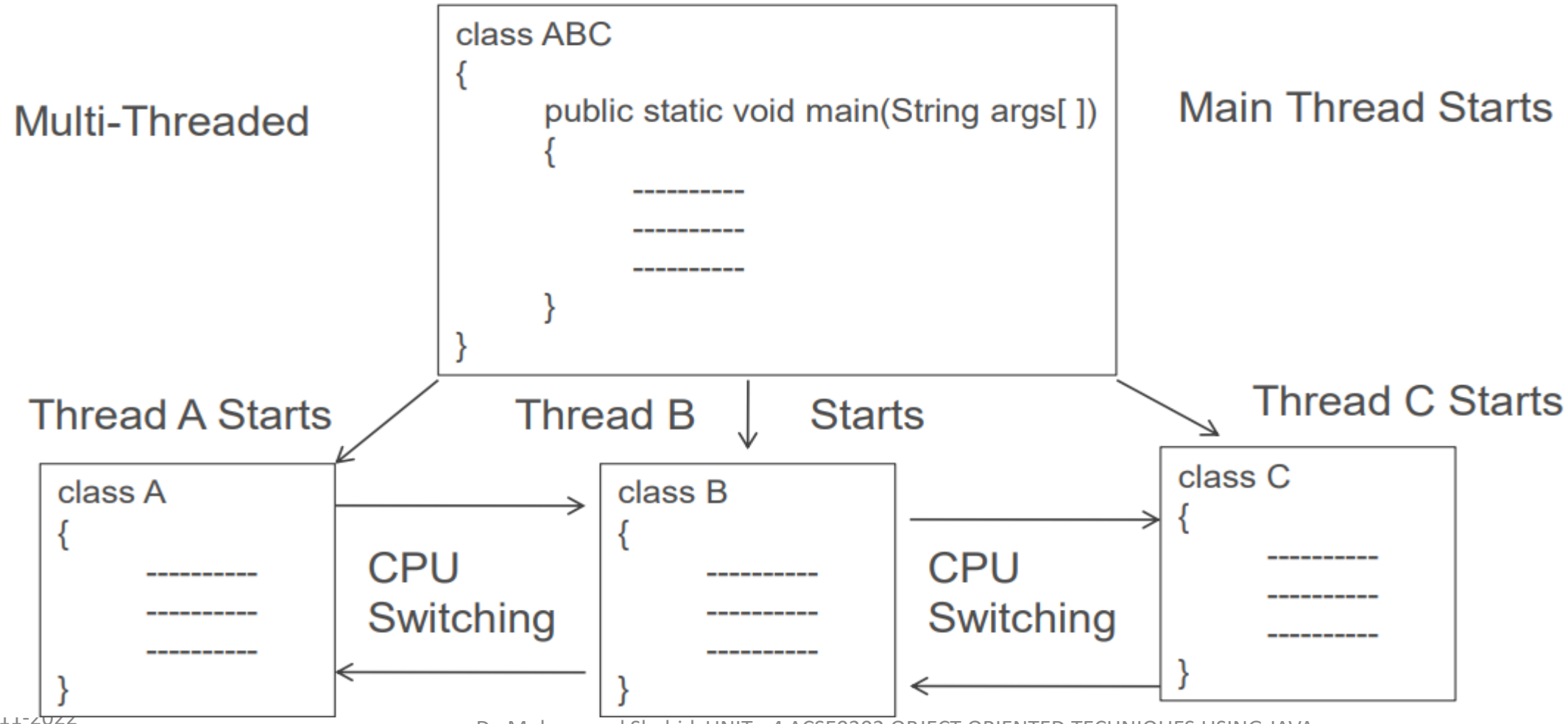
- It is similar to a program that has single flow of control.
- It has a beginning, a body and an end and execute commands sequentially.
- All main program can be called single-threaded programs.



Threads: Introduction

Multi-Threading:

- A program that contains multiple flow of control called multi threaded program.



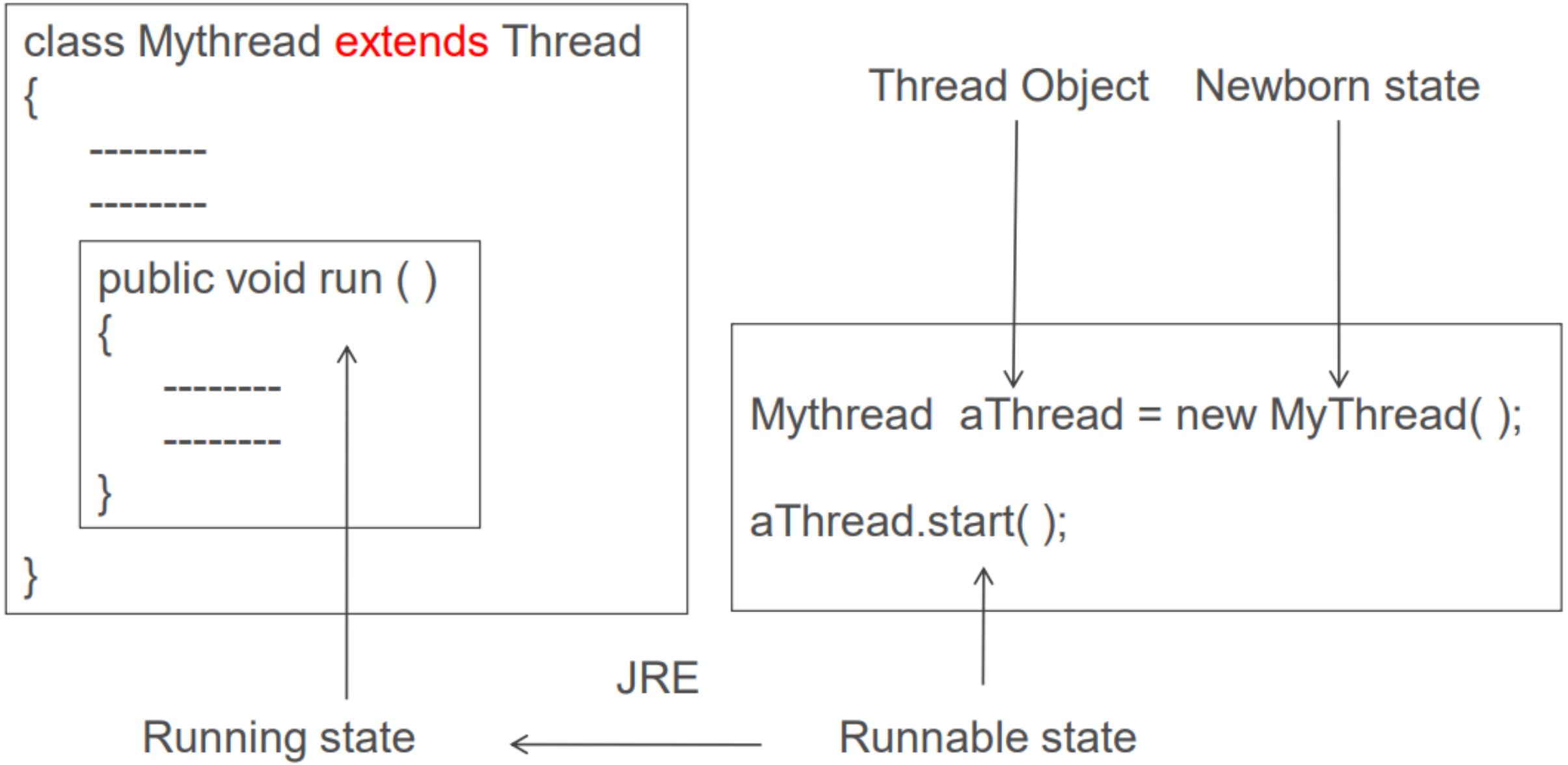
Creating a Threads

- Threads are implemented in form of a object containing a method called **run().**
- **run**() makes up entire body of thread and only a method that thread's behavior can be implemented. It is invoked by the object of concerned thread.
- Thread is initiated using start( ) method
- A new thread can be implemented by two ways:

1. **By creating a thread class**: by extends **Thread** class and override its **run**() method.
2. **By converting a class to a thread:** by implements **Runnable** interface.

```
public void run( )
{
    Statements for implementing Thread
}
```

Extending Thread Class - overriding the run( )



1. By creating a thread class

```
class A extends Thread
{
    public void run ( )
    {
      for ( int i = 1; i <= 5; i++ )
      {
          System.out.println("From Thread A : i = " + i);
      }
      System.out.println("Exit From Thread A");
    }
}
```

```
class B extends Thread
{
    public void run ( )
    {
      for ( int j = 1; j <= 5; j++ )
      {
        System.out.println("From Thread B : j = " + j);
      }
      System.out.println("Exit From Thread B");
    }
}
```

1. By creating a thread class

```
class C extends Thread
{
    public void run ( )
    {
      for ( int k = 1; k <= 5; k++ )
      {
        System.out.println("From Thread C : k =  " + k);
      }
      System.out.println("Exit From Thread C");
    }
}
```

```
class ThreadTest
{
    public static void main(String args[ ])
    {
            new A( ).start( );
            new B( ).start( );
            new C( ).start( );
    }
}
```

1. By creating a thread class(Output)

```
H:\SPS\Thread>java ThreadTest
From Thread A : i = 1
From Thread C : k = 1
From Thread B : j = 1
From Thread C : k = 2
From Thread A : i = 2
From Thread C : k = 3
From Thread B : j = 2
From Thread C : k = 4
From Thread C : k = 5
Exit From Thread C
From Thread A : i = 3
From Thread B : j = 3
From Thread B : j = 4
From Thread B : j = 5
Exit From Thread B
From Thread A : i = 4
From Thread A : i = 5
Exit From Thread A
```

Run 1

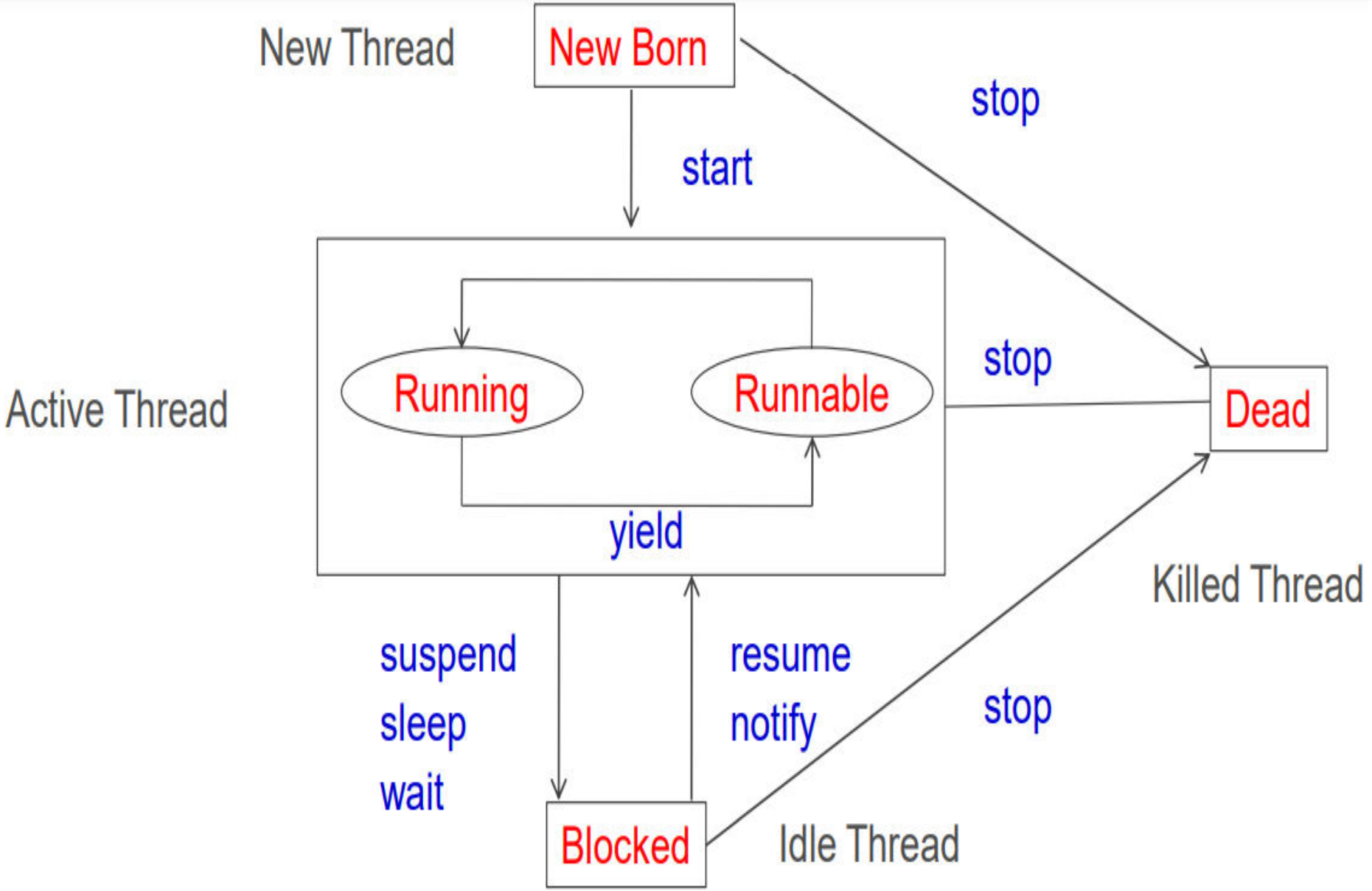
```
H:\SPS\Thread\1>java ThreadTest
From Thread A : i = 1
From Thread B : j = 1
From Thread C : k = 1
From Thread B : j = 2
From Thread A : i = 2
From Thread B : j = 3
From Thread B : j = 4
From Thread C : k = 2
From Thread B : j = 5
From Thread A : i = 3
Exit From Thread B
From Thread C : k = 3
From Thread A : i = 4
From Thread C : k = 4
From Thread A : i = 5
From Thread C : k = 5
Exit From Thread C
Exit From Thread A
```

Run 2

A Thread Life Cycle

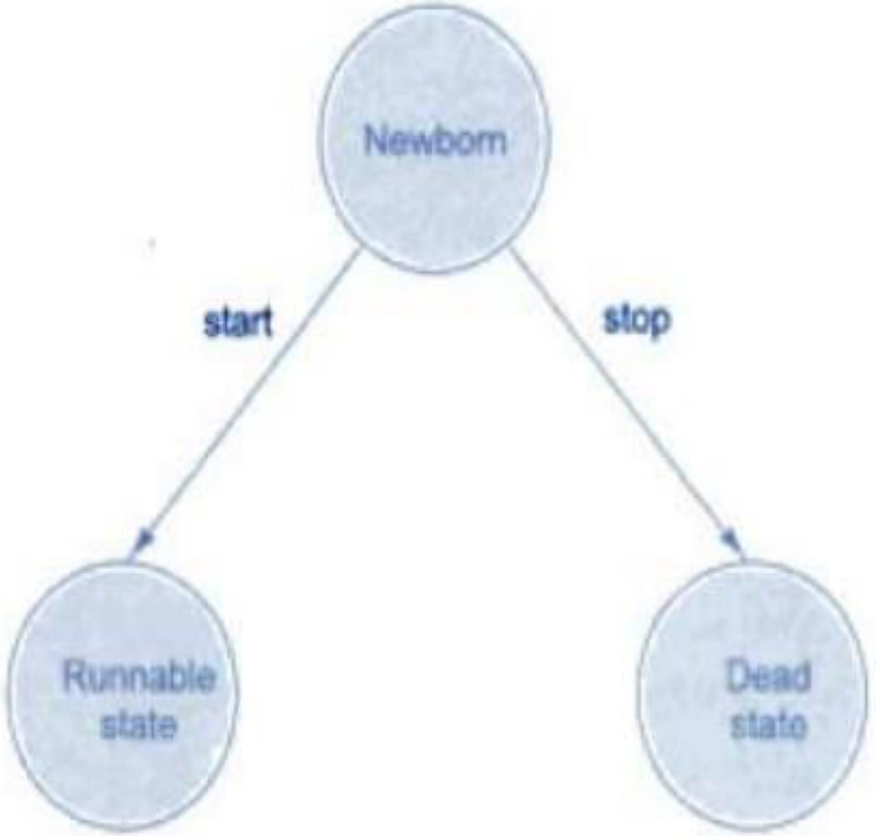
java thread states

1. New
2. Runnable
3. Running
4. Blocked
5. Terminated



Thread states

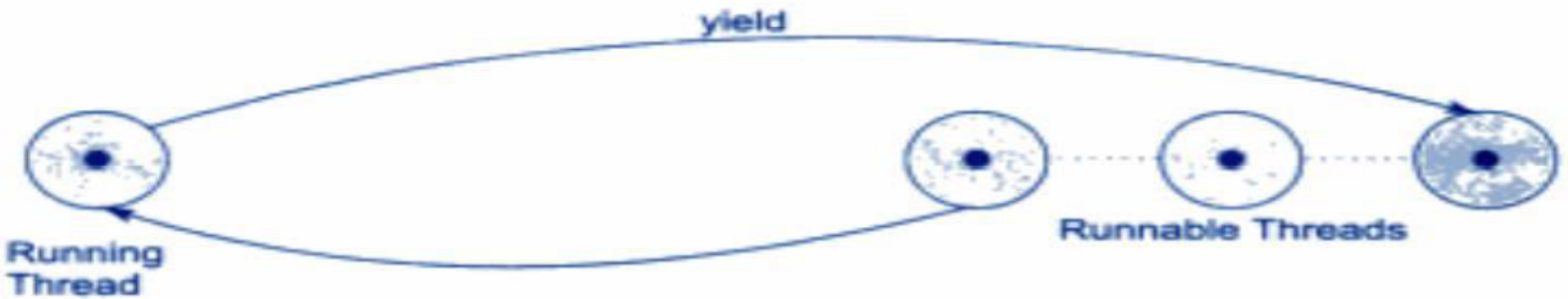
- ✓ New Born State: Thread is created
- ✓ Thread is invoked using
  - ✓ Thread.start( );
  - ✓ Moving a Thread to Runnable state
- ✓ Thread Object is killed using
  - ✓ Thread.stop( );
  - ✓ Moving a Thread to Dead state



Thread states

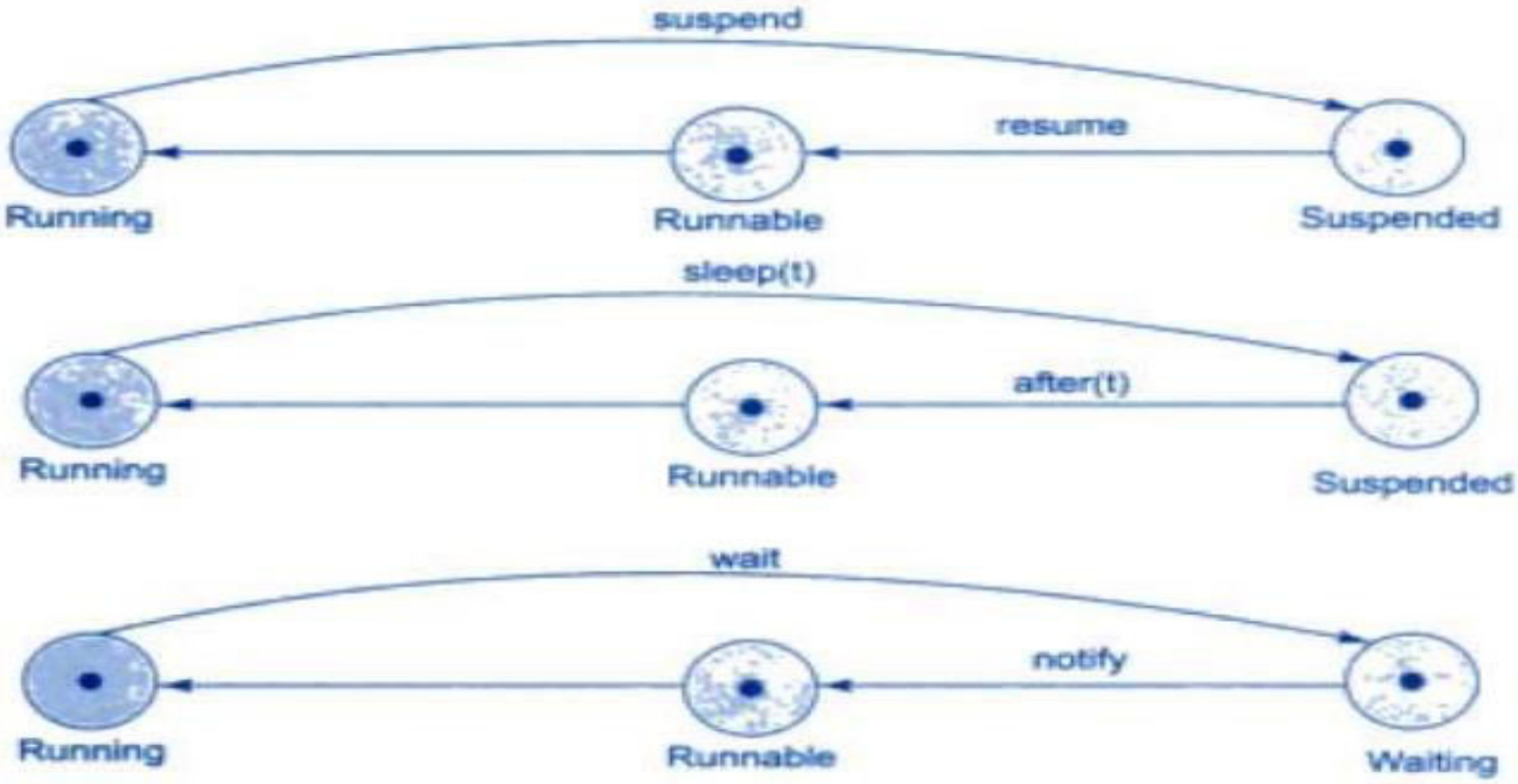
Runnable State

- Thread is ready for execution
- Thread is waiting in Queue to get CPU
- Round Robin: Equal Time Slots for Threads with same Priorities
- Thread.yield( ): Relinquish Control from Running Thread and move it to Runnable State Threads may be assigned Priorities
- Threads are arranged based on Priorities



Thread states

Running State: Thread is executed in CPU



Thread states

Thread.stop( );	Moving a Thread to Dead state
Thread.suspend( );	Block a Thread until further Orders Resumed
Thread.sleep( );	Block a Thread to Specified Time (ms)
Thread.wait( );	Block a Thread until a Condition occurs Notified

Thread states

Blocked State

Satisfied requirements is

- Thread is suspended,
- Thread is sleeping
- Thread is waiting

Dead State

- Life of the Thread Ends
- Thread Completes its execution
- Thread is killed after
  - ✓ New born
  - ✓ Running/ Runnable
  - ✓ Blocked

Thread Program using sleep(), yield(),stop() methods

```
class A extends Thread
{
    public void run ( )
    {
      for ( int i = 1; i <= 5; i++ )
      {
          if ( i == 1) yield( );
          System.out.println("From Thread A : i =  " + i);
      }
      System.out.println("Exit From Thread A");
    }
}
```

```
class B extends Thread
{
    public void run ( )
    {
      for ( int j = 1; j <= 5; j++ )
      {
          System.out.println("From Thread B : j =  " + j);
          if ( j == 3 ) stop( );
      }
      System.out.println("Exit From Thread B");
    }
}
```

Thread Program using sleep(), yield(),stop() methods

```
class C extends Thread
{
    public void run ( )
    {
      for ( int k = 1; k <= 5; k++ )
      {
        System.out.println("From Thread C : k =  " + k);
        if ( k == 1 )
            try
            {
               sleep(1000);
            }
            catch( Exception  e)  {        }

      }
      System.out.println("Exit From Thread C");
    }
}
```

```
class ThreadTest2
{
  public static void main(String args[ ])
  {
    A  threadA   = new A( );
    B  threadB   = new B( );
    C  threadC   = new C( );
    System.out.println(" Start Thread A  ");
    threadA.start( );
    System.out.println(" Start Thread B  ");
    threadB.start( );
    System.out.println(" Start Thread C  ");
    threadC.start( );
    System.out.println(" End of Main Thread  ");
  }
}
```

Thread Program Output

```
H:\SPS\Thread\2>java ThreadTest2
 Start Thread A
 Start Thread B
 Start Thread C
 End of Main Thread
From Thread B : j = 1
From Thread C : k = 1
From Thread A : i = 1
From Thread B : j = 2
From Thread A : i = 2
From Thread B : j = 3
From Thread A : i = 3
From Thread A : i = 4
From Thread A : i = 5
Exit From Thread A
From Thread C : k = 2
From Thread C : k = 3
From Thread C : k = 4
From Thread C : k = 5
Exit From Thread C
```

Run 1

```
H:\SPS\Thread\2>java ThreadTest2
 Start Thread A
 Start Thread B
 Start Thread C
 End of Main Thread
From Thread C : k = 1
From Thread A : i = 1
From Thread B : j = 1
From Thread A : i = 2
From Thread B : j = 2
From Thread A : i = 3
From Thread B : j = 3
From Thread A : i = 4
From Thread A : i = 5
Exit From Thread A
From Thread C : k = 2
From Thread C : k = 3
From Thread C : k = 4
From Thread C : k = 5
Exit From Thread C
```

Run 2

Implementing Runnable Interface

- ✓ Runnable interface declares run( ) method
- ✓ Declare a class that implements Runnable interface
- ✓ Implement run( )
- ✓ Create a Thread by defining an object that is instantiated from this “runnable” class of as the target of the Thread
- ✓ Call the Thread’s start( )

Implementing Runnable Interface

```
class X implements Runnable
{
    public void run( )
    {
        for(int i = 1;  i <= 10; i++)
        {
            System.out.println("\t ThreadX  :  " + i );
        }
         System.out.println("End of  ThreadX  " );
    }
}
```

Implementing Runnable Interface

```
Class RunnableTest
{
    public static void main(String args[ ])
    {
        X runnable = new X( );
        Thread threadX  =  new Thread(runnable);
        threadX.start( );
        System.out.println("End of  main Thread" );
    }
}
```

Thread by Runnable Interface

```
H:\SPS\Thread>java RunnableTest
End of main Thread
        ThreadX : 1
        ThreadX : 2
        ThreadX : 3
        ThreadX : 4
        ThreadX : 5
        ThreadX : 6
        ThreadX : 7
        ThreadX : 8
        ThreadX : 9
        ThreadX : 10
End of ThreadX
```

Thread Priorities

- Java thread priorities are in the range between MIN_PRIORITY (a constant of 1) and MAX_PRIORITY (a constant of 10). By default, every thread is given priority NORM_PRIORITY (a constant of 5).
- Threads with higher priority are more important to a program and should be allocated processor time before lower-priority threads.
- However, thread priorities cannot guarantee the order in which threads execute and are very much platform dependent.

Thread Priorities

```
import java.io.*;
class A extends Thread
{
  public void run()
  {
    for(int i=1;i<=5;i++)
    {
     System.out.println(i + "*" +3+ "=" +(i*3));
    }
    System.out.println("End of the 1st Thread");
  }
}
```

```
class B extends Thread
{
  public void run()
  {
    for(int j=1;j<=5;j++)
    {
     System.out.println(j + "*" +5+ "=" +(j*5));
    }
    System.out.println("End of the 2nd Thread");
  }
}
```

Thread Priorities

```
class C extends Thread
{
   public void run()
   {
      for(int k=1;k<=5;k++)
      {
        System.out.println(k + "*" +7+ "=" +(k*7));
       }
       System.out.println("End of the 3rd Thread");
    }
}
```

Thread Priorities

```
public class Multithread
{
    public static void main(String args[ ])throws IOException
    {
            A ThreadA=new A();
            B ThreadB=new B();
            C ThreadC=new C();
            ThreadA.setPriority(Thread.NORM_PRIORITY);
            ThreadB.setPriority(Thread.MAX_PRIORITY);
            ThreadC.setPriority(Thread.MIN_PRIORITY);
            System.out.println("The priority of Thread A is "+ThreadA.getPriority());
            System.out.println("The priority of Thread B is "+ThreadB.getPriority());
            System.out.println("The priority of Thread C is "+ThreadC.getPriority());
            ThreadA.start();
            ThreadB.start();
            ThreadC.start();
    }
}
```

Thread Priorities

```
H:\SPS\Thread>java ThreadPriority
The priority of Thread A is 5
The priority of Thread B is 10
The priority of Thread C is 1
B:1
A:1
C:1
B:2
C:2
A:2
C:3
B:3
C:4
A:3
C:5
B:4
End of the Thread **C**
A:4
B:5
A:5
End of the Thread **B**
End of the Thread **A**
```

Run 1

```
The priority of Thread A is 5
The priority of Thread B is 10
The priority of Thread C is 1
B:1
B:2
A:1
B:3
C:1
B:4
A:2
B:5
C:2
End of the Thread **B**
A:3
C:3
A:4
C:4
A:5
C:5
End of the Thread **C**
End of the Thread **A**
```

Run 2

Daemon Thread

- **Daemon thread in java** is a service provider thread that provides services to the user thread.
- Its life depend on the mercy of user threads i.e. when all the user threads dies, JVM terminates this thread automatically.
- There are many java daemon threads running automatically e.g. gc, finalizer etc.
- Points to remember for Daemon Thread in Java
  - It provides services to user threads for background supporting tasks. It has no role in life than to serve user threads.
  - Its life depends on user threads.
  - It is a low priority thread.

Daemon Thread

- Methods for Java Daemon thread by Thread class
  - The java.lang.Thread class provides two methods for java daemon thread.

No.	Method	Description
1)	public void setDaemon(boolean status)	is used to mark the current thread as daemon thread or user thread.
2)	public boolean isDaemon()	is used to check that current is daemon.

Daemon Thread Example

```
public class TestDaemonThread1 extends Thread{
 public void run(){
  if(Thread.currentThread().isDaemon()){//checking for daemon thread
   System.out.println("daemon thread work");
  }
  else{
  System.out.println("user thread work");
 }
 }
 public static void main(String[] args){
  TestDaemonThread1 t1=new TestDaemonThread1();//creating thread
  TestDaemonThread1 t2=new TestDaemonThread1();
  TestDaemonThread1 t3=new TestDaemonThread1();
```

Daemon Thread Example

```
t1.setDaemon(true);//now t1 is daemon thread

t1.start();//starting threads
t2.start();
t3.start();
}
}
```

Output

```
daemon thread work
user thread work
user thread work
```

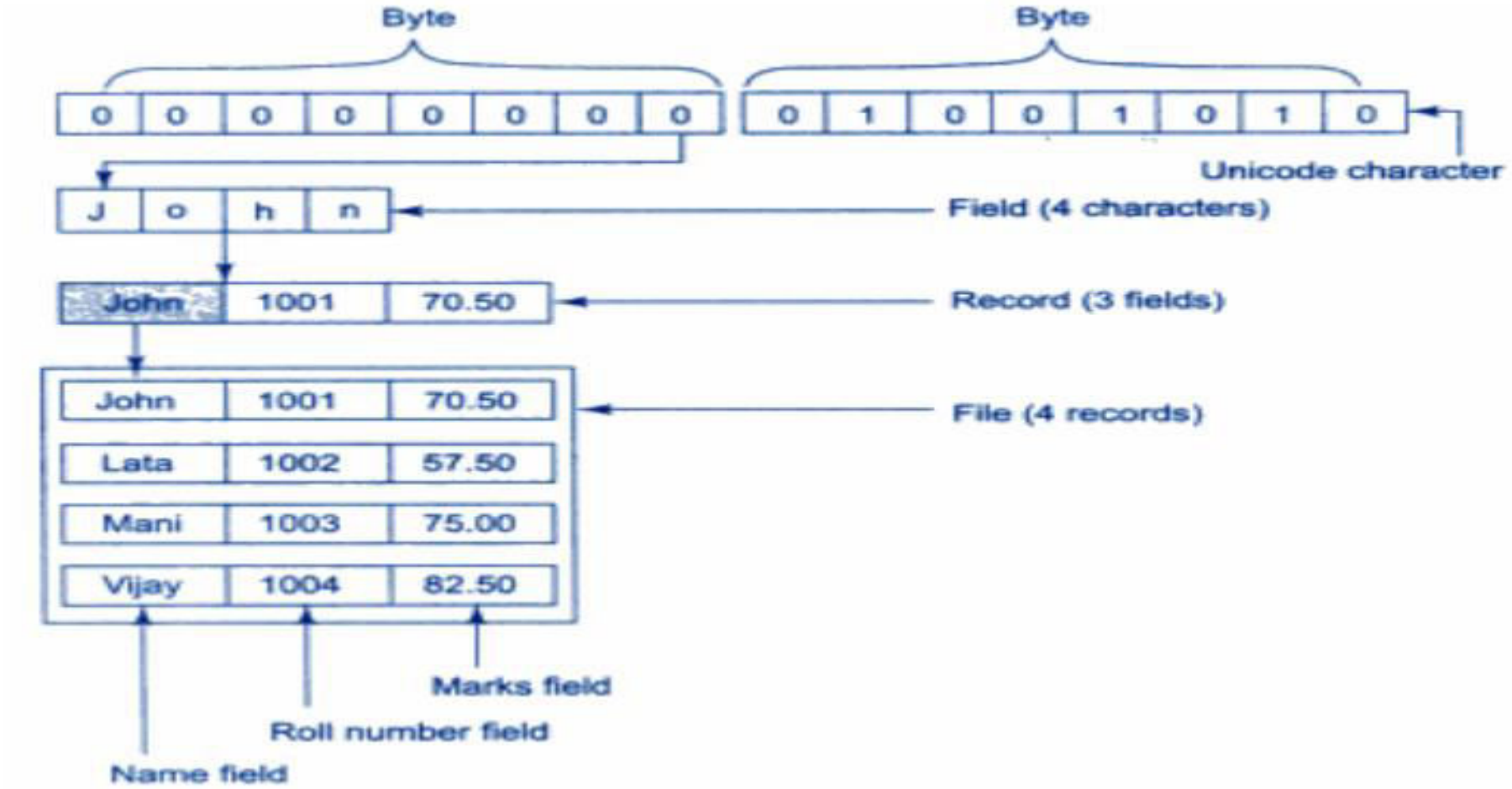
I/O Stream: Introduction

Data can be stored using variables and their values are used during execution. There are 2 drawbacks in it. They are

1. The data is lost either when a variable goes out of scope or when the program is terminated. That is, the storage is temporary.
2. It is difficult to handle large volumes of data using variables and arrays.

I/O Stream: Introduction

A file is a collection of related *records* placed in a particular area on the disk. A record is composed of several fields and a field is a group of characters as illustrated in Fig. Characters in Java are *Unicode* characters composed of two *bytes*, each byte containing eight binary digits, 1 or 0.



Data representation in Java files

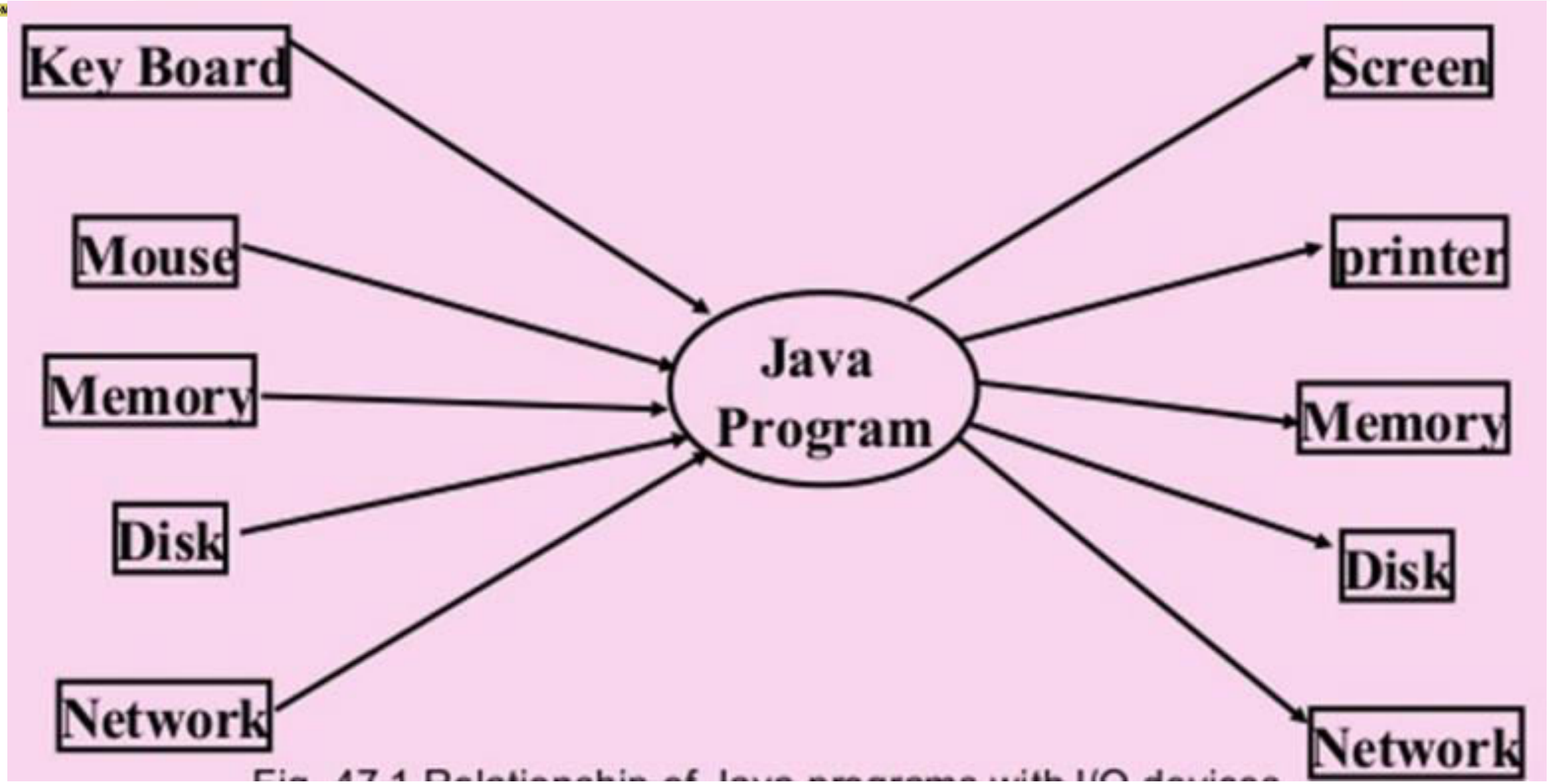
I/O Stream: Introduction

Storing and managing data using files is known as *file processing* which includes tasks such as creating files, updating files and manipulation of data. Java supports many powerful features for managing input and output of data using files. Reading and writing of data in a file can be done at the level of bytes or characters or fields depending on the requirements of a particular application. Java also provides capabilities to read and write class objects directly. Note that a record may be represented as a class object in Java. The process of reading and writing objects is called *object serialization*.

Concept of Streams

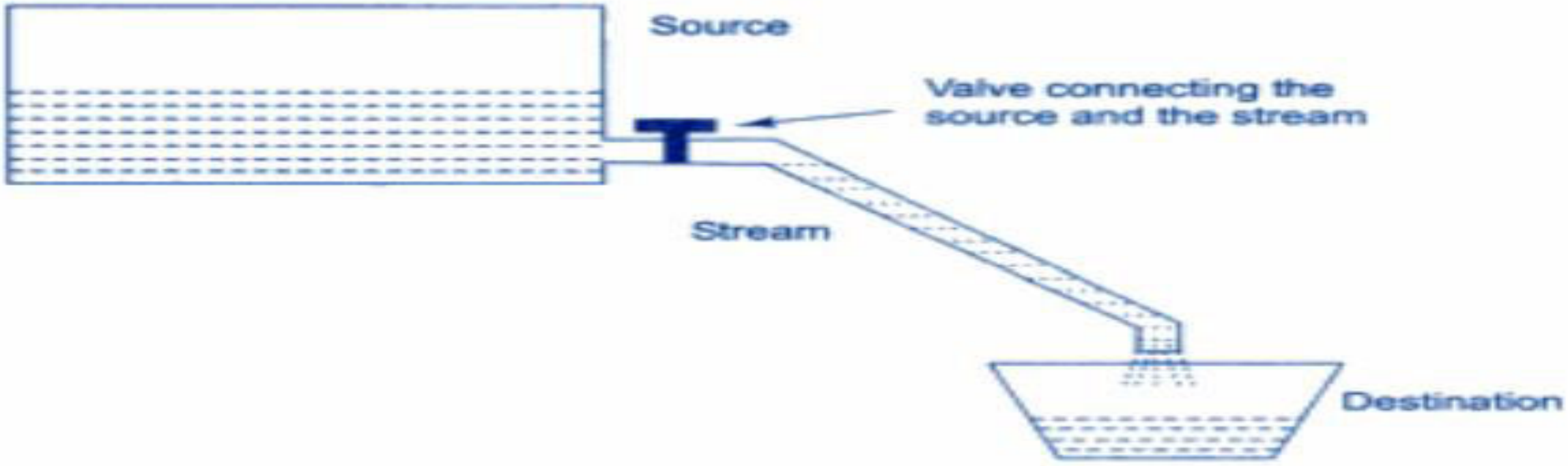
In file processing, input refers to the flow of data into a program and output means the flow of data out of a program. Input to a program may come from the keyboard, the mouse, the memory, the disk, a network, or another program. Similarly, output from a program may go to the screen, the printer, the memory, the disk, a network, or another program.

Concepts of Stream



I/O Stream: Introduction

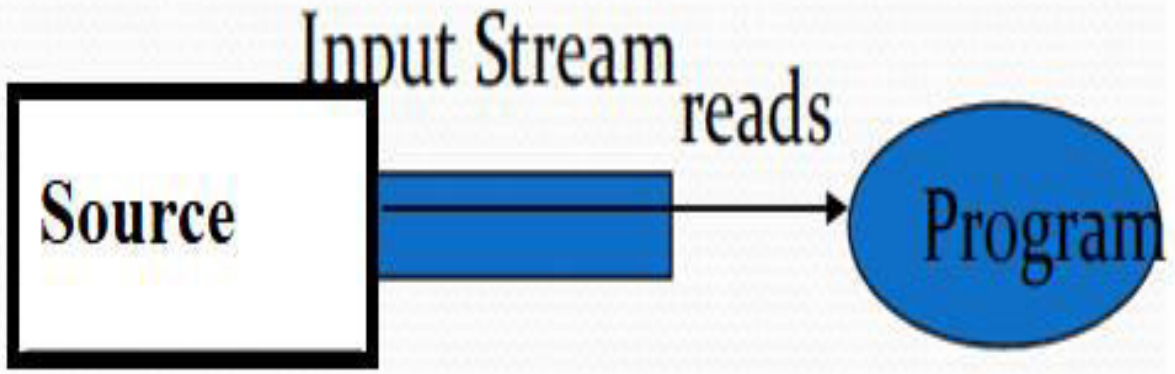
A stream in Java is a path along which data flows (like a river or a pipe along which water flows). It has a *source* (of data) and a *destination* (for that data) as depicted in Fig. Both the source and the destination may be physical devices or programs or other streams in the same program.

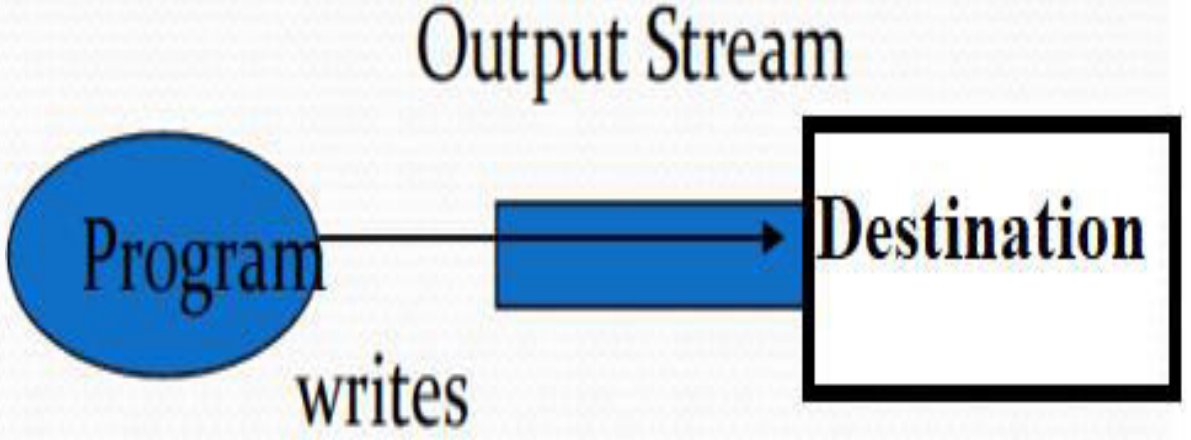


Conceptual view of a stream

Types of Streams

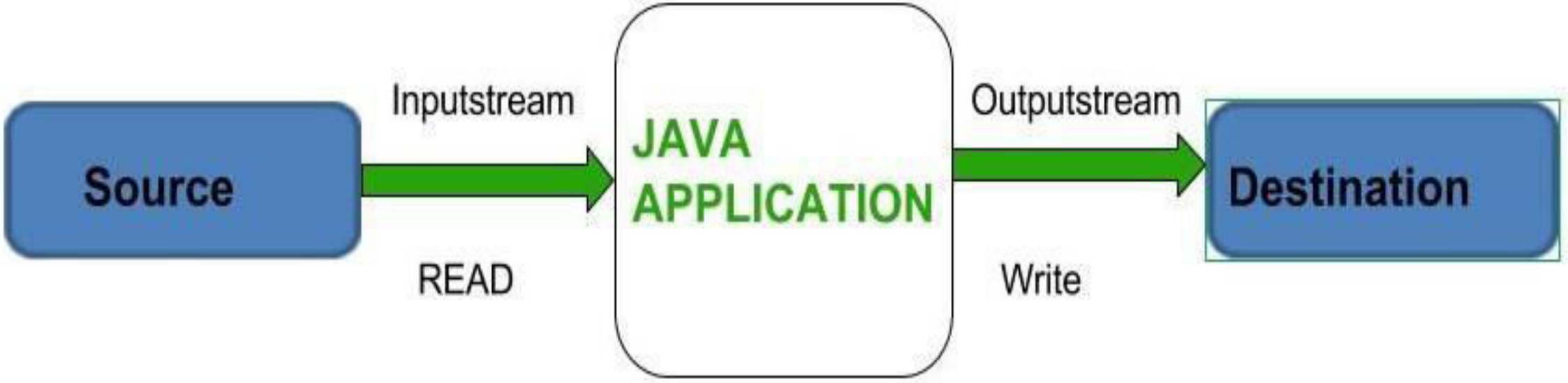
- A stream can be defined as a sequence of data.
- Two kinds of stream:
  - **InputStream:** The InputStream is used to read data from a source.
  - **OutputStream:** The OutputStream is used for writing data to a destination.



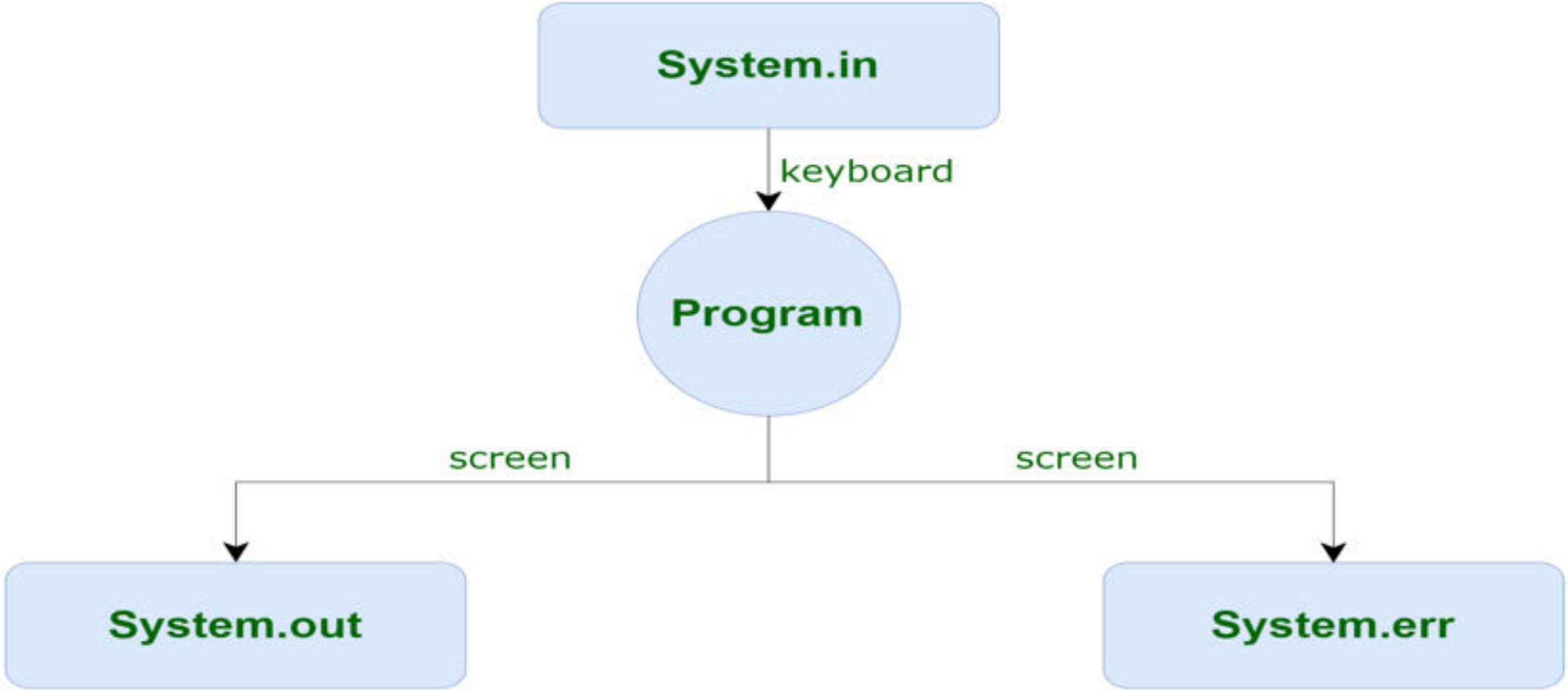


I/O Stream: Introduction

- Java Streams with its I/O package that helps the user to perform all the input-output operations.
- These streams support all the types of objects, data-types, characters, files etc to fully execute the I/O operations.



I/O Stream: Introduction



Standard I/O Streams in Java

I/O Stream: Introduction

- **System.in**: This is the standard input stream that is used to read characters from the keyboard or any other standard input device.
- **System.out:** This is the standard output stream that is used to produce the result of a program on an output device like the computer screen.

List of the various print functions that we use to output statements:

- print(): display a text on the console the cursor remains at the end of the text at the console. Used as **System.out.print(parameter);**
- println(): prints the text on the console and the cursor moves to the start of the next line at the console. Used as **System.out.println(parameter);**
- printf()**:** similar to printf in C it may take multiple arguments.

I/O Stream: Introduction

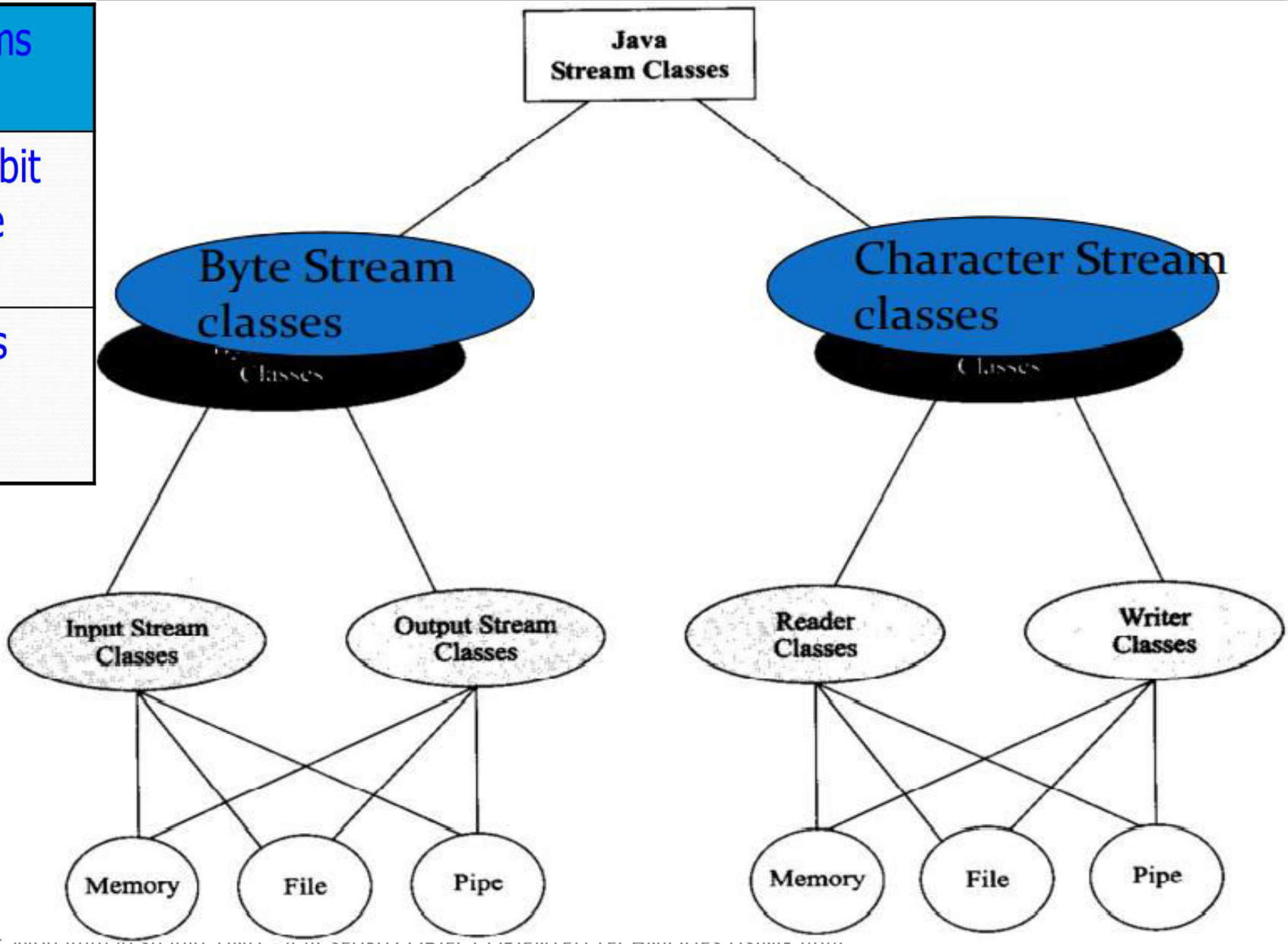
- **System.err:** This is the standard error stream that is used to output all the error data that a program might throw, on a computer screen or any standard output device.

This stream also uses all the 3 above-mentioned functions to output the error data:

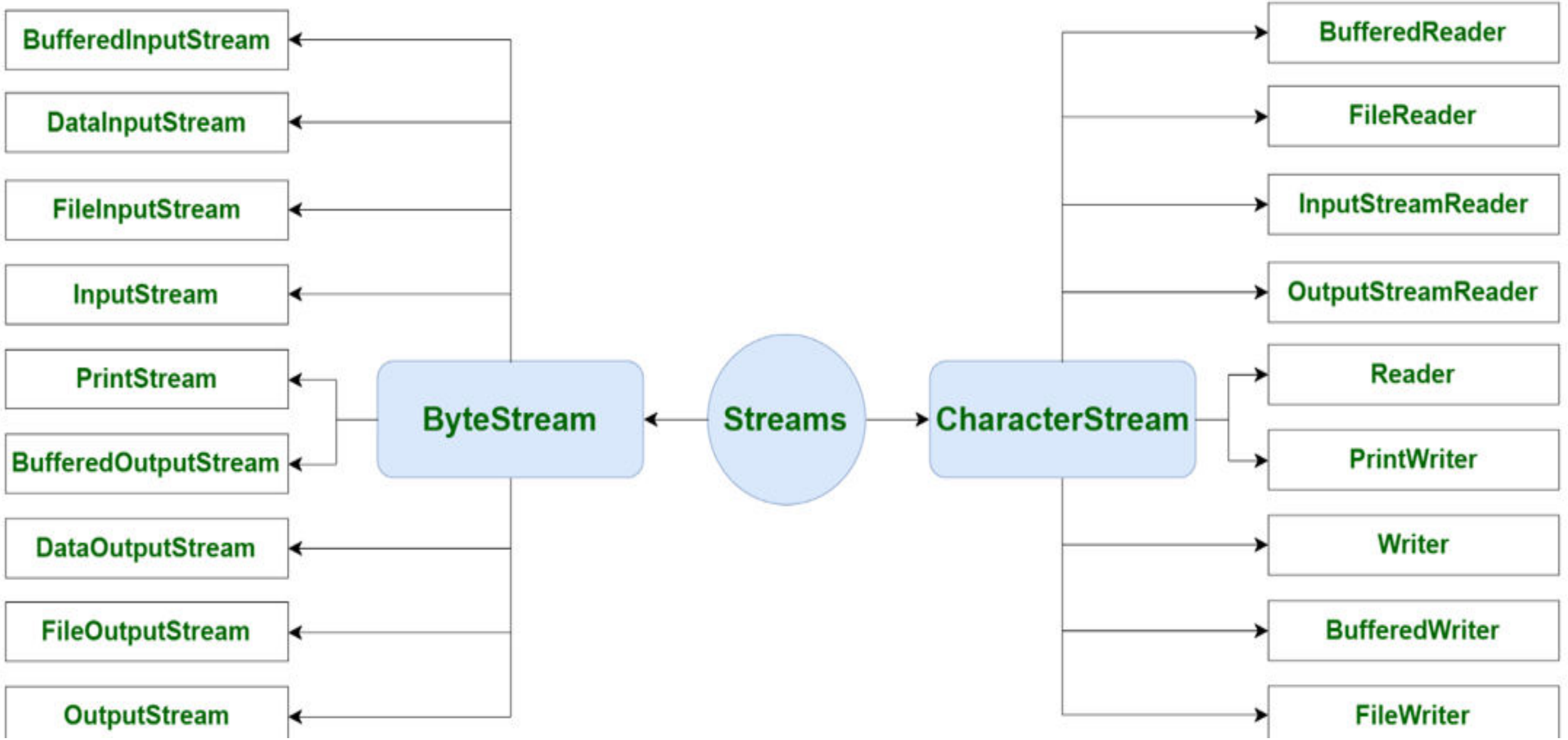
- print()
- println()
- printf()

Classification of java stream classes

Byte Streams	Character streams
Operated on 8 bit (1 byte) data.	Operates on 16-bit (2 byte) unicode characters.
Input streams/Output streams	Readers/ Writers



Types of Streams



Types of Streams

- Byte**Stream:** This is used to process data byte by byte (8 bits).
- Though it has many classes, the FileInputStream and the FileOutputStream are the most popular ones.
- **FileInputStream** is used to read from the source and **FileOutputStream** is used to write to the destination.
- **CharacterStream:** In Java, characters are stored using Unicode conventions. Character stream automatically allows us to read/write data character by character. Though it has many classes, the FileReader and the FileWriter are the most popular ones.
- **FileReader** and **FileWriter** are character streams used to read from the source and write to the destination respectively.

Conti.....

- **Useful methods of Input Stream**

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

Common I/O Stream Operations

- **Useful methods of Output Stream**

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

Hierarchy of Byte Input Stream Classes

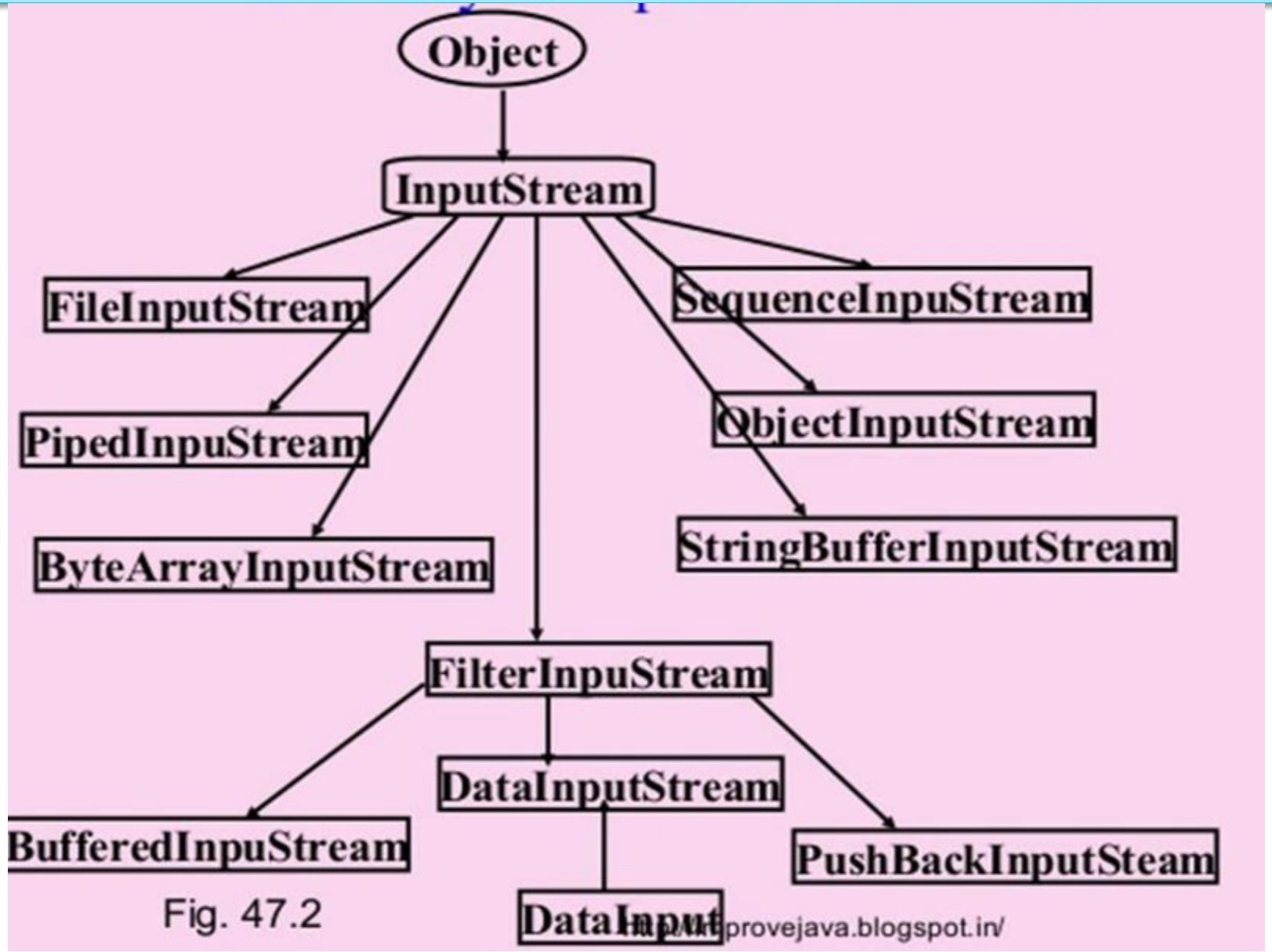


Fig. 47.2

Hierarchy of Byte Output Stream Classes

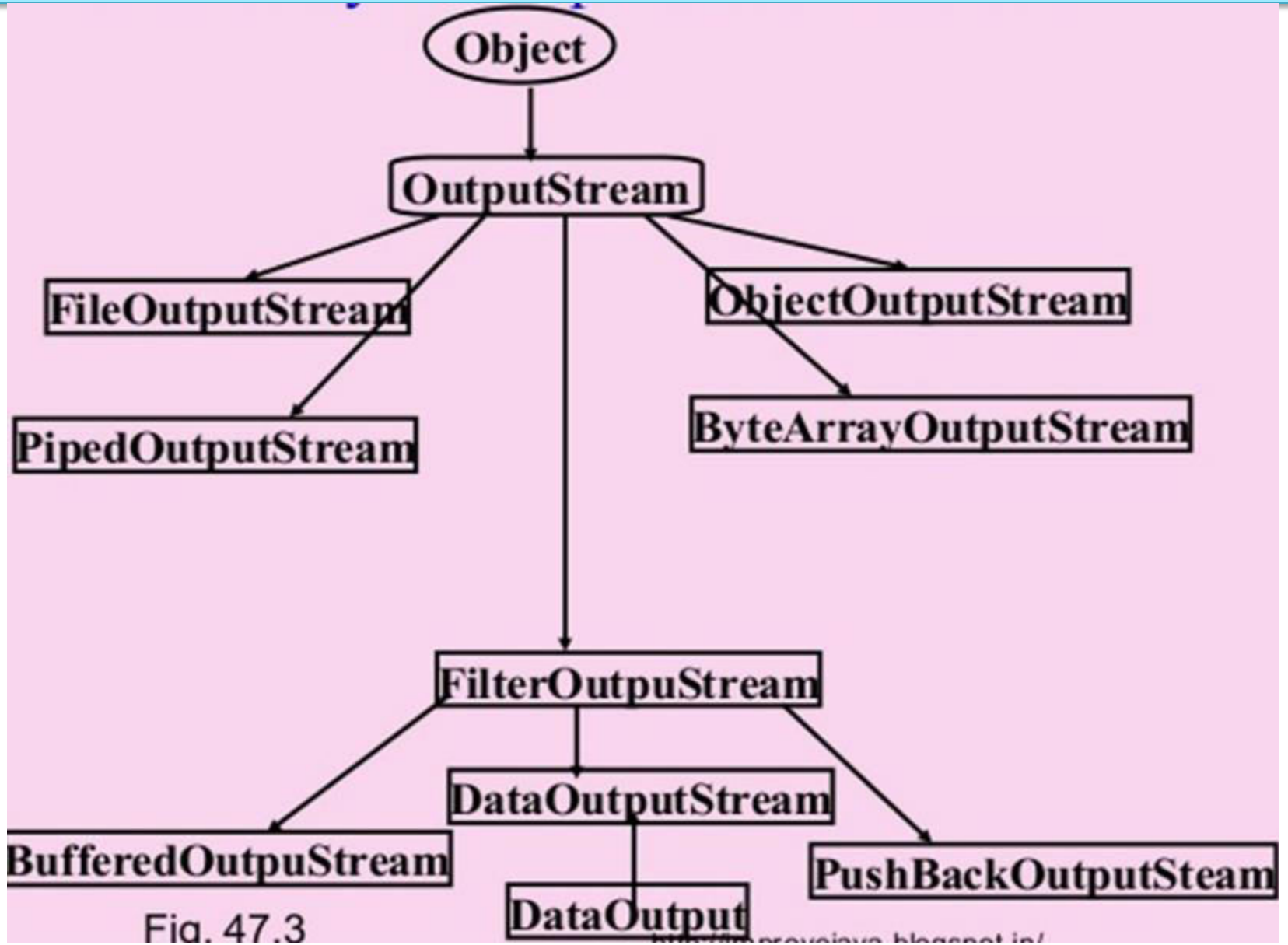


Fig. 47.3

Character Stream

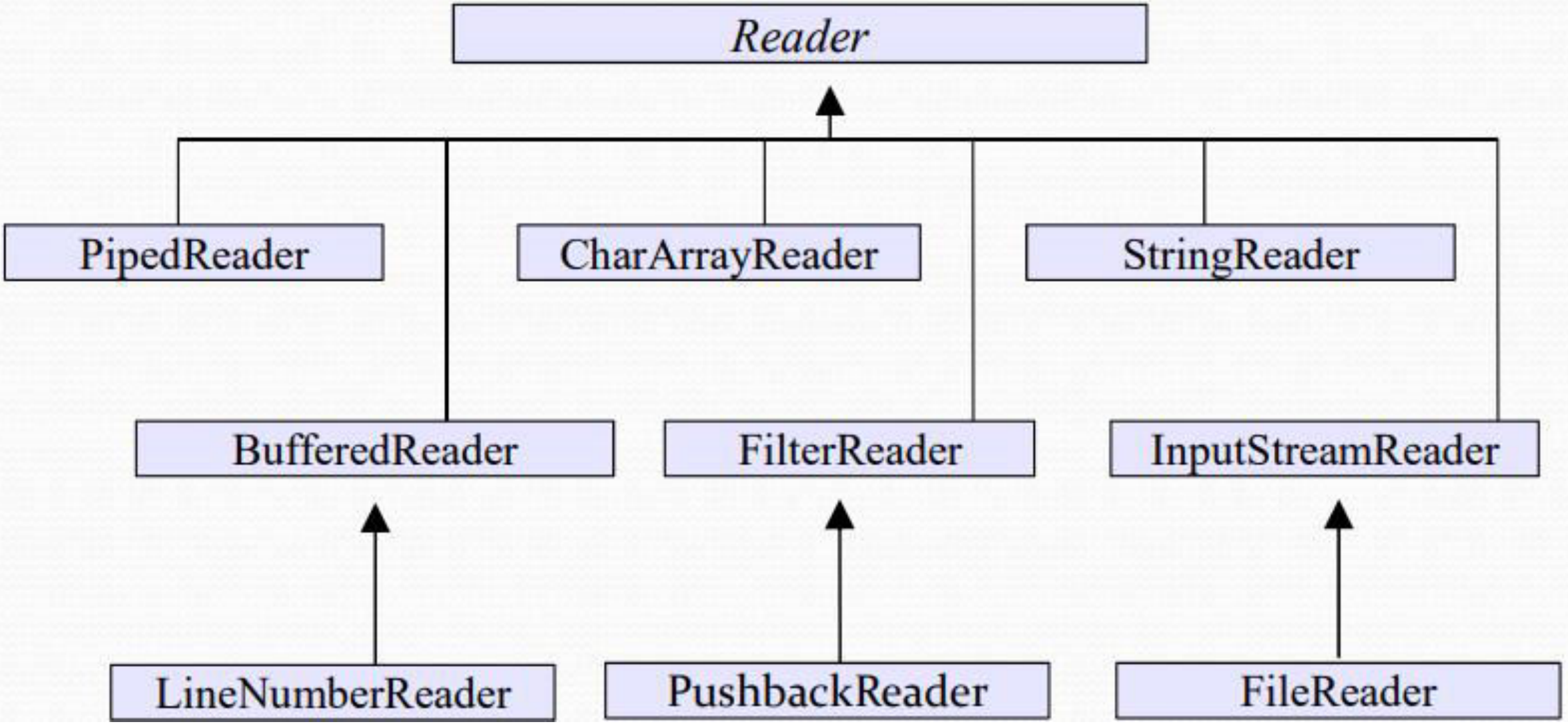
Character Streams:

It supports 16-bit Unicode character input and output. There are two classes of character stream as follows:

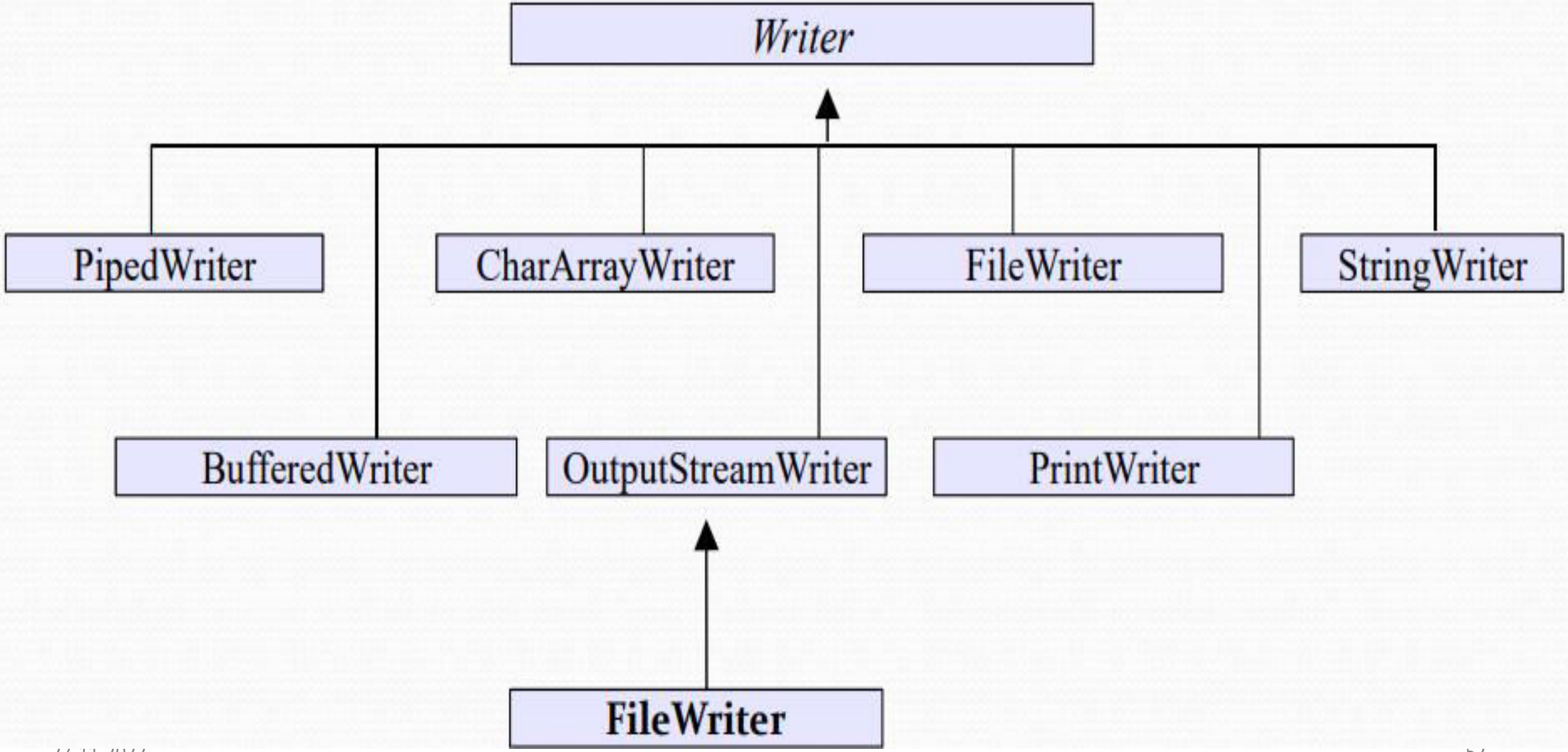
1. Reader
2. Writer

These classes allow internationalization of Java I/O and also allow text to be stored using international character encoding.

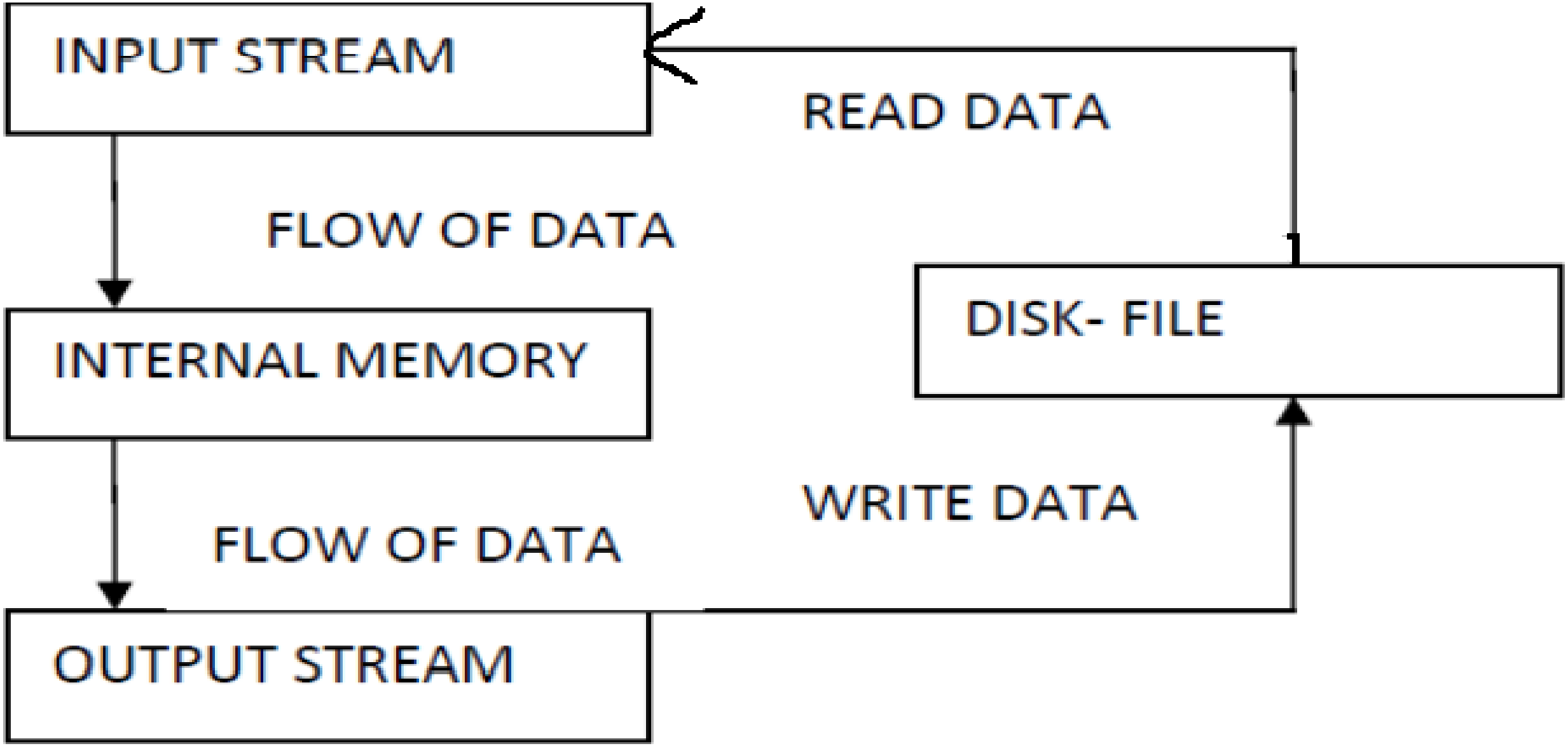
Hierarchy of Reader Classes



Hierarchy of Writer Classes



Working of I/O Stream



Working on File

Using the File Class

The **java.io** package includes a class known as the **File** class that provides support for creating files and directories. The class includes several constructors for instantiating the **File** objects. This class also 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 a file
- Checking the existence of a file
- Renaming a file
- Checking whether the file is writable
- Checking whether the file is readable

Input/ Output Exceptions

I/O Exception class	*Function*
EOFException	Signals that an end of the file or end of stream has been reached unexpectedly during input
FileNotFoundException	Informs that a file could not be found
InterruptedIOException	Warns that an I/O operations has been interrupted
IOException	Signals that an I/O exception of some sort has occurred

A Program copy data from input.dat to output.dat

```
//  Copying characters from one file into another
import  java.io.*;
class  CopyCharacters
{    public  static  void  main  (String  args  [  ]  )
     {
          //  Declare and create input and output files
          File inFile = new File ("input.dat") ;
          File outFile  = new File ("output.dat") ;
          FileReader ins = null;                    //  Creates file stream ins
          FileWriter outs = null;                   //  Creates file stream outs
          try
          {
               ins = new FileReader (inFile) ;      //  Opens inFile
               outs = new FileWriter (outFile) ;    //  Opens outFile
               //  Read and write till the end
               int  ch;
               while ( (ch = ins.read( ) ) != -1)
               {
                    outs.write  (ch)  ;
               }
          }
          catch  (IOException  e)
          {
               System.out.println  (e)  ;
               System.exit  (-1)  ;
          }
          finally    //  Close files
          {
               try
               {
                    ins.close  ( )  ;
                    outs.close  ( )  ;
               }
               catch  (IOException  e)  {  }
          }
     }
}
```

Program Filer handling

Program copy data from input.txt to output.txt

```
import java.io.*;
class copychar
{
 public static void main(String str[])
 {
  File inFile=new File("H:\\SPS\\IO\\File\\input.txt");
  File outFile=new File("H:\\SPS\\IO\\File\\output.txt");
  FileReader instream =null;
  FileWriter outstream =null;H:\SPS\IO\File
  try{
      instream=new FileReader(inFile);
      outstream=new FileWriter(outFile);
      int ch;
```

Program Filer handling

```
            while((ch=instream.read())!= -1)
            {
                outstream.write(ch);
            }
        }
    catch(IOException e)
      {
        System.out.println(e);
        System.exit(-1);
      }
    finally
      {
          try
              {
                  instream.close();
                  outstream.close();
              }
          catch(IOException e){}
      }
}
```

Program1: Copy file1 to file 2

```
import java.io.*;
class copychar
{
 public static void main(String str[])
 {
  File inFile=new File("H:\\NIET\\IO\\File\\input.txt");
  File outFile=new File("H:\\NIET\\IO\\File\\output.txt");
  FileReader instream =null;
  FileWriter outstream =null;
  try{
      instream=new FileReader(inFile);
      outstream=new FileWriter(outFile);
      int ch;
      while((ch=instream.read())!= -1)
      {
          outstream.write(ch);
      }
     }
   catch(IOException e)
```

Program1: Copy file1 to file 2

```
    catch(IOException e)
      {
         System.out.println(e);
         System.exit(-1);
      }
    finally
      {
          try
              {
                 instream.close();
                 outstream.close();
              }
          catch(IOException e){}

      }
  }

}
```

Program2: Merge two file f1, f2

```
//concatenating and buffering
import java.io.*;
class FileBuffering
{
 public static void main(String str[]) throws IOException
 {

   FileInputStream file1 =null;
   FileInputStream file2 =null;
   SequenceInputStream file3=null;
   //open file to be concatenated
   file1=new FileInputStream("H:\\NIET\\IO\\File\\f1.txt");
   file2=new FileInputStream("H:\\NIET\\IO\\File\\f2.txt");
   //f3=f1+f2
   file3 = new SequenceInputStream(file1,file2);
   //create buffering input and output stream
   BufferedInputStream inBuffer=new   BufferedInputStream(file3);
   BufferedOutputStream outBuffer=new   BufferedOutputStream(System.out);
```

Program2: Merge two file f1, f2

```
//read and write till the end of buffer
int ch;
while((ch = inBuffer.read()) != -1)
  {
    outBuffer.write((char)ch);

  }
 System.out.println(outBuffer);
  inBuffer.close();
  outBuffer.close();
  file1.close();
  file2.close();
}
}
```

Annotations: Introduction

- **Annotation** is a tag that represents the *metadata* i.e. attached with class, interface, methods or fields to indicate some additional information which can be used by java compiler and JVM.
- Annotations are used to provide supplement (additional) information about a program.
- Annotations start with '@'.
- Annotations do not change action of a compiled program.

Built-In Java Annotations used in Java code

- @Override
- @SuppressWarnings
- @Deprecated

Annotations: Introduction (CO4)

@Override

@Override annotation assures that the subclass method is overriding the parent class method. If it is not so, compile time error occurs.

@SuppressWarnings

@SuppressWarnings annotation is used to suppress warnings issued by the compiler.

@Deprecated

@Deprecated annotation marks that this method is deprecated so compiler prints warning. It informs user that it may be removed in the future versions. So, it is better not to use such methods.

Override Annotations

```
class Animal{
void eatSomething(){  System.out.println("eating something");}
                              }

class Dog extends Animal {
        @Override

void eatsomething(){System.out.println("eating foods");}//should be eatSomething
}

class TestAnnotation1{
public static void main(String args[]){
Animal a=new Dog();
a.eatSomething();
}}
```

```
Output:Comple Time Error
```

Deprecated Annotations

```
class A{
        void m(){System.out.println("hello m");}

         @Deprecated
        void n(){System.out.println("hello n");}
          }

class TestAnnotation3{
        public static void main(String args[]){

         A a=new A();
        a.n();
}}
```

```
Note: Test.java uses or overrides a deprecated API.

Note: Recompile with -Xlint:deprecation for details.
```

Runtime:

```
hello n
```

SuppressWarnings Annotations

```
import java.util.*;
class TestAnnotation2{
    @SuppressWarnings("unchecked")
    public static void main(String args[]){
      ArrayList list=new ArrayList();
     list.add("sonoo");
     list.add("vimal");
     list.add("ratan");
     for(Object obj:list)
       System.out.println(obj);
}}
```

```
Now no warning at compile time.
```

Custom (User-defined)Annotations

@interface element is used to declare an custom annotation.

- For example: @interface MyAnnotation{ }
- Points to remember for java custom annotation signature
  - Method should not have any throws clauses
  - Method should return one of the following: primitive data types, String Class, enum or array of these data types.
  - Method should not have any parameter.
  - Example**:** **String myMethod** ( ) {

return mystring;}

Applying Annotations

Steps:

1 creating a custom annotation is **to declare it using the *@interface* keyword**

2 add meta-annotations to specify the scope and the target of our custom annotation.

first annotation **has runtime visibility, and we can apply it to types (classes).**

3 Apply annotation

Applying Annotations

- Example of custom annotation: creating, applying and accessing annotation

```
//Creating annotation
import java.lang.annotation.*;
import java.lang.reflect.*;

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
@interface MyAnnotation{
int value();
}

//Applying annotation
class Hello{
@MyAnnotation(value=10)
public void sayHello(){System.out.println("hello annotation");}
}
```

Applying Annotations

```
//Accessing annotation
class TestCustomAnnotation1{
public static void main(String args[])throws Exception{


Hello h=new Hello();
Method m=h.getClass().getMethod("sayHello");


MyAnnotation manno=m.getAnnotation(MyAnnotation.class);
System.out.println("value is: "+manno.value());
}}
```

```
Output:value is: 10
```

DAILY QUIZ

1. What is Multithreading? What are the ways to create multiple threads in java. [CO4]
2. Explain about Thread Life Cycle. [CO4]
3. Define Daemon Threads? Explain with an example. [CO4]
4. Write a java program to implement join() method in multithreading. [CO4]
5. What are the states of a thread? [CO4]
6. What are the threads will start, when you start the java program? [CO4]
7. What are the different identifier states of a Thread? [CO4]
8. Why do threads block on I/O? [CO4]
9. What is synchronization and why is it important? [CO4]

Weekly Assignment

Q1. Explain Threads and types of threads in java.[CO4]

Q2. Explain Thread Life Cycle in detail.[CO4]

Q3. Explain Thread Creation in java with suitable example.[CO4]

Q4. What is Daemon Thread? Give its example.[CO4]

Q5. Explain input output stream operations in java.[CO4]

Q6. What are Custom Annotations? Explain how to create custom annotations in java with suitable example.[CO4]

Q7. Explain various types of Output Streams in java.[CO4]

Q8. What do you understand by Annotations? Explain various built in java annotations in java.[CO4]

Q9.Explain various methods of input stream in detail.[CO4]

Q10. Explain various output streams methods in detail. [CO4]

TOPIC LINKS

- https://www.youtube.com/watch?v=qQVqfvs3p48
- https://www.youtube.com/watch?v=TCd8QIS-2KI
- https://www.youtube.com/watch?v=vd9nJK22BwU

MCQ

Q1. Which of these method of Thread class is used to find out the priority given to a thread?[CO4]

a) get()

b) ThreadPriority()

c) getPriority()

d) getThreadPriority()

Q2. Which of these method of Thread class is used to Suspend a thread for a period of time?[CO4]

a) sleep()

b) terminate()

c) suspend()

d) stop()

MCQ

Q3.Which function of pre defined class Thread is used to check weather current thread being checked is still running?[CO4]

a) isAlive()

b) Join()

c) isRunning()

d) Alive()

Q4. In java a thread can be created by .....[CO4]

A. Extending the thread class.

B.Implementing Runnable interface.

C.Both of the above

D.None of these

MCQ

Q5.Thread synchronization in a process will be required when[CO4]

A.All threads sharing the same address space

B.All threads sharing the same global variables

C.All threads sharing the same files

D.All

Q6.Which thread will be executed first if two threads have same priority[CO4]

A. They will fall in starvation and none will be executed.

B. Both will be executed simultaneously

C. It depends upon operating system

D.They will be executed on first come first serve basis

MCQ

Q7.The life cycle of a thread in java is controlled by[CO4]

A.JRE

B.JDK

C.JVM

D.None

Q8.Which method is used to get current running thread object?[CO4]

A.runningThread()

B.currentThread()

C.runnableThread()

D.None

MCQs

Q9.Which version of Java introduced annotation?[CO4]

a) Java 5

b) Java 6

c) Java 7

d) Java 8

Q10.Annotations which are applied to other annotations are called meta annotations.[CO4]

a) True

b) False

Q11.Which of these is used to perform all input & output operations in Java?[CO4]

a) streams

b) Variables

c) classes

d) Methods

MCQ

Q12.Which of these is a type of stream in Java?[CO4]
a) Integer stream
b) Short stream
c) Byte stream
d) Long stream

Q13.Which of these classes are used by Byte streams for input and output operation?[CO4]
a) InputStream
b) InputOutputStream
c) Reader
d) All of the mentioned

Q14. Which of these classes are used by character streams for input and output operations?[CO4]
a) InputStream
b) Writer
c) ReadStream
d) InputOutputStream

MCQ

Q15.Daemon thread runs in [CO4]

A. Background

B. Foreground

C. Both

D.None

Q16. Which method is used to create a daemon thread?[CO4]

A. setDaemon(boolean value)

B. setThread(boolean value)

C. createDaemon(boolean value)

D. makeDeamon(boolean value);

MCQ

Q17. Which will contain the body of the thread in Java?[CO4]

A. Start()

B. Run()

C. Main()

D. Execute()

Q18. To create a daemon thread [CO4]

A. First thread setDaemon() is called then start()

B. First thread start() is called then setDaemon()

C. Call thread start() and setDaemon() in any order

D.All correct

Glossary Questions

1. Attempt all the parts. Pick the correct option from glossary. [CO4]

i) JVM ii) Run() iii) Background iv) streams

a) _____method contain the body of the thread in java.

b) By using _____ input and output operations are performed in java.

c) Daemon threads run in the _____.

d) Thread Lifecycle in java is controlled by _____.

Sessional Paper-1

Printed page: 2 Subject Code: ACSE0302

Roll No: | | | | | | | | | | | | | |

NOIDA INSTITUTE OF ENGINEERING AND TECHNOLOGY ,GREATER NOIDA

(An Autonomous Institute Affiliated to AKTU, Lucknow)

B. Tech (AI / AIML / IOT / DS)

(SEM:III SESSIONAL EXAMINATION – I)(2021-2022)

Subject Name: OBJECT ORIENTED TECHNIQUES USING JAVA

Time: 1.15 Hours Max. Marks:30

General Instructions:

- All questions are compulsory. Answers should be brief and to the point.
- This Question paper consists of 2 pages & 5 questions.
- It comprises of three Sections, A, B, and C. You are to attempt all the sections.
- Section A -Question No- 1 is objective type questions carrying 1 mark each, Question No- 2 is very short answer type carrying 2 mark each. You are expected to answer them as directed.
- Section B - Question No-3 is short answer type questions carrying 5 marks each. You need to attempt any two out of three questions given.
- Section C - Question No. 4 &5are Long answer type (within unit choice) questions carrying 6marks each. You need to attempt any one part a or b.
- Students are instructed to cross the blank sheets before handing over the answer sheet to the invigilator.
- No sheet should be left blank. Any written material after a blank sheet will not be evaluated/checked.

		SECTION – A	[8]	
1.	Attempt all parts.		(4×1=4)	CO
	a.	The type of Arguments the main method accepts is ____. a) integer[ ] b) String c) float[ ] d) String[ ]	(1)	CO2
	b.	The default value for data field of a boolean type, numeric type is _____ , _____ respectively. a) false, 1 b) true, 0 c) false, 0 d) true, 1	(1)	CO1
	c.	Using _____, we can force immediate termination of a loop. a) break b) continue c) return d) goto	(1)	CO1
	d.	The _____ statement is used to explicitly return from a method. a) break b) continue c) return d) goto	(1)	CO2
2.	Attempt all parts.		(2×2=4)	CO
	a.	Write a JAVA program to check whether the number entered by user is even or odd by using if-else. (Use the Scanner class to enter the integer	(2)	CO1

Page 1 of 2

Conti….

		number)		

	b.	What is the output of the following Java code snippet?
		int i=0;
		for(i=1; i <= 6; i++)
		{
		if (i % 3 = = 0)
		continue;
		System.out.print (i+",");
		}
			(2)	CO1

		SECTION – B		
3.	Answer any two of the following-		[2×5=10]	CO
	a.	Draw a class diagram of Student class and explain how the access modifiers are represented in the class diagrams?	(5)	CO1
	b.	Discuss different levels of access specifiers available in JAVA.	(5)	CO1
	c.	What is the purpose of constructors? Explain all the types of constructors in JAVA with the help of example of your choice.	(5)	CO2
		SECTION – C		
4	Answer any one of the following-(Any one can be applicative if applicable)		[2×6=12]	CO
	a.	Explain the four pillars of Object-Oriented Programming with the help of examples.	(6)	CO1
	b.	Write a JAVA program to display all even numbers from 100 to 50 using all the loops statements you have studied.	(6)	CO1
5.	Answer any one of the following-			
	a.	Write a program in JAVA that takes arguments name, department and marks of 4 subjects from the user and then print total and average marks obtained. (Use command line arguments for giving input).	(6)	CO1
	b.	Write a JAVA program to display the reverse of an input number. (Use Scanner class to input the positive integer)	(6)	CO1

Sessional Paper-2

Printed page: 2 **Subject Code:ACSE0302**

Roll No: | | | | | | | | | | | | | |

NOIDA INSTITUTE OF ENGINEERING AND TECHNOLOGY, GREATER NOIDA

(An Autonomous Institute)

Affiliated to Dr. A.P. J. Abdul Kalam Technical University, Uttar Pradesh, Lucknow

Course ...B.Tech.....Branch...IT....

Semester.....III...Sessional Examination.....II.....Year- (2021 - 2022)

Subject Name: OBJECTS ORIENTED TECHNIQUES USING JAVA

Time: 1.15Hours **[SET- A]** **Max. Marks:30**

General Instructions:

- This Question paper consists of 2 pages & 5 questions.It comprises of three Sections, A, B, and C
- Section A -Question No- 1 is objective type questions carrying 1 mark each, Question No- 2 is very short answer type carrying 2 mark each. You are expected to answer them as directed.
- Section B - Question No-3 is Short answer type questions carrying 5 marks each. Attempt any two out of three questions given.
- Section C -Question No. 4 &5are Long answer type (within unit choice) questions carrying 6marks each. Attempt any one part *a* or *b*.

Conti….

		SECTION – A	[08Marks]	
1.	All questions are compulsory		(4×1=4)	
	a.	Identify the correct way to call the constructor of class "Super" that is inherited by the class "Child". (i) class Child extends Super{ Child(){ Super();}} (ii) class Child extends Super{ Child(){ super();}} (iii) class Child extends Super{ Child() { super.Super();}} (iv) All the ways are correct.	(1)	CO2
	b.	Total abstraction can be achieved using? (i) abstract class (ii) interface (iii) both (iv) total abstraction cannot be achieved	(1)	CO2
	c.	The …… class inherits all the properties of the …… class? (i) base, derived (ii) derived base (iii) base, initial (iv) base, final	(1)	CO2
	d.	Runtime polymorphism is also known as: (i) Dynamic binding (ii) Static binding (iii) Early binding (iv) None of the above	(1)	CO2
2.	All questions are compulsory		(2×2=4)	
	a.	Explain a simple program showing garbage collection.	(2)	CO2
	b.	Explain the concept of interface using suitable example.	(2)	CO2

Conti..

SECTION – B			**[10Marks]**	
3.	**Answer any two of the following-**		**(2×5=10)**	
	a.	**Explain the types of polymorphism in java using suitable examples for each type.**	**(5)**	**CO2**
	b.	**Explain the difference between interface and abstract class in java using suitable example.**	**(5)**	**CO2**
	c.	**Explain inheritance in java. Explain all types of inheritance supported by java using suitable examples.**	**(5)**	**CO2**
SECTION – C			**[12Marks]**	
4	**Answer any one of the following-**		**(1×6=6)**	
	a.	**Explain the working of "this" and "super" keyword in java. Illustrate each using suitable example.**	**(6)**	**CO2**
	b.	**Explain abstract class. State the use of abstract class with suitable example. Also write the names of OOPs concepts that is used to implement abstract class and that is implemented using abstract class.**	**(6)**	**CO2**
5.	**Answer any one of the following-**		**(1×6=6)**	
	a.	**Explain the concept of access modifiers (public, private, protected) using packages.**	**(6)**	**CO3**
	b.	**Explain overloading and overriding of methods? Illustrate overloading and overriding of methods in Java with suitable examples.**	**(6)**	**CO2**

NOTE: No example should be repeated.

Old University Question Paper

Printed Page:-

Subject Code:- ACSE0302

Roll. No:

NOIDA INSTITUTE OF ENGINEERING AND TECHNOLOGY, GREATER NOIDA

(An Autonomous Institute Affiliated to AKTU, Lucknow)

B.Tech.

SEM: III - THEORY EXAMINATION (2021 - 2022)

Subject: Object Oriented Techniques using Java

Time: 03:00 Hours

Max. Marks: 100

General Instructions:

1. All questions are compulsory. It comprises of three Sections A, B and C.

- Section A - Question No- 1 is objective type question carrying 1 mark each & Question No- 2 is very short type questions carrying 2 marks each.
- Section B - Question No- 3 is Long answer type - I questions carrying 6 marks each.
- Section C - Question No- 4 to 8 are Long answer type - II questions carrying 10 marks each.
- No sheet should be left blank. Any written material after a Blank sheet will not be evaluated/checked

Old University Question Paper

SECTION A 20

1. Attempt all parts:-

1-a. In JAVA main method returns value of type _____ [CO1] 1

1. float
2. int
3. void
4. String

1-b. What is bytecode in Java? [CO1] 1

1. Code generated by a Java compiler
2. Code generated by a Java Virtual Machine
3. Name of Java source code file
4. Block of code written inside a class

1-c. If same message is passed to objects of several different classes and all of those can respond in a different way, what is this feature called? [CO2] 1

1. Inheritance
2. Overloading
3. Polymorphism
4. Overriding

1-d. Java _____ is invoked at the time of object creation. [CO2] 1

1. constructor
2. class
3. method
4. array

1-e. Which of these keywords must be used to monitor for exceptions? [CO3] 1

1. try
2. catch
3. throw
4. finally

1-f. An _____ statement can be used to access the classes and interface of a different package from the current package. [CO3] 1

1. instanceOf
2. import
3. extends
4. implement

Old University Question Paper

1-e. Which of these keywords must be used to monitor for exceptions? [CO3] 1

1. try
2. catch
3. throw
4. finally

1-f. An _____ statement can be used to access the classes and interface of a different package from the current package. [CO3] 1

1. instanceOf
2. import
3. extends
4. implement

1-g. The keyword that is used to protect the methods from simultaneous access in Threads is _____ [CO4] 1

1. save
2. synchronized
3. Both
4. This task is not possible in threads

1-h. Which of these classes are used by Byte streams for input and output operation? [CO4] 1

1. Input Stream
2. InputOutputStream
3. Reader
4. All of the mentioned

1-i. A _____ dictates the style of arranging the components in a container. [CO5] 1

1. border layout
2. grid layout
3. panel
4. layout manager

1-j. _____ interface provides the capability to store objects using a key-value pair. [CO5] 1

1. Java.util.Map
2. Java.util.Set
3. Java.util.List
4. Java.util.Collection

Old University Question Paper

2. Attempt all parts:-

2-a.	What is JVM? [CO1]	2
2-b.	What is the use of final keyword in JAVA? [CO2]	2
2-c.	What is an assertion in Java? How is it different from if - else conditions? [CO3]	2
2-d.	Describe any two Annotations from the Java Standard Library. [CO4]	2
2-e.	What Are Wrapper Classes? Why do we need wrapper classes in JAVA? [CO5]	2

Old University Question Paper

SECTION B 30

3. Answer any five of the following:-

3-a.	What is the difference between object diagrams and class diagrams? Draw a class diagram of order management system. [CO1]	6
3-b.	How to take an input from a user with the help of scanner class in JAVA? Explain using JAVA code. [CO1]	6
3-c.	Explain Abstract class concept with an example program. [CO2]	6
3-d.	Compare overloading and overriding of methods in java using proper examples. [CO2]	6
3-e.	Write a method to check if input string is Palindrome? [CO3]	6
3-f.	Explain the concept of multithreading in java and explain how even and odd numbers can be printed by using multithreading concept. (CO4)	6
3-g.	Examine ArrayList with Example. [CO5]	6

Old University Question Paper

SECTION C 50

4. Answer any one of the following:-

4-a.	What are command line arguments? How are they useful? Write a program to compute the sum of the digits of an input number (Using command line arguments) eg if 4523 is an integer then the sum of digits displayed will be 14. [CO1]	10
4-b.	Write a JAVA program that takes values of name, age, department and marks of 4 subjects from the user. Display the name, total and average of marks computed. [CO1]	10

5. Answer any one of the following:-

5	Explain the following with respect to JAVA: [CO2] a) super keyword b) Garbage collection c) Interface d) Static data members e) final keyword	10
5	What is the lambda expression in Java and what are the features of a lambda expression? Briefly explain its use with the help of suitable example. [CO2]	10

Old University Question Paper

6. Answer any one of the following:-

6	Write the differences between String, StringBuffer and StringBuilder classes. With proper syntax, explain the following methods. [CO3] 1. Method to extract a particular character of a string. 2. Reverse a String.	10
6	What is the difference between an error and exception? Write the following Java program for illustrating the use of throw keyword. Write a class ThrowExample contains a method checkEligibilty(int age, int weight) which throw an ArithmeticException with a message "Student is not eligible for registration" when age < 12 and weight < 40, otherwise it prints "Student Entry is Valid!!". [CO3]	10

7. Answer any one of the following:-

7-a.	What is the difference between thread and a process? Explain the concept of Inter Thread Communication and describe the role of wait(), notify(), and notifyAll() methods in inter thread communication. [CO4]	10

Old University Question Paper

7-b.	While reading a file, how would you check whether you have reached to the end of file? Write a JAVA program to copy the content of "file1.txt" to "file2.txt". [CO4]	10

8. Answer any one of the following:-

8-a.	Discuss some general rules for using layout managers. Describe the various layout managers available in AWT. [CO5]	10
8-b.	Differentiate between List and ArrayList. Create a class TestArrayList having main method. Perform following functionality. [CO5]	10

1. Create an ArrayList having fruits name of type String.
2. Store different fruit names. (Try to add duplicate fruit names).
3. Print all fruit names.
4. Print the first and last fruit names.
5. Print the size of ArrayList.
6. Remove a particular fruit from ArrayList.

Expected Questions

1. Discuss how to set the priority to threads? What are the different ranges. [CO4]
2. Write a java program to create two threads and execute simultaneously. [CO4]
3. Write a Java program that creates three threads. First thread displays “Hello!” every one second, the second thread displays “Wear Mask !” every two seconds and “Use Sanitizer !” every 5 seconds. [CO4]
4. Write the difference between Extending thread and implementing runnable? [CO4]
5. Explain in detail about thread methods? [CO4]

Old Question Papers

- https://www.iare.ac.in/sites/default/files/IARE_JAVA_MODEL_QP.pdf
- https://www.manaresults.co.in/jntuh/download.php?subcode=133BM

Recap of Unit

- A package is a collection of similar types of Java entities such as classes, interfaces, subclasses, exceptions, errors, and enum.
- In this Unit, we have studied Java exceptions, it's types, and the difference between checked and unchecked exceptions.
- Exception Handling is a mechanism to handle runtime errors such as Class Not Found Exception, IO Exception, SQL Exception, Remote Exception, etc.
- An assertion allows testing the correctness of any assumptions that have been made in the program.
- **String Handling in java**

References

Text Books:
(1) Herbert Schildt," Java - The Complete Reference", McGraw Hill Education 12th edition
(2) Herbert Schildt," Java: A Beginner's Guide", McGraw-Hill Education 2nd edition
(3) James Rumbaugh et. al, "Object Oriented Modeling and Design", PHI 2nd Edition
Reference Books:
(4) Cay S. Horstmann, "Core Java Volume I – Fundamentals", Prentice Hall
(5) Joshua Bloch," Effective Java", Addison Wesley
(6) E Balagurusamy, "Programming with Java A Primer", TMH, 4th edition.

Thank You