3 Exception Handling and Threading and Streams(Input & Output)

Topics Covered

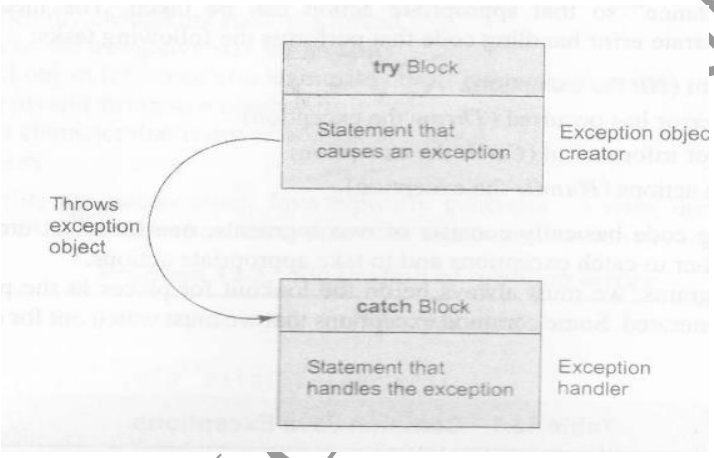
1. introduction to exception handling
2. keywords
   - try
   - catch
   - throw
   - throws
   - finally
3. doing inheritance
4. constructor in inheritance
5. method overriding
6. interface
7. nested and inner class
8. abstract and final class
9. Stream and its types
   - Input
   - Output
   - Character
   - Byte
10. File and RandomAccessFile Class
11. Reading and Writing through Character Stream Classes
    - FileReader
    - BufferedReader
    - FileWriter
    - BufferedWriter)
12. Reading and Writing through Byte Stream Classes
    - InputStream
    - FileInputStream
    - BufferedInputStream
    - DataInputStream
    - OutputStream
    - FileOutputStream
    - BufferedOutputStream
    - DataOutputStream
13. StreamTokenizer Class

Introduction to exception handling

- An exception is a condition that is caused by a run-time error in the program.
- When the java interpreter encounters an error such as dividing an integer by zero , it creates an exception object and throws it (i.e. informs us that an error has occurred).
- If we want the program to continue with the execution of the remaining code , then we should try to catch the exception object thrown by the error condition and then display an appropriate message for taking corrective actions. This known as exception handling.
- The exception handling mechanism has the following tasks :
    - Find the problem (Hit the exception).
    - Inform that an error has occurred (Throw the exception).
    - Receive the error information (Catch the exception).
    - Take corrective actions (Handle the exception).

Syntax of exception handling code

- The basic concepts of exception handling are thrown an exception and catching it.
- This is shown in following fig.
-



- Java uses a keyword try to preface a block of code that is likely to cause an error condition and "throw"' an exception.
- A catch block defined by the keyword catch "catches" the exception ,"thrown" by the try block and handles it appropriately.
- The catch block is added immediately after the try block.
- The following example illustrates the use of simple try and catch statements:

```
……..
……
try
{
    Statement;      // generates an exception
}
Catch (Exception-type e)
{
    Statement;      // processes the exception
}
```

- The try block can have one or more statements that could generate an exception.

Shri M.M.Ghodasara Mahila College

- If anyone statement generates an exception, the remaining statements in the block are skipped and execution jumps to catch block that is placed next to the try block.
- The catch block too can have one or more statements that are necessary to process the exception.
- Every try statement should be followed by at least one catch statement , otherwise compilation error will occur.
- The catch statement works like a method definition.
- The catch statement is passed a single parameter , which is reference to the exception object thrown by try block.
- If the catch parameter matches with the type of exception object, then the exception is caught and statements in the catch block will be executed. Otherwise , the exception is not caught and the default exception handler will cause the execution to terminate.

try statement

- The try block can have one or more statements that could generate an exception.
- If anyone statement generates an exception, the remaining statements in the block are skipped and execution jumps to catch block that is placed next to the try block.

Catch statement

- The catch statement works like a method definition.
- The catch statement is passed a single parameter , which is reference to the exception object thrown by try block.
- If the catch parameter matches with the type of exception object, then the exception is caught and statements in the catch block will be executed. Otherwise , the exception is not caught and the default exception handler will cause the execution to terminate.

finally statement

- Java supports another statement known as flnally statement that can be used to handle an exception that is not caught by any of the previous catch statements.
- finally block can be used to handle any exception generated within a try block.
- It may be added immediately after the try block or after the last catch block shown as follows:

```
try
{
        ……
        …….
}
finally
{
```

Shri M.M.Ghodasara Mahila College

```
        ……
        …….
}
Or another way….
try
{
        ……
        …….
}
catch(…….)
{
        ……
}
catch(…….)
{
        ……
}
finally
{
        ……
        …….
}
```

- When a finally block is defined, this is guaranteed to execute, regardless of whether or not in exception is thrown it means statements written in finally block are executed whether the exception is caught or not.
- As a result, we can use it to perform certain house-keeping operations such as closing files and releasing system resources.

Throw keyword

- We can throw an exception manually by the throw keyword.
- The syntax is
  Throw obj
- Object is the throwable class.
- You can create object of any throwable class or subclass using new keyword.

Example

```
Throw new ArithmeticException("divide by zero");
```

Throws keyword

- It is used to specify that a method can throw an exception that it does not handle.
- You can do this by throws keyword.
- After the throws keyword specify the exception class that it can throw.
- There are two types of exception in java.
  - Checked exception
  - Unchecked exception
- **Checked exception:** These exception are explicitly handled in the code itself with the help of try-catch blocked. Checked exceptions are extended from the **java.lang.Exception** class.

Shri M.M.Ghodasara Mahila College

- ClassNotFoundException :caused when class location is not used properly.
- NoSuchMethodException : caused when such method does not exist.
- IllegalAccessException : caused when try to attempt illegal operation.

- **Unchecked exception:** These exceptions are not essentially handled in the program code; instead the JVM handled such exceptions. Unchecked exceptions are extended from the **java.lang.RuntimeException** class.
  - ArithmeticException : caused by math errors such as division by zero.
  - ArrayIndexOutOfBoundsException : caused by bad array index.
  - NullPointerException : caused by referencing a null object.
  - NumberFormatException : caused when a conversion between strings and number fails.

Create user defined exception class

- We can create our own exception class by extending the exception class.
- After extending the exception class, we can override the methods of Throwable class which is the super class of exception class.
- Following are the methods of throwable class.

Method	Description
String toString()	It return the string description of the exception.
String getMessage()	It returns the description of the exception passed in the constructor.

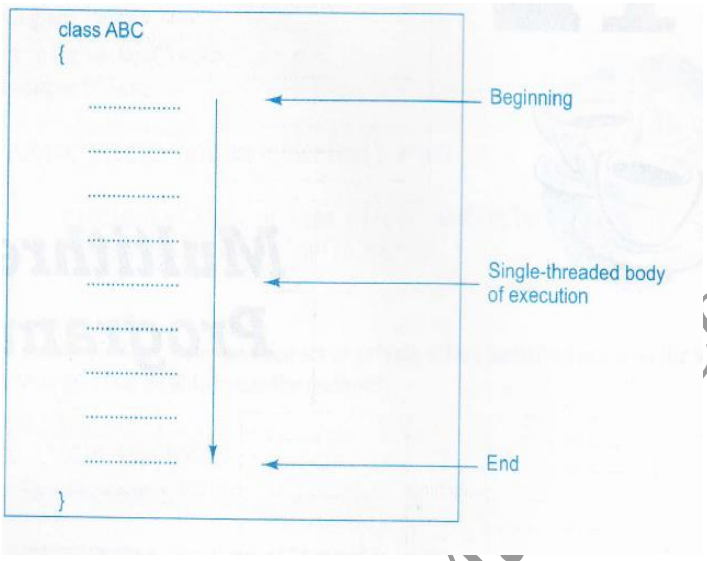
Thread

- Modern operating system such as Windows 95 and windows xp may recognize that they can execute several programs simultaneously. This ability is known as multitasking. In system's terminology, it is called multithreading.
- Multithreading is where a program (process) is divided into two or more subprograms (processes), which can be implemented at the same time in parallel.
- For example, one subprogram can display an animation on the screen while another may build the next animation to be displayed.
- A thread is similar to a program that has a single flow of control. It has a beginning, a body, an end and executes commands sequentially as shown in fig. 1.
- A unique property of Java is its support for multithreading. That is, Java enables us to use multiple flows of control in developing programs. Each flow of control may be thought of

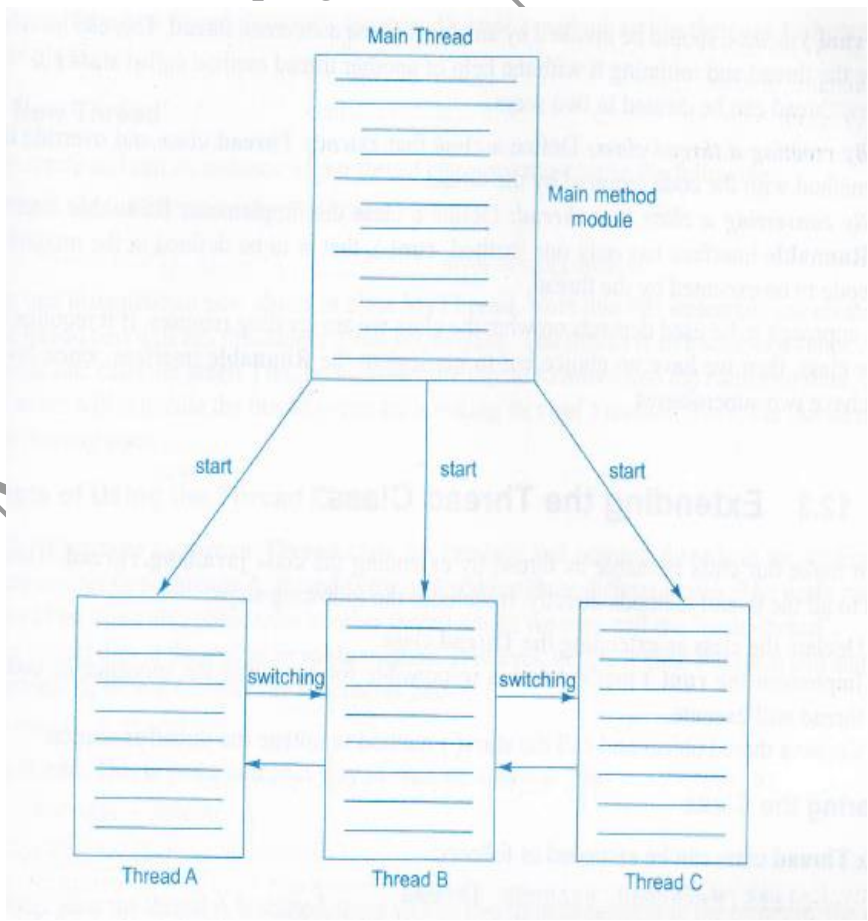
Shri M.M.Ghodasara Mahila College

as a separate tiny program (or module) known as a thread that runs in parallel to others as shown in Fig. 2.

- **Fig. 1. single threaded program**



- **Fig. 2. Multithreaded program**



- A program that contains multiple flows of control is known as multithreaded program.

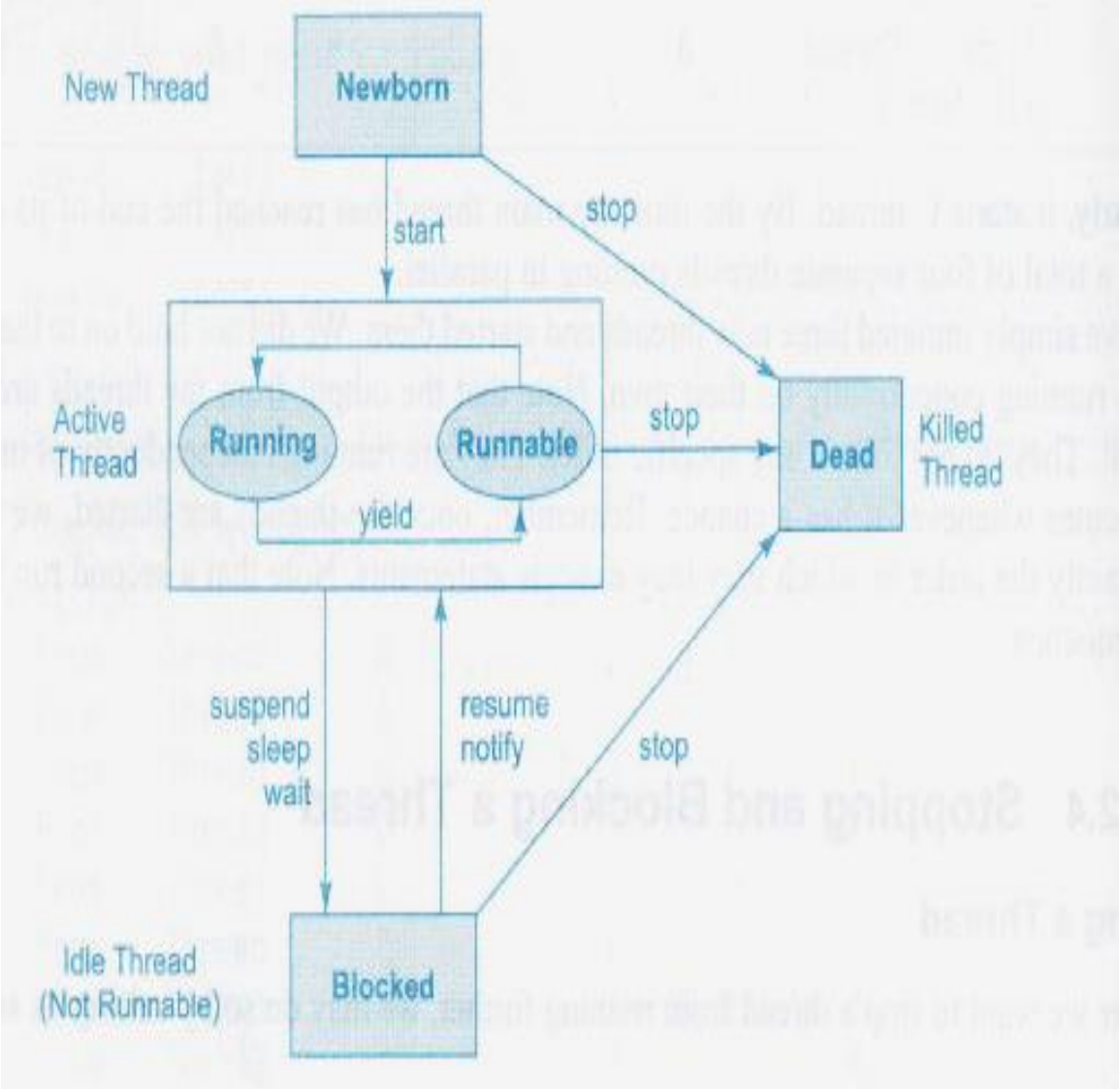
- fig. 2 illustrates a Java program with four threads, one main and three others. The main thread is actually the main method module, which is designed to create and start the other three threads, namely A, B and C.
- It is like people living in joint families and sharing certain resources among all of them.
- It is important to remember that 'threads running in parallel' does not really mean that they actually run at the same time. Since all the threads are running on a single processor , the flow of execution is shared between the threads. The java interpreter handles the switching of control between the threads in such a way that it appears they are running concurrently.
- Multithreading is useful in a number of ways.
- It enables programmers to do multiple things at one time.
- Any application we working on that requires two or more things to be done at the same time is probably a best one for use of threads.

Thread Life Cycle

- Every thread has a life cycle.
- During the life time of thread , there are many states it can enter.
    - Newborn state
    - Runnable state
    - Running state
    - Blocked state
    - Dead state
- A thread is always in one of these five states. It can move from one state to another via a variety of ways as shown in following figure.

Fig. a-1 State transition diagram of thread

Shri M.M.Ghodasara Mahila College



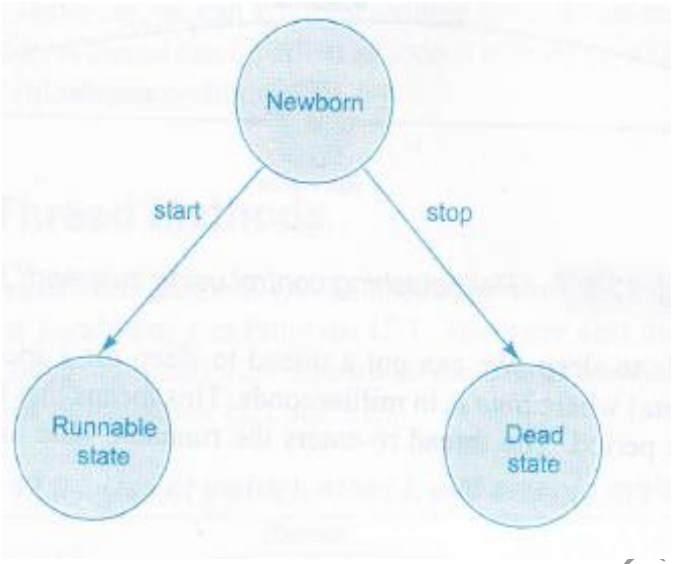
1) Newborn state

- When we create a thread object, the thread is born and is said to be in newborn state.
- The thread is not yet scheduled for running.
- At this state, we can do only one of the following things with it:
    - Schedule it for running using start( ) method.
    - Kill it using stop() method.

- If scheduled , it moves to the runnable state.
- If we attempt to use any other method at this stage, an exception will be thrown.

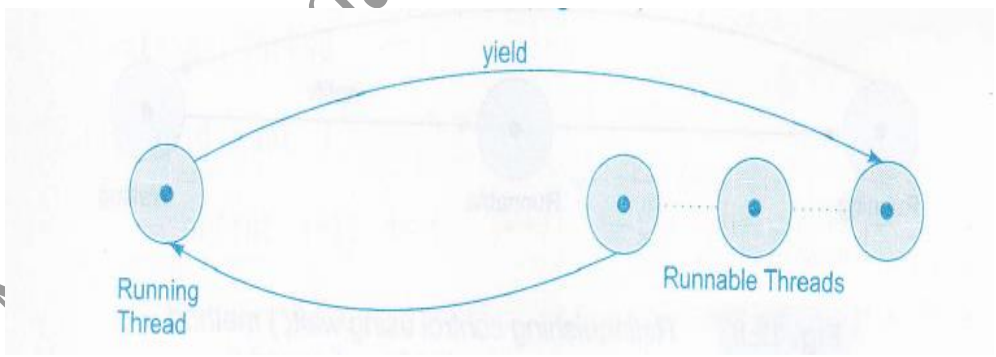
Scheduling a newborn thread

Shri M.M.Ghodasara



2) Runnable state

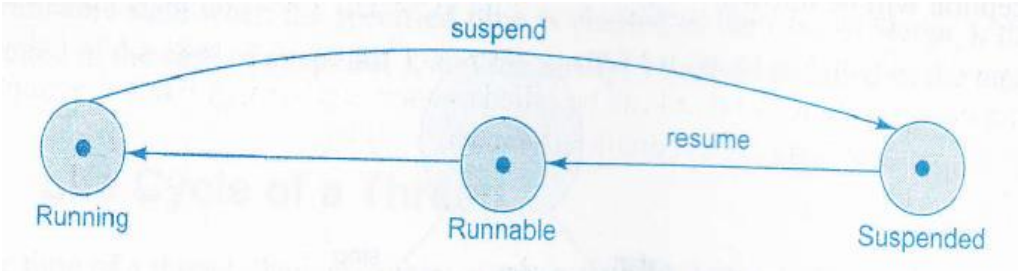
- The runnable state means that the thread is ready for execution and is waiting for the availability of the processor.
- That is , the thread has joined the queue of threads that are waiting for execution.
- If all threads have equal priority, then they are given time slots for execution in round robin fashion , i.e. first-come , first-serve manner.
- The thread that relinquish control joins the queue at the end and again waits for its turn.
- This process of assigning time to threads is known as time-slicing.
- However , if we want a thread to relinquish control to another thread to equal priority before its turn comes , we can do so by using the yield() method as shown in following figure.



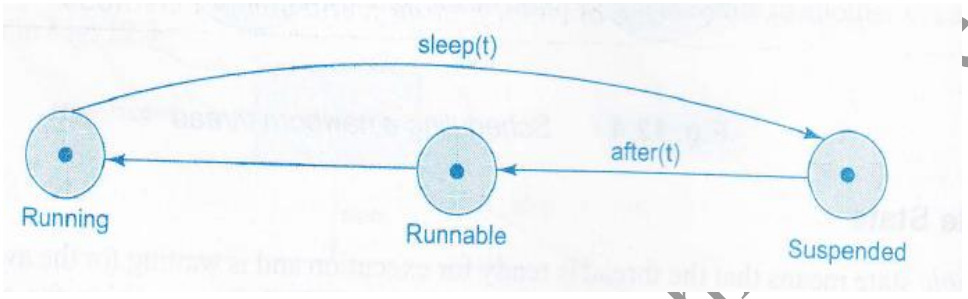
3) Running state

- Running means that the processor has given its time to the thread for its execution.
- The thread runs until it relinquishes control on its own or it is preempted by a higher priority thread.
- A running thread may relinquish its control in one of the following situations.
- It has been suspended using suspend() method. A suspended thread can be revived by using the resume() method. This approach is useful when we want to suspend a thread for some time due to certain reason , but do not want to kill it.

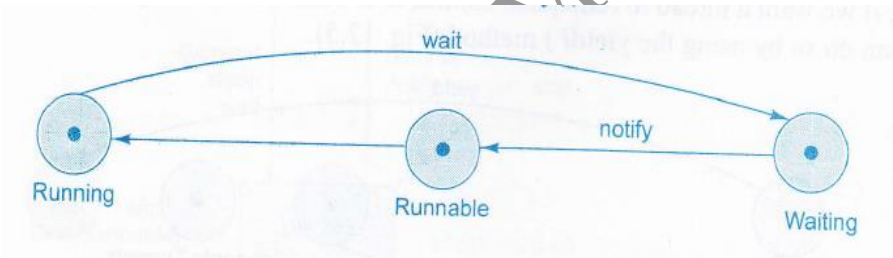
Shri M.M.Ghodasara Mahila College



- It has been made to sleep. We can put a thread to sleep for a specified time period using the method sleep(time) where time is in milliseconds. This means that the thread is out of the queue during this time period. The thread re-enters the runnable state as soon as this time period is elapsed.



- It has been told to wait until some event occurs. This is done using the wait() method. The thread can be scheduled to run again using the notify() method.



4) Blocked state

- A thread is said to be blocked when it is prevented from entering into the runnable state and subsequently the running state. This happens when the thread is suspended, sleeping or waiting in order to satisfy certain requirements.
- A blocked thread is considered "not runnable" but not dead and therefore fully qualified to run again.

5) Dead state

- Every thread has a life cycle.
- A running thread ends its life when it has completed executing its run() method.
- It is a natural death.
- However, we can kill it by sending the stop message to it at any state thus causing a premature death to it.
- A thread can be killed as soon it is born , or while it is running , or even when it is in " not runnable " (blocked) condition.

Thread class and it's methods

- **A new thread can be created in two ways.**
  1. By creating a thread class.
  2. By implementing runnable interface.

1. By creating a thread class

- It means by extending thread class.
- Define a class that extends thread class and override its run() method with the code required by the thread.
- It includes the following steps :
  - Declare the class as extending the Thread class.
  - Implement the run() method that is responsible for executing the sequence of code that the thread will execute.
  - Create a thread object and call the start() method to initiate the thread execution.

Declaring the class

- The Thread class can be extended as follows :

```
class MyThread extends Thread
{
        …….
        …….
}
```

- Now we have new type of thread MyThread.

Implementing run() method

- The run() method has been inherited by the class MyThread.
- We have to override this method in order to implement the code to be executed by our thread.
- The basic implementation of run() will look like this :

```
public void run()
{
        ……. // thread code here
}
```

- When we start the new thread , java calls the thread's run() method , so it is the run() where all the action takes place.

Starting new thread

- To actually create and run an instance of our thread class , we must write the following :

```
MyThread aThread=new MyThread();
Thread.start();    // invokes run() method
```

- The first line instantiates a new object of class MyThread.
- This statement just creates the object.
- The thread that will run this object is not yet running. The thread is in a newborn state.

Starting new thread

- The second line calls the start() method causing the thread to move into the runnable state.
- Now, thread is said to be in the running state.

2. By implementing runnable interface

Shri M.M.Ghodasara Mahila College

- The Runnable interface declares the run() method that is required for implementing threads in our program. To do this , we must perform the steps listed below :
    - Declare the class as implementing the Runnable interface.
    - Implement the run method.
    - Create a thread by defining an object that is instantiated from this " runnable " class as the target of the thread.
    - Call the thread's start method to run the thread.
- To implement Runnable, a class need only implement a single method called run( ), which is look like this:

```
public void run( )
 {
                .......
                ......
}
```

Synchronization

- When more than one threads attempt to use a shared resource , that is known as "race condition " .
- In the programming term , " a shared resource can be a method or an object " .
- This race condition must be avoided because when two or more threads access a shared resource such as a file , one may read the file while the other is writing in the file. This may lead to unexpected results. Therefore , synchronization is necessary.
- The synchronization ensures that only one thread will access a shared resource at a time.
- When a thread enters a shared resource it will lock that resource and after the completion of its execution the resource will be unlocked.
- The synchronization is done by the synchronized keyword.
- For example , the method that will read information from a file and the method that will update the same file may be declared as synchronized.

Example :

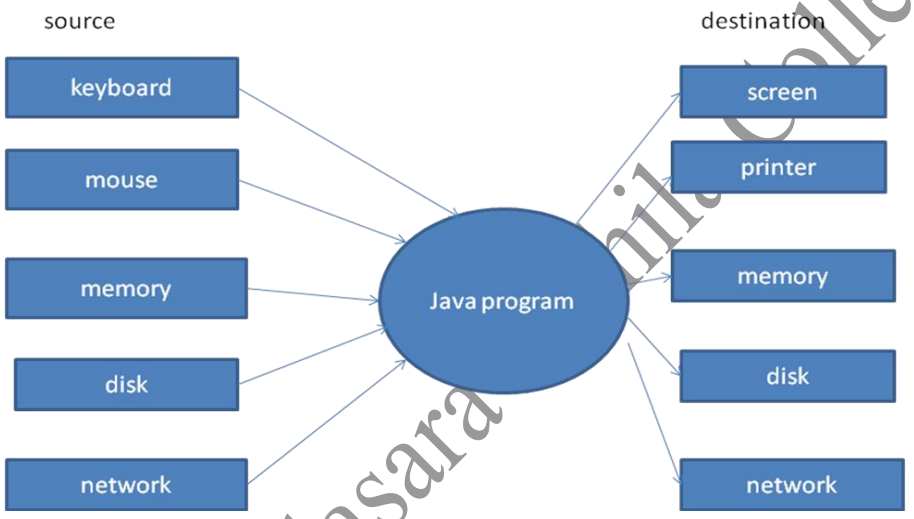
```
synchronized void update()
{
        ........
        ..........
}
```

- When we declare a method synchronized , java creates a " monitor " and hands it over to thread that calls the method first time.
- As long as the thread holds the monitor , no other thread can enter the synchronized section of code.
- A monitor is like a key and the thread that holds the key can only open the lock.
- Whenever a thread has completed its work of using synchronized method , it will hand over the monitor to the next thread that is ready to use the same resource.

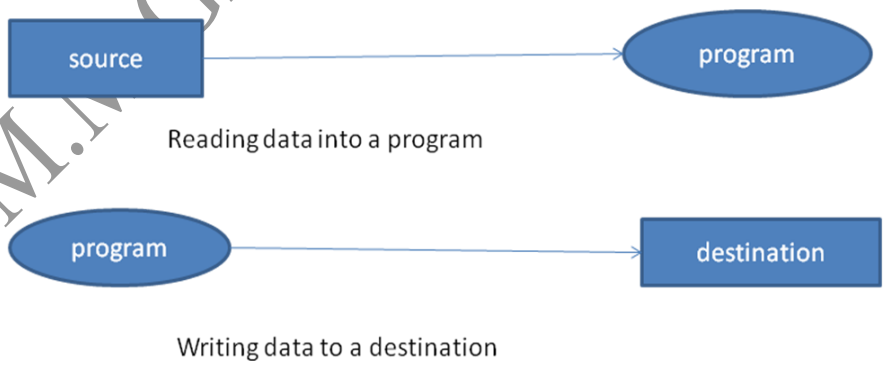
Shri M.M.Ghodasara Mahila College

Stream and its type

- 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 a program may come from the keyboard , 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.
- Java uses the concept of streams to represent the ordered sequence of data , a common characteristic shared by all the Input/output devices as stated above.
- The concept of sending data from one stream to another (like one pipe feeding into another pipe) has made stream in java a powerful tool for processing.



- We can build a complex file processing sequence using a series of simple stream operations.
- This features can be used to filter data along the pipeline of streams so that we obtain data in a desired format.



- Java streams are classified into two basic types, namely , input stream and output stream.
- An input stream extracts (i.e.reads) data from the source (file) and sends in to the program.
- Similarly, an output stream takes data from the program and sends(i.e. writes) it to the destination file.

Character streams

- It allows you to read and write characters and strings.

Shri M.M.Ghodasara Mahila College

- An input character streams converts bytes to characters.
- An output character stream converts character to byte.

Byte streams

- It allows you to read and write binary data.
- In this data are accessed as a sequence of bytes.

Reader

- These classes are used to read characters from the file.
- The reader class defines the functionality that is available for all characters input stream.
- Reader is an abstract class so, we can not create an instance of this class.
- We can use the subclass buffered reader and input stream reader of Reader class.

InputStreamReader

- It is a subclass of Reader class.
- It converts a stream of bytes to a stream of characters.
- The constructors are as follow:
    - InputStreamReader(InputStream is)
    - InputStreamReader(InputStream is,String encoding)

FileReader

- it is a subclass of InputStreamReader class and input characters from a file.
    - FileReader(String filepath)
    - FileReader(File object)

Writer

- These classes are used to perform all output operations on files.
- The writer class is an abstract which acts as a base class for all the other writer stream class.
- The constructor are as follow:
    - Writer()
    - Writer(Object obj)

OutputStreamWriter

- The OutputStreamWriter class is a sub class of writer class.
- It converts a stream of characters to a stream of bytes.
- This is done according to the rules of a specific character encoding.
- Its constructor are as follow:-
    - OutputStreamWriter (OuputStream os)
    - OutputStreamWriter (OutputStream os , String encoding)

FileWriter

- It is a subclass of OutputStreamWriter class and outputs characters to a file.
- Its constructors are as Follow:-

Shri M.M.Ghodasara Mahila College

- FileWriter (String Filepath) throws IOException
- FileWriter (string filepath , boolean append) throws IOException
- FileWriter (File Object) throws IOException

PrintWriter

- The PrintWriter is a subclass of Writer class and display string equipments of simple types such as int, float, char and object.
- It provides the formatted output methods print () and println().
- PrintWriter has four constructors:
  - PrintWriter (OutputStream os)
  - PrintWriter (OutputStream os,boolean flushOnNewline)
  - Printwriter (Writer OutputStream)
  - Printwriter (Writer os,boolean flushOnNewline)
- Here, flushOnNewline controls whether java flushes the Output stream everytime println() is called.
- If flushOnNewline is true,flushing automatically takes place.If flase,Flushing is not automatic.
- The first and third constructors do not automatically flush.
- Java's printwriter object support the print() and Println() methods for all types, including object.
- If an argument is not a simple type, the Printwriter methods will call the object's toString () method and then print the result.

BufferedWriter

- The BufferedWriter class is a subclass of writer class and buffers output to a character stream.
- Its constructors are as follows:
  - BufferedWriter(Writer w)
  - BufferedWriter(Writer w,int buf size)
  - The first From Creates a buffered stream using a buffer with a default size.
  - In second ,the size of the buffer is specified by bufsize.

BufferedReader

- It is a Subclass of Reader class and buffers input from a charater stream.
- Its constructors are as follows:
  - BufferedReader (reader r)
  - Bufferedreader (Reader r ,int bufsize)
- The first from createds a buffered stream using a buffer with a default size
- In second,the size of the buffer is specified by butsize.

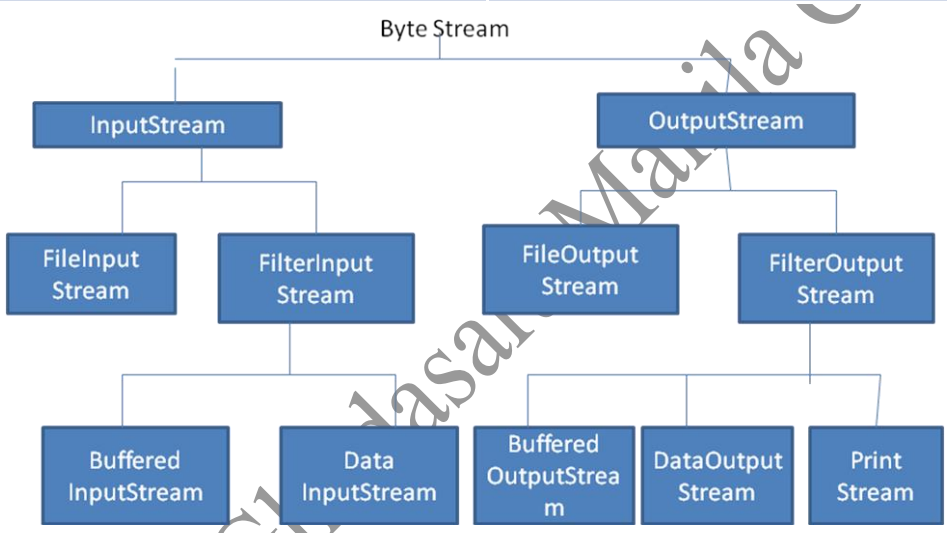
Byte Streams

- Byte streams allow a programmer to work with the binary data in a file.
- In this stream data are accessed as a sequence of bytes.
- All types other than text or character are dealt in this stream.

Shri M.M.Ghodasara Mahila College

- Byte stream includes OutputStream, FileOutputStream, DataOutputStream and PrintStream classes that are used for output, while the inputstream, FileInputStream, FilterInputStream, BufferedInputStream are use for input.
- The following table summerize the methods provides by this class.

Methods	Desccription
Void close()throws IOException	Close the output stream.
Void flush()throws IOException	Flushes the output stream
Void write(int i)throws IOException	Writes lowest order 8 bits of stream
Void write(byte buffer[])throws IOException	Writes buffer to the stream.
Void write(byte buffer[],int index,int size)throws IOException	Writes the size bytes from buffer starting at position index to the stream,



InputStream

- This class defines the functionality that is available for all byte input streams.

Int available()	Returns the number of bytes currently available for reading.
Void close()	Closes the input stream.
Void mark(int numbytes)	Places a mark at the current point in the input stream. it remains until numbytes are read.
Boolean markSupported()	Returns true if mark()/resets() are supported otherwise returns false.
Int read()	Reads one byte from the input stream.
Int read(byte buffer[])	Attempts to read up to buffer length bytes into buffer and returns the actual number of bytes that were successfully read.
Int read(byte buffer[],int offset,int numBytes)	Attempts to read up to buffer starting at buffer. It returns the number of bytes successfully read.
Void reset()	Resets the input printer to the previously set mark.
Int skip(long numBytes)	Skips numBytes of input.Returns the number

Shri M.M.Ghodasara Mahila C

	of bytes actually skipped.

FileInputStream

- It is a subclass of InputStream class and allows you to read binary data from a file.
- its constructor are as follow:
  - FileInputStream(String filepath)throws FileNotFoundException
  - FileInputStream(File object)throws FileNotFoundException
- Here, filepath is full path name of a file and object is a file object that describes the file.

FilterInputStream

- It is a subclass of InputStream class and filters an input stream.
- It is an abstract class so you can create an instance of this class but you can create a subclass to implemented to desired functionality.
- It provides this constructor:
  - FilterInputStream(InputStream is)

BufferedInputStream

- It is a subclass of FilterInputStream class and buffers input from a byte stream.
- Its constructor are as follow:
  - BufferdInputStream(InputStream is)
  - BufferdInputStream(InputStream is,int bufsize)
  - The first argument to both constructors is a referance to the input stream.
- The first form creates a buffered stream by using a buffer with a default size.
- In second , the size is specified by bufsize.

DataInputStream

- The DataInputStream is a subclass of FilterInputStream class and implements DataInput.
- It allows you to read the simple java types from a ByteInputStream.
- Its constructor is as follow:
  - DataInputStream(InputStream is)

DataInput

- The DataInput interface defines methods that can be used to read the simple java types from a byte inputstream.

OutputStream

- This class defines the functionality that is available for all byte OutputStream

FileOutputStream

- It is a subclass of OutputStream and allows you to write binary data to a file.
- Its constructor are as follow:
  - FileOutputStream(String filepath)throws IOException

Shri M.M.Ghodasara Mahila College

- FileOutputStream(String filepath,boolean append)throws IOException
- FileOutputStream(File Object)throws IOException

FilterOutputStream

- This class is a subclass of OutputStream class.
- It is an abstract class so you can not create an instance of this class.
- Its constructor is as follow:
  - FilterOutputStream(OutputStream os)

BufferedOutputStream

- it is a subclass of FilterOutputStream class and buffers output to byte stream.
- Its constructors are as follow:
  - BufferedOutputStream(OutputStream os)
  - BufferedOutputStream(OutputStream os,int bufsize)

DataOutputStream

- It is subclass of FilterOutputStream class and implements DataOutput
- It allows you to write the simple java types to a byte OutputStrem.
- Its constructor is as follow:
  - DataOutputStream(OutputStream os)

DataOutput interface

- It defines the methods that can be used to write the simple java types to abyte output stream.

PrintStream

- It is a subclass of FilterOutputStream class and provides all of the formatting capabilities.
- The static System.out variable is a printstream.
- It has two constructors.
  - Printstream(OutputStream os)
  - PrintStream(OutputSTream os,boolean flushonnewline)

StreamTokenizer class

- This class differentiate the data froma character input stream and generates a sequence of tokens.
- A token is group of charcters that represents a number of word.
- The constructor of this class is:
  - StreamTokenizer(Reader r)

Shri M.M.Ghodasara Mahila College

Random Access Files

- This class has no superclass except for the super object.
- It allows you to write program that can seek to any location in a file and read or write data at that point.
    - RandomAccessFile(String file,String mode)
    - RandomAccessFile(file obj,String mode)

Here file is name of file and value of mode is “r” for read only and “rw” for read and write.

Shri M.M.Ghodasara Mahila College