(Java Programming - 22412)

Unit IV : Exception Handling and Multithreading

(weightage - 12 marks)

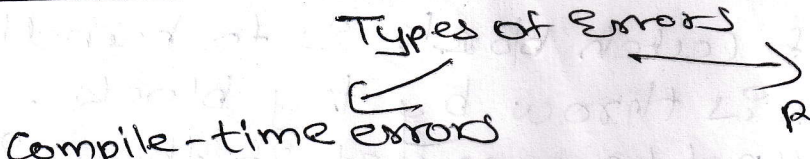
TRANJAL SAVE (JYRO)

4.1 Errors and Exception : Types of errors, exception, try catch statement, nested try statement, throw and Finally statement, built in exceptions, chained exceptions, creating own exception, subclasses.

Exception: An exception is unexpected/unwanted/abnormal condition/situation that occured at runtime called exception.

Exception Handling: In Exception Handing, we should have an alternate source through which we can handle the error.

Question : Enlist any four compile time errors. (2M)
Define error. List types of error. (2M)



Compile-time errors

All syntax errors will be detected and displayed by java compiler and therefore these errors are known as compile time errors.

- Missing semicolon - Missing double quotes - Use of undeclared variable - Misspelling of keywords.

Run-time errors

The error which are generated while program is running are known as Run-time errors.

- Dividing integer by zero
- Converting invalid string to integer.
- Opening file which does not exist.

Question
Explain exception handling mechanism w.r.t try, catch, throw and finally. (4M)

Java provides some mechanism to work with exception

Java handles exception with 5 keywords.
1) try 2) catch 3) Finally 4) Threw 5) Throws.

Try

whenever we write a statement and if statement is error suspecting statement or risky code then put that code inside try block.

This block applies a monitor on the statements written inside it. If there is any exception, the control is transferred to catch or finally block.

Syntax:
```
try
{
// block of code
}
```
contains risky code

Catch

The main purpose of catch block is to handle the exception which is throw by try block.

Catch block will not be executed if there is no exception inside the try block

This block includes the action to be taken if particular exception occurs.

Syntax:
```
catch (Exception type1 exObject)
{
//exception handler code
}
```
handle exception

finally

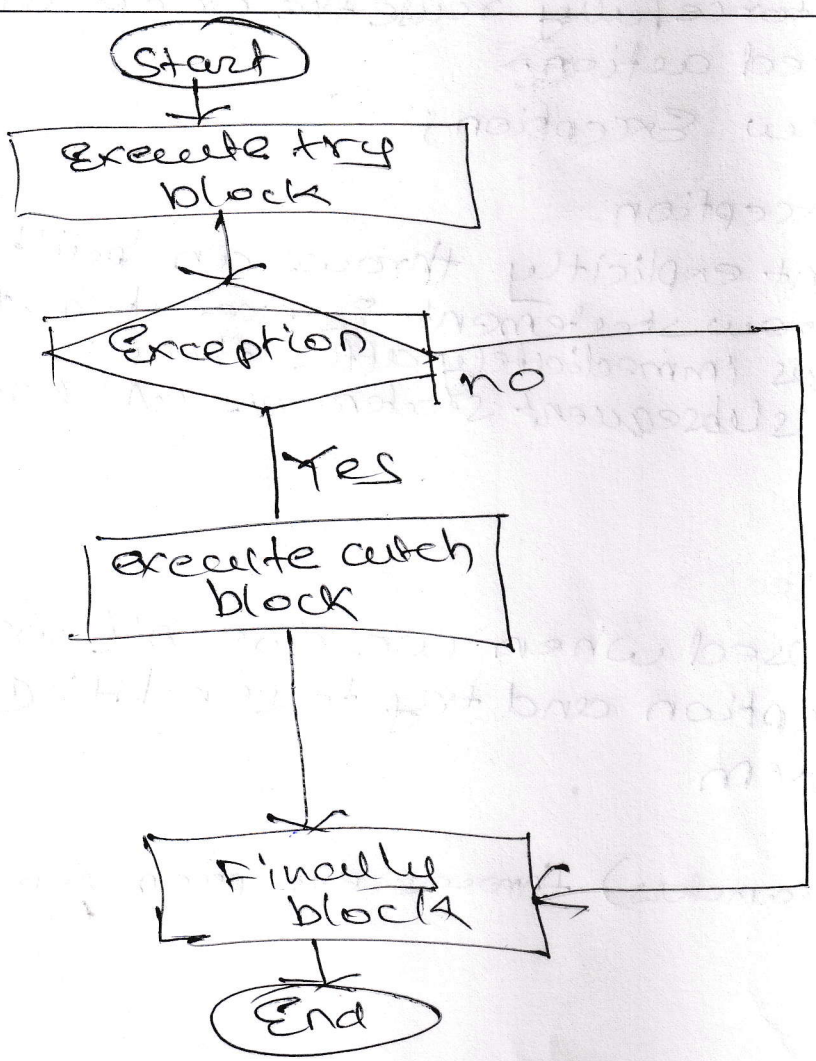
Finally block is reeltime block and the main purpose of finally block to handle the resources

finally block includes the statements which are to be executed in any case, in case the exception is raised or not.

Syntax:

```
finally
{
//block of code to executed before try block ends.
}
```

try-catch-Finally (Syntax-02 M)



Syntax

```
try {
    //code
}
catch (Exception E1)
{
    //code
}
catch (Exception E2)
{
    //code
}
finally
{
```

Throw

explicitly

Throw keyword is used to throw the user-defined or customized exception object to the JVM explicitily for that purpose we use throw keyword.

This is generally used in case of user defined exception, to forcefully raise the exception and take the required action.

Syntax: throw new Exception;
or
throw Exception

Throw statement explicitly throws an built in exception. when throw statement is executed, the flow of execution stops immediately after throw statement, and any subsequent statements are not executed.

Throws

by caller

Throws keyword is used when we doesn't want to handle the exception and try to send the exception to the JVM

Syntax:
Type method name (parameters) throws exception_name
{
//body
}

Example

```
class XYZ
{
public static void main(String args[]) throws
 InterruppedException
  {for(i=1; i<=10; i++)
   {System.out.println(i);
    Thread.sleep(1000); //1 sec
  }
}
```

program1

write a program to input name and salary of employee and throw user defined exception if entered salary is negative ..... (6M)

```
import java.util.Scanner;
import java.io.*;
class NegativeSalaryException extends Exception
  {
    public NegativeSalaryException(String m)
      {
       super(message)
      }
  }
class main2
  {
  public static void main(String args[])
    {
   Scanner sc = new Scanner(System.in);
System.out.println("Enter Employee name");
 String name = sc.nextLine();
 try{
System.out.println("Enter salary");
double salary = ~~scanner~~ sc.NextDouble();
      if(salary<0)
      {
    throw new NegativeSalaryException("salary
     cannot be Negative");
      }
  System.out.println("Details");
System.out.println("Name"+name);
System.out.println("Salary"+salary);
}
catch(NegativeSalaryException e)
  {
  System.out.println("Error:"+e.getMessage());
}}
```

program2

Define an exception called 'No match Exception' that is thrown when the password accepted is not equal to 'MSBTE' ; write the program. (6M)

```
import java.util.Scanner;
import java.io.*;
class NomatchException extends Exception
  {
   public NomatchException(String m)
     {
      super(message)
     }
  }
class My
  {
  public static void main(String args[])
    {
  String password;
  Scanner sc = new Scanner(System.in)
System.out.println("Enter Password");
password=sc.nextLine();
  try
    {
     if(password.compareTo("MSBTE")!=0)
     if(password.equals("MSBTE")
      {
    System.out.println("password matches");
      }
    else
      {
   throw new Nomatch Exception("Password does
     not match");
    }
  catch(NomatchException e)
   {
   System.out.println(e.getMessage());
```

program3

write a program that throws an exception called "Nomatchexception" when string is not equal to "India". (6marks)

```
import java.util.Scanner;
import java.io.*;
class Nomatchexception extends Exception.
  {
  public Nomatchexception(string m)
      {
      super(message);
      }
  }
class ABC
  {
  public static void main (string args[])
  {
  Scanner sc = new Scanner (System.in)
System.out.println ("Enter String : ");
 String s = sc.nextLine();
  try
  {
    if(s.equals ("India"))
      {
   System.out.println ("strings are equal");
      }
    else
      {
   throw new Nomatchexception (" String are unequal")
    }
  }
  catch(Nomatchexception e)
    {
  System.out.println(e.getmessage());
    }
  }
}
```

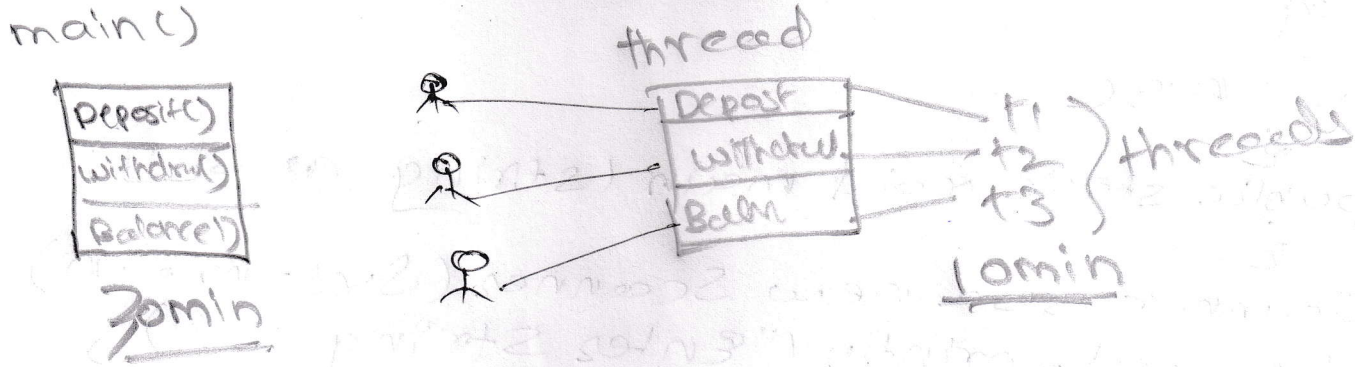
4.2 Multithreaded Programming Creating a Thread: By extending to thread class and implemeting runnable interface.

Thread methods: wait(), sleep(), notify(), resume(), suspend(), stop().

Thread exceptions, thread priority and methods, synchronization, inter-thread communication, deadlock

Multithreading Programming

- Multithreading is the process to execute multiple threads at the same time without dependency of other threads called multithreading.
- Because of multithreading Java is popular.



What is Thread?

⇒ Thread is a predefined class which is available in Java.lang package.

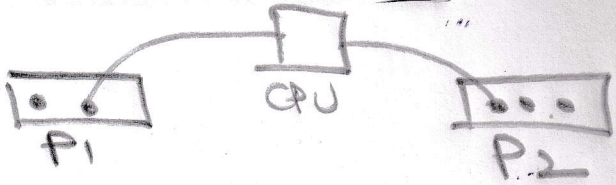
Thread is a Basic unit of CPU and it is well known for independent execution.

Thread is independent path of execution within a program. It shares memory area process.

There are two ways to create Thread.

1) By extending Thread class
2) By implementing Runnable interface.

Multitasking



Multithreading

multithreading - By extending Thread class in Java

Thread is an super class that is present in java.lang package of Java.

We can used Thread class by using extends keyword.

When we extend the class Thread, we need to implement the method run().

There run() method is overriden method.

Once we create an object, we can call the start() of the thread class for executing the method run().

As we placed start() the thread calls run() method and execution starts.

Syntax

```
class Thread_name extends Thread
    {
      public void run()
        {
          //code
        }
    }
class main
  {
    public static void main(String args[])
      {
        Thread_name object = new Thread_name();
        object.start();
      }
  }
```

Example

```
class ABC extends Thread
  {
    public void run()
      {
        for(i=1;i<5;i++)
          {
            System.out.println(i);
          }
      }
  }
```

```
class ABCD
{
public static void main(String a[])
  {
ABC a = new ABC();
a.start();
  }
}
```

Multithreading - by implementing Runnable interface

There is an interface called or named Runnable.
It has only one method in it called run()
by using "implements Runnable" we can perform multithreading by Runnable interface.
Runnable interface is included in package of java that is java.lang.
But to called start() we need to create a thread object. so it will give execution to run().

Syntax

```
Class ABC implements Runnable
 {
   public void run()
    {
      //code
    }
  public static void main (string args[])
   {
   ABC obj1= new ABC();
  Thread obj =new Thread(obj1)
   obj.start();
  }
}
```

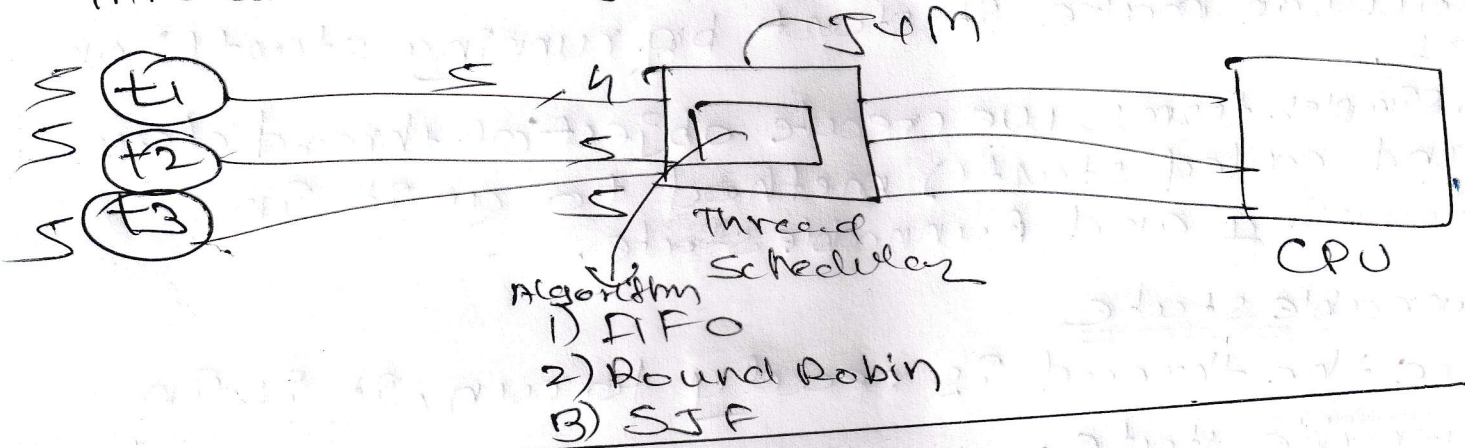
Example

```
Class Myinterface implements Runnable
 {public void run ()
  { for(i=1;i<=20;i++)
    {
     System.out.print(i);
    }
 }}
  class B
 {public static void main
  (string args[])
 {
```

```
Myinterface m=
 new Myinterface();
Thread t=new
Thread(m);
 t.start();
   }
}
```

Thread Schedeuler

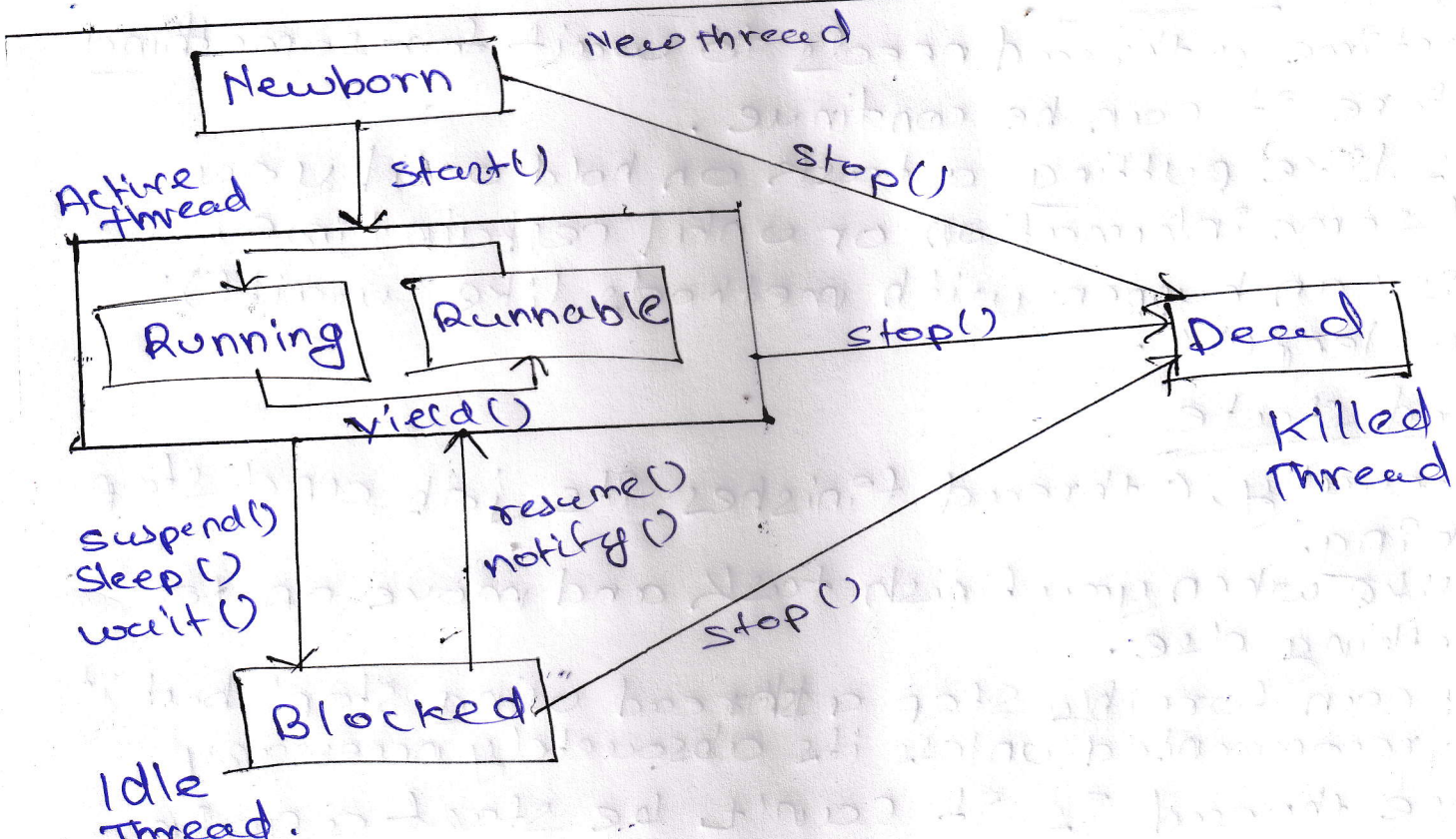
It is a part of JVM which execetes multiple thread on a single processor randomly.



Thread Life Cycle

As we all known a thread is well-known for independent execeetion.
During the life cycle a thread can move from different states.

1) Neeo state (Born)
2) Runnable state (Reeady)
3) Running state (Execetion)
4) waiting state (Blocked)
5) Deed State (Exit)



Newborn state

When thread is born created, it's not running.
It's like a baby waiting to be fed.
You can make it start by running start() on it.
In simpler terms, we create object of thread class and called start() method to go it in Running and Runnable state.

Runnable state

Once the thread is ready to run, it is in Runnable state.
It's like a person in a queue waiting for their turn.
When the processer is available (JVM), it will get its chance to run after calling start() method.

Running state

When the thread is actively using the processor, its in running state.
It's like someone doing activity or task when they are given.

Blocked state

Sometime a thread needs to wait for something before it can be continue.
It's like putting a task on hold until you get some information or until certain time.
This can happen with methods like 'wait()' or 'sleep()'.

Dead state

Eventually, a thread finishes its job and stop running.
It's like when you finish task and move on to something else.
You can forcibly stop a thread using 'stop' but it's not recommended unless its absolutely necessary.
Once thread is it can't be start again.

Thread methods

① start()

This method starts the execution of a thread.
It's like pressing button on video.

Syntax: thread.start();

② run()

This method contains the code that will be executed by the thread.

```
class xyz extends Thread
    {  public void ran()
        {
          //code
        }
    }
```

③ ~~sleep(ms)~~ join()

This method waits for thread to finish its execution before continuing.
It's like waiting for friend to finish the task before starting a new one together.

Syntax: thread.join();

④ sleep(ms)

This method paues the execution of the thread for specified amount of time.
It's like taking nap before continuing.

Syntax: Thread.sleep(milliseconds)

⑤ ~~$~~ yield()

This method suggest to schedular that current thread is willing to yield its current use of processor.
It's like saying, "I'm done will my task no others can go ahead".

Syntax: thread.yield()

⑥ resume()

This method is used to resume a thread that has been suspend(). pausing movie and starting again.

Syntax: thread.resume()

Question: Compare run() method and start() method. (2M)

start()	run()
Create a new thread and run() method is executed on the created newly thread.	No new thread is created and run() method is executed on calling thread itself.
Can't be invoked more than one time	multiple time invoked is possible
Java.lang.Thread class	Java.lang.Runnable
public void run()	public void run()
does not requires to call when thread.	It requires thread.

Question

Define thread priority? write default priority values and the methods to set and change them. (6M)

⇒ what is thread prority?

⇒ In Java it is possible to assign the priority of thread.

In Java each thread is assigned a priority which affects the order in which scheduled for running.

Thread of equal priority are given equal treatment of Java scheduler.

To set the priority java thread class has provide two predefined methods.

i) SetPriority() ii) GetPriority()

SetPriority(int newPriority)

The java.lang.Thread.getPriority() method updates or gives the priority of given thread (new).

GetPriority()

The java.lang.Thread.getPriority() method returns the priority of given thread.

Thread

* Thread class defines several priority :-

Thread.MIN_PRIORITY=1

NORM_PRIORITY=5

MAX_PRIORITY=10

* Maximum priority can be upto 1-10 only if exceed it gives exception IllegalArguementException.
* By default NORM_PRIORITY.

Example

```
import java.lang.*;
class A extends Threads
    {  public void run()
       {
System.out.println(Thread.currentThread().getName());
System.out.println(Thread.currentThread().getPrority());
       }
}
class B
  {
   public static void main (String arg[])
     {
```

```
A t1 = new A()
A t2 = new A()
A t3 = new A()

t1.setName("t1 thread");
t2.setName("t2 thread");
t3.setName("t3 thread");

t1.start(); t1.setPriority(10);
t2.start(); t2.setPriority(5);
t3.start(); t3.setPriority(6);

t1.start();
t2.start();
t3.start();

}
}
```

programs

Program 1

Write the program to create two threads. One thread will display numbers from 1 to 50 (ascending order) and other thread will display numbers from 50 to 1 (Descending order) ... 6M

⇒
```
import java.util.Scanner;
import java.lang.*;

Class Ascending extends Thread
{
    public void run()
```

```
    }
  }
}
class Descending extends Thread
{
  public void run()
  {
    for(i=50; i>=1; i--)
    {
System.out.println("Descending Thread " + i);
    }
  }
}
class XYZ
{
  public static void main(String args[])
  {
Ascending a = new Ascending();
Descending b = new Descending();
  a.start();
  b.start();
  }
}
```

Program 2

Write a program to create two threads one thread will print even no between 1 to 50 and other will print odd no between 1 to 50. 6M

```
import java.util.Scanner;
import java.lang;
Class EvenThread extends Thread.
  {
    public void run()
      {
      for(i=2; i<=50; i+=2)
        {
  System.out.println("Even thread:" +i);
      }
    }
  }
Class oddThread extends Thread
  {
    public void run()
      {
      for(int i=1; i<=50; i+=2)
        {
  System.out.println("odd thread: "+i);
      }
    }
  }
Class ABC
  {
    public static void main(String args[])
      {
  EvenThread x = new EvenThread();
  OddThread y = new OddThread();
    x.start();
    y.start();

    }
  }
```

Program3

write a program to print even and odd number using two threads with delay of 1000ms after each number. (6M)

```
import java.lang.*;
class EvenThread extends Thread
    {
    public void run()
     {
  for(int i=2 ; i<=10 ; i+=2)
     {
  System.out.println("Even : "+i);
    try
    {
    Thread.sleep(1000);
    }
   catch (InterruptedException e)
    {
System.out.println("Even Thread interrupted "+
                    e.getMessage());

    }
  }
}

class OddThread extends Thread
   {
    public void run()
     {
  for(int i=1 ; i<=10 ; i+=2)
     {
System.out.println("Odd: "+i);
  try
  {
  Thread.sleep(1000);
   }
  catch (InterruptedException e)
    {
System.out.println("odd Thread interrupted" +
                    e.getMessage());
    }
  }
}
```

```
Class XYZ
{
public static void main(String args[])
  {
EvenThread p = new EvenThread();
OddThread q = new OddThread();
      p.start();
      q.start();
  }
}
```

Program 4

write a Program in which thread A will be display even numbers between 1 to 50 and thread B will display odd numbers between 1 to 50. After 3 iterations thread should go to sleep for 500ms. (6M)

⇒

```
class EvenThread extends Thread
  {
   public void run()
   {
  for(i=2;i<=50;i+=2)
    {
     S.O.P("A = "+i);
    }
  if(i%6==0)
   try
   {
   Thread.sleep(500);
   }
   catch(InterruppledException e)
    {
  tr OP("Even thread Interrupted "+e.getMessage());
    }
  }
}
```