SNS COLLEGE OF ENGINEERING

Kurumbapalayam (Po), Coimbatore – 641 107

An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with ‘A’ Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

COURSE NAME : 23CSB101- OBJECT ORIENTED PROGRAMMING

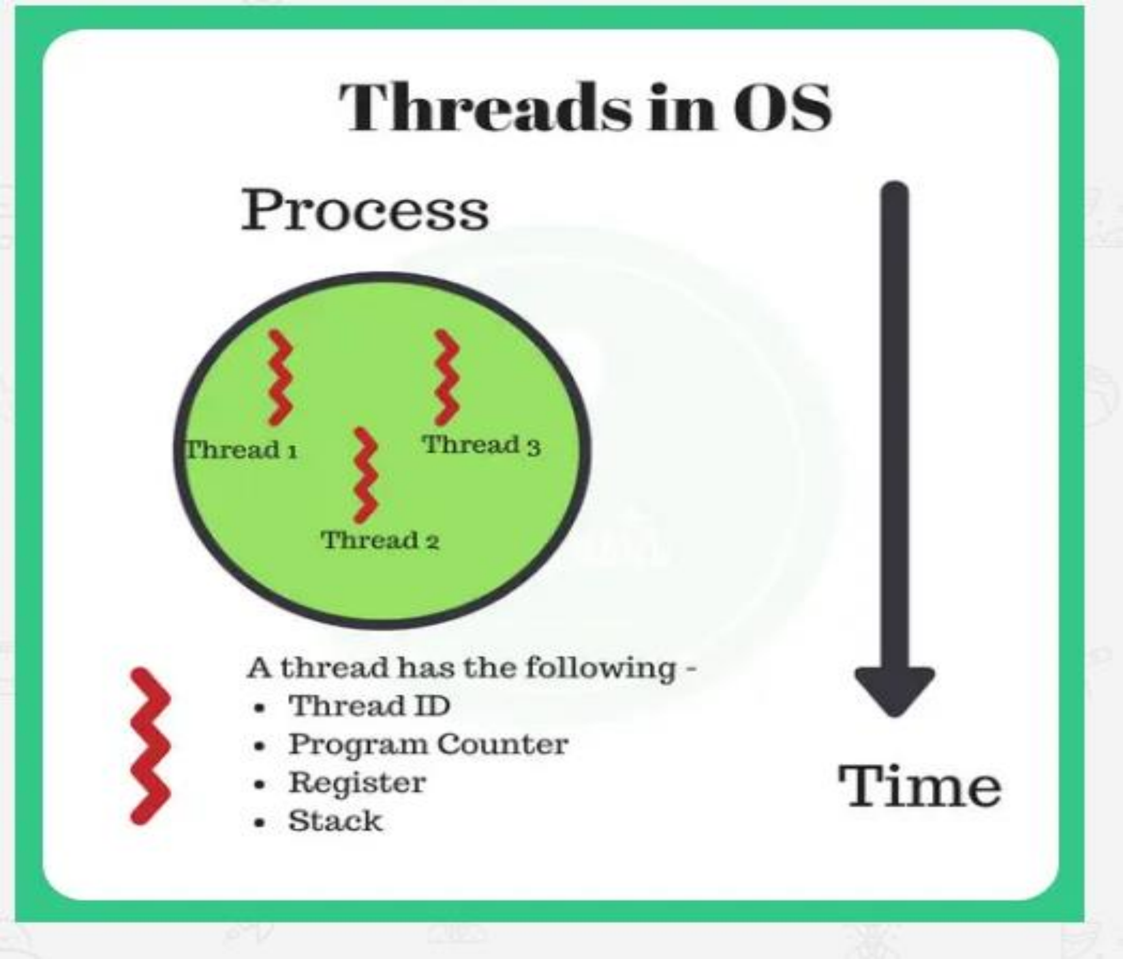
I YEAR /II SEMESTER

Unit III – EXCEPTION HANDLING AND MULTITHREADING

Topic : PRIORITIES – SYNCHRONIZATION

Thread

- A thread is a **lightweight sub-process** that defines a separate path of execution. It is the smallest unit of processing that can run concurrently with other threads of the same process.
- **Multithreading** is a **technique** of executing more than one thread, performing different tasks, simultaneously.
- **Multitasking** is a process of executing multiple tasks simultaneously. It is used to maximize CPU utilization.
- **Process**: Process is a **heavy weight program**. Each process has a complete set of its own variables. Use **IPC** to communicate between processes.



Threads in OS

A thread has the following -

- Thread ID
- Program Counter
- Register
- Stack

THE “main” THREAD

- The “main” thread is a thread that begins running immediately when a java program starts up.
- The “main” thread is important for two reasons:

  1. It is the thread form which other child threads will be spawned.

  2. It must be the last thread to finish execution because it performs various shutdown actions.

- Although the main thread is **created automatically** when our program is started, it can be **controlled through** a Thread object for which a reference to it is done by calling the **method currentThread().**

Example

```
class CurrentThreadDemo {
public static void main(String args[])
{ Thread t=Thread.currentThread();
System.out.println(“Current Thread: “+t);
// change the name of the main thread
t.setName(“My Thread”);
System.out.println(“After name change : “+t);
try {
for(int n=5;n>0;n--) {
System.out.println(n);
Thread.sleep(1000);// delay for 1 second
}
```

Cont…

Example

```
} catch(InterruptedException e) {
 System.out.println(“Main Thread Interrrupted”);
 }
}
}
```

Output:

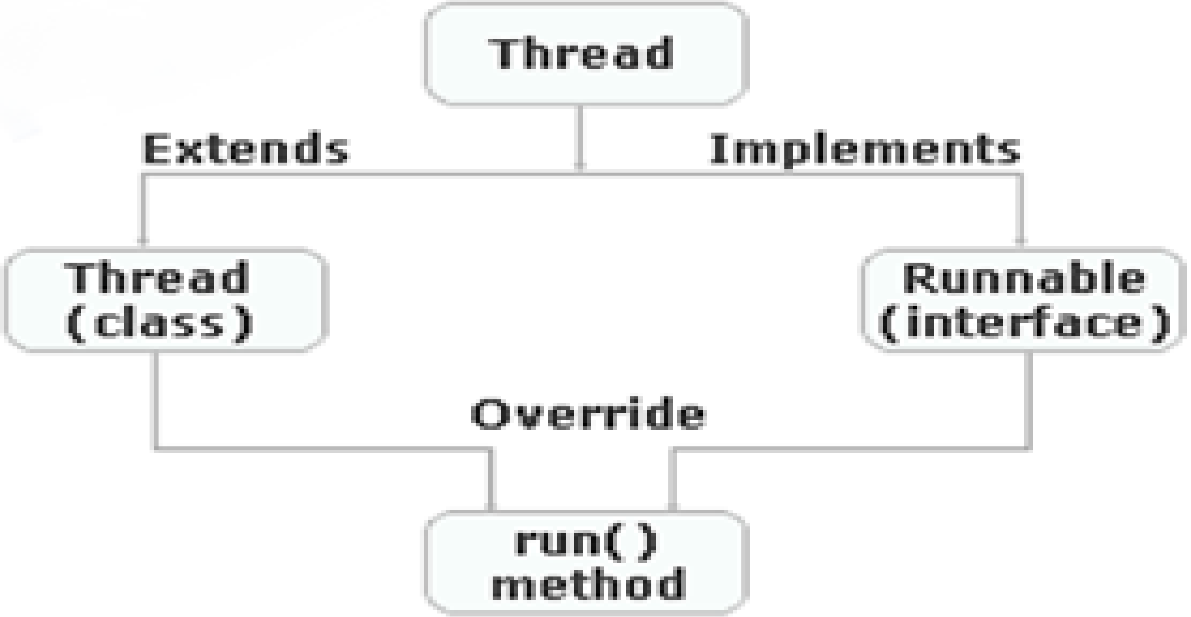
```
Current Thread: Thread[main,5,main]
After name change: Thread[My Thread,5,main]
5
4
3
2
1
```

Creating Threads

Threads are created by instantiating an object of type **Thread**.

Java defines **two ways** to create threads:

1. By implementing Runnable interface (java.lang.Runnable)
2. By extending the Thread class (java.lang.Thread)



Creating threads by implementing Runnable interface:

- The Runnable interface should be implemented by any class whose instances are intended to be executed as a thread.
- Implementing thread program using Runnable is preferable than implementing it by extending Thread class because of the following two reasons:

   1. If a class extends a Thread class, then it cannot extend any other class.

   2. If a class Thread is extended, then all its functionalities get inherited. This is an expensive operation.

Cont…

Creating threads by implementing Runnable interface:

- The Runnable interface has only one method that must be overridden by the class which implements this interface:

```
public void run()   // run() contains the logic of the thread
    {
            // implementation code
    }
```

Creating threads by implementing Runnable interface:

Steps for thread creation:

1. Create a class that implements Runnable interface. An object of this class is Runnable object.

public class MyThread implements Runnable

{ -------------------- }

2. Override the run() method to define the code executed by the thread.

3. Create an object of type Thread by passing a Runnable object as argument.

Thread t=new Thread(Runnable threadobj, String threadName);

4. Invoke the start() method on the instance of the Thread class.

t.start();

Creating threads by implementing Runnable interface:

```
class MyThread implements Runnable
{
public void run()
{
for(int i=0;i<3;i++)
{
System.out.println(Thread.currentThread().getName()+" # Printing "+i);
try
{
Thread.sleep(1000);
}
```

Cont…

Creating threads by implementing Runnable interface:

```
catch(InterruptedException e)
{
System.out.println(e);
 }
 }
}
}
public class RunnableDemo {
public static void main(String[] args)
{
```

```
MyThread obj=new MyThread();
MyThread obj1=new MyThread();
Thread t=new Thread(obj,"Thread-1");
t.start();
Thread t1=new Thread(obj1,"Thread-2");
t1.start();
}
}
```

Creating threads by implementing Runnable interface:

Output:

```
Thread-0 # Printing 0
Thread-1 # Printing 0
Thread-1 # Printing 1
Thread-0 # Printing 1
Thread-1 # Printing 2
Thread-0 # Printing 2
```

Creating threads by extending Thread class

- Thread class provide constructors and methods to create and perform operations on a thread.
- Commonly used **Constructors** of Thread class **to create a new Thread**:
  1. Thread()
  2. Thread(String name)
  3. Thread(Runnable r)
  4. Thread(Runnable r, String name)

Creating threads by extending Thread class

Commonly used methods of Thread class:

1. public void run(): is used to perform action for a thread.

2. public void start(): starts the execution of the thread.JVM calls the run() method on the thread.

3. public void sleep(long miliseconds): Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds.

4. public void join(): waits for a thread to die.

5. public void join(long miliseconds): waits for a thread to die for the specified miliseconds.

6. public int getPriority(): returns the priority of the thread.

7. public int setPriority(int priority): changes the priority of the thread.

Creating threads by extending Thread class

Commonly used methods of Thread class:

8. public String getName(): returns the name of the thread.

9. public void setName(String name): changes the name of the thread.

10.public Thread currentThread(): returns the reference of currently executing thread.

11.public boolean isAlive(): tests if the thread is alive.

12.public void yield(): causes the currently executing thread object to temporarily pause and allow other threads to execute.

13.public void suspend(): is used to suspend the thread(depricated).

Creating threads by extending Thread class

Commonly used methods of Thread class:

14.public void resume(): is used to resume the suspended thread(depricated).

15.public void stop(): is used to stop the thread(depricated).

16.public boolean isDaemon(): tests if the thread is a daemon thread.

17.public void setDaemon(boolean b): marks the thread as daemon or user thread.

18.public void interrupt(): interrupts the thread.

19.public boolean isInterrupted(): tests if the thread has been interrupted.

20.public static boolean interrupted(): tests if the current thread has been interrupted.

Creating threads by extending Thread class

Steps for thread creation:

1. Create a class that extends java.lang.Thread class.

```
public class MyThread extends Thread
{            ---             }
```

2. Override the run() method in the sub class to define the code executed by the thread.

3. Create an object of this sub class.

```
MyThread t=new MyThread(String threadName);
```

4. Invoke the start() method on the instance of the subclass to make the thread for running.

```
start();
```

Creating threads by extending Thread class

```
class SampleThread extends Thread
{
  public void run()
  {
   for(int i=0;i<3;i++)
   {
    System.out.println(Thread.currentThread().getName()+" # Printing "+i);
     try
        {
          Thread.sleep(1000);
        }
    catch(InterruptedException e)
        {  System.out.println(e); }
   }
  }
}
```

CONT…

Creating threads by extending Thread class

```
public class ThreadDemo
{
public static void main(String[] args)
{
SampleThread obj=new SampleThread();
obj.start();
SampleThread obj1=new SampleThread();
obj1.start();
}
}
```

Output:

```
Thread-0 # Printing 0
Thread-1 # Printing 0
Thread-1 # Printing 1
Thread-0 # Printing 1
Thread-0 # Printing 2
Thread-1 # Printing 2
```

THREAD PRIORITY

- Thread priority determines how a thread should be treated with respect to others.
- Every thread in java has some priority, it may be default priority generated by JVM or customized priority provided by programmer.
- Priorities are represented by a number between 1 and 10.

1 – Minimum Priority5 – Normal Priority 10 – Maximum Priority

- Thread scheduler will use priorities while allocating processor. The thread which is having highest priority will get the chance first.

THREAD PRIORITY

Three constants defined in Thread class:

1.public static int MIN_PRIORITY

2.public static int NORM_PRIORITY

3.public static int MAX_PRIORITY

Default priority of a thread is **5** (NORM_PRIORITY).

The value of **MIN_PRIORITY** is **1**

and the value of **MAX_PRIORITY** is **10**.

THREAD PRIORITY

Three constants defined in Thread class:

1.public static int MIN_PRIORITY

2.public static int NORM_PRIORITY

3.public static int MAX_PRIORITY

Default priority of a thread is **5** (NORM_PRIORITY).

The value of **MIN_PRIORITY** is **1**

and the value of **MAX_PRIORITY** is **10**.

To set a thread's priority, **setPriority()** and to get the current priority **getPriority()** method is used.

Example: THREAD PRIORITY

```
class TestMultiPriority1 extends Thread{
public void run(){
 System.out.println("running thread name is:"+Thread.currentThread().getName());
 System.out.println("running thread priority is:"+  Thread.currentThread().getPriority());
 }
public static void main(String args[]) {
 TestMultiPriority1 m1=new TestMultiPriority1();
 TestMultiPriority1 m2=new TestMultiPriority1();
 m1.setPriority(Thread.MIN_PRIORITY);
 m2.setPriority(Thread.MAX_PRIORITY);
 m1.start();
 m2.start();
}
}
```

```
running thread name is:Thread-0
 running thread priority is:10
 running thread name is:Thread-1
 running thread priority is:1
```

Thread Synchronization

- Thread synchronization is the concurrent execution of two or more threads that share critical resources.
- When two or more threads need to use a shared resource, they need some way to ensure that the resource will be used by only one thread at a time. **The process of ensuring single thread access to a shared resource at a time is called synchronization.**

Thread Synchronization

- There are **two types of thread synchronization** mutual exclusive and inter-thread communication.

1. **Mutual Exclusive**
    1. Synchronized method.
    2. Synchronized block.
    3. static synchronization.
2. **Cooperation** (Inter-thread communication in java)

Thread Synchronization - Mutual Exclusive

1. Synchronized method.

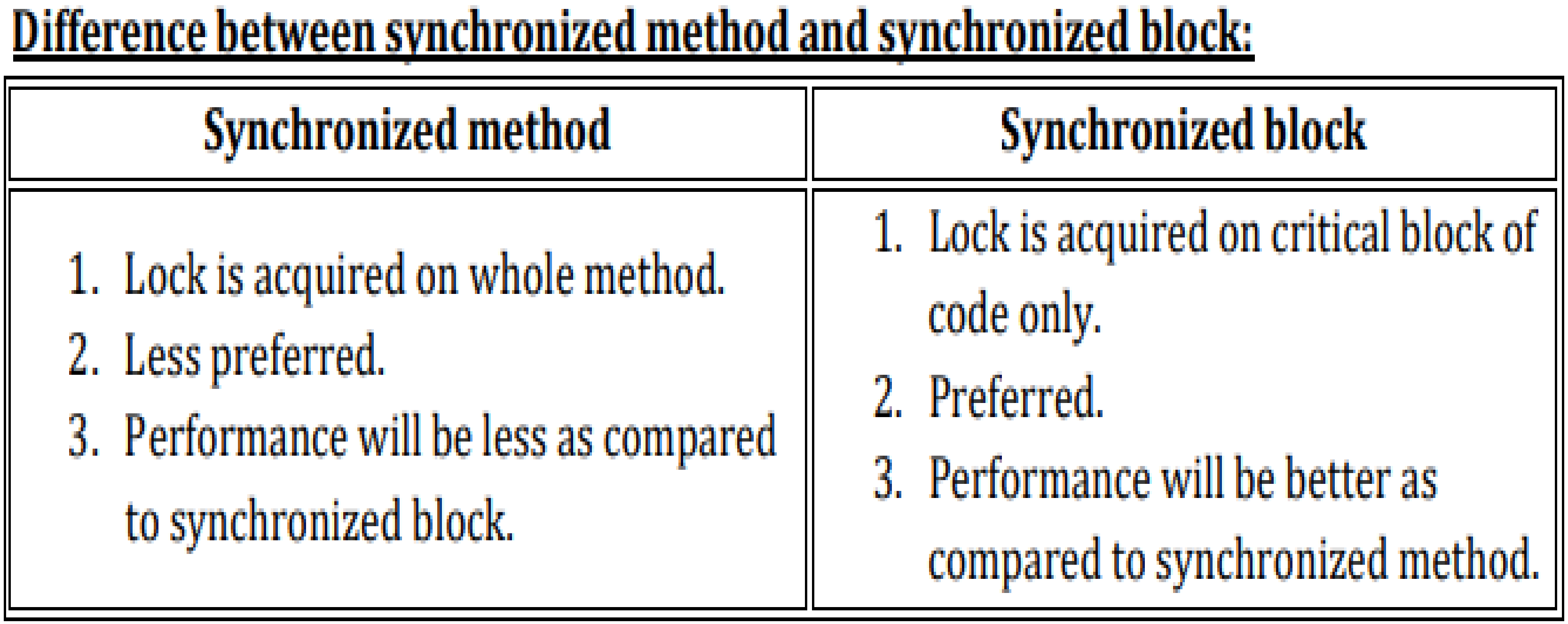
Syntax :

```
 Access_modifier synchronized return_type method_name(parameters)
 { …….. }
```

2. Synchronized block in java

Syntax:

```
synchronized (object reference expression)
{
 //code block
}
```



Difference between synchronized method and synchronized block:

Synchronized method	Synchronized block
1. Lock is acquired on whole method. 2. Less preferred. 3. Performance will be less as compared to synchronized block.	1. Lock is acquired on critical block of code only. 2. Preferred. 3. Performance will be better as compared to synchronized method.

```
class SharedResource {
    // Synchronized method (locks the entire method)
    synchronized void synchronizedMethod(String msg) {
        System.out.print("[ " + msg);
        try { Thread.sleep(1000); } catch (InterruptedException e) { }
        System.out.println(" ]");
    }

    // Method using a synchronized block (locks only critical section)
    void synchronizedBlock(String msg) {
        System.out.print("Start ");
        synchronized (this) {  // Only this block is synchronized
            System.out.print("[ " + msg);
```

Cont…

```
try { Thread.sleep(1000); } catch (InterruptedException e) { }
            System.out.println(" ]");   }
           System.out.println("End");     }
}
class SyncExample {
    public static void main(String[] args) {
        SharedResource resource = new SharedResource();
        // Using threads to demonstrate synchronization
        Thread t1 = new Thread(() -> resource.synchronizedMethod("Hello"));
        Thread t2 = new Thread(() -> resource.synchronizedBlock("World"));
        t1.start();
        t2.start();
    }
}
```

Order may vary

```
[ Hello ]
Start [ World ]
End
```

`

Thank you