UNIT - III

MULTITHREADING

Multithreading Concepts

- A multithreaded program contains two or more parts that can run concurrently.
- Each part of such a program is called a thread, and each thread defines a separate path of execution.
- Multithreading is a specialized form of multitasking.

Process & Thread

- Process and Thread are two basic units of Java program execution.
- Process: A process is a self contained execution environment and it can be seen as a program or application.
- Thread: It can be called lightweight process
  - Thread requires less resources to create and exists in the process
  - Thread shares the process resources

Thread Model

- Threads are implemented in the form of objects.
- The run() and start() are two inbuilt methods which helps to thread implementation
- The run() method is the heart and soul of any thread – It makes up the entire body of a thread
- The run() method can be initiating with the help of start() method.

Creating Thread

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

1. By Extending Thread class

```java
public class MultiThread1 extends Thread {

    public void run() {
        // code to be executed in this thread
        System.out.println("Thread " + this.getName() + " is running.");
    }

    Run | Debug
    public static void main(String[] args) {
        MultiThread1 t1 = new MultiThread1();
        MultiThread1 t2 = new MultiThread1();
        MultiThread1 t3 = new MultiThread1();

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

```
Thread Thread-0 is running.
Thread Thread-1 is running.
Thread Thread-2 is running.
```

2. By implementing Runnable interface

- Define a class that implements Runnable interface.

- The Runnable interface has only one method, run(), that is to be defined in the method with the code to be executed by the thread.

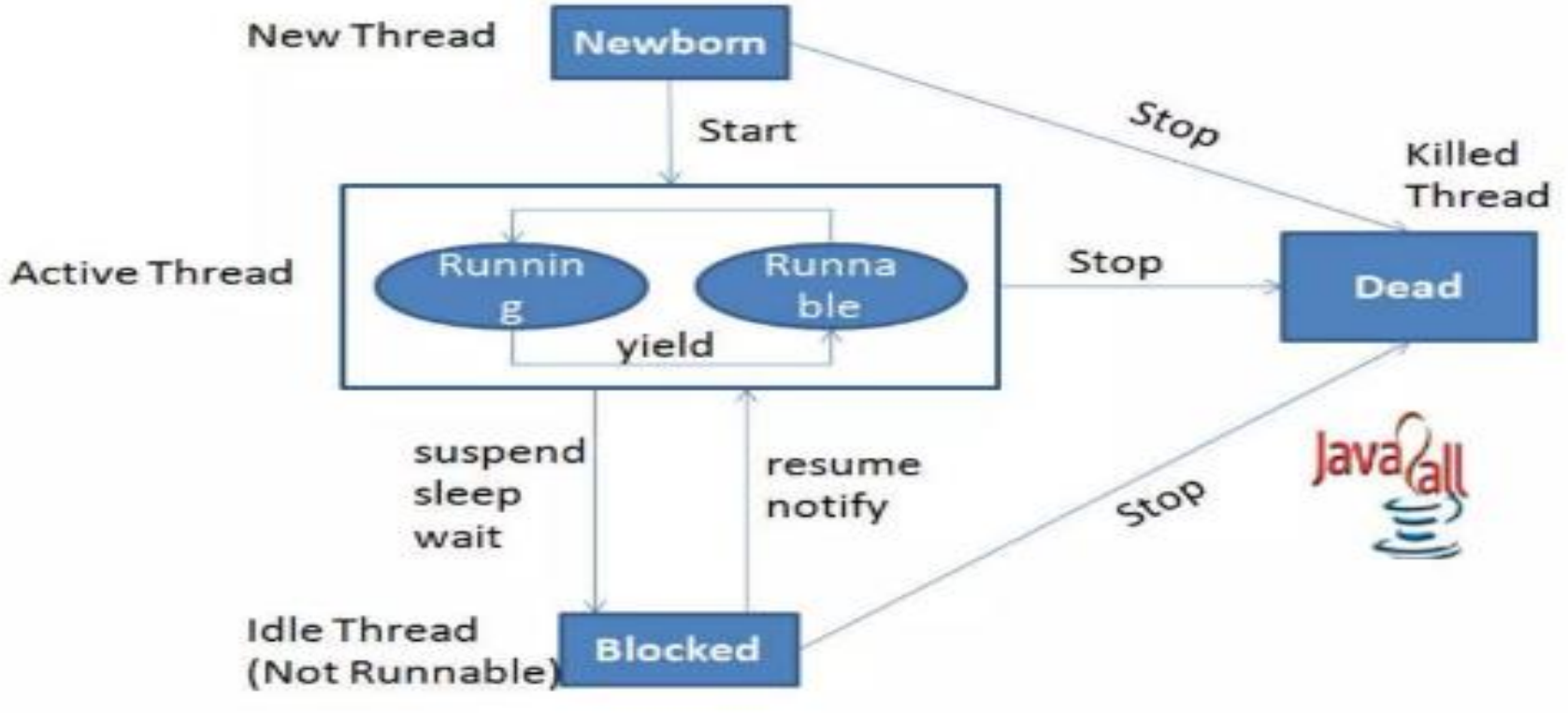
```java
class MyRunnable implements Runnable {
    public void run() {
        for (int i = 1; i <= 10; i++) {
            System.out.println(i);
        }
    }
}

public class Run1 {
    Run | Debug
    public static void main(String[] args) {
        MyRunnable myRunnable = new MyRunnable();
        Thread thread = new Thread(myRunnable);
        thread.start();
    }
}
```

```
c:\Java Programs>
ShowCodeDetailsInE
75add73\redhat.jav
1
2
3
4
5
```

Life cycle of a Thread

- During the life time of a thread, there are many states it can enter. • They include:



Sample program

```
public class Healthcare5 {

    Run | Debug
    public static void main(String[] args) {

        // create an instance of a medical simulation
        MedicalSimulation simulation = new MedicalSimulation();

        // create multiple threads to run the simulation
        Thread thread1 = new Thread(simulation);
        Thread thread2 = new Thread(simulation);
        Thread thread3 = new Thread(simulation);

        // start the threads
        thread1.start();
        thread2.start();
        thread3.start();
    }
}
```

```java
class MedicalSimulation implements Runnable {

    // simulate medical data processing
    public void processData() {
        System.out.println(x:"Patient Records");
    }

    // simulate medical scenario simulation
    public void simulateScenario() {
        System.out.println(x:"Doctor Details");
    }

    @Override
    public void run() {
        // run the medical simulation in a loop
        for (int i = 0; i < 2; i++) {
            processData();
            simulateScenario();
        }
    }
}
```

```
ShowCodeDetailsInEx
75add73\redhat.java
Patient Records
Doctor Details
Patient Records
Patient Records
Patient Records
Doctor Details
Doctor Details
Patient Records
Doctor Details
```

Java Thread Priorities

- You can set the priority of a thread using the setPriority(int priority) method of the Thread class.
- Whenever we create a thread in Java, it always has some priority assigned to it.
- Priority can either be given by JVM while creating the thread or it can be given by the programmer explicitly.
- We can use the getPriority() method of the Thread class to get the current priority of a thread.
- Threads with the same priority are scheduled in a round-robin fashion.
- You can use the Thread.MIN_PRIORITY, Thread.NORM_PRIORITY, and Thread.MAX_PRIORITY constants to set the minimum, default, and maximum priorities for threads, respectively.

Sample Program

```
class MyThread implements Runnable {
    private String name;

    MyThread(String name) {
        this.name = name;
    }


    public void run() {
        System.out.println("Thread " + name + " started.");

        for(int i = 0; i < 4; i++) {
            System.out.println("Thread " + name + " running " + i);

            // Yield to allow other threads to run
            Thread.yield();
        }

        System.out.println("Thread " + name + " finished.");
    }
}
```

```java
public class ThreadPriorityDemo1 {
    Run | Debug
    public static void main(String[] args) {
        // Create three threads
        Thread t1 = new Thread(new MyThread(name:"Thread 1"));
        Thread t2 = new Thread(new MyThread(name:"Thread 2"));
        Thread t3 = new Thread(new MyThread(name:"Thread 3"));

        // Set thread priorities
        t1.setPriority(Thread.MIN_PRIORITY);
        t2.setPriority(Thread.NORM_PRIORITY);
        t3.setPriority(Thread.MAX_PRIORITY);

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

Thread Class and the Runnable Interface

- Java’s multithreading system is built upon the Thread class, its methods, and its companion interface, Runnable. Thread encapsulates a thread of execution.
- Since you can’t directly refer to the ethereal state of a running thread, you will deal with it through its proxy, the Thread instance that spawned it.
- To create a new thread, your program will either extend Thread or implement the Runnable interface.

Thread Class

- The Thread class is a predefined class in Java that represents a thread of execution. To create a new thread using the Thread class, you can either extend the Thread class and override its run() method, or you can create an instance of the Thread class and pass a Runnable object to its constructor. The run() method contains the code that the thread will execute when it starts running.
- The Thread class defines several methods that help manage threads.
- Several of those used in this chapter are shown here:

Method	Meaning
getName	Obtain a thread's name.
getPriority	Obtain a thread's priority.
isAlive	Determine if a thread is still running.
join	Wait for a thread to terminate.
run	Entry point for the thread.
sleep	Suspend a thread for a period of time.
start	Start a thread by calling its run method.

Runnable Interface

- The Runnable interface, on the other hand, is a functional interface that represents a unit of work that can be executed in a separate thread.
- To use the Runnable interface, you need to implement the run() method defined in the interface.
- Then, you can either create an instance of a class that implements the Runnable interface, or you can use a lambda expression to create a Runnable object.

Thread Synchronization

- When two or more threads need access to a shared resource, they need some way to ensure that the resource will be used by only one thread at a time. The process by which this is achieved is called synchronization. As you will see, Java provides unique, language-level support for it.
- Key to synchronization is the concept of the monitor. A monitor is an object that is used as a mutually exclusive lock. Only one thread can own a monitor at a
- given time. When a thread acquires a lock, it is said to have entered the monitor.
- All other threads attempting to enter the locked monitor will be suspended until
- the first thread exits the monitor. These other threads are said to be waiting for
- the monitor. A thread that owns a monitor can reenter the same monitor if it so
- desires.

Thread Synchronization

- Why use Synchronization?

- The synchronization is mainly used to
  - To prevent thread interference.
  - To prevent consistency problem.

- Types of Synchronization
- There are two types of synchronization
  - Process Synchronization
  - Thread Synchronization

Non Synchronized method

```java
public class NonSynchronizedMethodExample {
    private int count;

    public void incrementCount() {
        count++;
    }

    public int getCount() {
        return count;
    }

    Run | Debug
    public static void main(String[] args) {
        NonSynchronizedMethodExample example = new NonSynchronizedMethodExample();

        // create two threads to increment the count
        Thread thread1 = new Thread(new Runnable() {
            public void run() {
                for (int i = 0; i < 100000; i++) {
                    example.incrementCount();
                }
            }
        });
```

```
        Thread thread2 = new Thread(new Runnable() {
            public void run() {
                for (int i = 0; i < 100000; i++) {
                    example.incrementCount();
                }
            }
        });

        // start the threads
        thread1.start();
        thread2.start();

        // wait for the threads to finish
        try {
            thread1.join();
            thread2.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        // print the final count
        System.out.println("Final count: " + example.getCount());
    }
}
```

```
Final count: 137613
```

Synchronized method

```java
public class Synchronization2 {
    private int count = 0;

    public synchronized void increment() {
        count++;
    }

    Run | Debug
    public static void main(String[] args) throws InterruptedException {
        Synchronization2 example = new Synchronization2();

        Thread thread1 = new Thread(new Runnable() {
            public void run() {
                for (int i = 0; i < 100000; i++) {
                    example.increment();
                }
            }
        });
```

```
        Thread thread2 = new Thread(new Runnable() {
            public void run() {
                for (int i = 0; i < 100000; i++) {
                    example.increment();
                }
            }
        });

        thread1.start();
        thread2.start();
        thread1.join();
        thread2.join();

        System.out.println("Count: " + example.count);
    }
}
```

```
Count: 200000
```

Inter Thread Communication.

- Inter-thread communication or Co-operation is all about allowing synchronized threads to communicate with each other.

- Cooperation (Inter-thread communication) is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same critical section to be executed. It is implemented by following methods of Object class:
- wait()
- notify()
- notifyAll()

Sample Program

```
public class InterThreadCommunicationExample {
    Run | Debug
    public static void main(String[] args) {
        Message message = new Message();
        Thread senderThread = new Thread(new Sender(message));
        Thread receiverThread = new Thread(new Receiver(message));
        senderThread.start();
        receiverThread.start();
    }
}

class Message {
    private String message;
    private boolean empty = true;

    public synchronized String read() {
        while (empty) {
            try {
                wait();
            } catch (InterruptedException e) {}
        }
        empty = true;
        notifyAll();
        return message;
    }
```

```
    public synchronized void write(String message) {
        while (!empty) {
            try {
                wait();
            } catch (InterruptedException e) {}
        }
        empty = false;
        this.message = message;
        notifyAll();
    }
}

class Sender implements Runnable {
    private Message message;

    public Sender(Message message) {
        this.message = message;
    }
```

```
    public void run() {
        String[] messages = {
            "Message 1",
            "Message 2",
            "Message 3",
            "Message 4"
        };
        for (String message : messages) {
            this.message.write(message);
            try {
                Thread.sleep(millis:1000);
            } catch (InterruptedException e) {}
        }
        this.message.write(message:"Finished");
    }
}

class Receiver implements Runnable {
    private Message message;

    public Receiver(Message message) {
        this.message = message;
    }
```

```
    public void run() {
        String message = "";
        while (!message.equals(anObject:"Finished")) {
            message = this.message.read();
            System.out.println("Received message: " + message);
        }
    }
}
```

```
Received message: Message 1
Received message: Message 2
Received message: Message 3
Received message: Message 4
Received message: Finished
```