Java™

Introduction to Multithreading

What is Process ?

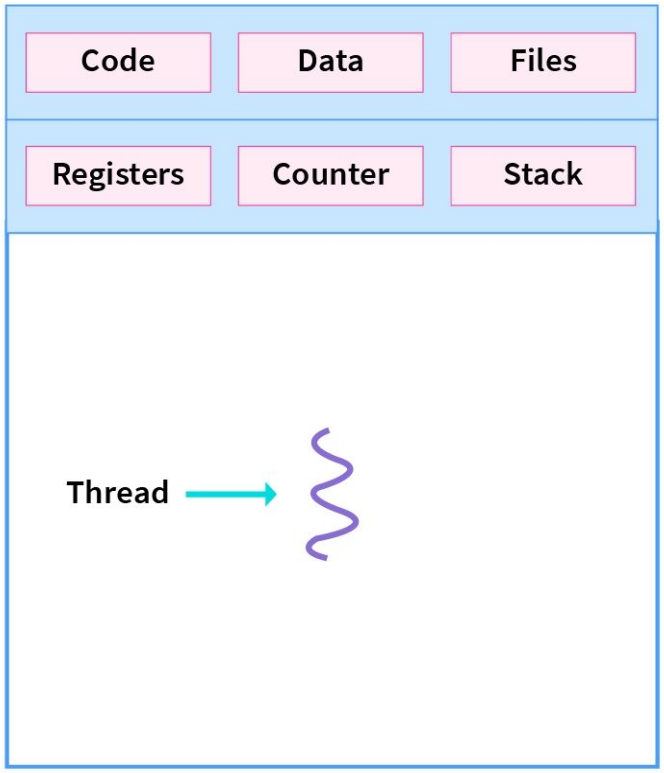
A process is an independent execution unit with its own memory space, resources, and at least one thread. Think of it as a complete, self-contained running program on your computer.

What is Thread ?

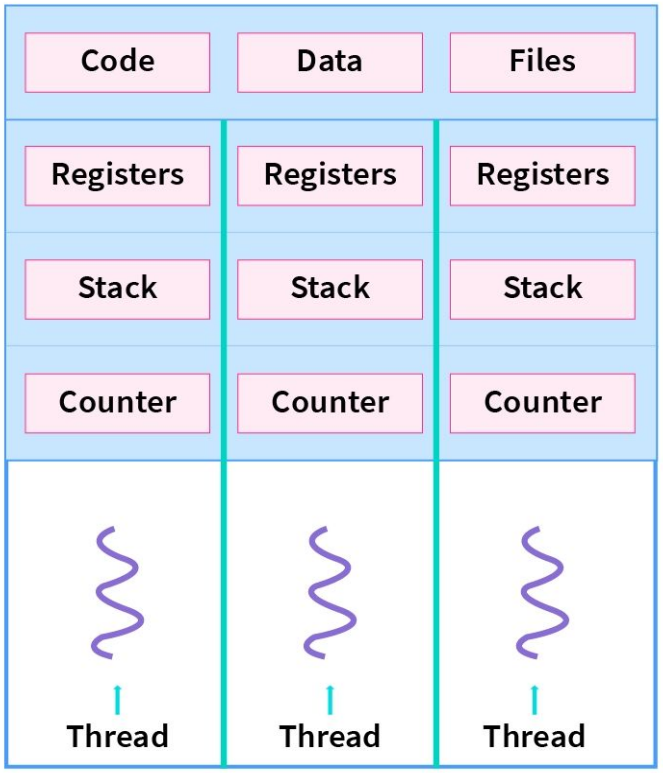
A thread is the smallest unit of execution within a process. In Java, a thread represents an independent path of execution of a program.

1. A process is a running instance of a program, and it can contain multiple threads.
2. A thread is like a lightweight subprocess that runs in parallel with other threads in the same process.
3. All threads within a process share the same memory space, which makes communication easier but can also cause issues like race conditions.
4. The main thread is automatically created when any Java program starts. Additional threads can be created for parallel tasks.

What is Thread ?



Single-threaded process

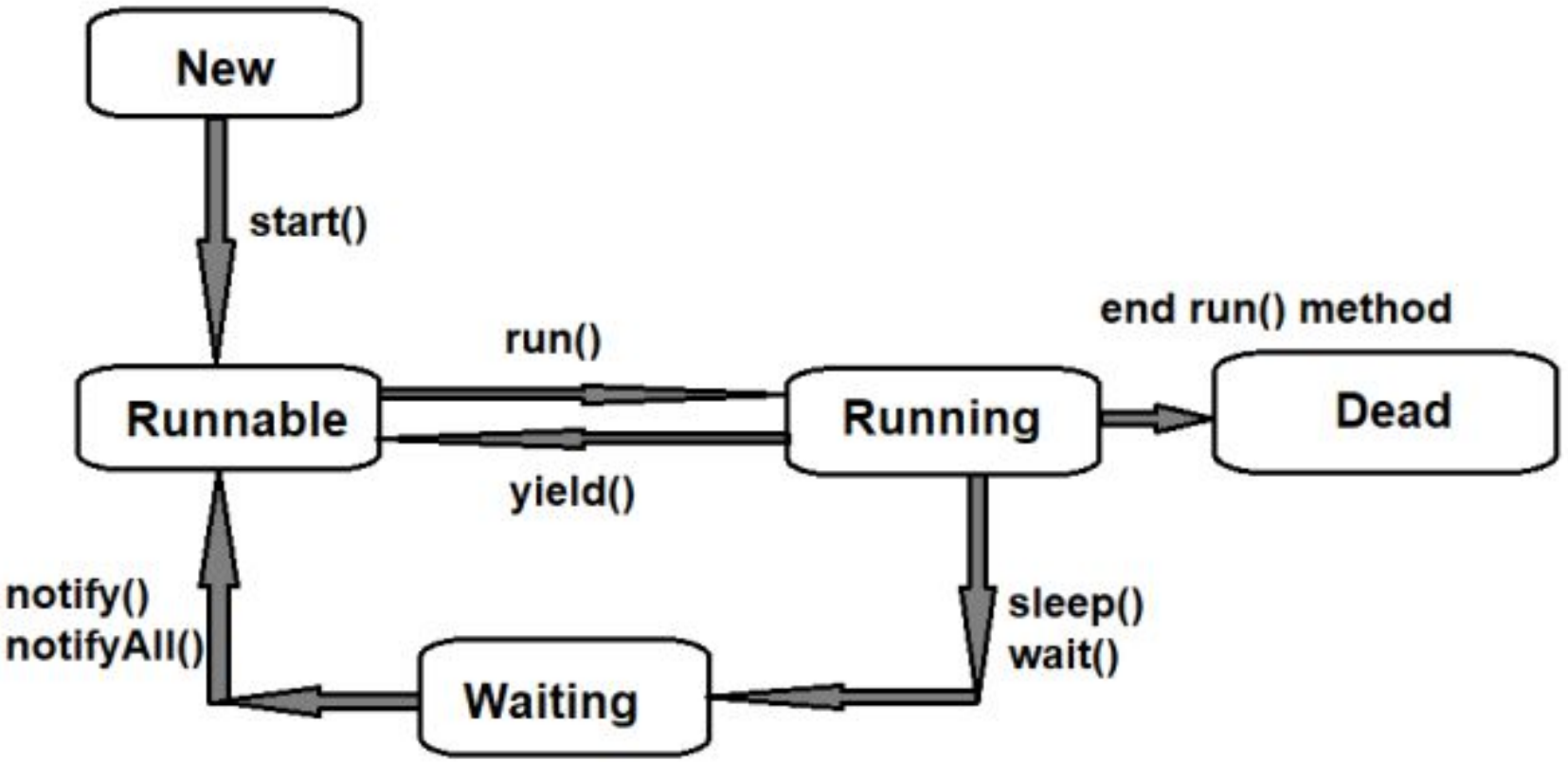


Multithreaded process

Process vs Thread

Aspect	Process	Thread
Definition	Program in execution	Sub-task of a process
Memory	Separate for each process	Shared memory (heap & code)
Stack	Separate stack	Separate stack
Creation Cost	High (heavyweight)	Low (lightweight)
Communication	Complex (IPC)	Easy (shared memory)
Context Switch	Slow (save entire process state)	Fast (save only thread state)
Failure Impact	Independent	Affects entire process
Overhead	High	Low
Execution	Independent	Depends on the process

Thread lifecycle



Life Cycle of Thread in Java

Advantages of Thread

1. Better CPU Utilization
2. Faster Execution
3. Improved Application Responsiveness
4. Simplified Program Structure for Concurrent Tasks
5. Resource Sharing
6. Better User Experience

Creating Threads

In Java, thread creation can happen in two ways.

1. Thread Class of java.lang package

2. implement the java.lang.Runnable interface.

Thread Class

Steps to create threads using Thread class:

1. Write a class that extends the Thread class.
2. Override/redefine the run() of the Thread class to define the operations that need to be performed by the thread.
3. Create instances of the subclass of Thread and start invoking start() method.

```
class MyThread extends Thread {

@Override
 public void run() {

      for (int i = 1; i <= 5; i++) {

         System.out.println("Thread: " + i);

         try { Thread.sleep(500); }
 catch (InterruptedException e) {}

      }

   }

}
```

```
public class ThreadExample {
   public static void main(String[] args) {
      MyThread t1 = new MyThread();
      t1.start(); // Starts the thread

      for (int i = 1; i <= 5; i++) {
         System.out.println("Main: " + i);
         try { Thread.sleep(500); } catch
(InterruptedException e) {}
      }
   }
}
```

Runnable interface.

Steps in order to come up with a thread using the Runnable interface:

1. Write a class that implements the Runnable interface of the java.lang package.
2. Implement the run() method to define the operations to be performed by the thread.
3. Create instances of Thread class by passing the instance of the class that implements the Runnable interface. Then, invoke the start() method on the instance that got created.

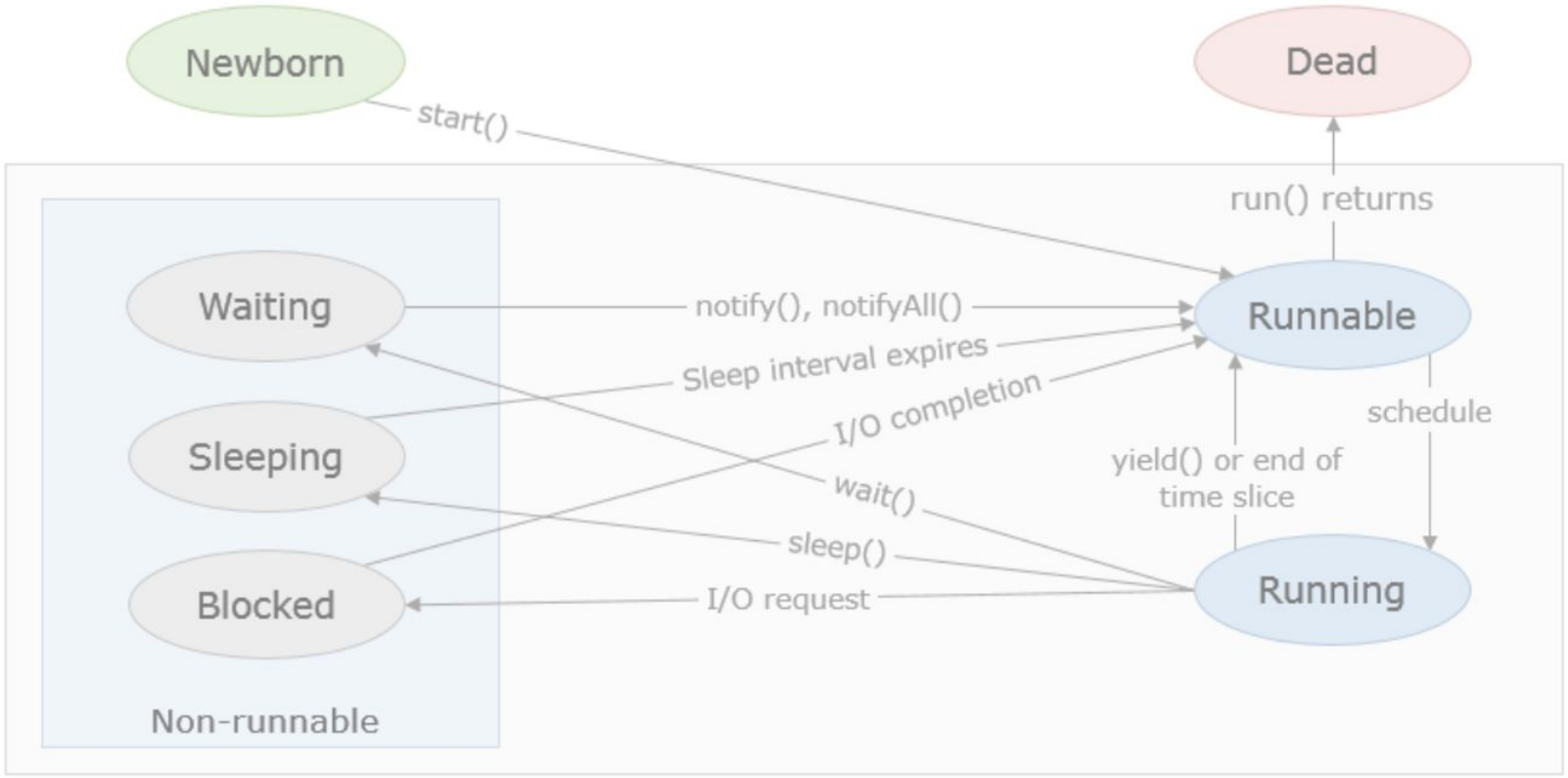

```
class UploadResult implements Runnable {
   @Override
   public void run() {
      // Thread implementation
   }
}
```

```
class Test {
   public static void main(String[] args) {
      UploadResult uploadRunnable = new
UploadResult();
      Thread threadObj = new
Thread(uploadRunnable);
      threadObj.start();
   }
}
```

Thread methods

Method Name	Description
void start()	Begins the thread's execution. Then JVM invokes the run()method.
void run()	contains the code which needs to be executed by the threads.
void sleep(int duration)	For the specified milliseconds of duration, it suspends the execution of the thread.
void yield()	Pauses the execution of the thread temporarily and allows other threads with the same priority to start their execution/continue.
void join()	Waits for the thread to die.
boolean isAlive()	Returns whether the thread is alive or not.

Thread LifeCycle



Thread Scheduling and Priority

Thread moves from RUNNABLE to RUNNING state. The scheduler makes use of two techniques of thread scheduling here.

Preemptive: The thread of higher priority preempts the threads of lower priority and grabs/avails the CPU.

Time Sliced: Also named as round-robin scheduling which makes each thread get some time of the CPU.

Methods:

1. **setPriority()**
2. **getPriority()**

The priority of a thread can vary from 1 (Thread.MIN_PRIORITY) to 10 (Thread.MAX_PRIORITY). 5 is the default priority(Thread.NORM_PRIORITY).

Synchronization

- Synchronization is a technique to control access of multiple threads to shared resources.
- It ensures only one thread can access the critical section of code (where shared data is modified) at a time.
- Without synchronization, race conditions can occur, leading to inconsistent results.

Synchronization

```
class SharedResource {
    private boolean isAvailable = false;
    synchronized void produce() {
        System.out.println("Producer thread running...");
        isAvailable = true;
        notify(); // Notify the waiting consumer
    }
    synchronized void consume() {
        while (!isAvailable) {
            try {
                wait(); // Wait until producer notifies
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        System.out.println("Consumer thread consumed the item.");
    }
}
```

```
public class WaitNotifyDemo {
    public static void main(String[] args) {
        SharedResource resource = new SharedResource();

        Thread t1 = new Thread(() -> resource.consume());
        Thread t2 = new Thread(() -> resource.produce());

        t1.start();
        t2.start();
    }
}
```

Assignments

1.Write a Java program that:

- Defines a class MyThread that extends Thread.
- Overrides run() to print "Thread is running" five times, sleeping 500 ms between each print.
- In main, creates two objects of MyThread and starts them using start().

2. Implement Runnable in a class MyRunnable.

- In run(), print numbers from 1 to 10.
- Start two threads using the same Runnable object.

3. Start a thread that prints numbers 1 to 5.

- From the main thread, check isAlive() before and after calling join().
- Print messages showing when the thread finishes.

4. Create a thread that prints the multiplication table of 5.

- Use Thread.sleep(300) between each print.

Assignments

5. Write a Java program that creates three threads.

    a) Assign them priorities 1, 5, and 10.

    b) Print each thread's name and priority when it starts.

THANK YOU!!

C-DAC Patna