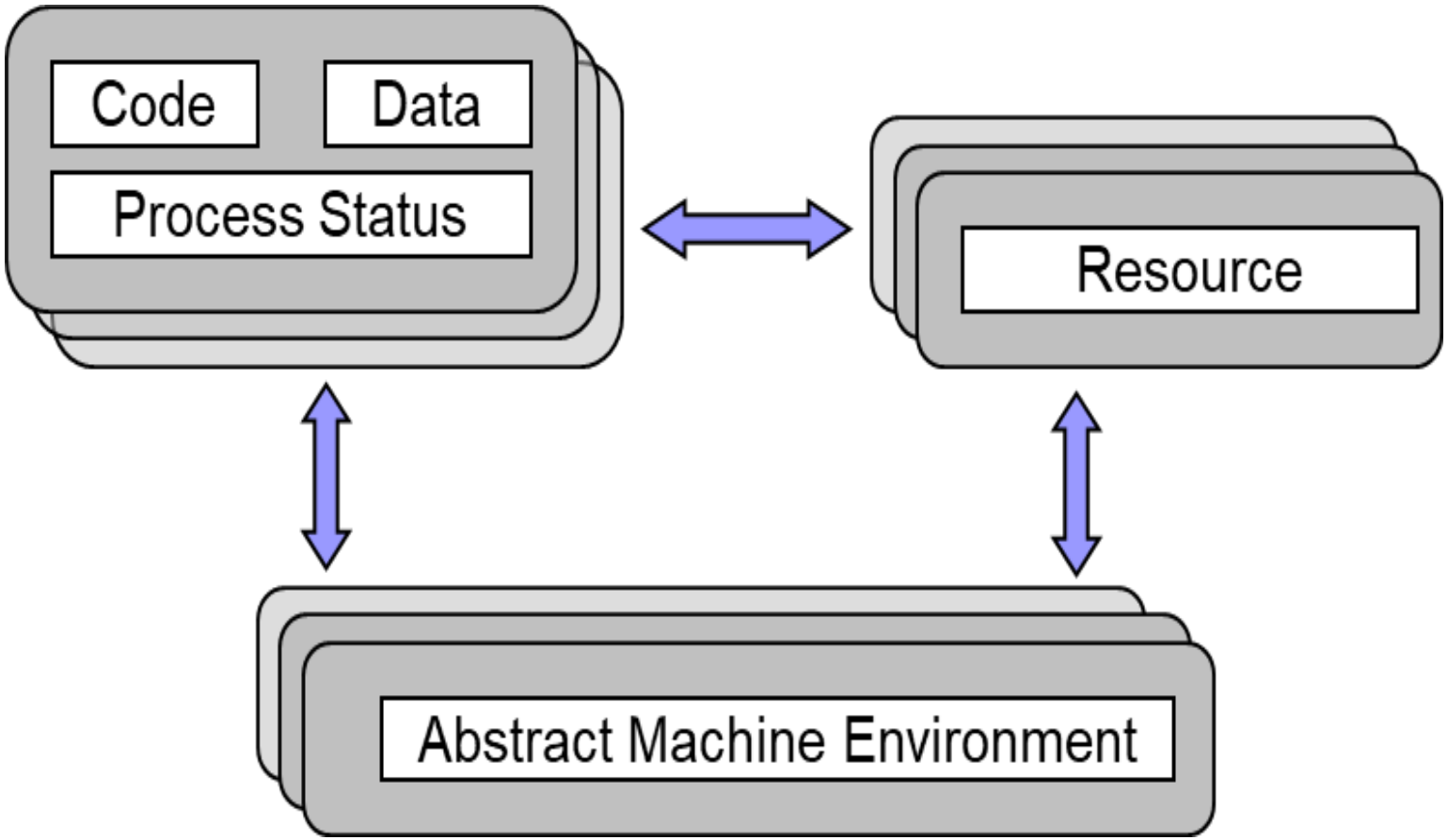
Multithreading in Java

Dr. Partha Pratim Sarangi
School of Computer Engineering

Process

- A process is a sequential program in execution.
- A process is a unit of computation.
- Process components:
  - The program (code) to be executed.
  - The data on which the program will execute.
  - Resources required by the program.
  - The status of the process execution.
- A process runs in an Operating System runtime environment that manages the sharing and isolation of resources among the community of processes.

Process Model



Program and Process

- A program is a static entity made up of program statements.
- A process is a dynamic entity that executes a program on a particular set of data.
- Two or more processes could execute the same program, each using their **own data** and **resources**.

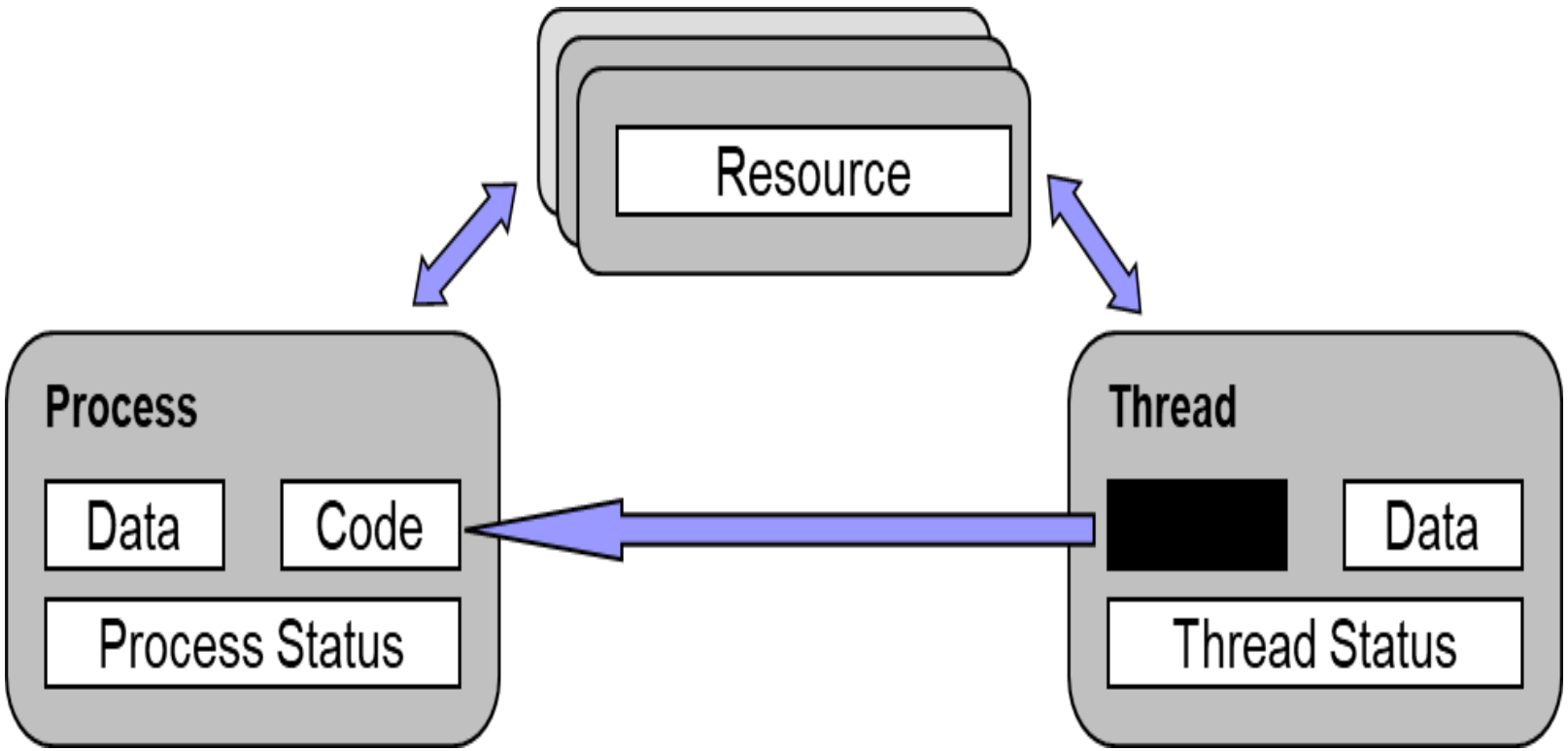
Thread

- A thread is an alternative form (to the process) of schedulable unit of computation.
- In the thread model:
  - Each thread is associated with a process.
  - A thread is an entity that executes by relying on the code and resources, holding by the associated process.
  - Several threads could be associated with a single process. Those threads share the code and resources of the process.
  - A thread allocates part of the process’s resources for its needs.
  - A thread has its own data and status.

Tread Model

- Control in a normal program usually follows a single thread of execution.
- What differentiates threads from normal processes is the shared memory (of the process), which is visible to all threads in a multithreading program.
- A thread has much less overhead than a process so it is called as a **light weight process**.
- Multithreading allows an application to have multiple threads of execution running concurrently.

Thread Model



Why do we need threads?

- To enhance parallel processing
- To increase response to the user
- To utilize the idle time of the CPU
- Prioritize your work depending on priority

Example

- Consider a simple web server
- The web server listens for request and serves it.
- If the web server was not multithreaded, the requests processing would be in a queue, thus increasing the response time and also might hang the server if there was a bad request.
- By implementing in a multithreaded environment, the web server can serve multiple request simultaneously thus improving response time

Multitasking and Multithreading

- Multitasking refers to a computer's ability to perform multiple jobs concurrently
  - more than one program are running concurrently.
- A thread is a single sequence of execution within a program.
- Multithreading refers to multiple threads of control within a single program.
  - each program can run multiple threads of control within it, e.g., Web Browser

Thread in Java

- When we execute an application:
  - The JVM creates a Thread object whose task is defined by the **main()** method.
  - It starts the thread.
  - The thread executes the statements of the program one by one until the method returns and the thread dies.

Multiple Threads in Java

- Each thread has its private **run-time stack**.
- If two threads execute the same method, each will have its own copy of the local variables the methods uses.
- However, all threads see the same dynamic memory (heap)
- Two different threads can act on the **same object** and same **static fields concurrently**.

Creating threads

- In java threads can be created by **extending the Thread class** or **implementing the Runnable Interface**.

- It is more preferred to implement the Runnable Interface so that we can extend properties from other classes.

- Implement the run() method which is the starting point for thread execution

Thread Class Methods

- **void start()**
  - Creates a new thread and makes it runnable.
  - This method can be called only once.
- **void run()**
  - The new thread begins its life inside this method.
- **void stop()** (deprecated)
  - The thread is being terminated.

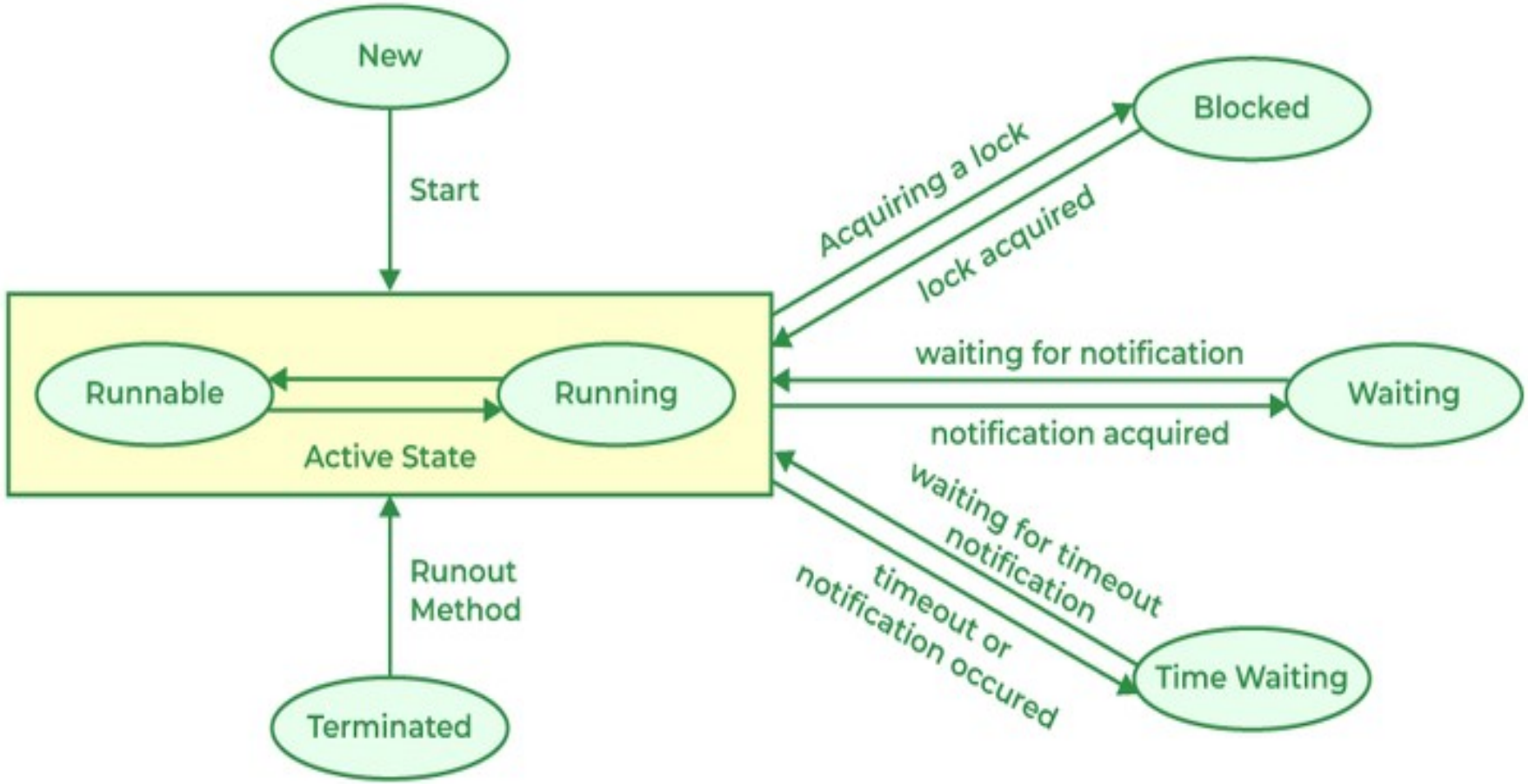
Thread Methods

- **yield()**
  - Causes the currently executing thread object to temporarily pause and allow other threads to execute.
  - Allow only threads of the same priority to run
- **sleep(int *m*)/sleep(int *m*, int *n*)**
  - The thread sleeps for *m* milliseconds, plus *n* nanoseconds.

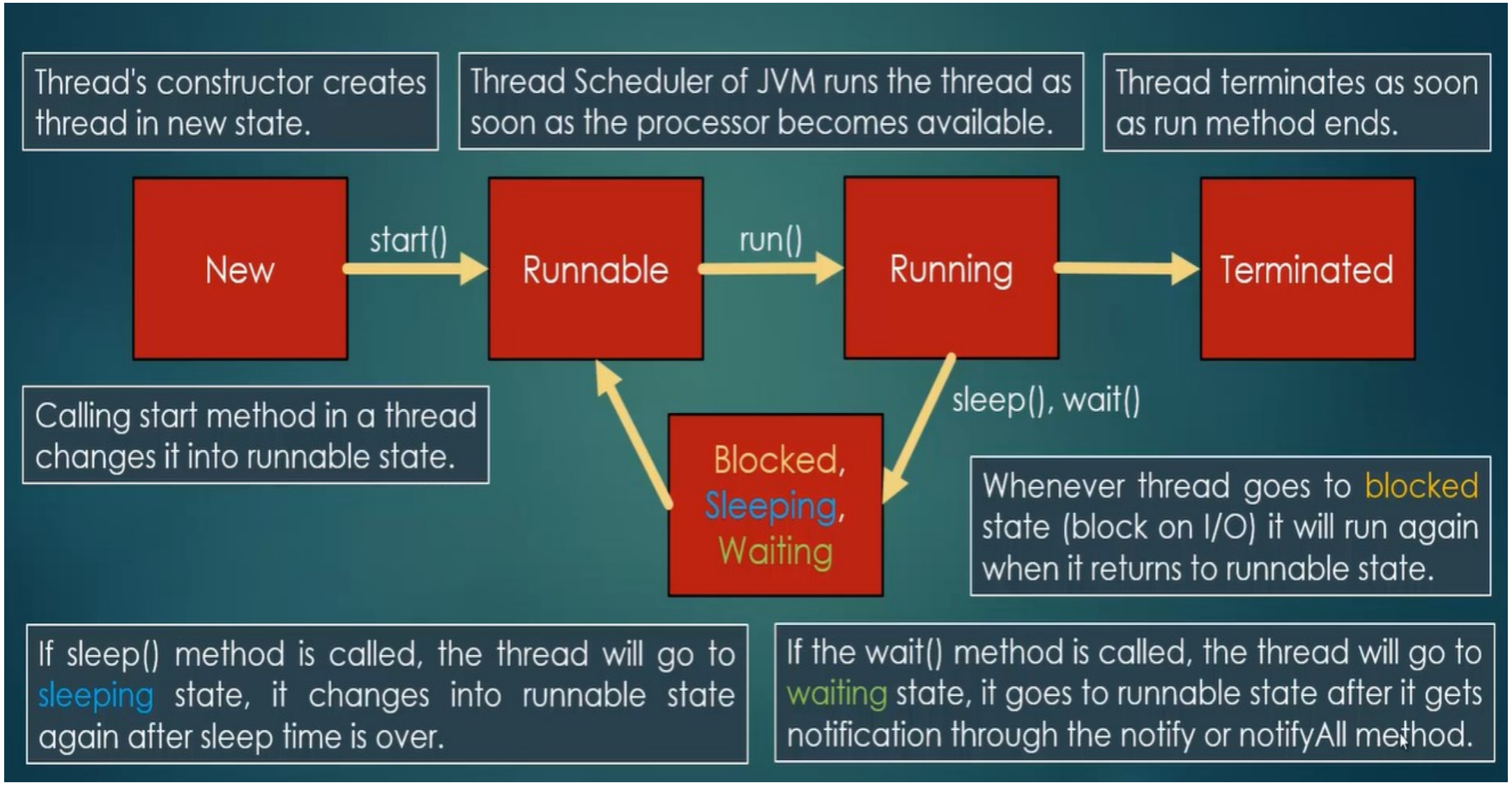
Life cycle of a Thread (Thread States)

- In Java, a thread always exists in any one of the following states. These states are:
  1. New State
  2. Runnable State
  3. Blocked State
  4. Waiting State
  5. Timed Waiting State
  6. Terminated State

Life cycle in a Thread



Life cycle in a Thread



Implementation of Thread States

- In Java, one can get the current state of a thread using the **Thread.getState()** method.
- The **java.lang.Thread.State** class of Java provides the constants ENUM to represent the state of a thread.
- The first state of a thread that is the NEW state, it is represented as: **public static final Thread.State NEW**
- Next state is runnable state, It means a thread is waiting in the queue to run. It is represented as: **public static final Thread.State RUNNABLE**.
- Next state is the blocked state. In this state, the thread is waiting to acquire a lock. It is represented as: **public static final Thread.State BLOCKED.**

- Next state is the waiting state. A thread will go to this state when it invokes the Object.wait() method or Thread.join() method with no timeout. It is represented as **public static final Thread.State TIMED_WAITING**.
- The main difference between waiting and timed waiting is the time constraint.
- The final state of a thread that is terminated or dead. A terminated thread means it has completed its execution. It is represented as **public static final Thread.State TERMINATED**.

How to create user defined thread in Java

- In Java, programmers create user-defined thread in two ways:
  - By inheriting from runnable interface
  - By inheriting from Thread class
- **Runnable Interface**
  - Runnable is a predefined interface declared in java.lang package.
  - It is implemented by a class intended to execute a thread.
  - Runnable interface has only one pre-defined method.

Thread Class

- Thread is a predefined class in java.lang package.
- JVM allows an application to have multiple threads of execution running concurrently.
- Thread class provides methods to create, start, pause, resume, and stop threads.
- These methods are start(), sleep(), yield(), join(), and interrupt() to manage the lifecycle of threads effectively.
- The run() method contains the code to be executed by the thread. It is not called directly, instead, call start() to begin execution of the thread, which internally calls the run() method.

What is thread-safe?

- “Thread-safe” refers to a program or piece of code that can be safely executed by multiple threads concurrently without causing unexpected behavior or data corruption.
- In other words, a thread-safe implementation ensures that shared data structures and resources are accessed and modified in a manner that preserves their integrity and consistency, even when accessed by multiple threads simultaneously.
- Achieving thread safety typically involves employing **synchronization mechanisms to coordinate access to shared resources**.

Benefits of Multithreading

1. In a multithreaded application, different parts of the application are executed by different threads.
2. Different threads are allocated to different processors and each thread is executed in different processors in parallel.
3. Reduces the computation time.
4. Improves the performance of an application.
5. Threads share the same address space and hence, it saves the memory.
6. Cost of communication between threads is relatively low.
7. Utilizes hardware resources in a better way.

Drawbacks of Multithreading

1. Increased complexity.
2. Synchronization of shared resources (objects, data)
3. Debugging is difficult and at times result is unpredictable.
4. Potential deadlocks.
5. Constructing and Synchronizing threads is CPU-Memory intensive.

Remember Important Points

1. Multithreading is a technique that allows a program or a process to executes many tasks concurrently (at the same time and in a parallel manner.
2. It allows a process to run its tasks in parallel mode on a single processor system.
3. Several lightweight processors are run in a single process by a single processor.
4. When a program contains multiple threads then the CPU can switch between the two thread to execute them at the same time.

Creating Thread using Runnable Interface

```
class SecondThreadExample implements Runnable
{
	public void run()
	{
		System.out.println("thread is running...");
		System.out.println(Thread.currentThread().getState());
	}

	public static void main(String args[])
	{
		Thread obj = new Thread(new SecondThreadExample());
		obj.start();
	}
}
```

Creating Thread using Thread Class

```
class FirstThreadExample extends Thread
{
	public void run()
	{
		System.out.println("thread is running...");
		System.out.println(Thread.currentThread().getState());
	}

	public static void main(String args[])
	{
		FirstThreadExample obj = new FirstThreadExample();
		obj.start();
	}
}
```

Main Thread

- Whenever we run a simple Java program, it is executed within the **main thread**.
- The **main thread** is created by **the JVM** to execute the main() method of any Java program.
- The main thread starts execution from main() method.
- User-defined thread starts execution from run() method.

Main Thread code

```
class MainThreadDemo{
    public static void main(String[] args){
        System.out.println(Thread.currentThread());
    }
  }
```

- currentThread() is a static method in Thread Class which is present in java.lang package.
- **Output:** Thread[main, 5, main]
- The output indicates that the currently executing thread is a Thread Class object, whose name is main, thread priority is 5 and it belongs to the main method.

Thread Priority

- Every thread has priority, which is an integer from 1 to 10.
- Based on priority, the concept is threads with higher priority should get preference over threads with lower priority.
- Priority is taken into account by the thread scheduler that decides which ready thread should be executed.
- The thread having the highest priority is chosen by the scheduler to be executed first.
- The default priority is 5.
- To set a thread’s priority, the setPriority() method is used. The getPriority() method returns priority of a the thread.

```
class TestThread implements Runnable{
        public static void main(String[] args){
                TestThread tt = new TestThread();
                Thread t1 = new Thread(tt, "Thread-1");
                Thread t2 = new Thread(tt, "Thread-2");
                Thread t3 = new Thread(tt, "Thread-3");
                t1.setPriority(Thread.MAX_PRIORITY);
                t2.setPriority(7);
                t3.setPriority(Thread.MIN_PRIORITY);
                System.out.println(t1.getPriority());
                System.out.println(t2.getPriority());
                System.out.println(t3.getPriority());
                t1.start();
                t2.start();
                t3.start();
        }
        public void run(){
                System.out.println(System.out.println(Thread.currentThread().getName()+" called
        “);
        }
}
```

Join() Method

- Join() method waits until the thread upon which is called is terminated.
- That means When the main thread calls join() on a thread object, the main thread will block and wait for that thread to finish.
- This method is useful in developing multithreading programming.
- The join method ensures that the main thread will wait till the created thread expires, in this scenario the sleep() method cannot give guarantee that the main thread will wail till termination of the created thread.
- When sleep() is called, the main thread is temporarily suspended from execution for the specified time duration.
- In summary, join() and sleep() serve different purposes in Java threading. join() is used for thread synchronization and coordination, while sleep() is used for introducing delays in thread execution.

```
class ThreadDemo implements Runnable{
    public void run(){
        System.out.println("Inside "+Thread.currentThread().getName());
    }
    public static void main(String[] args){
        ThreadDemo x = new ThreadDemo();
        Thread t = new Thread(x, "MyThread");
        t.start();
        System.out.println("Inside Main Thread");
    }
}
```

```
class Test implements Runnable{
   public void run(){
      System.out.println("Inside "+Thread.currentThread().getName());
   }
   public static void main(String[] args){
      Test x = new Test();
      Thread t = new Thread(x, "MyThread");
      t.start();
      try{
            t.join();
         }
         catch(InterruptedException ie){
             ie.printStackTrace();
         }
         System.out.println("Inside Main Thread");
    }
}
```