SOT
SCHOOL OF TECHNOLOGY

MULTITHREADING IN JAVA

Presented by:

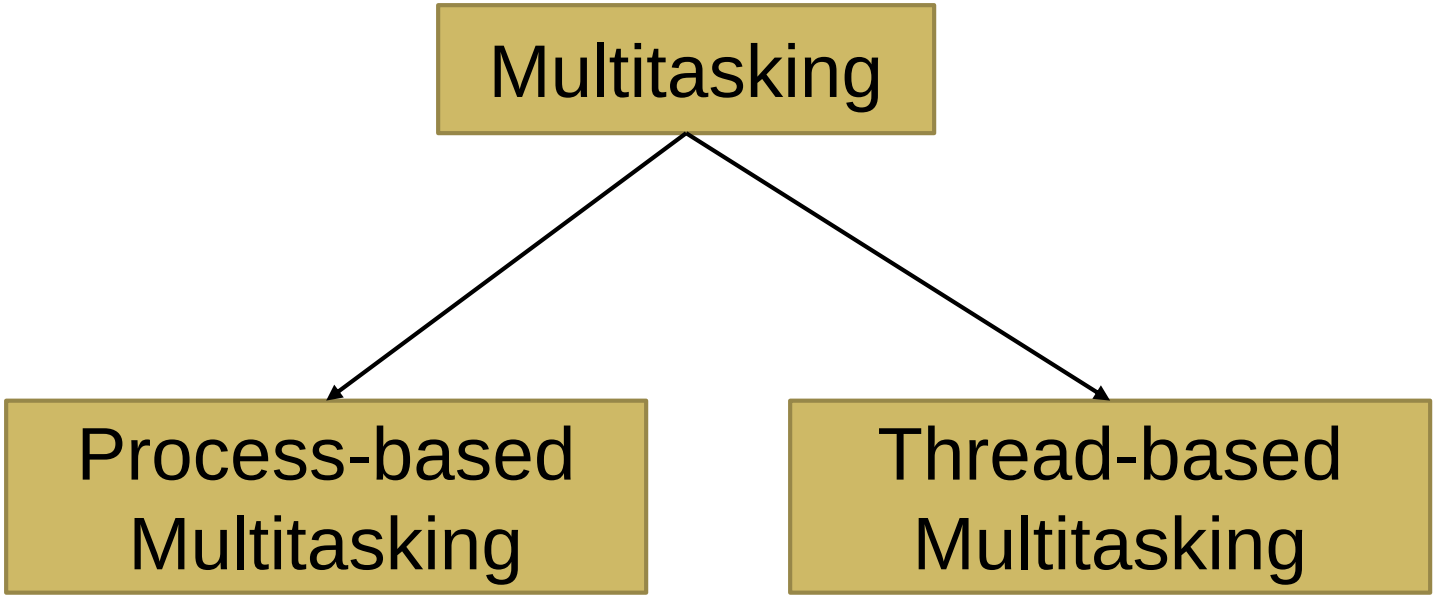
Dr. Shivangi K. Surati

Assistant Professor,
Department of Computer Science and Engineering,
School of Technology,
Pandit Deendayal Energy University

Outline

Multitasking

- Performing more than one tasks concurrently



Multitasking…

- Process-based
  - Allows computer to run two or more programs concurrently
  - The smallest unit of code that can be dispatched- Program
  - Ex- run Java compiler, visit a website, use a text editor
- Thread-based
  - Allows a single program to perform two or more tasks simultaneously
  - The smallest unit of code that can be dispatched- Thread
  - Ex- text editor – formatting the text file, printing of that file- these two processes by threads

Multitasking…

- Processes are heavyweight tasks – require separate address space, Interprocess Communication is expensive
- Threads are lightweight processes- executed in same address space of a single process, Interthhread Communication is inexpensive

Main Thread

```
public class ThreadsJava {

    public static void main(String[] args) {
        Thread thOb=Thread.currentThread();
        System.out.println("current thread: "+thOb);

        thOb.setName("Created thread");
        System.out.println("After Name Change:
"+thOb);
    }
}
```

Creating a Thread

(1) By extending Thread class and override run() method

(2) By implementing Runnable interface

Creating a Thread…

(1) By extending Thread class and override run() method.

Some of the methods:

getName(), setName(), start(), sleep(), join(), run(), isalive(), getPriority(), getId()

Creating a Thread-Example

(1) By extending Thread class and override run() method (Prog-1, Prog-2)

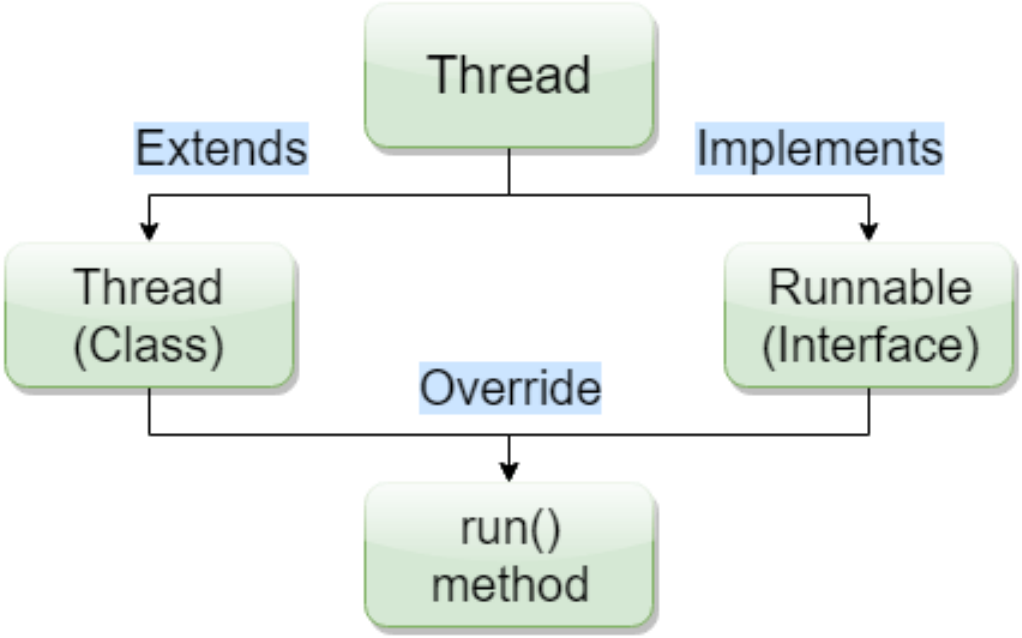
```
public class ThreadsTest extends Thread{
    public void run(){ //overrides run() method
        System.out.println("New thread");
    }
    public static void main(String[] args) {
        ThreadsTest thOb=new ThreadsTest();
        thOb.start();
    } }
```

Creating a Thread…

(2) By implementing Runnable interface (Prog-3, Prog-4, Prog-5)

```
public class ThreadsTest implements Runnable {
    public void run(){ //overrides run() method
        System.out.println("New thread");
    }
    public static void main(String[] args) {
        ThreadsTest thOb=new ThreadsTest();
        new Thread(thOb).start();     }}
```

Extends Thread v/s Implements Runnable

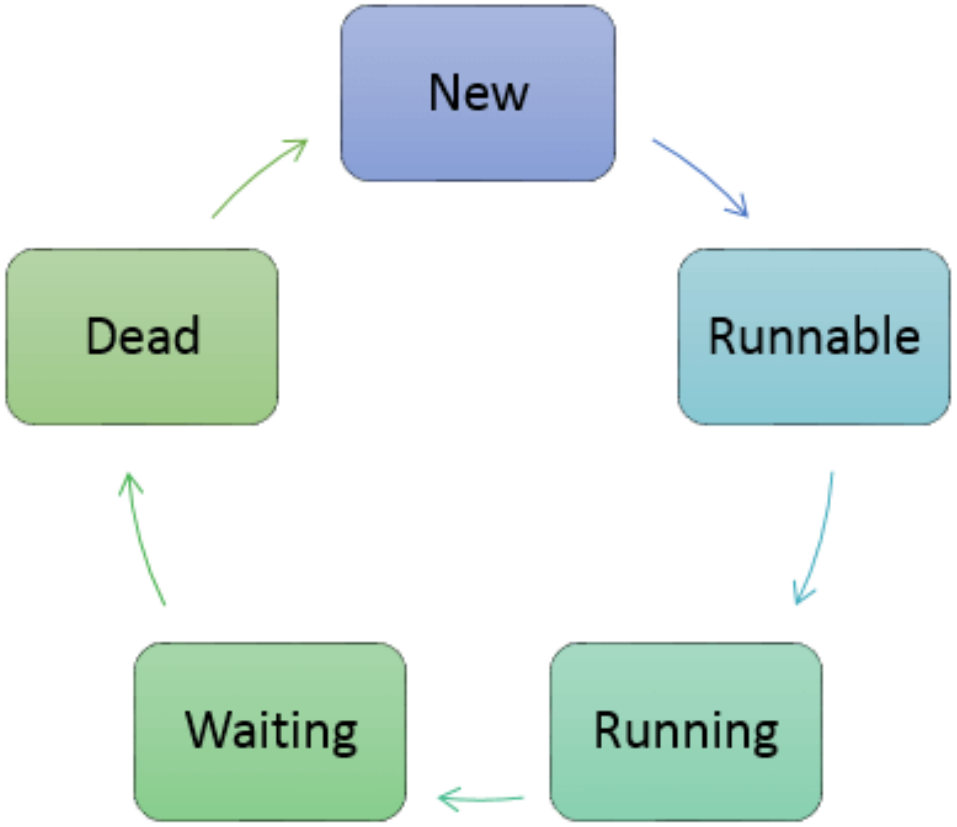


Source: https://www.javatpoint.com/runnable-interface-in-java

Extends Thread v/s Implements Runnable

Extends Thread	Implements Runnable
The class inherits all methods of parent Thread class – overhead of additional methods (consume excess or indirect memory, computation time, or other resources).	The class needs to override only abstract method run() provided by Runnable interface- less memory requirements
The class cannot extend (inherit) any other classes	The class can extend (inherit) the other class EX- class A extends B implements Runnable
The code will only be in a thread environment	More flexible code, so preferable for various executor services, EX- N/W programming (sockets)
Every thread creates a unique object and associates with it.	Multiple threads can share the same objects.

Thread Life Cycle



Thread Life Cycle…

- **New:** In this phase, the thread is created using class “Thread class”. It remains in this state till the program **starts** the thread. It is also known as born thread.
- **Runnable:** In this page, the instance of the thread is invoked with a start method. The thread control is given to scheduler to finish the execution. It depends on the scheduler, whether to run the thread.
- **Running:** When the thread starts executing, then the state is changed to “running” state. The scheduler selects one thread from the thread pool, and it starts executing in the application.

Thread Life Cycle…

- **Waiting:** This is the state when a thread has to wait. As multiple threads are running in the application, there is a need for synchronization between threads. Hence, one thread has to wait, till the other thread gets executed. Therefore, this state is referred as waiting state.
- **Dead:** This is the state when the thread is terminated. The thread is in running state and as soon as it completes the processing, it goes in “dead state”.

Creating Multiple Threads

- Prog-6: Creating multiple threads
- Prog-7: Creating multiple threads- with extending threads

isAlive() and join() method

- isAlive() method:
  - Checks whether the thread is still running or not
  - Returns true if the thread is running
- join() method:
  - the current thread stops its execution
  - the current thread goes into the wait state
  - until the thread on which the join() method is invoked has achieved its dead state
  - Calling thread waits until the specified thread joins it

Thread Priorities

- setPriority() method
- getPriority() method

Synchronization

- To control the access of multiple threads to any shared resource
- To allow only one thread to access the shared resource
- To prevent thread interference
- To prevent consistency problem
- Can be achieved through-
  - Mutual Exclusive
  - Inter-thread communication (Co-operation)

Synchronization through Mutual Exclusive

- To prevent interfering of the other threads while one thread uses shared resource
- Concept of Lock or monitor
- Every object has a lock with them
- Thread needs to acquire the object's lock to access it
- Only one thread is executing inside the object at a time- Synchronized Block
- Prog-8

Inter-thread Communication

- A mechanism in which a thread is paused running in its critical section
- Another thread is allowed to enter (or lock) in the same critical section for execution
- Polling- The process of testing a condition repeatedly till it becomes true
  - Usually implemented with the help of loops
  - Wastes many CPU cycles and makes the implementation inefficient
  - EX- Producer-Consumer

Producer-Consumer Problem

- The producer's job is to generate data, put it into the buffer, and start again.
- At the same time, the consumer is consuming the data (i.e. removing it from the buffer), one piece at a time.
  - **Problem**

  To make sure that the producer won't try to add data into the buffer if it's full and that the consumer won't try to remove data from an empty buffer.

Solution with multithreading

- **wait():** It tells the calling thread to give up the lock and go to sleep until some other thread enters the same monitor and calls notify().
- **notify():** It wakes up one single thread called wait() on the same object. It should be noted that calling notify() does not give up a lock on a resource.
- **notifyAll():** It wakes up all the threads called wait() on the same object.
- Prog-9

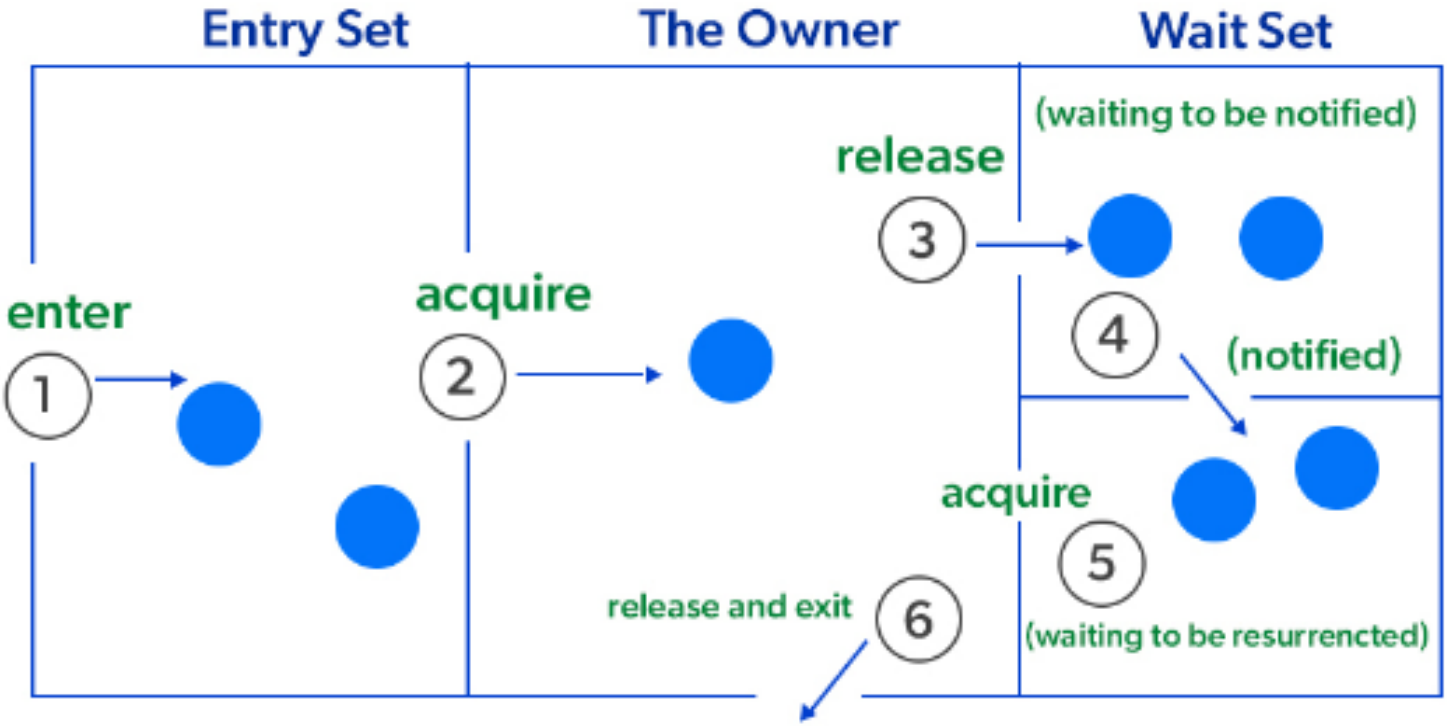


Image source: https://www.geeksforgeeks.org/inter-thread-communication-java/