# Java

## *Thread*

# *Multitasking*

- Multitasking allows several activities to occur concurrently on the computer.
- Levels of multitasking:
  - ***Process-based multitasking***
    - Allows programs (processes) to run concurrently.
  - ***Thread-base multitasking (multithreading)***
    - Allows parts of the same process (threads) to run concurrently.

# *Multitasking*

- Advantages of multithreading over process-based multitasking
  - Threads share the same address space
  - Context switching between threads is usually inexpensive
  - Communication between thread is usually inexpensive.
- Java supports thread-based multitasking and provides high-level facilities for multithreaded programming.

# *Main Thread*

- When a Java program starts up, one thread begins running immediately.
- This is called the main thread of the program.
- It is the thread from which the child threads will be spawned.
- Often, it must be the last thread to finish execution.
- **Example- MainThread.java**

# *How to create Thread*

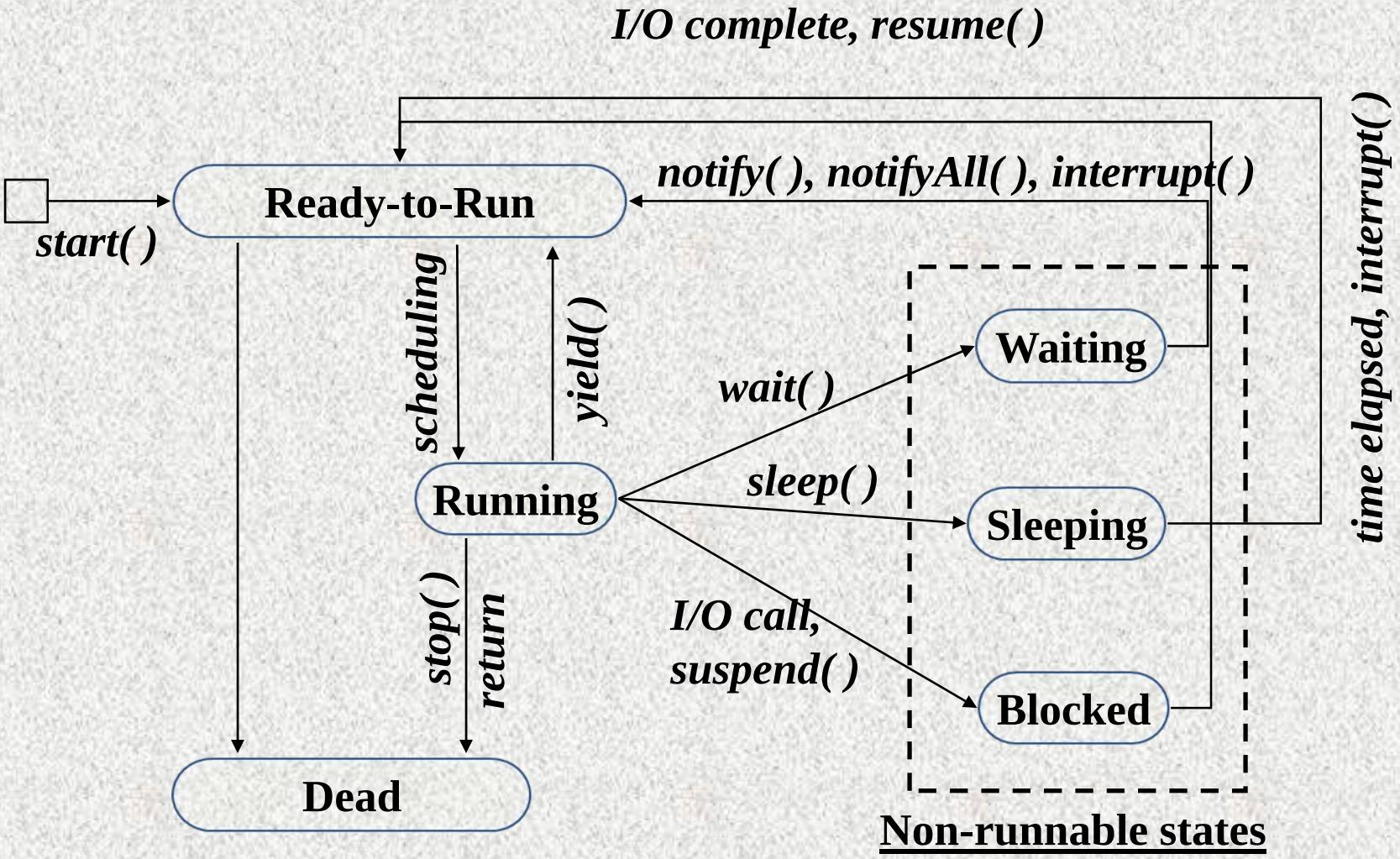
1. By implementing ***Runnable*** Interface
2. By extending the ***Thread*** class itself

- *Implementing Runnable*
  - Need to implement the public void run() method
  - **Example : RunnableThread.java**
- *Extends Thread*
  - Need to override the public void run() method
  - **Example: ExtendsThread.java**
- Which one is better ?

# *Multiple Threads*

- It is possible to create more than one thread inside the main. **Example: MultipleThreads.java**
- In multiple threads, often you will want the main thread to finish last. This is accomplished in the above program using a large delay in the main thread. But this can be done by using the *join()* method.
- Whether a thread has finished or not can be known using *isAlive()* method.
- **Example: JoinAliveThread.java**

# Thread States



# *Thread Priority*

- Threads can be assigned different priorities.
- Thread priorities are used by the thread scheduler to decide when each thread should be allowed to run.
- Theoretically higher priority threads get more CPU time than lower priority threads.
- To set a thread's priority the *setPriority(int level)* method is used. The level must be within the range MIN_PRIORITY(0) and MAX_PRIORITY(10). The default value is NORM_PRIORITY(5).
- **Example: ThreadPriority.java**

# *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.
- 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.

# *Synchronization*

- 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.
- **Example: NonSynchronized.java**

# *Synchronization*

- Two way to achieve synchronization.
  - *Synchronized method*

    ***synchronized*** void call(String msg) {}

    **Example: SynchronizedMethod.java**
  - *Synchronized block*

```
public void run()
 {
   synchronized(target)
  {
          target.call(msg);} }
```

    **Example: SynchronizedBlock.java**

# *Inter Thread Communication*

- Polling is usually implemented by a loop that is used to check some condition repeatedly. Once the condition is true, appropriate action is taken. This wastes CPU time.
- Java includes an elegant inter thread communication mechanism via the *wait( ), notify( )* and *notifyAll( )* methods.
- These methods are implemented as final methods in Object, so all classes have them.
- All three methods can be called only from within a synchronized method.

# *Inter Thread Communication*

- ***wait( )*** - tells the calling thread to give up the monitor and go to sleep until some other thread enters the same monitor and calls notify( ).
- ***notify( )*** - wakes up the first thread that called wait( ) on the same object.
- ***notifyAll( )*** - wakes up all the threads that called wait( ) on the same object. The highest priority thread will run first.
- Example: **IncorrectProducerConsumer.java, ProducerConsumer.java**

# *Suspend, Resume and Stop*

- Thread can be suspended, resumed and stopped.
- *Suspend*
  - Thread t;  t.suspend();
- *Resume*
  - Thread t; t.resume();
- *Stop*
  - Thread t; t.stop(); *Cannot be resumed later.*
- suspend and stop can sometimes cause serious system failures. So new way [Pg: 251]

***End***