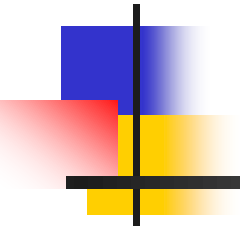
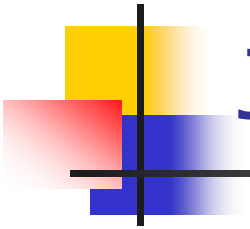
6. Java Threads





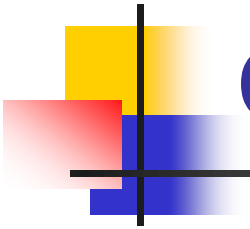
Java threads

- The Java Virtual Machine allows an application to have multiple threads of execution running concurrently.
- Java provides a Thread class:

  public class **Thread**

  extends Object

  implements Runnable

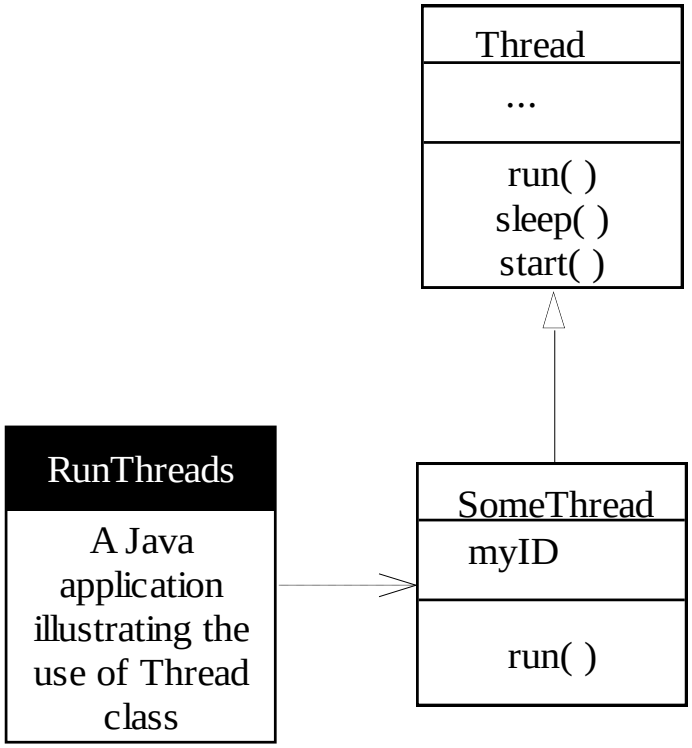


Creating a new Thread

- Using a subclass of the **Thread** class
- Using a class that implements the **Runnable** interface

Create a class that is a subclass of the Thread class

- Declare a class to be a subclass of **Thread**. This subclass should override the run method of class Thread. An instance of the subclass can then be allocated and started:



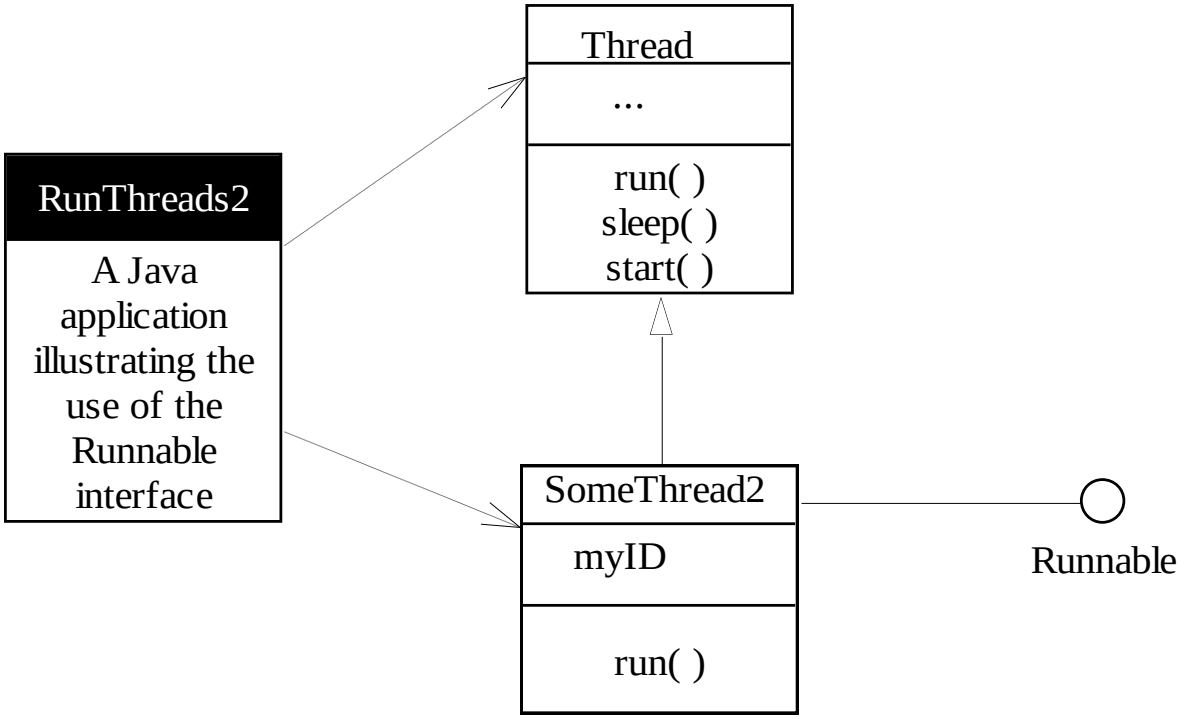
Create a class that is a subclass of the Thread class

```
import SomeThread;
public class RunThreads
{
   public static void main (String[] args)
   {
     SomeThread p1 = new SomeThread(1);
     p1.start();

     SomeThread p2 = new SomeThread(2);
     p2.start();

     SomeThread p3 = new SomeThread(3);
     p3.start();
   }
}// end class RunThreads
```

```
public class SomeThread extends Thread {
   int myID;

   SomeThread(int id) {
      this.myID = id;
   }

   public void run( ) {
      int i;
      for (i = 1; i < 11; i++)
         System.out.println ("Thread"+myID + ": " + i);
   }
} //end class SomeThread
```

Create a class that implements the Runnable interface

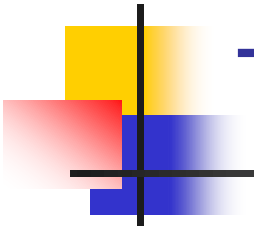
The other way to create a thread is to declare a class that implements the Runnable interface. That class then implements the run method. An instance of the class can then be allocated, passed as an argument when creating Thread, and started.



Create a class that implements the Runnable interface

```
public class RunThreads2
{
   public static void main (String[] args)
   {
     Thread p1 = new Thread(new SomeThread2(1));
     p1.start();

     Thread p2 = new Thread(new SomeThread2(2));
     p2.start();

     Thread p3 = new Thread(new SomeThread2(3));
     p3.start();
   }
}
```

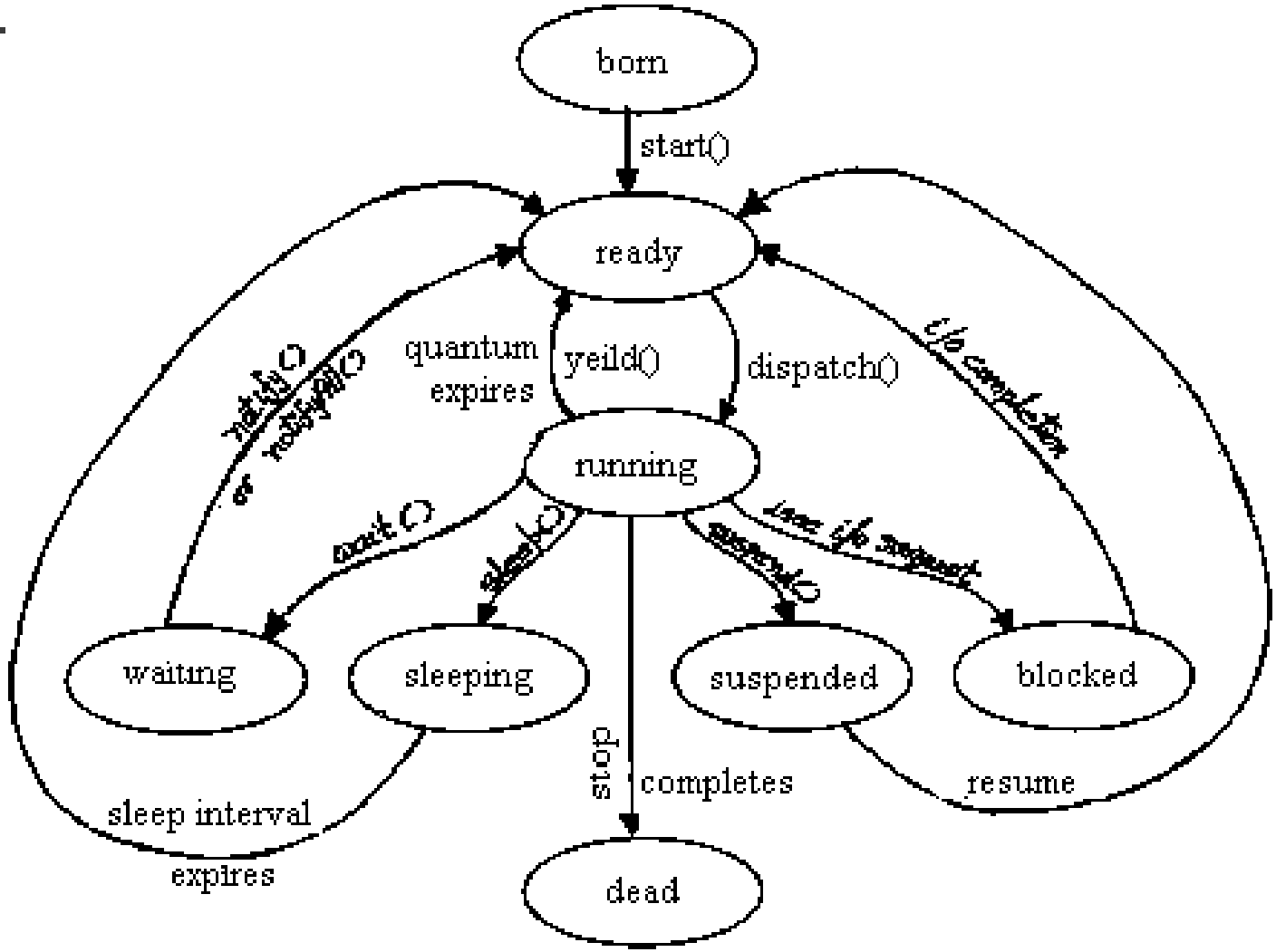
```
class SomeThread2 implements Runnable {
   int myID;

   SomeThread2(int id) {
      this.myID = id;
   }

   public void run() {
      int i;
      for (i = 1; i < 11; i++)
        System.out.println ("Thread"+myID + ": " + i);
   }
} //end class SomeThread
```



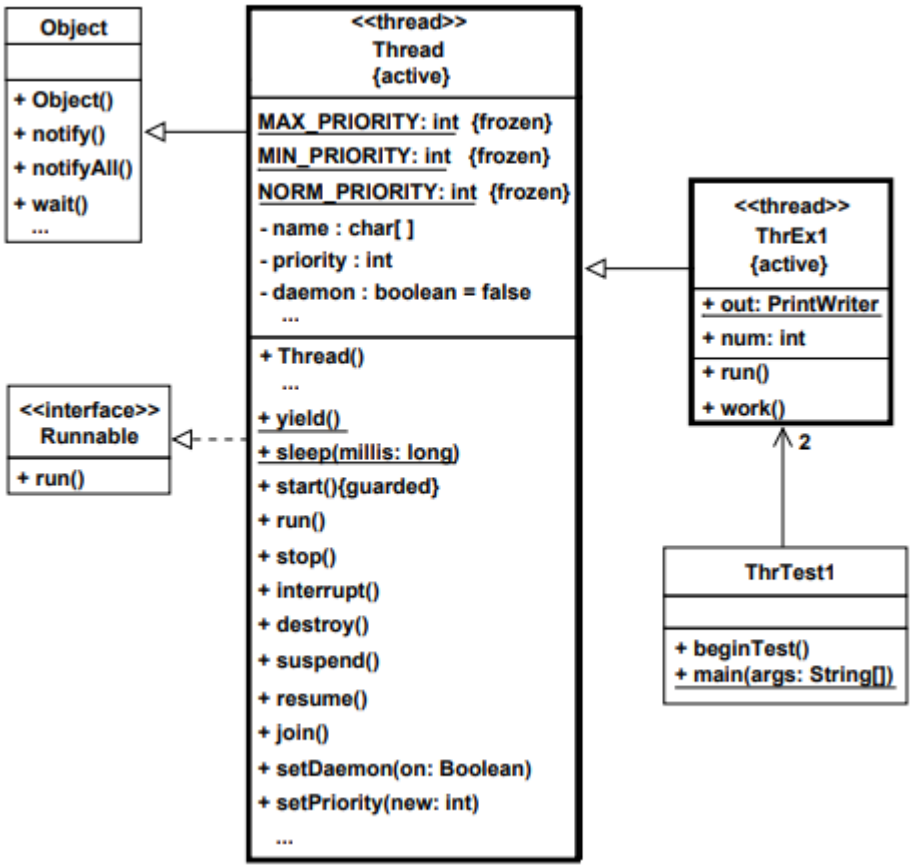
Thread Constructors

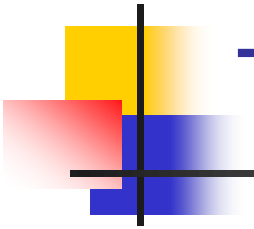
- **public Thread()**
- **public Thread(Runnable *target*)**
- **public Thread(String threadName)**

Life Cycle of a Thread



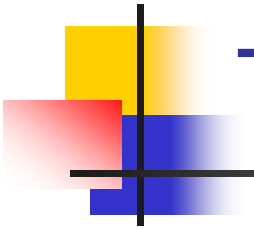
Multithread Program





Thread Operations

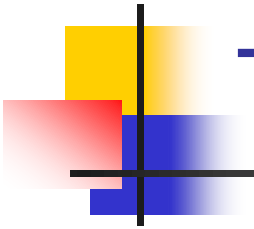
- ***Thread.sleep()*** and other methods that can pause a thread for periods of time can be interrupted. Threads can call another thread's ***interrupt()*** method, which signals the paused thread with an InterruptedException.



Thread Operations

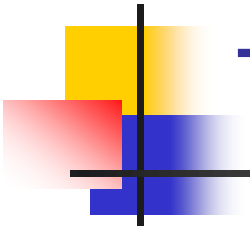
- ***yield():***
- It is used to give the other threads of the same priority a chance to execute. If other threads at the same priority are runnable, ***yield()*** places the calling thread in the running state into the runnable pool and allows another thread to run. If no other threads are runnable at the same priority, ***yield()*** does nothing.

- **Note:** When a thread completes execution and terminates, it cannot run again.



Thread Operations

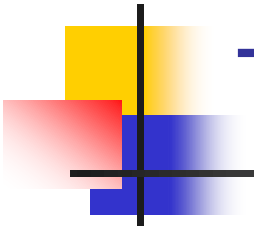
- Thread can be in unknown state. ***IsAlive()*** method is used to determine if a thread is till viable. The term alive does not imply that the thread is running; it returns true for a thread that has been started but not completed its task.
  - ***final boolean isAlive()***



Thread Operations

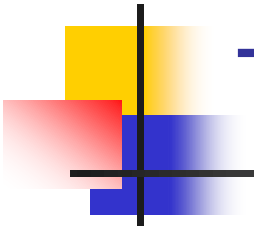
- The method that you will more commonly use to wait for a thread to finish is called join(), shown here:
  - ***final void join() throws InterruptedException***

- This method waits until the thread on which it is called terminates



Thread Priorities

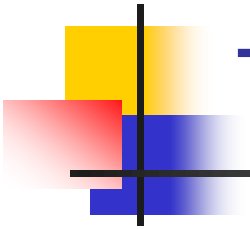
- Thread priorities are used by the thread scheduler to decide when each thread should be allowed to run. In theory, higher-priority threads get more CPU time than lower priority threads.
- Use defined constants to set priorities:
  - **MAX_PRIORITY, NORM_PRIORITY, MIN_PRIORITY**



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***.

- Key to synchronization is the concept of the monitor.



Thread synchronization - contd

- Declare the critical section as *synchronized.*

Eg.:-

public synchronized void update(int value) {… }

- Only one thread can execute a critical section at a time.
- Other threads wait until the current thread exits the critical section and notifies the waiting threads.