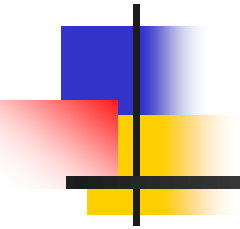
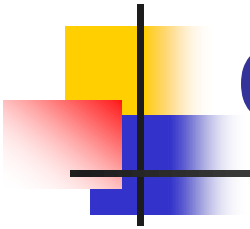
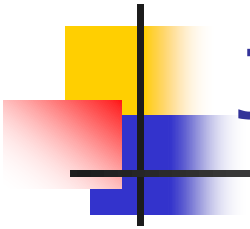
6. Java Threads





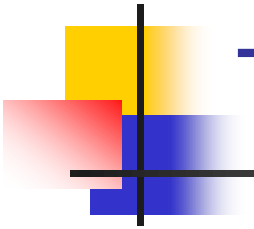
Operating systems basics

- A process consists of an executing program, its current values, state information, and the resources used by the operating system to manage its execution.
- A program is an artifact constructed by a software developer; a process is a dynamic entity which exists only when a program is run.



Java processes

- There are three types of Java program: applications, applets, and servlets, all are written as a class.
  - A Java application program has a main method, and is run as an independent(standalone) process.
  - An applet does not have a main method, and is run using a browser or the appletviewer.
  - A servlet does not have a main method, and is run in the context of a web server.
- A Java program is compiled into bytecode, a universal object code. When run, the bytecode is interpreted by the Java Virtual Machine (JVM).



Three Types of Java programs

- **Applications**

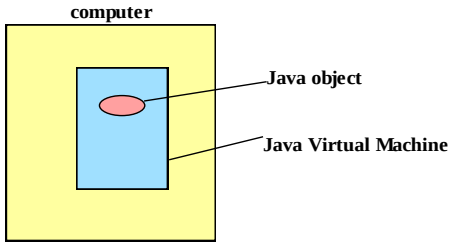
  a program whose byte code can be run on any system which has a Java Virtual Machine. An application may be standalone (monolithic) or distributed (if it interacts with another process).
- **Applets**

  A program whose byte code is downloaded from a remote machine and is run in the browser's Java Virtual Machine.
- **Servlets**

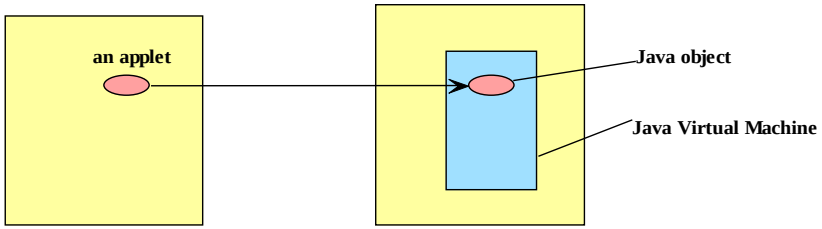
  A program whose byte code resides on a remote machine and is run at the request of an HTTP client (a browser).

Three Types of Java programs

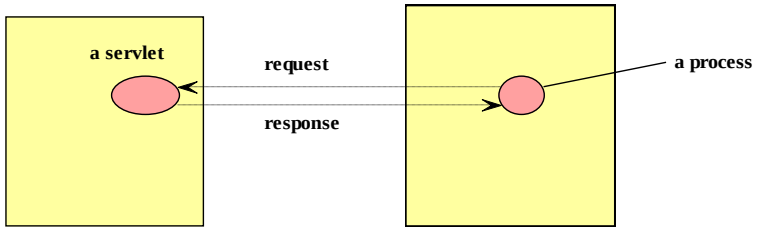
A standalone Java application is run on a local machine



An applet is an object downloaded (transferred) from a remote machine, then run on a local machine.



A servlet is an object that runs on a remote machine and interacts with a local program using a request-response protocol



A sample Java application

```
/*************************************************************
 *  A sample of a simple Java application.
 *   M. Liu              1/8/02
 *************************************************************/

import java.io.*;

class MyProgram{

   public static void main(String[ ] args)
      throws IOException{

      BufferedReader keyboard = new
         BufferedReader(new InputStreamReader(System.in));
      String theName;
      System.out.println("What is your name?");
      theName = keyboard.readLine( );
      System.out.print("Hello " + theName );
      System.out.println(" - welcome to CSC369.\n");

   } // end main

} //end class
```

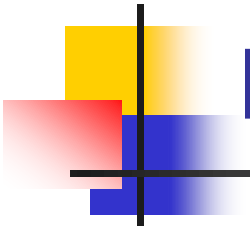
A Sample Java Applet

```
/*****************************************************
 *  A sample of a simple applet.
 *   M. Liu              1/8/02
 *****************************************************/

import java.applet.Applet;
import java.awt.*;

public class MyApplet extends Applet{

   public void paint(Graphics g){
      setBackground(Color.blue);

      Font Claude = new Font("Arial", Font.BOLD, 40);
      g.setFont(Claude);
      g.setColor(Color.yellow);
      g.drawString("Hello World!", 100, 100);
   } // end paint

} //end class
```

```
<!-- A web page which, when browsed, will run >
<!-- the MyApplet applet>
<!-- M. Liu 1/8/02>

<title>SampleApplet</title>
<hr>

<applet code="MyApplet.class" width=500 height=500>
</applet>

<hr>
<a href="Hello.java">The source.</a>
```

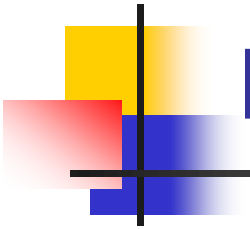
A Sample Java Servlet

```
/*********************************************************
 *  A sample of a simple Java servlet.
 *   M. Liu              1/8/02
 *********************************************************/
import java.io.*;
import java.text.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class MyServlet extends HttpServlet {

   public void doGet (HttpServletRequest request,
                      HttpServletResponse response)
      throws ServletException, IOException {

      PrintWriter out;
      String title = "MyServlet Output";
      // set content type and other response header
      //   fields first
      response.setContentType("text/html");
      // then write the data of the response
      out = response.getWriter();
      out.println("<HTML><HEAD><TITLE>");
      out.println(title);
      out.println("</TITLE></HEAD><BODY>");
      out.println("<H1>" + title +   "</H1>");
      out.println("<P>Hello World!");
      out.println("</BODY></HTML>");
      out.close();
   } //end doGet

} //end class
```



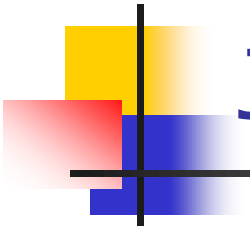
Definition of Thread

- A multithreaded program contains two or more parts that can run concurrently. Each part of such a program is called a thread, and each defines a separate path of execution.



How Threads are useful?

- Better utilization of system resources:
  - Another line of execution can grab the CPU when one line of execution is blocked
- Multiple threads solve numerous problems better:
  - single threaded program cannot be written to show animation, play music, display documents, and download files from the network at the same time



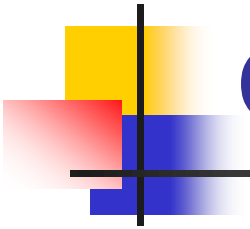
Java threads

- 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

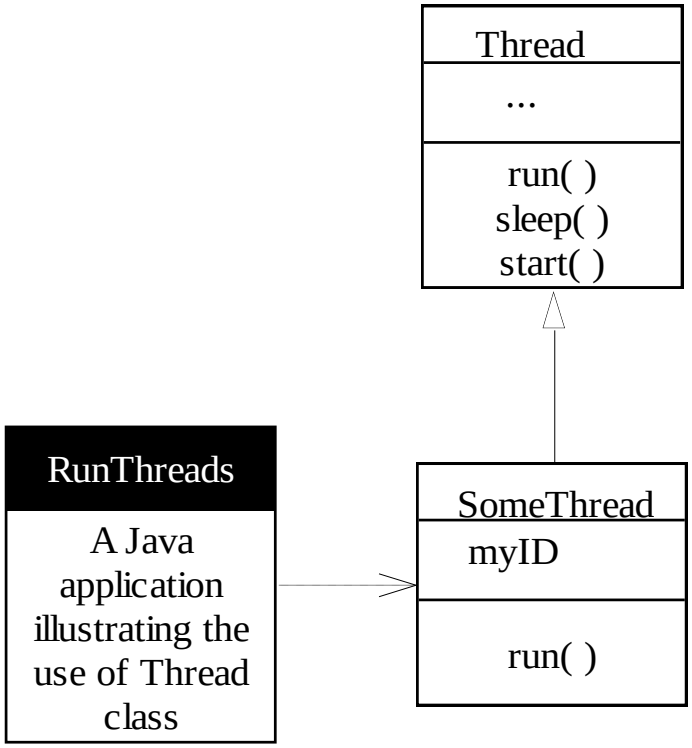


Creating a new Thread

- 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:



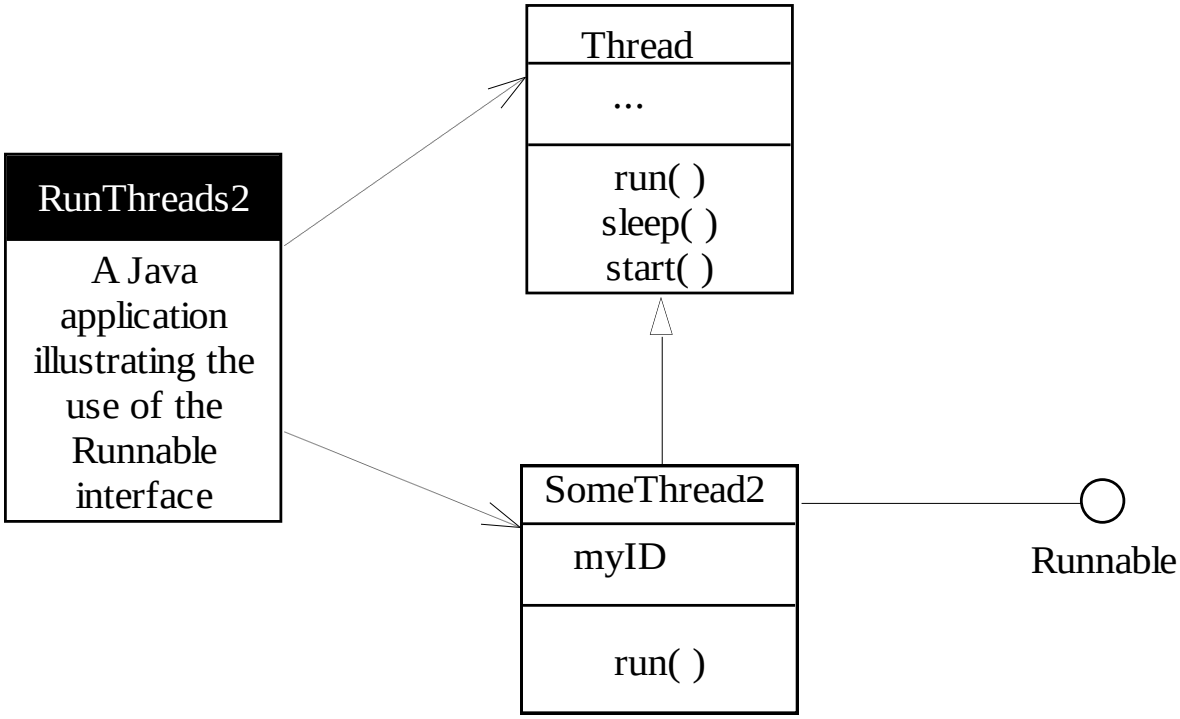
Create a class that is a subclass of the Thread class

```
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

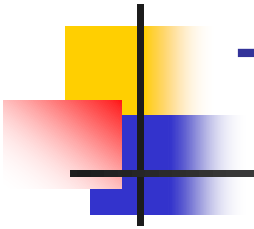
The other way to create a thread is to declare a class that implements the Runnable interface. That class then implements the run method. An instance of the class can then be allocated, passed as an argument when creating Thread, and started.



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();
   }
}
```

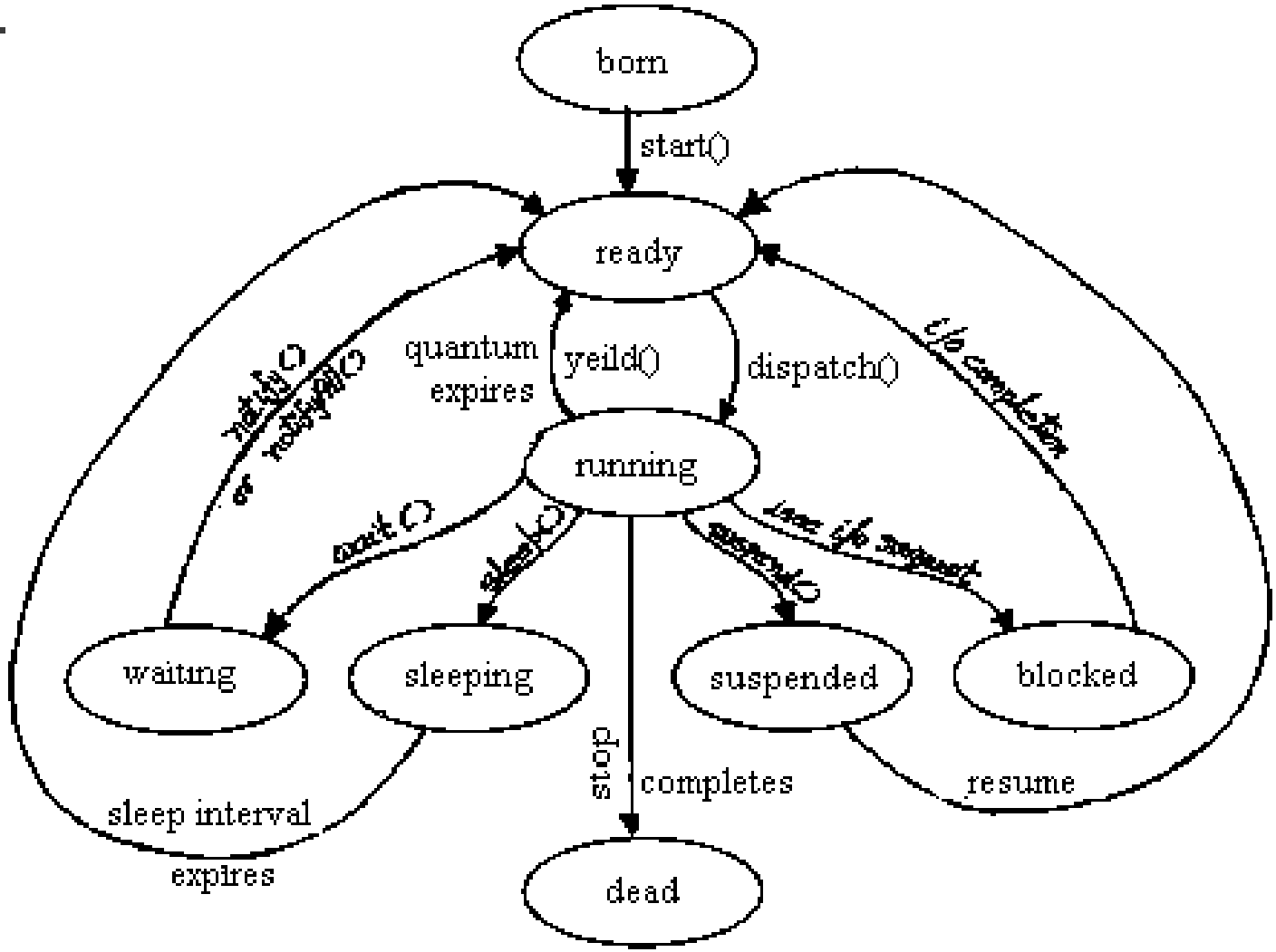
```
class SomeThread2 implements Runnable {
   int myID;

   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
```



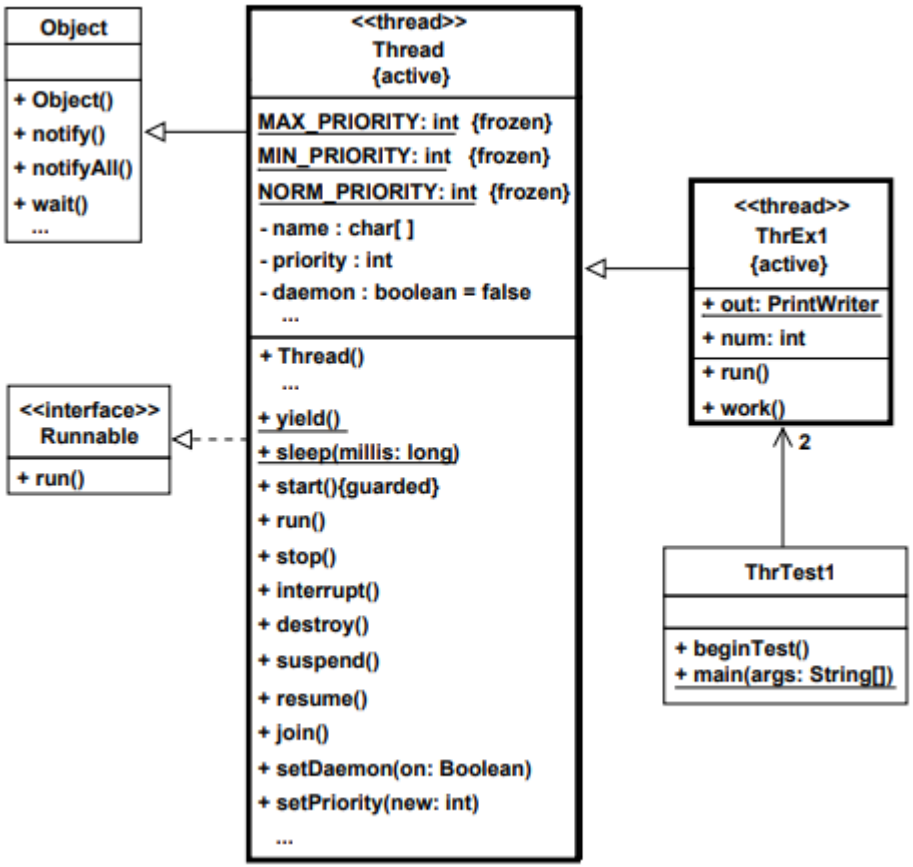
Thread Constructors

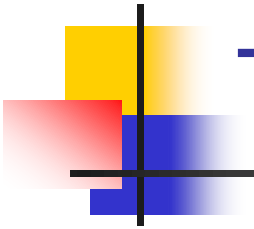
- **public Thread()**
- **public Thread(Runnable *target*)**
- **public Thread(String threadName)**

Life Cycle of a Thread



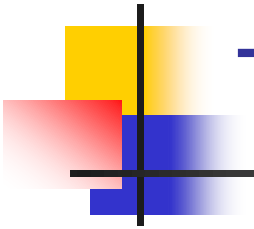
Multithread Program





Thread Operations

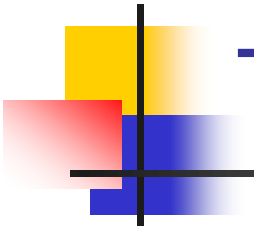
- ***Thread.sleep()*** and other methods that can pause a thread for periods of time can be interrupted. Threads can call another thread's ***interrupt()*** method, which signals the paused thread with an InterruptedException.



Thread Operations

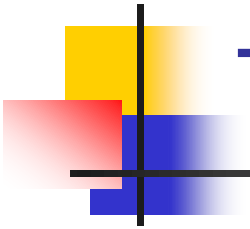
- ***yield():***
- It is used to give the other threads of the same priority a chance to execute. If other threads at the same priority are runnable, ***yield()*** places the calling thread in the running state into the runnable pool and allows another thread to run. If no other threads are runnable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

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



Thread Operations

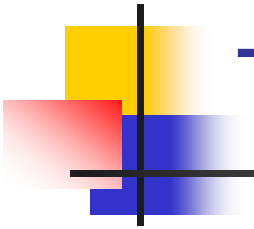
- Thread can be in unknown state. ***IsAlive()*** method is used to determine if a thread is till viable. The term alive does not imply that the thread is running; it returns true for a thread that has been started but not completed its task.
  - ***final boolean isAlive()***



Thread Operations

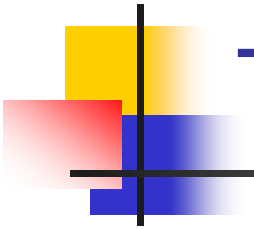
- The method that you will more commonly use to wait for a thread to finish is called join(), shown here:
  - ***final void join() throws InterruptedException***

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



Thread Priorities

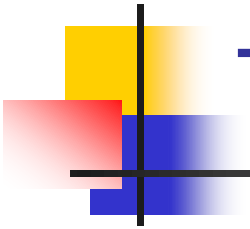
- Thread priorities are used by the thread scheduler to decide when each thread should be allowed to run. In theory, higher-priority threads get more CPU time than lower priority threads.
- Use defined constants to set priorities:
  - **MAX_PRIORITY, NORM_PRIORITY, MIN_PRIORITY**



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.