Bharathidasan University

Centre for Differently Abled Persons

Khajamalai Campus
Tiruchirappalli-620 023
Tamilnadu

Bachelor of Computer Applications
For Students with Speech and Hearing Impairment)

Course: Java Programming

Unit-5

Compiled By
Dr.M.Prabavathy
(Assistant Professor)
Ms.V.Vijayalakshmi

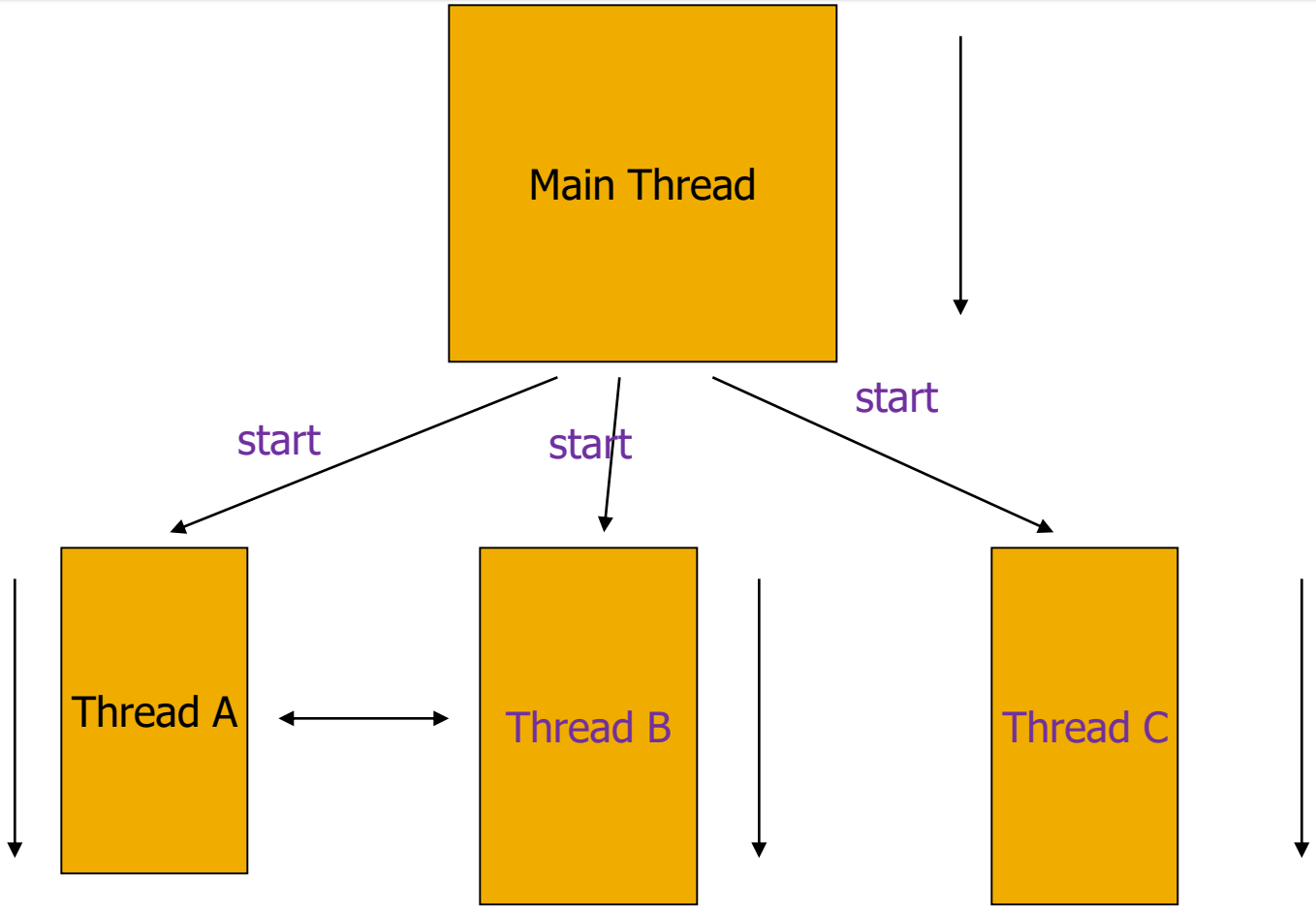
- Multithreaded Programming in Java

- Introduction
- Thread Applications
- Defining Threads
- Java Threads and States
- Examples

A single threaded program

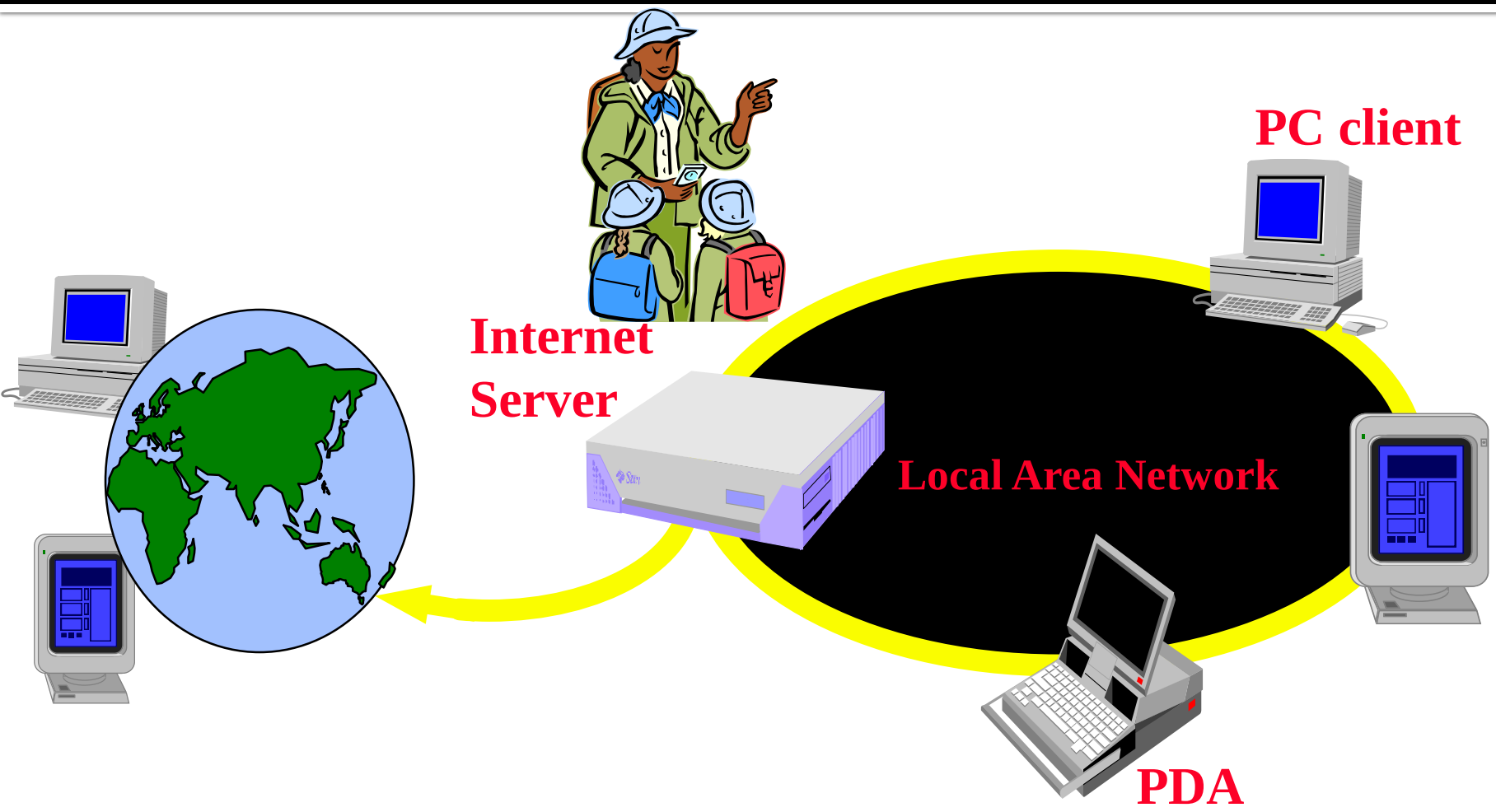
```
class ABC
{
....
    public void main(..)
    {
    ...
    ..
    }
}
```

A Multithreaded Program

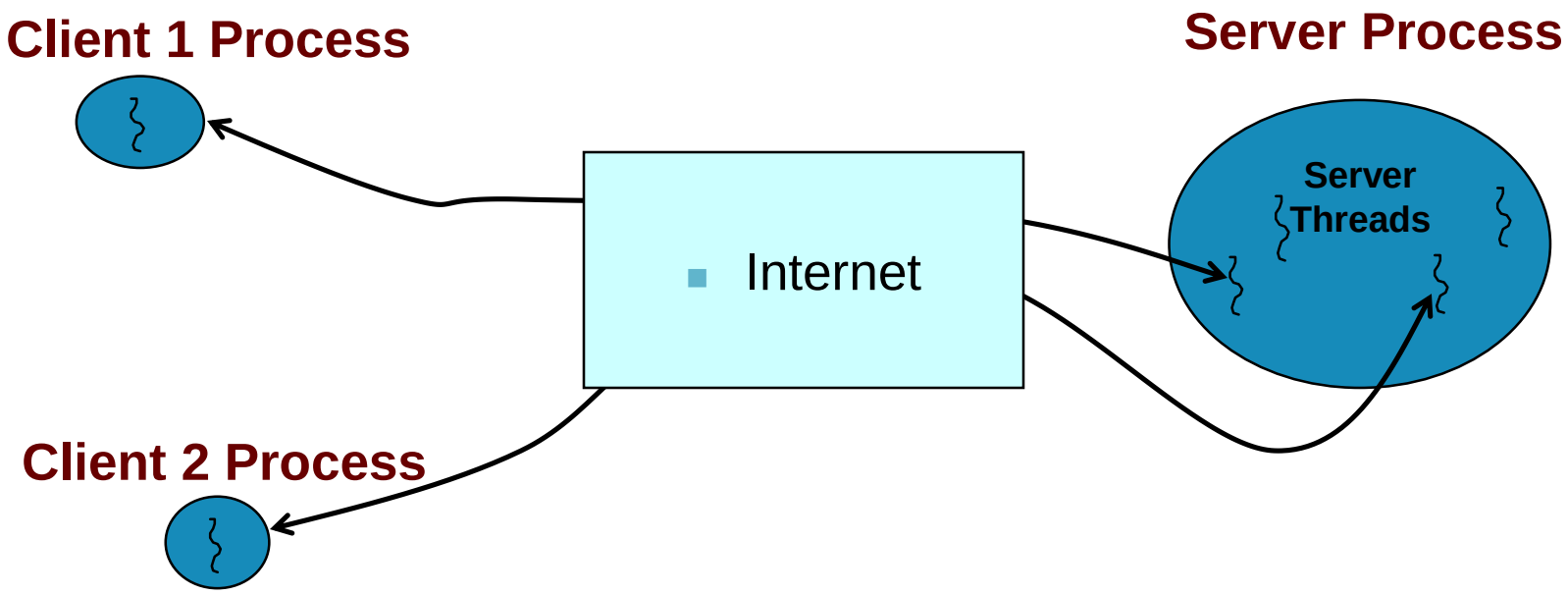


Threads may switch or exchange data/results

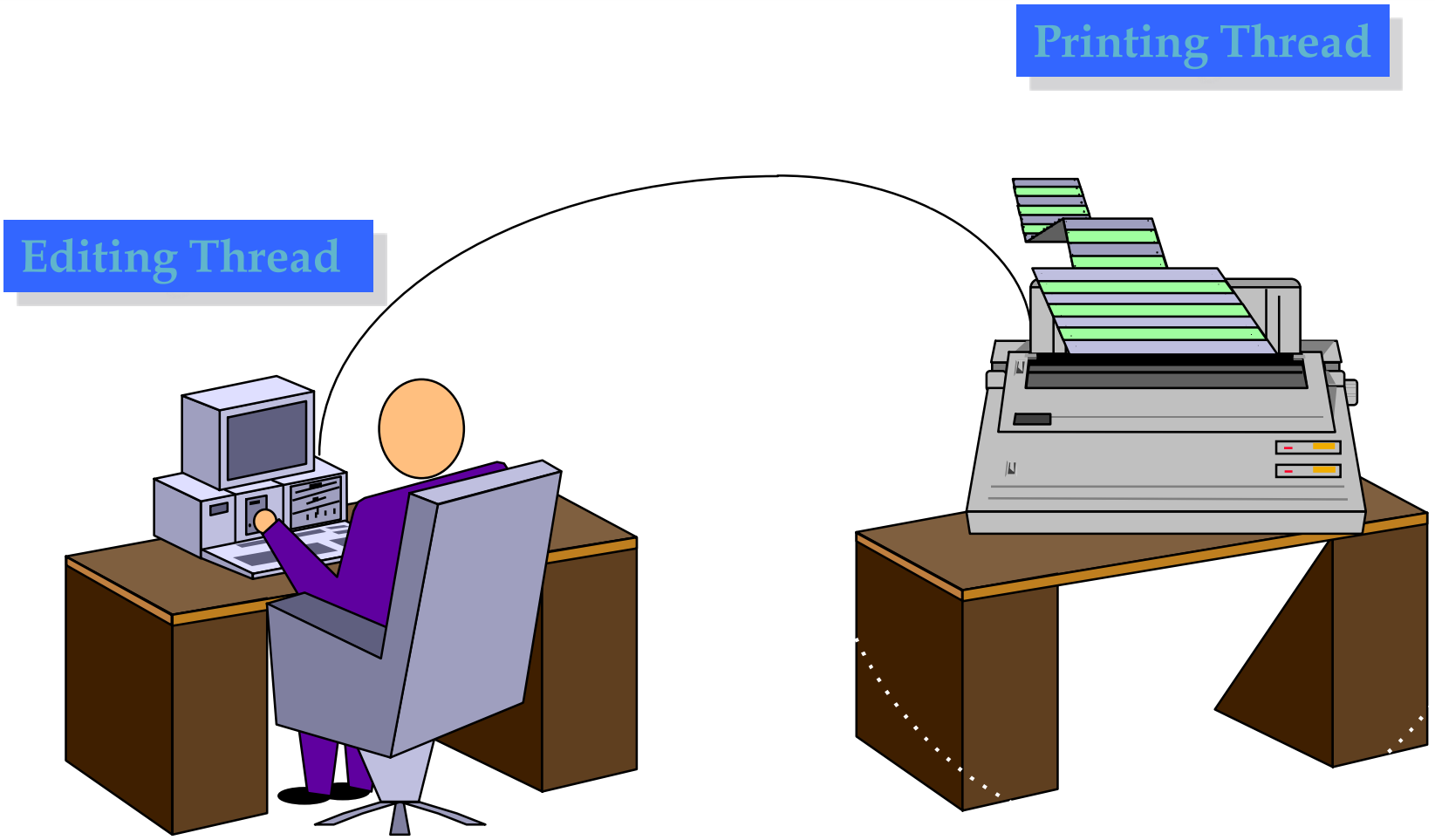
Web/Internet Applications: Serving Many Users Simultaneously



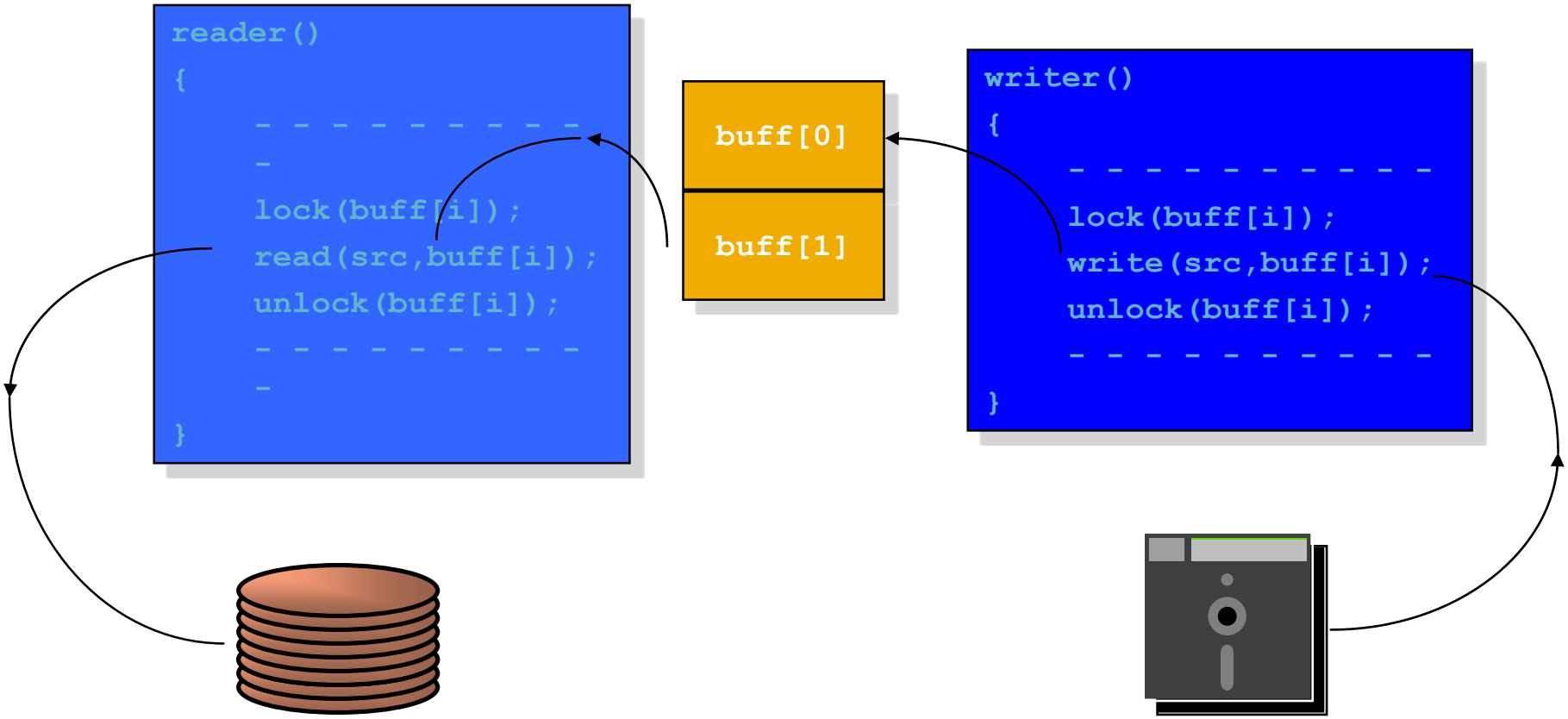
Multithreaded Server: For Serving Multiple Clients Concurrently



Modern Applications need Threads (ex1): Editing and Printing documents in background.



Multithreaded/Parallel File Copy



Cooperative Parallel Synchronized Threads

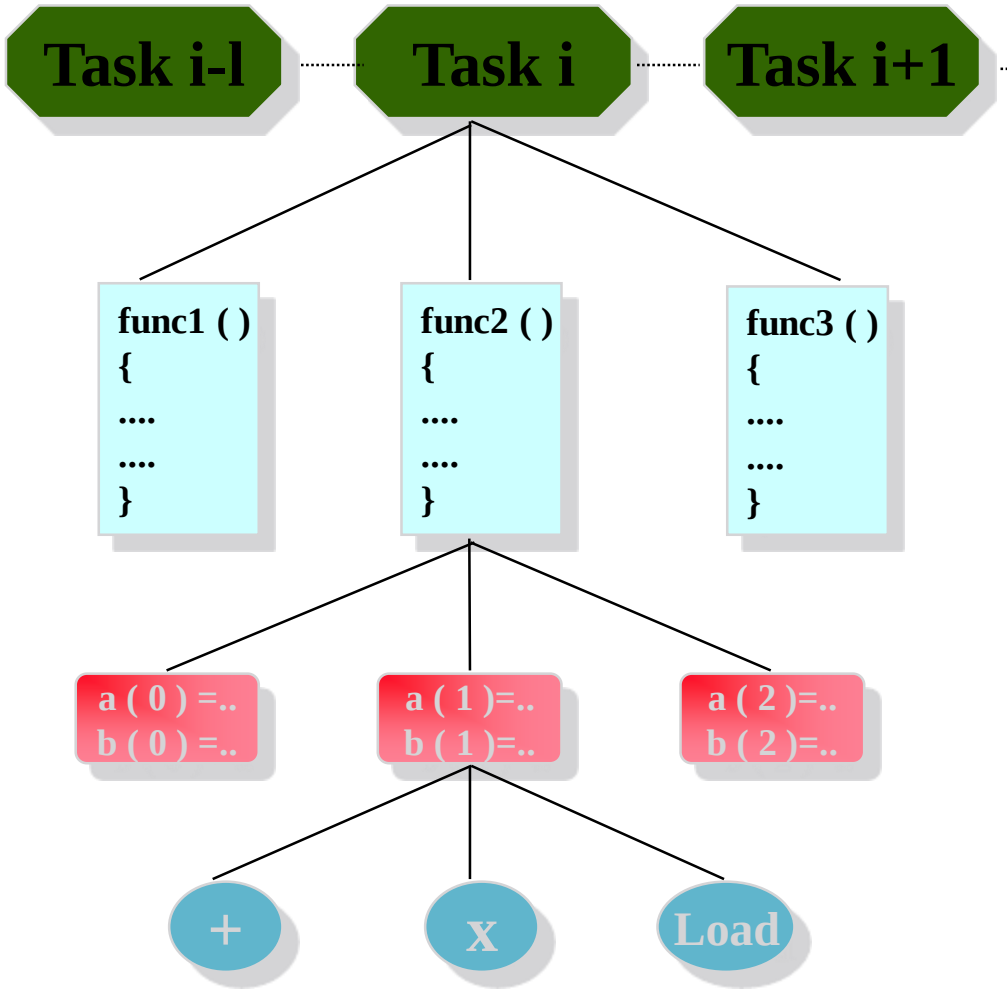
Levels of Parallelism

Sockets/
PVM/MPI

Threads

Compilers

CPU



Code-Granularity

Code Item

Large grain
(task level)
Program

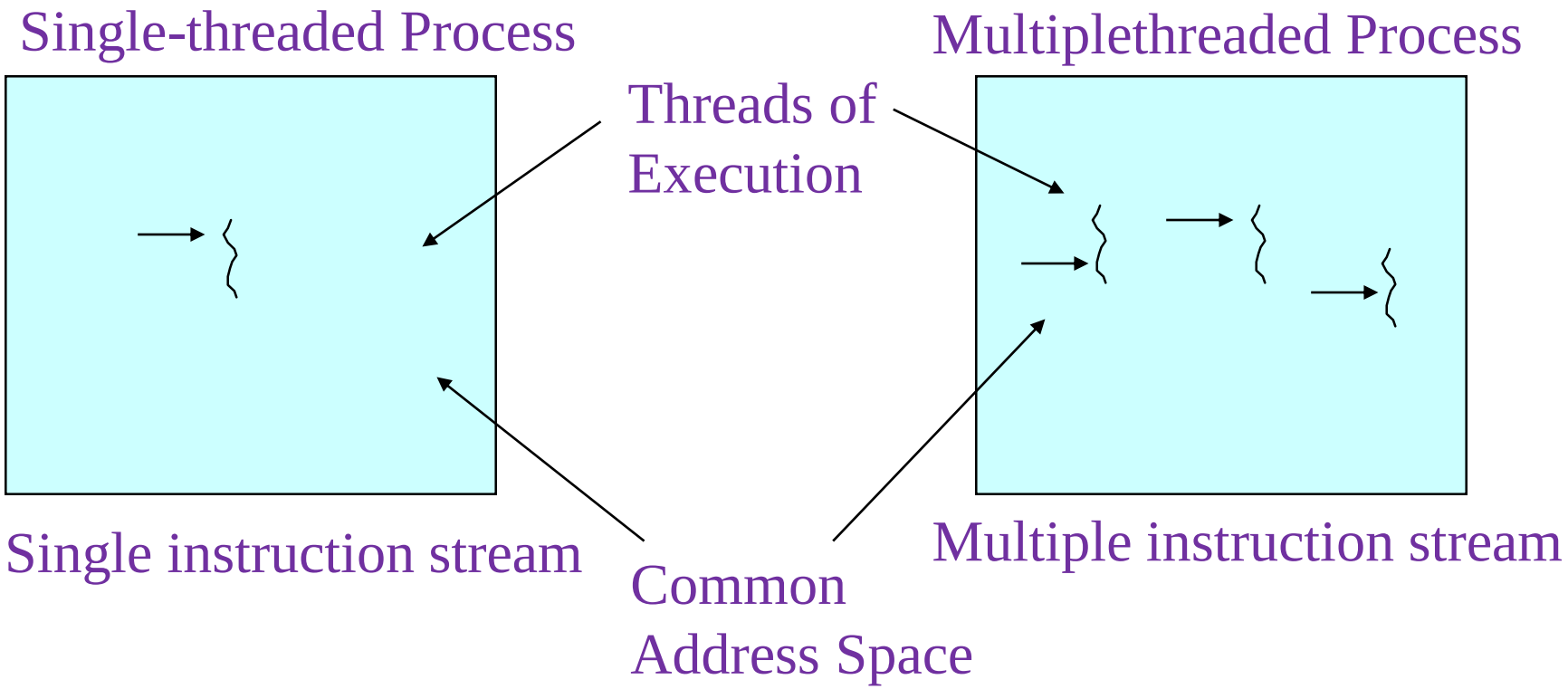
Medium grain
(control level)
Function (thread)

Fine grain
(data level)
Loop (Compiler)

Very fine grain
(multiple issue)
With hardware

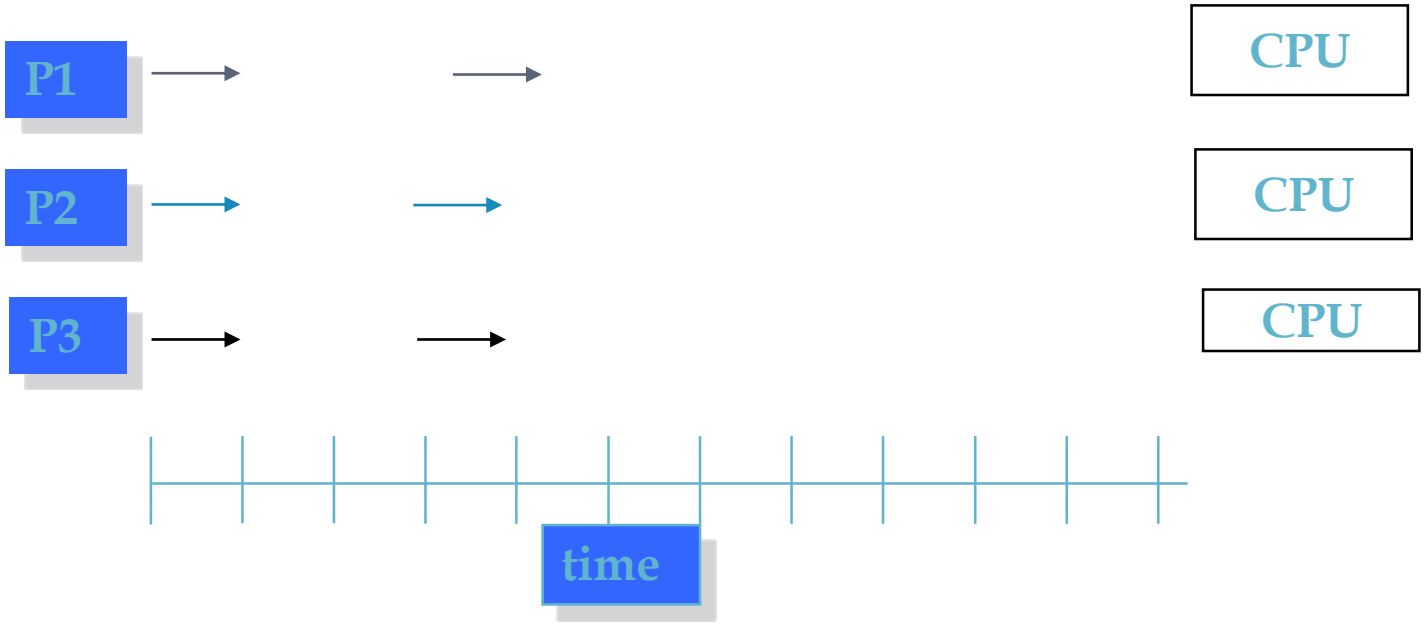
Single and Multithreaded Processes

Threads are light-weight processes within a process



Multithreading - Multiprocessors

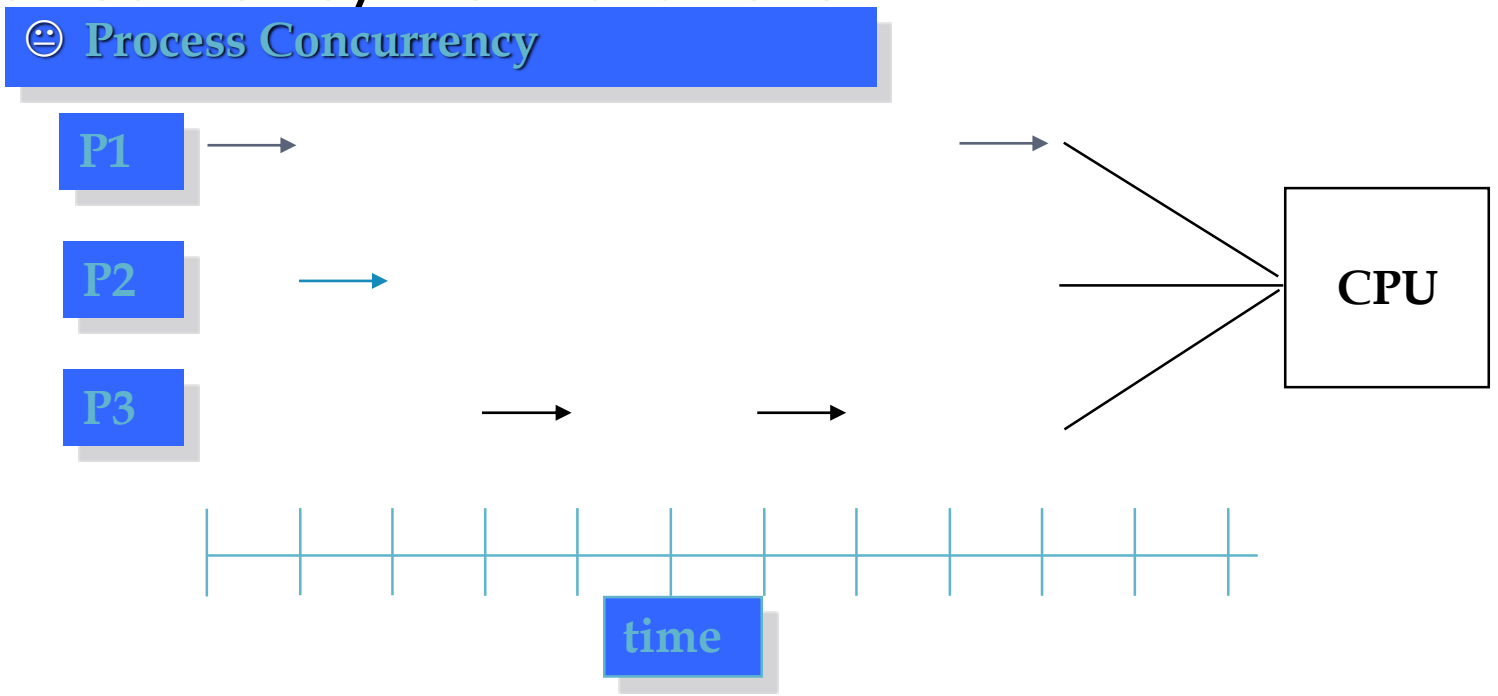
Process Parallelism



No of execution process more the number of CPUs

Multithreading on Uni-processor

- Concurrency Vs Parallelism



Number of Simultaneous execution units > number of CPUs

What are Threads?

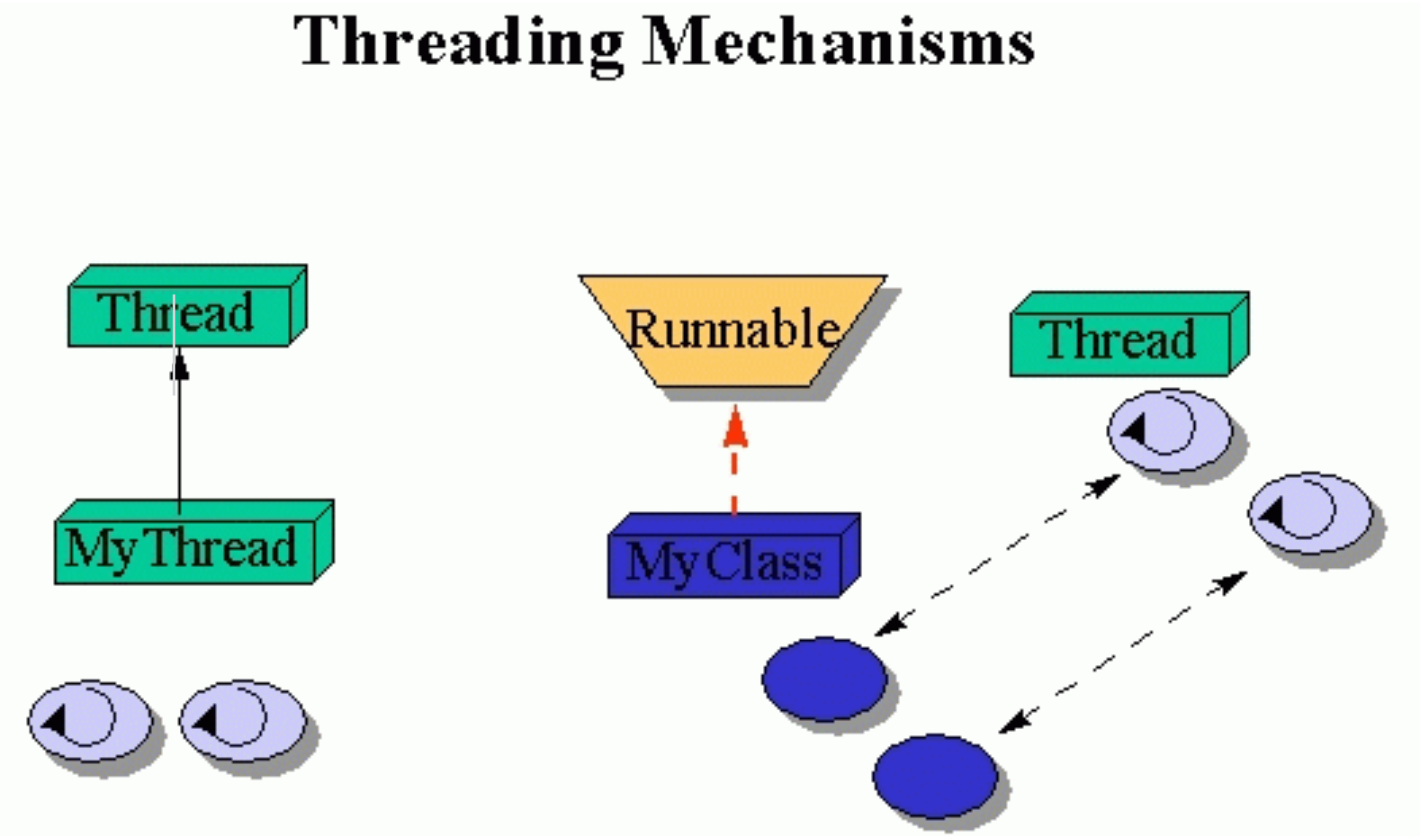
- A piece of code that run in concurrent with other threads.
- Each thread is a statically ordered sequence of instructions.
- Threads are being extensively used express concurrency on both single and multiprocessors machines.
- Programming a task having multiple threads of control – Multithreading or Multithreaded Programming.

Java Threads

- Java has built in thread support for Multithreading
- Synchronization
- Thread Scheduling
- Inter-Thread Communication:
  - currentThread
  - yield
  - sleep
  - resume
  - start
  - run
  - stop
  - setPriority
  - getPriority
  - suspend
- Java Garbage Collector is a low-priority thread

Threading Mechanisms...

- Create a class that extends the Thread class
- Create a class that implements the Runnable interface



1st method: Extending Thread class

- Threads are implemented as objects that contains a method called run()

```
class MyThread extends Thread
{
   public void run()
   {
      // thread body of execution
   }
}
```

- Create a thread:

```
MyThread thr1 = new MyThread();
```

- Start Execution of threads:

```
thr1.start();
```

An example

```
class MyThread extends Thread {
// the thread
      public void run() {
            System.out.println(" this thread is running ... ");
      }
}
// end class MyThread

class ThreadEx1 {
// a program that utilizes the thread
      public static void main(String [] args  ) {

MyThread t = new MyThread();
            // due to extending the Thread class (above)
            // I can call start(), and this will call
            // run(). start() is a method in class Thread.

 t.start();
     } // end main()
}     // end class ThreadEx1
```

2nd method: Threads by implementing Runnable interface

```
class MyThread implements Runnable
{
   .....
   public void run()
   {
      // thread body of execution
   }
}
```

- Creating Object:

```
MyThread myObject = new MyThread();
```

- Creating Thread Object:

```
Thread thr1 = new Thread( myObject );
```

- Start Execution:

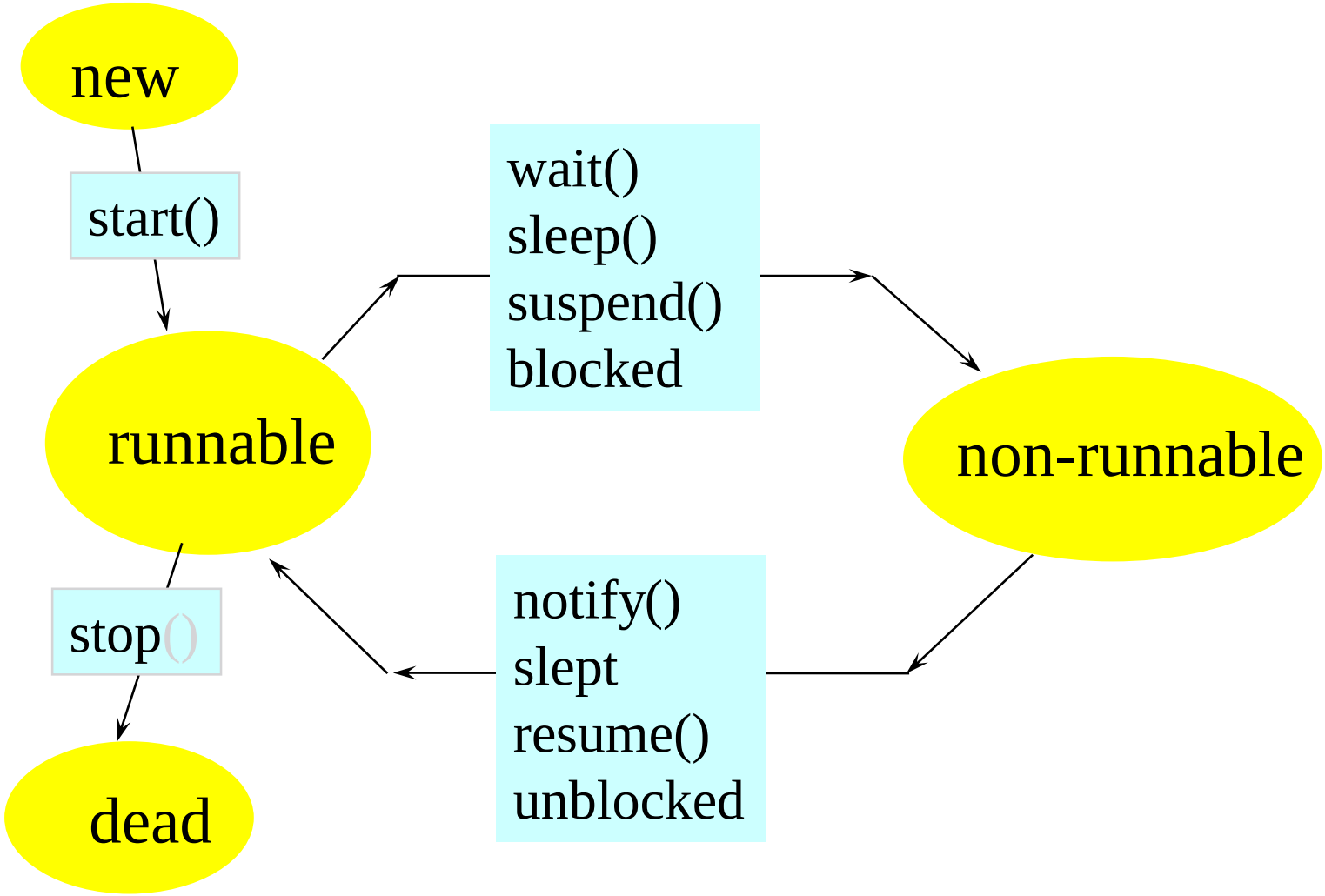
```
thr1.start();
```

An example

```
class MyThread implements Runnable  {
      public void run() {
            System.out.println(" this thread is running ... ");
      }
} // end class MyThread

class ThreadEx2 {
      public static void main(String [] args  ) {
            Thread t = new Thread(new MyThread());
                  // due to implementing the Runnable interface
                  // I can call start(), and this will call run().
            t.start();
      } // end main()
}     // end class ThreadEx2
```

Life Cycle of Thread



A Program with Three Java Threads

- Write a program that creates 3 threads

Three threads example

```
class A extends Thread
{
    public void run()
     {
        for(int i=1;i<=5;i++)
         {
            System.out.println("\t From ThreadA: i= "+i);
         }

          System.out.println("Exit from A");
     }

}

class B extends Thread
{
    public void run()
     {

        for(int j=1;j<=5;j++)
         {
            System.out.println("\t From ThreadB: j= "+j);
         }

          System.out.println("Exit from B");
     }

}
```

```
class C extends Thread
{
    public void run()
     {

        for(int k=1;k<=5;k++)
         {
            System.out.println("\t From ThreadC: k= "+k);
         }

          System.out.println("Exit from C");
     }

}


class ThreadTest
{
      public static void main(String args[])

       {
            new A().start();
            new B().start();
            new C().start();

       }


}
```

Run 1

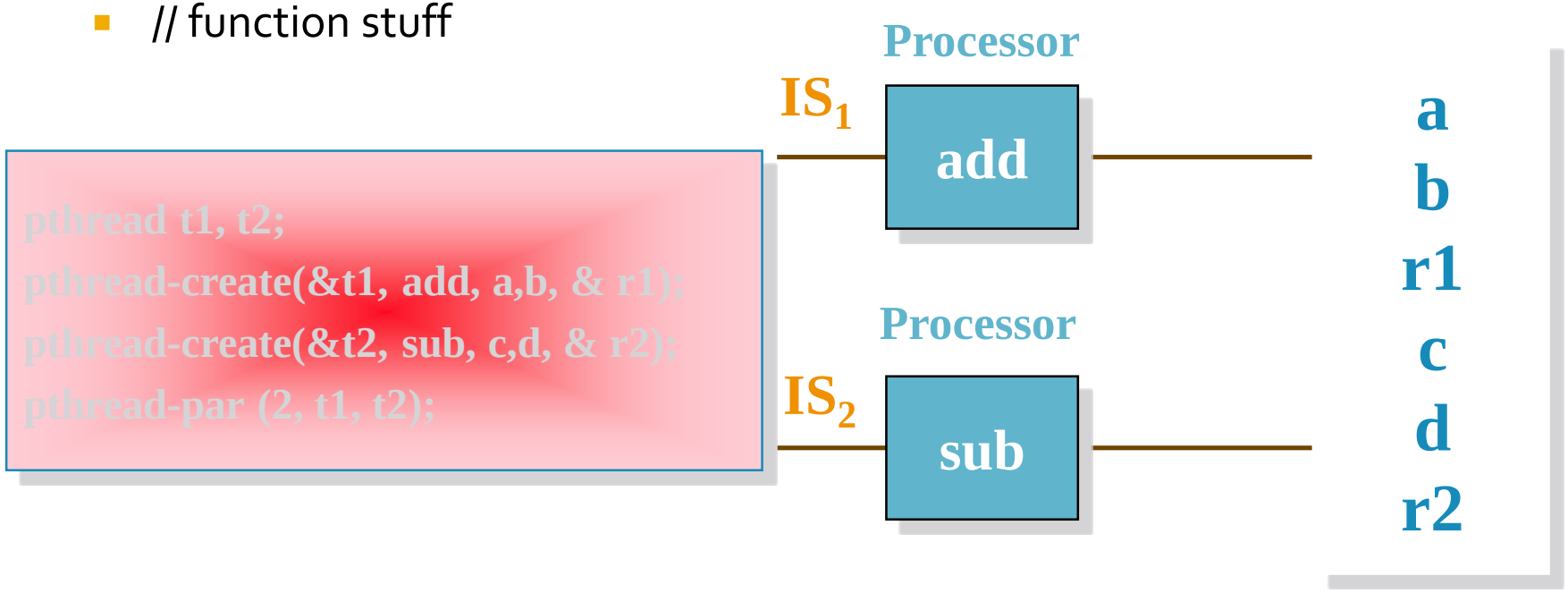
- [raj@mundroo] threads [1:76] java ThreadTest
  From ThreadA: i= 1
  From ThreadA: i= 2
  From ThreadA: i= 3
  From ThreadA: i= 4
  From ThreadA: i= 5
Exit from A
  From ThreadC: k= 1
  From ThreadC: k= 2
  From ThreadC: k= 3
  From ThreadC: k= 4
  From ThreadC: k= 5
Exit from C
  From ThreadB: j= 1
  From ThreadB: j= 2
  From ThreadB: j= 3
  From ThreadB: j= 4
  From ThreadB: j= 5
Exit from B

Run2

- [raj@mundroo] threads [1:77] java ThreadTest
  From ThreadA: i= 1
  From ThreadA: i= 2
  From ThreadA: i= 3
  From ThreadA: i= 4
  From ThreadA: i= 5
  From ThreadC: k= 1
  From ThreadC: k= 2
  From ThreadC: k= 3
  From ThreadC: k= 4
  From ThreadC: k= 5
Exit from C
  From ThreadB: j= 1
  From ThreadB: j= 2
  From ThreadB: j= 3
  From ThreadB: j= 4
  From ThreadB: j= 5
Exit from B
Exit from A

Process Parallelism

- int add (int a, int b, int & result)
- // function stuff
- int sub(int a, int b, int & result)
- // function stuff

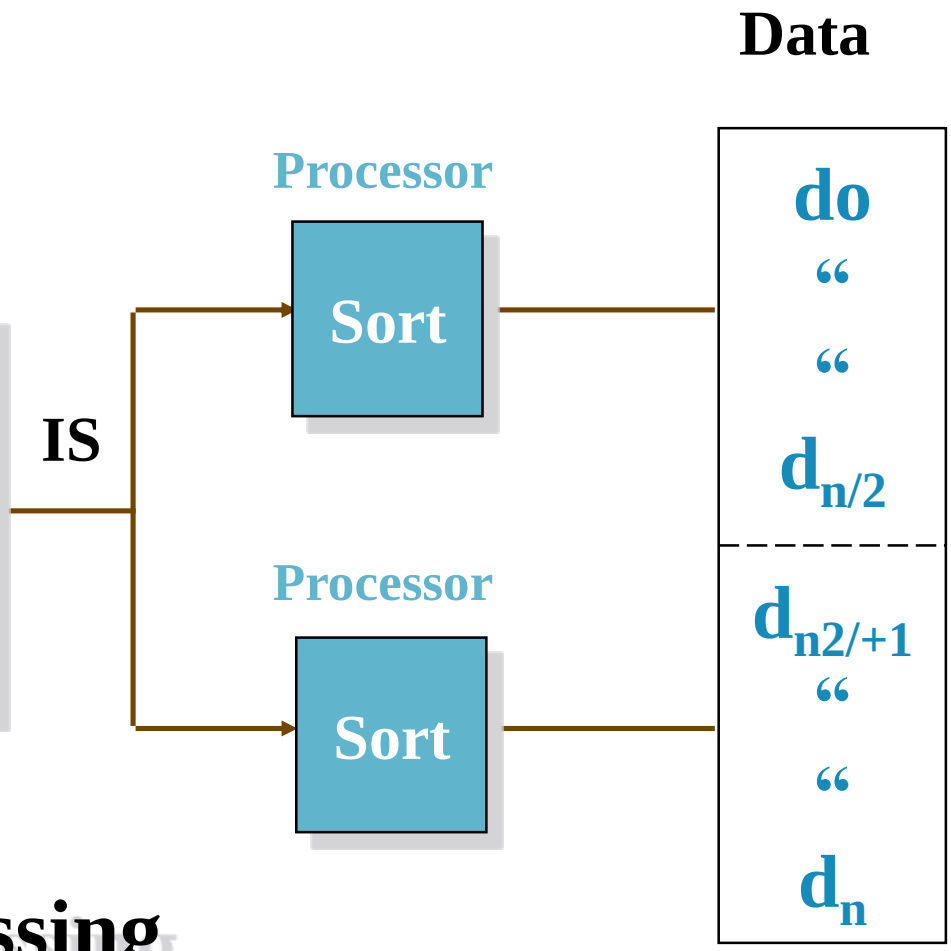


MISD and MIMD Processing

Data Parallelism

- sort(int *array, int count)
- //.....
- //.....

```
pthread-t, thread1, thread2;
“
“
pthread-create(& thread1, sort, array, N/2);
pthread-create(& thread2, sort, array, N/2);
pthread-par(2, thread1, thread2);
```



SIMD Processing

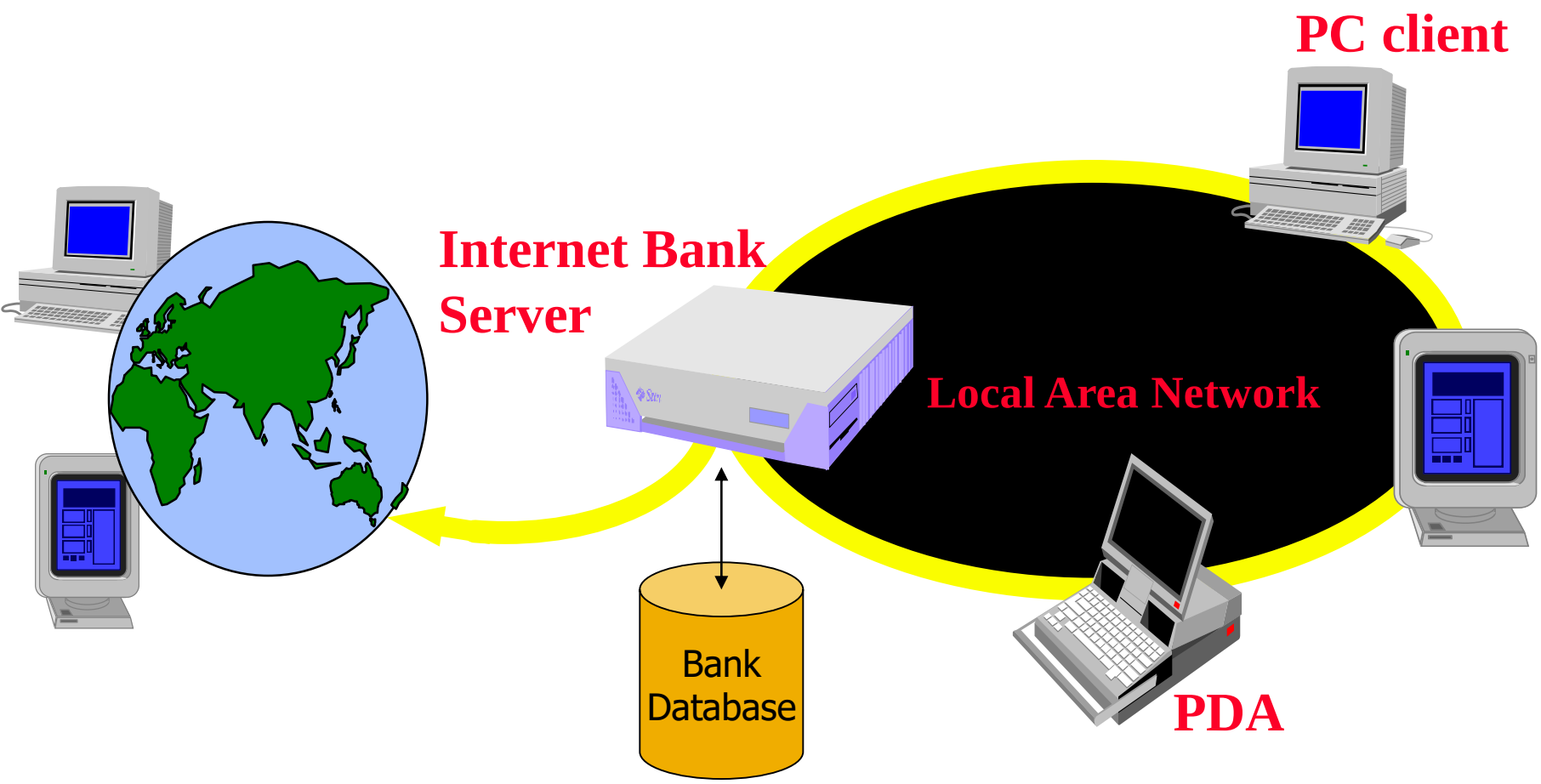
Next Class

- Thread Synchronisation
- Thread Priorities

Accessing Shared Resources

- Applications Access to Shared Resources need to be coordinated.
  - Printer (two person jobs cannot be printed at the same time)
  - Simultaneous operations on your bank account

Online Bank: Serving Many Customers and Operations



Shared Resources

- If one thread tries to read the data and other thread tries to update the same date, it leads to inconsistent state.
- This can be prevented by synchronising access to data.
- In Java: “Synchronized” method:
  - syncronised void update()
  - {
    - …
  - }

3rd Threads sharing the same object

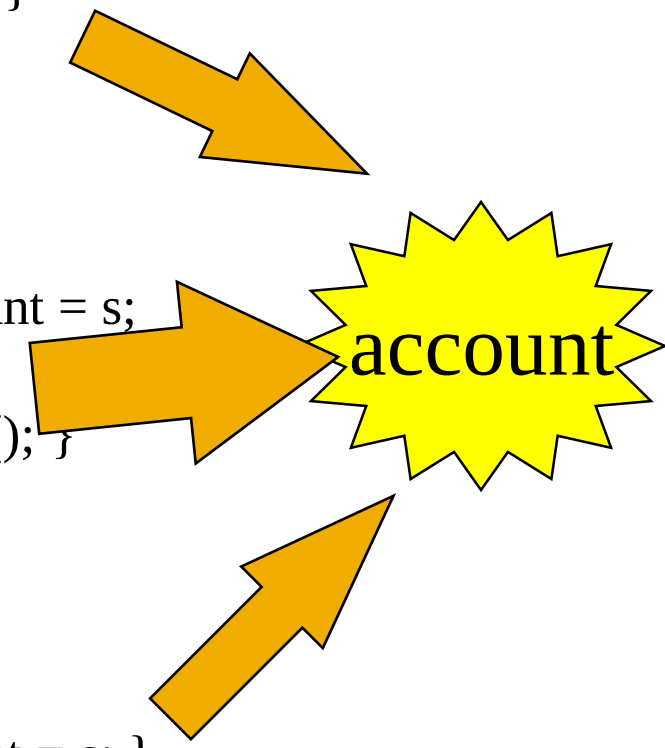
```
class InternetBankingSystem {
        public static void main(String [] args  ) {
            Account accountObject = new Account ();
            Thread t1 = new Thread(new MyThread(accountObject));
            Thread t2 = new Thread(new YourThread(accountObject));
            Thread t3 = new Thread(new HerThread(accountObject));
            t1.start();
            t2.start();
            t3.start();
          // DO some other operation
       } // end main()
}
```

Program with 3 threads and shared object

```
class MyThread implements Runnable  {
 Account account;
        public MyThread (Account s) {  account = s;}
        public void run() { account.deposit(); }
} // end class MyThread

class YourThread implements Runnable  {
 Account account;
        public YourThread (Account s) { account = s;
}
        public void run() { account.withdraw(); }
} // end class YourThread

class HerThread implements Runnable  {
 Account account;
        public HerThread (Account s) { account = s; }
        public void run() {account.enquire(); }
} // end class HerThread
```



Monitor (shared object) example

```
class Account {   // the 'monitor'
// DATA Members
  int balance;

// if 'synchronized' is removed, the outcome is unpredictable
 public synchronized void deposit( ) {
    // METHOD BODY : balance += deposit_amount;
  }

   public synchronized void withdraw( ) {
       // METHOD BODY: balance -= deposit_amount;
   }
   public synchronized void enquire( ) {
       // METHOD BODY: display balance.
   }
}
```

Thread Priority

- In Java, each thread is assigned priority, which affects the order in which it is scheduled for running. The threads so far had same default priority (ORM_PRIORITY) and they are served using FCFS policy.
  - Java allows users to change priority:
    - ThreadName.setPriority(intNumber)
      - MIN_PRIORITY = 1
      - NORM_PRIORITY=5
      - MAX_PRIORITY=10

Thread Priority Example

```
class A extends Thread
{
    public void run()
     {
        System.out.println("Thread A started");

        for(int i=1;i<=4;i++)
         {
            System.out.println("\t From ThreadA: i= "+i);
         }

          System.out.println("Exit from A");
     }

}

class B extends Thread
{
    public void run()
     {
        System.out.println("Thread B started");

        for(int j=1;j<=4;j++)
         {
            System.out.println("\t From ThreadB: j= "+j);
         }

          System.out.println("Exit from B");
     }

}
```

Thread Priority Example

```
class C extends Thread
{
     public void run()
      {
         System.out.println("Thread C started");

         for(int k=1;k<=4;k++)
          {
              System.out.println("\t From ThreadC: k= "+k);
          }
           System.out.println("Exit from C");
      }
}
class ThreadPriority
{
       public static void main(String args[])
        {
                A threadA=new A();
                B threadB=new B();
                C threadC=new C();

              threadC.setPriority(Thread.MAX_PRIORITY);
              threadB.setPriority(threadA.getPriority()+1);
              threadA.setPriority(Thread.MIN_PRIORITY);

              System.out.println("Started Thread A");
               threadA.start();

              System.out.println("Started Thread B");
               threadB.start();

              System.out.println("Started Thread C");
               threadC.start();

               System.out.println("End of main thread");
       }
}
```

APPLETS

accessed on an Internet server, transported over the Internet, automatically installed, and run as part of a Web document

- After an applet arrives on the client, it has limited access to resources, so that it can produce an arbitrary multimedia user interface and run complex computations without introducing the risk of viruses or breaching data integrity.

```
public class SimpleApplet extends Applet {
public void paint(Graphics g) {
g.drawString("A Simple Applet", 20, 20);
}   }
```

- Applets interact with the user through the AWT, not through the console-based I/O classes. The AWT contains support for a window-based, graphical interface.
- The second **import** statement imports the **applet package,** which contains the class **Applet.**

- outputs a string beginning at the specified X,Y location. It has the following general form:
  void drawString(String *message*, *int x*, *int y)*
- The applet does not have a **main( )** method. Unlike Java programs, applets do not begin execution at **main( ).**
- An applet begins execution when the name of its class is passed to an applet viewer or to a network browser.
- Compile in the same way that you have been compiling programs.

-
-

Web browser.

- Using an applet viewer, such as the standard SDK tool, **appletviewer.** An applet viewer executes your applet in a window.
- To execute an applet in a Web browser, you need to write a short HTML text file that contains the appropriate APPLET tag.

```
<applet code="SimpleApplet" width=200
height=60>
</applet>
```

called **RunApp.html,** then the following command line will run **SimpleApplet:**
C:\>appletviewer RunApp.html

- In general, you can quickly iterate through applet development by using these three steps:
  1. Edit a Java source file.
  2. Compile your program.
  3. Execute the applet viewer, specifying the name of your applet's source file. The applet viewer will encounter the APPLET tag within the comment and execute your applet.

-
-

or a Java-compatible browser.

- User I/O is not accomplished with Java's stream I/O classes. Instead, applets use the interface provided by the AWT.

The Applet Class

- **Applet** provides all of the necessary support for window-based activities (methods that load and display images, load and display audio clips). Applet extends panel, extends container, extends component.
- Methods defined by applet follows:

Method	Description
void destroy( )	Called by the browser just before an applet is terminated. Your applet will override this method if it needs to perform any cleanup prior to its destruction.
AccessibleContext getAccessibleContext( )	Returns the accessibilty context for the invoking object.
AppletContext getAppletContext( )	Returns the context associated with the applet.
String getAppletInfo( )	Returns a string that describes the applet.
AudioClip getAudioClip(URL *url*)	Returns an **AudioClip** object that encapsulates the audio clip found at the location specified by *url*.
AudioClip getAudioClip(URL *url*, String *clipName*)	Returns an **AudioClip** object that encapsulates the audio clip found at the location specified by *url* and having the name specified by *clipName*.
URL getCodeBase( )	Returns the URL associated with the invoking applet.
URL getDocumentBase( )	Returns the URL of the HTML document that invokes the applet.

Method	Description
Image getImage(URL *url*)	Returns an **Image** object that encapsulates the image found at the location specified by *url*.
Image getImage(URL *url*, String *imageName*)	Returns an **Image** object that encapsulates the image found at the location specified by *url* and having the name specified by *imageName*.
Locale getLocale( )	Returns a **Locale** object that is used by various locale-sensitive classes and methods.
String getParameter(String *paramName*)	Returns the parameter associated with *paramName*. **null** is returned if the specified parameter is not found.
String[ ] [ ] getParameterInfo( )	Returns a **String** table that describes the parameters recognized by the applet. Each entry in the table must consist of three strings that contain the name of the parameter, a description of its type and/or range, and an explanation of its purpose.
void init( )	Called when an applet begins execution. It is the first method called for any applet.
boolean isActive( )	Returns **true** if the applet has been started. It returns **false** if the applet has been stopped.
static final AudioClip newAudioClip(URL *url*)	Returns an **AudioClip** object that encapsulates the audio clip found at the location specified by *url*. This method is similar to **getAudioClip( )** except that it is static and can be executed without the need for an **Applet** object. (Added by Java 2)

Applet Architecture

- An applet is a window-based program
- First, Applets are event driven; An applet waits until an event occurs
- The AWT notifies the applet about an event by calling an event handler that has been provided by the applet
- Once this happens, the applet must take appropriate action and then quickly return control to the AWT

window based programs)

- The user interacts with the applet as he or she wants. These interactions are sent to the applet as events to which the applet must respond
- For example, when the user clicks a mouse inside the applet's window, a mouse-clicked event is generated
- When the user interacts with one of these controls, an event is generated.

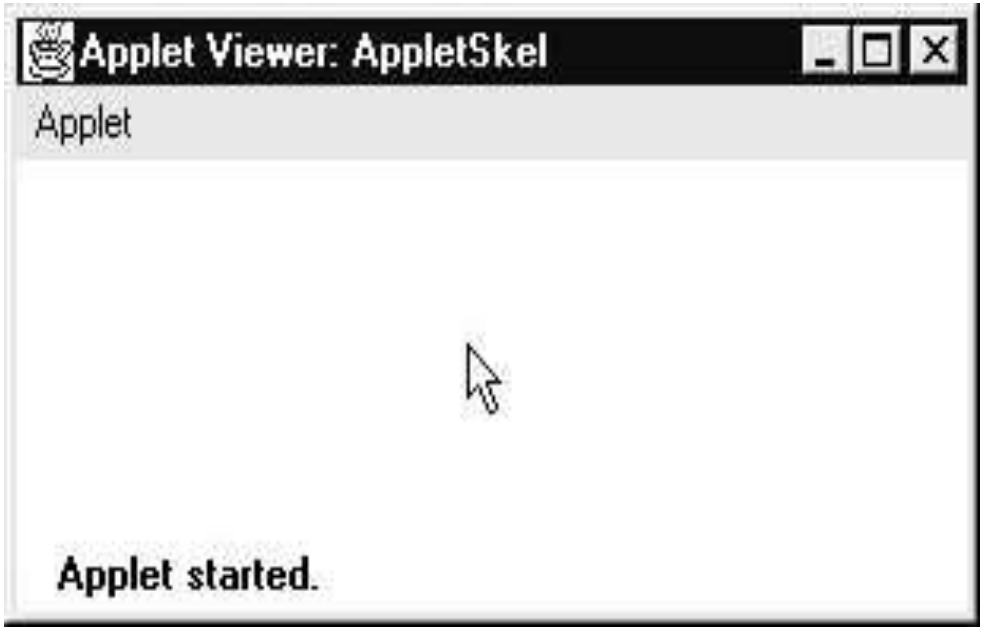
An Applet Skeleton

- All but the most trivial applets override a set of methods that provides the basic mechanism by which the browser or applet viewer interfaces to the applet and controls its execution.
- Four of these methods—**init( ), start( ), stop( ),** and **destroy( )**—are defined by **Applet.**
- Another, **paint( ),** is defined by the **AWT Component** class.
- Default implementations for all of these methods are provided. Applets do not need

```
/* <applet code="AppletSkel" width=300
    height=100> </applet> */
public class AppletSkel extends Applet {
// Called first.
public void init() {
// initialization
}
/* Called second, after init(). Also called
    whenever the applet is restarted. */
public void start() {
// start or resume execution
}
```

```
// suspends execution
}
/* Called when applet is terminated. This is the
    last method executed. */
public void destroy() {
// perform shutdown activities
}
// Called when an applet's window must be
    restored.
public void paint(Graphics g) {
// redisplay contents of window
}
}
```

generates the following window when viewed with an applet viewer:



- following methods, in this sequence:
  1. **init( )**
  2. **start( )**
  3. **paint( )**

- When an applet is terminated, the following sequence of method calls takes place:
  1. **stop( )**
  2. **destroy( )**

init( ) method

- The **init( )** method is the first method to be called
- This is where we should initialize variables
- This method is called only once during the run time of the applet.

start ( ) method

- The **start( )** method is called after **init( )**
- It is also called to restart an applet after it has been stopped
- Whereas **init( )** is called once - the first time an applet is loaded - **start( )** is called each time an applet's HTML document is displayed onscreen
- So, if a user leaves a web page and comes back, the applet resumes execution at **start( )**

paint ( ) method

- applet's output must be redrawn
- This situation can occur for several reasons
- The window in which the applet is running can be overwritten by another window and then uncovered
- Or, the applet window can be minimized and then restored
- **paint( )** is also called when the applet begins execution
- The **paint( )** method has one parameter of type **Graphics**
- This parameter contains the graphics context, which describes the graphics environment in which the applet is running

stop( ) method

browser leaves the HTML document containing the applet - when it goes to another page, for example.

- When **stop( )** is called, the applet is probably running
- You should use **stop( )** to suspend threads that don't need to run when the applet is not visible
- You can restart them when **start( )** is called if the user returns to the page

destroy( ) method

- The **destroy( )** method is called when the environment determines that your applet needs to be removed completely from memory

- At this point, you should free up any resources the applet may be using

- The **stop( )** method is always called before **destroy( )**

window be redrawn.

- The default version of **update( )** first fills an applet with the default background color and then calls **paint( ).**
- If you fill the background using a different color in **paint( ),** the user will experience a flash of the default background each time **update( )** is called—that is, whenever the window is repainted.
- override the **update(**) method so that it performs all necessary display activities. Then have **paint( )** simply call update( ). Thus, for some applications, the applet skeleton will override **paint( )** and **update( ),** as shown here:

```
public void update(Graphics g) {
// redisplay your window, here.        }
public void paint(Graphics g) {
update(g);        }
```

Simple Applet Display Methods

- To output a string to an applet, use **drawString( )**, which is a member of the **Graphics** class

  void drawString(String *message*, int *x*, int *y*)

- Here, *message* is the string to be output beginning at *x*,*y*

- If you want to start a line of text on another line, you must do so manually, specifying the precise X,Y location where you want the line to begin
- To set the background color of an applet's window, use **setBackground( ).** To set the foreground color (the color in which text is shown), use **setForeground( ).**
- These methods are defined by **Component,** and they have the following general forms:

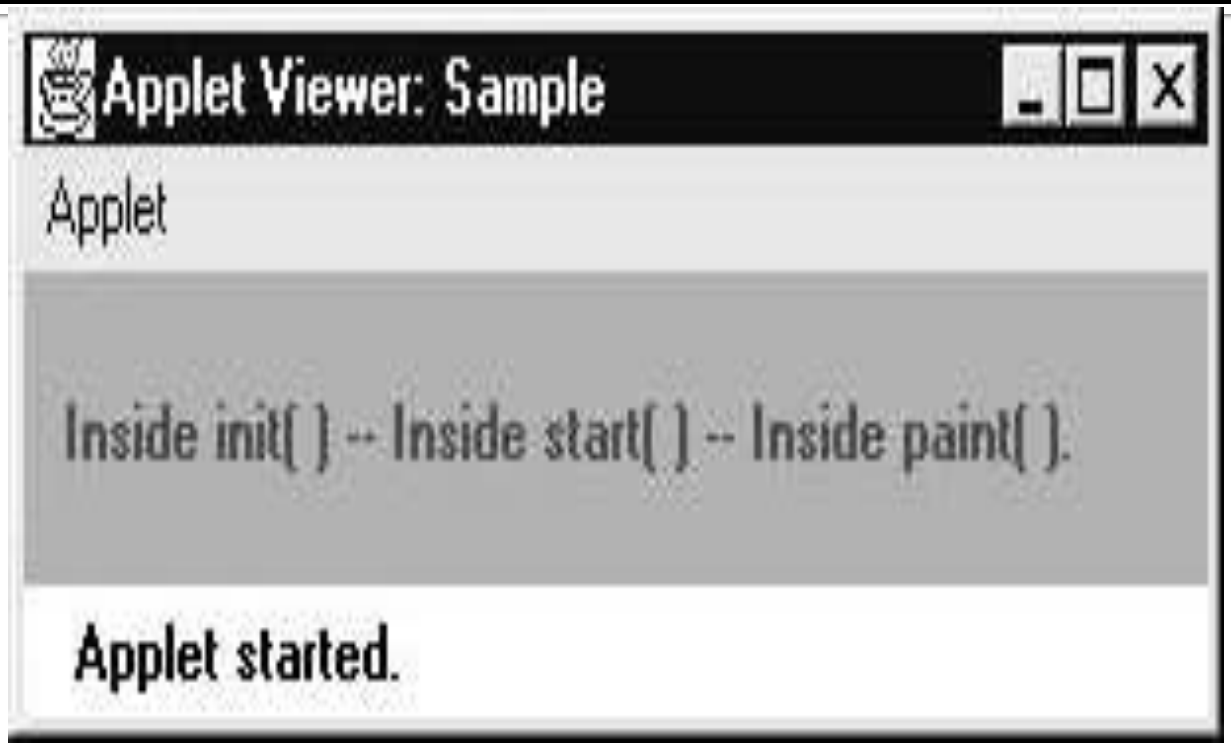
- The class **Color** defines the constants that can be used to specify colors

Color.black
Color.blue
Color.cyan
Color.darkGray
Color.gray
Color.green
Color.lightGray
Color.magenta
Color.orange
Color.pink
Color.red
Color.white
Color.yellow

- setForeground(Color.red);
  Color getBackground( )
  Color getForeground( )
- Example program:

```
import java.awt.*;
import java.applet.*;
/*  <applet code="Sample" width=300
height=50>
</applet> */
public class Sample extends Applet{
String msg;
```

```
setForeground(Color.red);
msg = "Inside init( ) --";
}
// Initialize the string to be displayed.
public void start() {
msg += " Inside start( ) --";
}
// Display msg in applet window.
public void paint(Graphics g) {
msg += " Inside paint( ).";
g.drawString(msg, 10, 30);
} }
```


Applet Viewer: Sample
Applet
Inside init( ) -- Inside start( ) -- Inside paint( ).
Applet started.

Requesting Repainting

- An applet writes to its window only when its **update( )** or **paint( )** method is called by the AWT
- an applet must quickly return control to the AWT run-time system. It cannot create a loop inside **paint( )** to scroll
- Whenever your applet needs to update the information displayed in its window, it simply calls **repaint( )**

repainted

void repaint(int left, int top, int width, int height)
- The coordinates of the upper-left corner of the region are specified by *left* and *top*, and the width and height of the region are passed in *width* and *height.*
- These dimensions are specified in pixels

-You save time by specifying a region to repaint .

- If you need to update only a small portion of the window, it is more efficient to repaint only that region.

for repainting that occur within a short time can be collapsed by the AWT in a manner such that **update( )** is only called sporadically. This can be a problem in many situations, including animation, in which a consistent update time is necessary. One solution to this problem is to use the following forms of **repaint( ):**

void repaint(long maxDelay)

void repaint(long maxDelay, int x, int y, int width, int height)

- Here, *maxDelay* specifies the maximum number of milliseconds that can elapse before **update( )** is called

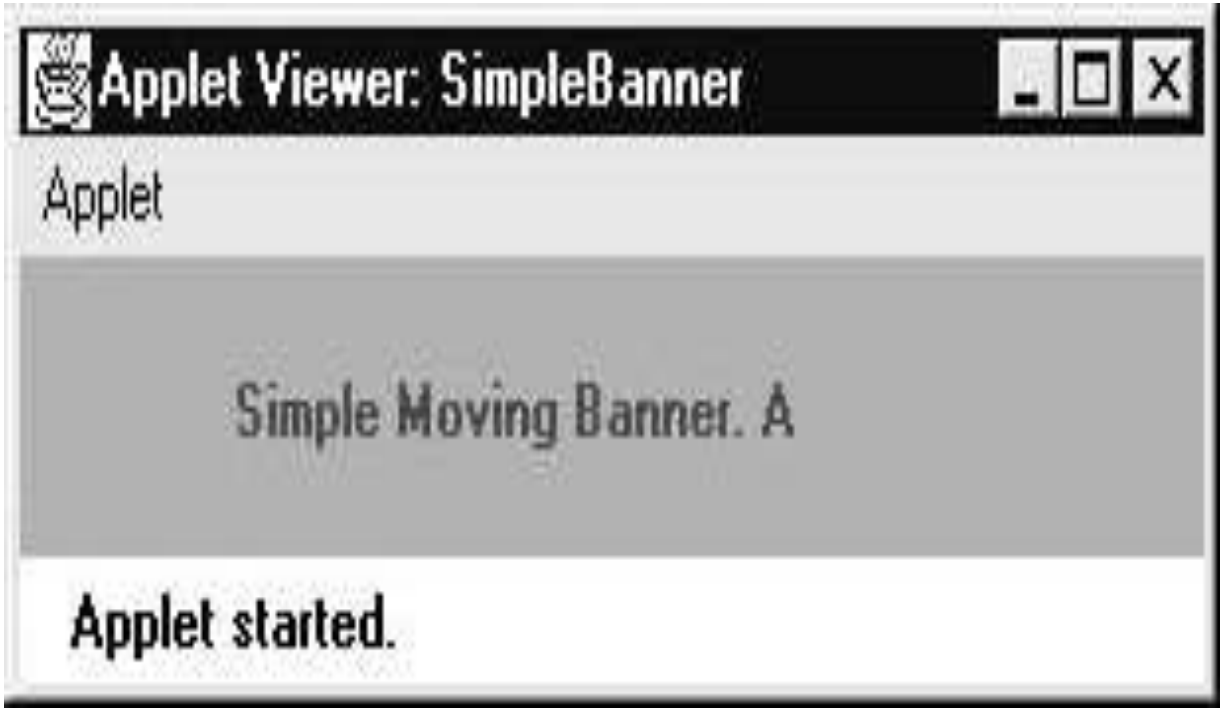
```
the message contained in msg right to left
across the applet's window.
*/
import java.awt.*;
import java.applet.*;
/*
<applet code="SimpleBanner" width=300
height=50>
</applet>
*/
```

```
Thread t = null;
int state;
boolean stopFlag;
public void init() {     // Set colors and initialize
    thread.
setBackground(Color.cyan);
setForeground(Color.red);
}
public void start() {         // Start()  of applet
t = new Thread(this);
stopFlag = false;
t.start();                        // Start thread
}
```

```
char ch;
// Display banner
for( ; ; ) {
try {
repaint();
Thread.sleep(250);
ch = msg.charAt(0);
msg = msg.substring(1, msg.length());
msg += ch;
if(stopFlag)
break;
} catch(InterruptedException e) {}
}   }
```

```
public void stop() {
stopFlag = true;
t = null;
}
// Display the banner.
public void paint(Graphics g) {
g.drawString(msg, 50, 30);
}
}
```

- Following is sample output:



since the applet will be creating a second thread of execution that will be used to scroll the banner.

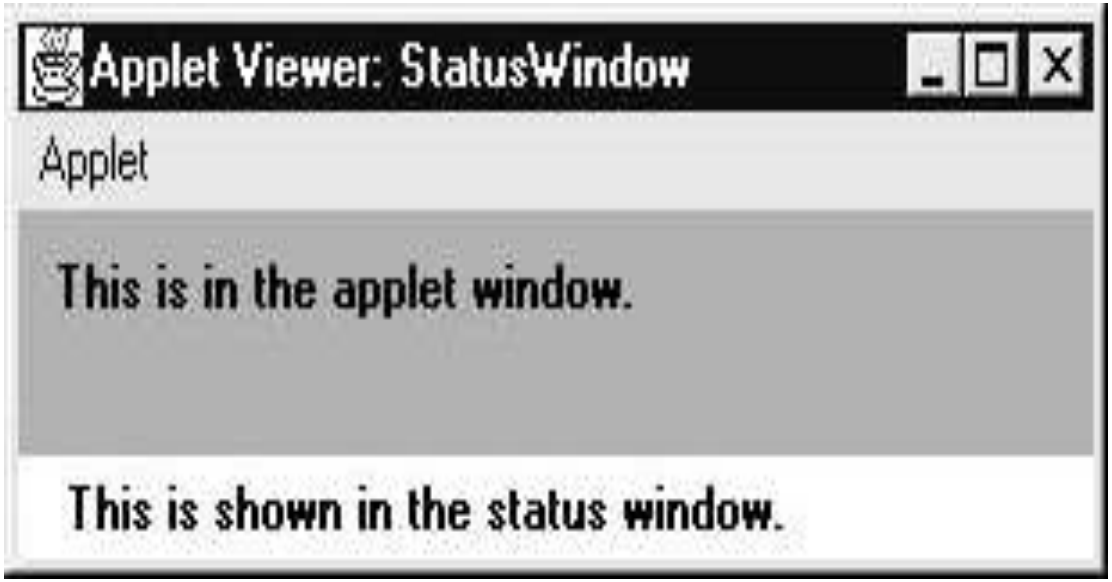
- Inside **init( ),** the foreground and background colors of the applet are set.
- After initialization, the AWT run-time system calls **start( )** to start the applet running. Inside **start( ),** a new thread of execution is created and assigned to the **Thread** variable **t**.
- Then, the **boolean** variable **stopFlag,** which controls the execution of the applet, is set to **false.** Next, the thread is started by a call to

This eventually causes the **paint( )** method to be called and the current contents of **msg** is displayed. Between each iteration, **run( )** sleeps for a quarter of a second. The net effect of **run( )** is that the contents of **msg** is scrolled right to left in a constantly moving display. The **stopFlag** variable is checked on each iteration. When it is **true,** the **run( )** method **terminates.**

- If a browser is displaying the applet when a new page is viewed, the **stop( )** method is called, which sets **stopFlag** to **true,** causing **run( )** to terminate. When the applet is brought back into view, **start( )** is once again called, which starts a new thread to

- status window of the browser or applet viewer on which it is running
- To do so, call **showStatus( )** with the string that you want displayed
- The status window is a good place to give the user feedback about what is occurring in the applet, suggest options, or report errors.
- The status window also makes an excellent debugging aid, because it gives you an easy way to output information about your applet.

```
/
<applet code="StatusWindow" width=300
    height=50>
</applet>    */
public class StatusWindow extends Applet{
public void init() {
setBackground(Color.cyan);
}
// Display msg in applet window.
public void paint(Graphics g) {
g.drawString("This is in the applet window.", 10,
    20);
showStatus("This is shown in the status window.");
}  }
```


Applet Viewer: StatusWindow
Applet
This is in the applet window.
This is shown in the status window.

from both an HTML document and from an applet viewer

- An applet viewer will execute each APPLET tag that it finds in a separate window, while web browsers like Netscape Navigator, Internet Explorer, and HotJava will allow many applets on a single page

CODE = *appletFile*
[ALT = *alternateText*]
[NAME = *appletInstanceName*]
WIDTH = *pixels* HEIGHT = *pixels*
[ALIGN = *alignment*]
[VSPACE = *pixels*] [HSPACE = *pixels*] >
[< PARAM NAME = *AttributeName* VALUE =
AttributeValue>]
[< PARAM NAME = *AttributeName2* VALUE =
AttributeValue>] . . .
[*HTML Displayed in the absence of Java*]
</APPLET>

- URL of the applet code, which is the directory that will be searched for the applet's executable class file (specified by the CODE tag)

CODE

- Is a required attribute that gives the name of the file containing your applet's compiled **.class** file. This file is relative to the code base URL of the applet

browser understands the APPLET tag but can't currently run Java applets

NAME

- NAME is an optional attribute used to specify a name for the applet instance

- Applets must be named in order for other applets on the same page to find them by name and communicate with them. To obtain an applet by name, use
  getApplet( )

- the size (in pixels) of the applet display area

ALIGN

- ALIGN is an optional attribute that specifies the alignment of the applet

- This attribute is treated the same as the HTML IMG tag with these possible values: LEFT, RIGHT, TOP, BOTTOM, MIDDLE, BASELINE, TEXTTOP, ABSMIDDLE, and ABSBOTTOM

- VSPACE specifies the space, in pixels, above and below the applet
- HSPACE specifies the space, in pixels, on each side of the applet

- appletspecific arguments in an HTML page
- Applets access their attributes with the **getParameter( )** method
- [*HTML Displayed in the absence of Java*]

Passing Parameters to Applets

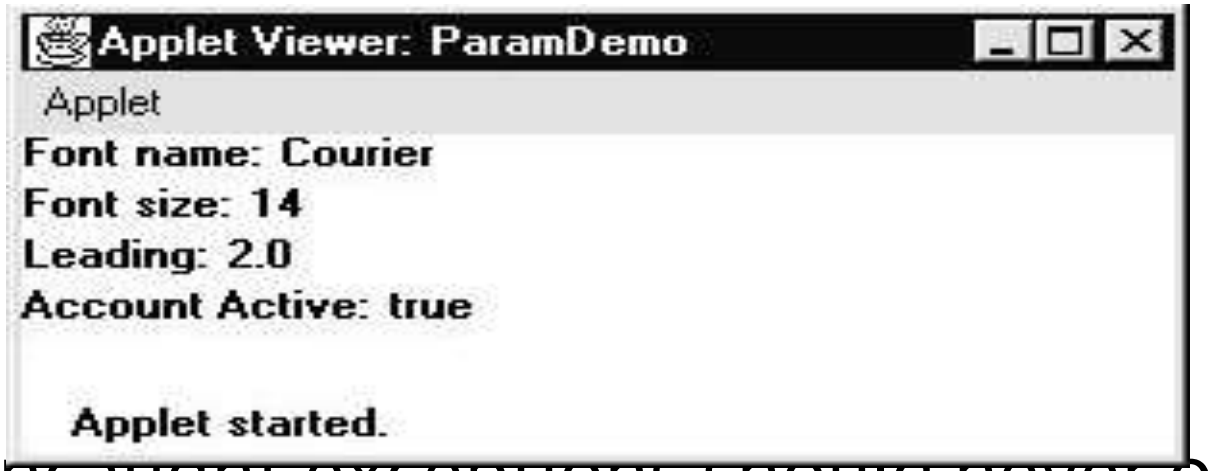
- the APPLET tag in HTML allows us to pass parameters to the applet
- To retrieve a parameter, use the **getParameter( )** method
- It returns the value of the specified parameter in the form of a **String** object
- For numeric and **boolean** values, you will need to convert their string representations into their internal formats

```
/* <applet code="ParamDemo" width=300
    height=80>
<param name=fontName value=Courier>
<param name=fontSize value=14>
<param name=leading value=2>
<param name=accountEnabled value=true>
</applet>    */
public class ParamDemo extends Applet{
   String fontName;
   int fontSize;
   float leading;
   boolean active;
```

```
fontName = getParameter("fontName");
if(fontName == null)
    fontName = "Not Found";
param = getParameter("fontSize");
try {
    if(param != null)
        fontSize = Integer.parseInt(param);
    else                                  // if not found
        fontSize = 0;
     } catch(NumberFormatException e) {
            fontSize = -1;
      }
```

```
            leading =
  Float.valueOf(param).floatValue();
          else                                  // if not found
              leading = 0;
          } catch(NumberFormatException e) {
                leading = -1;
            }
    param = getParameter("accountEnabled");
    if(param != null)
        active =
  Boolean.valueOf(param).booleanValue();
     }
// Display parameters.
```

```
        g.drawString("Font size: " + fontSize, 0, 26);
        g.drawString("Leading: " + leading, 0, 42);
        g.drawString("Account Active: " + active, 0,
    58);
    }
}
```



- Uncaught exceptions should never occur within an applet.

```
/* <applet code="ParamBanner" width=300 height=50>
<param name=message value="Java makes the Web move!">
</applet> */
public class ParamBanner extends Applet implements
Runnable {
String msg;
Thread t = null;
int state;
boolean stopFlag;
// Set colors and initialize thread.
public void init() {
setBackground(Color.cyan);
```

```
msg = getParameter("message");
if(msg == null) msg = "Message not found.";
msg = " " + msg;
t = new Thread(this);
stopFlag = false;
t.start();
} // Entry point for the thread that runs the banner.
public void run() {
char ch;
// Display banner
for( ; ; ) {
```

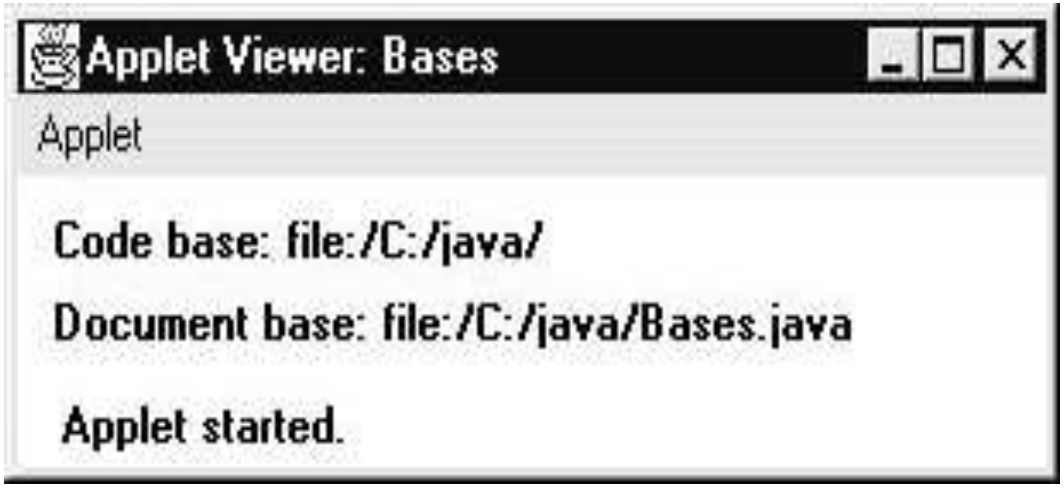
```
msg = msg.substring(1, msg.length());
msg += ch;
if(stopFlag)
break;
} catch(InterruptedException e) {}
}    }    // Pause the banner.
public void stop() {
stopFlag = true;
t = null;                }          // Display the banner.
public void paint(Graphics g) {
g.drawString(msg, 50, 30);
}              }
```

getDocumentBase( ) and getCodeBase( )

- Java will allow the applet to load data from the directory holding the HTML file that started the applet (the *document base*) and the directory from which the applet's class file was loaded (the *code base*)

```
/*
<applet code="Bases" width=300 height=50>
</applet>
*/
public class Bases extends Applet{
// Display code and document bases.
public void paint(Graphics g) {
String msg;
URL url = getCodeBase(); // get code base
msg = "Code base: " + url.toString();
g.drawString(msg, 10, 20);
```

```
g.drawString(msg, 10, 40);
}
}
```



AppletContext and showDocument

- To allow the applet to transfer control to another URL, we must use the **showDocument( )** method defined by the **AppletContext** interface

- **AppletContext** is an interface that lets us get information from the applet's execution environment

- The context of the currently executing applet is obtained by a call to the **getAppletContext( )** method defined by **Applet**

- applet's context, you can bring another document into view by calling **showDocument( )**

- This method has no return value and throws no exception if it fails

- There are two **showDocument( )** methods

- The method **showDocument(URL)** displays the document at the specified **URL**

- displays the specified document at the specified location within the browser window

- Valid arguments for *where* are "_self" (show in current frame), "_parent" (show in parent frame), "_top" (show in topmost frame), and "_blank" (show in new browser window)

```
/*
<applet code="ACDemo" width=300 height=50>
</applet>
*/
public class ACDemo extends Applet{
public void start() {
AppletContext ac = getAppletContext();
URL url = getCodeBase(); // get url of this applet
try {
ac.showDocument(new URL(url+"Test.html"));
} catch(MalformedURLException e) {
showStatus("URL not found");
} }  }
```

The AudioClip Interface

- The ***AudioClip*** interface defines these methods: ***play( )*** (play a clip from the beginning), ***stop( )*** (stop playing the clip), and ***loop( )*** (play the loop continuously)

- After you have loaded an audio clip using ***getAudioClip( )***, you can use these methods to play it

Outputting to the Console

- It is possible to use console output in the applet—especially for debugging purposes
- In an applet, when you call a method such as **System.out.println( )**, the output is not sent to your applet's window
- Instead, it appears either in the console session in which you launched the applet viewer

- Thank You