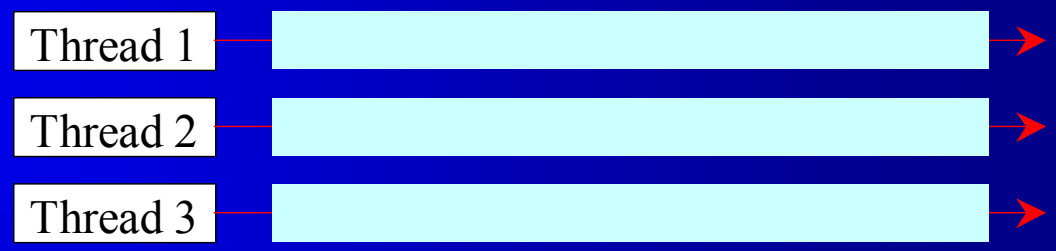
# Chapter 13: Multithreading

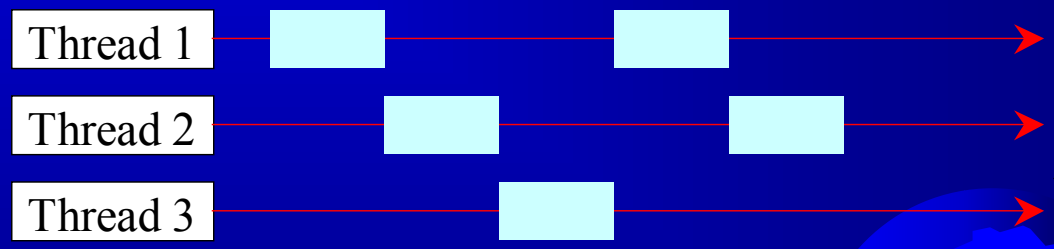
- Threads Concept
- Creating Threads by Extending the `Thread` class
- Creating Threads by Implementing the `Runnable` Interface
- Controlling Threads and Thread Status
- Thread Groups
- Synchronization
- Creating Threads for Applets
- Case Studies

# Threads Concept

Multiple threads on multiple CPUs




Multiple threads sharing a single CPU



# Creating Threads by Extending the `Thread` class

```
// Custom thread class
public class CustomThread extends Thread
{
  ...
  public CustomThread(...)
  {
    ...
  }

  // Override the run method in Thread
  public void run()
  {
    // Tell system how to run custom thread
    ...
  }

  ...
}
```

```
// Client class
public class Client
{
  ...
  public someMethod()
  {
    ...
    // Create a thread
    CustomThread thread = new CustomThread(...);

    // Start a thread
    thread.start();
    ...
  }

  ...
}
```

# Example 13.1 Using the `Thread` Class to Create and Launch Threads

- Objective: Create and run three threads:
  - The first thread prints the letter *a* 100 times.
  - The second thread prints the letter *b* 100 times.
  - The third thread prints the integers 1 through 100.

# Example 13.1 Using the `Thread` Class to Create and Launch Threads, cont.

TestThread

Run

Click the Run button to access the DOS prompt; then type `java TestThread`

# Creating Threads by Implementing the `Runnable` Interface

```
// Custom thread class
public class CustomThread
  implements Runnable
{
  ...
  public CustomThread(...)
  {
    ...
  }

  // Implement the run method in Runnable
  public void run()
  {
    // Tell system how to run custom thread
    ...
  }

  ...
}
```

```
// Client class
public class Client
{
  ...
  public someMethod()
  {
    ...
    // Create an instance of CustomThread
    CustomThread customThread
      = new CustomThread(...);

    // Create a thread
    Thread thread = new Thread(customThread);

    // Start a thread
    thread.start();
    ...
  }

  ...
```

# Example 13.2
# Using the `Runnabel` Interface to Create and Launch Threads

- Objective: Create and run three threads:
  - The first thread prints the letter *a* 100 times.
  - The second thread prints the letter *b* 100 times.
  - The third thread prints the integers 1 through 100.

Run

TestRunnable

Click the Run button to access the DOS prompt; then type

```
java TestRunnable
```

# Controlling Threads and Thread States

- `void run()`

  Invoked by the Java runtime system to execute the thread. You must override this method and provide the code you want your thread to execute.

- `void start()`

  Starts the thread, which causes the run() method to be invoked. Called by the runnable object in the client class.

- `static void sleep(long millis) throws InterruptedException`

  Puts the runnable object to sleep for a specified time in milliseconds.

# Controlling Threads and Thread States, cont.

- `void stop()`

  Stops the thread. (deprecated in JDK 1.2)

- `void suspend()` (deprecated in JDK 1.2)

  Suspends the thread. Use the `resume()` method to resume.

- `void resume()` (deprecated in JDK 1.2)

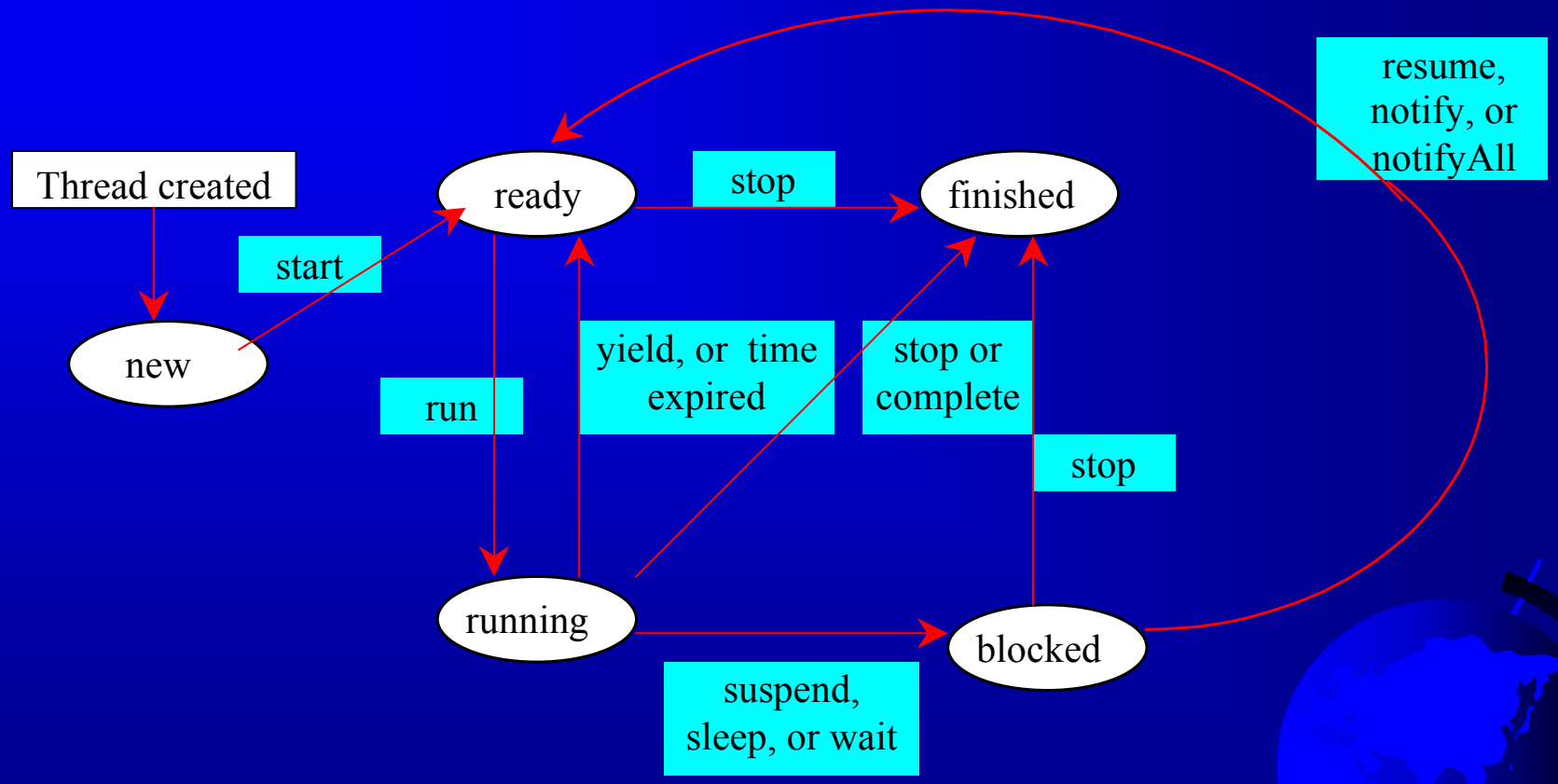
  Resumes the thread suspended with the `suspend()` method.

# Thread Priority

- Each thread is assigned a default priority of `Thread.NORM_PRIORITY`. You can reset the priority using `setPriority(int priority)`.

- Some constants for priorities include
  `Thread.MIN_PRIORITY`
  `Thread.MAX_PRIORITY`
  `Thread.NORM_PRIORITY`

# Thread States



# Thread Groups

- Construct a thread group using the `ThreadGroup` constructor:

```
ThreadGroup g = new ThreadGroup("timer
thread group");
```

- Place a thread in a thread group using the `Thread` constructor:

```
Thread t = new Thread(g, new
ThreadClass(), "This thread");
```

# Thread Groups, cont.

- To find out how many threads in a group are currently running, use the `activeCount()` method:

```
System.out.println("The number of “
   + “ runnable threads in the group ” +
   g.activeCount());
```

# Synchronization

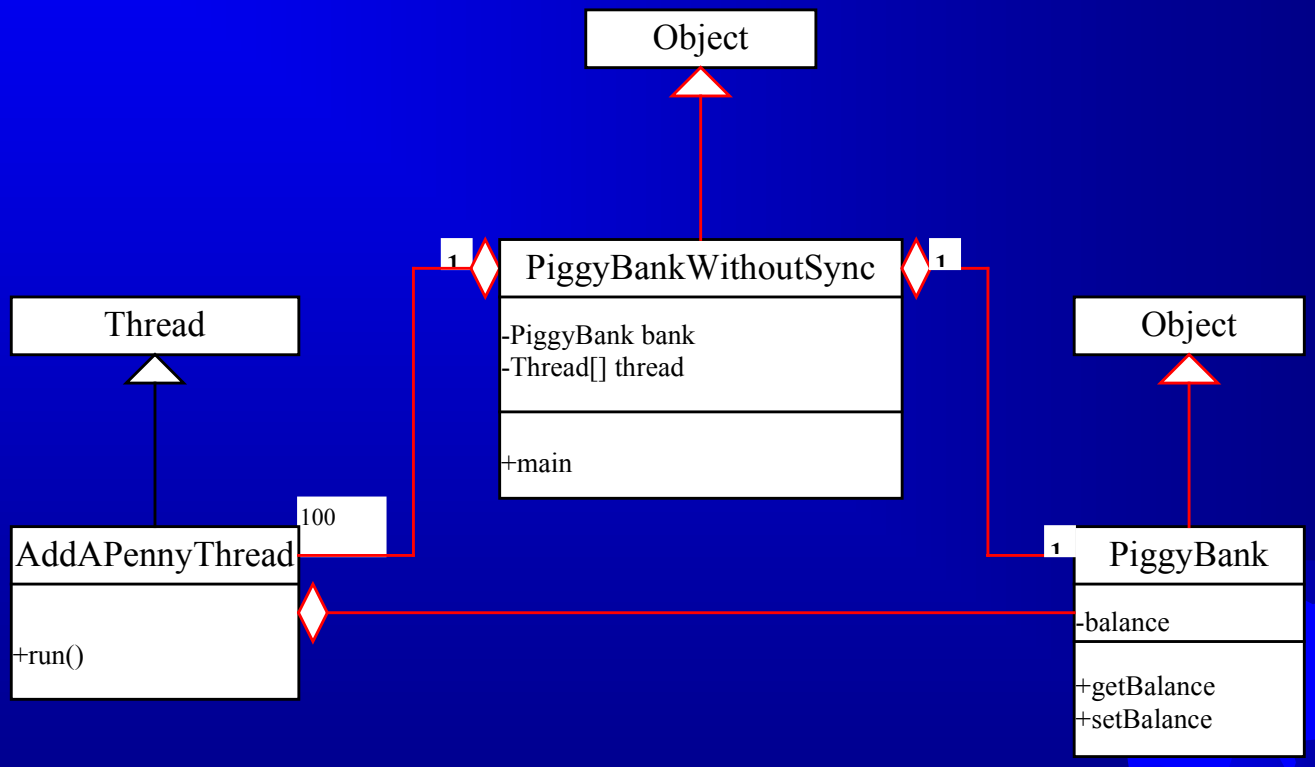
A shared resource may be corrupted if it is accessed simultaneously by multiple threads. For example, two unsynchronized threads accessing the same bank account causes conflict.

| Step | balance | thread[i] | thread[j] |
|---|---|---|---|
| 1 | 0 | newBalance = bank.getBalance() + 1; | |
| 2 | 0 | | newBalance = bank.getBalance() + 1; |
| 3 | 1 | bank.setBalance(newBalance); | |
| 4 | 1 | | bank.setBalance(newBalance); |

# Example 13.3
# Showing Resource Conflict

- Objective: create and launch 100 threads, each of which adds a penny to a piggy bank. Assume that the piggy bank is initially empty.

# Example 13.3, cont




PiggyBankWithoutSync

Run

# The `synchronized` keyword

To avoid resource conflicts, Java uses the keyword `synchronized` to synchronize method invocation so that only one thread can be in a method at a time. To correct the data-corruption problem in Example 13.3, you can rewrite the program as follows:

PiggyBankWithSync

Run

# Creating Threads for Applets

In Example 12.1, "Displaying a Clock," you drew a clock to show the current time in an applet. The clock does not tick after it is displayed. What can you do to let the clock display a new current time every second? The key to making the clock tick is to repaint it every second with a new current time. You can use the code given below to override the `start()` method in `CurrentTimeApplet`:

# Creating Threads for Applets

```
public void start()
{
  while (true)
  {
    stillClock.repaint();
    try
    {
      Thread.sleep(1000);
    }
    catch(InterruptedException ex)
    {
    }
  }
}
```

What is wrong in this code? As long as the while loop is running, the browser cannot serve any other event that might be occurring.

# Creating a Thread to run the `while` loop

```
public class MyApplet extends
  JApplet implements Runnable
{ private Thread timer = null;
  public void init()
  { timer = new Thread(this);
    timer.start();
  }
  ...
  public void run()
  { ... }
}
```

# Creating a Thread to run the `while` loop, cont.

```
public void run()
{ while (true)
  { repaint();
    try
    { thread.sleep(1000);
      waitForNotificationToResume();
    }
    catch (InterruptedException ex)
    {      }
  }
}
```

# Creating a Thread to run the `while` loop, cont.

```
private synchronized void
  waitForNotificationToResume()
  throws InterruptedException
{
  while (suspended)
    wait();
}
```

# Creating a Thread to run the `while` loop, cont.

```
public synchronized void resume()
{
  if (suspended)
  {
    suspended = false;
    notify();
  }
}

public synchronized void suspend()
{
  suspended = true;
}
```

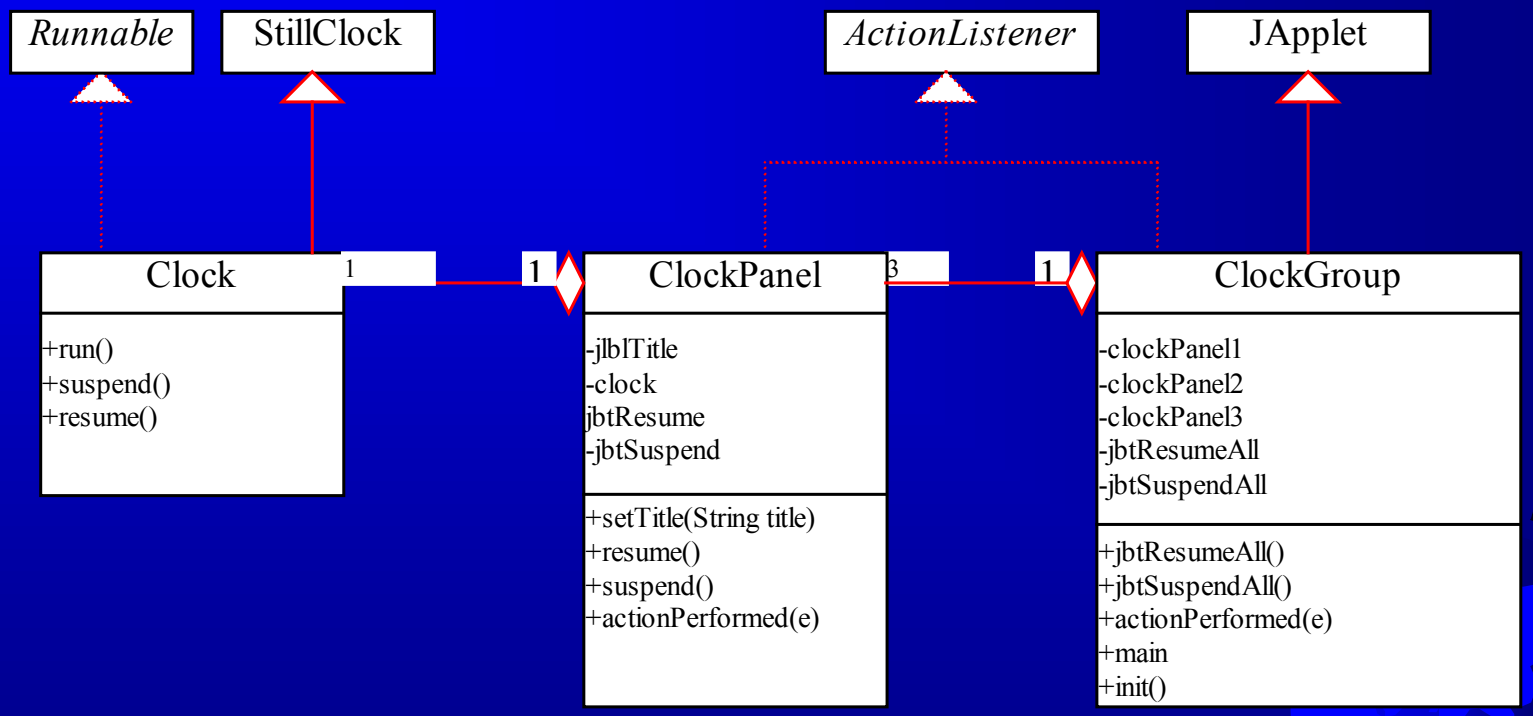
# Example 13.4 Displaying a Running Clock in in an Applet

- Objective: Simulate a running clock by using a separate thread to repaint the clock.

ClockApplet

Run Applet Viewer

# Example 13.5
# Controlling a Group of Clocks




ClockGroup | ClockPanel | Clock | Run