Programming in Java

Threads

Example Application

- Implement a simple alphabet program
  - Cycle through the 26 letters of the alphabet continuously
  - Print them on the terminal (using System.println)
  - Print them in the applet window (using drawString)

First attempt (doesn't work)

```
import java.applet.*;
import java.awt.*;

public class ABC extends Applet {
  char   theChar = 'A';
  public void start() {
    System.out.println ("Applet Starting");
    while ( true ) {
      System.out.println( theChar );
      repaint();
      try { Thread.sleep( 1000 );         }
      catch (InterruptedException e ) { ; }
      if ( (++theChar) > 'Z' ) theChar = 'A';
    }
  }
  public void paint( Graphics g ) {
    g.drawString( "" + theChar, 100, 100 );
  }
}
```


Thread

- Creates a separate execution flow
- Two ways to do threads
  - Define a class that implements the interface **`Runnable`**
    - Define a method called **`run`** in the class
    - Create an instance of **`Thread`** and pass it an instance of a **`Runnable`** object
  - Extend the class **`Thread`**
    - Define a method called **`run`** in the class
    - Create an instance of that class
- Next slide shows example program using threads

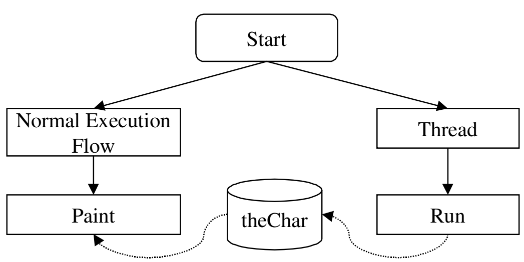
Second Attempt (uses a thread)

```
import java.applet.*;
import java.awt.*;

public class ABCThread extends Applet implements Runnable {
  char   theChar = 'A';
  Thread theThread;
  public void start() {
    System.out.println ("Applet Starting");
    if ( theThread == null ) {
      theThread = new Thread( this );
      theThread.start();
    }
  }
  public void stop() {
    System.out.println ("Applet Stopping");
    if ( theThread != null ) {
      theThread.stop();
      theThread = null;
    }
  }
  ...
```


Second Attempt (cont.)

```
  ...
  public void run() {
    while ( true ) {
      System.out.println( theChar );
      repaint();
      try { Thread.sleep( 1000 );          }
      catch (InterruptedException e ) { ; }
      if ( (++theChar) > 'Z' ) theChar = 'A';
    }
  }

  public void paint( Graphics g ) {
    g.drawString( "" + theChar, 100, 100 );
  }
}
```


Thread: new execution flow



Extending Thread Class

```
// Creating a simple thread by inheritance
public class ThreadByBirth extends Thread {
  public static void main(String args[]) {
    ThreadByBirth theApp = new ThreadByBirth();
    theApp.start();
  }
  public void run() {
    System.out.println ("Running Thread");
    this.getThreadGroup().list();
    System.exit(0);
  }
}
```

– Sample Execution

> java ThreadByBirth
Running Thread
java.lang.ThreadGroup[name=main,maxpri=10]
Thread[main,5,main]
Thread[Thread-1,5,main]

Threads can be named

```
// Giving your threads a name
public class NamedThread extends Thread {
  public static void main(String args[]) {
    NamedThread theApp = new NamedThread();
    theApp.start();
  }

  public void run() {
    this.setName("My thread.");
    System.out.println("Running Thread");
    this.getThreadGroup().list();
    System.exit(0);
  }
}
```

– Sample Execution

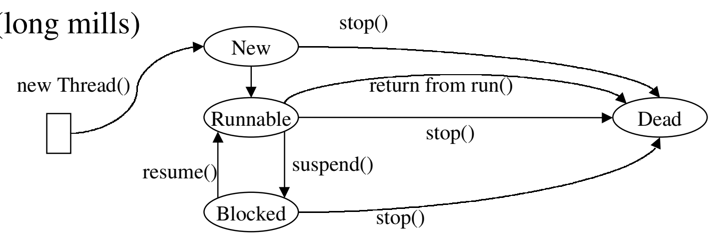
> java ThreadByBirth
Running Thread
java.lang.ThreadGroup[name=main,maxpri=10]
Thread[main,5,main]
Thread[My thread,5,main]

Adding Thread Capability

- Threads can be created two ways
  - Implement interface (Runnable)
  - Extend class (Thread)
- How do you decide which is better?
  - Inheritance is usually easiest
    - But: Java only allows a class to extend at most one other class
  - Choose the "most appropriate" superclass
  - Example: If class has to be an Applet AND a Thread
    `public class Xyz extends Applet implements Runnable`
  - If no other inheritance is necessary
    `public class Xyz extends Thread`

Useful Thread Methods

- Some useful thread methods
  - start() - starts a thread, invokes the thread's "run" method
  - stop() - stops execution of a thread
  - suspend() - Temporarily suspends execution of a thread
  - resume() - Resumes execution of a suspended thread
  - yield() - Pauses thread and allows other threads to execute
  - sleep(long mills)



Daemon Threads; Thread Groups

- Two types of threads
  - Regular threads - independent threads
  - Daemon threads - not independent; these support other threads
    - Use the **`setDaemon()`** method, before calling the **`start()`** method
    - Daemon threads die when the threads they support die
- Threads belong to ThreadGroups
  - ThreadGroups contain Threads and (child) ThreadGroups

Problems with Shared Data

- When multiple threads share data
  - Output may depend on which thread executes, in what order
    - Race Condition
  - Could result in unintended processing, incorrect results
- Example: Two simultaneous withdrawals from bank

task: withdraw 300		balance		task: withdraw 300
Is amount <= balance	→	531	←	Is amount <= balance
YES				YES
balance -= 300	→	231		
		-69	←	balance -= 300

Synchronization

- Multiple threads of control can be made safe if areas of code that use shared data are *synchronized*
- When a set of code is synchronized, then only one thread can be using that code at a time
- The other threads must wait until the first thread is complete
- This is an implementation of a synchronization mechanism called a *monitor*

Synchronized Bank Application

```
public class ATM_Accounts {
   public static void main (String[] args) {
      Savings_Account savings = new Savings_Account (4321, 531);
      ATM west_branch = new ATM (savings);
      ATM east_branch = new ATM (savings);
      west_branch.start();
      east_branch.start();
   }  // method main
}  // class ATM_Accounts

class ATM extends Thread {
   Savings_Account account;
   public ATM (Savings_Account savings) {
      account = savings;
   }  // constructor ATM
   public void run () {
      account.withdrawal (300);
   }  // method run
}  // class ATM
```


Bank Application (cont.)

```
class Savings_Account {
   protected int account;
   protected int balance;
   public Savings_Account (int account_num, int initial) {
      account = account_num;
      balance = initial;
   }  // constructor Savings_Account
   public synchronized boolean withdrawal (int amount) {
      boolean result = false;
      System.out.println ("Withdrawing Amount: " + amount);
      if (amount <= balance) {
         balance -= amount;
         System.out.println ("New balance: " + balance);
         result = true;
      } else
         System.out.println ("Insufficient funds.");
      System.out.println();
      return result;
   }  // method withdrawal
}  // class Savings_Account
```

How Synchronization Works

- Every object has a bit known as its monitor
  - While inside a synchronized method of an object, the object's bit is set to "unavailable"; otherwise, it's set to "available"
  - Only one synchronized method of an object can be called at any time

```
public synchronized boolean withdrawal (int amount) { … }
public synchronized void deposit (int amount) { … }
```

  - Can also set the monitor bit of an object using synchronized statement

```
synchronized (object-expression) protected-statement
```


Example Uses of Threads

- Animation Examples
  - jdk1.1.5/demo/Animator
  - jdk1.1.5/demo/Blink
  - jdk1.1.5/demo/SortDemo
  - jdk1.1.5/demo/GraphLayout
- Database Applications
  - Example
    - Run a long database query
    - Can still interact with user
    - Or formulate next query
- Network Applications
  - Example
    - While downloading a Web page, can do other things