

Concurrent Programming: Threads

Threads

- Threads are Objects, subclasses of Thread (or implement Runnable)
- The class for a thread must implement the run method
- To run a thread:
 - Create an object of the thread class
 - Call the start method
 - start makes run execute
 - run typically consists of a loop
- The Thread and Object classes provide methods to manage and synchronize threads

2

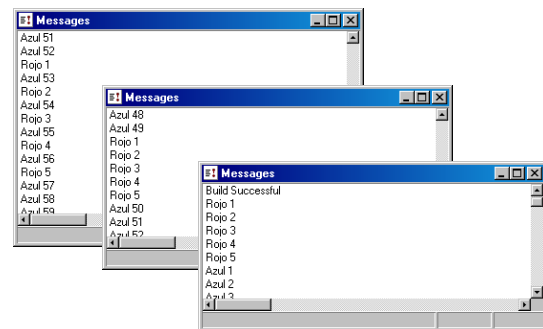
Creating a Thread: the Thread Class

```
class Repetition extends Thread {
    private int repetitions;
    private String message;
    Repetition (String msg, int n) {
        message = msg; repetitions = n;
    }
    public void run () {
        for (int i = 1; i <= repetitions; i++)
            System.out.println (message + " " + i);
    }
}

public static void main (String args[]) {
    Repetition r1 = new Repetition ("Red", 5);
    Repetition r2 = new Repetition ("Blue", 80);
    r1.start ();
    r2.start ();
}
```

3

Execution of Concurrent Threads



4

Creating a Thread: the Runnable Interface

```
class Repetition2 implements Runnable {
    private int repetitions;
    private String message;
    Repetition2 (String msg, int n) {
        message = msg; repetitions = n;
    }
    public void run () {
        for (int i = 1; i <= repetitions; i++)
            System.out.println (message + " " + i);
    }
}

public static void main (String args[]) {
    Repetition r1 = new Repetition ("Red", 5);
    Thread r2 = new Thread (new Repetition2 ("Blue", 80));
    r1.start ();
    r2.start ();
}
```

5

Mutual Exclusion: synchronized methods

- Avoid interferences between threads that manipulate the same object
- synchronized methods of the same object cannot execute simultaneously in two concurrent threads
- The first thread that invokes one of the methods locks the object
- The next threads to call the method remain blocked in the meantime
- The first thread unlocks the object when:
 - It gets out of the method
 - It pauses the execution with sleep or wait
- At this time the other threads compete to continue
 - Only one gets to do so
 - The other ones go back to waiting
- synchronized can also apply to blocks of sentences

6

Mutual Exclusion: Example

```
class Seat {
    Passenger passenger = null;
    boolean bookd () { return passenger == null; }
    void book (Passenger p) { passenger = p; }
}

class Flight {
    Seat seats[];
    Flight (...) { ... }
    synchronized void book (Passenger p) {
        for (int i = 0; i < seats.length; i++)
            if (!seats[i].bookd ())
                seats[i].book (p);
    }
}
```

7

```
class SellingPoint extends Thread {
    public void run () {
        Flight v;
        Passenger p;
        ...
        v.book (p);
        ...
    }
}
```

```
public static void main (String args[]) {
    SellingPoint t1 = new SellingPoint (...);
    SellingPoint t2 = new SellingPoint (...);
    t1.start ();
    t2.start ();
}
```

8

```
class SellingPoint extends Thread {
    public void run () {
        Flight v;
        Passenger p;
        ...
        // If book is not synchronized
        synchronized (v) {
            v.book (p);
        }
        ...
    }
}
```

9

Thread Coordination: Example

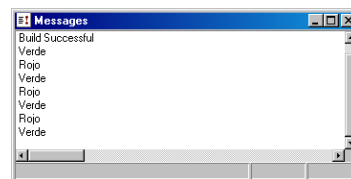
```
class Pedestrian extends Thread {
    TrafficLight trafficLight;
    Pedestrian (TrafficLight s) { trafficLight = s; }
    public void run () {
        while (true)
            if (trafficLight.getColor () .equals ("Red"))
                trafficLight.setColor ("Green");
    }
}
```

```
class LightsController extends Thread {
    TrafficLight light;
    LightsController (TrafficLight s) { light = s; }
    public void run () {
        while (true)
            if (light.getColor () .equals ("Green"))
                light.setColor ("Red");
    }
}
```

```
class TrafficLight {
    private String color = "Red";
    void getColor () { return color; }
    synchronized void setColor (String str) {
        try {
            wait (2000); // 2 sec. delay
            color = str;
            System.out.println (color);
            notifyAll ();
            wait ();
        } catch (InterruptedException ex) {}
    }
}
```

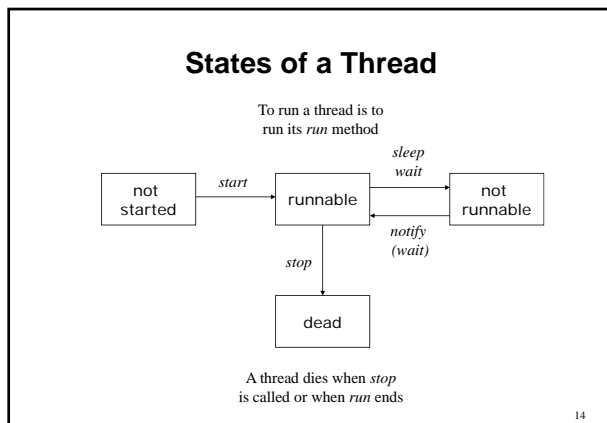
11

```
public static void main (String args[]) {
    TrafficLight light = new TrafficLight ();
    LightsController controller =
        new LightsController (light);
    Pedestrian p = new Pedestrian (light);
    p.start ();
    controller.start ();
}
```



12

```
// Avoid the other thread to change the light during
// the 2 sec. delay before the light change
class TrafficLight {
    private String color = "Red";
    private boolean available = true;
    synchronized void setColor (String str) {
        try {
            while (!available) wait ();
            available = false;
            wait (2000);
            color = str;
            System.out.println (color);
            available = true;
            notifyAll ();
            wait ();
        } catch (InterruptedException ex) {}
    }
}
```



- ### Methods to Manage a Thread's State
- **Runnable**
 - run ()
 - **Thread** implements Runnable
 - start ()
 - stop ()
 - sleep (long)
 - **Object** (must be called from a synchronized method)
 - wait (), wait (long)
 - notify (), notifyAll ()

Make a Thread Stand Idle: wait vs. sleep

sleep	wait
<ul style="list-style-type: none"> ▪ Is called on a Thread or on the Thread class 	<ul style="list-style-type: none"> ▪ Is called on an Object ▪ Can only be called from a synchronized method
<ul style="list-style-type: none"> ▪ The thread cannot be awoken until the sleep time elapses 	<ul style="list-style-type: none"> ▪ Can be awakened with notify from any other synchronized method of the same object
<ul style="list-style-type: none"> ▪ Exits after a time lapse 	<ul style="list-style-type: none"> ▪ wait() ends with notify() ▪ wait(n) returns when n msec. elapse, or with notify()

End a Thread with stop

Deprecated!

```
class MyThread extends Thread {
    public void run () {
        while (true) {
            ...
        }
    }
}

public static void main (String args[] ) {
    MyThread h = new MyThread ();
    h.start ();
    ...
    h.stop ();
}
```

End a Thread by Making run Finish

```
class MyThread extends Thread {
    boolean cont;
    public void run () {
        cont = true;
        while (cont) {
            ...
        }
    }
}

public static void main (String args[] ) {
    MyThread h = new MyThread ();
    h.start ();
    ...
    h.cont = false;
}
```

Thread Priority

- Assigning priorities: `setPriority (int)`
 - `Thread.MIN_PRIORITY` 1
 - `Thread.NORM_PRIORITY` 5
 - `Thread.MAX_PRIORITY` 10
- A thread stops running only when:
 - It is no longer runnable
 - A higher priority thread is started
 - In some OSs, when the scheduler activates another thread of same priority
- Dependence on OS

19

Other Thread Methods

- `Thread.currentThread()`
- `setName(String), toString()`
- `setDaemon(boolean)`
- `yield(), interrupt(), suspend(), resume(), destroy()`
- `isAlive()`
- `getThreadGroup()`

20

Summary

- Threads can be created subclassing `Thread` (or implementing `Runnable`) and defining `run`
- `synchronized` methods are mutually excluding for the same object
- With `sleep`, `wait`, `notify`, and using priorities, threads can be coordinated
- The interaction between threads is tricky, and there is no general rule: it is usually handled by foreseeing all possible combinations
- It is the programmer's responsibility to prevent deadlocks

21