Lecture 8: Java Resources Language

- Introduction
- Description of functionality of Java Resources (JR)
- Hello World Implementation
- Bounded Buffer Using Monitors in JR
- JR Virtual Machines
- Remote objects on Virtual Machines in JR
- Producer Consumer with ASMP in JR
- Readers/Writers in JR

CA463D Lecture Notes (Martin Crane 2013)

21

Java Resources Language

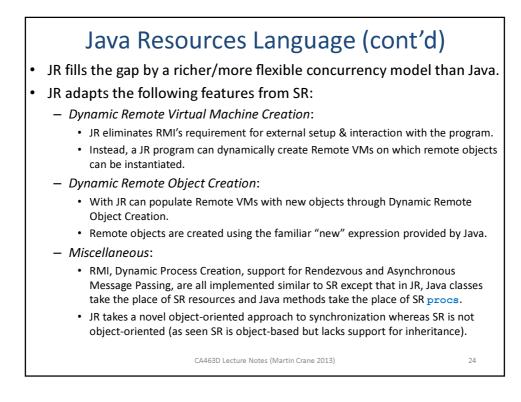
- As seen above, a concurrent program specifies two or more processes working together to perform a task.
- Each process consists of a sequential program.
- The processes interact by communicating, giving rise to the need for synchronization.
- Communication/synchronization are programmed by reading/ writing shared variables or sending/receiving messages:
 - Shared variables are most appropriate for concurrent code executing on a shared-memory multiprocessor.
 - Message passing is most appropriate for distributed programs that execute on HPC Clusters or networks of workstations.
- Message passing can also be used on shared-memory machines.

CA463D Lecture Notes (Martin Crane 2013)

2

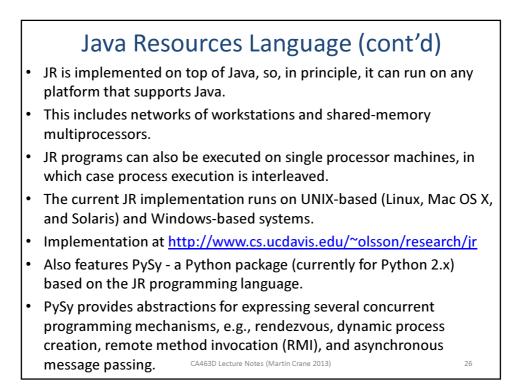
Java Resources Language (cont'd)

- JR is an extension of the Java programming language with additional concurrency mechanisms based on those in the SR (Synchronizing Resources) programming language.
- Can also be used as a Distributed Language
- Java has proven to be a clean and simple (and popular) language for object-oriented programming.
- Even so, the standard Java concurrency model is limited.
- As seen above, it provides threads, a primitive monitor-like mechanism, and remote method invocation (RMI).
- Though the basic functionality has been supplemented in Java 5.0, aspects of SR still remain that are difficult to implement using the basic Java package.



Java Resources Language (cont'd)

- So JR inherits & extends SR's powerful attribute: its expressive power.
- Communication, synchronization mechanisms listed above include the most popular and useful.
- This makes JR suitable for writing concurrent programs for both shared- and distributed-memory applications and machines.
- As well as being expressive, its easy to use for those who know Java.
- Its variety of concurrent programming mechanisms is based on only a few underlying concepts.
- These concepts are generalizations of ones that have been found useful in sequential programs.
- The concurrency mechanisms are also integrated with the sequential ones, so that similar things are expressed in similar ways.
- An important design goal has been to retain the "feel" of Java while providing a richer concurrency model.
 CA453D Lecture Notes (Martin Grane 2013)



27

JR Basics: A "Hello World" Program

```
public class HelloProcess{
   static process Hello = ((int id = 0; id < 25; id++)) {
      System.out.println("Hello World from thread " + id);
      }
   public static void main string args[] { }
}</pre>
```

• This Hello World Program kicks off 25 Hello process

- First thing to note about the **process** is that they are similar but not the same as java threads.
- Confusingly, a multithreaded Java program and a multiprocess JR program both run as a single OS process.
- Instantiating a process is simpler than in Java i.e without need of instantiation, just in a declarative way letting JR handle the rest.
- To declare a process, JR introduces a new keyword process
- So a very simple bit of JR code for declaring a JR Process could be:

process Hello {System.out.println("It's a Process!");}

CA463D Lecture Notes (Martin Crane 2013)

Bounded Buffer Using Monitors in JR monitor BoundedBuffer { _var String[] buffer = new String[N];_var int front; ____var int rear; __var int count; condvar notFull; condvar notEmpty; _proc void deposit(String data) { while(count == N) { _wait(notFull); buffer[rear] = data; rear = (rear + 1) % N; count++; signal(notEmpty); 3 proc String fetch() { while(count == 0) { _wait(notEmpty); String result = buffer[front]; front = (front + 1) % N;
count--; signal(notFull); _____return result; CA463D Lecture Notes (Martin Crane 2013) 28

