Parallel programming in Java

- Java has 2 forms of support for parallel programming

  Multithreading
  Remote Method Invocation (RMI)

- Multithreading

  Multiple threads of control (subprocesses)
  Useful for

    Pseudo-parallellism within a single machine
    Real parallellism on shared-memory machine

- Remote Method Invocation

  Allows invocation on an object located at another machine
  Useful for distributed-memory machines

Multithreading

A thread has

- Its own program counter
- Its own local variables

All threads on same Java Virtual Machine share global variables

Threads can communicate through shared variables

Threads can run concurrently (on multiprocessor) or are time-sliced

Creating threads in Java

```
public class mythread implements Runnable {
    public void hi() {
        System.out.println("hi");
    }
    public void run() {
        System.out.println("hello");
    }
}
mythread t1 = new mythread(); // allocates a thread
mythread t2 = new mythread(); // allocates another thread
t1.run();  // starts first thread
t2.run();  // starts second thread
t1.hi();
```

Thread synchronization

Problem-1:

Thread-1 does: $X = X + 1$;

Thread-2 does: $X = X + 2$;

Result should be $+3$, not $+1$ or $+2$.

Need to prevent concurrent access to same data: mutual exclusion synchronization

Mutual exclusion in Java

```
public class example {
    int value;

    public synchronized increment (int amount) {
        value = value + amount;
    }
}
```

The *synchronized* keyword guarantees that only one call to *increment* is executed at a time

More thread synchronization

Problem-2:

- Sometimes threads have to wait for each other
- Condition synchronization
- Supported in Java with wait/notify/notifyAll

wait blocks (suspends) a thread

notify wakes up (resumes) one blocked thread

notifyAll wakes up all blocked threads

Example: bounded buffer

```
public class BoundedBuffer {
    int buf[SIZE], count, writepos, readpos = 0; // shared
    public synchronized void put(int x) {
        while (count == SIZE) wait(); //block if buffer full
        buf[writepos] = x; count++;
        writepos = (writepos + 1) mod SIZE;
        if (count == 1) notifyAll();}
    public synchronized int get() {
        int x; // local
        while (count == 0) wait(); //block if buffer empty
        x = buf[readpos]; count--;
        readpos = (readpos + 1) mod SIZE;
        if (count == SIZE-1) notifyAll();
        return x;}
```

Remote Method Invocation

RMI is two-way synchronous communication, much like RPC

RMI invokes a method on a (possibly) remote object

Integrates cleanly into Java's object-oriented model

Example

```
public interface Hello extends Remote {
   String sayHello();
}

public class HelloImpl extends UnicastRemoteObject
                                implements Hello {
   public String sayHello() {
      return "Hellow World!;
   }
}
```

Asynchronous communication in Java

Can you do asynchronous messages passing in Java?

Yes: create a new thread to do the RMI for you and wait for the result

Awkward to program, performance overhead

More information

Tutorials about multithreading and RMI are available at the web site of the practicum: http://www.cs.vu.nl/pp-cursus