**Programación concurrente en el lenguaje Java**

## Presentado por developerWorks, su fuente de grandes clases particulares

**ibm.com/developerWorks**

# Tabla de contenido

Si está viendo este documento en línea, puede hacer clic en cualquiera de los temas a continuación para acceder directamente a esa sección.

## [Acerca de esta tutorial](#_bookmark0) 2

## [Introducción](#_bookmark1) 3

## [Comenzando Java trapos](#_bookmark3) 4

## [Hilo estados, prioridades, y métodos](#_bookmark8) 9

## [los volátil modificador](#_bookmark15) 18

## [Carrera condiciones](#_bookmark19) 24

## [sincronizada bloques](#_bookmark26) 30

## [monitores](#_bookmark31) 33

## [Los semáforos](#_bookmark40) 50

## [Mensaje paso](#_bookmark51) 60

## [Cita](#_bookmark60) 72

## [Remoto Método Invocación (RMI)](#_bookmark69) 98

## [Envolver](#_bookmark83) 122

# Sección 1. Sobre este tutorial ¿Debo tomar este tutorial?

### Una de las características más importantes del lenguaje Java es el soporte para multiproceso (también llamado

concurrente) de programación.

This tutorial is an introduction to the use of multiple threads in a Java program and will appeal to systems or application programmers who want to learn about multithreaded Java programming.

A multithreaded program can take advantage of the additional CPUs in a shared-memory multiprocessor architecture in order to execute more quickly. The use of multiple threads can also simplify the design of a program. As an example, consider a server program in which each incoming client request is handled by a dedicated thread.

However, to avoid race conditions and corruption of shared data, the threads in a concurrent program must be properly synchronized. Many example programs are used in this tutorial to illustrate these concepts.

Este tutorial asume un conocimiento general previo de programación Java; el contexto y el nivel de conocimiento usado en este tutorial es el equivalente de un curso de sistemas operativos de grado. Para una explicación más explícita de la experiencia necesaria para obtener el máximo provecho de este tutorial, consulte[supuestos y contexto](#_bookmark2) en la página 3.

Sobre el Autor

Stephen J. Hartley es Profesor Asociado de Ciencias de la Computación de la Universidad de Rowan en Glassboro, Nueva Jersey. Ha publicado dos libros con Oxford University Press sobre el tema de la programación concurrente: Programación Sistemas Operativos: el lenguaje de programación SR (1995) y programación concurrente: el lenguaje de programación Java (1998). Póngase en contacto con Stephen en[*hartley@elvis.rowan.edu*](mailto:hartley@elvis.rowan.edu).

### Si tiene preguntas sobre el contenido de este tutorial, por favor, póngase en contacto con el autor.

Sección 2. Introducción Metas y objetivos

The two goals of this tutorial are to:

* Learn the "nuts and bolts" of creating multiple threads of control in a Java program.
* Learn the pitfalls and areas that can trip you up when synchronizing those threads to avoid race conditions and corruption of shared data.

Assumptions and context

Before we move into the nuts and bolts of concurrent programming, let's elaborate on the knowledge and experience necessary to effectively wring the most use from this tutorial.

We assume that you have a general knowledge of concurrency issues, at a level commensurate with an undergraduate computer science operating systems course. We also assume a familiarity with the following terms and concepts: multiple threads, shared data, race conditions, critical sections, mutual exclusion, monitors, and semaphores. Finally, you should have knowledge of object-oriented programming and sequential Java: classes, objects, interfaces, inheritance, polymorphism, packages, and exceptions.

For further reference, [Resources](#_bookmark84) on page 122 at the end of the tutorial includes online and print resources on concurrent programming and the Java language.

Tutorial platform specifications

All example Java programs in this tutorial have been executed on a PC running [*Red Hat's*](http://www.redhat.com/)[*version 7.0 of Linux*](http://www.redhat.com/), using the [*IBM Java software developer kit version 1.3.0 for Linux*](http://www.ibm.com/developerworks/java/jdk/linux130/). Este kit de desarrollador utiliza subprocesos nativos y por lo tanto el tiempo les rebana automáticamente.

### Los ejemplos son compilados y ejecutados con el compilador Just-In-Time (JIT) discapacitados (comando de exportación JAVA\_COMPILER = NINGUNO) para facilitar la manifestación de las condiciones de carrera en [Ejemplo race.java](#_bookmark20) en la página 24 y[Ejemplo rac2.java](#_bookmark22) en la página 25.

Puede descargar un archivo zip que contiene todos los programas de ejemplo de Java en este tutorial en [recursos](#_bookmark84) en la página 122.

Sección 3. A partir de las hebras Java Dos formas de iniciar las hebras Java

Hay dos formas de iniciar las hebras Java. Una forma es subclase de la clase Thread:

Una clase se extiende de rosca {public void run () {

... // código para el nuevo hilo para ejecutar

}

}

...

* 1. **a = new A (); // crear el objeto hilo a.Inicie (); // inicia el nuevo hilo ejecutar**

...

### La segunda manera es implementar la interfaz Ejecutable:

la clase B se extiende ... implements Runnable {public void run () {

... // código para el nuevo hilo para ejecutar

}

}

...

* 1. **b = new B();// create the Runnable object Thread t = new Thread(b); // create a thread object t.start();// start the new thread**

...

### [Example prit.java](#_bookmark4) on page 6 demonstrates multithreaded prime number generation with one thread per number checked. [Sample run of prit.java](#_bookmark5) on page 7 shows the results. The [Class](#_bookmark6) [Prime.java](#_bookmark6) on page 7 is used. [Unit testing of Prime.java](#_bookmark7) on page 8 demonstrates unit testing of the Prime.java class.

Background material on threads

First a quick refresher on threading.

A process is an executing program. It has been allocated memory by the operating system. A thread is an execution or flow of control in the address space of a process; the program counter register points to the next instruction to be executed.

Un proceso es un programa con al menos un hilo. Un proceso puede tener más de un hilo. Todos los hilos en un proceso tienen su propio contador de programa y su propia pila para las variables locales (también llamadas automáticas) y las direcciones de los procedimientos invocados regresan.

En el lenguaje Java, un hilo en el intérprete de tiempo de ejecución llama al método main () de la clase en la línea de comandos de Java. Cada objeto creado puede tener uno o más hilos, los cuales comparten el acceso a los campos de datos del objeto.

El artículo "[*Un Introducción a Programación con Trapos*](ftp://gatekeeper.dec.com/pub/DEC/SRC/research-reports/SRC-035.ps.Z)"Por Andrew D. Birrell (1989, un informe de investigación DEC) ofrece los siguientes motivaciones para la programación concurrente con hilos:

### multiprocesadores de memoria compartida son más baratos y más común de modo que cada hilo se pueden asignar una CPU.

* It is less expensive and more efficient to create several threads in one process that share data than to create several processes that share data.
* I/O on slow devices (such as networks, terminals, and disks) can be done in one thread while another thread does useful computation in parallel.
* Multiple threads can handle the events (such as mouse clicks) in multiple windows in the windowing system on a workstation.
* In a LAN cluster of workstations or in a distributed operating system environment, a server running on one machine can spawn a thread to handle an incoming request in parallel with the main thread continuing to accept additional incoming requests.

Cuando dos hilos realizan una función tal como N = N + 1 en aproximadamente el mismo tiempo, tiene una condición de carrera. Ambos hilos son "de carreras" entre sí para acceder a los datos y una de las actualizaciones puede perderse. En general, las condiciones de carrera son posibles cuando dos o más hilos de compartir datos, que están leyendo y escribiendo los datos compartidos al mismo tiempo, y el resultado final depende de lo que uno hace cuando.

Al mismo tiempo la ejecución de hilos que comparten datos necesitan sincronizar sus operaciones y tratamiento para evitar las condiciones de carrera sobre los datos compartidos. la sincronización de hilos se puede hacer con las variables de bandera y espera ocupada. Debido a que utiliza una gran cantidad de ciclos de CPU, espera ocupada es ineficiente. El bloqueo sería mejor.

A critical section is a block of code in a thread that accesses one or more shared variables in a read-update-write fashion. In such a situation we want mutual exclusion in which only one thread can access (read-update-write) a shared variable at a time.

The mutual exclusion problem is how to keep two or more threads from being in their critical sections at the same time, where we make no assumptions about the number of CPUs or their relative speeds.

A thread outside its critical section should not keep other threads outside their critical sections from entering. This is also called a safety property (or absence of unnecessary delay).

Also, no thread should have to wait forever to enter its critical section. This is also called a

*liveness* property (or eventual entry).

### Andrews caracteriza una acción atómica como una que "hace una transición del estado indivisible: cualquier estado intermedio que pudiera existir en la ejecución de la acción no debe ser visible para otros temas." Esto significa que nada de otro hilo puede ser intercalada en la ejecución de la acción para que sea atómica.

Las secciones críticas necesitan ser definidos como si fueran una sola acción atómica para evitar condiciones de carrera.

Aquí están las cuestiones básicas a tener en cuenta para resolver el problema de la exclusión mutua cuando

la elaboración de un protocolo previo y un post-protocolo basado en hardware o software. Los protocolos deben:

* Evitar que dos hilos de estar en sus secciones críticas al mismo tiempo
* Tener la seguridad y las propiedades deseables LIVENESS
* Permitir a las secciones críticas que se ejecuten de forma atómica reglas básicas del sistema son los siguientes:
* It is a load/store register architecture.
* Multiple, concurrently executing threads are sharing data.
* There are single or multiple CPUs and we cannot make relative speed assumptions.
* Access to shared variables can be interleaved if two threads are in their critical sections at the same time.
* Threads may not halt in their pre- or post-protocols.
* Threads may not halt in their critical sections.
* Threads may halt outside their critical sections.

Thread Ti, i = 1, 2, 3, ...

while (true) { outsideCS();

wantToEnterCS(i);// pre-protocol insideCS();

finishedInCS(i);// post-protocol

}

### The next several panels display the code described in this section. To view the code, click Next; or you can go directly to the next section, [Thread states, priorities, and methods](#_bookmark8) on page 9 , and return to the code samples at another time.

Example prit.java

class PrimeThread extends Thread { private int m = 0; PrimeThread(int m) { this.m = m; } public void run() {

if (Prime.prime(m)) System.out.println(m + " is prime");

}

}

class TestPrimeThreads {

public static void main(String[] args) { int n1 = 0, n2 = 0;

try {

n1 = Integer.parseInt(args[0]); n2 = Integer.parseInt(args[1]);

} catch (NumberFormatException e) { System.out.println("improper format"); System.exit(1);

} catch (ArrayIndexOutOfBoundsException e) { System.out.println("not enough command line arguments"); System.exit(1);

}

if (n1 < 2 || n2 < 2 || n1 > n2) { System.out.println("illegal command line arguments "

+ n1 + ", " + n2 );

System.exit(1);

}

System.out.println("printing primes from "

+ n1 + " to " + n2);

nuevo PseudoTimeSlicing (); // para Solaris, no Windows 95 / NT for (int i = n1; i <= n2; i ++) {

Thread t = new PrimeThread (i); t.Start ();

}

}

}

# muestra de ejecución de prit.java

prit.java% javac

% TestPrimeThreads java 10 20

números primos de impresión de 10 a 20 versión Java = 1.3.0

proveedor de Java = IBM Corporation Nombre del sistema operativo Linux =

arco OS = i586

versión del sistema operativo = # 1 Lun Sep 27 de 10:25:54 EDT 1999.2.2.12-20

No es necesario PseudoTimeSlicing

11 es un número primo

13 es un número primo

17 es un número primo

19 es un número primo

% TestPrimeThreads java 1000000 1000060

primos de impresión desde 1000000 a 1000060 versión de Java 1.3.0 =

proveedor de Java = IBM Corporation Nombre del sistema operativo Linux =

arco OS = i586

versión del sistema operativo = # 1 Lun Sep 27 de 10:25:54 EDT 1999.2.2.12-20

No se necesita PseudoTimeSlicing 1000003 es primo

1000033 es primo

1000037 es primo

1000039 es primo

# clase Prime.java

public class Primer {

public static boolean prime (int k) {if (k <2) return false;

int límite = k / 2;

for (int i = 2; i <= k / 2; i ++) {if ((k% i) == 0) return false;

}

return true;

}

void main (String [] args) public static {int n = 0;

tratar {

n = Integer.parseInt (args [0]);

} Catch (NumberFormatException e) {

System.out.println ( "formato incorrecto"); System.exit (1);

} Catch (ArrayIndexOutOfBoundsException e) {System.out.println ( "ningún argumento de línea de comandos"); System.exit (1);

}

si (n <2) {

System.out.println ( "argumento de línea de comando" + n

+ "Es demasiado pequeño"); System.exit (1);

}

System.out.println ( "números primos de impresión de 2 a" + n); for (int i = 2; i <= n; i ++) {

si (Prime.prime (i)) System.out.println (i + "es primo");

}

}

}

# Prueba de la unidad de Prime.java

Prime.java% javac

Primer% java

ningún argumento de línea de comandos

Java% abc primer formato incorrecto

Primer% java 0

línea de argumento de comando 0 es demasiado pequeña

Primer% java 10

impresión de números primos de 2 a 10

1. **es primo**
2. **es primo**

5 es primo

7 es primo

# Section 4. Thread states, priorities, and methods Thread states

### Thread states are defined as:

* *New* before the thread's start() method is called
* *Runnable* if the thread is in the ready queue

### *Running* if the thread is executing on the CPU

* *Dead* after the thread's run() method completes or stop() method is called
* *Blocked* if the thread is blocked on I/O, a join() method call, or a sleep(ms)method call
* *Suspended* if the thread's suspend() method is called from the running or runnable

### states

* *Suspended-blocked* if the thread's suspend() method is called from the blocked state

# Thread priorities

### Thread priorities (class variables) are:

* **MAX\_PRIORITY**
* **NORM\_PRIORITY**
* **MIN\_PRIORITY**

Priority set and get instance methods include:

* **setPriority(Thread.***priority***)**
* **getPriority()**

### El planificador JVM normalmente permite garantizar la máxima prioridad del hilo se está ejecutando en la CPU, anticipándose a la rosca actualmente en ejecución cuando sea necesario, pero esto no es una garantía. (Véase la página 415 de la especificación del lenguaje Java, en[recursos](#_bookmark84) en la página 122).

segmentación de tiempo

segmentación de tiempo de hilos también se conoce como programación round-robin. Se realiza mediante la JVM. El IBM JDK 1.3.0 para Linux y las ventanas de Microsoft JVM realizar esta tarea; la versión de Solaris no lo hace.

los [Clase PseudoTimeSlicing.java](#_bookmark9) en la página 11 implementa "pseudo" segmentación de tiempo. El uso de esta clase no garantiza la segmentación de tiempo, pero funciona en la práctica.

métodos de la clase hilo

Los siguientes métodos son estáticos en la clase Thread y se aplican a la rosca llamando a:

* **Thread.sleep (***Sra***)** bloquea el subproceso de llamada durante el tiempo especificado.

### En Thread.yield (), el subproceso de llamada renuncia a la CPU (pero no está garantizada por la

*JLS*).

* Cualquier método puede utilizar Thread.currentThread () para obtener una referencia al hilo que llama el método, por ejemplo, Thread.currentThread (). GetPriority () ;.
* Utilice Thread.interrupted () para ver si el método de la rosca de interrupción () se ha llamado (que borra la bandera interrumpe).

# Los métodos de instancia

### Y aquí están los métodos de instancia (por ejemplo, t.Start () en el que t es una variable de referencia a un objeto Thread):

* **comienzo()**: Iniciar un nuevo hilo de ejecutar el método run ().

### **detener()**: Terminar el hilo (en desuso, no utilizar).

* **suspender()**: Suspender el hilo (en desuso, no utilizar).
* **currículum()**: Reanudar el hilo suspendido (en desuso, no utilizar).
* **unirse()**: Registrado con otro hilo cuando ésta termina.
* **interrumpir()**: Dile al hilo para comprobar si hay un cambio en lo que debería estar haciendo.
* **isInterrupted()**: Check if the thread's interrupt() method has been called (this does not clear the interrupted flag).

### **isAlive()**: Check if the thread has terminated.

* **setDaemon(boolean)**: Make the thread a daemon (the JVM ignores this thread when determining if all threads in a program have terminated).
* **isDaemon()**: Check if the thread is a daemon.
* **setPriority(int)**: Change the priority of the thread.
* **getPriority()**: Return the prioritiy of the thread.
* **setName(string)**: Change the name of the thread to be equal to the argument name.
* **getName()**: Return the name of the thread.

# Examples

### The following example programs demonstrate what we've just been covering.

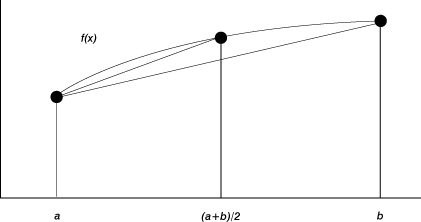
[Class Sugar.java](#_bookmark10) on page 12 is "syntactic sugar." It provides class methodage(), which returns the number of milliseconds since the program started, and class method random(range).

[Example beep.java](#_bookmark11) on page 13 allows you to test a platform for timeslicing.[Sample run of](#_bookmark12) [beep.java](#_bookmark12) on page 14 shows the results.

[Example quad.java](#_bookmark13) on page 15 implements adaptive quadrature numerical integration with multiple threads and uses join() for synchronization. If the sum of the areas of the two sub-trapezoids is not close enough to the area of the trapezoid containing them, two threads are spawned to repeat the calculation for each sub-trapezoid. The spawning thread cannot

continue until each of the two spawned threads finishes. [Sample run of quad.java](#_bookmark14) on page 16 shows the results.

This figure shows the implementation of adaptive quadrature numerical integration with multiple threads.



The next several panels display the code described in this section. To view the code, click Next; or you can go directly to the next section, [The volatile modifier](#_bookmark15) on page 18 , and return to the code samples at another time.

Class PseudoTimeSlicing.java

public class PseudoTimeSlicing implements Runnable {

private static final String

JAVA\_VERSION = System.getProperty("java.version"), JAVA\_VENDOR = System.getProperty("java.vendor"), OS\_NAME = System.getProperty("os.name"),

OS\_ARCH = System.getProperty("os.arch"), OS\_VERSION = System.getProperty("os.version");

private static Thread me = null; private static int timeSlice;

public PseudoTimeSlicing() { this(100); } public PseudoTimeSlicing(int ts) {

// See <http://www.javaworld.com/javaworld/jw-04-1999/>

// jw-04-toolbox\_p.html for the correct way to do singletons

// in a multithreaded situation. if (me == null) {

System.out.println("Java version=" + JAVA\_VERSION

+ "\nJava vendor=" + JAVA\_VENDOR

+ "\ Nombre nos =" + OS\_NAME + "\ arco nos =" + OS\_ARCH

+ "\ Versión nos =" + OS\_VERSION); si (OS\_NAME.equals ( "Solaris") ||

(OS\_NAME.equals ( "Linux") &&

! JAVA\_VENDOR.startsWith ( "IBM"))) {TimeSlice = ts;

Me = new Thread (this); me.setPriority (Thread.MAX\_PRIORITY); me.setDaemon (true);

me.start ();

System.out.println ( "PseudoTimeSlicing instalado");

} else

System.out.println ( "No se necesita PseudoTimeSlicing");

} Else System.out.println ( "PseudoTimeSlicing ya instalado");

}

public void run () {

si (Thread.currentThread () = yo!) return;

// este hilo de mayor prioridad despertar envía el

// se está ejecutando actualmente hilo de vuelta al conjunto runnable while (true) {

try {Thread.sleep (TimeSlice); }

captura (InterruptedException e) {/ \* ignorado \* /}

}

}

}

# clase Sugar.java

import java.util.Random; public abstract class Sugar {

private static final long startTime

= System.currentTimeMillis();

private static final Random rnd = new Random();

// utility methods

protected static final long age() {

return System.currentTimeMillis() - startTime;

}

protected static final double random() { return rnd.nextDouble(); // in range [0, 1)

}

protected static final double random(int ub) { return rnd.nextDouble()\*ub; // in range [0, ub)

}

protected static final double random(int lb, int ub) {

return lb + rnd.nextDouble()\*(ub - lb); // in range [lb, ub)

}

}

# Example beep.java

class Beeper extends Sugar implements Runnable { private int beep = 0;

private String name = null;

public Beeper(String name, int beep) { this.name = name;

this.beep = beep;

System.out.println(name + " is alive, beep=" + beep);

}

public void run() { long value = 1;

System.out.println ( "edad () =" + edad () + ""

+ + Nombre de "funcionamiento");

// () hilo para principal tiene prioridad y

// seguramente obtiene la CPU después de interrumpir () nos Thread.currentThread (). SetPriority (

. Thread.currentThread () getPriority () - 1); while (true) {

si (valor ++% pitido == 0) {System.out.println ( "edad () =" + edad ()

+ " "+ Nombre +" bips, valor =" + valor); si (Thread.interrupted ()) {

System.out.println ( "edad =" + edad () + ""

+ Nombre + "interrumpidas"); regreso;

}

}

}

}

}

clase de pitido se extiende azúcar {

public void (String [] args) {int numBeepers = 4;

int pitido = 100.000; Cadena TimeSlice = "no";

int tiempo de ejecución = 60; // por defecto en cuestión de segundos tratar {

numBeepers = Integer.parseInt (args [0]); bip = Integer.parseInt (args [1]); TimeSlice = args [2];

tiempo de ejecución = Integer.parseInt (args [3]);

} catch (Exception e) { /\* use defaults \*/ } System.out.println("Beeping: numBeepers=" + numBeepers

+ ", beep=" + beep + ", timeSlice=" + timeSlice

+ ", runTime=" + runTime); if (timeSlice.equals("yes"))

// for Solaris, not Windows 95/NT new PseudoTimeSlicing();

// start the Beeper threads

Thread[] b = new Thread[numBeepers]; for (int i = 0; i < numBeepers; i++)

b[i] = new Thread(new Beeper("Beeper"+i, beep)); for (int i = 0; i < numBeepers; i++) b[i].start(); System.out.println("All Beeper threads started");

// let the Beepers run for a while try {

Thread.sleep(runTime\*1000); System.out.println("age=" + age()

+ ", time to interrupt the Beepers and exit"); for (int i = 0; i < numBeepers; i++)

b[i].interrupt();

for (int i = 0; i < numBeepers; i++) b[i].join();

} catch (InterruptedException e) { /\* ignored \*/ } System.out.println("All Beeper threads interrupted"); System.exit(0);

}

}

# Sample run of beep.java

% javac beep.java

% java Beeping 4 100000 no 3

Beeping: numBeepers=4, beep=100000, timeSlice=no, runTime=3 Beeper0 is alive, beep=100000

Beeper1 is alive, beep=100000 Beeper2 is alive, beep=100000 Beeper3 is alive, beep=100000 age()=36, Beeper0 running age()=134, Beeper1 running age()=195, Beeper2 running All Beeper threads started age()=225, Beeper3 running

edad () = 374, Beeper3 pitidos, valor = 100001 edad () = 600, Beeper2 pitidos, valor = 100001 edad () = 769, Beeper1 pitidos, valor = 100001 edad () = 901, Beeper0 pitidos, valor = 100001 edad ( ) = 1050, Beeper0 pitidos, valor = 200001 edad () = 1125, Beeper1 pitidos, valor = 200001 edad () = 1287, Beeper3 pitidos, valor = 200001 edad () = 1392, Beeper2 pitidos, valor = 200001 edad () = 1671, Beeper1 pitidos, valor = 300001 edad () = 1757, Beeper0 pitidos, valor = 300001 edad () = 1988, Beeper3 pitidos, valor = 300001 edad () = 2063, Beeper2 pitidos, valor = 300001 edad () = 2209, Beeper2 pitidos, valor = 400001 edad () = 2426, Beeper0 pitidos, valor = 400001 edad () = 2499, Beeper1 pitidos, valor = 400001 edad () = 2688, Beeper3 pitidos, valor = 400001 edad () = 2979, pitidos Beeper1 , valor = 500001 edad () = 3073, Beeper0 pitidos, valor = 500001 edad () = 3219, Beeper0 pitidos, valor = 600001 edad = 3234, momento para interrumpir el Beepers y edad de salida () = 3290, Beeper2 pitidos, valor = 500.001 edad = 3,291,Beeper2 interrumpido

edad () = 3321, Beeper3 pitidos, valor = 500001 edad = 3,322, Beeper3 interrumpió edad () = 3444, Beeper1 pitidos, valor = 600001 edad = 3,445, Beeper1 interrumpió edad () = 3578, Beeper0 pitidos, valor = 700001 edad = 3578, Beeper0 interrumpido

Todos los hilos del indicador sonoro interrumpidos

% Java pitido 4 100000 sí 3

Pitidos: numBeepers = 4, bip = 100000, TimeSlice = sí, el tiempo de ejecución = 3 versión de Java 1.3.0 =

proveedor de Java = IBM Corporation Nombre del sistema operativo Linux =

arco OS = i586

versión del sistema operativo = # 1 Lun Sep 27 de 10:25:54 EDT 1999.2.2.12-20

No es necesario PseudoTimeSlicing Beeper0 está vivo, bip = 100000 Beeper1 está vivo, bip = 100000 Beeper2 está vivo, bip = 100000 Beeper3 está vivo, bip = 100000 edad () = 38, edad corriendo Beeper0 () = 168, Beeper1 consecutivo

edad () = 293, Beeper0 pitidos, valor = 100001 edad () = 439, Beeper0 pitidos, valor = 200001 edad () = 536, Beeper1 pitidos, valor = 100001 edad () = 618, Beeper2 correr

Todos los hilos del indicador sonoro comenzaron edad () = 741, Beeper3 consecutivo

age()=855, Beeper2 beeps, value=100001 age()=922, Beeper1 beeps, value=200001 age()=1195, Beeper0 beeps, value=300001 age()=1319, Beeper3 beeps, value=100001 age()=1562, Beeper2 beeps, value=200001 age()=1702, Beeper0 beeps, value=400001 age()=1977, Beeper2 beeps, value=300001 age()=2078, Beeper1 beeps, value=300001 age()=2225, Beeper1 beeps, value=400001 age()=2266, Beeper3 beeps, value=200001 age()=2413, Beeper3 beeps, value=300001 age()=2561, Beeper1 beeps, value=500001 age()=2740, Beeper2 beeps, value=400001 age()=2743, Beeper0 beeps, value=500001 age()=3021, Beeper2 beeps, value=500001 age()=3124, Beeper0 beeps, value=600001 age()=3185, Beeper3 beeps, value=400001 age()=3322, Beeper1 beeps, value=600001 age()=3672, Beeper0 beeps, value=700001 age()=3723, Beeper3 beeps, value=500001 age=3748, time to interrupt the Beepers and exit age()=3840, Beeper1 beeps, value=700001 age=3841, Beeper1 interrupted

edad () = 3906, Beeper2 pitidos, valor = 600001 edad = 3,906, Beeper2 interrumpió edad () = 4066, Beeper3 pitidos, valor = 600001 edad = 4,067, Beeper3 interrumpió edad () = 4108, Beeper0 pitidos, valor = 800001 edad = 4109, Beeper0 interrumpido

Todos los hilos del indicador sonoro interrumpidos

# Ejemplo quad.java

interfaz theFunction {

pública evaluar doble (doble x); public String toString ();

}

MyFunction clase implementa theFunction {

pública evaluar doble (doble x) {return x \* x; } Public String toString () {return "x \*\* 2"; }

}

clase zona se extiende Tema {

privada doble p, q, épsilon, resultado; TheFunction f privada;

La zona pública (doble a, doble b, eps dobles, theFunction fn) {p = a; q = b; epsilon = eps; f = fn;

}

doble getResult pública () {return resultado; } Privada doble trapezoidArea estática

(Doble p, q doble, theFunction f) {double área =

(Math.abs (qp)) / 2 \* (f.evaluate (p) + f.evaluate (q)); área de retorno;

}

public void run() {

double bigArea = trapezoidArea(p, q, f);

double leftSmallArea = trapezoidArea (p, ((p+q)/2), f); double rightSmallArea = trapezoidArea(((p+q)/2), q, f); double sumOfAreas = leftSmallArea + rightSmallArea; double relError = Math.abs(bigArea - sumOfAreas);

if (relError <= (epsilon \* sumOfAreas)) result = bigArea; else {

Area leftArea =new Area(p, (p+q)/2, epsilon, f); leftArea.start();

Area rightArea = new Area((p+q)/2, q, epsilon, f); rightArea.start();

try { leftArea.join(); }

catch (InterruptedException e) { /\* ignored \*/ } try { rightArea.join(); }

catch (InterruptedException e) { /\* ignored \*/ } result = leftArea.getResult() + rightArea.getResult();

}

}

}

class AdaptiveQuadrature {

public static void main(String[] args) { double a = 0, b = 0, epsilon = 0;

try {

a = (Double.valueOf (args [0])) doubleValue (.); b = (Double.valueOf (args [1])) doubleValue (.);

epsilon = (Double.valueOf (args [2])) doubleValue (.);

} Catch (NumberFormatException e) {System.out.println ( "formato incorrecto"); System.exit (1);

} Catch (ArrayIndexOutOfBoundsException e) {System.out.println ( "no los argumentos de línea de comandos suficiente"); System.exit (1);

}

si (b <= a || epsilon <= 0) {

System.err.println ( "b <= a || epsilon <= 0, la salida"); System.exit (1);

}

TheFunction fn = new MyFunction (); System.out.println ( "Adaptive cuadratura de" + Fn + "desde"

+ A + "a" + b + "con error relativo" + epsilon); área Area = nuevo en la zona (a, b, epsilon, fn);

nuevo PseudoTimeSlicing (); // para Solaris, no Windows 95 / NT area.start ();

try {area.join (); }

captura (InterruptedException e) {/ \* ignorado \* /} doble resultado = area.getResult (); System.out.println ( "número de" + fn + "=" + resultado); System.exit (0);

}

}

# muestra de ejecución de quad.java

quad.java% javac

% Java AdaptiveQuadrature 0,5 1,5 0,001

Cuadratura adaptativa de x \*\* 2 de 0,5 a 1,5 con error relativo 0,0010 versión Java = 1.3.0

proveedor de Java = IBM Corporation Nombre del sistema operativo Linux =

arco OS = i586

versión del sistema operativo = # 1 Lun Sep 27 de 10:25:54 EDT 1999.2.2.12-20

No es necesario PseudoTimeSlicing

Resultado para x \*\* 2 = 1.084136962890625

# Sección 5. El modificador volátil algunas definiciones

### El modificador volátil le dice al compilador que la variable se accede por más de una

hilo a la vez e inhibe inapropiados código optimizaciones por el compilador, como el almacenamiento en caché el valor de la variable en un registro de la CPU en lugar de actualizar la memoria principal con cada ejercicio para la variable.

La especificación del lenguaje Java garantiza que las actualizaciones de cualquier variable compartida por un hilo en particular son vistos por otros hilos en el orden realizada por ese hilo en particular. Sin embargo, el JLS no requiere otros hilos para ver cambios a diferentes variables compartidas en el orden realizada por el hilo actualización a menos que las variables se declaran volátil.

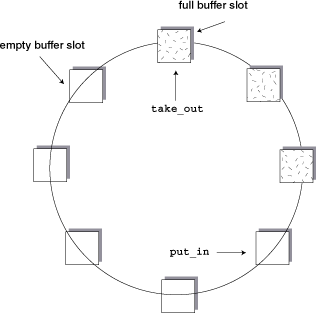
Los ejemplos del modificador volátil

[Ejemplo bwbb.java](#_bookmark16) en la página 19 implementa un búfer limitado de espera ocupado por un productor y el consumidor. Los artículos depósitos hilo productor y ocupado espera si el búfer limitado llena.

El hilo consumidor va a buscar artículos y ocupado espera si el búfer limitado está vacía.

El hilo productor debe "ver" el valor y campos ocupados actualizada por el hilo consumidor en el orden exacto de las actualizaciones se llevan a cabo por el hilo consumidor (o el consumidor debe ver las actualizaciones de productores en el orden realizado). Si no, el hilo productor podría sobrescribir un elemento en una ranura de tampón que el consumidor todavía no ha leído (o el consumidor podría leer de nuevo un elemento de una ranura tampón que ya ha leído).[Conductor bbdr.java](#_bookmark17) en la página 20 crea los hilos de productores y consumidores. [Muestra correr de bwbb.java](#_bookmark18) en la página 22 muestra los resultados de bwbb.java.

La figura a continuación ilustra esta interacción.



Los próximos varios paneles muestran el código descrito en esta sección. Para ver el código, haga clic en Siguiente; o puede ir directamente a la siguiente sección,[Carrera condiciones](#_bookmark19) en la página 24, y volver a los ejemplos de código en otro momento.

Ejemplo bwbb.java

class BufferItem {

// multiple threads access so make these 'volatile' public volatile double value = 0;

public volatile boolean occupied = false;

}

class BoundedBuffer {

// designed for a single producer thread and

// a single consumer thread private int numSlots = 0;

private BufferItem[] buffer = null; private int putIn = 0, takeOut = 0; public BoundedBuffer(int numSlots) {

if (numSlots <= 0)

throw new IllegalArgumentException("numSlots <= 0"); this.numSlots = numSlots;

buffer = new BufferItem[numSlots]; for (int i = 0; i < numSlots; i++)

buffer[i] = new BufferItem();

}

public void deposit(double value) throws InterruptedException {

while (buffer[putIn].occupied)// busy wait

Thread.currentThread().yield(); buffer[putIn].value = value; buffer[putIn].occupied = true; putIn = (putIn + 1) % numSlots;

}

public double fetch()

lanza InterruptedException {valor doble;

mientras (! Buffer [Comida para llevar] .occupied) // ocupados Espere Thread.currentThread () rendimiento (.);

value = buffer [Comida para llevar] .value; buffer [Comida para llevar] .occupied = false; Comida para llevar = (Comida para llevar + 1)% numSlots; valor de retorno;

}

}

# bbdr.java controlador

Productor clase extiende azúcar implementa Ejecutable {private String nombre = null;

int PNAP privada = 0; // milisegundos privado BoundedBuffer bb = null; Me privada = null rosca;

Productor pública (String nombre, int PNAP, BoundedBuffer bb) {this.name = nombre;

this.pNap = PNAP; this.bb = bb;

(Me = new Thread (este)) start ().;

}

pública timeToQuit void () {me.interrupt (); }

public void pauseTilDone () lanza InterruptedException

{Me.join (); } Public void run () {

si (Thread.currentThread () = yo!) return; doble elemento;

int napping; while (true) {

si (Thread.interrupted ()) {System.out.println ( "edad =" + edad () + "" + nombre

+ "Interrumpido"); regreso;

}

napping = 1 + (int) aleatorio (PNAP); System.out.println ( "edad =" + edad () + "" + nombre

+ "Siesta para" + "ms" + napping); try {Thread.sleep (napping); }

captura (InterruptedException e) {System.out.println ( "edad =" + edad () + "" + nombre

+ "Del sueño interrumpido"); regreso;

}

item = random ();

System.out.println ( "edad =" + edad () + "" + nombre

+ "Artículo producido" + artículo); try {bb.deposit (punto); }

captura (InterruptedException e) {System.out.println ( "edad =" + edad () + "" + nombre

+ "Interrumpido de depósito"); regreso;

}

System.out.println ( "edad =" + edad () + "" + nombre

+ "Artículo depositado" + artículo);

}

}

}

Consumidor clase extiende azúcar implementa Ejecutable {private String nombre = null;

private int CNAP = 0; // milisegundos privado BoundedBuffer bb = null; Me privada = null rosca;

Consumidor pública (String nombre, int CNAP, BoundedBuffer bb) {this.name = nombre;

this.cNap = CNAP; this.bb = bb;

(Me = new Thread (este)) start ().;

}

pública timeToQuit void () {me.interrupt (); }

public void pauseTilDone () lanza InterruptedException

{Me.join (); } Public void run () {

si (Thread.currentThread () = yo!) return; doble elemento;

int napping; while (true) {

si (Thread.interrupted ()) {System.out.println ( "edad =" + edad () + "" + nombre

+ "Interrumpido"); regreso;

}

napping = 1 + (int) aleatorio (CNAP); System.out.println ( "edad =" + edad () + "" + nombre

+ "Siesta para" + "ms" + napping); try {Thread.sleep (napping); }

catch (InterruptedException e) { System.out.println("age=" + age() + ", " + name

+ " interrupted from sleep"); return;

}

System.out.println("age=" + age() + ", " + name

+ " wants to consume"); try { item = bb.fetch(); }

catch (InterruptedException e) { System.out.println("age=" + age() + ", " + name

+ " interrupted from fetch"); return;

}

System.out.println("age=" + age() + ", " + name

+ " fetched item " + item);

}

}

}

class ProducersConsumers extends Sugar { public static void main(String[] args) {

int numSlots = 10; int numProducers = 1; int numConsumers = 1;

int pNap = 2;// defaults

int cNap = 2;// in

int runTime = 60;// seconds

// following set true in srbb.java runs

// so as not to try to join with a

// suspended thread and thus deadlock boolean doJoin = true;

try {

numSlots = Integer.parseInt(args[0]);

numProducers = Integer.parseInt(args[1]); numConsumers = Integer.parseInt(args[2]); pNap = Integer.parseInt(args[3]);

cNap = Integer.parseInt(args[4]); runTime = Integer.parseInt(args[5]); doJoin = args[6].equals("yes");

} catch (Exception e) { /\* use defaults \*/ } System.out.println("ProducersConsumers:\n numSlots="

+ numSlots + ", numProducers=" + numProducers

+ ", numConsumers=" + numConsumers + ", pNap="

+ pNap + ", cNap=" + cNap + ", runTime=" + runTime);

// create the bounded buffer

BoundedBuffer bb = new BoundedBuffer(numSlots);

// start the Producers and Consumers

// (they have self-starting threads) Producer[] p = new Producer[numProducers]; Consumer[] c = new Consumer[numConsumers];

new PseudoTimeSlicing(); // for Solaris, not Windows 95/NT for (int i = 0; i < numProducers; i++)

p[i] = new Producer("PRODUCER"+i, pNap\*1000, bb); for (int i = 0; i < numConsumers; i++)

c[i] = new Consumer("Consumer"+i, cNap\*1000, bb); System.out.println("All threads started");

// let them run for a while try {

Thread.sleep(runTime\*1000); System.out.println("age=" + age()

+ ", time to terminate the threads and exit"); for (int i = 0; i < numProducers; i++)

p[i].timeToQuit();

for (int i = 0; i < numConsumers; i++) c[i].timeToQuit();

Thread.sleep(1000); if (doJoin) {

for (int i = 0; i < numProducers; i++) p[i].pauseTilDone();

for (int i = 0; i < numConsumers; i++) c[i].pauseTilDone();

} else

System.out.println(" skipping pauseTilDone()");

} catch (InterruptedException e) { /\* ignored \*/ } System.out.println("age=" + age()

+ ", all threads are done"); System.exit(0);

}

}

# Sample run of bwbb.java

% javac bwbb.java bbdr.java

ProducersConsumers% java 10 1 1 2 2 5 ProducersConsumers:

numSlots = 10, numProducers = 1, numConsumers = 1, PNAP = 2, CNAP = 2, el tiempo de ejecución = 5 versión Java = 1.3.0

proveedor de Java = IBM Corporation Nombre del sistema operativo Linux =

arco OS = i586

versión del sistema operativo = # 1 Lun Sep 27 de 10:25:54 EDT 1999.2.2.12-20

No es necesario PseudoTimeSlicing

edad = 47, PRODUCER0 siesta para 1349 edad ms = 66, Consumer0 siesta para 201 ms

Todas las discusiones iniciadas

edad = 286, Consumer0 quiere consumir

edad = 1,406, PRODUCER0 produjo artículo 0.11775977233610768 edad = 1,422, PRODUCER0 artículo depositado 0.11775977233610768 edad = 1,423, PRODUCER0 siesta para 1203 ms

edad = 1424, Consumer0 elemento descabellada ,11775977233610768 edad = 1426, Consumer0 siesta durante 39 ms

edad = 1475, Consumer0 quiere consumir

edad = 2,636, PRODUCER0 produjo artículo ,717652488961075 edad = 2,637, PRODUCER0 artículo depositado ,717652488961075 edad = 2,638, PRODUCER0 siesta para 143 ms

edad = 2640, Consumer0 elemento descabellada ,717652488961075 edad = 2,641, Consumer0 siestas para los 1020 ms

edad = 2,795, PRODUCER0 produjo artículo 0.29972388090543556 edad = 2,797, PRODUCER0 artículo depositado 0.29972388090543556 edad = 2,798, PRODUCER0 siesta para 1898 ms

edad = 3668, Consumer0 quiere consumir

edad = 3668, Consumer0 elemento descabellada ,29972388090543556 edad = 3669, Consumer0 siestas para los 1046 ms

edad = 4,708, PRODUCER0 produjo artículo ,5794441336470957 edad = 4,709, PRODUCER0 artículo depositado ,5794441336470957 edad = 4,710, PRODUCER0 siesta para 955 ms

edad = 4725, Consumer0 quiere consumir

edad = 4726, Consumer0 elemento descabellada ,5794441336470957 edad = 4727, Consumer0 siestas para los 528 ms

tiempo de la edad = 5078, para terminar los hilos y edad de salida = 5081, PRODUCER0 interrumpió el sueño de la edad = 5082, Consumer0 interrumpió el sueño de la edad = 6088, todas las discusiones se llevan a cabo

# Sección 6. Las condiciones de carrera algunas definiciones

### Si dos hilos ejecutan n = n + 1 en un n variable compartida en aproximadamente el mismo tiempo, su carga y

almacenar instrucciones pueden intercalar de manera que un hilo sobrescribe la actualización de la otra.

Esta actualización perdida conduce a un resultado erróneo y es un ejemplo de una condición de carrera. son condiciones de carrera posible cuando dos o más hilos comparten datos, que están leyendo y escribiendo los datos compartidos al mismo tiempo, y el resultado final del cálculo depende de cuál hace qué y cuándo.

Ejemplos de condiciones de carrera

[Ejemplo race.java](#_bookmark20) en la página 24 muestra una actualización perdida en whichsum = fn (suma, m) desempeña el papel de n = n + 1. [Muestra correr de race.java](#_bookmark21) en la página 25 ilustra los resultados.

En [Ejemplo rac2.java](#_bookmark22) en la página 25, existe una condición de carrera entre un hilo de ATM y una rosca Auditor en un banco. [Muestra correr de rac2.java](#_bookmark23) en la página 26 muestra los resultados.

[Ejemplo srbb.java](#_bookmark24) on page 27 shows we should not synchronize threads withsuspend() and resume() because a race condition is possible. If we try to replace busy waiting with blocking in the bounded-buffer producer and consumer by having a thread suspend itself until resumed by the other thread, we run the risk of both the producer thread and the consumer thread becoming suspended, each waiting for the other to resume it.

[Driver bbdr.java](#_bookmark17) on page 20 creates the producer and consumer threads. [Sample run of](#_bookmark25) [srbb.java](#_bookmark25) on page 28 demonstrates a sample run.

The next several panels display the code described in this section. To view the code, click Next; or you can go directly to the next section, [Synchronized blocks](#_bookmark26) on page 30 , and return to the code samples at another time.

Example race.java

class Racer implements Runnable {

// these two fields are shared by both threads since

// sólo hay un objeto creado a partir de esta clase private int M = 0;

suma a largo volátil privada = 0; // Nota `volátil' público Racer (int M) {this.M = M; }

fn privada de largo (long j, int k) {long Total = j;

for (int i = 1; i <= k; i ++) Total + = yo; rendimiento total;

}

public void run () {

for (int m = 1; m <= M; m ++) suma = fn (suma, m); System.out.println ( "suma =" + suma);

}

}

Carreras de la clase {

void main (String [] args) {public static Racer racerObject = new Racer (2000);

Hilo racerThread1 = new Thread (racerObject); Hilo racerThread2 = new Thread (racerObject);

nuevo PseudoTimeSlicing (); // para Solaris, no Windows 95 / NTracerThread1.start (); racerThread2.start ();

tratar { racerThread1.join (); racerThread2.join (); } captura (InterruptedException e) {/ \* ignorado \* /}

}

}

# muestra de ejecución de race.java

race.java% javac

% Racing java

java version = 1.3.0

Java vendor=IBM Corporation OS name=Linux

OS arch=i586

OS version=#1 Mon Sep 27 10:25:54 EDT 1999.2.2.12-20

No PseudoTimeSlicing needed sum = 1335334000

sum = 1394734020

# Example rac2.java

class SavingsAccount { public volatile int balance = 0; } class ATM extends Sugar implements Runnable {

private int numAccounts = 0;

private SavingsAccount[] savingsAccount = null;

public ATM(int numAccounts, SavingsAccount[] savingsAccount) { this.numAccounts = numAccounts;

this.savingsAccount = savingsAccount;

}

public void run() {

int fromAccount, toAccount, amount; while (true) {

if (Thread.interrupted()) { System.out.println("age()=" + age()

+ ", ATM was interrupted"); return;

}

fromAccount = (int) random(numAccounts); toAccount = (int) random(numAccounts); amount = 1 +

(Int) aleatorio (SavingsAccount [fromAccount] .balance); SavingsAccount [fromAccount] .balance - = cantidad; SavingsAccount [toAccount] .balance + = cantidad;

}

}

}

Auditor clase extiende azúcar implementa Runnable {private int NUMACCOUNTS = 0;

SavingsAccount privado [] SavingsAccount = null;

Auditor pública (int NUMACCOUNTS, SavingsAccount [] SavingsAccount) {this.numAccounts = NUMACCOUNTS;

this.savingsAccount = SavingsAccount;

}

public void run () {int totales;

while (true) {

try {Thread.sleep (1000); } Catch (InterruptedException e) {

System.out.println ( "edad () =" + edad ()

+ "Auditor interrumpido por el sueño"); regreso;

}

Total = 0;

for (int i = 0; i <NUMACCOUNTS; i ++) Total + = SavingsAccount [i] .balance;

System.out.println ( "edad () =" + edad ()

+ "Total es $" + total); si (Thread.interrupted ()) {

System.out.println ( "edad () =" + edad ()

+ "Auditor fue interrumpido"); regreso;

}

}

}

}

Banco clase extiende azúcar {

public void (String [] args) {int NUMACCOUNTS = 100;

int initialValue = 1,000; // dólares SavingsAccount [] SavingsAccount = null; tratar {

NUMACCOUNTS = Integer.parseInt (args [0]); initialValue = Integer.parseInt (args [1]);

} Catch (Exception e) {/ \* Usar valores predeterminados \* /} SavingsAccount = new SavingsAccount [NUMACCOUNTS]; for (int i = 0; i <NUMACCOUNTS; i ++) {

SavingsAccount [i] = new SavingsAccount (); SavingsAccount [i] = .balance initialValue;

}

System.out.println ( "Banco abierto con" + NUMACCOUNTS

+ "cuentas, cada una comenzando con $" + initialValue); nuevo PseudoTimeSlicing (); // para Solaris, no Windows 95 / NT Tema atm = new Thread (

nuevo ATM (NUMACCOUNTS, SavingsAccount)); Tema auditor = new Thread (

nuevo auditor (NUMACCOUNTS, SavingsAccount)); atm.start (); auditor.start ();

tratar {

Thread.sleep(10000); atm.interrupt();atm.join(); Thread.sleep(3000); auditor.interrupt(); auditor.join();

} catch (InterruptedException e) { /\* ignored \*/ } System.exit(0);

}

}

# Sample run of rac2.java

% javac rac2.java

% java Bank

Bank open with 100 accounts, each starting with $1000

Java version=1.3.0

Java vendor=IBM Corporation OS name=Linux

OS arch=i586

OS version=#1 Mon Sep 27 10:25:54 EDT 1999.2.2.12-20

No PseudoTimeSlicing needed age()=1171, total is $100000 age()=2191, total is $100000 age()=3201, total is $100000 age()=4211, total is $100000 age()=5221, total is $100000 age()=6231, total is $100000 age()=7241, total is $100000 age()=8251, total is $100000 age()=9261, total is $99999 age()=10171, ATM was interrupted age()=10271, total is $100000 age()=11283, total is $100000 age()=12291, total is $100000

age()=13182, Auditor interrupted from sleep

% Banco java 500000

Banco abierto con 500000 cuentas, cada una comenzando con $ 1000 = 1.3.0 versión de Java

proveedor de Java = IBM Corporation Nombre del sistema operativo Linux =

arco OS = i586

versión del sistema operativo = # 1 Lun Sep 27 de 10:25:54 EDT 1999.2.2.12-20

Sin PseudoTimeSlicing edad necesaria () = 10763, total es de $ 499996.968 mil años () = 12418, total es de $ 499992,296 mil años () = 14018, total es de $ 499990,606 mil años () = 15669, total es de $ 499,986,034 edad () = 17326, total es de $ 500 012 696 años ( ) = 19.020, total es de $ 499974,189 mil edad () = 19.213, ATM se interrumpió edad () = 20.968, total es de $ 500,000,000 edad () = 23.113, total es de $ 500,000,000 edad () = 23.114, se interrumpió Auditor

# Ejemplo srbb.java

BufferItem clase {

pública valor doble volátil = 0;

public boolean volátil ocupado = false; hilo de rosca volátil pública = null;

}

BoundedBuffer clase {

// diseñado para un solo hilo productor y

// un solo hilo consumidor int numSlots privadas = 0;

private BufferItem[] buffer = null; private int putIn = 0, takeOut = 0; public BoundedBuffer(int numSlots) {

if (numSlots <= 0)

throw new IllegalArgumentException("numSlots <= 0"); this.numSlots = numSlots;

buffer = new BufferItem[numSlots]; for (int i = 0; i < numSlots; i++)

buffer[i] = new BufferItem();

}

public void deposit(double value)

throws InterruptedException { if (buffer[putIn].occupied) {

Thread producer = Thread.currentThread(); buffer[putIn].thread = producer;

// context switch possible here

producer.suspend(); buffer[putIn].thread = null;

}

buffer[putIn].value = value; buffer[putIn].occupied = true;

Thread consumer = buffer[putIn].thread; putIn = (putIn + 1) % numSlots;

if (consumer != null) consumer.resume();

}

public double fetch()

throws InterruptedException { double value;

if (!buffer[takeOut].occupied) {

Thread consumer = Thread.currentThread(); buffer[takeOut].thread = consumer;

// context switch possible here

consumer.suspend(); buffer[takeOut].thread = null;

}

value = buffer[takeOut].value; buffer[takeOut].occupied = false;

Thread producer = buffer[takeOut].thread; takeOut = (takeOut + 1) % numSlots;

if (producer != null) producer.resume(); return value;

}

}

# Sample run of srbb.java

% javac srbb.java bbdr.java

% java ProducersConsumers 10 1 1 2 2 5 no ProducersConsumers:

numSlots=10, numProducers=1, numConsumers=1, pNap=2, cNap=2, runTime=5 Java version=1.3.0

Java vendor=IBM Corporation OS name=Linux

OS arch=i586

OS version=#1 Mon Sep 27 10:25:54 EDT 1999.2.2.12-20

No PseudoTimeSlicing needed

age=48, PRODUCER0 napping for 1549 ms age=67, Consumer0 napping for 1861 ms All threads started

edad = 1,609, PRODUCER0 produjo artículo 0.37648945305426074 edad = 1,625, PRODUCER0 artículo depositado 0.37648945305426074 edad = 1,626, PRODUCER0 siesta para 974 ms

edad = 1949, Consumer0 quiere consumir

edad = 1950, Consumer0 elemento descabellada ,37648945305426074 edad = 1952, Consumer0 siestas para los 381 ms

edad = 2,347, Consumer0 quiere consumir

edad = 2,617, PRODUCER0 produjo artículo ,7493684193792439 edad = 2,618, Consumer0 artículo descabellada ,7493684193792439 edad = 2,619, Consumer0 siesta para 377 ms

edad = 2,638, PRODUCER0 deposita artículo ,7493684193792439 edad = 2,640, PRODUCER0 siesta para 1014 ms

edad = 3009, Consumer0 quiere consumir

edad = 3,667, PRODUCER0 produjo artículo ,8117997960074402 edad = 3,668, Consumer0 artículo descabellada ,8117997960074402 edad = 3,669, Consumer0 napping por 365 ms

edad = 3,686, PRODUCER0 deposita artículo ,8117997960074402 edad = 3,688, PRODUCER0 siesta para 484 ms

edad = 4048, Consumer0 quiere consumir

edad = 4,187, PRODUCER0 produjo artículo ,8961043263431506 edad = 4,188, Consumer0 artículo descabellada ,8961043263431506 edad = 4,189, Consumer0 siesta para 675 ms

edad = 4,207, PRODUCER0 deposita artículo ,8961043263431506 edad = 4,208, PRODUCER0 siesta para 504 ms

edad = 4,727, PRODUCER0 produjo artículo 0.34322613800540913 edad = 4,728, PRODUCER0 artículo depositado 0.34322613800540913 edad = 4,729, PRODUCER0 siesta para 1195 ms

edad = 4877, Consumer0 quiere consumir

edad = 4,877, Consumer0 artículo descabellada ,34322613800540913 edad = 4,878, Consumer0 napping durante 19 ms

edad = 4906, Consumer0 quiere consumir

edad = 5077, el tiempo para terminar el roscas y salida edad = 5098, PRODUCER0 interrumpido por el sueño

saltarse pauseTilDone () edad = 6087, todas las discusiones se hacen

# Sección 7. sincronizada bloques bloqueos de objetos

Cada objeto Java tiene una cerradura. Un bloque sincronizado utiliza bloqueo de un objeto a actuar como un binario

### semaphore with the initial value "1", solving the mutual exclusion critical section problem:

Object obj = new Object();

...

synchronized (obj) {// in a method

... // any code, e.g., critical section

}

### The construct:

... synchronized method(...) {

...// body of method

}

### is an abbreviation for:

... method(...) { synchronized (this) {

...// body of method

}

}

### That is, the entire body of the instance method is a synchronized block on the object (keyword this) the method is in.

The JLS does not guarantee that the thread that has waited the longest to lock an object will be the next to obtain the lock when the object is unlocked.

Examples of synchronized blocks

[Example parp.java](#_bookmark27) on page 31 offers multithreaded prime number generation with a fixed number of threads (using [Class Prime.java](#_bookmark6) on page 7 ). [Sample run of parp.java](#_bookmark28) on page 31 shows the results.

In [Example norc.java](#_bookmark29) on page 32 , only one thread at a time is allowed to execute

**sum=fn(sum,m)**. [Sample run of norc.java](#_bookmark30) on page 32 demonstrates the sample run.

The next several panels contain an exercise and display the code described in this section. To view the exercise and code, click Next; or you can go directly to the next section, [Monitors](#_bookmark31) on page 33 , and return to the code samples at another time.

Try this exercise

Use a synchronized block to eliminate the race condition in [Example rac2.java](#_bookmark22) on page 25 .

Example parp.java

class ParallelPrimes implements Runnable {

private static int n1, n2, nChecked, nThreads, next; private static boolean[] taken, isPrime;

private Object mutex = this; // or = new Object(); public void run() {

int mine = 0; while (true) {

synchronized (mutex) {

while (next < nChecked && taken[next]) next++; mine = next;

si (el mío> = nChecked) return; tomada [mina] = true;

}

si (Prime.prime (n1 + mina)) esPrimo [mina] = true;

}

}

void Main (args String []) public static {try {

n1 = Integer.parseInt (args [0]); n2 = Integer.parseInt (args [1]);

nThreads = Integer.parseInt (args [2]);

} Catch (NumberFormatException e) {System.out.println ( "formato incorrecto"); System.exit (1);

} Catch (ArrayIndexOutOfBoundsException e) {System.out.println ( "no los argumentos de línea de comandos suficiente"); System.exit (1);

}

System.out.println ( "primos impresión de" + n1 + "a"

+ n2 + "utilizando" + nThreads + "hilos"); nChecked = n2 - n1 + 1;

si (nChecked <1 || nThreads> nChecked) {System.out.println ( "argumentos de línea de comando incorrecto"); System.exit (1);

}

taken = new boolean[nChecked]; isPrime = new boolean[nChecked]; for (int i = 0; i < nChecked; i++)

taken[i] = isPrime[i] = false; next = 0;

Thread[] t = new Thread[nThreads];

// All threads execute inside the SAME object and thus

// SHARE all data.

Runnable a = new ParallelPrimes();

for (int i = 0; i < nThreads; i++) t[i] = new Thread(a); new PseudoTimeSlicing(); // for Solaris, not Windows 95/NT for (int i = 0; i < nThreads; i++) t[i].start();

try {

for (int i = 0; i < nThreads; i++) t[i].join();

} catch (InterruptedException e) { /\* ignored \*/ } for (int i = 0; i < nChecked; i++)

if (isPrime[i])

System.out.println((n1 + i)+ " is prime");

}

}

# Sample run of parp.java

% javac parp.java

% java ParallelPrimes 1000000 1000060 5

printing primes from 1000000 to 1000060 using 5 threads Java version=1.3.0

Java vendor=IBM Corporation OS name=Linux

arco OS = i586

versión del sistema operativo = # 1 Lun Sep 27 de 10:25:54 EDT 1999.2.2.12-20

No se necesita PseudoTimeSlicing 1000003 es primo

1000033 es primo

1000037 es primo

1000039 es primo

# Ejemplo norc.java

Racer clase implementa Ejecutable {

// estos dos campos son compartidos por ambos hilos desde

// sólo hay un objeto creado a partir de esta clase private int M = 0;

suma larga privada = 0; // `volátil' ya no necesario público Racer (int M) {this.M = M; }

fn privada de largo (long j, int k) {long Total = j;

for (int i = 1; i <= k; i ++) Total + = yo; rendimiento total;

}

public void run () {

for (int m = 1; m <= M; m ++)

sincronizado (este) {// protocolo de entrada suma = fn (suma, m); // crítico sección

}// salida protocolo

System.out.println ( "suma =" + suma);

}

}

# muestra de ejecución de norc.java

norc.java% javac

% Racing java

java version = 1.3.0

proveedor de Java = IBM Corporation Nombre del sistema operativo Linux =

arco OS = i586

versión del sistema operativo = # 1 Lun Sep 27 de 10:25:54 EDT 1999.2.2.12-20

No PseudoTimeSlicing needed sum = 1335334000

sum = 2670668000

# Section 8. Monitors

Monitor structure and properties

### Every Java object possesses a lock and these methods: wait(), notify(), and

**notifyAll()**.

A thread invoking a synchronized method must acquire the lock of the object containing the method before executing the method's code. The thread blocks if the object is already locked.

A monitor has the following structure or pattern:

class Monitor extends ... {

private ...// data fields (state variables)

Monitor(...) {...}// constructor public synchronized *type* method1(...)

throws InterruptedException {

...

notifyAll(); // if any wait conditions altered while (!condition1) wait();

...

notifyAll(); // if any wait conditions altered

}

public synchronized type method2(...) throws InterruptedException {

...

notifyAll(); // if any wait conditions altered while (!condition2) wait();

...

notifyAll(); // if any wait conditions altered

}

...

}

### A Java thread is interrupted when its interrupt() method is called by another thread. This call sets a flag in the interrupted thread that the latter can check periodically, allowing one thread to tell another thread to stop itself or return allocated resources if it is not in the middle of some critical operation. (A thread should check its interrupt flag before or after such operations and take appropriate action when interrupted.)

Note the following important points:

* The thread blocked the longest on a monitor synchronized method call is not guaranteed to be the next thread to acquire the monitor lock when the monitor lock is released.
* The thread blocked the longest in a monitor wait() call is not guaranteed to be the one removed from the wait set when a notify() is done by some other thread in the monitor.
* The signaling discipline is signal-and-continue so barging is possible -- a thread waiting for the monitor lock to execute a monitor synchronized method might get the lock before a signaled thread re-acquires it, even if the notify() occurred earlier than the monitor method call. Thus:

while (!condition) ... wait() ... notifyAll()

### is safer than:

**if (!condition) ... wait() ... notify()**

### Each monitor object has a single nameless anonymous condition variable. We cannot signal with notify() one of several threads waiting on a specific condition. It is safer to use notifyAll() to awaken all waiting threads so they can recheck their waiting conditions.

* A notifyAll() needs to be done by a thread before a wait() if any state variables were altered by the thread after entering the monitor, which might affect other

thread-waiting conditions. This also applies before leaving the monitor (returning from the method).

* The data fields in a monitor need not be declared volatile because all writes to shared variables by a thread are completed before obtaining and before releasing the monitor lock.
* If a thread that is blocked inside a call to sleep(ms), join(), or wait() is interrupted, then these methods clear the thread's interrupt flag and throw an InterruptedException instead of returning normally. Note that no exception is thrown if a thread is interrupted while blocked waiting to acquire a monitor's lock to execute a synchronized method.
* In contrast, InterruptedException is thrown by wait() if a thread that has been notified is interrupted while blocked and waiting to reacquire the monitor lock. If an InterruptedException occurs while a thread is in wait(), the thread must reacquire the monitor lock before executing the code in the catch block.
* Ignoring InterruptedException, as in:

while (!condition) try {wait(); } catch (InterruptedException e) { }

### is undesirable. The enclosing method should throw the exception back to the caller.

* The following code:

if (!condition) try {wait(); } catch (InterruptedException e) { }

### is incorrect because a thread interrupted out of its wait() then re-enters the monitor without being notified.

* When a call to wait(milliseconds) returns, the program cannot tell for sure if the wait was notified or if the wait timed out after the number of milliseconds elapsed.
* In some situations, we can use notify() instead of notifyAll() and if ... wait() instead of while ... wait(). However, it is extremely tricky and not recommended because of a race condition between interrupt() and notify().

### Suppose several threads are blocked inside wait() and then one of them is notified and then interrupted before it reacquires the monitor lock. The notify() gets "lost" in that one of the other waiting threads should now proceed. We need to catch the exception when a thread is interrupted out of wait() and regenerate the notify().

* It is usually wrong to Thread.sleep(ms) while inside a monitor object holding the lock (during a synchronized method invocation). Other threads wanting to enter the monitor will block to acquire the monitor object's lock and they cannot be interrupted from this state.

Es mejor establecer una bandera y dejar el monitor; otros hilos pueden esperar () para la bandera para cambiar. No se sostiene el hilo de bloqueo de la pantalla durante más tiempo que para fijar o confirmar esta bandera.

Como muestran los siguientes ejemplos, los monitores se pueden utilizar para sincronizar hilos que solicitar recursos desde y recursos volver a un servidor. Esto se llama una relación de cliente / servidor.

Los clientes interactúan con el servidor, pero no entre sí. El monitor de servidor es un objeto pasivo en el sentido de que ningún hilo independiente realiza dentro de él; el código en el monitor sólo se ejecuta cuando un método de supervisión de es invocado por un hilo cliente.

Los monitores pueden ser difíciles de usar si los hilos tienen una relación que no sea un cliente / servidor de una.

El material de base en los monitores

Semaphores are like gotos and pointers -- they work okay but are error prone and lack structure and "discipline."

For example, a disastrous typo such as:

*V(S); criticalSection(); V(S)*

can lead to deadlock:

*P(S); criticalSection(); P(S)*.

Nested critical sections can also lead to deadlock:

*P1: P(Q); P(S); ... V(S); V(Q);*

*P2: P(S); P(Q); ... V(Q); V(S);*

A monitor is an object with some built-in mutual exclusion and thread-synchronization capabilities. Monitors are an integral part of the programming language so the compiler can generate the correct code to implement the monitor. Only one thread can be active at a time in the monitor ("active" meaning executing a method of the monitor).

Monitors also have condition variables on which a thread can wait if conditions are not right for it to continue executing in the monitor. Some other thread can then get in the monitor and perhaps change the state of the monitor. If conditions are now right, that thread can signal a

waiting thread, moving the latter to the ready queue to get back into the monitor when it becomes free.

Monitors can use either a signal-and-exit or signal-and-continue signaling discipline. In signal-and-exit , a signaling thread must leave the monitor immediately, at which point it is guaranteed that the signaled thread is the next one in the monitor.

### En relación señal-y continuar, el hilo señalado no se garantiza que sea la siguiente en el monitor. De hecho, irrumpir puede tener lugar - un poco de hilo que se ha llamado un método de supervisión y se bloquea hasta que el monitor es libre puede entrar en el monitor antes de un hilo señalado.

Semáforos y monitores se pueden utilizar para resolver los problemas de los llamados "clásicos" de sincronización que se encuentran en muchos libros sistemas operativos: el barbero dormir, los cinco filósofos comedores, y los lectores de bases de datos y escritores.

### **El barbero de dormir.** Un barbero espera para cortar el pelo. Los clientes entran en la sala de espera y en un segundo plano si hay uno disponible. Si la sala de espera está llena, inténtelo de nuevo más tarde. De lo contrario, esperan hasta que su turno para un corte de pelo.

**Cinco filósofos comedor.** Five philosophers sit around a table and think until hungry. Between each is a fork (for a total of five forks). To eat, a hungry philosopher must have exclusive access to both the fork on his left and right. If both forks are not free, the philosopher waits.

The algorithm in [Example dpmo.java](#_bookmark32) on page 39 does not deadlock (it never happens that all philosophers are hungry, each holding one fork and waiting for the other), allows maximal parallelism (a philosopher never picks up and holds a fork while waiting for the other fork to become available when the fork he is holding could be used for eating by its neighbor), but also allows starvation (a philosopher's two neighbors can collaborate and alternate their eating so the one in the middle never can use the forks).

Si un filósofo puede sostener un tenedor a la espera de otro tenedor, estancamiento es posible, un caso extremo de no tener paralelismo máximo. Sin embargo, el hambre no es posible. Cada tenedor es representado por un semáforo y cada filósofo hambre hace una "P" en su tenedor a la izquierda y luego a su tenedor derecho.

Podemos solucionar el problema de interbloqueo y retener sin hambre, pero todavía no tienen paralelismo máximo. Todos los filósofos recogen izquierda luego a la derecha, excepto uno filósofo designado que recoge justo luego a la izquierda.

Philosopher starvation can also be prevented by introducing a new state: very hungry. A philosopher is put into this state if he is hungry, if one of his neighbors puts down his forks, and if he cannot eat because the other fork is in use. A new rule is added -- a hungry philosopher cannot eat if he has a very hungry neighbor. These changes prevent a collaboration of two philosophers trying to starve the philosopher between them.

**Readers and writers.** A database can be accessed concurrently by threads that only want to read, but a writer thread must have exclusive access with respect to other readers and writers.

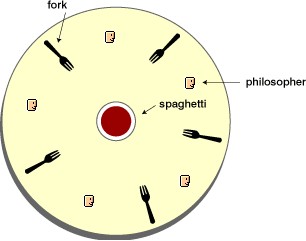
A solution might allow writers to starve if enough readers keep coming along to read the database so that the number of current readers is always above zero.

Writer starvation is prevented by requiring readers that come along to read the database to wait if there is a waiting writer even if other readers are currently reading the database. When the current readers finish, the waiting writer writes the database and then signals into the database a waiting reader. Each entering reader signals another waiting reader into the database.

Examples of monitors

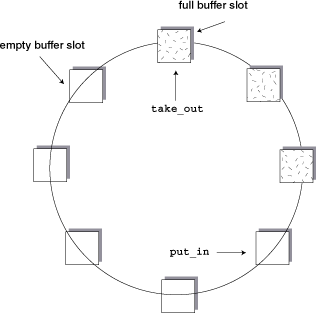
[Example dpmo.java](#_bookmark32) on page 39 illustrates the dining philosophers monitor. Five philosophers sit around a table and think until hungry. Between each pair of philosophers is one fork. A hungry philosopher must have exclusive simultaneous access to both its left and right forks in order to eat. If they are not both free, the philosopher waits. [Driver dpdr.java](#_bookmark33) on page 40 creates the philosopher threads. [Sample run of dpmo.java](#_bookmark34) on page 42 shows the sample run.

The figure below illustrates the dining philosophers monitor.



[Example bbmo.java](#_bookmark35) on page 42 shows a bounded buffer monitor for a producer and consumer. Multiple producer threads and multiple consumer threads are handled. A producer thread deposits items and blocks if the bounded buffer fills up. A consumer thread fetches items and blocks if the bounded buffer is empty. [Driver bbdr.java](#_bookmark17) on page 20 creates the producer and consumer threads. [Sample run of bbmo.java](#_bookmark36) on page 43 shows the sample run.

The figure below illustrates the bounded buffer monitor for a producer and consumer, handling multiple producer and consumer threads.

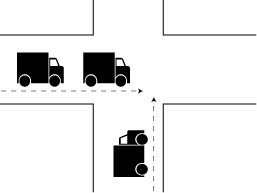


[Example inmo.java](#_bookmark37) on page 44 simulates cars crossing at an intersection of two one-way streets so that:

* Only one car can cross at a time
* A car can cross if there are no cars on the intersecting street waiting to cross
* Si dos vehículos se aproximan a la intersección más o menos al mismo tiempo, uno de ellos se cruzarán (sin bloqueo)
* Si hay coches en las calles que se cruzan esperando para cruzar, a continuación, los coches de las calles que se cruzan se turnan para evitar el hambre

[Conductor indr.java](#_bookmark38) en la página 45 crea los hilos del coche.[Muestra correr de inmo.java](#_bookmark39) en la página 47 muestra los resultados.

La siguiente figura ilustra coches que cruzan en una intersección de dos calles de un solo sentido.



Los próximos paneles contienen ejercicios y mostrar el código descrito en esta sección. Para ver los ejercicios y el código, haga clic en Siguiente; o puede ir directamente a la siguiente sección,[Los semáforos](#_bookmark40) en la página 50, y volver a los ejemplos de código en otro momento.

Pruebe estos ejercicios

**Ejercicio 1:** Write a monitor for the database readers and writers problem. Multiple reader threads can read the database simultaneously, but writer threads must have exclusive access with respect to other reader and writer threads.

**Exercise 2:** Write a barrier monitor. Threads wait until all threads arrive at the barrier, then they are all released.

**Exercise 3:** When a notify() or notifyAll() is done inside a Java monitor, the next thread to get inside the monitor (acquire the lock) is arbitrary. Therefore, the cars in the intersection simulation going in the same direction do not necessarily go through the intersection in the order they arrived at it (it is not FCFS). Fix this.

Example dpmo.java

class DiningServer extends Sugar { private int numPhils = 0; private int[] state = null;

privado static final int PENSAMIENTO = 0, hambriento = 1, comer = 2; DiningServer pública (numPhils int) {

this.numPhils = numPhils; estado = new int [numPhils];

for (int i = 0; i <numPhils; i ++) estado [i] = pensar;

}

public void cenar (String nombre, int id, int napEat) lanza InterruptedException {

tratar {

takeForks (id); comer (nombre, napEat);

} finalmente {// Asegúrese de que volvamos la putForks (id); // horquillas si interrumpido

}

}

final privado int izquierda (int i)

{return (numPhils + i - 1);}% numPhils final privado int derecha (int i)

{Return (i + 1)% numPhils; } Prueba de private void (int k) {

si (estado [izquierda (k)]! = COMER estado && [k] == estado HAMBRE && [derecha (k)]! = comer)

estado [k] = comer;

}

comer private void (String nombre, int napEat) lanza InterruptedException {

int napping;

napping = 1 + (int) aleatorio (napEat); System.out.println ( "edad =" + edad () + "" + nombre

+ " is eating for " + napping + " ms"); Thread.sleep(napping);

}

private synchronized void takeForks(int i) throws InterruptedException {

state[i] = HUNGRY;test(i);

while (state[i] != EATING) wait();

}

private synchronized void putForks(int i) { if (state[i] != EATING) return;

state[i] = THINKING; test(left(i)); test(right(i)); notifyAll();

}

}

# Driver dpdr.java

class Philosopher extends Sugar implements Runnable { private String name = null;

private int id = 0;

private int napThink = 0; // both are in private int napEat = 0;// milliseconds private DiningServer ds = null;

private Thread me = null;

public Philosopher(int id, int napThink, int napEat, DiningServer ds) {

this.name = "Philosopher " + id; this.id = id;

this.napThink = napThink; this.napEat = napEat; this.ds = ds;

(me = new Thread(this)).start();

}

public void timeToQuit() { me.interrupt(); }

public void pauseTilDone() throws InterruptedException

{ me.join(); }

private void think() throws InterruptedException { int napping;

napping = 1 + (int) random(napThink); System.out.println("age=" + age() + ", " + name

+ " is thinking for " + napping + " ms"); Thread.sleep(napping);

}

public void run() {

if (Thread.currentThread() != me) return; while (true) {

if (Thread.interrupted()) { System.out.println("age=" + age() + ", " + name

+ " interrupted"); return;

}

try {

think();

} catch (InterruptedException e) { System.out.println("age=" + age() + ", " + name

+ " interrupted out of think"); return;

}

System.out.println("age=" + age() + ", " + name

+ " wants to dine"); try {

ds.dine(name, id, napEat);

} catch (InterruptedException e) { System.out.println("age=" + age() + ", " + name

+ " interrupted out of dine"); return;

}

}

}

}

class DiningPhilosophers extends Sugar { public static void main(String[] args) {

int numPhilosophers = 5;

int runTime = 60;// seconds int napThink = 8, napEat = 2; try {

numPhilosophers = Integer.parseInt(args[0]); runTime = Integer.parseInt(args[1]); napThink = Integer.parseInt(args[2]);

napEat = Integer.parseInt(args[3]);

} catch (Exception e) { /\* use defaults \*/ } System.out.println("DiningPhilosophers: numPhilosophers="

+ numPhilosophers + ", runTime=" + runTime

+ ", napThink=" + napThink + ", napEat=" + napEat);

// create the DiningServer object

DiningServer ds = new DiningServer(numPhilosophers);

// create the Philosophers

// (they have self-starting threads)

Philosopher[] p = new Philosopher[numPhilosophers]; for (int i = 0; i < numPhilosophers; i++) p[i] =

new Philosopher(i, napThink\*1000, napEat\*1000, ds); System.out.println("All Philosopher threads started");

// let the Philosophers run for a while try {

Thread.sleep(runTime\*1000); System.out.println("age=" + age()

+ ", time to terminate the Philosophers and exit"); for (int i = 0; i < numPhilosophers; i++)

p[i].timeToQuit(); Thread.sleep(1000);

for (int i = 0; i < numPhilosophers; i++) p[i].pauseTilDone();

} catch (InterruptedException e) { /\* ignored \*/ } System.out.println("age=" + age()

+ ", all Philosophers are done"); System.exit(0);

}

}

# Sample run of dpmo.java

% javac dpmo.java dpdr.java

% java DiningPhilosophers 5 6 4 1

DiningPhilosophers: numPhilosophers=5, runTime=6, napThink=4, napEat=1 age=37, Philosopher 0 is thinking for 2952 ms

age=56, Philosopher 1 is thinking for 3012 ms age=59, Philosopher 2 is thinking for 508 ms age=61, Philosopher 3 is thinking for 456 ms All Philosopher threads started

age=63, Philosopher 4 is thinking for 120 ms age=186, Philosopher 4 wants to dine age=187, Philosopher 4 is eating for 205 ms age=406, Philosopher 4 is thinking for 839 ms age=525, Philosopher 3 wants to dine age=526, Philosopher 3 is eating for 107 ms age=575, Philosopher 2 wants to dine age=646, Philosopher 2 is eating for 111 ms

age=646, Philosopher 3 is thinking for 1746 ms age=776, Philosopher 2 is thinking for 3522 ms age=1258, Philosopher 4 wants to dine age=1259, Philosopher 4 is eating for 861 ms age=2136, Philosopher 4 is thinking for 3759 ms age=2406, Philosopher 3 wants to dine age=2406, Philosopher 3 is eating for 878 ms age=3008, Philosopher 0 wants to dine age=3009, Philosopher 0 is eating for 881 ms age=3086, Philosopher 1 wants to dine age=3296, Philosopher 3 is thinking for 546 ms age=3855, Philosopher 3 wants to dine age=3856, Philosopher 3 is eating for 648 ms age=3908, Philosopher 0 is thinking for 2945 ms age=3909, Philosopher 1 is eating for 102 ms age=4026, Philosopher 1 is thinking for 3699 ms age=4316, Philosopher 2 wants to dine age=4516, Philosopher 2 is eating for 10 ms age=4517, Philosopher 3 is thinking for 414 ms age=4536, Philosopher 2 is thinking for 1593 ms age=4947, Philosopher 3 wants to dine age=4948, Philosopher 3 is eating for 898 ms age=5856, Philosopher 3 is thinking for 687 ms age=5905, Philosopher 4 wants to dine age=5906, Philosopher 4 is eating for 516 ms

age=6066, time to terminate the Philosophers and exit age=6069, Philosopher 1 interrupted out of think age=6070, Philosopher 2 interrupted out of think age=6071, Philosopher 4 interrupted out of dine age=6073, Philosopher 3 interrupted out of think age=6086, Philosopher 0 interrupted out of think age=7076, all Philosophers are done

# Example bbmo.java

class BoundedBuffer {

// designed for multiple producer threads and

// multiple consumer threads private int numSlots = 0; private double[] buffer = null;

private int putIn = 0, takeOut = 0; private int count = 0;

public BoundedBuffer(int numSlots) { if (numSlots <= 0)

throw new IllegalArgumentException("numSlots <= 0"); this.numSlots = numSlots;

buffer = new double[numSlots];

}

public synchronized void deposit(double value) throws InterruptedException {

while (count == numSlots) wait(); buffer[putIn] = value;

putIn = (putIn + 1) % numSlots;

count++;// wake up all those waiting due to notifyAll(); // signal-and-continue and barging

}

public synchronized double fetch() throws InterruptedException {

double value;

while (count == 0) wait(); value = buffer[takeOut];

takeOut = (takeOut + 1) % numSlots;

count--;// wake up all those waiting due to notifyAll(); // signal-and-continue and barging return value;

}

}

# Sample run of bbmo.java

% javac bbmo.java bbdr.java

% java ProducersConsumers 10 2 3 2 3 6 ProducersConsumers:

numSlots=10, numProducers=2, numConsumers=3, pNap=2, cNap=3, runTime=6 Java version=1.3.0

Java vendor=IBM Corporation OS name=Linux

OS arch=i586

OS version=#1 Mon Sep 27 10:25:54 EDT 1999.2.2.12-20

No PseudoTimeSlicing needed

age=42, PRODUCER0 napping for 1248 ms age=61, PRODUCER1 napping for 1702 ms age=65, Consumer0 napping for 226 ms age=67, Consumer1 napping for 94 ms All threads started

age=69, Consumer2 napping for 2504 ms age=171, Consumer1 wants to consume age=301, Consumer0 wants to consume

age=1303, PRODUCER0 produced item 0.5251246705209369 age=1319, PRODUCER0 deposited item 0.5251246705209369 age=1320, PRODUCER0 napping for 889 ms

age=1321, Consumer0 fetched item 0.5251246705209369 age=1323, Consumer0 napping for 1049 ms

age=1781, PRODUCER1 produced item 0.9915848089197059 age=1782, PRODUCER1 deposited item 0.9915848089197059 age=1783, PRODUCER1 napping for 297 ms

age=1784, Consumer1 fetched item 0.9915848089197059

age=1785, Consumer1 napping for 1253 ms

age=2093, PRODUCER1 produced item 0.4866393343763298 age=2094, PRODUCER1 deposited item 0.4866393343763298 age=2095, PRODUCER1 napping for 1018 ms

age=2221, PRODUCER0 produced item 0.40569282834803577 age=2222, PRODUCER0 deposited item 0.40569282834803577 age=2223, PRODUCER0 napping for 974 ms

age=2380, Consumer0 wants to consume

age=2381, Consumer0 fetched item 0.4866393343763298 age=2382, Consumer0 napping for 1531 ms

age=2580, Consumer2 wants to consume

age=2581, Consumer2 fetched item 0.40569282834803577 age=2582, Consumer2 napping for 2197 ms

age=3051, Consumer1 wants to consume

age=3123, PRODUCER1 produced item 0.9581218200181911 age=3124, PRODUCER1 deposited item 0.9581218200181911 age=3125, PRODUCER1 napping for 1125 ms

age=3126, Consumer1 fetched item 0.9581218200181911 age=3127, Consumer1 napping for 2387 ms

age=3211, PRODUCER0 produced item 0.9155450402123771 age=3212, PRODUCER0 deposited item 0.9155450402123771 age=3213, PRODUCER0 napping for 118 ms

age=3340, PRODUCER0 produced item 0.2431689653301975 age=3342, PRODUCER0 deposited item 0.2431689653301975 age=3343, PRODUCER0 napping for 235 ms

age=3590, PRODUCER0 produced item 0.06587542278093239 age=3592, PRODUCER0 deposited item 0.06587542278093239 age=3593, PRODUCER0 napping for 1014 ms

age=3931, Consumer0 wants to consume

age=3931, Consumer0 fetched item 0.9155450402123771 age=3932, Consumer0 napping for 1100 ms

age=4261, PRODUCER1 produced item 0.2895572679671733 age=4262, PRODUCER1 deposited item 0.2895572679671733 age=4263, PRODUCER1 napping for 1776 ms

age=4624, PRODUCER0 produced item 0.08728828492428509 age=4625, PRODUCER0 deposited item 0.08728828492428509 age=4626, PRODUCER0 napping for 446 ms

age=4790, Consumer2 wants to consume

age=4791, Consumer2 fetched item 0.2431689653301975 age=4792, Consumer2 napping for 2471 ms

age=5041, Consumer0 wants to consume

age=5041, Consumer0 fetched item 0.06587542278093239 age=5042, Consumer0 napping for 1068 ms

age=5081, PRODUCER0 produced item 0.8313015436389664 age=5082, PRODUCER0 deposited item 0.8313015436389664 age=5083, PRODUCER0 napping for 1416 ms

age=5521, Consumer1 wants to consume

age=5521, Consumer1 fetched item 0.2895572679671733 age=5522, Consumer1 napping for 2887 ms

age=6051, PRODUCER1 produced item 0.14766226608250066 age=6052, PRODUCER1 deposited item 0.14766226608250066 age=6053, PRODUCER1 napping for 534 ms

age=6071, time to terminate the threads and exit age=6074, PRODUCER1 interrupted from sleep age=6075, Consumer0 interrupted from sleep age=6077, Consumer2 interrupted from sleep age=6078, Consumer1 interrupted from sleep age=6091, PRODUCER0 interrupted from sleep age=7081, all threads are done

# Example inmo.java

class Intersection extends Sugar {

public static final int LEFT = 0, RIGHT = 1; private int[] waiting = {0, 0};

private int lastToCross = 0; private boolean crossing = false; public String how(int direction) {

if (direction == LEFT) return "left";

else if (direction == RIGHT) return "right"; else return "invalid";

}

public void crossIntersection

(String name, int direction, int cNap) throws InterruptedException {

wantToCross(direction); try {

cross(name, direction, cNap);

} finally {

// If we are interrupted while crossing, we must do this. doneCrossing();

}

}

private int other(int direction) {

if (direction == LEFT) return RIGHT;

else if (direction == RIGHT) return LEFT; else return -1;

}

private synchronized void wantToCross (int direction) throws InterruptedException {

waiting[direction]++; try {

while (crossing || (waiting[other(direction)] > 0 && lastToCross == direction))

wait(); lastToCross = direction; crossing = true;

} finally {// If we are interrupted while waiting[direction]--;// waiting to cross, do this.

}

}

private void cross(String name, int direction, int cNap) throws InterruptedException {

int napping;

napping = 1 + (int) random(cNap); System.out.println("age=" + age() + ", " + name

+ " CROSSING " + how(direction) + " for "

+ napping + " ms"); Thread.sleep(napping);

}

private synchronized void doneCrossing () { crossing = false;

notifyAll();

}

}

# Driver indr.java

class Car extends Sugar implements Runnable { private String name = null;

private int dNap = 0; // milliseconds private int cNap = 0; // milliseconds private int direction;

private Intersection in = null; private Thread me = null;

public Car(String name, int dNap, int cNap, int direction, Intersection in) {

this.name = name; this.dNap = dNap; this.cNap = cNap;

this.direction = direction; this.in = in;

(me = new Thread(this)).start();

}

public void timeToQuit() { me.interrupt(); }

public void pauseTilDone() throws InterruptedException

{ me.join(); } public void run() {

if (Thread.currentThread() != me) return; int napping;

while (true) {

if (Thread.interrupted()) { System.out.println("age=" + age() + ", " + name

+ " interrupted"); return;

}

napping = 1 + (int) random(dNap); System.out.println("age=" + age() + ", " + name

+ " napping for " + napping + " ms"); try { Thread.sleep(napping); }

catch (InterruptedException e) { System.out.println("age=" + age() + ", " + name

+ " interrupted from sleep"); return;

}

System.out.println("age=" + age() + ", " + name

+ " wants to cross " + in.how(direction));

try { in.crossIntersection(name, direction, cNap); } catch (InterruptedException e) {

System.out.println("age=" + age() + ", " + name

+ " interrupted from crossing"); return;

}

System.out.println("age=" + age() + ", " + name

+ " crossed " + in.how(direction));

}

}

}

class LeftRightCars extends Sugar {

public static void main(String[] args) { int numLefts = 3;

int numRights = 3;

int lNap = 2;// defaults

int rNap = 2;// are

int cNap = 2;// in

int runTime = 60;// seconds try {

numLefts = Integer.parseInt(args[0]); numRights = Integer.parseInt(args[1]); lNap = Integer.parseInt(args[2]);

rNap = Integer.parseInt(args[3]); cNap = Integer.parseInt(args[4]); runTime = Integer.parseInt(args[5]);

} catch (Exception e) { /\* use defaults \*/ } System.out.println("LeftsRights:\n numLefts=" + numLefts

+ ", numRights=" + numRights + ", lNap=" + lNap

+ ", rNap=" + rNap + ", cNap=" + cNap

+ ", runTime=" + runTime);

// create the intersection Intersection in = new Intersection();

// start the left crossing and right crossing

// cars (they have self-starting threads) Car[] c = new Car[numLefts + numRights];

for (int i = 0; i < numLefts + numRights; i++)

c[i] = new Car("Car"+i, (i<numLefts?lNap:rNap)\*1000, cNap\*1000, (i<numLefts?in.LEFT:in.RIGHT), in);

System.out.println("All threads started");

// let them run for a while try {

Thread.sleep(runTime\*1000); System.out.println("age=" + age()

+ ", time to terminate the threads and exit"); for (int i = 0; i < numLefts + numRights; i++)

c[i].timeToQuit(); Thread.sleep(1000);

for (int i = 0; i < numLefts + numRights; i++) c[i].pauseTilDone();

} catch (InterruptedException e) { /\* ignored \*/ } System.out.println("age=" + age()

+ ", all threads are done"); System.exit(0);

}

}

# Sample run of inmo.java

% javac inmo.java indr.java

% java LeftRightCars 3 3 2 2 2 5 LeftsRights:

numLefts=3, numRights=3, lNap=2, rNap=2, cNap=2, runTime=5 age=40, Car0 napping for 1040 ms

age=59, Car1 napping for 1187 ms age=62, Car2 napping for 1603 ms age=64, Car3 napping for 1932 ms age=66, Car4 napping for 986 ms All threads started

age=68, Car5 napping for 1426 ms age=1061, Car4 wants to cross right age=1063, Car4 CROSSING right for 1791 ms age=1088, Car0 wants to cross left age=1258, Car1 wants to cross left age=1509, Car5 wants to cross right age=1679, Car2 wants to cross left age=2011, Car3 wants to cross right age=2869, Car2 CROSSING left for 1180 ms age=2870, Car4 crossed right

age=2871, Car4 napping for 1441 ms age=4061, Car2 crossed left age=4062, Car2 napping for 580 ms

age=4063, Car3 CROSSING right for 974 ms age=4329, Car4 wants to cross right age=4648, Car2 wants to cross left age=5051, Car2 CROSSING left for 405 ms age=5052, Car3 crossed right

age=5052, Car3 napping for 1988 ms

age=5069, time to terminate the threads and exit age=5072, Car2 interrupted from crossing age=5073, Car4 interrupted from crossing age=5075, Car3 interrupted from sleep

age=5076, Car5 interrupted from crossing age=5089, Car0 interrupted from crossing age=5090, Car1 interrupted from crossing age=6079, all threads are done

% java LeftRightCars 5 1 1 5 1 5 LeftsRights:

numLefts=5, numRights=1, lNap=1, rNap=5, cNap=1, runTime=5 age=33, Car0 napping for 550 ms

age=53, Car1 napping for 760 ms age=55, Car2 napping for 123 ms age=57, Car3 napping for 59 ms age=59, Car4 napping for 874 ms All threads started

age=61, Car5 napping for 1033 ms age=122, Car3 wants to cross left age=123, Car3 CROSSING left for 220 ms age=191, Car2 wants to cross left age=352, Car2 CROSSING left for 499 ms age=353, Car3 crossed left

age=354, Car3 napping for 295 ms age=591, Car0 wants to cross left age=661, Car3 wants to cross left age=821, Car1 wants to cross left age=863, Car1 CROSSING left for 161 ms age=865, Car2 crossed left

age=867, Car2 napping for 134 ms age=944, Car4 wants to cross left age=1011, Car2 wants to cross left age=1042, Car0 CROSSING left for 867 ms age=1043, Car1 crossed left

age=1043, Car1 napping for 936 ms age=1111, Car5 wants to cross right age=1924, Car0 crossed left age=1925, Car0 napping for 129 ms

age=1926, Car5 CROSSING right for 682 ms age=1992, Car1 wants to cross left age=2062, Car0 wants to cross left age=2622, Car0 CROSSING left for 377 ms age=2623, Car5 crossed right

age=2624, Car5 napping for 276 ms age=2914, Car5 wants to cross right age=3012, Car0 crossed left age=3012, Car0 napping for 417 ms

age=3013, Car5 CROSSING right for 740 ms age=3441, Car0 wants to cross left age=3762, Car0 CROSSING left for 95 ms age=3763, Car5 crossed right

age=3764, Car5 napping for 2783 ms age=3872, Car0 crossed left age=3872, Car0 napping for 307 ms

age=3873, Car1 CROSSING left for 165 ms age=4053, Car1 crossed left

age=4054, Car1 napping for 24 ms age=4055, Car2 CROSSING left for 681 ms age=4091, Car1 wants to cross left age=4191, Car0 wants to cross left age=4752, Car0 CROSSING left for 566 ms age=4753, Car2 crossed left

age=4754, Car2 napping for 329 ms

age=5062, time to terminate the threads and exit age=5065, Car1 interrupted from crossing age=5067, Car2 interrupted from sleep

age=5068, Car4 interrupted from crossing age=5069, Car3 interrupted from crossing age=5070, Car5 interrupted from sleep

age=5082, Car0 interrupted from crossing age=6072, all threads are done

# Section 9. Semaphores User-written classes

### The following user-written semaphore classes are Java monitors:

* Abstract [Class Semaphore.java](#_bookmark41) on page 53
* Counting semaphore [Class CountingSemaphore.java](#_bookmark42) on page 54
* Binary semaphore [Class BinarySemaphore.java](#_bookmark43) on page 54
* Syntactic sugar [Class SugarSM.java](#_bookmark44) on page 54 soP(S) can be used instead of S.P()

Background material on semaphores

*Semaphores* can be used for mutual exclusion and thread synchronization. Instead of busy waiting and wasting CPU cycles, a thread can block on a semaphore (the operating system removes the thread from the CPU scheduling or "ready" queue) if it must wait to enter its critical section or if the resource it wants is not available.

Here's an example of mutual exclusion pseudocode:

semaphore S = 1; ...P(S); N=N+1; V(S);

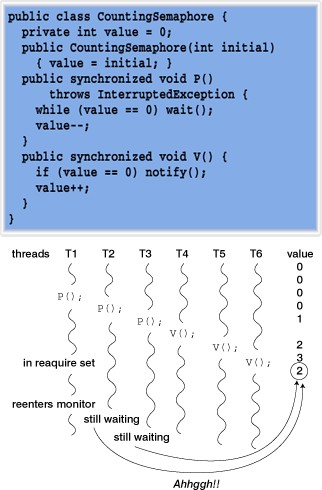
### And here's an example of condition synchronization pseudocode (resource availability):

semaphore tapeDrives = 7; ... P(tapeDrives); useTapeDrive(); V(tapeDrives);

### Java does not have explicit binary and counting semaphores, so they are provided as classes.

Pitfalls

The example of a counting semaphore shown in the following figure and in [Class](#_bookmark45) [BadCountingSemaphore1.java](#_bookmark45) on page 54 is not correct because ofbarging. We can repair the problem by changing the semantics of the semaphore value field. For code samples, see [Class Semaphore.java](#_bookmark41) on page 53 ,[Class CountingSemaphore.java](#_bookmark42) on page 54 ,[Class](#_bookmark43) [BinarySemaphore.java](#_bookmark43) on page 54 ,[Class SugarSM.java](#_bookmark44) on page 54 ,[Class](#_bookmark45) [BadCountingSemaphore1.java](#_bookmark45) on page 54 , and[Class BadCountingSemaphore2.java](#_bookmark46) on page 55 .



Suppose the semaphore's current value is "0". Three threads invoke the semaphore's P() method and wait. Then another thread calls V(), which moves one of the three waiting threads to the runnable set.

Now suppose a couple of other threads barge ahead of the signaled thread and perform two more V() operations on the semaphore. Because the semaphore's value is positive, notify() is not called and none of the waiting threads is moved to the runnable set.

Finally, the thread signaled by the first V() re-enters the semaphore monitor, decrements the semaphore value from "3" to "2", and leaves the monitor. The monitor has two waiting threads and a positive value. This is an inconsistent state for a counting semaphore.

To fix this, we could try changing the semantics of the semaphore value field, as in this counting semaphore, [Class BadCountingSemaphore2.java](#_bookmark46) on page 55 . The value is allowed to go negative, in which case its absolute value equals the number of waiting threads.

This approach fixes the barging problems but introduces an interrupt() problem. Suppose the semaphore value is "-1" due to one thread blocked in wait() inside P(). Then suppose that thread is interrupted. The value is left at "-1" even though no threads are blocked in P(). The next V() will increment the value to "0" whereas it should now be "1".

Another, more insidious problem is present: a race condition between interrupt() and notify(). Suppose several threads are blocked inside wait() and then one of them is notified and then interrupted before it reacquires the monitor lock. The notify() gets "lost" in that one of the other waiting threads should now proceed.

So we need to catch the exception when a thread is interrupted out of wait() and regenerate the notify().

Examples of semaphores

[Example bbou.java](#_bookmark47) on page 56 shows the bounded buffer producer and consumer.[Driver](#_bookmark17) [bbdr.java](#_bookmark17) on page 20 creates the producer and consumer threads. [Sample run of bbou.java](#_bookmark48) on page 56 is the sample run.

[Example dphi.java](#_bookmark49) on page 57 highlights dining philosophers.[Driver dpdr.java](#_bookmark33) on page 40 creates the philosopher threads. The sample run is [Sample run of dphi.java](#_bookmark50) on page 58 .

Try these exercises

**Exercise 1:** Modify [Example bbou.java](#_bookmark47) on page 56 so that it correctly handles multiple producer and multiple consumer threads.

**Exercise 2:** Write a semaphore solution for the database readers and writers problem.

**Exercise 3:** Write a semaphore solution for the cars at an intersection problem.

Security issues in monitors and synchronization

The public void run() method in a Thread or Runnable object can be invoked by any thread that has a reference to the Thread or Runnable object. So we prevent that at the beginning of the run() method with if (Thread.currentThread() != me) return;.

### An object's lock is accessible to any thread that has a reference to the object. This can upset the operation of a monitor if some thread decides to do something like this:

synchronized (monitor) { Thread.sleep(veryLongTime);

// or invert(veryLargeMatrix);

}

The following technique is not so bad because of the while (!condition) loop that a

### **wait()** is done in. But it does add overhead:

synchronized (monitor) { monitor.notifyAll();

}

### To protect our code from this mischief, we can code a monitor as follows, using a counting semaphore as an example. We use a wrapper class and delegate P() and V() to a private counting semaphore inside the wrapper class:

class SecureCountingSemaphore { private CountingSemaphore S = null; SecureCountingSemaphore(int initial)

{ S = new CountingSemaphore(initial); }

void P() throws InterruptedException { S.P(); } void V() { S.V(); }

}

### Another way is to use a private lock object inside the semaphore class and change synchronized methods to use the private lock:

class SecureCountingSemaphore { private int value = 0;

private Object mutex = new Object(); SecureCountingSemaphore(int initial)

{ value = initial; }

void P() throws InterruptedException { synchronized (mutex)

{ while (value == 0) mutex.wait();value--; }

}

void V() { synchronized (mutex)

{ value++;mutex.notifyAll(); }

}

}

### The next several panels display the code described in this section. To view the code, click Next; or you can go directly to the next section, [Message passing](#_bookmark51) on page 60 , and return to the code samples at another time.

Class Semaphore.java

public abstract class Semaphore { private int value = 0;

public Semaphore() {}// constructors public Semaphore(int initial) {

if (initial >= 0) value = initial;

else throw new IllegalArgumentException("initial < 0");

}

public final synchronized void P() throws InterruptedException {

while (value == 0) wait(); value--;

}

protected final synchronized void Vc() { value++;notifyAll();

}

protected final synchronized void Vb() { this.Vc();if (value > 1) value = 1;

}

public abstract void V();

public String toString() { return ".value=" + value; }

}

# Class CountingSemaphore.java

public final class CountingSemaphore extends Semaphore { public CountingSemaphore() { super(); }// constructors public CountingSemaphore(int initial) { super(initial); } public final synchronized void V() { super.Vc(); }

}

# Class BinarySemaphore.java

public final class BinarySemaphore extends Semaphore { public BinarySemaphore() { super(); }// constructors public BinarySemaphore(int initial) {

super(initial); if (initial > 1)

throw new IllegalArgumentException("initial > 1");

}

public final synchronized void V() { super.Vb(); }

}

# Class SugarSM.java

public abstract class SugarSM extends Sugar {

// syntactic sugar for semaphores protected static final void P(Semaphore s)

throws InterruptedException { s.P(); }

protected static final void V(Semaphore s) { s.V(); }

}

# Class BadCountingSemaphore1.java

public class BadCountingSemaphore1 { private int value = 0;

public BadCountingSemaphore1(int initial)

{ if (initial > 0) value = initial; } public synchronized void P()

throws InterruptedException {

while (value == 0) wait(); value--;

}

public synchronized void V() {

if (value == 0) notify();// barging causes problems value++;

}

}

# Class BadCountingSemaphore2.java

public class BadCountingSemaphore2 { private int value = 0;

public BadCountingSemaphore2(int initial)

{ if (initial > 0) value = initial; } public synchronized void P()

throws InterruptedException { value--;

if (value < 0) wait();

}

public synchronized void V() { value++;

if (value <= 0) notify(); // interrupt causes problems

}

}

# Class EfficientCountingSemaphore.java

public class EfficientCountingSemaphore { private int value = 0;

private int waitCount = 0; private int notifyCount = 0;

public EfficientCountingSemaphore() {}// constructors public EfficientCountingSemaphore(int initial) {

if (initial >= 0) value = initial;

else throw new IllegalArgumentException("initial < 0");

}

public synchronized void P() throws InterruptedException {

if (value <= waitCount) { waitCount++;

try {

do { wait(); }

while (notifyCount == 0);

} catch(InterruptedException e) { notify();

throw e;

} finally { waitCount--; } notifyCount--;

} else {

if (notifyCount > waitCount) notifyCount--;

}

value--;

}

public synchronized void V() {

value++;

if (waitCount > notifyCount) { notifyCount++;

notify();

}

}

}

# Example bbou.java

class BoundedBuffer extends SugarSM {

// designed for a single producer thread and

// a single consumer thread private int numSlots = 0; private double[] buffer = null;

private int putIn = 0, takeOut = 0; private int count = 0;

private BinarySemaphore mutex = null; private CountingSemaphore elements = null; private CountingSemaphore spaces = null; public BoundedBuffer(int numSlots) {

if (numSlots <= 0)

throw new IllegalArgumentException("numSlots <= 0"); this.numSlots = numSlots;

buffer = new double[numSlots]; mutex = new BinarySemaphore(1);

elements = new CountingSemaphore(0); spaces = new CountingSemaphore(numSlots);

}

public void deposit(double value) throws InterruptedException {

P(spaces); buffer[putIn] = value;

putIn = (putIn + 1) % numSlots; P(mutex);count++;V(mutex); V(elements);

}

public double fetch()

throws InterruptedException { double value;

P(elements);

value = buffer[takeOut];

takeOut = (takeOut + 1) % numSlots; P(mutex);count--;V(mutex); V(spaces);

return value;

}

}

# Sample run of bbou.java

% javac bbou.java bbdr.java

% java ProducersConsumers 10 1 1 2 2 5 ProducersConsumers:

numSlots=10, numProducers=1, numConsumers=1, pNap=2, cNap=2, runTime=5 Java version=1.3.0

Java vendor=IBM Corporation OS name=Linux

OS arch=i586

OS version=#1 Mon Sep 27 10:25:54 EDT 1999.2.2.12-20

No PseudoTimeSlicing needed

age=44, PRODUCER0 napping for 1701 ms age=63, Consumer0 napping for 982 ms All threads started

age=1065, Consumer0 wants to consume

age=1763, PRODUCER0 produced item 0.8235080062047989 age=1779, PRODUCER0 deposited item 0.8235080062047989 age=1780, PRODUCER0 napping for 1223 ms

age=1781, Consumer0 fetched item 0.8235080062047989 age=1783, Consumer0 napping for 529 ms

age=2325, Consumer0 wants to consume

age=3016, PRODUCER0 produced item 0.5752891675481963 age=3018, PRODUCER0 deposited item 0.5752891675481963 age=3019, PRODUCER0 napping for 1794 ms

age=3020, Consumer0 fetched item 0.5752891675481963 age=3021, Consumer0 napping for 1824 ms

age=4825, PRODUCER0 produced item 0.5010012702583668 age=4826, PRODUCER0 deposited item 0.5010012702583668 age=4827, PRODUCER0 napping for 1299 ms

age=4852, Consumer0 wants to consume

age=4853, Consumer0 fetched item 0.5010012702583668 age=4854, Consumer0 napping for 273 ms

age=5074, time to terminate the threads and exit age=5077, Consumer0 interrupted from sleep age=5093, PRODUCER0 interrupted from sleep age=6083, all threads are done

# Example dphi.java

class DiningServer extends SugarSM { private int numPhils = 0;

private int[] state = null;

private static final int THINKING = 0, HUNGRY = 1, EATING = 2; private BinarySemaphore[] self = null;

private BinarySemaphore mutex = null; public DiningServer(int numPhils) {

this.numPhils = numPhils; state = new int[numPhils];

for (int i = 0; i < numPhils; i++) state[i] = THINKING; self = new BinarySemaphore[numPhils];

for (int i = 0; i < numPhils; i++) self[i] = new BinarySemaphore(0);

mutex = new BinarySemaphore(1);

}

public void dine(String name, int id, int napEat) throws InterruptedException {

try {

takeForks(id); eat(name, napEat);

} finally {// Make sure we return the putForks(id);// forks if interrupted

}

}

private final int left(int i)

{ return (numPhils + i - 1) % numPhils;} private final int right(int i)

{ return (i + 1) % numPhils; }

private void test(int k) {

if (state[left(k)] != EATING && state[k] == HUNGRY && state[right(k)] != EATING) {

state[k] = EATING;

V(self[k]);

}

}

private void eat(String name, int napEat) throws InterruptedException {

int napping;

napping = 1 + (int) random(napEat); System.out.println("age=" + age() + ", " + name

+ " is eating for " + napping + " ms"); Thread.sleep(napping);

}

private void takeForks(int i) throws InterruptedException {

P(mutex);state[i] = HUNGRY;test(i); V(mutex); P(self[i]);

}

private void putForks(int i) throws InterruptedException {

if (state[i] != EATING) return; P(mutex);

state[i] = THINKING;test(left(i)); test(right(i)); V(mutex);

}

}

# Sample run of dphi.java

% javac dphi.java dpdr.java

% java DiningPhilosophers 5 6 4 1

DiningPhilosophers: numPhilosophers=5, runTime=6, napThink=4, napEat=1 age=42, Philosopher 0 is thinking for 683 ms

age=61, Philosopher 1 is thinking for 437 ms age=64, Philosopher 2 is thinking for 3039 ms age=66, Philosopher 3 is thinking for 3190 ms All Philosopher threads started

age=68, Philosopher 4 is thinking for 3893 ms age=511, Philosopher 1 wants to dine age=513, Philosopher 1 is eating for 82 ms

age=648, Philosopher 1 is thinking for 3631 ms age=740, Philosopher 0 wants to dine

age=741, Philosopher 0 is eating for 699 ms age=1453, Philosopher 0 is thinking for 2390 ms age=3111, Philosopher 2 wants to dine age=3111, Philosopher 2 is eating for 526 ms age=3260, Philosopher 3 wants to dine age=3651, Philosopher 2 is thinking for 2095 ms age=3652, Philosopher 3 is eating for 448 ms age=3850, Philosopher 0 wants to dine age=3851, Philosopher 0 is eating for 324 ms age=3973, Philosopher 4 wants to dine age=4111, Philosopher 3 is thinking for 2346 ms age=4191, Philosopher 0 is thinking for 3695 ms age=4192, Philosopher 4 is eating for 477 ms age=4290, Philosopher 1 wants to dine age=4291, Philosopher 1 is eating for 259 ms age=4561, Philosopher 1 is thinking for 127 ms age=4681, Philosopher 4 is thinking for 751 ms

age=4700, Philosopher 1 wants to dine age=4701, Philosopher 1 is eating for 355 ms age=5073, Philosopher 1 is thinking for 3921 ms age=5450, Philosopher 4 wants to dine age=5451, Philosopher 4 is eating for 492 ms age=5761, Philosopher 2 wants to dine age=5761, Philosopher 2 is eating for 323 ms age=5962, Philosopher 4 is thinking for 3225 ms

age=6071, time to terminate the Philosophers and exit age=6074, Philosopher 1 interrupted out of think age=6075, Philosopher 2 interrupted out of dine age=6077, Philosopher 4 interrupted out of think age=6078, Philosopher 3 interrupted out of think age=6091, Philosopher 0 interrupted out of think age=7081, all Philosophers are done

# Section 10. Message passing Some definitions

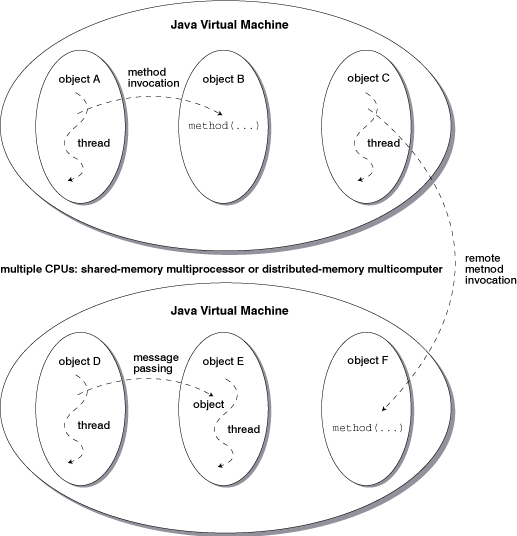
### Object-oriented programming blurs the distinction between invoking a method and sending a

message. It also blurs the distinction between shared and distributed memory computer architectures.

The figure below shows the difference between (1) invoking a method and (2) sending a message:

1. Thread leaves code in one object to execute code in another object, then comes back (shown at the top of the figure).
2. Thread sends an object to another thread, then optionally blocks until the other thread receives the message (shown at the bottom of the figure).

Message passing leads to "safer" concurrent programming since the receiving object only has one thread executing inside it.



It is important to note that multiple threads invoking methods in an object might lead to race conditions unless synchronization is properly done by making the object a monitor. With message passing, the receiving object has just one thread executing inside it, leading to "safer" concurrent programming.

If the threads have a relationship other than client/server, monitors can be awkward to use. Using message passing between the threads is easier in these situations. If the threads communicate with each other, they are called peers or filters. In this situation, the threads form a pipeline in which each thread gets its input from its predecessor in the pipeline and sends its output to its successor in the pipeline.

Options for user-written classes implementing synchronous (blocking send) and asynchronous (non-blocking send) message passing (receive always blocks) include:

* Sending object references from one thread to another within the same JVM
* Sending serialized objects through connected sockets from a thread in one JVM to a thread in another JVM

Each message passing class implements a mailbox or channel shared by a collection of threads. The one-way flow of information from sender to receiver in synchronous message passing is sometimes called a simple rendezvous. Following are examples:

* Shared type:

class Message { ... }

### Shared mailbox:

// non-blocking sends:

MessagePassing mailbox = new MessagePassing();

// capacity controlled:

MessagePassing mailbox = new MessagePassing(capacity);

### One thread:

Message ms = new Message(...); send(mailbox, ms);

### Another thread:

Message mr;

mr = (Message) receive(mailbox);

Values like int and double can be sent using the wrapper classes Integer and Double.

# Background material on message passing

### Sometimes the phrase "send a message to an object" is used to describe a thread in one object calling a method in another object. Here, that phrase is used to describe a thread in one object sending a message to a thread in another object, where the message is itself an object.

This technique is used for thread communication and synchronization in a computing environment where the threads do not have shared memory (since the threads reside in different virtual or physical machines). Hence the threads cannot share semaphores or monitors and cannot use shared variables to communicate. Message passing can still be used, of course, in a shared memory platform.

Messages are sent through a port or channel with an operation like send(channel, message)

### and received from a port or channel with an operation like receive(channel, message). Messages can be passed synchronously, meaning the sender blocks until the receiver does a receive and the receiver blocks until the sender does a send.

Because the sender and receiver are at specific known points in their code at a known

specific instant of time, synchronous message passing is also called a simple rendezvous

with a one-way flow of information from the sender to the receiver.

In asynchronous message passing, the sender does not block. If there is not a receiver waiting to receive the message, the message is queued or buffered. The receiver still blocks if there is no queued or buffered message when a receive is executed.

In conditional message passing, the message remains queued until some condition, specified by the receiver, becomes true. At that time, the message is passed to the receiver, unblocking it.

A two-way flow of information, perhaps over the network, is called an extended rendezvous and can be implemented with a pair of sends and receives. Typically a client thread uses this technique to communicate with a server thread and requests a service to be performed on its behalf.

A similar situation exists when a worker thread contacts a master thread, asking for more work to do. The client or worker sends a request and receives the reply. The server or master receives the request, performs the service, and sends the reply.

The [Example qsrt.java](#_bookmark54) on page 65 algorithm can be parallelized for a shared-memory, multiple-CPU machine by dedicating each CPU to a worker thread and using a message passing channel as a bag of tasks. The main() method puts the whole array to be sorted into the bag.

A worker extracts the task, chooses a pivot point, and partitions the array. Each of the two partitions is then put back into the bag as a task for one of the workers to perform.

Even though message passing is being used for a bag of tasks, shared memory is still required because the array is being sorted "in place" and the work requests being put into the bag are array index pairs and not pieces of the array itself.

A bag of tasks communication channel, object task, is shared by the quicksort worker threads:

MessagePassing task = new MessagePassing();

### The quicksort worker threads get tasks from the task bag inside a while loop:

while (true) {

m = (Task) receive(task); quickSort(id, m.left, m.right);

}

### The quicksort worker threads create tasks and put them into the task bag:

if (right-(l+1) > 0) send(task, new Task(l+1, right)); if ((l-1)-left > 0) send(task, new Task(left, l-1));

# User-written classes

### The following are two user-written classes:

* [Class SugarMP.java](#_bookmark52) on page 64 provides syntactic sugar so thatsend(mailbox, ms) can be used instead of mailbox.send(ms) and mr = receive(mailbox) instead of mr = mailbox.receive().

### [Class MessagePassing.java](#_bookmark53) on page 64 sends object references within one JVM asynchronously. Synchronous message passing can be done with the Rendezvous class, which we'll discuss later.

Examples of message passing

The first example contains worker threads and a bag of tasks, the second example contains

*filter* threads, and the third example contains peer threads.

* [Example qsrt.java](#_bookmark54) on page 65 : Quicksort (worker threads).[Sample run of qsrt.java](#_bookmark55) on page 67 is the sample run.
* [Example pasv.java](#_bookmark56) on page 67 : Parallel Sieve of Eratosthenes (filter threads).[Sample run of pasv.java](#_bookmark57) on page 69 is the sample run.
* [Example rads.java](#_bookmark58) on page 69 : Parallel Radix Sort (peer threads).[Sample run of rads.java](#_bookmark59) on page 71 is the sample run.

You can use [Remote Method Invocation (RMI)](#_bookmark69) on page 98 to implement message passing between threads in different JVMs that may also be on different physical machines.

The next several panels display the code described in this section. To view the code, click Next; or you can go directly to the next section, [Rendezvous](#_bookmark60) on page 72 , and return to the code samples at another time.

Class SugarMP.java

public abstract class SugarMP extends Sugar {

// syntactic sugar for message passing

protected static final void send(MessagePassing mp, Object o) throws InterruptedException { mp.send(o); }

protected static final Object receive(MessagePassing mp) throws InterruptedException { return mp.receive(); }

}

# Class MessagePassing.java

import java.util.Vector;

public final class MessagePassing {

// Implements asynchronous message passing:

// sends do not block (until the message is

// received), receives block of course until

// a message is received.

private int capacity = 0;// for capacity control

// messages are delivered FIFO (in the order they are sent) private final Vector messages = new Vector();

// receivers get messages FIFO (in the order they call receive) private final Vector receivers = new Vector();

public MessagePassing() { this(0); }

public MessagePassing(int c) {// capacity limit super();

if (c < 0) throw new IllegalArgumentException("capacity < 0");

// zero means no limit imposed here else if (c > 0) {

capacity = c; messages.ensureCapacity(capacity);

}

}

public final synchronized void send(Object m) throws InterruptedException {

if (m == null) throw new NullPointerException("null message"); if (capacity > 0)

while (messages.size() == capacity) wait(); messages.addElement(m);// add at end notifyAll();

}

public final synchronized Object receive() throws InterruptedException {

Object receivedMessage = null; Thread me = Thread.currentThread();

receivers.addElement(me);// add at end try {

while (messages.isEmpty() || me != receivers.elementAt(0)) wait();

// If we are interrupted after being notified and there is a

// message here for us, pretend we were interrupted before

// being notified and leave the message for someone else.

// Thus, there is no `catch (InterruptedException e) {...}'

// block here.

receivedMessage = messages.elementAt(0); messages.removeElementAt(0);

return receivedMessage;

} finally {

// We need to do this if we get a message or if we were

// interrupted. receivers.removeElement(me);

// The notifyAll is needed because several messages

// might be put in the messages vector before any

// waiting receivers get back in.The receiver who

// is first in the receivers vector might not get back

// in until last!So it needs to cause the waiting

// receivers to come back in again so the second in

// line can get a message. notifyAll();

}

}

}

# Example qsrt.java

class Task {

public int left = -1, right = -1; public Task(int left, int right)

{ this.left = left; this.right = right; }

}

class QuickSort extends SugarMP implements Runnable { private static int N = 10;

private static int RANGE = 100; private static int NCPU = 4;

private static final MessagePassing doneCount

= new MessagePassing();

private static final MessagePassing task

= new MessagePassing(); private static int[] nums = null; private String name = null; private int id = -1;

private Thread me = null; private QuickSort(int id) {

this.name = "Worker" + id; this.id = id;

(me = new Thread(this)).start();

}

private static void quickSort

(int worker, int left, int right) throws InterruptedException {

int pivot = nums[left]; int l = left, r = right; boolean done = false;

Integer doneMessage = new Integer(worker); while (!done) {

if (nums[l+1] > pivot) {

while (r > l+1 && nums[r] > pivot) { r--; } if (r > l+1) { l++;

int temp = nums[r]; nums[r] = nums[l]; nums[l] = temp;

done = l >= r-1;

} else done = true;

} else { l++; done = l >= r; }

}

int temp = nums[left]; nums[left] = nums[l]; nums[l] = temp;

if (right-(l+1) > 0) send(task, new Task(l+1, right)); else if (right-(l+1) == 0) send(doneCount, doneMessage); send(doneCount, doneMessage);

if ((l-1)-left > 0) send(task, new Task(left, l-1)); else if ((l-1)-left == 0) send(doneCount, doneMessage);

}

public void timeToQuit() { me.interrupt(); }

public void pauseTilDone() throws InterruptedException

{ me.join(); } public void run() {

Task m = null; while (true) {

if (Thread.interrupted()) { System.out.println("age=" + age() + ", " + name

+ " interrupted"); return;

}

try {

m = (Task) receive(task); quickSort(id, m.left, m.right);

} catch (InterruptedException e) { System.out.println("age=" + age() + ", " + name

+ " interrupted out of send/receive"); return;

}

}

}

public static void main(String[] args) { try {

N = Integer.parseInt(args[0]); RANGE = Integer.parseInt(args[1]);

NCPU = Integer.parseInt(args[2]);

} catch (Exception e) { /\* use defaults \*/ } System.out.println("Quick sorting " + N

+ " random numbers between 1 and " + RANGE

+ " using " + NCPU + " CPUs");

nums = new int[N];

for (int i = 0; i < N; i++)

nums[i] = 1 + (int) (random()\*RANGE); System.out.println("Original numbers:"); for (int i = 0; i < N; i++)

System.out.print(" " + nums[i]); System.out.println();

// create the workers with self-starting threads QuickSort[] q = new QuickSort[NCPU];

new PseudoTimeSlicing(); // for Solaris, not Windows 95/NT for (int i = 0; i < NCPU; i++) q[i] = new QuickSort(i); try {

send(task, new Task(0, N-1));

// wait for enough "singletons" to be produced for (int i = 0; i < N; i++) receive(doneCount); System.out.println("Sortednumbers:");

for (int i = 0; i < N; i++) System.out.print(" " + nums[i]);

System.out.println();

for (int i = 0; i < NCPU; i++) q[i].timeToQuit(); Thread.sleep(1000);

for (int i = 0; i < NCPU; i++) q[i].pauseTilDone();

} catch (InterruptedException e) { /\* ignored \*/ } System.out.println("age()=" + age() + ", done"); System.exit(0);

}

}

# Sample run of qsrt.java

% javac qsrt.java

% java QuickSort 15 1000 3

Quick sorting 15 random numbers between 1 and 1000 using 3 CPUs Original numbers:

594 637 361 87 207 803 8 870 979 333 121 601 353 613 586

Java version=1.3.0

Java vendor=IBM Corporation OS name=Linux

OS arch=i586

OS version=#1 Mon Sep 27 10:25:54 EDT 1999.2.2.12-20

No PseudoTimeSlicing needed Sortednumbers:

8 87 121 207 333 353 361 586 594 601 613 637 803 870 979

age=77, Worker2 interrupted

age=96, Worker0 interrupted out of send/receive age=97, Worker1 interrupted out of send/receive age()=1087, done

# Example pasv.java

class Filter extends SugarMP implements Runnable { private MessagePassing in = null, out = null;

private int prime = 0, countIn = 0, countOut = 0; public Filter(MessagePassing in, MessagePassing out) {

this.in = in;this.out = out;

}

private void print() {

System.out.println("age()=" + age() + " received prime " + prime

+ ", countIn=" + countIn + ", countOut=" + countOut);

}

public void run () {

if (in == null) {// source thread int number = 3;

while (true) {

try { send(out, new Integer(number)); } catch (InterruptedException e) { return; } number += 2;

}

} else {// filter threads int number = 0;

try { prime = ((Integer) receive(in)).intValue(); } catch (InterruptedException e) { return; }

while (true) {

if (Thread.interrupted()) { print();return; } try {

number = ((Integer) receive(in)).intValue(); countIn++; if (number % prime != 0) {

send(out, new Integer(number));countOut++;

}

} catch (InterruptedException e) { print();return; }

}

}

}

}

class ParallelSieve extends SugarMP { public static void main(String[] args) {

int n = 8;

try { n = Integer.parseInt(args[0]); } catch (Exception e) { /\* use default \*/ } if (n < 1) {

System.out.println("Generate at least one prime number."); System.exit(1);

}

System.out.println("ParallelSieve: generating the first "

+ n + " prime numbers greater than 2"); MessagePassing in = null, out = null; Thread[] filter = new Thread[n];

for (int i = 0; i < n; i++) {

// Use capacity control so the early threads do

// not get way ahead of what is needed by the

// latter threads and fill up JVM memory. out = new MessagePassing(n);

filter[i] = new Thread(new Filter(in, out)); in = out;

}

new PseudoTimeSlicing(); // for Solaris, not Windows 95/NT for (int i = 0; i < n; i++) filter[i].start();

try {

int prime = ((Integer) receive(out)).intValue(); for (int i = 0; i < n; i++) filter[i].interrupt(); for (int i = 0; i < n; i++) filter[i].join();

System.out.println("age()=" + age() + " lastprime " + prime);

} catch (InterruptedException e) { /\* ignored \*/ } System.exit(0);

}

}

# Sample run of pasv.java

% javac pasv.java

% java ParallelSieve 20

ParallelSieve: generating the first 20 prime numbers greater than 2 Java version=1.3.0

Java vendor=IBM Corporation OS name=Linux

OS arch=i586

OS version=#1 Mon Sep 27 10:25:54 EDT 1999.2.2.12-20

No PseudoTimeSlicing needed

age()=2158 received prime 11, countIn=458, countOut=416 age()=2165 received prime 13, countIn=395, countOut=365 age()=2166 received prime 7, countIn=559, countOut=479 age()=2167 received prime 5, countIn=725, countOut=580 age()=2169 received prime 29, countIn=230, countOut=223 age()=2171 received prime 19, countIn=301, countOut=284 age()=2172 received prime 17, countIn=344, countOut=322 age()=2174 received prime 43, countIn=133, countOut=132 age()=2175 received prime 23, countIn=263, countOut=251 age()=2176 received prime 47, countIn=111, countOut=110 age()=2178 received prime 37, countIn=177, countOut=176 age()=2179 received prime 31, countIn=202, countOut=198 age()=2180 received prime 3, countIn=1120, countOut=746 age()=2182 received prime 41, countIn=155, countOut=154 age()=2184 received prime 53, countIn=89, countOut=88 age()=2185 received prime 59, countIn=67, countOut=66 age()=2187 received prime 61, countIn=45, countOut=44 age()=2188 received prime 67, countIn=23, countOut=22 age()=2190 received prime 71, countIn=1, countOut=1 age()=2191 lastprime 73

# Example rads.java

class Result { public int number, count;

public Result(int n, int c) { number = n; count = c; }

}

class Peer extends SugarMP implements Runnable { private int N = -1, id = -1, mine = 0; private MessagePassing[] channel = null; private MessagePassing reply = null;

public Peer(int N, int id, int mine, MessagePassing[] channel, MessagePassing reply) {

this.N = N; this.id = id; this.mine = mine;

this.channel = channel; this.reply = reply;

new Thread(this).start();

}

public void run() {

int count = 0, other = 0; try {

// Send my number to all the other workers. for (int i = 0; i < N; i++)

if (i != id) send(channel[i], new Integer(mine));

// Of the N-1 numbers sent to me by the other workers,

// count how many are less than my number. for (int i = 1; i < N; i++) {

other = ((Integer) receive(channel[id])).intValue(); if (other < mine) count++;

}

// Send my count of less-than-mine-seen back to main(). send(reply, new Result(mine, count));

} catch (InterruptedException e) { return; }

}

}

class RadixSort extends SugarMP {

public static void main(String[] args) { int N = 15;

int RANGE = 1000;

int[] nums = null; MessagePassing[] channel = null; MessagePassing reply = null;

try {

N = Integer.parseInt(args[0]); RANGE = Integer.parseInt(args[1]);

} catch (Exception e) { /\* use defaults \*/ } System.out.println("Radix sorting " + N

+ " random integers between 1 and " + RANGE); nums = new int[N];

for (int i = 0; i < N; i++)

nums[i] = 1 + (int)random(RANGE); System.out.println("Original numbers:"); for (int i = 0; i < N; i++)

System.out.print(" " + nums[i]);System.out.println();

// Set up the reply channel. reply = new MessagePassing(); channel = new MessagePassing[N];

// Set up the communication channels. for (int i = 0; i < N; i++)

channel[i] = new MessagePassing();

// Start the worker threads. for (int i = 0; i < N; i++)

new Peer(N, i, nums[i], channel, reply); int[] tallyCounts = new int[N];

for (int i = 0; i < N; i++) tallyCounts[i] = 0; try {

// Gather the results.

for (int i = 0; i < N; i++) {

Result r = (Result) receive(reply);

// Put the number where it belongs in the sorted order,

// which is the value of the counter in which it recorded

// the number of less-than-its-own numbers it saw. nums[r.count] = r.number;

tallyCounts[r.count]++;

}

} catch (InterruptedException e) { /\* ignored \*/ } System.out.println("Sortednumbers:");

for (int i = 0; i < N; i++)

System.out.print(" " + nums[i]); System.out.println(); for (int i = 0; i < N; i++)

// Zeros show where duplicates have occured. System.out.print(" " + tallyCounts[i]); System.out.println();

System.out.println("age()=" + age() + ", done"); System.exit(0);

}

}

# Sample run of rads.java

% javac rads.java

% java RadixSort 20 100

Radix sorting 20 random integers between 1 and 100 Original numbers:

58 71 78 26 47 34 30 9 99 60 60 70 51 71 93 76 2 87 49 14

Sortednumbers:

2 9 14 26 30 34 47 49 51 58 60 70 70 71 93 76 78 87 93 99

1 1 1 1 1 1 1 1 1 1 2 0 1 2 0 1 1 1 1 1

age()=164, done

# Section 11. Rendezvous Some definitions

### In general client-server programming, a client thread interacts with the server thread by

sending a request message followed immediately by a receive that blocks until the server sends a reply message containing the results of the request. This is called a rendezvous (or sometimes an extended rendezvous).

A monitor is a passive object and can be used to implement a server. An active object, in which an independent thread executes, can also act as a server by using the rendezvous style of message passing. Here are a few examples:

* Mailbox shared by the client and server:

MessagePassing mailbox = new MessagePassing();

### Client:

send(mailbox, request); reply = receive(mailbox);

### Server:

request = receive(mailbox);

***compute reply*; send(mailbox, reply);**

# Background material on rendezvous

### An extended rendezvous is also called a remote procedure call from a client to a server (or a worker to the master) because it resembles (and syntactic sugar can make it nearly identical to) a call to a procedure on a remote machine that is executed there.

Typically the call represents a request for service, such as reading a file that resides on the remote machine. The server may handle the request in its main thread or the server may spawn a new thread to handle the request while the server's main thread handles additional requests for service from other clients. The latter gives greater throughput and efficiency because a lengthy request would otherwise delay the handling of requests from the other clients.

An addressing mechanism is needed so the client can contact an appropriate server. In the local case (everything in the same JVM), an object can be used as the place for the client and server to "meet" and establish a rendezvous. The server calls a method in the object and blocks until the client calls a method.

At this point in time, both methods return a newly created object that the client and server subsequently use for the two-way flow of information. This object contains a message passing channel shared by them. In the remote case, the client uses the server's machine

name and a TCP/IP port number to address the server; the server "listens" on the TCP/IP port. A client creates an addressing object using the server's machine name and port number in the object's constructor; the server uses just the port number.

When the rendezvous occurs, the object is constructed and returned to both the client and server. In the local case (within the same JVM), the client and server share this object and use it to transact (synchronous message passing of object references).

In the remote case (between JVMs that might be on different physical machines), each gets its own object and the object contains a socket to the other JVM (and machine). Objects are serialized through the socket.

User-written classes

Now let's look at some user-written classes and definitions.

In a transaction, two threads exchange information synchronously. [Class Transaction.java](#_bookmark61) on page 76 is a user-written class used for one such exchange. Coding is as follows:

* Created by the client, shared by the client and server:

Transaction t = new Transaction(request);

### Client:

reply = t.clientAwaitReply();

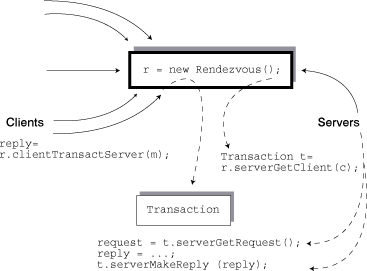
### Server:

request = t.serverGetRequest();

***compute reply*; t.serverMakeReply(reply);**

### A conditional rendezvous builds on the idea of a transaction. It allows the server to specify criteria of whom to rendezvous with next.

The figure below illustrates the concepts of rendezvous.



[Interface RendezvousCondition.java](#_bookmark62) on page 76 demonstrates the criterion interface.

An object created from [Class Rendezvous.java](#_bookmark63) on page 77 is shared by the client and server. Coding is as follows:

* Shared by the client and server:

Rendezvous r = new Rendezvous();

### Client:

reply = r.clientTransactServer(request);

### Server:

RendezvousCondition c =

new RendezvousCondition(...); Transaction t = r.serverGetClient(c); Object request = t.serverGetRequest(); Object reply = ...;// compute reply t.serverMakeReply(reply);

Multiple servers can share a single Rendezvous object.

### A server can call serverGetClient(c) (a "nested" call) again while in the middle of a transaction with a client (meaning after calling serverGetClient to get a client's request

but before calling serverMakeReply to reply to that client).

### [Remote Method Invocation (RMI)](#_bookmark69) on page 98 can be used to implement rendezvous between threads in different JVMs that may also be on different physical machines.

Peer-to-peer programming offers other options.

[Class Rendezvous.java](#_bookmark63) on page 77 has four more methods --send(m), call(m), receive(c), and receive() -- that can be used for asynchronous and synchronous (conditional) message passing among a collection of peer threads.

A receiving peer can specify a condition for messages it is willing to receive, just as can be specified by a server for a rendezvous. To block until the message is received, use call(m) instead of send(m).

[Class SugarRE.java](#_bookmark64) on page 79 provides syntactic sugar so thatsend(rn, ms) can be used instead of rn.send(ms) (ditto for call(ms)) and mr = receive(rn, rc) instead of mr = rn.receive(rc) (ditto for receive()).

# Examples of rendezvous

### Example of clients and a server include:

* [Example dpre.java](#_bookmark65) on page 79 : Rendezvous dining philosophers
* [Driver dpdr.java](#_bookmark33) on page 40 : Creates the philosopher threads
* [Sample run of dpre.java](#_bookmark66) on page 81 : Displays the sample run

Because the philosophers only need to block until their request is conditionally received by the server and because they are not interested in the reply message, the philosophers use call instead of clientTransactServer. Similarly, the server uses receive instead of serverGetClient.

[Example pcre.java](#_bookmark67) on page 82 is an example of producer and consumer peers in which a consumer specifies that it receive the message with the smallest value among the yet unreceived messages. A producer can act asynchronously by using send or synchronously to find out which consumer got its message by using clientTransactServer. [Sample run](#_bookmark68) [of pcre.java](#_bookmark68) on page 85 is the sample run.

The next several panels contain an exercise and display the code described in this section. To view the exercise and code, click Next; or you can go directly to the next section, [Remote](#_bookmark69) [Method Invocation (RMI)](#_bookmark69) on page 98 , and return to the code samples at another time.

Try this exercise

Consider a bank that makes loans and accepts loan repayments from its customers. Use nested serverGetClient(c) calls by the bank server thread to prevent starvation of a customer needing a particularly large loan: the bank accepts only repayments until it has enough funds to make the large loan.

Try implementing the same bank server as a passive monitor object. Which is easier?

Class Transaction.java

public class Transaction {

// Designed to be used by exactly one client

// transacting exactly once with one specific server. private Object request = null;

private Object reply = null; public Transaction (Object m) {

if (m == null)

throw new NullPointerException("m == null"); request = m;

}

public synchronized Object clientAwaitReply() throws InterruptedException {

Object m = null; try {

while (reply == null) wait();

} catch (InterruptedException e) {

// We have been interrupted while waiting for the

// server to process our request and/or generate

// a reply.Since we no longer want the server to

// process our request, we will null it out.This

// means the server must check for a null return

// value from the serverGetRequest method.If the

// request is already null at this point, then the

// server must have already gotten it. request = null;

if (reply == null)

throw new InterruptedException("reply not available"); else

Thread.currentThread().interrupt();// reply available

}

m = reply; reply = null; return m;

}

public synchronized Object serverGetRequest() { Object m = request;

request = null; return m;

}

public synchronized void serverMakeReply(Object m) { if (m == null)

throw new NullPointerException("m == null"); reply = m;

notify();// at most one thread (the client) waiting

}

}

# Interface RendezvousCondition.java

import java.util.Vector;

public interface RendezvousCondition {

/\*

* **The information available to the checkCondition method is:**
* **the particular message being evaluated,**
* **blockedMessages.elementAt(messageNum);**
* **the queue of blocked messages itself, blockedMessages; and**
* **the number of blocked servers, numBlockedServers.**
* **This is the state of the Rendezvous object.The**
* **particular message can be checked to see if it meets the**
* **condition and this test may involve counting how many**
* **blocked messages meet some other criterion and/or the number**
* **of blocked servers.**

\*

* **We are depending on the programmer not to mess with the**
* **blockedMessages Vector.The Rendezvous object is**
* **graciously making it available, so do not abuse!**

\*/

public abstract boolean checkCondition (int messageNum, Vector blockedMessages,

int numBlockedServers);

}

# Class Rendezvous.java

import java.util.Vector; public final class Rendezvous {

private final Vector messages = new Vector(); private final Vector transactions = new Vector(); private int numServers = 0;

// An anonymous class whose checkCondition method returns true. private RendezvousCondition alwaysTrue =

new RendezvousCondition() {

public boolean checkCondition(int messageNum,

Vector blockedMessages, int numBlockedServers) { return true;

}

};

// If there are more waiting servers than messages, then

// starvation might occur among the waiting servers because

// no attempt is made to match a message with the longest

// waiting server.On the other hand, messages are checked

// for a matching condition in the order the messages arrive. public Rendezvous() { super(); }

// The server calls this method to get the Transaction object to

// use with the client.When this method returns, the server

// will do two things with the Transaction return value: invoke

// serverGetRequest() and then invoke serverMakeReply().

// The Transaction object is then discarded by the server. public synchronized Transaction serverGetClient

(RendezvousCondition condition) throws InterruptedException {

if (condition == null)

throw new NullPointerException("null condition"); numServers++;

Transaction client = null; boolean matched = false; try {

while (true) {

int numMessages = messages.size();

for (int j = 0; j < numMessages; j++) {

/\*

* + **We are running security and protection risks making the**
  + **messages Vector available to the outside.**
  + **Caveat emptor!**

\*/

if (condition.checkCondition(j, messages, numServers)) { messages.removeElementAt(j);

client = (Transaction) transactions.elementAt(j);

transactions.removeElementAt(j); matched = true;

break;

}

}

if (matched) return client;

else wait();// for another message to arrive

}

} finally {

// We need to do this if we get a message or if we were

// interrupted. numServers--;

// Since we have changed numServers, we need to force

// all servers to check for a match again because

// numServers is passed to checkCondition(). notifyAll();

}

}

// Transact with any waiting client.

public synchronized Transaction serverGetClient() throws InterruptedException {

return serverGetClient(alwaysTrue);

}

// The client calls this method to transact with the server. public Object clientTransactServer(Object message)

throws InterruptedException { return put(message, true);

}

// The client calls this method indirectly.

private Object put(Object message, boolean synchronous) throws InterruptedException {

if (message == null)

throw new NullPointerException("null message"); Transaction t = new Transaction(message); synchronized (this) {

messages.addElement(message); transactions.addElement(t); this.notifyAll();

}

Object m = null;

// If not synchronous, the server removes the message

// and transaction from their vectors. if (synchronous) {

try { m = t.clientAwaitReply(); }

finally { // in case interrupted out of waiting synchronized (this) {

messages.removeElement(message); transactions.removeElement(t);

}

}

}

return m;

}

// A peer calls this method to send asynchronously a message

// to another peer (using receive). public void send(Object message)

throws InterruptedException { put(message, false);

}

// A peer calls this method to send synchronously a message

// to another peer.

public void call(Object message) throws InterruptedException {

put(message, true);// Discard the reply.

}

// A peer calls this method to receive a message conditionally

// from another peer.

public Object receive(RendezvousCondition condition) throws InterruptedException {

Transaction t = serverGetClient(condition); Object m = t.serverGetRequest();

// A kludge just in case receive() is called

// erroneously when a client is waiting inside t's

// clientAwaitReply() inside clientTransactServer();

// but really is needed when a client is waiting

// inside call(). t.serverMakeReply("Fake reply."); return m;

}

// Receive any waiting message.

public Object receive() throws InterruptedException { return receive(alwaysTrue);

}

}

# Class SugarRE.java

import java.rmi.\*;

public abstract class SugarRE extends Sugar {

// syntactic sugar for rendezvous

protected static final void send(Rendezvous rn, Object o) throws InterruptedException { rn.send(o); }

protected static final void call(Rendezvous rn, Object o) throws InterruptedException { rn.call(o); }

protected static final Object receive(Rendezvous rn, RendezvousCondition rc)

throws InterruptedException { return rn.receive(rc); } protected static final Object receive(Rendezvous rn)

throws InterruptedException { return rn.receive(); }

// syntactic sugar for remote rendezvous

protected static final void send(RemoteRendezvous rn, Object o) throws RemoteException, InterruptedException

{ rn.send(o); }

protected static final void call(RemoteRendezvous rn, Object o) throws RemoteException, InterruptedException

{ rn.call(o); }

protected static final Object receive(RemoteRendezvous rn, RendezvousCondition rc)

throws RemoteException, InterruptedException

{ return rn.receive(rc); }

protected static final Object receive(RemoteRendezvous rn) throws RemoteException, InterruptedException

{ return rn.receive(); }

}

# Example dpre.java

import java.util.Vector;

class EatCondition implements RendezvousCondition { private int numPhils = 0;

private int[] state = null;

private int EATING = -1;

public EatCondition(int[] state, int EATING) { this.state = state;

numPhils = state.length; this.EATING = EATING;

}

private final int left(int i)

{ return (numPhils + i-1) % numPhils; } private final int right(int i)

{ return (i+1) % numPhils; } public boolean checkCondition

(int messageNum, Vector blockedMessages, int numBlockedServers) {

Object message = blockedMessages.elementAt(messageNum); int id = ((Integer) message).intValue();

int size = blockedMessages.size(); // not used if (id < 0) return true;// putForks() else if (state[left(id)] != EATING

&& state[right(id)] != EATING)

return true;// takeForks() else return false;

}

}

class DiningServer extends SugarRE implements Runnable { private int numPhils = 0;

private int[] state = null; private Rendezvous r = null;

private static final int THINKING = 0, HUNGRY = 1, EATING = 2; private String name = "DiningServer";

private Thread me = null;

public DiningServer(int numPhils) { this.numPhils = numPhils;

state = new int[numPhils];

for (int i = 0; i < numPhils; i++) state[i] = THINKING; r = new Rendezvous();

(me = new Thread(this)).start();

}

public void dine(String name, int id, int napEat) throws InterruptedException {

try {

takeForks(id); eat(name, napEat);

} finally {// Make sure we return the putForks(id);// forks if interrupted

}

}

private void eat(String name, int napEat) throws InterruptedException {

int napping;

napping = 1 + (int) random(napEat); System.out.println("age=" + age() + ", " + name

+ " is eating for " + napping + " ms"); Thread.sleep(napping);

}

private void takeForks(int id) throws InterruptedException { state[id] = HUNGRY;// not used call(r, new Integer(id));

}

private void putForks(int id) throws InterruptedException { if (state[id] != EATING) return;

call(r, new Integer(-id-1));

}

public void run() {// makes atomic state changes if (Thread.currentThread() != me) return; while (true) {

if (Thread.interrupted()) { System.out.println("age=" + age() + ", " + name

+ " interrupted"); return;

}

try {

Object m = receive(r,

new EatCondition(state, EATING)); if (m != null) {

int id = ((Integer) m).intValue(); if (id < 0) state[-id-1] = THINKING; else state[id] = EATING;

} else

System.out.println("age=" + age() + ", " + name

+ " received null request");

} catch (InterruptedException e) { System.out.println("age=" + age() + ", " + name

+ " interrupted out of rendezvous"); return;

}

}

}

}

# Sample run of dpre.java

% javac dpre.java dpdr.java

% java DiningPhilosophers 5 6 4 1

DiningPhilosophers: numPhilosophers=5, runTime=6, napThink=4, napEat=1 age=315, Philosopher 0 is thinking for 3524 ms

age=323, Philosopher 1 is thinking for 1910 ms age=325, Philosopher 2 is thinking for 2383 ms age=327, Philosopher 3 is thinking for 1477 ms All Philosopher threads started

age=329, Philosopher 4 is thinking for 3131 ms

|  |  |  |  |
| --- | --- | --- | --- |
| **age=1845,** | **Philosopher** | **3** | **wants to dine** |
| **age=1851,** | **Philosopher** | **3** | **is eating for 264 ms** |
| **age=2127,** | **Philosopher** | **3** | **is thinking for 1336 ms** |
| **age=2234,** | **Philosopher** | **1** | **wants to dine** |
| **age=2236,** | **Philosopher** | **1** | **is eating for 151 ms** |
| **age=2405,** | **Philosopher** | **1** | **is thinking for 143 ms** |
| **age=2564,** | **Philosopher** | **1** | **wants to dine** |
| **age=2565,** | **Philosopher** | **1** | **is eating for 680 ms** |
| **age=2724,** | **Philosopher** | **2** | **wants to dine** |
| **age=3257,** | **Philosopher** | **1** | **is thinking for 1453 ms** |
| **age=3258,** | **Philosopher** | **2** | **is eating for 539 ms** |
| **age=3475,** | **Philosopher** | **3** | **wants to dine** |
| **age=3476,** | **Philosopher** | **4** | **wants to dine** |
| **age=3477,** | **Philosopher** | **4** | **is eating for 963 ms** |
| **age=3805,** | **Philosopher** | **2** | **is thinking for 3403 ms** |
| **age=3854,** | **Philosopher** | **0** | **wants to dine** |
| **age=4458,** | **Philosopher** | **0** | **is eating for 367 ms** |
| **age=4459,** | **Philosopher** | **3** | **is eating for 502 ms** |
| **age=4459,** | **Philosopher** | **4** | **is thinking for 814 ms** |
| **age=4725,** | **Philosopher** | **1** | **wants to dine** |
| **age=4835,** | **Philosopher** | **0** | **is thinking for 2087 ms** |
| **age=4836,** | **Philosopher** | **1** | **is eating for 689 ms** |
| **age=4975,** | **Philosopher** | **3** | **is thinking for 1231 ms** |
| **age=5285,** | **Philosopher** | **4** | **wants to dine** |
| **age=5286,** | **Philosopher** | **4** | **is eating for 857 ms** |
| **age=5535,** | **Philosopher** | **1** | **is thinking for 3514 ms** |

age=6155, Philosopher 4 is thinking for 704 ms age=6252, Philosopher 3 wants to dine age=6253, Philosopher 3 is eating for 618 ms

age=6335, time to terminate the Philosophers and exit age=6337, Philosopher 0 interrupted out of think age=6339, Philosopher 1 interrupted out of think age=6340, Philosopher 2 interrupted out of think age=6342, Philosopher 3 interrupted out of dine age=6343, Philosopher 4 interrupted out of think age=7345, all Philosophers are done

# Example pcre.java

import java.util.Vector;

class Producer extends SugarRE implements Runnable { private String name = null;

private boolean synchronous = false; private int pNap = 0; // milliseconds private Rendezvous rn = null; private Thread me = null;

public Producer(String name, boolean synchronous, int pNap, Rendezvous rn) {

this.name = name; this.synchronous = synchronous; this.pNap = pNap;

this.rn = rn;

(me = new Thread(this)).start();

}

public void timeToQuit() { me.interrupt(); }

public void pauseTilDone() throws InterruptedException

{ me.join(); } public void run() {

if (Thread.currentThread() != me) return; double item;

int napping; while (true) {

if (Thread.interrupted()) { System.out.println("age=" + age() + ", " + name

+ " interrupted"); return;

}

napping = 1 + (int) random(pNap); System.out.println("age=" + age() + ", " + name

+ " napping for " + napping + " ms"); try { Thread.sleep(napping); }

catch (InterruptedException e) { System.out.println("age=" + age() + ", " + name

+ " interrupted from sleep"); return;

}

item = random();

System.out.println("age=" + age() + ", " + name

+ " produced item " + item); try {

Double d = new Double(item); if (synchronous) {

Object reply = rn.clientTransactServer(d); System.out.println("age=" + age() + ", " + name

+ ", reply= " + reply);

} else send(rn, d);

} catch (InterruptedException e) {

System.out.println("age=" + age() + ", " + name

+ " interrupted from send"); return;

}

System.out.println("age=" + age() + ", " + name

+ " sent item " + item);

}

}

}

class ConsumerCondition implements RendezvousCondition { public ConsumerCondition() { }

public boolean checkCondition

(int messageNum, Vector blockedMessages, int numBlockedServers) {

int size = blockedMessages.size(); if (size == 1) return true;

/\*

* **Select the smallest value in the queue.**

\*/

double smallest = ((Double) blockedMessages.elementAt(0)).doubleValue(); int where = 0;

for (int i = 1; i < size; i++) {

double d = ((Double) blockedMessages.elementAt(i)).doubleValue(); if (d < smallest) { smallest = d; where = i; }

}

if (where == messageNum) return true; else return false;

}

}

class Consumer extends SugarRE implements Runnable { private String name = null;

private boolean synchronous = false; private int cNap = 0; // milliseconds private Rendezvous rn = null; private Thread me = null;

private RendezvousCondition rc = null;

public Consumer(String name, boolean synchronous, int cNap, Rendezvous rn) {

this.name = name; this.synchronous = synchronous; this.cNap = cNap;

this.rn = rn;

rc = new ConsumerCondition(); (me = new Thread(this)).start();

}

public void timeToQuit() { me.interrupt(); }

public void pauseTilDone() throws InterruptedException

{ me.join(); } public void run() {

if (Thread.currentThread() != me) return; double item;

int napping; while (true) {

if (Thread.interrupted()) { System.out.println("age=" + age() + ", " + name

+ " interrupted"); return;

}

napping = 1 + (int) random(cNap); System.out.println("age=" + age() + ", " + name

+ " napping for " + napping + " ms"); try { Thread.sleep(napping); }

catch (InterruptedException e) { System.out.println("age=" + age() + ", " + name

+ " interrupted from sleep");

return;

}

System.out.println("age=" + age() + ", " + name

+ " wants to consume"); try {

if (synchronous) {

Transaction t = rn.serverGetClient(rc); Double d = (Double) t.serverGetRequest(); if (d != null) {

item = d.doubleValue(); t.serverMakeReply(name + " got it!");

System.out.println("age=" + age() + ", " + name

+ " received item " + item);

} else {

System.out.println("age=" + age() + ", " + name

+ " received null item");

}

} else {

Double d = (Double) receive(rn, rc); if (d != null) {

item = d.doubleValue(); System.out.println("age=" + age() + ", " + name

+ " received item " + item);

} else {

System.out.println("age=" + age() + ", " + name

+ " received null item");

}

}

} catch (InterruptedException e) { System.out.println("age=" + age() + ", " + name

+ " interrupted from receive"); return;

}

}

}

}

class ProducersConsumers extends Sugar { public static void main(String[] args) {

boolean synchronous = false; int numProducers = 1;

int numConsumers = 1;

int pNap = 2;// defaults

int cNap = 2;// in

int runTime = 60;// seconds try {

synchronous = args[0].equals("yes"); numProducers = Integer.parseInt(args[1]); numConsumers = Integer.parseInt(args[2]); pNap = Integer.parseInt(args[3]);

cNap = Integer.parseInt(args[4]); runTime = Integer.parseInt(args[5]);

} catch (Exception e) { /\* use defaults \*/ } System.out.println("ProducersConsumers:\n synchronous="

+ synchronous + ", numProducers="

+ numProducers + ", numConsumers=" + numConsumers

+ "\n pNap=" + pNap + ", cNap=" + cNap

+ ", runTime=" + runTime);

// create the message passing channel Rendezvous rn = new Rendezvous();

// start the Producers and Consumers

// (they have self-starting threads) Producer[] p = new Producer[numProducers]; Consumer[] c = new Consumer[numConsumers]; for (int i = 0; i < numProducers; i++)

p[i] = new Producer("PRODUCER"+i, synchronous, pNap\*1000, rn);

for (int i = 0; i < numConsumers; i++)

c[i] = new Consumer("Consumer"+i, synchronous, cNap\*1000, rn); System.out.println("All threads started");

// let them run for a while try {

Thread.sleep(runTime\*1000); System.out.println("age=" + age()

+ ", time to terminate the threads and exit"); for (int i = 0; i < numProducers; i++)

p[i].timeToQuit();

for (int i = 0; i < numConsumers; i++) c[i].timeToQuit();

Thread.sleep(1000);

for (int i = 0; i < numProducers; i++) p[i].pauseTilDone();

for (int i = 0; i < numConsumers; i++) c[i].pauseTilDone();

} catch (InterruptedException e) { /\* ignored \*/ } System.out.println("age=" + age()

+ ", all threads are done"); System.exit(0);

}

}

# Sample run of pcre.java

% javac pcre.java

% java ProducersConsumers yes 1 1 2 2 6 ProducersConsumers:

synchronous=true, numProducers=1, numConsumers=1 pNap=2, cNap=2, runTime=6

age=45, PRODUCER0 napping for 185 ms age=64, Consumer0 napping for 1202 ms All threads started

age=244, PRODUCER0 produced item 0.15382515179890965 age=1286, Consumer0 wants to consume

age=1289, PRODUCER0, reply= Consumer0 got it! age=1290, PRODUCER0 sent item 0.15382515179890965

age=1292, PRODUCER0 napping for 745 ms

age=1292, Consumer0 received item 0.15382515179890965 age=1294, Consumer0 napping for 816 ms

age=2046, PRODUCER0 produced item 0.3602103173386145 age=2123, Consumer0 wants to consume

age=2124, PRODUCER0, reply= Consumer0 got it! age=2125, PRODUCER0 sent item 0.3602103173386145

age=2126, PRODUCER0 napping for 1954 ms

age=2127, Consumer0 received item 0.3602103173386145 age=2128, Consumer0 napping for 313 ms

age=2453, Consumer0 wants to consume

age=4096, PRODUCER0 produced item 0.9710857065446037 age=4098, PRODUCER0, reply= Consumer0 got it! age=4098, PRODUCER0 sent item 0.9710857065446037

age=4099, PRODUCER0 napping for 961 ms

age=4100, Consumer0 received item 0.9710857065446037 age=4101, Consumer0 napping for 1231 ms

age=5075, PRODUCER0 produced item 0.5231916912410289 age=5343, Consumer0 wants to consume

age=5344, PRODUCER0, reply= Consumer0 got it! age=5345, PRODUCER0 sent item 0.5231916912410289

age=5346, PRODUCER0 napping for 113 ms

age=5347, Consumer0 received item 0.5231916912410289

age=5348, Consumer0 napping for 661 ms

age=5473, PRODUCER0 produced item 0.5591064157961917 age=6024, Consumer0 wants to consume

age=6025, PRODUCER0, reply= Consumer0 got it! age=6025, PRODUCER0 sent item 0.5591064157961917

age=6026, PRODUCER0 napping for 153 ms

age=6027, Consumer0 received item 0.5591064157961917 age=6028, Consumer0 napping for 721 ms

age=6074, time to terminate the threads and exit age=6077, Consumer0 interrupted from sleep age=6094, PRODUCER0 interrupted from sleep age=7084, all threads are done

% java ProducersConsumers no1 1 2 2 6 ProducersConsumers:

synchronous=false, numProducers=1, numConsumers=1 pNap=2, cNap=2, runTime=6

age=47, PRODUCER0 napping for 810 ms age=66, Consumer0 napping for 48 ms All threads started

age=126, Consumer0 wants to consume

age=875, PRODUCER0 produced item 0.7568324366583216 age=893, PRODUCER0 sent item 0.7568324366583216

age=894, PRODUCER0 napping for 308 ms

age=896, Consumer0 received item 0.7568324366583216 age=898, Consumer0 napping for 290 ms

age=1195, Consumer0 wants to consume

age=1206, PRODUCER0 produced item 0.5312913550562778 age=1207, PRODUCER0 sent item 0.5312913550562778

age=1208, PRODUCER0 napping for 1326 ms

age=1208, Consumer0 received item 0.5312913550562778 age=1209, Consumer0 napping for 1840 ms

age=2548, PRODUCER0 produced item 0.034237591810933 age=2550, PRODUCER0 sent item 0.034237591810933

age=2551, PRODUCER0 napping for 989 ms age=3058, Consumer0 wants to consume

age=3059, Consumer0 received item 0.034237591810933 age=3060, Consumer0 napping for 1558 ms

age=3546, PRODUCER0 produced item 0.6571437055152907 age=3547, PRODUCER0 sent item 0.6571437055152907

age=3548, PRODUCER0 napping for 1848 ms age=4627, Consumer0 wants to consume

age=4628, Consumer0 received item 0.6571437055152907 age=4629, Consumer0 napping for 1445 ms

age=5406, PRODUCER0 produced item 0.9481282378191348 age=5407, PRODUCER0 sent item 0.9481282378191348

age=5408, PRODUCER0 napping for 618 ms

age=6036, PRODUCER0 produced item 0.626005963354672 age=6037, PRODUCER0 sent item 0.626005963354672

age=6038, PRODUCER0 napping for 47 ms

age=6076, time to terminate the threads and exit age=6097, PRODUCER0 interrupted from sleep age=6098, Consumer0 interrupted from sleep age=7086, all threads are done

% java ProducersConsumers yes 1 5 2 2 6 ProducersConsumers:

synchronous=true, numProducers=1, numConsumers=5 pNap=2, cNap=2, runTime=6

age=43, PRODUCER0 napping for 1429 ms age=62, Consumer0 napping for 388 ms age=66, Consumer1 napping for 657 ms age=68, Consumer2 napping for 1679 ms age=70, Consumer3 napping for 495 ms All threads started

age=73, Consumer4 napping for 1088 ms age=462, Consumer0 wants to consume

age=571, Consumer3 wants to consume age=731, Consumer1 wants to consume age=1174, Consumer4 wants to consume

age=1482, PRODUCER0 produced item 0.5470149827381032 age=1501, PRODUCER0, reply= Consumer0 got it! age=1503, PRODUCER0 sent item 0.5470149827381032

age=1504, PRODUCER0 napping for 847 ms

age=1505, Consumer0 received item 0.5470149827381032 age=1506, Consumer0 napping for 4 ms

age=1507, Consumer0 wants to consume age=1752, Consumer2 wants to consume

age=2364, PRODUCER0 produced item 0.39132362870008386 age=2366, PRODUCER0, reply= Consumer0 got it! age=2366, PRODUCER0 sent item 0.39132362870008386

age=2368, PRODUCER0 napping for 729 ms

age=2368, Consumer0 received item 0.39132362870008386 age=2370, Consumer0 napping for 1825 ms

age=3124, PRODUCER0 produced item 0.9201218658294597 age=3126, PRODUCER0, reply= Consumer1 got it! age=3127, PRODUCER0 sent item 0.9201218658294597

age=3128, PRODUCER0 napping for 1292 ms

age=3129, Consumer1 received item 0.9201218658294597 age=3130, Consumer1 napping for 552 ms

age=3692, Consumer1 wants to consume age=4203, Consumer0 wants to consume

age=4432, PRODUCER0 produced item 0.1103752768928924 age=4433, PRODUCER0, reply= Consumer0 got it! age=4434, PRODUCER0 sent item 0.1103752768928924

age=4435, PRODUCER0 napping for 887 ms

age=4436, Consumer0 received item 0.1103752768928924 age=4437, Consumer0 napping for 947 ms

age=5332, PRODUCER0 produced item 0.615086398663721 age=5334, PRODUCER0, reply= Consumer1 got it! age=5334, PRODUCER0 sent item 0.615086398663721

age=5335, PRODUCER0 napping for 1651 ms

age=5336, Consumer1 received item 0.615086398663721 age=5337, Consumer1 napping for 801 ms

age=5392, Consumer0 wants to consume

age=6082, time to terminate the threads and exit age=6085, Consumer0 interrupted from receive age=6087, Consumer1 interrupted from sleep age=6088, Consumer3 interrupted from receive age=6089, Consumer2 interrupted from receive age=6091, Consumer4 interrupted from receive age=6102, PRODUCER0 interrupted from sleep age=7092, all threads are done

% java ProducersConsumers no1 5 2 2 6 ProducersConsumers:

synchronous=false, numProducers=1, numConsumers=5 pNap=2, cNap=2, runTime=6

age=48, PRODUCER0 napping for 295 ms age=77, Consumer0 napping for 1571 ms age=98, Consumer1 napping for 162 ms age=100, Consumer2 napping for 160 ms age=102, Consumer3 napping for 39 ms All threads started

age=104, Consumer4 napping for 848 ms age=147, Consumer3 wants to consume age=266, Consumer2 wants to consume age=276, Consumer1 wants to consume

age=357, PRODUCER0 produced item 0.581001917177535 age=374, PRODUCER0 sent item 0.581001917177535

age=376, PRODUCER0 napping for 1586 ms

age=378, Consumer1 received item 0.581001917177535 age=379, Consumer1 napping for 596 ms

age=959, Consumer4 wants to consume age=986, Consumer1 wants to consume age=1667, Consumer0 wants to consume

age=1969, PRODUCER0 produced item 0.5306153831037835 age=1970, PRODUCER0 sent item 0.5306153831037835

age=1971, PRODUCER0 napping for 1347 ms

age=1972, Consumer0 received item 0.5306153831037835 age=1973, Consumer0 napping for 636 ms

age=2617, Consumer0 wants to consume

age=3329, PRODUCER0 produced item 0.2682601940276499 age=3330, PRODUCER0 sent item 0.2682601940276499

age=3331, PRODUCER0 napping for 241 ms

age=3332, Consumer0 received item 0.2682601940276499 age=3333, Consumer0 napping for 400 ms

age=3587, PRODUCER0 produced item 0.47542284179688665 age=3588, PRODUCER0 sent item 0.47542284179688665

age=3589, PRODUCER0 napping for 371 ms

age=3590, Consumer3 received item 0.47542284179688665 age=3591, Consumer3 napping for 497 ms

age=3736, Consumer0 wants to consume

age=3978, PRODUCER0 produced item 0.1426846229950136 age=3979, PRODUCER0 sent item 0.1426846229950136

age=3980, PRODUCER0 napping for 539 ms

age=3981, Consumer0 received item 0.1426846229950136 age=3982, Consumer0 napping for 1760 ms

age=4096, Consumer3 wants to consume

age=4527, PRODUCER0 produced item 0.8579208083111102 age=4528, PRODUCER0 sent item 0.8579208083111102

age=4529, PRODUCER0 napping for 891 ms

age=4530, Consumer2 received item 0.8579208083111102 age=4531, Consumer2 napping for 744 ms

age=5287, Consumer2 wants to consume

age=5437, PRODUCER0 produced item 0.7190220228546758 age=5438, PRODUCER0 sent item 0.7190220228546758

age=5439, PRODUCER0 napping for 1192 ms

age=5440, Consumer1 received item 0.7190220228546758 age=5441, Consumer1 napping for 97 ms

age=5546, Consumer1 wants to consume age=5746, Consumer0 wants to consume

age=6107, time to terminate the threads and exit age=6110, Consumer3 interrupted from receive age=6112, Consumer2 interrupted from receive age=6113, Consumer1 interrupted from receive age=6114, Consumer4 interrupted from receive age=6127, PRODUCER0 interrupted from sleep age=6128, Consumer0 interrupted from receive age=7117, all threads are done

% java ProducersConsumers yes 5 1 2 2 6 ProducersConsumers:

synchronous=true, numProducers=5, numConsumers=1 pNap=2, cNap=2, runTime=6

age=46, PRODUCER0 napping for 904 ms age=66, PRODUCER1 napping for 257 ms age=67, PRODUCER2 napping for 1740 ms age=70, PRODUCER3 napping for 1405 ms age=73, PRODUCER4 napping for 670 ms All threads started

age=76, Consumer0 napping for 1620 ms

age=335, PRODUCER1 produced item 0.17631033059969203 age=745, PRODUCER4 produced item 0.5982192991443873 age=967, PRODUCER0 produced item 0.346069281928469 age=1485, PRODUCER3 produced item 0.1777369982527307 age=1705, Consumer0 wants to consume

age=1708, PRODUCER1, reply= Consumer0 got it! age=1709, PRODUCER1 sent item 0.17631033059969203

age=1711, PRODUCER1 napping for 1185 ms

age=1712, Consumer0 received item 0.17631033059969203 age=1713, Consumer0 napping for 1381 ms

age=1815, PRODUCER2 produced item 0.11184681708602617 age=2907, PRODUCER1 produced item 0.03828430977553221 age=3105, Consumer0 wants to consume

age=3106, PRODUCER1, reply= Consumer0 got it! age=3107, PRODUCER1 sent item 0.03828430977553221

age=3108, PRODUCER1 napping for 1362 ms

age=3108, Consumer0 received item 0.03828430977553221 age=3109, Consumer0 napping for 1281 ms

age=4407, Consumer0 wants to consume age=4408, PRODUCER2, reply= Consumer0 got it!

age=4409, PRODUCER2 sent item 0.11184681708602617

age=4410, PRODUCER2 napping for 654 ms

age=4411, Consumer0 received item 0.11184681708602617 age=4412, Consumer0 napping for 1190 ms

age=4485, PRODUCER1 produced item 0.08158951384451485 age=5076, PRODUCER2 produced item 0.0983728197982997 age=5605, Consumer0 wants to consume

age=5606, PRODUCER1, reply= Consumer0 got it! age=5606, PRODUCER1 sent item 0.08158951384451485

age=5608, PRODUCER1 napping for 969 ms

age=5608, Consumer0 received item 0.08158951384451485 age=5609, Consumer0 napping for 1542 ms

age=6085, time to terminate the threads and exit age=6089, PRODUCER4 interrupted from send age=6091, PRODUCER3 interrupted from send age=6092, PRODUCER2 interrupted from send age=6093, PRODUCER1 interrupted from sleep age=6094, Consumer0 interrupted from sleep age=6105, PRODUCER0 interrupted from send age=7095, all threads are done

% java ProducersConsumers no5 1 2 2 6 ProducersConsumers:

synchronous=false, numProducers=5, numConsumers=1 pNap=2, cNap=2, runTime=6

age=49, PRODUCER0 napping for 1009 ms age=68, PRODUCER1 napping for 499 ms age=89, PRODUCER2 napping for 570 ms age=91, PRODUCER3 napping for 676 ms age=94, PRODUCER4 napping for 1718 ms All threads started

age=97, Consumer0 napping for 1153 ms

age=578, PRODUCER1 produced item 0.654109824432186 age=596, PRODUCER1 sent item 0.654109824432186

age=597, PRODUCER1 napping for 1240 ms

age=667, PRODUCER2 produced item 0.18068514475674902 age=669, PRODUCER2 sent item 0.18068514475674902

age=670, PRODUCER2 napping for 589 ms

age=778, PRODUCER3 produced item 0.4102712629046613 age=779, PRODUCER3 sent item 0.4102712629046613

age=780, PRODUCER3 napping for 1802 ms

age=1070, PRODUCER0 produced item 0.5091696195916051 age=1071, PRODUCER0 sent item 0.5091696195916051

age=1072, PRODUCER0 napping for 1319 ms

age=1268, PRODUCER2 produced item 0.11994991872131833 age=1269, PRODUCER2 sent item 0.11994991872131833

age=1270, PRODUCER2 napping for 1459 ms age=1271, Consumer0 wants to consume

age=1274, Consumer0 received item 0.11994991872131833 age=1275, Consumer0 napping for 983 ms

age=1818, PRODUCER4 produced item 0.3179813502495561 age=1819, PRODUCER4 sent item 0.3179813502495561

age=1820, PRODUCER4 napping for 904 ms

age=1848, PRODUCER1 produced item 0.6786836744500554 age=1849, PRODUCER1 sent item 0.6786836744500554

age=1850, PRODUCER1 napping for 584 ms age=2270, Consumer0 wants to consume

age=2271, Consumer0 received item 0.18068514475674902 age=2272, Consumer0 napping for 1020 ms

age=2398, PRODUCER0 produced item 0.6765052327934625 age=2399, PRODUCER0 sent item 0.6765052327934625

age=2400, PRODUCER0 napping for 112 ms

age=2448, PRODUCER1 produced item 0.6087623006574409 age=2449, PRODUCER1 sent item 0.6087623006574409

age=2450, PRODUCER1 napping for 1110 ms

age=2527, PRODUCER0 produced item 0.2943617678806405 age=2529, PRODUCER0 sent item 0.2943617678806405

age=2530, PRODUCER0 napping for 52 ms

age=2597, PRODUCER0 produced item 0.43951489579213454 age=2599, PRODUCER0 sent item 0.43951489579213454

age=2600, PRODUCER0 napping for 1010 ms

age=2601, PRODUCER3 produced item 0.6752658089401261 age=2602, PRODUCER3 sent item 0.6752658089401261

age=2603, PRODUCER3 napping for 1878 ms

age=2738, PRODUCER2 produced item 0.3501856253437148 age=2739, PRODUCER2 sent item 0.3501856253437148

age=2740, PRODUCER2 napping for 163 ms

age=2741, PRODUCER4 produced item 0.08523336769878354 age=2742, PRODUCER4 sent item 0.08523336769878354

age=2743, PRODUCER4 napping for 1810 ms

age=2920, PRODUCER2 produced item 0.13301474356217313 age=2921, PRODUCER2 sent item 0.13301474356217313

age=2922, PRODUCER2 napping for 1163 ms age=3355, Consumer0 wants to consume

age=3357, Consumer0 received item 0.08523336769878354 age=3358, Consumer0 napping for 1003 ms

age=3568, PRODUCER1 produced item 0.7517292199496083 age=3569, PRODUCER1 sent item 0.7517292199496083

age=3570, PRODUCER1 napping for 1800 ms

age=3618, PRODUCER0 produced item 0.268743465819824 age=3619, PRODUCER0 sent item 0.268743465819824

age=3620, PRODUCER0 napping for 337 ms

age=3969, PRODUCER0 produced item 0.32319739222576216 age=3970, PRODUCER0 sent item 0.32319739222576216

age=3971, PRODUCER0 napping for 1715 ms

age=4098, PRODUCER2 produced item 0.8618141010061056 age=4099, PRODUCER2 sent item 0.8618141010061056

age=4100, PRODUCER2 napping for 1896 ms age=4377, Consumer0 wants to consume

age=4379, Consumer0 received item 0.13301474356217313 age=4380, Consumer0 napping for 1435 ms

age=4488, PRODUCER3 produced item 0.23912464069393813 age=4489, PRODUCER3 sent item 0.23912464069393813

age=4490, PRODUCER3 napping for 263 ms

age=4558, PRODUCER4 produced item 0.10655946770202618 age=4559, PRODUCER4 sent item 0.10655946770202618

age=4560, PRODUCER4 napping for 924 ms

age=4767, PRODUCER3 produced item 0.12318563460348397 age=4769, PRODUCER3 sent item 0.12318563460348397

age=4770, PRODUCER3 napping for 591 ms

age=5378, PRODUCER1 produced item 0.3722613130125716 age=5379, PRODUCER1 sent item 0.3722613130125716

age=5380, PRODUCER1 napping for 1625 ms

age=5381, PRODUCER3 produced item 0.16025856103329428 age=5382, PRODUCER3 sent item 0.16025856103329428

age=5383, PRODUCER3 napping for 1701 ms

age=5498, PRODUCER4 produced item 0.733038334914754 age=5499, PRODUCER4 sent item 0.733038334914754

age=5500, PRODUCER4 napping for 688 ms

age=5698, PRODUCER0 produced item 0.5541625041131112 age=5699, PRODUCER0 sent item 0.5541625041131112

age=5700, PRODUCER0 napping for 1411 ms age=5827, Consumer0 wants to consume

age=5830, Consumer0 received item 0.10655946770202618 age=5831, Consumer0 napping for 1386 ms

age=6008, PRODUCER2 produced item 0.7258819670653431 age=6009, PRODUCER2 sent item 0.7258819670653431

age=6010, PRODUCER2 napping for 1982 ms age=6098, time to terminate the threads and exit age=6101, PRODUCER4 interrupted from sleep age=6103, PRODUCER3 interrupted from sleep age=6104, PRODUCER2 interrupted from sleep age=6105, PRODUCER1 interrupted from sleep age=6106, Consumer0 interrupted from sleep age=6118, PRODUCER0 interrupted from sleep age=7108, all threads are done

% java ProducersConsumers yes 5 5 2 2 6 ProducersConsumers:

synchronous=true, numProducers=5, numConsumers=5 pNap=2, cNap=2, runTime=6

age=42, PRODUCER0 napping for 1664 ms age=61, PRODUCER1 napping for 38 ms age=64, PRODUCER2 napping for 1016 ms age=65, PRODUCER3 napping for 1829 ms age=69, PRODUCER4 napping for 60 ms age=72, Consumer0 napping for 56 ms age=74, Consumer1 napping for 1142 ms age=76, Consumer2 napping for 1969 ms age=78, Consumer3 napping for 1500 ms All threads started

age=81, Consumer4 napping for 25 ms

age=111, PRODUCER1 produced item 0.9185428067197055 age=129, Consumer4 wants to consume

age=292, PRODUCER1, reply= Consumer4 got it! age=294, PRODUCER1 sent item 0.9185428067197055

age=295, PRODUCER1 napping for 914 ms

age=296, Consumer4 received item 0.9185428067197055 age=297, Consumer4 napping for 766 ms

age=131, PRODUCER4 produced item 0.08777957250723711 age=190, Consumer0 wants to consume

age=299, Consumer0 received item 0.08777957250723711 age=300, Consumer0 napping for 620 ms

age=301, PRODUCER4, reply= Consumer0 got it! age=302, PRODUCER4 sent item 0.08777957250723711

age=303, PRODUCER4 napping for 1479 ms age=933, Consumer0 wants to consume age=1070, Consumer4 wants to consume

age=1091, PRODUCER2 produced item 0.8940799526535598 age=1092, PRODUCER2, reply= Consumer4 got it! age=1093, PRODUCER2 sent item 0.8940799526535598

age=1094, PRODUCER2 napping for 87 ms

age=1095, Consumer4 received item 0.8940799526535598 age=1096, Consumer4 napping for 1535 ms

age=1190, PRODUCER2 produced item 0.10081751591092203 age=1192, PRODUCER2, reply= Consumer0 got it! age=1193, PRODUCER2 sent item 0.10081751591092203

age=1194, PRODUCER2 napping for 1821 ms

age=1195, Consumer0 received item 0.10081751591092203 age=1196, Consumer0 napping for 1396 ms

age=1221, PRODUCER1 produced item 0.3663133766503859 age=1230, Consumer1 wants to consume

age=1231, PRODUCER1, reply= Consumer1 got it! age=1232, PRODUCER1 sent item 0.3663133766503859

age=1233, PRODUCER1 napping for 918 ms

age=1234, Consumer1 received item 0.3663133766503859 age=1235, Consumer1 napping for 503 ms

age=1581, Consumer3 wants to consume

age=1721, PRODUCER0 produced item 0.16926696419542775 age=1722, PRODUCER0, reply= Consumer3 got it! age=1723, PRODUCER0 sent item 0.16926696419542775

age=1724, PRODUCER0 napping for 1393 ms

age=1725, Consumer3 received item 0.16926696419542775 age=1726, Consumer3 napping for 1382 ms

age=1750, Consumer1 wants to consume

age=1791, PRODUCER4 produced item 0.24476649246179505 age=1792, Consumer1 received item 0.24476649246179505 age=1794, Consumer1 napping for 484 ms

age=1796, PRODUCER4, reply= Consumer1 got it! age=1798, PRODUCER4 sent item 0.24476649246179505

age=1799, PRODUCER4 napping for 220 ms

age=1903, PRODUCER3 produced item 0.800190863174061 age=2021, PRODUCER4 produced item 0.8202427064995859 age=2051, Consumer2 wants to consume

age=2052, PRODUCER3, reply= Consumer2 got it! age=2053, PRODUCER3 sent item 0.800190863174061

age=2054, PRODUCER3 napping for 1756 ms

age=2055, Consumer2 received item 0.800190863174061 age=2056, Consumer2 napping for 638 ms

age=2161, PRODUCER1 produced item 0.9924100353971543 age=2290, Consumer1 wants to consume

age=2291, Consumer1 received item 0.8202427064995859 age=2292, Consumer1 napping for 1935 ms

age=2293, PRODUCER4, reply= Consumer1 got it! age=2294, PRODUCER4 sent item 0.8202427064995859

age=2295, PRODUCER4 napping for 816 ms age=2600, Consumer0 wants to consume age=2601, PRODUCER1, reply= Consumer0 got it!

age=2602, PRODUCER1 sent item 0.9924100353971543

age=2603, PRODUCER1 napping for 52 ms

age=2604, Consumer0 received item 0.9924100353971543 age=2605, Consumer0 napping for 203 ms

age=2640, Consumer4 wants to consume

age=2670, PRODUCER1 produced item 0.5720839002063113 age=2672, PRODUCER1, reply= Consumer4 got it! age=2673, PRODUCER1 sent item 0.5720839002063113

age=2674, PRODUCER1 napping for 830 ms

age=2674, Consumer4 received item 0.5720839002063113 age=2676, Consumer4 napping for 135 ms

age=2700, Consumer2 wants to consume age=2820, Consumer4 wants to consume age=2821, Consumer0 wants to consume

age=3033, PRODUCER2 produced item 0.38032840213175745 age=3034, PRODUCER2, reply= Consumer2 got it! age=3035, PRODUCER2 sent item 0.38032840213175745

age=3036, PRODUCER2 napping for 1095 ms

age=3037, Consumer2 received item 0.38032840213175745 age=3038, Consumer2 napping for 1230 ms

age=3120, Consumer3 wants to consume

age=3121, PRODUCER4 produced item 0.9604224841343072 age=3123, Consumer3 received item 0.9604224841343072 age=3124, Consumer3 napping for 1633 ms

age=3125, PRODUCER4, reply= Consumer3 got it! age=3125, PRODUCER4 sent item 0.9604224841343072

age=3126, PRODUCER4 napping for 608 ms

age=3131, PRODUCER0 produced item 0.9650427611915405 age=3132, PRODUCER0, reply= Consumer4 got it! age=3133, PRODUCER0 sent item 0.9650427611915405

age=3134, PRODUCER0 napping for 1713 ms

age=3135, Consumer4 received item 0.9650427611915405 age=3136, Consumer4 napping for 307 ms

age=3450, Consumer4 wants to consume

age=3511, PRODUCER1 produced item 0.918506532764693 age=3512, PRODUCER1, reply= Consumer4 got it! age=3513, PRODUCER1 sent item 0.918506532764693

age=3514, PRODUCER1 napping for 1238 ms

age=3515, Consumer4 received item 0.918506532764693 age=3516, Consumer4 napping for 1264 ms

age=3740, PRODUCER4 produced item 0.754027144422452 age=3742, PRODUCER4, reply= Consumer0 got it! age=3743, PRODUCER4 sent item 0.754027144422452

age=3744, PRODUCER4 napping for 1578 ms

age=3745, Consumer0 received item 0.754027144422452 age=3746, Consumer0 napping for 208 ms

age=3821, PRODUCER3 produced item 0.27122994909004716 age=3962, Consumer0 wants to consume

age=3963, PRODUCER3, reply= Consumer0 got it! age=3964, PRODUCER3 sent item 0.27122994909004716

age=3965, PRODUCER3 napping for 579 ms

age=3966, Consumer0 received item 0.27122994909004716 age=3967, Consumer0 napping for 372 ms

age=4141, PRODUCER2 produced item 0.27589255387330824 age=4240, Consumer1 wants to consume

age=4241, PRODUCER2, reply= Consumer1 got it! age=4242, PRODUCER2 sent item 0.27589255387330824

age=4243, PRODUCER2 napping for 1469 ms

age=4244, Consumer1 received item 0.27589255387330824 age=4245, Consumer1 napping for 1952 ms

age=4270, Consumer2 wants to consume age=4350, Consumer0 wants to consume

age=4550, PRODUCER3 produced item 0.8619194399545967 age=4552, PRODUCER3, reply= Consumer2 got it! age=4553, PRODUCER3 sent item 0.8619194399545967

age=4554, PRODUCER3 napping for 398 ms

age=4554, Consumer2 received item 0.8619194399545967 age=4556, Consumer2 napping for 472 ms

age=4761, PRODUCER1 produced item 0.7701099520047171 age=4762, PRODUCER1, reply= Consumer0 got it! age=4763, PRODUCER1 sent item 0.7701099520047171

age=4764, PRODUCER1 napping for 1625 ms

age=4765, Consumer0 received item 0.7701099520047171 age=4766, Consumer0 napping for 129 ms

age=4771, Consumer3 wants to consume age=4790, Consumer4 wants to consume

age=4861, PRODUCER0 produced item 0.2228435588271499 age=4862, PRODUCER0, reply= Consumer3 got it! age=4863, PRODUCER0 sent item 0.2228435588271499

age=4864, PRODUCER0 napping for 951 ms

age=4864, Consumer3 received item 0.2228435588271499 age=4866, Consumer3 napping for 675 ms

age=4900, Consumer0 wants to consume

age=4961, PRODUCER3 produced item 0.6894975962370399 age=4962, PRODUCER3, reply= Consumer4 got it! age=4963, PRODUCER3 sent item 0.6894975962370399

age=4964, PRODUCER3 napping for 743 ms

age=4965, Consumer4 received item 0.6894975962370399 age=4966, Consumer4 napping for 403 ms

age=5041, Consumer2 wants to consume

age=5331, PRODUCER4 produced item 0.8065414987883718 age=5332, Consumer2 received item 0.8065414987883718 age=5333, Consumer2 napping for 1824 ms

age=5334, PRODUCER4, reply= Consumer2 got it! age=5334, PRODUCER4 sent item 0.8065414987883718

age=5336, PRODUCER4 napping for 1217 ms

age=5380, Consumer4 wants to consume age=5551, Consumer3 wants to consume

age=5721, PRODUCER2 produced item 0.9920886064917681 age=5722, PRODUCER3 produced item 0.903554039796397 age=5723, PRODUCER3, reply= Consumer3 got it! age=5724, PRODUCER3 sent item 0.903554039796397

age=5725, PRODUCER3 napping for 828 ms

age=5726, Consumer3 received item 0.903554039796397 age=5727, Consumer3 napping for 997 ms

age=5728, PRODUCER2, reply= Consumer4 got it! age=5729, PRODUCER2 sent item 0.9920886064917681

age=5730, PRODUCER2 napping for 1454 ms

age=5731, Consumer4 received item 0.9920886064917681 age=5732, Consumer4 napping for 1344 ms

age=5831, PRODUCER0 produced item 0.13174758677755616 age=5832, PRODUCER0, reply= Consumer0 got it! age=5833, PRODUCER0 sent item 0.13174758677755616

age=5834, PRODUCER0 napping for 689 ms

age=5835, Consumer0 received item 0.13174758677755616 age=5836, Consumer0 napping for 1170 ms

age=6091, time to terminate the threads and exit age=6094, PRODUCER3 interrupted from sleep age=6096, PRODUCER1 interrupted from sleep age=6097, PRODUCER2 interrupted from sleep age=6098, Consumer1 interrupted from sleep age=6099, Consumer2 interrupted from sleep age=6100, Consumer3 interrupted from sleep age=6101, Consumer4 interrupted from sleep age=6103, PRODUCER4 interrupted from sleep age=6104, Consumer0 interrupted from sleep age=6111, PRODUCER0 interrupted from sleep age=7101, all threads are done

% java ProducersConsumers no5 5 2 2 6 ProducersConsumers:

synchronous=false, numProducers=5, numConsumers=5 pNap=2, cNap=2, runTime=6

age=44, PRODUCER0 napping for 1548 ms age=63, PRODUCER1 napping for 677 ms age=66, PRODUCER2 napping for 1441 ms age=68, PRODUCER3 napping for 1283 ms age=71, PRODUCER4 napping for 435 ms age=74, Consumer0 napping for 1925 ms age=76, Consumer1 napping for 1681 ms age=78, Consumer2 napping for 1632 ms age=81, Consumer3 napping for 1953 ms All threads started

age=83, Consumer4 napping for 1600 ms

age=513, PRODUCER4 produced item 0.3410400811142391 age=530, PRODUCER4 sent item 0.3410400811142391

age=532, PRODUCER4 napping for 1 ms

age=532, PRODUCER4 produced item 0.9047856169302597 age=533, PRODUCER4 sent item 0.9047856169302597

age=534, PRODUCER4 napping for 1025 ms

age=753, PRODUCER1 produced item 0.8843759516570732 age=754, PRODUCER1 sent item 0.8843759516570732

age=755, PRODUCER1 napping for 357 ms

age=1125, PRODUCER1 produced item 0.19197557779325958 age=1126, PRODUCER1 sent item 0.19197557779325958

age=1127, PRODUCER1 napping for 491 ms

age=1363, PRODUCER3 produced item 0.830373247233264 age=1364, PRODUCER3 sent item 0.830373247233264

age=1365, PRODUCER3 napping for 901 ms

age=1523, PRODUCER2 produced item 0.8098838675941346 age=1524, PRODUCER2 sent item 0.8098838675941346

age=1525, PRODUCER2 napping for 436 ms

age=1573, PRODUCER4 produced item 0.6027106304058345 age=1574, PRODUCER4 sent item 0.6027106304058345

age=1575, PRODUCER4 napping for 319 ms

age=1603, PRODUCER0 produced item 0.4292486419460507 age=1604, PRODUCER0 sent item 0.4292486419460507

age=1605, PRODUCER0 napping for 325 ms

age=1632, PRODUCER1 produced item 0.5872721123774989 age=1634, PRODUCER1 sent item 0.5872721123774989

age=1635, PRODUCER1 napping for 1474 ms age=1693, Consumer4 wants to consume

age=1695, Consumer4 received item 0.19197557779325958 age=1697, Consumer4 napping for 1939 ms

age=1722, Consumer2 wants to consume

age=1723, Consumer2 received item 0.3410400811142391 age=1724, Consumer2 napping for 608 ms

age=1772, Consumer1 wants to consume

age=1773, Consumer1 received item 0.4292486419460507 age=1774, Consumer1 napping for 148 ms

age=1905, PRODUCER4 produced item 0.15349615717846132 age=1906, PRODUCER4 sent item 0.15349615717846132

age=1907, PRODUCER4 napping for 974 ms age=1932, Consumer1 wants to consume

age=1933, Consumer1 received item 0.15349615717846132 age=1935, Consumer1 napping for 734 ms

age=1943, PRODUCER0 produced item 0.6143545981931535 age=1944, PRODUCER0 sent item 0.6143545981931535

age=1945, PRODUCER0 napping for 214 ms

age=1973, PRODUCER2 produced item 0.5517125342572194 age=1974, PRODUCER2 sent item 0.5517125342572194

age=1975, PRODUCER2 napping for 105 ms age=2012, Consumer0 wants to consume

age=2014, Consumer0 received item 0.5517125342572194 age=2015, Consumer0 napping for 896 ms

age=2042, Consumer3 wants to consume

age=2043, Consumer3 received item 0.5872721123774989 age=2044, Consumer3 napping for 1964 ms

age=2093, PRODUCER2 produced item 0.6032968491632188 age=2094, PRODUCER2 sent item 0.6032968491632188

age=2095, PRODUCER2 napping for 1052 ms

age=2173, PRODUCER0 produced item 0.306508087515337 age=2174, PRODUCER0 sent item 0.306508087515337

age=2175, PRODUCER0 napping for 346 ms

age=2283, PRODUCER3 produced item 0.039727663768907906 age=2284, PRODUCER3 sent item 0.039727663768907906

age=2285, PRODUCER3 napping for 1444 ms age=2342, Consumer2 wants to consume

age=2344, Consumer2 received item 0.039727663768907906 age=2345, Consumer2 napping for 32 ms

age=2392, Consumer2 wants to consume

age=2393, Consumer2 received item 0.306508087515337 age=2395, Consumer2 napping for 1172 ms

age=2533, PRODUCER0 produced item 0.9501617999455061 age=2534, PRODUCER0 sent item 0.9501617999455061

age=2535, PRODUCER0 napping for 1394 ms age=2682, Consumer1 wants to consume

age=2683, Consumer1 received item 0.6027106304058345 age=2684, Consumer1 napping for 1554 ms

age=2902, PRODUCER4 produced item 0.9917188318473676 age=2903, PRODUCER4 sent item 0.9917188318473676

age=2904, PRODUCER4 napping for 940 ms age=2922, Consumer0 wants to consume

age=2923, Consumer0 received item 0.6032968491632188 age=2924, Consumer0 napping for 1654 ms

age=3123, PRODUCER1 produced item 0.9290907832359444 age=3124, PRODUCER1 sent item 0.9290907832359444

age=3125, PRODUCER1 napping for 897 ms

age=3163, PRODUCER2 produced item 0.7092244510592798 age=3164, PRODUCER2 sent item 0.7092244510592798

age=3165, PRODUCER2 napping for 775 ms age=3583, Consumer2 wants to consume

age=3584, Consumer2 received item 0.6143545981931535 age=3585, Consumer2 napping for 544 ms

age=3642, Consumer4 wants to consume

age=3644, Consumer4 received item 0.7092244510592798 age=3645, Consumer4 napping for 371 ms

age=3743, PRODUCER3 produced item 0.09280502991799322 age=3744, PRODUCER3 sent item 0.09280502991799322

age=3745, PRODUCER3 napping for 1858 ms

age=3853, PRODUCER4 produced item 0.7592324485274388 age=3854, PRODUCER4 sent item 0.7592324485274388

age=3855, PRODUCER4 napping for 366 ms

age=3943, PRODUCER0 produced item 0.7209106579261657 age=3944, PRODUCER0 sent item 0.7209106579261657

age=3946, PRODUCER0 napping for 1845 ms

age=3953, PRODUCER2 produced item 0.47272956301074043 age=3954, PRODUCER2 sent item 0.47272956301074043

age=3955, PRODUCER2 napping for 365 ms age=4022, Consumer3 wants to consume

age=4024, Consumer3 received item 0.09280502991799322 age=4025, Consumer3 napping for 142 ms

age=4033, PRODUCER1 produced item 0.3280222480542785 age=4034, PRODUCER1 sent item 0.3280222480542785

age=4035, PRODUCER1 napping for 617 ms age=4035, Consumer4 wants to consume

age=4037, Consumer4 received item 0.3280222480542785 age=4038, Consumer4 napping for 23 ms

age=4072, Consumer4 wants to consume

age=4074, Consumer4 received item 0.47272956301074043 age=4075, Consumer4 napping for 1886 ms

age=4142, Consumer2 wants to consume

age=4144, Consumer2 received item 0.7209106579261657 age=4145, Consumer2 napping for 245 ms

age=4182, Consumer3 wants to consume

age=4184, Consumer3 received item 0.7592324485274388 age=4185, Consumer3 napping for 1482 ms

age=4233, PRODUCER4 produced item 0.9186461316635379 age=4234, PRODUCER4 sent item 0.9186461316635379

age=4235, PRODUCER4 napping for 138 ms age=4252, Consumer1 wants to consume

age=4253, Consumer1 received item 0.8098838675941346 age=4254, Consumer1 napping for 601 ms

age=4333, PRODUCER2 produced item 0.35806399650517884 age=4334, PRODUCER2 sent item 0.35806399650517884

age=4335, PRODUCER2 napping for 166 ms

age=4383, PRODUCER4 produced item 0.2058488238208972 age=4384, PRODUCER4 sent item 0.2058488238208972

age=4385, PRODUCER4 napping for 1147 ms age=4402, Consumer2 wants to consume

age=4404, Consumer2 received item 0.2058488238208972 age=4405, Consumer2 napping for 1199 ms

age=4513, PRODUCER2 produced item 0.5743572081217496 age=4514, PRODUCER2 sent item 0.5743572081217496

age=4515, PRODUCER2 napping for 1721 ms age=4593, Consumer0 wants to consume

age=4594, Consumer0 received item 0.35806399650517884 age=4595, Consumer0 napping for 1554 ms

age=4663, PRODUCER1 produced item 0.2768196943978283 age=4664, PRODUCER1 sent item 0.2768196943978283

age=4665, PRODUCER1 napping for 1674 ms age=4872, Consumer1 wants to consume

age=4874, Consumer1 received item 0.2768196943978283 age=4875, Consumer1 napping for 17 ms

age=4902, Consumer1 wants to consume

age=4903, Consumer1 received item 0.5743572081217496 age=4905, Consumer1 napping for 8 ms

age=4923, Consumer1 wants to consume

age=4923, Consumer1 received item 0.830373247233264 age=4924, Consumer1 napping for 20 ms

age=4952, Consumer1 wants to consume

age=4953, Consumer1 received item 0.8843759516570732 age=4954, Consumer1 napping for 361 ms

age=5332, Consumer1 wants to consume

age=5333, Consumer1 received item 0.9047856169302597 age=5334, Consumer1 napping for 613 ms

age=5543, PRODUCER4 produced item 0.6109297009381629 age=5544, PRODUCER4 sent item 0.6109297009381629

age=5545, PRODUCER4 napping for 1325 ms

age=5613, PRODUCER3 produced item 0.874675354317203 age=5614, PRODUCER3 sent item 0.874675354317203

age=5615, PRODUCER3 napping for 664 ms age=5615, Consumer2 wants to consume

age=5616, Consumer2 received item 0.6109297009381629 age=5617, Consumer2 napping for 1072 ms

age=5682, Consumer3 wants to consume

age=5683, Consumer3 received item 0.874675354317203 age=5684, Consumer3 napping for 474 ms

age=5804, PRODUCER0 produced item 0.1782579961776114 age=5806, PRODUCER0 sent item 0.1782579961776114

age=5808, PRODUCER0 napping for 640 ms age=5963, Consumer1 wants to consume

age=5963, Consumer1 received item 0.1782579961776114 age=5965, Consumer1 napping for 1642 ms

age=5972, Consumer4 wants to consume

age=5973, Consumer4 received item 0.9186461316635379 age=5974, Consumer4 napping for 891 ms

age=6093, time to terminate the threads and exit age=6096, PRODUCER4 interrupted from sleep age=6098, Consumer0 interrupted from sleep age=6099, Consumer1 interrupted from sleep age=6100, Consumer4 interrupted from sleep age=6101, Consumer3 interrupted from sleep age=6103, Consumer2 interrupted from sleep age=6113, PRODUCER0 interrupted from sleep age=6114, PRODUCER1 interrupted from sleep age=6115, PRODUCER2 interrupted from sleep age=6116, PRODUCER3 interrupted from sleep age=7103, all threads are done

# Section 12. Remote Method Invocation (RMI) Some definitions

RMI is a Java package (java.rmi) used to make remote procedure calls.

### RMI allows a thread in one JVM to invoke a method in an object in another JVM that is perhaps on a different computer.

*Object serialization* is used to send an object from one JVM to another as an argument of a remote method invocation. This converts an object into a byte stream that is sent through a socket and converted into a copy of the object on the other end. A new thread is created in the remote object to execute the called method's code.

[Example Compute.java](#_bookmark70) on page 99 shows several clients accessing a remote server executing in a different JVM, which can be on a different physical machine. [Sample run of](#_bookmark71) [Compute.java server](#_bookmark71) on page 102 shows the sample server outpu[St; ample run of](#_bookmark72) [Compute.java clients](#_bookmark72) on page 105 shows the sample client output.

Notice that the sample output shows the clients' remote method invocations are interleaved -- that is, overlapping executions by new threads created in the server for each RMI. There are no race conditions or synchronization problems in this example because the clients are independent and do not share any data.

In the sample run, the clients all execute in one JVM and the server in another JVM. Both JVMs are on the same physical machine. If the clients are on a different physical machine, pass the name of the machine on which the server runs as a command-line argument when starting the clients.

RMI can be used by a thread in one JVM to send a message to or rendezvous with a thread in another JVM. A thread willing to receive or rendezvous registers an interface that other threads can use and implements the interface using a message passing or rendezvous object.

Background material on RMI

RMI, or remote method invocation, is the ability to make remote procedure calls. We use "remote procedure calls" to describe an extended rendezvous between two threads in different JVMs, perhaps on different physical machines.

Sun's RMI allows a thread in one JVM to invoke (call) a method in an object in another JVM that is perhaps on a different physical machine. A new thread is created in the other (remote) JVM to execute the called method.

The ComputeServer remote object implements a Compute interface containing a compute() method that a local Client can call, passing a Work object whose doWork() method the server calls. The client is using the remote server to have work performed on its behalf (adding vectors). Presumably the server is running on a computer architecture that can perform the work more efficiently. Parameters to the remote method and the method's return results, if any, are passed from one JVM to the other using object serialization over the

network.

User-written classes

Rendezvous client and server classes for RMI include:

* [Interface RemoteRendezvous.java](#_bookmark73) on page 107
* [Class RemoteRendezvousClient.java](#_bookmark74) on page 108 . Used by client or peer
* [Class RemoteRendezvousServer.java](#_bookmark75) on page 109 . Used by server or peer

Examples of RMI

Rendezvous [Example Transact.java](#_bookmark76) on page 110 -- Several clients access a remote server executing in a different JVM, which can be on a different physical machine. Server and client output is in [Sample run of Transact.java server](#_bookmark77) on page 113 an[Sd ample run of Transact.java](#_bookmark78) [clients](#_bookmark78) on page 114 .

Multiple clients transact (read and write operations) with a database on a remote server. The transactions are serialized to avoid race conditions on the shared database maintained by the server. Also, the server gives client number zero highest priority by always handling its transaction first among those waiting to be performed.

In the sample run, the clients all execute in one JVM and the server in another JVM. Both JVMs are on the same physical machine. If the clients are on a different physical machine, pass the name of the machine on which the server runs as a command-line argument when starting the clients.

Message passing [Example Ring.java](#_bookmark79) on page 115 -- Several peers are arranged in a circular ring. Each ring member executes in its own JVM and the ring members need not all be on the same physical machine. A single token object is passed around the ring from each member to its successor. These are the sample outputs ([Sample run of Ring.java ring](#_bookmark80) [member](#_bookmark80) 0 on page 119[S, ample run of Ring.java ring member](#_bookmark81) 1 on page 120 , an[Sd ample run](#_bookmark82) [of Ring.java ring member](#_bookmark82) 2 on page 121 ) in a three-member ring.

In the sample run, the three ring members execute in different JVMs, all on the same physical machine. If the JVMs are on different physical machines, give each ring member on its command line the machine name of its successor. Each physical machine running one or more ring member JVMs needs to be executing one instance of rmiregistry, started either manually or internally by one of the ring members on that machine.

The next several panels display the code described in this section. To view the code, click Next; or you can go directly to the next section, [Wrapup](#_bookmark83) on page 122 , and return to the code samples at another time.

Example Compute.java

import java.io.Serializable; import java.rmi.\*;

import java.rmi.server.UnicastRemoteObject;

import java.rmi.registry.\*;

public interface Compute extends Remote {

public static final String SERVER\_NAME = "ComputeServer"; public static final String SERVER\_MACHINE = "localhost"; public static final int SERVER\_PORT = 8989;

public static final int RUN\_TIME = 20; public abstract Work compute(Work w)

throws RemoteException, InterruptedException;

}

class Work extends Sugar implements Serializable { private final int N = 3;

private String name = null;

private double[] a = null, b = null, c = null; private boolean performed = false;

public Work(String name) { this.name = name;

a = new double[N]; b = new double[N]; c = new double[N]; for (int i = 0; i < N; i++) {

a[i] = random(-N, N); b[i] = random(-N, N);

}

}

public void doWork() throws InterruptedException {

// sleep to simulate some computation time Thread.sleep(1+(int)random(1000\*N));

for (int i = 0; i < N; i++) c[i] = a[i] + b[i]; performed = true;

}

public String toString() { String value = "\n" + name; value += "\na=";

for (int i = 0; i < N; i++) value += " " + a[i]; value += "\nb=";

for (int i = 0; i < N; i++) value += " " + b[i]; if (performed) {

value += "\nc=";

for (int i = 0; i < N; i++) value += " " + c[i];

}

return value;

}

}

class ComputeServer extends UnicastRemoteObject implements Compute {

public ComputeServer() throws RemoteException { } public Work compute(Work w)

throws RemoteException, InterruptedException { System.out.println(SERVER\_NAME + " " + Thread.currentThread()

+ " got work request:" + w); w.doWork();

System.out.println(SERVER\_NAME + " " + Thread.currentThread()

+ " sending reply:" + w); return w;

}

public static void main(String args[]) { int serverPort = Compute.SERVER\_PORT;

int runTime = Compute.RUN\_TIME;// seconds try {

serverPort = Integer.parseInt(args[0]); runTime = Integer.parseInt(args[1]);

} catch (Exception e) { /\* use defaults \*/ } System.out.println("Server: serverMachine=" + SERVER\_MACHINE

+ ", serverName=" + SERVER\_NAME + ", serverPort="

+ serverPort + ", runTime=" + runTime);

// create a registry and register this server try {

Registry registry = LocateRegistry.createRegistry(serverPort);

ComputeServer server = new ComputeServer(); registry.bind(SERVER\_NAME, server);

} catch (Exception e) {

System.err.println(SERVER\_NAME + " exception " + e); System.exit(1);

}

System.out.println("server " + SERVER\_NAME

+ " has been created and bound in the registry"); try {

Thread.sleep((runTime+10)\*1000);

} catch (InterruptedException e) { /\* ignored \*/ } System.out.println("time to terminate the Server and exit"); System.exit(0);

}

}

class Client extends Sugar implements Runnable { private String name = null;

private int id = -1;

private Compute server = null; private int napTime = 0; private Thread me = null;

private Client(int id, Compute server, int napTime) { this.name = "Client " + id;

this.id = id; this.server = server; this.napTime = napTime;

(me = new Thread(this)).start();

}

public void timeToQuit() { me.interrupt(); }

public void pauseTilDone() throws InterruptedException

{ me.join(); } public void run() {

int napping;

Work w = null;

if (Thread.currentThread() != me) return; while (true) {

if (Thread.interrupted()) { System.out.println("age=" + age() + ", " + name

+ " interrupted"); return;

}

napping = 1 + (int) random(napTime); try {

Thread.sleep(napping);

} catch (InterruptedException e) { System.out.println("age=" + age() + ", " + name

+ " interrupted out of sleep"); return;

}

w = new Work(name); System.out.println("age=" + age() + ", "

+ name + " sending to server work:" + w); try {

w = server.compute(w);

} catch (Exception e) { System.err.println("Client exception " + e); return;

}

System.out.println("age=" + age() + ", "

+ name + " received from server reply:" + w);

}

}

public static void main(String[] args) { String serverName = Compute.SERVER\_NAME;

String serverMachine = Compute.SERVER\_MACHINE;

int serverPort = Compute.SERVER\_PORT; int numClients = 3;

int napTime = 4;// both in int runTime = Compute.RUN\_TIME; // seconds try {

serverMachine = args[0]; serverName = args[1];

serverPort = Integer.parseInt(args[2]); numClients = Integer.parseInt(args[3]); napTime = Integer.parseInt(args[4]); runTime = Integer.parseInt(args[5]);

} catch (Exception e) { /\* use defaults \*/ } System.out.println("Client: serverMachine=" + serverMachine

+ ", serverName=" + serverName + ", serverPort=" + serverPort

+ "\n numClients=" + numClients + ", napTime=" + napTime

+ ", runTime=" + runTime); Compute server = null;

try {

server = (Compute)

Naming.lookup("rmi://" + serverMachine + ":"

+ serverPort + "/" + serverName);

} catch (Exception e) { System.err.println("Client exception " + e); System.exit(1);

}

Client[] c = new Client[numClients]; for (int i = 0; i < numClients; i++)

c[i] = new Client(i, server, 1000\*napTime); System.out.println("All Client threads started");

// let the Clients run for a while try {

Thread.sleep(runTime\*1000); System.out.println("age=" + age()

+ ", time to terminate the Clients and exit"); for (int i = 0; i < numClients; i++)

c[i].timeToQuit(); Thread.sleep(1000);

for (int i = 0; i < numClients; i++) c[i].pauseTilDone();

} catch (InterruptedException e) { /\* ignored \*/ } System.out.println("age=" + age()

+ ", all Clients are done"); System.exit(0);

}

}

/\* ............... To run:

machineA% javac Compute.java machineA% rmic ComputeServer machineA% java ComputeServer &

machineA% rsh machineB "java Client machineA"

\*/

# Sample run of Compute.java server

% javac Compute.java

% rmic ComputeServer

% java ComputeServer &

Server: serverMachine=localhost, serverName=ComputeServer, serverPort=8989, runTime=20 server ComputeServer has been created and bound in the registry

Client 2

a= -2.6956833547799772 2.7056242913415076 1.5467036159966847

b= 0.9272845052351353 2.375308405708611 1.7315712879212102

ComputeServer Thread[TCP Connection(5)-barry.popesteen.org/134.210.51.61,5,RMI runtime got work request:

Client 1

a= 2.198285066955644 1.1014851982887102 2.1195367406109042

b= -2.40242980205884 -0.2716969010229877 1.7543779582559216

ComputeServer Thread[TCP Connection(4)-barry.popesteen.org/134.210.51.61,5,RMI runtime sending reply:

Client 2

a= -2.6956833547799772 2.7056242913415076 1.5467036159966847

b= 0.9272845052351353 2.375308405708611 1.7315712879212102

c= -1.768398849544842 5.080932697050119 3.278274903917895

ComputeServer Thread[TCP Connection(6)-barry.popesteen.org/134.210.51.61,5,RMI runtime got work request:

Client 0

a= 1.1018592459170318 2.998499863398912 -2.2753431857554913

b= 1.60740034099687 2.4918661638800934 -0.35813980483005325

ComputeServer Thread[TCP Connection(5)-barry.popesteen.org/134.210.51.61,5,RMI runtime sending reply:

Client 1

a= 2.198285066955644 1.1014851982887102 2.1195367406109042

b= -2.40242980205884 -0.2716969010229877 1.7543779582559216

c= -0.20414473510319597 0.8297882972657225 3.873914698866826

ComputeServer Thread[TCP Connection(5)-barry.popesteen.org/134.210.51.61,5,RMI runtime got work request:

Client 2

a= -0.8261092470419777 0.721414082931692 -0.42645707078792716

b= -0.5302742195582928 0.8483683515941847 -1.0025008898402417

ComputeServer Thread[TCP Connection(6)-barry.popesteen.org/134.210.51.61,5,RMI runtime sending reply:

Client 0

a= 1.1018592459170318 2.998499863398912 -2.2753431857554913

b= 1.60740034099687 2.4918661638800934 -0.35813980483005325

c= 2.709259586913902 5.490366027279006 -2.6334829905855446

ComputeServer Thread[TCP Connection(5)-barry.popesteen.org/134.210.51.61,5,RMI runtime sending reply:

Client 2

a= -0.8261092470419777 0.721414082931692 -0.42645707078792716

b= -0.5302742195582928 0.8483683515941847 -1.0025008898402417

c= -1.3563834666002705 1.5697824345258766 -1.4289579606281688

ComputeServer Thread[TCP Connection(5)-barry.popesteen.org/134.210.51.61,5,RMI runtime got work request:

Client 2

a= -0.10302907032858588 2.122280172623806 -2.972889012811118

b= -2.1063144034959604 1.2484700810438456 -0.34516873732456776

ComputeServer Thread[TCP Connection(5)-barry.popesteen.org/134.210.51.61,5,RMI runtime sending reply:

Client 2

a= -0.10302907032858588 2.122280172623806 -2.972889012811118

b= -2.1063144034959604 1.2484700810438456 -0.34516873732456776

c= -2.2093434738245463 3.3707502536676515 -3.3180577501356856

ComputeServer Thread[TCP Connection(5)-barry.popesteen.org/134.210.51.61,5,RMI runtime got work request:

Client 1

a= 1.8732549025182514 1.6775852683316153 -1.3943090135761338

b= 0.4459518118556396 0.010579789697764852 -2.106450761604641

ComputeServer Thread[TCP Connection(6)-barry.popesteen.org/134.210.51.61,5,RMI runtime got work request:

Client 0

a= -2.8410081979322523 2.5649645745053986 -0.026610740169620506

b= 0.7851856936449471 0.616151284898736 -2.9907779217992445

Client 2

a= -1.5686083274058547 1.9379634892476183 0.7979989612913752

b= -2.4602035559398168 2.111407097419087 -0.13995152334940153

ComputeServer Thread[TCP Connection(4)-barry.popesteen.org/134.210.51.61,5,RMI runtime sending reply:

Client 2

a= -1.5686083274058547 1.9379634892476183 0.7979989612913752

b= -2.4602035559398168 2.111407097419087 -0.13995152334940153

c= -4.028811883345671 4.049370586666705 0.6580474379419736

ComputeServer Thread[TCP Connection(6)-barry.popesteen.org/134.210.51.61,5,RMI runtime sending reply:

Client 0

a= -2.8410081979322523 2.5649645745053986 -0.026610740169620506

b= 0.7851856936449471 0.616151284898736 -2.9907779217992445

c= -2.0558225042873053 3.1811158594041347 -3.017388661968865

ComputeServer Thread[TCP Connection(5)-barry.popesteen.org/134.210.51.61,5,RMI runtime sending reply:

Client 1

a= 1.8732549025182514 1.6775852683316153 -1.3943090135761338

b= 0.4459518118556396 0.010579789697764852 -2.106450761604641

c= 2.319206714373891 1.68816505802938 -3.5007597751807746

ComputeServer Thread[TCP Connection(5)-barry.popesteen.org/134.210.51.61,5,RMI runtime got work request:

Client 1

a= 1.13032496867271 1.7356634369443213 -1.039424286223417

b= 1.6589595605000858 -2.2151755327196945 -2.2322555512103297

ComputeServer Thread[TCP Connection(5)-barry.popesteen.org/134.210.51.61,5,RMI runtime sending reply:

Client 1

a= 1.13032496867271 1.7356634369443213 -1.039424286223417

b= 1.6589595605000858 -2.2151755327196945 -2.2322555512103297

c= 2.7892845291727957 -0.4795120957753731 -3.271679837433747

ComputeServer Thread[TCP Connection(5)-barry.popesteen.org/134.210.51.61,5,RMI runtime got work request:

Client 0

a= -0.27919433374176084 -1.6389486800885282 0.49400802045625003

b= -1.0951363826810552 -1.7985257452276389 1.0206951985241437

ComputeServer Thread[TCP Connection(5)-barry.popesteen.org/134.210.51.61,5,RMI runtime sending reply:

Client 0

a= -0.27919433374176084 -1.6389486800885282 0.49400802045625003

b= -1.0951363826810552 -1.7985257452276389 1.0206951985241437

c= -1.374330716422816 -3.437474425316167 1.5147032189803937

ComputeServer Thread[TCP Connection(5)-barry.popesteen.org/134.210.51.61,5,RMI runtime got work request:

Client 2

a= -2.657830948742076 0.9247497273131033 -1.1837878935327522

b= -0.7360974998678449 -2.5187722515825985 1.3749770142429956

ComputeServer Thread[TCP Connection(5)-barry.popesteen.org/134.210.51.61,5,RMI runtime sending reply:

Client 2

a= -2.657830948742076 0.9247497273131033 -1.1837878935327522

b= -0.7360974998678449 -2.5187722515825985 1.3749770142429956

c= -3.393928448609921 -1.5940225242694952 0.1911891207102434

ComputeServer Thread[TCP Connection(5)-barry.popesteen.org/134.210.51.61,5,RMI runtime got work request:

Client 0

a= 1.4082232953660423 -0.9931870973853112 1.3910761385634984

b= -2.889407167103009 -0.7824050505085749 2.903892793441319

ComputeServer Thread[TCP Connection(6)-barry.popesteen.org/134.210.51.61,5,RMI runtime got work request:

Client 1

a= -0.955428681740063 -2.5673385086416904 -2.780364026600216

b= -1.8793447696742414 -1.1507818551369844 2.7499555247812895

ComputeServer Thread[TCP Connection(6)-barry.popesteen.org/134.210.51.61,5,RMI runtime

sending reply:

Client 1

a= -0.955428681740063 -2.5673385086416904 -2.780364026600216

b= -1.8793447696742414 -1.1507818551369844 2.7499555247812895

c= -2.8347734514143044 -3.7181203637786746 -0.030408501818926403

ComputeServer Thread[TCP Connection(5)-barry.popesteen.org/134.210.51.61,5,RMI runtime sending reply:

Client 0

a= 1.4082232953660423 -0.9931870973853112 1.3910761385634984

b= -2.889407167103009 -0.7824050505085749 2.903892793441319

c= -1.4811838717369668 -1.775592147893886 4.294968932004817

ComputeServer Thread[TCP Connection(5)-barry.popesteen.org/134.210.51.61,5,RMI runtime got work request:

Client 2

a= 1.9275748750871307 -1.585790045655693 -0.5744877856676425

b= -1.6055158235742573 -2.439632153002778 -1.8423081956633163

ComputeServer Thread[TCP Connection(5)-barry.popesteen.org/134.210.51.61,5,RMI runtime sending reply:

Client 2

a= 1.9275748750871307 -1.585790045655693 -0.5744877856676425

b= -1.6055158235742573 -2.439632153002778 -1.8423081956633163

c= 0.3220590515128734 -4.025422198658471 -2.4167959813309587

ComputeServer Thread[TCP Connection(5)-barry.popesteen.org/134.210.51.61,5,RMI runtime got work request:

Client 1

a= -2.0501067221742413 -0.8359759895038006 2.8004561794994416

b= 0.3434141542833764 1.5887901295117377 1.613726707462031

ComputeServer Thread[TCP Connection(6)-barry.popesteen.org/134.210.51.61,5,RMI runtime got work request:

Client 0

a= 2.6637065224520526 2.744898828042219 2.6241017194381673

b= -2.120103401814829 2.031211204287832 -0.6996379517312672

time to terminate the Server and exit

# Sample run of Compute.java clients

% java Client

Client: serverMachine=localhost, serverName=ComputeServer, serverPort=8989 numClients=3, napTime=4, runTime=20

All Client threads started

age=2755, Client 2 sending to server work:

Client 2

a= -2.6956833547799772 2.7056242913415076 1.5467036159966847

b= 0.9272845052351353 2.375308405708611 1.7315712879212102

age=2848, Client 1 sending to server work:

Client 1

a= 2.198285066955644 1.1014851982887102 2.1195367406109042

b= -2.40242980205884 -0.2716969010229877 1.7543779582559216

age=3918, Client 0 sending to server work:

Client 0

a= 1.1018592459170318 2.998499863398912 -2.2753431857554913

b= 1.60740034099687 2.4918661638800934 -0.35813980483005325

age=4362, Client 2 received from server reply:

Client 2

a= -2.6956833547799772 2.7056242913415076 1.5467036159966847

b= 0.9272845052351353 2.375308405708611 1.7315712879212102

c= -1.768398849544842 5.080932697050119 3.278274903917895

age=4423, Client 1 received from server reply:

Client 1

a= 2.198285066955644 1.1014851982887102 2.1195367406109042

b= -2.40242980205884 -0.2716969010229877 1.7543779582559216

c= -0.20414473510319597 0.8297882972657225 3.873914698866826

age=4941, Client 2 sending to server work:

Client 2

a= -0.8261092470419777 0.721414082931692 -0.42645707078792716

b= -0.5302742195582928 0.8483683515941847 -1.0025008898402417

age=5193, Client 0 received from server reply:

Client 0

a= 1.1018592459170318 2.998499863398912 -2.2753431857554913

b= 1.60740034099687 2.4918661638800934 -0.35813980483005325

c= 2.709259586913902 5.490366027279006 -2.6334829905855446

age=5223, Client 2 received from server reply:

Client 2

a= -0.8261092470419777 0.721414082931692 -0.42645707078792716

b= -0.5302742195582928 0.8483683515941847 -1.0025008898402417

c= -1.3563834666002705 1.5697824345258766 -1.4289579606281688

age=5912, Client 2 sending to server work:

Client 2

a= -0.10302907032858588 2.122280172623806 -2.972889012811118

b= -2.1063144034959604 1.2484700810438456 -0.34516873732456776

age=8254, Client 2 received from server reply:

Client 2

a= -0.10302907032858588 2.122280172623806 -2.972889012811118

b= -2.1063144034959604 1.2484700810438456 -0.34516873732456776

c= -2.2093434738245463 3.3707502536676515 -3.3180577501356856

age=8431, Client 1 sending to server work:

Client 1

a= 1.8732549025182514 1.6775852683316153 -1.3943090135761338

b= 0.4459518118556396 0.010579789697764852 -2.106450761604641

age=8521, Client 0 sending to server work:

Client 0

a= -2.8410081979322523 2.5649645745053986 -0.026610740169620506

b= 0.7851856936449471 0.616151284898736 -2.9907779217992445

age=8621, Client 2 sending to server work:

Client 2

a= -1.5686083274058547 1.9379634892476183 0.7979989612913752

b= -2.4602035559398168 2.111407097419087 -0.13995152334940153

age=10054, Client 2 received from server reply:

Client 2

a= -1.5686083274058547 1.9379634892476183 0.7979989612913752

b= -2.4602035559398168 2.111407097419087 -0.13995152334940153

c= -4.028811883345671 4.049370586666705 0.6580474379419736

age=10343, Client 0 received from server reply:

Client 0

a= -2.8410081979322523 2.5649645745053986 -0.026610740169620506

b= 0.7851856936449471 0.616151284898736 -2.9907779217992445

c= -2.0558225042873053 3.1811158594041347 -3.017388661968865

age=11144, Client 1 received from server reply:

Client 1

a= 1.8732549025182514 1.6775852683316153 -1.3943090135761338

b= 0.4459518118556396 0.010579789697764852 -2.106450761604641

c= 2.319206714373891 1.68816505802938 -3.5007597751807746

age=11591, Client 1 sending to server work:

Client 1

a= 1.13032496867271 1.7356634369443213 -1.039424286223417

b= 1.6589595605000858 -2.2151755327196945 -2.2322555512103297

age=12303, Client 1 received from server reply:

Client 1

a= 1.13032496867271 1.7356634369443213 -1.039424286223417

b= 1.6589595605000858 -2.2151755327196945 -2.2322555512103297

c= 2.7892845291727957 -0.4795120957753731 -3.271679837433747

age=12431, Client 0 sending to server work:

Client 0

a= -0.27919433374176084 -1.6389486800885282 0.49400802045625003

b= -1.0951363826810552 -1.7985257452276389 1.0206951985241437

age=12709, Client 0 received from server reply:

Client 0

a= -0.27919433374176084 -1.6389486800885282 0.49400802045625003

b= -1.0951363826810552 -1.7985257452276389 1.0206951985241437

c= -1.374330716422816 -3.437474425316167 1.5147032189803937

age=12802, Client 2 sending to server work:

Client 2

a= -2.657830948742076 0.9247497273131033 -1.1837878935327522

b= -0.7360974998678449 -2.5187722515825985 1.3749770142429956

age=14454, Client 2 received from server reply:

Client 2

a= -2.657830948742076 0.9247497273131033 -1.1837878935327522

b= -0.7360974998678449 -2.5187722515825985 1.3749770142429956

c= -3.393928448609921 -1.5940225242694952 0.1911891207102434

age=15061, Client 0 sending to server work:

Client 0

a= 1.4082232953660423 -0.9931870973853112 1.3910761385634984

b= -2.889407167103009 -0.7824050505085749 2.903892793441319

age=15711, Client 1 sending to server work:

Client 1

a= -0.955428681740063 -2.5673385086416904 -2.780364026600216

b= -1.8793447696742414 -1.1507818551369844 2.7499555247812895

age=16123, Client 1 received from server reply:

Client 1

a= -0.955428681740063 -2.5673385086416904 -2.780364026600216

b= -1.8793447696742414 -1.1507818551369844 2.7499555247812895

c= -2.8347734514143044 -3.7181203637786746 -0.030408501818926403

age=17493, Client 0 received from server reply:

Client 0

a= 1.4082232953660423 -0.9931870973853112 1.3910761385634984

b= -2.889407167103009 -0.7824050505085749 2.903892793441319

c= -1.4811838717369668 -1.775592147893886 4.294968932004817

age=17761, Client 2 sending to server work:

Client 2

a= 1.9275748750871307 -1.585790045655693 -0.5744877856676425

b= -1.6055158235742573 -2.439632153002778 -1.8423081956633163

age=18914, Client 2 received from server reply:

Client 2

a= 1.9275748750871307 -1.585790045655693 -0.5744877856676425

b= -1.6055158235742573 -2.439632153002778 -1.8423081956633163

c= 0.3220590515128734 -4.025422198658471 -2.4167959813309587

age=19181, Client 1 sending to server work:

Client 1

a= -2.0501067221742413 -0.8359759895038006 2.8004561794994416

b= 0.3434141542833764 1.5887901295117377 1.613726707462031

age=19621, Client 0 sending to server work:

Client 0

a= 2.6637065224520526 2.744898828042219 2.6241017194381673

b= -2.120103401814829 2.031211204287832 -0.6996379517312672

age=20771, time to terminate the Clients and exit age=20773, Client 2 interrupted out of sleep age=21781, all Clients are done

# Interface RemoteRendezvous.java

import java.rmi.\*;

public interface RemoteRendezvous extends Remote { public abstract Transaction serverGetClient

(RendezvousCondition condition)

throws RemoteException, InterruptedException; public abstract Transaction serverGetClient()

throws RemoteException, InterruptedException;

public abstract Object clientTransactServer(Object message) throws RemoteException, InterruptedException;

public abstract void send(Object message)

throws RemoteException, InterruptedException; public abstract void call(Object message)

throws RemoteException, InterruptedException;

public abstract Object receive(RendezvousCondition condition) throws RemoteException, InterruptedException;

public abstract Object receive()

throws RemoteException, InterruptedException;

}

# Class RemoteRendezvousClient.java

import java.rmi.\*;

import java.rmi.registry.\*;

public class RemoteRendezvousClient implements RemoteRendezvous { private RemoteRendezvous server = null;

public RemoteRendezvousClient(String serverName, String serverMachine, int serverPort)

throws NotBoundException, UnknownHostException, RemoteException { Registry registry = null; System.out.println("RemoteRendezvousClient: calling getRegistry("

+ serverMachine + "," + serverPort + ")"); if (serverPort > 0)

registry = LocateRegistry.getRegistry(serverMachine, serverPort); else

registry = LocateRegistry.getRegistry(serverMachine); System.out.println("RemoteRendezvousClient: getRegistry("

+ serverMachine + "," + serverPort + ") called"); System.out.println("RemoteRendezvousClient: calling lookup("

+ serverName + ")");

server = (RemoteRendezvous) registry.lookup(serverName); System.out.println("RemoteRendezvousClient: lookup("

+ serverName + ") called");

}

public RemoteRendezvousClient(String serverName, String serverMachine) throws NotBoundException, UnknownHostException, RemoteException {

this(serverName, serverMachine, 0);

}

public Transaction serverGetClient (RendezvousCondition condition)

throws RemoteException, InterruptedException { return server.serverGetClient(condition);

}

public Transaction serverGetClient()

throws RemoteException, InterruptedException { return server.serverGetClient();

}

public Object clientTransactServer(Object message) throws RemoteException, InterruptedException {

return server.clientTransactServer(message);

}

public void send(Object message)

throws RemoteException, InterruptedException { server.send(message);

}

public void call(Object message)

throws RemoteException, InterruptedException { server.call(message);

}

public Object receive(RendezvousCondition condition) throws RemoteException, InterruptedException {

return server.receive(condition);

}

public Object receive()

throws RemoteException, InterruptedException { return server.receive();

}

}

# Class RemoteRendezvousServer.java

import java.rmi.\*;

import java.rmi.registry.\*;

import java.rmi.server.UnicastRemoteObject; import java.rmi.server.ExportException;

public class RemoteRendezvousServer extends UnicastRemoteObject implements RemoteRendezvous {

private Rendezvous local = null;

public RemoteRendezvousServer(String serverName, int serverPort) throws RemoteException, AccessException, AlreadyBoundException {

super();

local = new Rendezvous(); Registry registry = null; try {

// See if a registry already exists. System.out.println("RemoteRendezvousServer: calling createRegistry("

+ serverPort + ")"); if (serverPort > 0)

registry = LocateRegistry.createRegistry(serverPort); else

registry = LocateRegistry.createRegistry(Registry.REGISTRY\_PORT); System.out.println("RemoteRendezvousServer: createRegistry("

+ serverPort + ") called");

} catch (ExportException e) {

System.out.println("ExportException: A regsitry already exists."); System.out.println("RemoteRendezvousServer: calling getRegistry("

+ serverPort + ")"); if (serverPort > 0)

registry = LocateRegistry.getRegistry(serverPort); else

registry = LocateRegistry.getRegistry(); System.out.println("RemoteRendezvousServer: getRegistry("

+ serverPort + ") called");

}

System.out.println("RemoteRendezvousServer: calling bind("

+ serverName + ")"); registry.bind(serverName, this);

System.out.println("RemoteRendezvousServer: bind("

+ serverName + ") called");

}

public RemoteRendezvousServer(String serverName)

throws RemoteException, AccessException, AlreadyBoundException { this(serverName, 0);

}

public Transaction serverGetClient (RendezvousCondition condition)

throws RemoteException, InterruptedException { return local.serverGetClient(condition);

}

public Transaction serverGetClient()

throws RemoteException, InterruptedException { return local.serverGetClient();

}

public Object clientTransactServer(Object message) throws RemoteException, InterruptedException {

return local.clientTransactServer(message);

}

public void send(Object message)

throws RemoteException, InterruptedException { local.send(message);

}

public void call(Object message)

throws RemoteException, InterruptedException { local.call(message);

}

public Object receive(RendezvousCondition condition) throws RemoteException, InterruptedException {

return local.receive(condition);

}

public Object receive()

throws RemoteException, InterruptedException { return local.receive();

}

}

# Example Transact.java

import java.util.Vector; import java.io.Serializable; import java.rmi.\*;

public class Transact {

public static final String SERVER\_NAME = "TransactServer"; public static final String SERVER\_MACHINE = "localhost"; public static final int SERVER\_PORT = 8989;

public static final int RUN\_TIME = 20;

}

class Request extends Sugar implements Serializable { private String name = null;

private int time;

private int performed = 0;

public Request(String name, int time) { this.name = name;

this.time = time;

}

public void doRequest() throws InterruptedException { System.out.println("age=" + age() + ", performing:" + this); performed = 1+(int)random(time);

// sleep to simulate some transaction time Thread.sleep(performed);

System.out.println("age=" + age() + ", performed:" + this);

}

public String getName() { return name; } public String toString() {

return "\n" + name + ", " + time + ", " + performed;

}

}

class ServerCondition implements RendezvousCondition { public ServerCondition() { }

public boolean checkCondition

(int messageNum, Vector blockedMessages, int numBlockedServers) {

Object message = blockedMessages.elementAt(messageNum); String client = ((Request) message).getName();

if (client.equals("Client 0")) return true; int size = blockedMessages.size();

/\*

* **If "Client 0" is not anywhere in the queue, then rendezvous**
* **with any client.**

\*/

for (int i = 0; i < size; i++) {

message = blockedMessages.elementAt(i); client = ((Request) message).getName();

if (client.equals("Client 0")) return false;

}

return true;

}

}

class TransactServer extends SugarRE implements Runnable { private String serverName = null;

private RemoteRendezvousServer rend = null; private ServerCondition sc = null;

private Thread me = null;

public TransactServer(String serverName, int serverPort)

throws AlreadyBoundException, AccessException, RemoteException { this.serverName = serverName;

rend = new RemoteRendezvousServer(serverName, serverPort); sc = new ServerCondition();

(me = new Thread(this)).start();

}

public void run() {

if (Thread.currentThread() != me) return; while (true) {

if (Thread.interrupted()) {

System.out.println("age=" + age() + ", " + serverName

+ " interrupted"); return;

}

try {

Transaction t = rend.serverGetClient(sc); Object m = t.serverGetRequest();

if (m != null) {

((Request) m).doRequest(); t.serverMakeReply(m);

} else

System.out.println(serverName + " got null request");

} catch (RemoteException e) {

System.out.println("age=" + age() + ", " + serverName

+ " rendezvous remote exception"); e.printStackTrace();

} catch (InterruptedException e) { System.out.println("age=" + age() + ", " + serverName

+ " interrupted out of rendezvous"); return;

}

}

}

public static void main(String args[]){ String serverName = Transact.SERVER\_NAME;

String serverMachine = Transact.SERVER\_MACHINE; int serverPort = Transact.SERVER\_PORT;

int runTime = Transact.RUN\_TIME;// seconds try {

serverPort = Integer.parseInt(args[0]); runTime = Integer.parseInt(args[1]);

} catch (Exception e) { /\* use defaults \*/ } System.out.println("Server: serverMachine=" + serverMachine

+ ", serverName=" + serverName + ", serverPort="

+ serverPort + ", runTime=" + runTime); try {

TransactServer server = new TransactServer(serverName, serverPort);

} catch (Exception e) {

System.err.println(serverName + " exception " + e); e.printStackTrace();

System.exit(1);

}

System.out.println("age=" + age() + ", " + serverName

+ " has been created and bound in the registry"); try {

Thread.sleep((runTime+10)\*1000);

} catch (InterruptedException e) { /\* ignored \*/ } System.out.println("age=" + age() + ", " + serverName

+ ", time to terminate and exit"); System.exit(0);

}

}

class Client extends SugarRE implements Runnable { private String name = null;

private int id = -1;

private RemoteRendezvousClient rend = null; private int napTime = 0;

private Thread me = null;

private Client(int id, RemoteRendezvousClient rend, int napTime) { this.name = "Client " + id;

this.id = id; this.rend = rend;

this.napTime = napTime;

(me = new Thread(this)).start();

}

public void timeToQuit() { me.interrupt(); }

public void pauseTilDone() throws InterruptedException

{ me.join(); } public void run() {

int napping;

Request r = null;

if (Thread.currentThread() != me) return; while (true) {

if (Thread.interrupted()) { System.out.println("age=" + age() + ", " + name

+ " interrupted"); return;

}

napping = 1 + (int) random(napTime); try {

Thread.sleep(napping);

} catch (InterruptedException e) { System.out.println("age=" + age() + ", " + name

+ " interrupted out of sleep"); return;

}

r = new Request(name, napTime); System.out.println("age=" + age() + ", "

+ name + " sending to server request:" + r); try {

r = (Request) rend.clientTransactServer(r);

} catch (Exception e) { System.err.println("Client exception " + e); e.printStackTrace();

return;

}

System.out.println("age=" + age() + ", "

+ name + " received from server reply:" + r);

}

}

public static void main(String[] args) { String serverName = Transact.SERVER\_NAME;

String serverMachine = Transact.SERVER\_MACHINE; int serverPort = Transact.SERVER\_PORT;

int numClients = 3;

int napTime = 4;// both in int runTime = Transact.RUN\_TIME; // seconds try {

serverMachine = args[0]; serverName = args[1];

serverPort = Integer.parseInt(args[2]); numClients = Integer.parseInt(args[3]); napTime = Integer.parseInt(args[4]); runTime = Integer.parseInt(args[5]);

} catch (Exception e) { /\* use defaults \*/ } System.out.println("Client: serverMachine=" + serverMachine

+ ", serverName=" + serverName + ", serverPort=" + serverPort

+ "\n numClients=" + numClients + ", napTime=" + napTime

+ ", runTime=" + runTime); RemoteRendezvousClient rend = null; try {

rend = new RemoteRendezvousClient(serverName, serverMachine, serverPort);

} catch (Exception e) { System.err.println("Client exception " + e); e.printStackTrace();

System.exit(1);

}

Client[] c = new Client[numClients]; for (int i = 0; i < numClients; i++)

c[i] = new Client(i, rend, 1000\*napTime); System.out.println("All Client threads started");

// let the Clients run for a while try {

Thread.sleep(runTime\*1000); System.out.println("age=" + age()

+ ", time to terminate the Clients and exit"); for (int i = 0; i < numClients; i++)

c[i].timeToQuit(); Thread.sleep(1000);

for (int i = 0; i < numClients; i++) c[i].pauseTilDone();

} catch (InterruptedException e) { /\* ignored \*/ } System.out.println("age=" + age()

+ ", all Clients are done"); System.exit(0);

}

}

/\* ............... To run:

machineA% javac Transact.java machineA% rmic RemoteRendezvousServer machineA% java TransactServer &

machineA% rsh machineB "java Client machineA"

\*/

# Sample run of Transact.java server

% javac Transact.java

% rmic RemoteRendezvousServer

% java TransactServer &

Server: serverMachine=localhost, serverName=TransactServer, serverPort=8989, runTime=2 RemoteRendezvousServer: calling createRegistry(8989)

RemoteRendezvousServer: createRegistry(8989) called RemoteRendezvousServer: calling bind(TransactServer)

RemoteRendezvousServer: bind(TransactServer) called

age=247, TransactServer has been created and bound in the registry age=11737, performing:

Client 1, 4000, 0 age=13946, performed:

Client 1, 4000, 2194

age=13948, performing:

Client 0, 4000, 0 age=16815, performed:

Client 0, 4000, 2852

age=16816, performing:

Client 2, 4000, 0 age=19025, performed:

Client 2, 4000, 2193

age=19026, performing:

Client 1, 4000, 0 age=21025, performed:

Client 1, 4000, 1983

age=21026, performing:

Client 0, 4000, 0 age=22675, performed:

Client 0, 4000, 1637

age=22676, performing:

Client 2, 4000, 0 age=25345, performed:

Client 2, 4000, 2658

age=25346, performing:

Client 1, 4000, 0 age=26055, performed:

Client 1, 4000, 700 age=26211, performing:

Client 2, 4000, 0 age=26607, performed:

Client 2, 4000, 367 age=26619, performing:

Client 0, 4000, 0 age=29055, performed:

Client 0, 4000, 2427

age=29201, performing:

Client 1, 4000, 0

age=30255, TransactServer, time to terminate and exit

# Sample run of Transact.java clients

% java Client

Client: serverMachine=localhost, serverName=TransactServer, serverPort=8989 numClients=3, napTime=4, runTime=20

RemoteRendezvousClient: calling getRegistry(localhost,8989) RemoteRendezvousClient: getRegistry(localhost,8989) called RemoteRendezvousClient: calling lookup(TransactServer) RemoteRendezvousClient: lookup(TransactServer) called

All Client threads started

age=1654, Client 1 sending to server request:

Client 1, 4000, 0

age=1850, Client 2 sending to server request: Client 2, 4000, 0

age=2832, Client 0 sending to server request:

Client 0, 4000, 0

age=3908, Client 1 received from server reply:

Client 1, 4000, 2194

age=6121, Client 1 sending to server request:

Client 1, 4000, 0

age=6776, Client 0 received from server reply:

Client 0, 4000, 2852

age=8986, Client 2 received from server reply:

Client 2, 4000, 2193

age=9101, Client 0 sending to server request:

Client 0, 4000, 0

age=10860, Client 2 sending to server request:

Client 2, 4000, 0

age=10986, Client 1 received from server reply:

Client 1, 4000, 1983

age=12636, Client 0 received from server reply:

Client 0, 4000, 1637

age=13310, Client 1 sending to server request:

Client 1, 4000, 0

age=15306, Client 2 received from server reply:

Client 2, 4000, 2658

age=16015, Client 1 received from server reply:

Client 1, 4000, 700

age=16160, Client 2 sending to server request:

Client 2, 4000, 0

age=16430, Client 0 sending to server request:

Client 0, 4000, 0

age=16570, Client 2 received from server reply:

Client 2, 4000, 367

age=19015, Client 0 received from server reply:

Client 0, 4000, 2427

age=19150, Client 1 sending to server request:

Client 1, 4000, 0

age=20080, Client 2 sending to server request:

Client 2, 4000, 0

age=20290, Client 0 sending to server request:

Client 0, 4000, 0

age=20730, time to terminate the Clients and exit age=21740, all Clients are done

# Example Ring.java

import java.io.Serializable; import java.rmi.\*;

public class Ring {

public static final String RING\_NAME = "RingMember"; public static final String RING\_MACHINE = "localhost";

}

class Token implements Serializable { private int value = 0;

private String owner = null;

public Token(String o, int v) { owner = o;value = v; } public String getOwner() { return owner; }

public int getValue() { return value; } public void setOwner(String o) { owner = o; } public void setValue(int v) { value = v; } public String toString() {

return "\nToken: owner=" + owner + ", value=" + value;

}

}

class RingMember extends SugarRE implements Runnable { private RemoteRendezvousServer channel = null; private String name = null;

private int id = 0;

private RemoteRendezvousClient successor = null; private String successorMachine = null;

private String successorName = null; private int napTime = 0;// seconds private Thread me = null;

public RingMember(int d, String i, String m, String n, int t)

throws AlreadyBoundException, AccessException, RemoteException { id = d;

name = i; successorMachine = m; successorName = n; napTime = t;

channel = new RemoteRendezvousServer(name);

}

private void start() { (me = new Thread(this)).start(); } public void timeToQuit() { me.interrupt(); }

public void pauseTilDone() throws InterruptedException

{ me.join(); } public void run() {

Token t = null;

if (Thread.currentThread() != me) return; System.out.println("age=" + age() + ", " + name + " go!"); if (id == 0) {

// Special case: create the token and pause

// for all other ring members to initialize

// and register themselves. try {

Thread.sleep(5000);// Yes, this is a kludge!

} catch (InterruptedException e) { } try {

successor =

new RemoteRendezvousClient(successorName, successorMachine); System.out.println("age=" + age() + ", " + name

+ ", successor looked up");

} catch (Exception e) {

System.err.println("age=" + age() + ", " + name

+ ", successor exception " + e); e.printStackTrace();

return;

}

t = new Token(name, 1000); System.out.println("age=" + age() + ", " + name

+ " creating initial token" + t); try {

send(successor, t);

} catch (Exception e) {

System.err.println("age=" + age() + ", " + name

+ ", initial token exception " + e); e.printStackTrace();

return;

}

System.out.println("age=" + age() + ", " + name

+ " passed initial token to successor " + successorName); while (true) {

if (Thread.interrupted()) { System.out.println("age=" + age() + ", " + name

+ " interrupted in run"); return;

}

t = null; try {

t = (Token) receive(channel);

} catch (RemoteException e) { System.out.println("age=" + age() + ", " + name

+ " rendezvous remote exception"); e.printStackTrace();

} catch (InterruptedException e) { System.out.println("age=" + age() + ", " + name

+ " interrupted in receive"); return;

}

int napping = 1 + (int) (Math.random()\*1000\*napTime); System.out.println("age=" + age() + ", " + name

+ " sleeping for " + napping

+ " ms after receiving token" + t); try {

Thread.sleep(napping);

} catch (InterruptedException e) { System.out.println("age=" + age() + ", " + name

+ " interrupted in sleep"); return;

}

t.setOwner(name);t.setValue(t.getValue()+1); System.out.println("age=" + age() + ", " + name

+ " passing token" + t); try {

send(successor, t);

} catch (Exception e) {

System.err.println("age=" + age() + ", " + name

+ ", pass exception " + e); e.printStackTrace();

return;

}

System.out.println("age=" + age() + ", " + name

+ " token passed to successor " + successorName);

}

} else {

// Everybody else waits to get the token before passing it on. while (true) {

if (Thread.interrupted()) { System.out.println("age=" + age() + ", " + name

+ " interrupted in run"); return;

}

t = null; try {

t = (Token) receive(channel);

} catch (RemoteException e) { System.out.println("age=" + age() + ", " + name

+ ", rendezvous remote exception"); e.printStackTrace();

} catch (InterruptedException e) { System.out.println("age=" + age() + ", " + name

+ " interrupted in receive"); return;

}

int napping = 1 + (int) (Math.random()\*1000\*napTime); System.out.println("age=" + age() + ", " + name

+ " sleeping for " + napping

+ " ms after receiving token" + t); try {

Thread.sleep(napping);

} catch (InterruptedException e) { System.out.println("age=" + age() + ", " + name

+ " interrupted in sleep"); return;

}

t.setOwner(name);t.setValue(t.getValue()+1); System.out.println("age=" + age() + ", " + name

+ " passing token" + t); if (successor == null) {

// Don't do this until you get the token from

// your predecessor to make sure your successor

// is registered try {

successor = new

RemoteRendezvousClient(successorName, successorMachine); System.out.println("age=" + age() + ", " + name

+ ", successor looked up");

} catch (Exception e) {

System.err.println("age=" + age() + ", " + name

+ ", successor exception " + e); e.printStackTrace();

return;

}

}

try {

send(successor, t);

} catch (Exception e) {

System.err.println("age=" + age() + ", " + name

+ ", pass exception " + e); e.printStackTrace();

return;

}

System.out.println("age=" + age() + ", " + name

+ " token passed to successor " + successorName);

}

}

}

public static void main(String args[]) { int id = 0;

int successorId = 1;

String successorMachine = Ring.RING\_MACHINE; String myMachine = Ring.RING\_MACHINE;

int napTime = 4; // both in int runTime = 30; // seconds try {

id = Integer.parseInt(args[0]); successorId = Integer.parseInt(args[1]); successorMachine = args[2];

myMachine = args[3];

napTime = Integer.parseInt(args[4]); runTime = Integer.parseInt(args[5]);

} catch (Exception e) { /\* use defaults \*/ } String name = Ring.RING\_NAME + id;

String successorName = Ring.RING\_NAME + successorId; System.out.println("age=" + age() + ", " + name

+ "\nid = " + id

+ "\nsuccessor machine = " + successorMachine

+ "\nsuccessor name = " + successorName); RingMember member = null;

try {

member = new RingMember(id, name, successorMachine, successorName, napTime);

} catch (Exception e) {

System.err.println("age=" + age() + ", " + name

+ ", exception " + e); e.printStackTrace(); System.exit(1);

}

System.out.println("age=" + age() + ", " + name

+ " has been created and bound in the registry"); member.start();

try {

Thread.sleep(1000\*runTime); System.out.println("age=" + age() + ", " + name

+ ", time to terminate and exit"); member.timeToQuit(); Thread.sleep(1000); member.pauseTilDone();

} catch (InterruptedException e) { /\* ignored \*/ } System.exit(0);

}

}

/\* ............... To run:

machineA% javac Ring.java

machineA% rmic RemoteRendezvousServer machineA% rsh machineC "rmiregistry &"

machineA% rsh machineC "java RingMember 2 0 machineA &" machineA% rsh machineB "rmiregistry &"

machineA% rsh machineB "java RingMember 1 2 machineC &" machineA% rmiregistry &

machineA% java RingMember 0 1 machineB &

\*/

# Sample run of Ring.java ring member 0

% javac Ring.java

% rmic RemoteRendezvousServer

% java RingMember 0 1 age=10, RingMember0

id = 0

successor machine = localhost successor name = RingMember1

RemoteRendezvousServer: calling createRegistry(0) RemoteRendezvousServer: createRegistry(0) called RemoteRendezvousServer: calling bind(RingMember0)

RemoteRendezvousServer: bind(RingMember0) called

age=600, RingMember0 has been created and bound in the registry age=610, RingMember0 go!

RemoteRendezvousClient: calling getRegistry(localhost,0) RemoteRendezvousClient: getRegistry(localhost,0) called RemoteRendezvousClient: calling lookup(RingMember1) RemoteRendezvousClient: lookup(RingMember1) called age=5655, RingMember0, successor looked up

age=5658, RingMember0 creating initial token Token: owner=RingMember0, value=1000

age=5694, RingMember0 passed initial token to successor RingMember1 age=9451, RingMember0 sleeping for 21 ms after receiving token

Token: owner=RingMember2, value=1002 age=9486, RingMember0 passing token

Token: owner=RingMember0, value=1003

age=9493, RingMember0 token passed to successor RingMember1 age=13681, RingMember0 sleeping for 490 ms after receiving token

Token: owner=RingMember2, value=1005 age=14175, RingMember0 passing token

Token: owner=RingMember0, value=1006

age=14182, RingMember0 token passed to successor RingMember1 age=20491, RingMember0 sleeping for 321 ms after receiving token

Token: owner=RingMember2, value=1008 age=20825, RingMember0 passing token

Token: owner=RingMember0, value=1009

age=20832, RingMember0 token passed to successor RingMember1 age=24211, RingMember0 sleeping for 2132 ms after receiving token

Token: owner=RingMember2, value=1011 age=26355, RingMember0 passing token

Token: owner=RingMember0, value=1012

age=26362, RingMember0 token passed to successor RingMember1 age=29581, RingMember0 sleeping for 3991 ms after receiving token

Token: owner=RingMember2, value=1014

age=30615, RingMember0, time to terminate and exit age=30618, RingMember0 interrupted in sleep

# Sample run of Ring.java ring member 1

% java RingMember 1 2 & age=10, RingMember1

id = 1

successor machine = localhost successor name = RingMember2

RemoteRendezvousServer: calling createRegistry(0) ExportException: A regsitry already exists.

RemoteRendezvousServer: calling getRegistry(0) RemoteRendezvousServer: getRegistry(0) called RemoteRendezvousServer: calling bind(RingMember1)

RemoteRendezvousServer: bind(RingMember1) called

age=1838, RingMember1 has been created and bound in the registry age=1844, RingMember1 go!

age=5782, RingMember1 sleeping for 2703 ms after receiving token Token: owner=RingMember0, value=1000

age=8502, RingMember1 passing token Token: owner=RingMember1, value=1001

RemoteRendezvousClient: calling getRegistry(localhost,0) RemoteRendezvousClient: getRegistry(localhost,0) called RemoteRendezvousClient: calling lookup(RingMember2) RemoteRendezvousClient: lookup(RingMember2) called age=8709, RingMember1, successor looked up

age=8747, RingMember1 token passed to successor RingMember2 age=9571, RingMember1 sleeping for 183 ms after receiving token

Token: owner=RingMember0, value=1003 age=9771, RingMember1 passing token

Token: owner=RingMember1, value=1004

age=9783, RingMember1 token passed to successor RingMember2 age=14258, RingMember1 sleeping for 3099 ms after receiving token

Token: owner=RingMember0, value=1006 age=17370, RingMember1 passing token

Token: owner=RingMember1, value=1007

age=17379, RingMember1 token passed to successor RingMember2 age=20908, RingMember1 sleeping for 1294 ms after receiving token

Token: owner=RingMember0, value=1009 age=22210, RingMember1 passing token

Token: owner=RingMember1, value=1010

age=22219, RingMember1 token passed to successor RingMember2 age=26438, RingMember1 sleeping for 1491 ms after receiving token

Token: owner=RingMember0, value=1012 age=27950, RingMember1 passing token

Token: owner=RingMember1, value=1013

age=27958, RingMember1 token passed to successor RingMember2 age=31850, RingMember1, time to terminate and exit age=31857, RingMember1 interrupted in receive

# Sample run of Ring.java ring member 2

% java RingMember 2 0 & age=10, RingMember2

id = 2

successor machine = localhost successor name = RingMember0

RemoteRendezvousServer: calling createRegistry(0) ExportException: A regsitry already exists.

RemoteRendezvousServer: calling getRegistry(0) RemoteRendezvousServer: getRegistry(0) called RemoteRendezvousServer: calling bind(RingMember2)

RemoteRendezvousServer: bind(RingMember2) called

age=1964, RingMember2 has been created and bound in the registry age=2024, RingMember2 go!

age=8926, RingMember2 sleeping for 474 ms after receiving token Token: owner=RingMember1, value=1001

age=9413, RingMember2 passing token Token: owner=RingMember2, value=1002

RemoteRendezvousClient: calling getRegistry(localhost,0) RemoteRendezvousClient: getRegistry(localhost,0) called RemoteRendezvousClient: calling lookup(RingMember0) RemoteRendezvousClient: lookup(RingMember0) called age=9689, RingMember2, successor looked up

age=9712, RingMember2 token passed to successor RingMember0 age=9963, RingMember2 sleeping for 3952 ms after receiving token

Token: owner=RingMember1, value=1004 age=13933, RingMember2 passing token

Token: owner=RingMember2, value=1005

age=13942, RingMember2 token passed to successor RingMember0 age=17559, RingMember2 sleeping for 3179 ms after receiving token

Token: owner=RingMember1, value=1007 age=20743, RingMember2 passing token

Token: owner=RingMember2, value=1008

age=20752, RingMember2 token passed to successor RingMember0 age=22400, RingMember2 sleeping for 2053 ms after receiving token

Token: owner=RingMember1, value=1010 age=24463, RingMember2 passing token

Token: owner=RingMember2, value=1011

age=24472, RingMember2 token passed to successor RingMember0 age=28139, RingMember2 sleeping for 1687 ms after receiving token

Token: owner=RingMember1, value=1013 age=29833, RingMember2 passing token

Token: owner=RingMember2, value=1014

age=29842, RingMember2 token passed to successor RingMember0 age=32035, RingMember2, time to terminate and exit age=32038, RingMember2 interrupted in receive

# Section 13. Wrapup Tutorial summary

### In this tutorial, we examined one of the Java language's most important features -- support

for multithreaded (concurrent) programming.

One benefit of multithreaded programs is that they can take advantage of the additional CPUs in a shared-memory multiprocessor architecture in order to execute more quickly.

Using multiple threads can also simplify the design of a program, as in the example of a server program in which each incoming client request is handled by a dedicated thread.

Thread synchronization is extremely important, and this tutorial provides many examples to illustrate this concept.

This tutorial has also illustrated the following concepts by providing definitions, examples, resources, and sample code: Starting Java threads; thread states, priorities, and methods; volatile modifiers; race conditions; synchronized blocks; monitors; semaphores; message passing; rendezvous; and Remote Method Invocation.

Resources

Download code.zip, a zip file containing all example Java programs used in this tutorial.

The following online and print resources will help you follow up on the material presented in this tutorial:

* All example Java programs in this tutorial have been executed on a PC running [*Red Hat's version 7.0 of Linux*](http://www.redhat.com/), using the [*IBM Java software developer kit version 1.3.0 for Linux*](http://www.ibm.com/developerworks/java/jdk/linux130/).

### In "[*Writing multithreaded Java applications*](http://www-106.ibm.com/developerworks/library/j-thread.html)" (developerWorks, March 2001), Alex Roetter explains the Java Thread API, outlines issues involved in multithreading, and offers solutions to common problems.

* Multithreaded programming expert Brian Goetz can help you understand the tricks and traps of the Java threading model in this developerWorks forum, "[*Multithreaded Java programming*](http://www-105.ibm.com/developerworks/java_df.nsf/AllViewTemplate?OpenForm&amp;RestrictToCategory=23)."
* Brian Goetz also offers Threading lightly -- a series on threaded programming.
  + The first installment, "[*Synchronization is not the enemy*](http://www-106.ibm.com/developerworks/library/j-threads1/)" (developerWorks, July 2001), explains when you have to synchronize and how expensive it is.

### The second article, "[*Reducing contention*](http://www-106.ibm.com/developerworks/java/library/j-threads2.html)" (developerWorks, September 2001), explores several techniques for reducing contention to improve scalability in programs.

* + The third article, "[*Sometimes it's best not to share*](http://www-106.ibm.com/developerworks/library/j-threads3.html)" (developerWorks, October 2001), gives tips on exploiting the power of ThreadLocal.

### In "[*Writing efficient thread-safe classes*](http://www-106.ibm.com/developerworks/library/threadsafe/index.html) " (developerWorks, April 2000), Neel V. Kumar uses programming examples to explain how language-level support for locking objects and for inter-thread signaling makes writing thread-safe classes easy.

* Andrew D. Birrell's "[*An Introduction to Programming with Threads*](ftp://gatekeeper.dec.com/pub/DEC/SRC/research-reports/SRC-035.ps.Z)" (1989; a DEC research report) provides excellent guidance on threading.
* Doug Lea's [*Java concurrent programming package*](http://gee.cs.oswego.edu/dl/classes/EDU/oswego/cs/dl/util/concurrent/intro.html) provides standardized, efficient versions of utility classes commonly encountered in concurrent Java programming. The author also explains how to use the Java platform's threading model more precisely by illuminating the patterns and trade-offs associated with concurrent programming in his book, [*Concurrent Programming in Java: Design Principles and Patterns*](http://cseng.aw.com/book/0%2C3828%2C0201310090%2C00.html), second edition (Addison Wesley, 2000).

### [*JCSP*](http://www.cs.ukc.ac.uk/projects/ofa/jcsp/)is a Java class library providing a base range of CSP primitives found at the University of Kent, Canterbury, UK. (CSP, or Communicating Sequential Processes, is a mathematical theory for specifying and verifying complex patterns of behavior arising from interactions between concurrent objects.)

* JavaPP introduces the CSP model into Java threads, enabling Java active processes to communicate and synchronize via CSP synchronization primitives, helping to eliminate race hazards, deadlock, livelock, and starvation. Two good JavaPP sites exist -- at the [*University of Bristol*](http://www.cs.bris.ac.uk/%7Ealan/javapp.html) and the [*University of Twente*](http://www.rt.el.utwente.nl/javapp/).
* This Bill Venners' article, "[*Design for thread safety*](http://www.javaworld.com/jw-08-1998/jw-08-techniques.html)" (JavaWorld, August 1998), offers design guidelines for thread safety and provides a background on the concept of thread safety with several examples of objects -- both thread safe and not thread safe; it also delivers guidelines to help determine when thread safety is appropriate and how best to achieve it.
* Allen Holub's "[*Programming Java threads in the real world, Parts 1 through*](http://www.javaworld.com/javaworld/jw-09-1998/jw-09-threads.html) *9*" (JavaWorld, September 1998 - June 1999), is a series that purports to deliver everything you need to know to effectively program threads in real-world applications and situations.
* [*Concurrent Programming: Principles and Practice*](http://cseng.aw.com/book/0%2C3828%2C0805300864%2C00.html) by Gregory R. Andrews (Benjamin/Cummings, 1991) provides an in-depth overview of principles and practical techniques that can be used to design concurrent programs.
* [*Foundations of Multithreaded, Parallel, and Distributed Programming*](http://cseng.aw.com/book/0%2C3828%2C0201357526%2C00.html) by Gregory R. Andrews (Addison Wesley, 2000) covers such current programming techniques as semaphores, locks, barriers, monitors, message passing, and remote invocation, providing examples with complete programs, both shared and distributed.

### Stephen Hartley's book, [*Concurrent Programming: The Java Programming Language*](http://www.oup-usa.org/isbn/0195113152.html)(Oxford University Press, 1998) shows readers how to use the Java language to code semaphores, monitors, message passing, remote procedure calls, and the rendezvous for thread synchronization and communication.

* [*Java Thread Programming*](http://www.samspublishing.com/detail_sams.cfm?item=0672315858) by Paul Hyde (Sams, 1999) demonstrates how to leverage Java's thread facilities to increase program efficiency and to avoid common mistakes.
* [*Concurrency: State Models and Java Programs*](http://www.wiley.com/Corporate/Website/Objects/Products/0%2C9049%2C104914%2C00.html) by Jeff Magee and Jeff Kramer (John Wiley & Sons, 1999) provides a systematic and practical approach to designing, analyzing, and implementing concurrent programs.

### Other, more generic books on operating-system and Java-language programming that have expanded sections on multithreading and concurrent programming include:

* + [*Operating Systems: Internals and Design Principles*](http://www.phptr.com/ptrbooks/esm_0130319996.html), fourth edition, by William Stallings (Prentice Hall, 2001) reflects ongoing changes in thread and process management and concurrency.

### [*Modern Operating Systems*](http://www.phptr.com/ptrbooks/esm_0130313580.html), second edition, by Andrew Tanenbaum (Prentice Hall, 2001) has expanded its coverage of process management, threads, and security issues.

* + [*The Java Language Specification*](http://cseng.aw.com/book/0%2C3828%2C0201634511%2C00.html) by James Gosling, Bill Joy, and Guy Steele (Addison-Wesley, 1996) covers all aspects of the Java execution model, including exceptions, threads, and binary compatibility.
  + [*Core Java 2, Volume II: Advanced Features*](http://www.phptr.com/ptrbooks/ptr_0130927384.html), fifth edition, by Cay Horstmann and Gary Cornell (Prentice Hall, 2002), which starts with a chapter on multithreading, is fully updated for Sun's JDK 2 Version 1.3 and 1.4.

Your feedback

Please let us know whether this tutorial was helpful to you and how we could make it better. We'd also like to hear about other tutorial topics you'd like to see covered. Thanks!

Colophon

This tutorial was written entirely in XML, using the developerWorks Toot-O-Matic tutorial generator. The open source Toot-O-Matic tool is an XSLT stylesheet and several XSLT extension functions that convert an XML file into a number of HTML pages, a zip file, JPEG heading graphics, and two PDF files. Our ability to generate multiple text and binary formats from a single source file illustrates the power and flexibility of XML. (It also saves our production team a great deal of time and effort.)

You can get the source code for the Toot-O-Matic at [www6.software.ibm.com/dl/devworks/dw-tootomatic-p](http://www6.software.ibm.com/dl/devworks/dw-tootomatic-p). The tutorial [Building tutorials with the](http://www-105.ibm.com/developerworks/education.nsf/xml-onlinecourse-bytitle/01F99F6B8BE60C9486256A69005BD21C?OpenDocument) [Toot-O-Matic](http://www-105.ibm.com/developerworks/education.nsf/xml-onlinecourse-bytitle/01F99F6B8BE60C9486256A69005BD21C?OpenDocument) demonstrates how to use the Toot-O-Matic to create your own tutorials. developerWorks also hosts a forum devoted to the Toot-O-Matic; it's available at

[www-105.ibm.com/developerworks/xml\_df.nsf/AllViewTemplate?OpenForm&RestrictToCategory=11](http://www-105.ibm.com/developerworks/xml_df.nsf/AllViewTemplate?OpenForm&amp;RestrictToCategory=11). We'd love to know what you think about the tool.