

Computer Engineering

Jean-Charles Pomerol

Concurrent, Real-Time and Distributed Programming in Java

Threads, RTSJ and RMI

Badr Benmammar

Color section

Chapter 2

```
public class Ecrivain extends Thread {
    private String texte;
    public Ecrivain(String t) {
        texte=t;
    }
    public void run() {
        for (int i=0; i<10; i++) {
            int j=0;
            for (j<texte.length()-1; j++) {
                System.out.print(texte.substring(j,j+1));
                try {sleep((long)(Math.random() * 100));}
                catch (InterruptedException e) {}
            } System.out.println(texte.substring(j,j+1));
        } //end for on i
        System.out.println("ecrivain de "+texte+" a fini");
    } // end run
}
```

```
public class Prog55 {
    public static void main (String argv[]) {
        Ecrivain ecrivainA, ecrivainB;
        ecrivainA = new Ecrivain ("ABC");
        ecrivainB = new Ecrivain ("XYZ");
        ecrivainA.start();
        ecrivainB.start();
    }
}
```

```
public class Imprimeur1 {
    private String texte;
    public Imprimeur1() {texte=""; }
    public void imprimer(String t) {
        texte=t;
        for (int j=0; j<texte.length()-1; j++) {
            System.out.print(texte.substring(j,j+1));
            try {Thread.sleep(100);}
            catch (InterruptedException e) {}
        } // end for
        System.out.println
        (texte.substring(texte.length()-1, texte.length()));
    } // end imprimer
}
```

```
public class Prog56 {
    public static void main (String argv[]) {
        Ecrivain2 ecrivainA, ecrivainB;
        Imprimeur1 imprim= new Imprimeur1();
        ecrivainA = new Ecrivain2("ABC", imprim);
        ecrivainB = new Ecrivain2("XYZ", imprim);
        ecrivainA.start();
        ecrivainB.start();
    }
}
```

```
public class Ecrivain2 extends Thread {
    private String texte;
    private Imprimeur1 imprim;
    public Ecrivain2(String t, Imprimeur1 i) {
        imprim=i;
        texte=t;
    }
    public void run() {
        for (int i=0; i<10; i++) {
            imprim.imprimer(texte);
            try {sleep((long)(Math.random() * 100));}
            catch (InterruptedException e) {}
        } // end for
        System.out.println("ecrivain de "+texte+" a fini");
    } // end run
}
```

```

public class Imprimeur1 {
    private String texte;
    public Imprimeur1() {texte=""; }
    public synchronized void imprimer(String t) {
        texte=t;
        for (int j=0;j<texte.length()-1;j++) {
            System.out.print(texte.substring(j,j+1));
            try {Thread.sleep(100);}
            catch (InterruptedException e) {};
        } // end for
        System.out.println
        (texte.substring(texte.length()-1,texte.length()));
    } // end imprimer
}

```

```

public class Prog56 {
    public static void main (String argv[]) {
        Ecrivain2 ecrivainA, ecrivainB;
        Imprimeur1 imprim= new Imprimeur1();
        ecrivainA = new Ecrivain2("ABC", imprim);
        ecrivainB = new Ecrivain2("XYZ", imprim);
        ecrivainA.start();
        ecrivainB.start();
    }
}

```

```

public class Ecrivain2 extends Thread {
    private String texte;
    private Imprimeur1 imprim;
    public Ecrivain2(String t, Imprimeur1 i) {
        imprim=i;
        texte=t;
    }
    public void run() {
        for (int i=0;i<10;i++) {
            imprim.imprimer(texte);
            try {sleep((long)(Math.random() * 100));}
            catch (InterruptedException e) {}
        } // end for
        System.out.println("ecrivain de "+texte+" a fini");
    } // end run
}

```

```

public class PerroquetsMatheux20 {
    private int compteur;
    public static void main(String args[]) {
        new PerroquetsMatheux20();
    }
    public PerroquetsMatheux20 () {
        compteur = 1;
        Perroquet20 perroquetA
        = new Perroquet20("coco", 10);
        Perroquet20 perroquetB
        = new Perroquet20("bonjour", 10);
        perroquetA.start();
        perroquetB.start();
        try {
            perroquetA.join();
            perroquetB.join();
        }
        catch(InterruptedException e) {}
        System.out.println("compteur = "+compteur);
    } // end of constructor
}

```

```

class Perroquet20 extends Thread {
    private String ci = null;
    private int fois = 0;
    public Perroquet20 (String s, int i) {
        ci = s;
        fois = i;
    }
    public void repeter() {
        String repete = ci + " " + compteur;
        System.out.println(repete);
        compteur++;
        try{ Thread.sleep((int)(Math.random()*1000)); }
        catch(InterruptedException e) {}
    } // fin de repeter
    public void run() {
        for (int n=0; n<fois; n++)
            repeter();
    } // end of run
} // end of Perroquet20 class

```

```

} // end of PerroquetsMatheux20 class

```

```

public class PerroquetsMatheux21{
    private int compteur;
    public static void main(String args[]) {
        new PerroquetsMatheux21(); }
    public PerroquetsMatheux21() {
        compteur = 0;
        Perroquet21 perroquetA
        = new Perroquet21("coco", 10);
        Perroquet21 perroquetB
        = new Perroquet21("bonjour", 10);
        perroquetA.start();
        perroquetB.start();
        try {
            perroquetA.join();
            perroquetB.join();
        }
        catch(InterruptedException e) { }
        System.out.println("compteur = "+compteur);
    }
}

```

```

class Perroquet21 extends Thread {
    private String cri = null;
    private int fois = 0;
    public Perroquet21(String s, int i) {
        cri = s;
        fois = i; }
    public void repeter() {
        int valeur = compteur + 1;
        String repete = cri + " " + valeur;
        System.out.println(repete);
        try { Thread.sleep((int)(Math.random()*100)); }
        catch(InterruptedException e) { }
        compteur = valeur;
        try {Thread.sleep((int)(Math.random()*100));}
        catch(InterruptedException e) { }
    }
    public void run(){
        for (int n=0; n<fois; n++) repeter();
    }
}
}

```

```

public class PerroquetsMatheux21{
    private int compteur;
    public static void main(String args[]) {
        new PerroquetsMatheux21();
    }
    public PerroquetsMatheux21() {
        compteur = 0;
        Perroquet21 perroquetA
        = new Perroquet21("coco", 10);
        Perroquet21 perroquetB
        = new Perroquet21("bonjour", 10);
        perroquetA.start();
        perroquetB.start();
        try {
            perroquetA.join();
            perroquetB.join();
        }
        catch (InterruptedException e) { }
        System.out.println("compteur = "+compteur);
    }
}

```

Define a critical section

```

class Perroquet21 extends Thread {
    private String cri = null;
    private int fois = 0;
    public Perroquet21(String s, int i) {
        cri = s;
        fois = i;
    }
    public void repeter() {
        int valeur = compteur + 1;
        String repete = cri + " " + valeur;
        System.out.println(repete);
        try { Thread.sleep((int)(Math.random()*100)); }
        catch (InterruptedException e) { }
        compteur = valeur;
        try { Thread.sleep((int)(Math.random()*100)); }
        catch (InterruptedException e) { }
    }
    public void run() {
        for (int n=0; n<fois; n++) repeter();
    }
}

```

```

public class PerroquetsMatheux22 {
    private Compteur compteur;
    public static void main(String args[]) {
        new PerroquetsMatheux22();
    }
    public PerroquetsMatheux22() {
        compteur = new Compteur();
        Perroquet22 perroquetA= new Perroquet22("coco", 10);
        Perroquet22 perroquetB= new Perroquet22("bonjour", 10);
        perroquetA.start();
        perroquetB.start();
        try { perroquetA.join();
            perroquetB.join(); }
        catch (InterruptedException e) { }
        System.out.println("compteur = "+compteur.getValeur());
    }
}

```

```

class Compteur {
    private int valeur = 0;
    public int getValeur() { return valeur; }
    public void setValeur(int v) { valeur = v; }
}

```

```

class Perroquet22 extends Thread {
    private String cri = null;
    private int fois = 0;
    public Perroquet22(String s, int i) {
        cri = s;
        fois = i;
    }
    public void repeter() {
        synchronized (compteur) {
            int val = compteur.getValeur() + 1;
            String repete = cri + " " + val;
            System.out.println(repete);
            try { Thread.sleep((int)(Math.random()*100)); }
            catch (InterruptedException e) { }
            compteur.setValeur(val);
        }
        try { Thread.sleep((int)(Math.random()*100)); }
        catch (InterruptedException e) { }
    }
    public void run() {
        for (int n=0; n<fois; n++) repeter();
    }
}

```

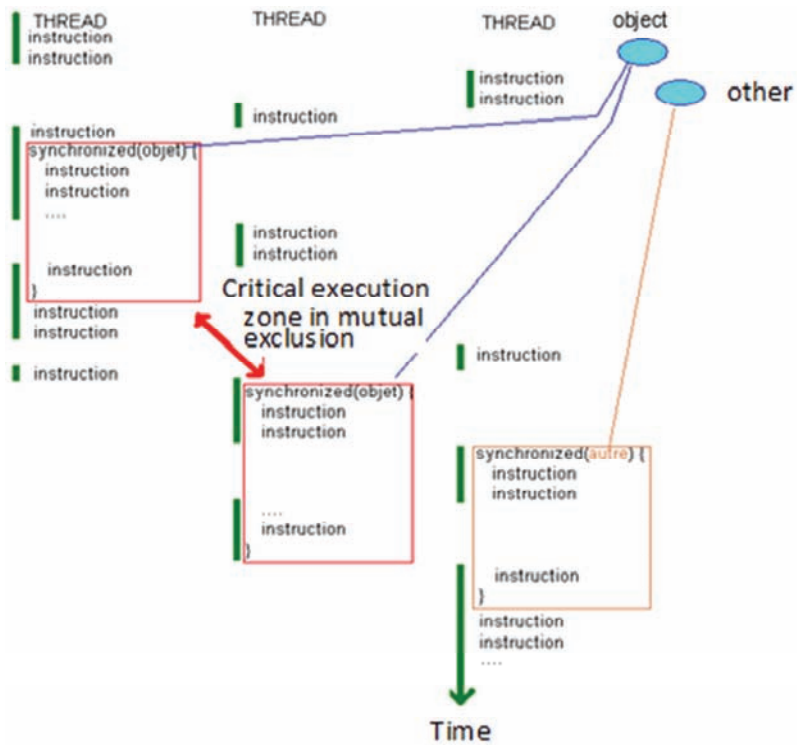



Figure 2.2. Critical section shared between three threads

```
class PerroquetsMatheux23 {  
    private Compteur compteur;  
    public static void main(String args[]) {  
        new PerroquetsMatheux23();  
    }  
    public PerroquetsMatheux23() {  
        compteur = new Compteur();  
        Perroquet23 perroquetA = new Perroquet23("coco", 10);  
        Perroquet23 perroquetB = new Perroquet23("bonjour", 10);  
        perroquetA.start();  
        perroquetB.start();  
        try {perroquetA.join(); perroquetB.join(); }  
        catch (InterruptedException e) { }  
        System.out.println("compteur = "+compteur.valeur);  
    }  
    class Compteur {  
        private int valeur = 0;  
        public synchronized int plus1() {  
            return ++valeur;  
        }  
    }  
}
```

```
class Perroquet23 extends Thread {  
    private String ca = null;  
    private int fois = 0;  
    public Perroquet23(String s, int i) {  
        ca = s;  
        fois = i; }  
    public void repeter() {  
        int val = compteur.plus1();  
        String repete = ca + " " + val;  
        System.out.println(repete);  
        try {Thread.sleep((int)(Math.random()*100));}  
        catch (InterruptedException e) { }  
    }  
    public void run() {  
        for (int n=0; n<fois; n++) repeter();  
    }  
}
```


Execution :

coco 1
bonjour 2
bonjour 3
coco 4
coco 5
bonjour 6
bonjour 7
coco 8
bonjour 9
coco 10
bonjour 11
coco 12
bonjour 13
bonjour 14
coco 15
bonjour 16
coco 17
bonjour 18
coco 19
coco 20
compteur = 20

Remark:

```
public synchronized int plus1() {  
    return ++valeur;  
}
```

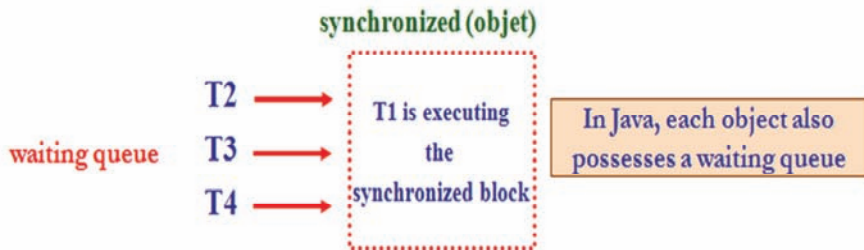
is equivalent to :

```
public int plus1(){  
    synchronized(this) {  
        return ++valeur;  
    }  
}
```

Synchronization slows down the whole execution; it is therefore preferable to limit the number of synchronized portions and their size (in instructions).

```
method(parameters) {  
    synchronized(this) {  
        instructions block  
    }  
    instructions block  
}
```

Only the code that is creating
an issue for concurrent access



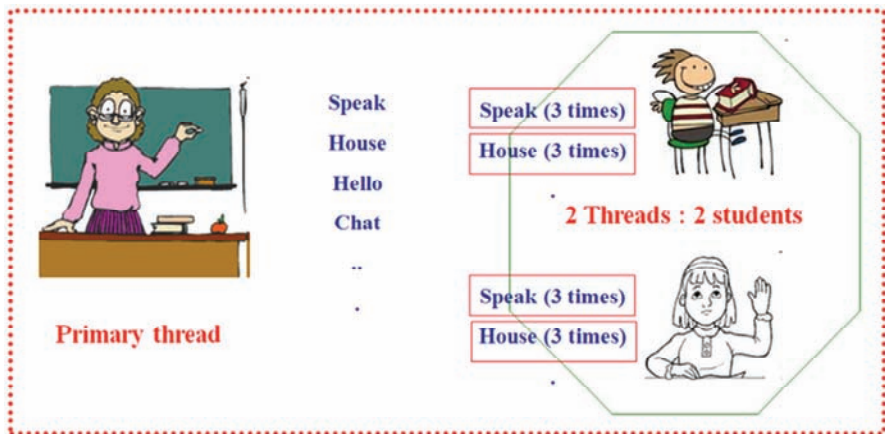


Figure 2.3. Consumer/producer example

```

public class EcoleDesPerroquets14 {
    static String mot = null;
    public static void main(String[] args) {
        Perroquet14 perroquet1 = new Perroquet14("coco");
        perroquet1.start();
        Perroquet14 perroquet2 = new Perroquet14("jaco");
        perroquet2.start();
        String reponse = null;
        do { mot = reponse;
            System.out.println
                ("nouveau mot pour perroquet ? (sinon non)");
            reponse = Saisie.litexte();
        }
        while (!reponse.equals("non"));
        System.exit(1);
    } // end main
} // end of EcoleDesPerroquets14

```

```

class Perroquet14 extends Thread {
    public Perroquet14(String n) {
        super(n);
    }
    public void repeter() {
        System.out.println(getName() + " "+EcoleDesPerroquets14.mot);
    }
    public void run() {
        while (true) { // tourne à l'infini
            while (EcoleDesPerroquets14.mot == null)
                try {Thread.sleep(2000);} catch (Exception e) { }
            for (int n=0; n<3; n++) repeter();
            try {Thread.sleep((int)(Math.random()*3000)); }
            catch (InterruptedException e) { }
        } // end of while true
    } // end of run
} // end of Perroquet14

```

```

public class EcoleDesPerroquets15 {
    public static void main(String[] args) {
        AuTableau autableau = new AuTableau();
        Perroquet15 perroquet1 = new Perroquet15("coco", autableau);
        perroquet1.start();
        Perroquet15 perroquet2 = new Perroquet15("jaco", autableau);
        perroquet2.start();
        String reponse = "bonjour";
        do {atableau.enseigner(reponse);
            System.out.println("nouveau mot pour perroquet ? (sinon non)");
            reponse = Saisie.lire();
        } while (!reponse.equals("non"));
        System.exit(1);
    } // end main
} // end class

```

```

class Perroquet15 extends Thread {
    private String cri;
    private String nom;
    private AuTableau autableau;
    public Perroquet15(String n, AuTableau a) {
        super(n); nom = n;
        autableau = a;
        cri = "";
    }
}

```

```

public void repeter() {
    System.out.println(nom + " " + cri);
}
public void run() {
    while (true) {
        cri = autableau.appendre();
        for (int n = 0; n < 3; n++) repeter();
    }
}

```

```

class AuTableau {
    private String motAappendre = null;
    synchronized String appendre() {
        try { wait(); } catch (Exception e) {}
        return motAappendre;
    }
    synchronized void enseigner (String mot) {
        motAappendre = mot;
        notifyAll();
    }
}

```

Used by the students

Used by the teacher

```

public class EcoleDesPerroquets16 {
    public static void main(String[] args) {
        AuTableau autableau = new AuTableau();
        Perroquet16 perroquet1= new Perroquet16("occo", autableau);
        perroquet1.start();
        Perroquet16 perroquet2 = new Perroquet16("jaco", autableau);
        perroquet2.start();
        String reponse = "bonjour";
        do {atableau.enseigner(reponse);
            System.out.println("nouveau mot pour perroquet ? (sinon non)");
            reponse = SaisieLitexte();
        } while (! reponse.equals("non"));
        System.exit(1);
    }
}

```

```

class Perroquet16 extends Thread {
    private String cn;
    private String nom;
    private AuTableau autableau;
    public Perroquet16(String n, AuTableau a) {
        super(n);
        nom = n;
        autableau = a; cn = "";
    }
}

```

```

    public void repeter() {
        System.out.println(nom + " " + cn);
    }
    public void run() {
        while (true) {
            cn = autableau.apprendre();
            for (int n=0; n<3; n++) repeter();
        }
    }
}

```

```

class AuTableau {
    private String motApprendre = null;
    synchronized String apprendre() {
        try {wait();} catch (Exception e) {}
        return motApprendre;
    }
    synchronized void enseigner (String mot) {
        motApprendre = mot;
        notify();
    }
}

```

```

class Station {
    public synchronized void attendreBus () {
        try { wait(); } catch (Exception e) {}
    }
    public synchronized void chargerUsagers ()
    { notifyAll (); }
}

class Usager extends Thread {
    private String nom ;
    private Station s ;
    private int heureArrivee ;
    public Usager (String nom, Station s, int heureArrivee) {
        this.nom = nom ;
        this.s = s ;
        this.heureArrivee = heureArrivee ;
    }
    public void run () {
        try { sleep (heureArrivee); }
        catch (InterruptedException e) {}
        System.out.println (nom + " arrive a la station");
        s.attendreBus () ;
        System.out.println (nom + " est monte dans le bus");
    }
}

```

Used by users (Usager class)

Used by the bus (Bus class)

```

class Bus extends Thread {
    private Station s ;
    private int heureArrivee ;
    public Bus (Station s, int heureArrivee) {
        this.s = s ;
        this.heureArrivee = heureArrivee ;
    }
    public void run () {
        try {
            sleep (heureArrivee);
        }
        catch (InterruptedException e) {}
        System.out.println ("Bus arrive a la station");
        s.chargerUsagers () ;
        try {sleep (500);}catch (InterruptedException e) {}
        System.out.println ("Bus repart de la station");
    }
}

```

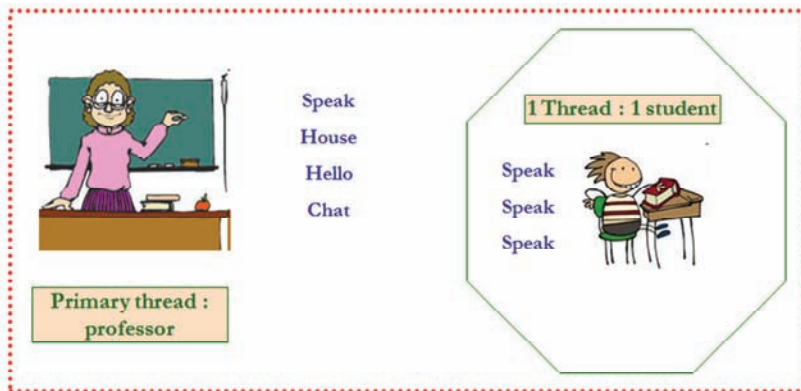
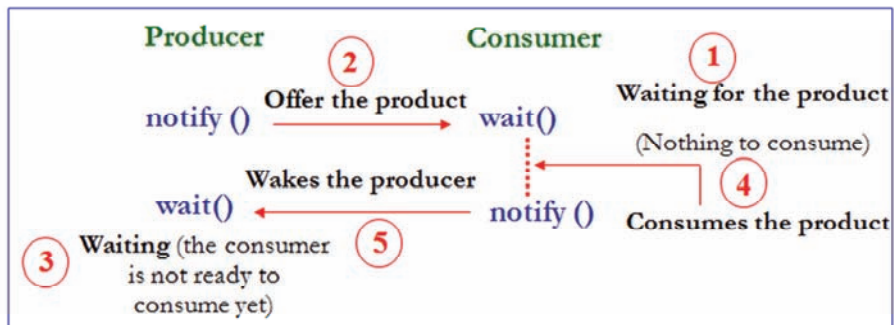
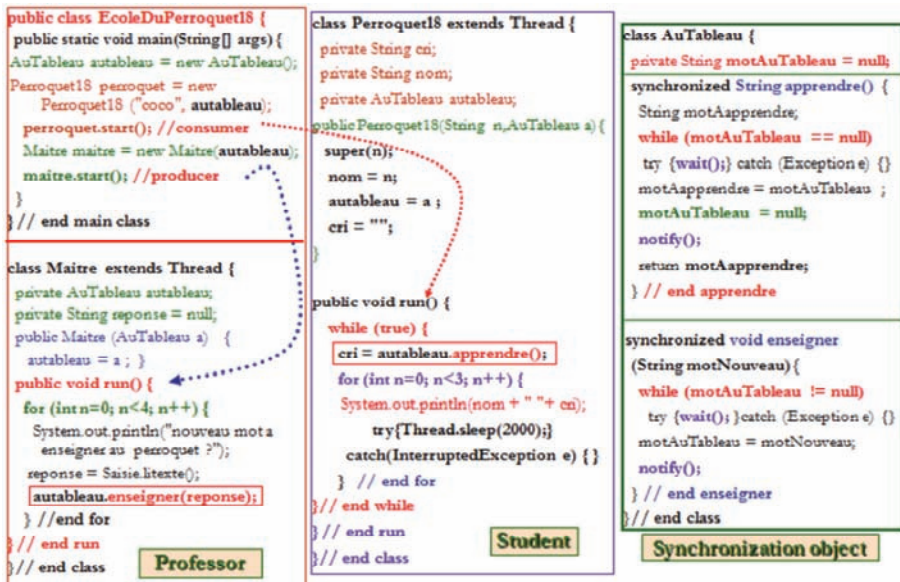
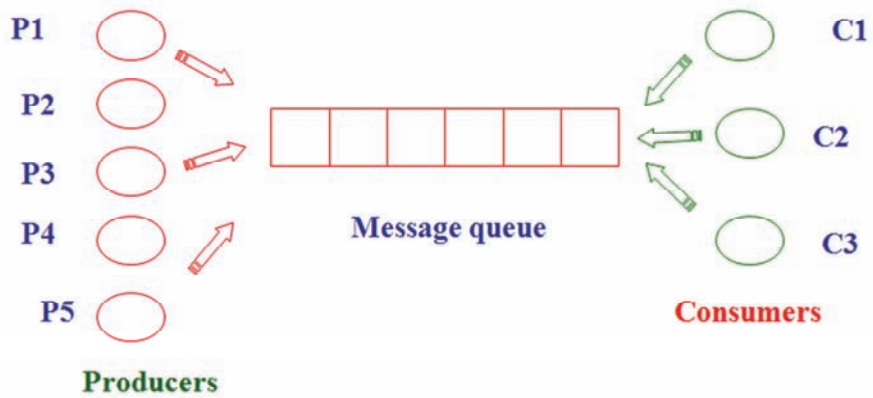


Figure 2.4. Classic Producer–Consumer pattern





```

class Semaphore {
    int counter;
    Semaphore (int init)
    {
        counter=init;
    }
    public synchronized void P(){
        counter--;
        if (counter<0) try{wait();}
        catch (InterruptedException e) {}
    }
    public synchronized void V(){
        counter++;
        if (counter<=0) notify();
    }
}

```

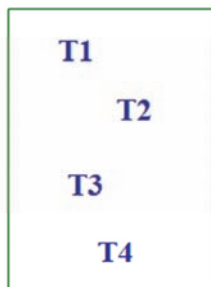
counter = 4
Critical section



The counter determines the maximum number of threads that can access the critical section

counter = -1

Critical section



T5

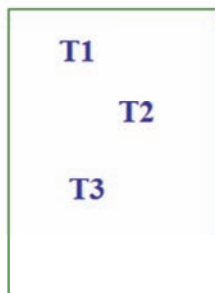
P0

wait ()

T5 is blocked because the maximum number of threads in the critical section is reached

counter = 0

Critical section



T5

wait ()

T4

V0

notify ()

counter = 0

Critical section

T1

T2

T3

T5

counter = 4

Critical section

After all threads have exited the critical section, the counter goes back to 4

```
import java.util.concurrent.Semaphore;
public class Car extends Thread{
```

```
    private int id;
    private Semaphore sem;
    public Car (int i, Semaphore s) {
        id = i;
        sem = s;
    }
}
```

```
private void outofLot() {
    System.out.println("Thread " + id + " is not in the
    parking lot");
    try{sleep((int)(Math.random()*1000));} catch
    (InterruptedException e){}
}
```

```
private void InOutParkingLot() {
    System.out.println("Thread " + id + " enters the
    parking lot");
    try{sleep((int)(Math.random()*2000));} catch
    (InterruptedException e){}
    System.out.println("Thread " + id + " exits the
    parking lot");
}
```

```
public void run() {
    horsParking();
    try {sem.acquire();} catch (Exception e){ }
    InOutParkingLot();
    sem.release();
}
```

```
public static void main(String[] args) {
    Semaphore sem = new Semaphore(3);
    Car [ ] p = new Car[5];
    for (int i = 0; i < 5; i++)
    {
        p[i] = new Car(i, sem);
        p[i].start();
    } // end for
} // end main
} // end Car
```

Chapter 3

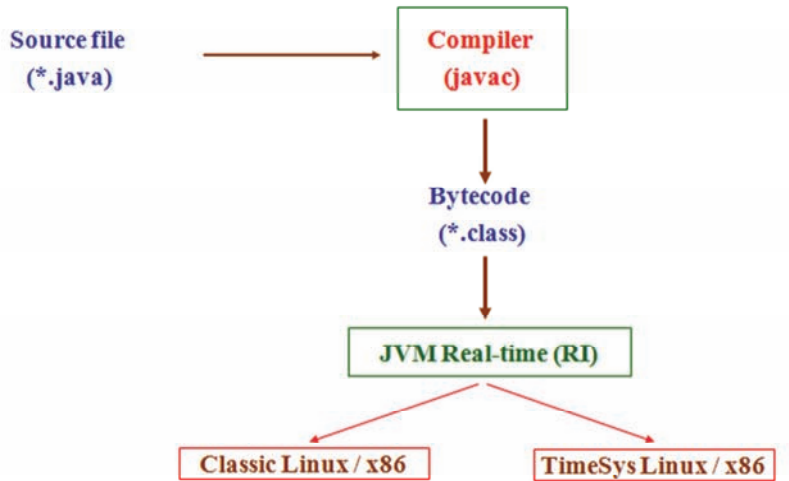


Figure 3.1. Reference implementation

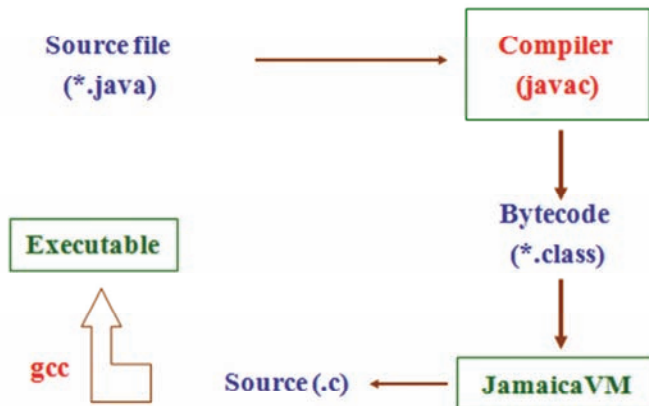


Figure 3.2. Implementation of JamaicaVM

Chapter 4

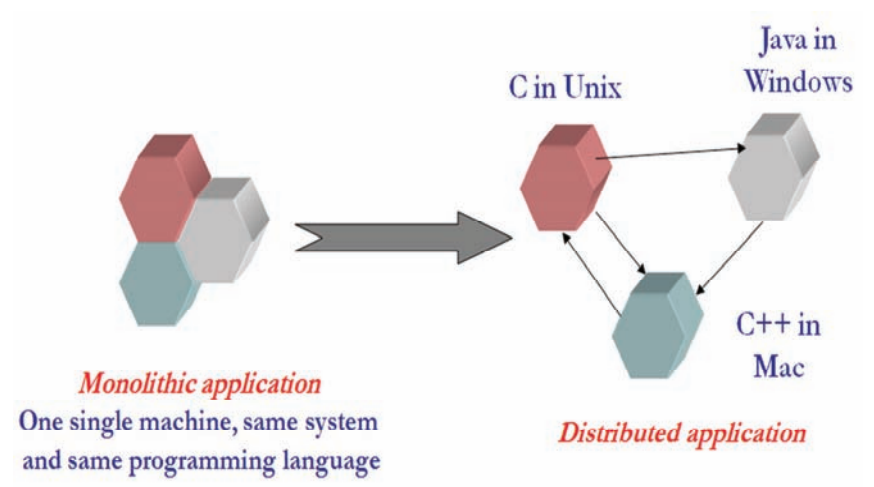


Figure 4.1. Monolithic application versus distributed application

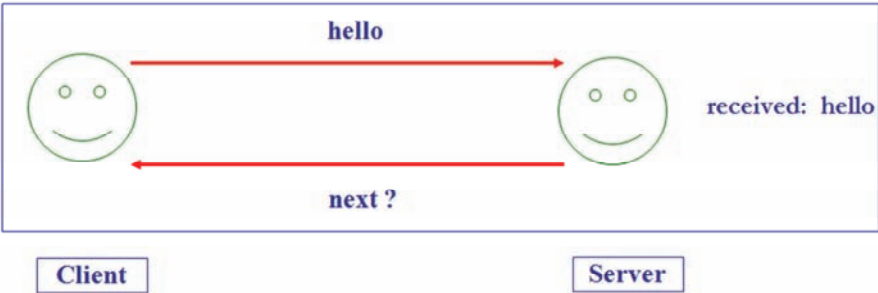


Figure 4.2. Example of communication with Sockets

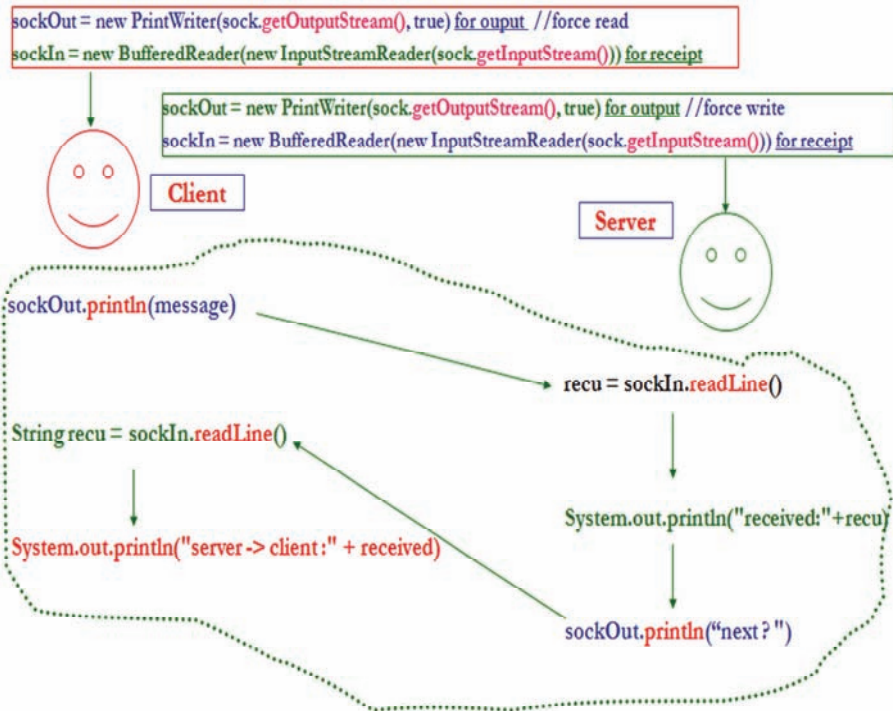


Figure 4.3. Diagram of communication with Sockets

```
import java.net.*;
import java.io.*;
class ServerStudent {
```

Server side

```
public static void main(String args[]) {
    Student[] tabStudent= {new Student ("A", "GL", 13), new Student ("B", "RSD", 12),
                           new Student ("C", "SIC", 14)};
    ServerSocket server=null;
    try {
        server =new ServerSocket(7777);
        while (true) {
            Socket sock = server.accept();
            System.out.println("connected");
            ObjectOutputStream sockOut = new ObjectOutputStream(sock.getOutputStream());
            BufferedReader sockIn = new BufferedReader(new InputStreamReader(sock.getInputStream()));
            String recu; while ((recu = sockIn.readLine()) != null) {
                System.out.println("received :"+recu);
                String name = recu.trim();
                for (int i=0;i<tabStudent.length;i++)
                    if (tabStudent[i].getNom().equals(name)) {sockOut.writeObject(tabStudent[i]);
                        break;
                    }
            }
            sockOut.close();
            sock.close();
        }
    } catch (IOException e) {try {server.close();} catch (IOException e2) {} }
} // endmain
} // endclass
```

Student table

Read the name sent by the client

Confirm receipt

Browse the table to find the student

Save the Student object in order to send it to the client

```

import java.io.*; import java.net.*;
public class ClientStudent {
    public static void main(String[] args) throws IOException {
        String hostName = "localhost";
        String NameStudent = "A";
        Socket sock = null;
        PrintWriter sockOut = null;
        ObjectInputStream sockIn = null;
        try {
            sock = new Socket(hostName, 7777);
            sockOut = new PrintWriter(sock.getOutputStream(), true);
            sockIn = new ObjectInputStream(sock.getInputStream());
        } catch (UnknownHostException e) { System.err.println("host cannot be reached : "+hostName); System.exit(1); }
        catch (IOException e) { System.err.println("cannot connect to: "+hostName); System.exit(1); }
        sockOut.println(NameStudent);
        try {
            Object recu = sockIn.readObject();
            if (recu == null) System.out.println("connection error");
            else { Student Student = (Student)recu;
                System.out.println("server -> client : " + Student);
            }
        } catch (ClassNotFoundException e) { System.err.println("Unknown class : "+hostName); System.exit(1); }
        sockOut.close();
        sockIn.close();
        sock.close();
    }
}

```

Client side

Send the name of the student to the server

Restore the student object received from the server

Display the characteristics of the student

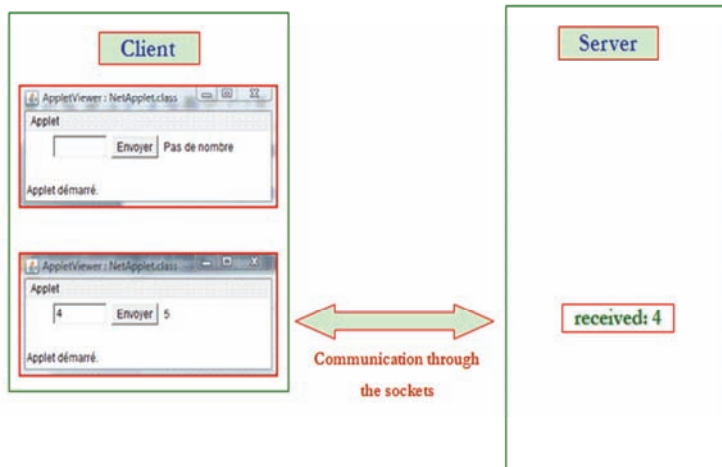


Figure 4.4. Communication between an applet and a server with the Sockets

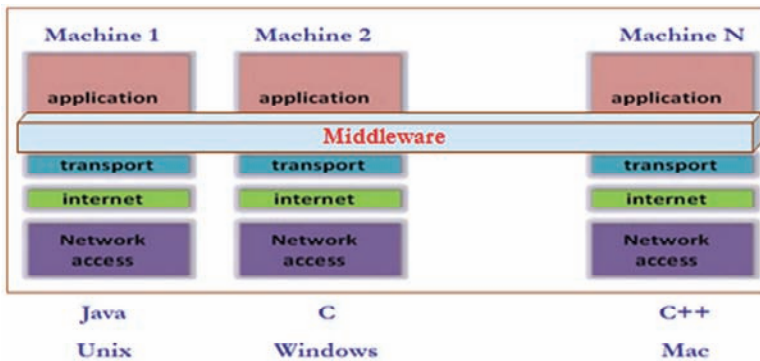


Figure 4.5. Position of Middleware in a network

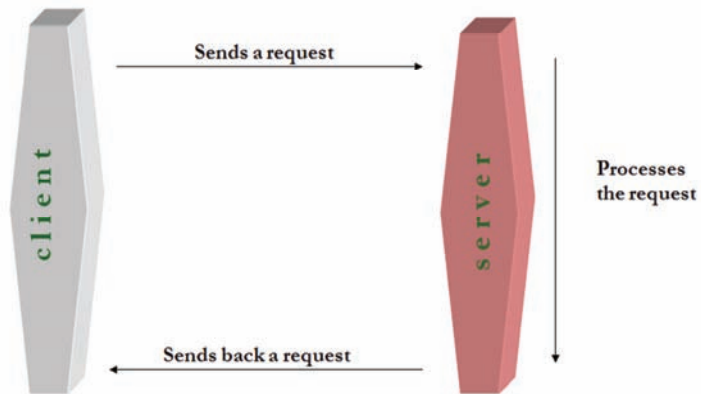


Figure 4.6. *RPC operation*

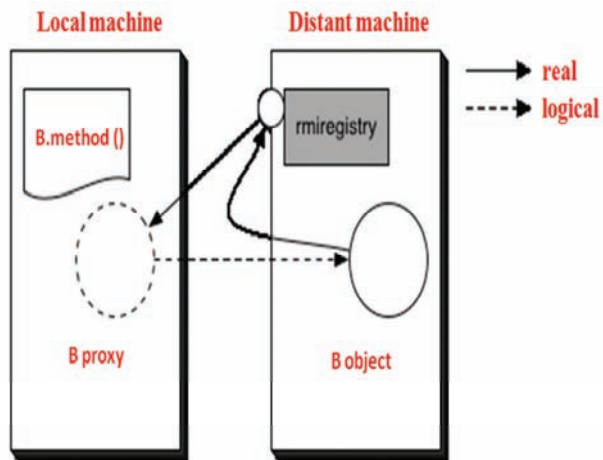


Figure 4.8. *Operation of an RMI application*

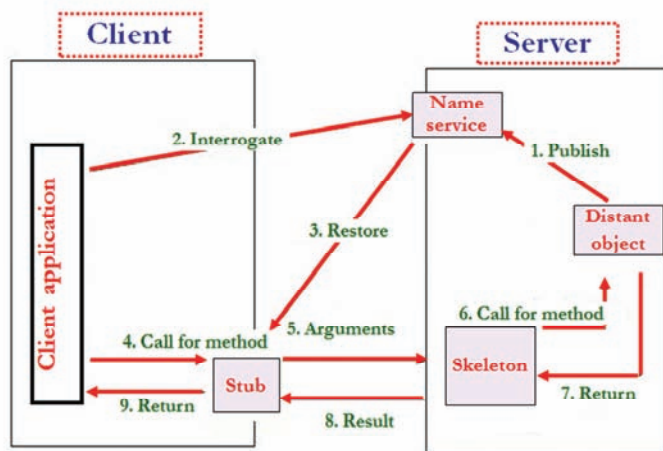


Figure 4.9. Stages of a distant method call

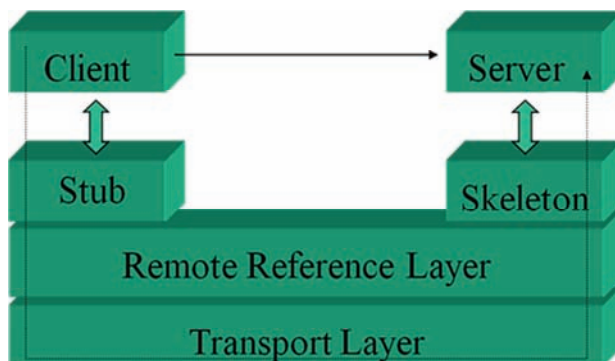


Figure 4.10. RMI architecture

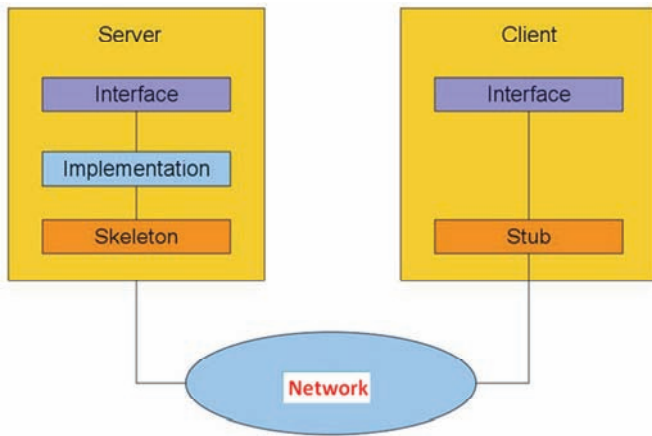


Figure 4.11. *Deploying RMI*

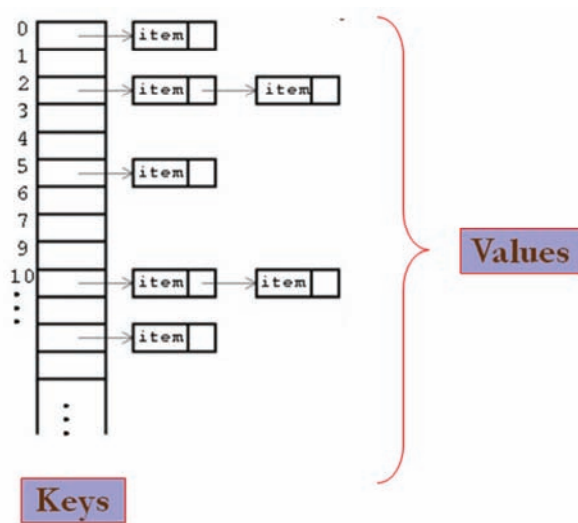


Figure 4.12. *rmiregistry*

```

import java.rmi.*;
import java.rmi.server.*;
import javax.swing.JTextArea;

public class Message extends UnicastRemoteObject implements Notify {

    private javax.swing.JTextArea textArea;
    private String name;

    public Message(JTextArea ta) throws RemoteException {
        textArea = ta;
    }

    public void joinMessage(String name) throws RemoteException
    {
        try {textArea.append(name + " has joined\n");}
        catch (Exception e) {System.out.println("error");}
    }
}

```

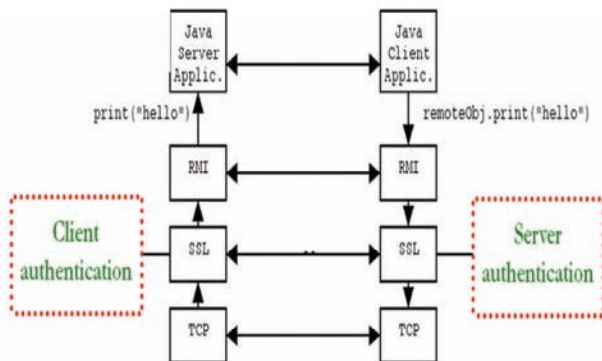


Figure 4.13. RMI with TLS

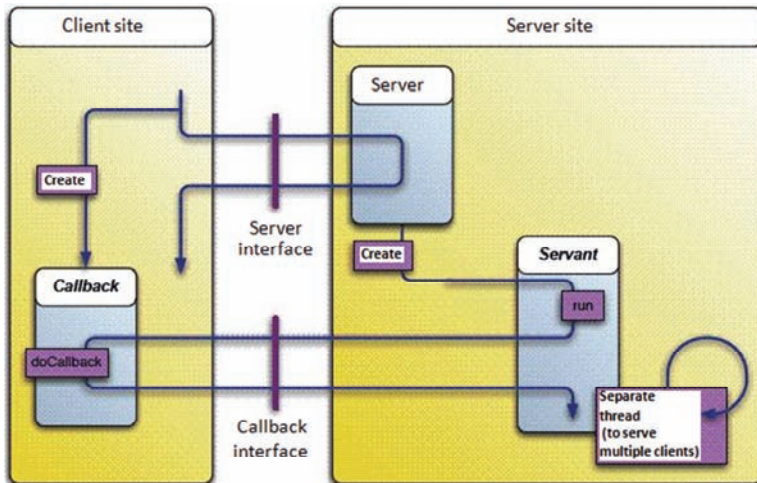


Figure 4.14. Callback in RMI

Client.java	Server.java
InterfaceServer.java	InterfaceServer.java
Callback.java	Servant.java
InterfaceCallback.java	InterfaceCallback.java

Figure 4.15. Necessary classes for a callback

Appendix

