



Tema 9. Paradigmas Avanzados de computación distribuida

1. Sistemas de colas de mensajes
2. Agentes móviles
3. Servicios de red
4. Espacios de objetos

Tema 9

Paradigmas Avanzados de Computación Distribuida

1



Sistemas de colas de mensajes

Tema 9

Paradigmas Avanzados de Computación Distribuida

2



Sistemas de colas de mensajes. Definición

- Lógica de mediación orientada a mensajes
 - MOM: message-oriented middleware
- Los mensajes se envían a un intermediario:
 - El sistema de mensajes
 - El intercambio es totalmente asíncrono:
 - emisor y receptor desacoplados
 - el emisor envía al sistema de mensajes
 - y sigue su ejecución
 - el sistema de mensajes reenvía a la cola del receptor
 - el receptor extrae los mensajes cuando quiere
- 2 modelos:
 - punto a punto
 - publicación-suscripción

Tema 9

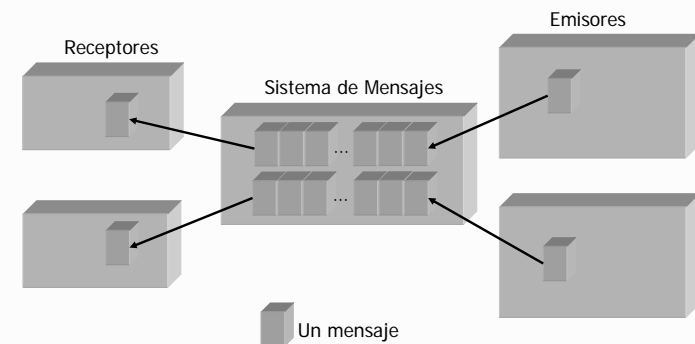
Paradigmas Avanzados de Computación Distribuida

3



Modelo de mensajes punto a punto

- Proporciona operaciones asíncronas
 - sin bloqueo en emisor ni en receptor
 - total desacoplo entre las operaciones de ambos



Tema 9

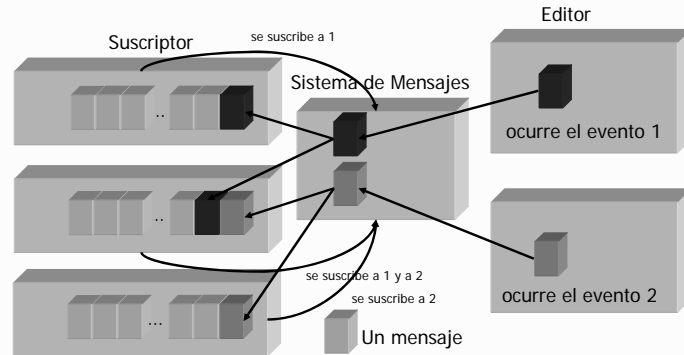
Paradigmas Avanzados de Computación Distribuida

4



Modelo de mensajes publicación/suscripción

- Los mensajes se asocian a temas o eventos
 - Los receptores se suscriben a dichos temas o eventos
 - Los emisores publican mensajes que anuncian dichos eventos
 - Comunicación en grupo desacoplada



Tema 9

Paradigmas Avanzados de Computación Distribuida

5



Ejemplos de sistemas comerciales de colas de mensajes

- WebSphere MQ* de IBM (antes *MQ*Series*)
- Message Queue* de Microsoft (*MSMQ*)
- Java Message Service* de Sun (*JMS*)
- Data Distribution Service* del OMG (*DDS*)
 - orientado a sistemas distribuidos de tiempo real
 - implementación: *NDDS* de RTI

Tema 9

Paradigmas Avanzados de Computación Distribuida

6



Java JMS

- Modelo punto a punto:
 - cada mensaje sólo tiene un consumidor
 - desacoplo temporal: se pueden recibir mensajes enviados cuando el consumidor no existía
 - el consumidor asiente la correcta recepción
 - el destino de los mensajes es de tipo *javax.jms.queue*
- Modelo publicación/suscripción:
 - cada mensaje puede tener múltiples consumidores
 - dependencia temporal: sólo se reciben los mensajes enviados con posterioridad a la suscripción
 - excepción: suscripciones duraderas
 - el destino de los mensajes es de tipo *javax.jms.topic*
- En ambos:
 - El consumo de mensajes puede ser:
 - síncrono: explícito mediante el método *receive*
 - asíncrono: registrando un *MessageListener*

Tema 9

Paradigmas Avanzados de Computación Distribuida

7



Emisor de mensajes con Java JMS

```
import javax.jms.*;
import javax.naming.*;

public class EmisorMensaje {
    public static void main (String[] args) {
        try {
            Context contextoJndi = new InitialContext();
            ConnectionFactory factoriaConexiones =
                (ConnectionFactory) contextoJndi.lookup("ConnectionFactory");
            Destination destino = (Destination) contextoJndi.lookup(args[0]);
            Connection conexion = factoriaConexiones.createConnection( );
            Session sesion = conexion.createSession(false, Session.AUTO_ACKNOWLEDGE);
            MessageProducer productor = sesion.createProducer(destino);
            TextMessage mensaje = sesion.createTextMessage();
            mensaje.setText("Hola Mundo");
            productor.send(mensaje);
            conexion.close();
        } catch (NamingException e) {
            System.out.println("Naming: " + e.getMessage());
        } catch (JMSException e) {
            System.out.println("JMS: " + e.getMessage());
        }
    }
}
```

Basado en los ejemplos del manual de la J2EE

Tema 9

Paradigmas Avanzados de Computación Distribuida

8



Receptor de mensajes síncrono con *Java JMS*

```
import javax.jms.*;
import javax.naming.*;

public class ReceptorMensajeSinc {
    public static void main (String[] args) {
        try {
            Context contextoJndi = new InitialContext( );
            ConnectionFactory factoriaConexiones =
                (ConnectionFactory) contextoJndi.lookup("ConnectionFactory");
            Destination origen = (Destination) contextoJndi.lookup(args[0]);
            Connection conexion = factoriaConexiones.createConnection();
            Session sesion = conexion.createSession(false, Session.AUTO_ACKNOWLEDGE);
            MessageConsumer consumidor = sesion.createConsumer(origen);
            conexion.start();
            Message m = consumidor.receive(0);
            TextMessage mensaje = (TextMessage) m;
            System.out.println("Leyendo Mensaje: " + mensaje.getText());
            conexion.close();
        } catch (NamingException e) {
            System.out.println("Naming: " + e.getMessage());
        } catch (JMSEException e) {
            System.out.println("JMS: " + e.getMessage());
        }
    }
}
```

Basado en los ejemplos del manual de la J2EE

Tema 9

Paradigmas Avanzados de Computación Distribuida

9



Receptor de mensajes asíncrono con *Java JMS*

```
import javax.jms.*;
import javax.naming.*;
import java.io.*;

public class ReceptorMensajeAsinc {
    public static void main (String[] args) {
        try {
            // Todo igual hasta la creación del "consumidor"
            // ...
            EscuchaTexto escucha = new EscuchaTexto();
            consumidor.setMessageListener(escucha);
            conexion.start();
            char resp = '\0';
            InputStreamReader entrada = new InputStreamReader(System.in);
            System.out.println("Para terminar, pulse 'x', " + "e <intro>");
            do { } while ((resp = (char) entrada.read()) != 'x');
            conexion.close();
        } catch (NamingException e) {
            System.out.println("Naming: " + e.getMessage());
        } catch (JMSEException e) {
            System.out.println("JMS: " + e.getMessage());
        } catch (IOException e) {
            System.out.println("E/S: " + e.getMessage());
        }
    }
}
```

Basado en los ejemplos del manual de la J2EE

Tema 9

Paradigmas Avanzados de Computación Distribuida

10



Escucha asíncrona de mensajes con *Java JMS*

```
import javax.jms.*;

public class EscuchaTexto implements MessageListener {
    public void onMessage(Message mensaje) {
        TextMessage msj = null;

        try {
            if (mensaje instanceof TextMessage) {
                msj = (TextMessage) mensaje;
                System.out.println("Leyendo mensaje: " + msj.getText());
            } else {
                System.out.println("El mensaje no es un TextMessage");
            }
        } catch (JMSEException e) {
            System.out.println("JMSEException en onMessage(): " + e.toString());
        } catch (Throwable t) {
            System.out.println("Exception en onMessage(): " + t.getMessage());
        }
    }
}
```

Basado en los ejemplos del manual de la J2EE

Tema 9

Paradigmas Avanzados de Computación Distribuida

11



Agentes móviles

Tema 9

Paradigmas Avanzados de Computación Distribuida

12



Programación Orientada a Agentes

- En sistemas complejos, el uso de metodologías Top-Down suele ser caro, difícil de mantener y de escalar
- La aparición de múltiples elementos como asistentes del usuario (PDAs, teléfonos móviles, domótica, etc.) incrementa la necesidad manejarlos como entes autónomos
- Además, en estos contextos, el uso de sistemas centralizados es inviable
 - por el contrario, los elementos deben incorporar cierta "inteligencia" para la toma de decisiones
- No hay una definición única y aceptada de lo que es un agente
 - Único punto de acuerdo \Rightarrow autonomía
- En la POO los objetos no tienen autonomía, actúan o responden cuando son invocados a través de métodos.
- En la POA los agentes deciden cuándo y cómo deben responder cuando reciben peticiones

Tema 9

Paradigmas Avanzados de Computación Distribuida

13



Agentes Inteligentes

- Agente inteligente sería aquél capaz de tener un comportamiento autónomo y flexible.
- Flexible implica tres propiedades relacionadas con el cumplimiento de sus objetivos:
 - Reactividad: reaccionan a cambios en el entorno
 - Pro-actividad: toman la iniciativa cuando es necesario
 - Habilidad social: para interactuar con otros agentes o humanos
- Estas propiedades implican capacidades de: aprendizaje, cooperación, negociación e incluso movilidad

Tema 9

Paradigmas Avanzados de Computación Distribuida

14



Agentes Móviles. Definición. 1

- La idea subyacente en los agentes móviles es que se trata de programas que viajan por la red con su código y su estado
- Pueden interrumpirse en un punto, viajar a otra máquina y continuar su ejecución exactamente en el punto en el que lo dejaron
- Java es el lenguaje más usado en sus implementaciones
- Tienen un problema grave, la seguridad que afecta al anfitrión y al propio agente móvil
- El principal desafío es proteger al agente móvil de un anfitrión malicioso

Tema 9

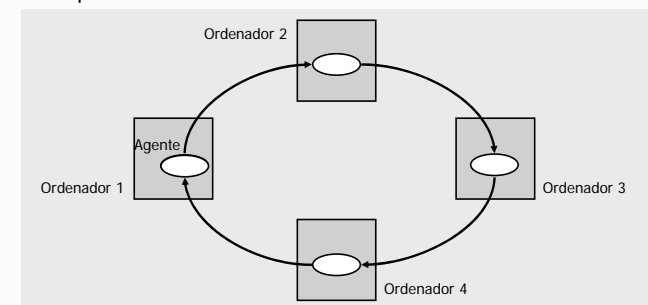
Paradigmas Avanzados de Computación Distribuida

15



Agentes móviles. Definición. 2

- Agente: proceso independiente que se ejecuta en representación del usuario
- Agente móvil:
 - puede viajar de una máquina a otra autónomamente
 - puede continuar en ausencia del usuario



Tema 9

Paradigmas Avanzados de Computación Distribuida

16



Agentes móviles. Arquitectura básica

- Un agente es un objeto que se puede alinear
 - (en Java: *serializable*)
- Contenido habitual de un agente móvil:
 - identidad del agente
 - itinerario: ordenadores que debe visitar
 - lógica y datos: propios de su tarea
- Funcionamiento:
 - en cada parada, llega a un servidor de agentes
 - a través de él, accede a los recursos locales
 - mediante un servicio de directorio localiza al siguiente servidor
 - se serializa y, con la ayuda del siguiente servidor,
 - se transporta al siguiente ordenador
- Implementaciones comerciales:
 - Aglets, Concordia, Grasshopper

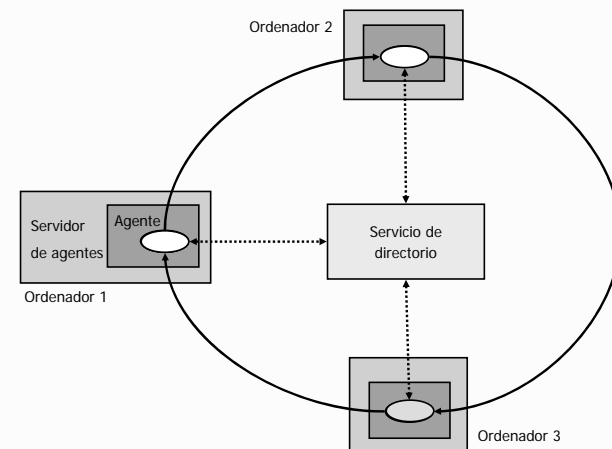
Tema 9

Paradigmas Avanzados de Computación Distribuida

17



Agentes móviles. Arquitectura básica. 2



Tema 9

Paradigmas Avanzados de Computación Distribuida

18



Interfaces *Java* para agentes móviles

- Interfaz Java de un agente

```
import java.io.Serializable;

public interface InterfazAgente extends Serializable {
    void ejecuta ();
}
```

Tomado de M.L. Liu – Computación Distribuida

- Interfaz Java RMI del servidor de agentes

```
import java.rmi.*;

public interface InterfazServidor extends Remote {
    public void recibe (Agente h) throws RemoteException;
}
```

Tomado de M.L. Liu – Computación Distribuida

Tema 9

Paradigmas Avanzados de Computación Distribuida

19



Implementación de un agente móvil en *Java*. 1

```
import java.io.*;
import java.util.*;
import java.rmi.*;

public class Agente implements InterfazAgente {
    int indiceNodo;
    String nombre;
    Vector listaNodos;

    public Agente (String miNombre, Vector laListaNodos) {
        nombre = miNombre;
        listaNodos = laListaNodos;
        indiceNodo = 0;
    }

    static void dormir (double tiempo) {
        try { Thread.sleep ((long) (tiempo*1000.0)); }
        catch (InterruptedException e) {
            System.out.println("Excepción en dormir" + e.getMessage());
        }
    }
    ...
}
```

Tomado de M.L. Liu – Computación Distribuida

Tema 9

Paradigmas Avanzados de Computación Distribuida

20



Implementación de un agente móvil en *Java*. 2

```

...
public void ejecuta ( ) {
    dormir (2);
    System.out.println("Soy el agente " + nombre);
    indiceNodo++;
    if (indiceNodo < listaNodos.size( )) {
        String siguiente = (String) listaNodos.elementAt(indiceNodo);
        dormir (5);
        try {
            InterfazServidor servidor = (InterfazServidor)
                Naming.lookup("rmi://" + siguiente + "/ServidorAgentes");
            System.out.println("Encontrado Servidor de Agentes en " + siguiente);
            dormir (5);
            servidor.recibe (this);
        } catch (Exception e) {
            System.out.println("Excepción en el ejecuta del Agente: " +
                e.getMessage());
        }
    } else {
        dormir (5);
        System.out.println("El agente ha regresado a casa");
        dormir (5);
    }
}
}

```

Tomado de M.L. Liu – Computación Distribuida

Tema 9

Paradigmas Avanzados de Computación Distribuida

21



Implementación de un servidor de agentes en *Java*

```

import java.net.*;
import java.rmi.*;

public class ServidorAgentes {

    public static void main (String args[ ]) {
        try {
            System.setSecurityManager(new RMISecurityManager());
            SirvienteAgentes sirviente = new SirvienteAgentes();
            String miNombre = InetAddress.getLocalHost().getHostName();

            Naming.rebind("//" + miNombre + "/ServidorAgentes", sirviente);
            System.out.println("Servidor de agentes en " + miNombre +
                " listo. ");
        } catch (Exception e) {
            System.out.println("Excepción en el main del Servidor: " +
                e.getMessage());
        }
    }
}

```

Tomado de M.L. Liu – Computación Distribuida

Tema 9

Paradigmas Avanzados de Computación Distribuida

22



Implementación de un sirviente de agentes en *Java*

```

import java.rmi.*;
import java.rmi.server.*;

public class SirvienteAgentes extends UnicastRemoteObject
    implements InterfazServidor {

    public SirvienteAgentes ( ) throws RemoteException {
        super ( );
    }

    static void dormir (double tiempo) { idéntico al del Agente }

    public void recibe (Agente agente) throws RemoteException {
        dormir (3);
        System.out.println ("*** El agente " + agente.nombre +
            " ha llegado.");
        agente.ejecuta( );
    }
}

```

Tomado de M.L. Liu – Computación Distribuida

Tema 9

Paradigmas Avanzados de Computación Distribuida

23



Implementación de un cliente de agentes en *Java*

```

import java.io.*;
import java.util.*;
import java.rmi.*;

public class ClienteAgentes {

    public static void main (String args[ ]) {
        System.setSecurityManager(new RMISecurityManager());
        try {
            Vector listaNodos = new Vector( );
            for (int i=0; i < args.length; i++)
                listaNodos.addElement(args[i]);
            InterfazServidor servidor = (InterfazServidor)
                Naming.lookup("rmi://" + args[0] + "/ServidorAgentes");
            System.out.println("servidor en " + args[0] + " encontrado. ");
            System.out.println("*** Buen viaje, agente.");
            Agente agente = new Agente ("Austin Powers", listaNodos);
            servidor.recibe(agente);
            System.out.println("*** Buen trabajo, agente.");
        } catch (Exception e) {
            System.out.println("Excepción en el main del Cliente" +
                e.getMessage());
        }
    }
}

```

Tomado de M.L. Liu – Computación Distribuida

Tema 9

Paradigmas Avanzados de Computación Distribuida

24



Características de los agentes móviles

- **Ventajas**
 - uso eficiente de los canales de comunicación
 - no necesidad de transferir grandes volúmenes de datos
 - bien adaptados a enlaces inalámbricos
 - uso de dispositivos portátiles y de bajo coste
 - ejecución desacoplada de nodo de origen
 - puede saltarse nodos en fallo o buscar *nodos refugio*
 - ejecución asíncrona y descentralizada
 - sin necesidad de coordinador
- **Peligro: riesgos de seguridad**
 - para los nodos y para los agentes móviles
 - contramedidas:
 - autenticación, cifrado, control de acceso a recursos

Tema 9

Paradigmas Avanzados de Computación Distribuida

25



Servicios de red

Tema 9

Paradigmas Avanzados de Computación Distribuida

26



Servicios de red

- **Red = infraestructura para servicios**
 - federación de proveedores y consumidores de servicios
 - los servicios incluyen:
 - aplicaciones, ficheros, bases de datos, ...
 - dispositivos móviles, impresoras, ...
 - los servicios se añaden y eliminan de forma autónoma
 - los clientes usan uno o varios servicios
 - localizan cuáles están disponibles a través de un servicio de directorio
- **Ejemplos:**
 - protocolo *SOAP (Simple Object Access Protocol)*
 - *Jini*:
 - basado en *Java RMI*
 - posibilidad de usar alquileres (leases)
 - mediante un gestor de renovación de alquileres

Tema 9

Paradigmas Avanzados de Computación Distribuida

27



Interfaz *Java RMI* para un servidor de *Jini*

```
import java.rmi.*;

public interface InterfazServidorHola extends Remote {
    public String Hola() throws RemoteException;
}
```

Tomado de Nuggets de Noel Enele

Tema 9

Paradigmas Avanzados de Computación Distribuida

28



Implementación de un servidor en *Jini*

```
import java.rmi.*;
import net.jini.core.entry.*;
import net.jini.lookup.entry.*;
import net.jini.discovery.*;
import net.jini.lookup.*;
import net.jini.lease.*;

public class ServidorHola {
    public static void main (String[] args) {
        try {
            System.setSecurityManager (new RMISecurityManager ());
            Entry[ ] conjAtributos = new Entry[1];
            conjAtributos[0] = new Name ("ServidorHola");
            SirvienteHola sirvienteHola = new SirvienteHola ();
            DiscoveryManagement gestorDescub = null;
            JoinManager gestorUnion = new JoinManager (sirvienteHola,
                conjAtributos, sirvienteHola,
                gestorDescub, new LeaseRenewalManager());
        } catch (Exception e) {
            System.out.println("Excepción en el main del Servidor: " +
                e.getMessage());
        }
    }
}
```

Tomado de Nuggets de Noel Enete

Tema 9

Paradigmas Avanzados de Computación Distribuida

29



Implementación de un sirviente en *Jini*

```
import java.rmi.*;
import java.rmi.server.*;
import net.jini.lookup.*;
import net.jini.core.lookup.*;

public class SirvienteHola extends UnicastRemoteObject
    implements InterfazServidorHola, ServiceIDListener {

    public SirvienteHola () throws RemoteException {
        super ();
    }

    public void serviceIDNotify (ServiceID sidIn) {
        System.out.println ("servidor: recibido ServiceID: " + sidIn);
    }

    public String Hola () throws RemoteException {
        return ("Hola Mundo desde ServidorHola!");
    }
}
```

Tomado de Nuggets de Noel Enete

Tema 9

Paradigmas Avanzados de Computación Distribuida

30



Implementación de un cliente en *Jini*

```
import net.jini.core.entry.*;
import net.jini.core.lookup.*;
import net.jini.core.discovery.*;
import net.jini.lookup.entry.*;
import java.rmi.*;

class ClienteHola {
    public static void main (String[ ] args) {
        try {
            System.setSecurityManager (new RMISecurityManager ());
            LookupLocator buscador = new LookupLocator ("jini://localhost:8081");
            ServiceRegistrar registrador = buscador.getRegistrar ();
            Entry[ ] conjAtributos = new Entry[1];
            conjAtributos[0] = new Name ("ServidorHola");
            ServiceTemplate plantilla =
                new ServiceTemplate (null, null, conjAtributos);
            InterfazServidorHola servidor = (InterfazServidorHola)
                registrador.lookup(plantilla);
            System.out.println (servidor.Hola());
        } catch (Exception e) {
            System.out.println("Excepción en el main del Cliente: " +
                e.getMessage());
        }
    }
}
```

Tomado de Nuggets de Noel Enete

Tema 9

Paradigmas Avanzados de Computación Distribuida

31



Espacios de objetos

Tema 9

Paradigmas Avanzados de Computación Distribuida

32



Espacios de objetos

- Origen:
 - Espacio de Tuplas de *Linda*
 - secuencias tipadas de datos en memoria compartida distribuida con operaciones para insertarlas y extraerlas
- Espacio:
 - repositorio de objetos compartido
- Cooperación entre procesos
 - a través de objetos en uno o más espacios
 - desacoplo en el espacio y en el tiempo
 - no se ofrece a los procesos operaciones para modificar los objetos ni para invocar sus métodos
 - modificación = extracción + actualización + inserción
- Ejemplo: API de *JavaSpaces* (parte de *Jini*)
 - los objetos compartidos implementan la interfaz *Entry*

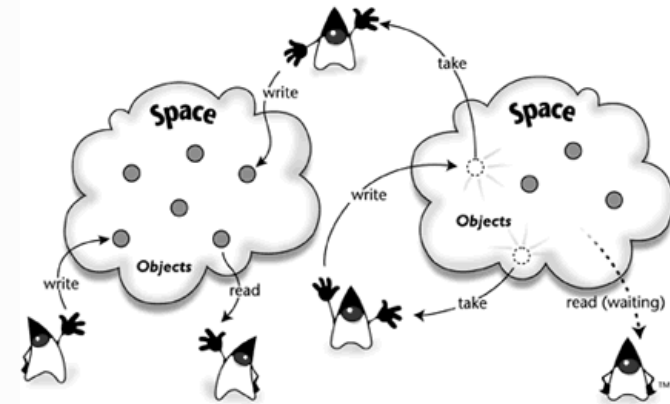
Tema 9

Paradigmas Avanzados de Computación Distribuida

33



El modelo de espacio de objetos de *JavaSpaces*



Tema 9

Paradigmas Avanzados de Computación Distribuida

34



Un objeto de espacio de *JavaSpaces*

```
import net.jini.core.entry.Entry;

public class ObjetoCompartido implements Entry {
    public String contenido;
    public Integer contador;

    public ObjetoCompartido() {
    }

    public ObjetoCompartido(String contenido, int valInicial) {
        this.contenido = contenido;
        contador = new Integer(valInicial);
    }

    public String toString() {
        return contenido + " modificado " + contador + " veces.";
    }

    public void incrementa() {
        contador = new Integer(contador.intValue() + 1);
    }
}
```

Tomado de Freeman et al. – *JavaSpaces: Principles, Patterns and Practice*

Tema 9

Paradigmas Avanzados de Computación Distribuida

35



Un programa que inicializa un objeto de *JavaSpaces*

```
import net.jini.core.lease.Lease;
import net.jini.space.JavaSpace;

public class HolaMundo {
    public static void main(String[] args) {
        try {
            ObjetoCompartido miObjeto = new ObjetoCompartido("Hola Mundo", 0);

            JavaSpace espacio = AccesoEspacio.buscaEspacio();
            espacio.write(miObjeto, null, Lease.FOREVER);

            ObjetoCompartido plantilla = new ObjetoCompartido();
            for (;;) {
                ObjetoCompartido valor = (ObjetoCompartido)
                    espacio.read(plantilla, null, Long.MAX_VALUE);
                System.out.println(valor);
                Thread.sleep(1000);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Tomado de Freeman et al. – *JavaSpaces: Principles, Patterns and Practice*

Tema 9

Paradigmas Avanzados de Computación Distribuida

36



Un cliente que accede a un objeto de *JavaSpaces*

```
import net.jini.core.lease.Lease;
import net.jini.space.JavaSpace;

public class ClienteHolaMundo {
    public static void main(String[] args) {
        try {
            JavaSpace espacio = AccesoEspacio.buscaEspacio();

            ObjetoCompartido plantilla = new ObjetoCompartido();
            for (;;) {
                ObjetoCompartido valor = (ObjetoCompartido)
                    espacio.take(plantilla, null, Long.MAX_VALUE);
                valor.incrementa();
                espacio.write(valor, null, Lease.FOREVER);
                Thread.sleep(1000);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Tomado de Freeman et al. - *JavaSpaces: Principles, Patterns and Practice*