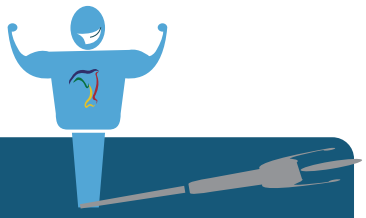
autentia

Avenida de Castilla,1 - Edificio Best Point - Oficina 21B
28830 San Fernando de Henares (Madrid)
tel./fax: +34 91 675 33 06
info@autentia.com - **www.autentia.com**

¿**Qué ofrece** Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.
Ese apoyo que siempre quiso tener...

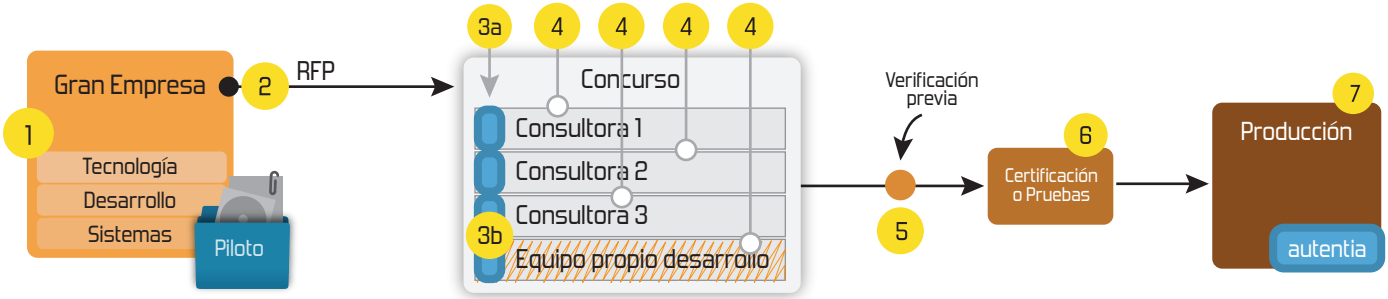
1. Desarrollo de componentes y proyectos a medida



2. Auditoría de código y recomendaciones de mejora

3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.



4. Cursos de formación (impartidos por desarrolladores en activo)

Spring MVC, JSF-PrimeFaces /RichFaces, HTML5, CSS3, JavaScript-jQuery

Gestor portales (Liferay)
Gestor de contenidos (Alfresco)
Aplicaciones híbridas

Control de autenticación y acceso (Spring Security)
UDDI
Web Services
Rest Services
Social SSO
SSO (Cas)

JPA-Hibernate, MyBatis
Motor de búsqueda empresarial (Solr)
ETL (Talend)

Dirección de Proyectos Informáticos.
Metodologías ágiles
Patrones de diseño
TDD

Tareas programadas (Quartz)
Gestor documental (Alfresco)
Inversión de control (Spring)

BPM (jBPM o Bonita)
Generación de informes (JasperReport)
ESB (Open ESB)

Compartimos nuestro conocimiento en:
www.adictosaltrabajo.com

Para más información visítenos en:
www.autentia.com

Tutorial desarrollado por:
Roberto Canales Mora 2003-2004.

Coste de Envio de mensajes de Log

Cuando desarrollamos una aplicación, muchas veces nos planteamos donde debemos escribir la información de log.

Si mostramos mensajes por la **consola** estandar (por pantalla)..... es cómodo pero es muy lento (aunque siempre podemos redirigir l salida a un fichero)

Podemos tomar la decisión de escribirlos en un **fichero local.**.... el problema puede ser, ¿quien limpia luego esos ficheros?. Si trabajamos en entornos serios, las herramientas de monitorización y backup pueden hacer este trabajo por nosotros ... pero no nos podemos despistar.

Podemos decidir enviar los mensajes a otra máquina y que esta sea quien los guarde. Si decidimos hacelo via **TCP**, debemos tener cuidado con controlar bien esta comunicación porque sería una pena que, porque fallase el sistema de Log, se parase nuestra aplicació

Si usamos **UDP** (comunicaciones ni orientadas a conexion), tenemos que tener en cuenta que que los paquetes no estan garantizados que lleguen.

Otra posibilidad, es utilizar **JMS**, es decir, mensajería .... y luego procesar esos mensajes cuando deseemos.

Pero ¿alguna vez os habeis parado a pensar los que cuesta escribir cada uno de los mensajes por distintos medios?

Os vamos a mostrar distintos ejemplos para que tengais una referencia.

Mensajes por pantalla y en fichero

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

/**
 *
 * @author  Roberto Canales
 */
public class velocidadLog {

   /** Creates a new instance of velocidadLog */
   public static void main(String [] cadenas)
   {
      long inicio = System.currentTimeMillis();

      for(int i=0; i<1000;i++)
      {
         System.out.println(i + " La hora es " + new Date().toString());
      }

        long fin = System.currentTimeMillis() - inicio;
        System.out.println("El tiempo transcurrido es " + fin + " milisegundos");

        try
        {
           FileWriter escritor = new FileWriter("logrob.txt",true);
```

```
          inicio = System.currentTimeMillis();
          for(int i=0; i<1000;i++)
          {
              escritor.write(i + " La hora es " + new Date().toString());
              escritor.flush();
          }
          escritor.close();
        }
        catch(Exception e)
        {
          System.out.println("Error a escribir en fichero" + e.getMessage());
        }

        fin = System.currentTimeMillis() - inicio;
        System.out.println("El tiempo transcurrido es " + fin + " milisegundos");
    }
}
```

Si ejecutamos este código ... la respuesta que optenemos es

```
.....
995 La hora es Fri Aug 22 20:35:44 CEST 2003
996 La hora es Fri Aug 22 20:35:44 CEST 2003
997 La hora es Fri Aug 22 20:35:44 CEST 2003
998 La hora es Fri Aug 22 20:35:44 CEST 2003
999 La hora es Fri Aug 22 20:35:44 CEST 2003
El tiempo transcurrido es 2994 milisegundos
El tiempo transcurrido es 111 milisegundos
```

Es decir, ha costado 25 veces más escribir en pantalla que en disco.

Escritura UDP

Si creamos un pequeño programa cliente UDP (que envia los mensajes de Log) .. y un servidor ..... tenemos que estar seguros de los que estamos haciendo ...

Programa cliente

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

class clienteUPD
{
      public static void main(String[] args)
      {
         try
         {
               String cadena = null;
               byte[] paqueteBytes = null;
               DatagramPacket paqueteUDP = null;

               InetAddress direccion = InetAddress.getByName("localhost") ;

               long inicio = System.currentTimeMillis();

               for(int i=0;i<1000;i++)
               {
                  DatagramSocket socketUDP = new DatagramSocket() ;
                  cadena = i + " La hora es " + new Date().toString();
                  paqueteBytes = cadena.getBytes() ;
                  paqueteUDP = new DatagramPacket(paqueteBytes, paqueteBytes.length,direccion,8031);
                  socketUDP.send(paqueteUDP) ;
                  socketUDP.close() ;

               }

               long fin = System.currentTimeMillis() - inicio;
               System.out.println("El tiempo transcurrido es " + fin + " milisegundos");
           }
         catch (IOException e)
         {
               System.out.println(e) ;
         }

      }
```

```
}
```

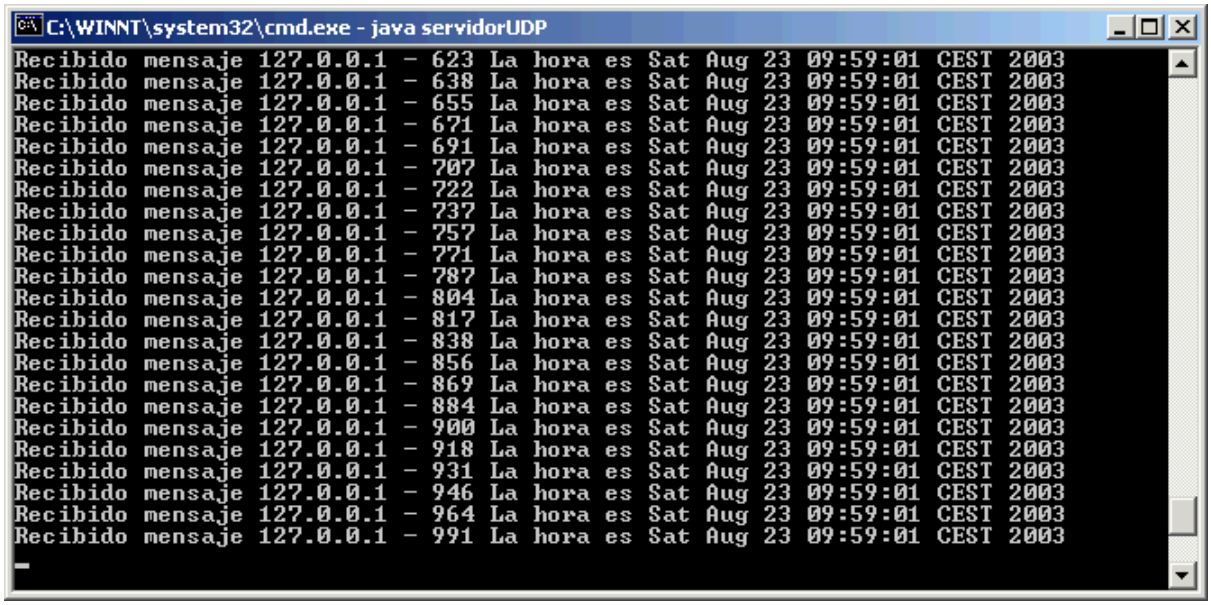
Si lo ejecutamos nos dice

El tiempo transcurrido es 1232 milisegundos

Y ahora el servidor

```
import java.net.* ;
import java.io.* ;

public class servidorUDP
{
   public static void main(String[] args)
   {
      try
      {
         int tamaPaquete = 1024;
         DatagramSocket s = new DatagramSocket(8031) ;

         while (true)
         {
               DatagramPacket datagramaUDP =  new DatagramPacket( new byte [tamaPaquete], tamaPaquete) ;
               s.receive(datagramaUDP) ;

               String msg = new String(datagramaUDP.getData()).trim() ;
               System.out.println("Recibido mensaje " + datagramaUDP.getAddress().getHostName() + " - " + msg) ;
         }
      }
      catch (Exception e)
      {
         System.out.println("Error en proceso");
      }
   }
}
```

Pero si vemos la consola del servidor ... se nos estan perdiendo datos

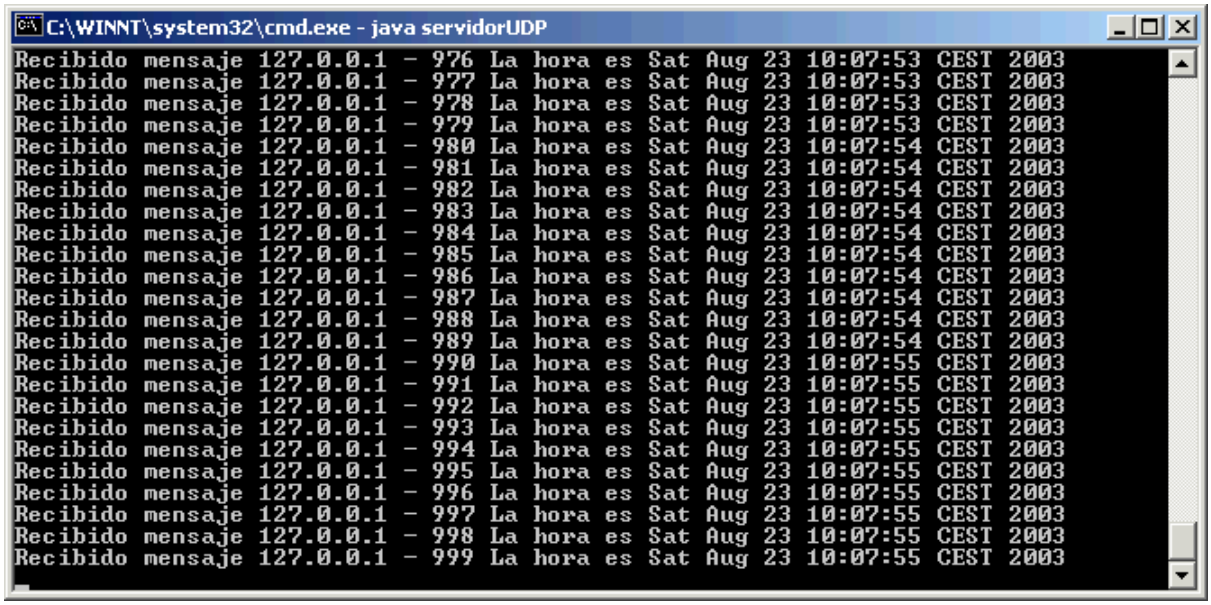


Posiblemente ... estemos mandandolos muy rápido

Si ahora modificamos un poco nuestro código y ponemos un retardo ... podemos ver que no se pierden datos

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

class clienteUPD
{
     static void retardo(int iMilisegundos)
     {
          try
          {
```

```
            Thread.currentThread().sleep(iMilisegundos);
         }
         catch(Exception e)
         {}
      }


      public static void main(String[] args)
      {
         try
         {
               String cadena = null;
               byte[] paqueteBytes = null;
               DatagramPacket paqueteUDP = null;

               InetAddress direccion = InetAddress.getByName("localhost") ;

               long inicio = System.currentTimeMillis();

               for(int i=0;i<1000;i++)
               {
                  DatagramSocket socketUDP = new DatagramSocket() ;
                  cadena = i + " La hora es " + new Date().toString();
                  paqueteBytes = cadena.getBytes() ;
                  paqueteUDP = new DatagramPacket(paqueteBytes, paqueteBytes.length,direccion,8031);
                  socketUDP.send(paqueteUDP) ;
                  retardo(100);
                  socketUDP.close() ;

               }

               long fin = System.currentTimeMillis() - inicio;
               System.out.println("El tiempo transcurrido es " + fin + " milisegundos");
            }
         catch (IOException e)
         {
               System.out.println(e) ;
         }
      }
}
```



Como conclusión, podemos decir que .... si se generan demasiados mensajes .... y los mandamos todos seguidos ..... se perderán paquetes ... así que tenemos que ser habiles y utilizar mecanismos un poco más elaborados para asegurarnos de no perder datos.

Envio de mensajes a través de JMS

JMS es un interfaz para interactuar con proveedores de servicio de mensajería (bueno... más o menos)

Nosotros podemos crear un programa cliente que se conecte a una cola de mensajes y enviar nuestra información.

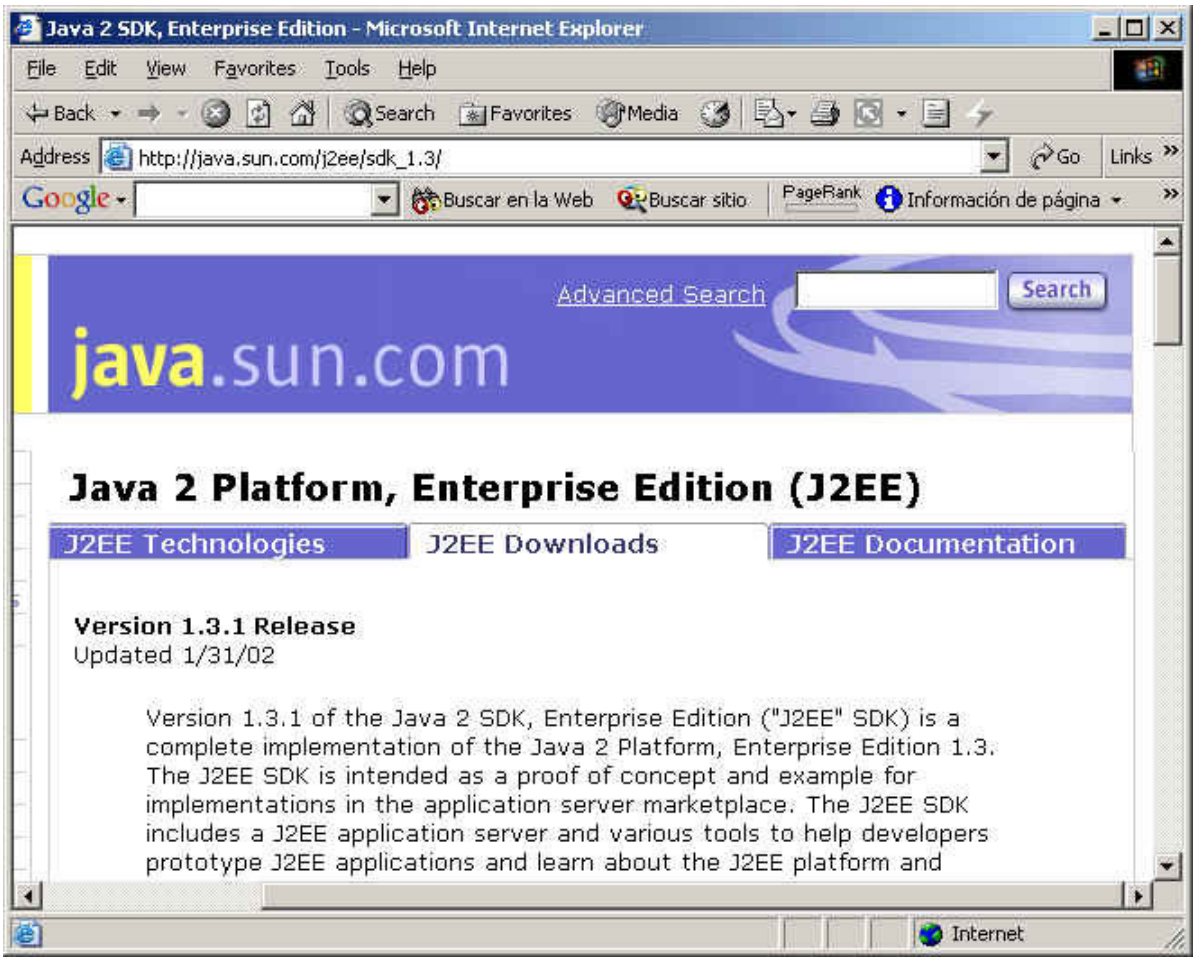
Una de las ventajas que puede presentarnos este sistema es que el gestor de colas podría ser cualquier cosa ..... de tan modo que se nos puede proporcionar de un modo transparente capacidad de persistencia, transaccionalidad, backup, politicas de enrutamiento ...etc

Nosotros vamos ha hacer un pequeño programa que se conecte a una cola que hemos creado en el servidor de aplicaciones J2EE de Su

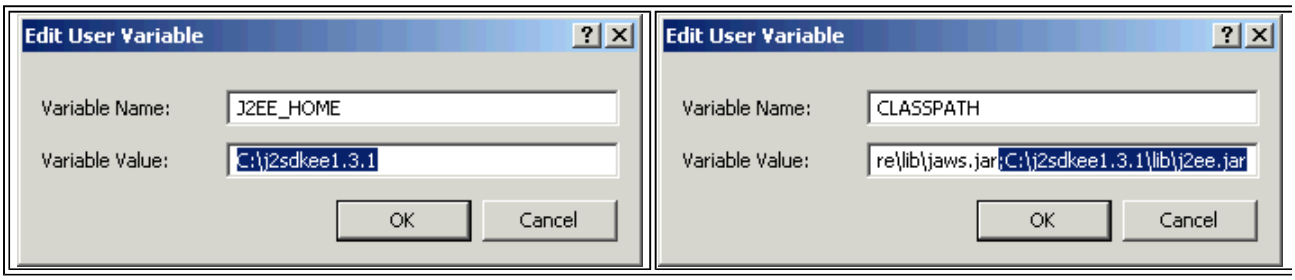
(versión 1.3.1). Tengo que decir que me he vuelto loco intentando hacer funcionar este ejemplo en la versión 1.4 (con pequeños camb siguiendo de tutorial de Sun) ..... por lo que al final he desistido y seguido las recomendaciones de muchos foros de (obviamente esto es un proyecto vital) usando la 1.3.

Os vamos a mostrar todos los pasos (que son muy sencillos)

- Nos descargamos el software J2EE de sun (y lo instalamos)

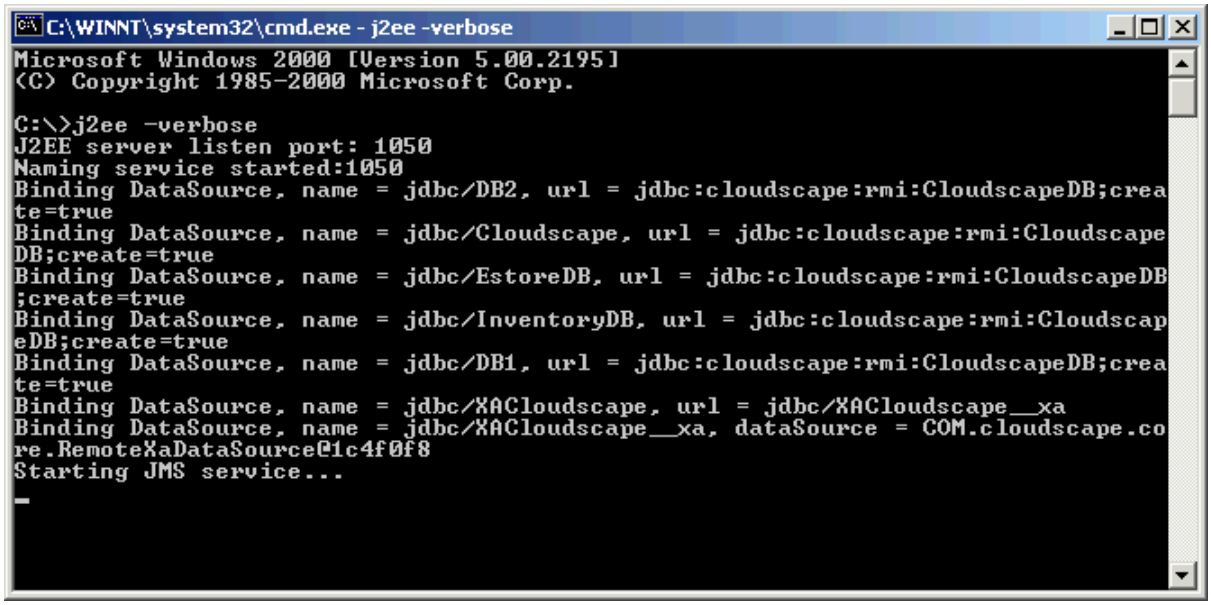


- Ponemos las variables de entorno



- Arrancamos el servidor escribiendo

j2ee -verbose



- Creamos la cola escribiendo

j2eeadmin -addJmsFactory ColaRoberto queue

Escribimos el código que manda los mensajes

```
/*
 * envioMensajeSimple.java
 *
 * Created on 12 de agosto de 2003, 22:33

 * @author  Roberto Canales
 */

import javax.naming.*;
import javax.jms.*;
import java.util.*;


public class envioMensajeSimple {
   /** Punto de entrada a la aplicacion*/
   public static void main(String[] args) {
      envioMensajeSimple programa = new envioMensajeSimple();
      programa.arranca();
   }

   /** Funcion para centralizar mensajes*/
   void depura(String cadena) {
      System.out.println("Mensaje: " + cadena);
   }

   private static InitialContext getInitialContext() throws NamingException
   {
      Hashtable env = new Hashtable();
      env.put(Context.INITIAL_CONTEXT_FACTORY,"com.sun.enterprise.naming.SerialInitContextFactory");
      env.put("java.naming.factory.url.pkgs", "com.sun.enterprise.naming");
      env.put(Context.PROVIDER_URL, "iiop://127.0.0.1:1050");

      return new InitialContext(env);
   }

   /** Metodo encargado de ejecutar la tarea*/
   void arranca() {
      // inicializamos las variables
      Context jndiContext = null;
      Queue destino = null;
      QueueConnection  conexion = null;
      QueueSession  sesionActual = null;
      QueueSender  enviadorMensajes = null;
      TextMessage mensaje = null;


      // identificamos en nombre de la cola
```

```
        String colaDestino = "ColaRoberto";

        try {
            depura("Creamos un contexto");
             //jndiContext = new InitialContext();
            jndiContext = getInitialContext();
        }
        catch (NamingException e) {
            System.out.println("Imposible crear el contexto  " + "contexto: " + e.toString());
            System.exit(1);
        }

        try {
            // creamos el objeto para construir la conexion
            depura("Buscamos la factoria");
            QueueConnectionFactory queueConnectionFactory = (QueueConnectionFactory)jndiContext.lookup("QueueConnectionFactory'

            // localizamos la cola destino
            depura("Buscamos la cola");
            destino = (Queue) jndiContext.lookup(colaDestino);

            // creamos una conexión
            depura("Creamos la conexion");
            conexion = queueConnectionFactory.createQueueConnection();

            depura("Creamos la sesion");
            sesionActual = conexion.createQueueSession(false, Session.AUTO_ACKNOWLEDGE);

            // Creamos una factoria de mensaje y un mensaje
            depura("Creamos el productor de mensajes");

            enviadorMensajes = sesionActual.createSender(destino);

            mensaje = sesionActual.createTextMessage();


                        mensaje.setStringProperty("TipoMensaje", "1");


           long inicio = System.currentTimeMillis();

           for(int i=0;i<1000;i++)
           {
              // Asociamos texto al mensaje
              mensaje.setText("Nuevo mensaje " + new Date() );

              //depura("Enviamos el mensaje");
              enviadorMensajes.send(mensaje);
           }
            enviadorMensajes.close();

            long fin = System.currentTimeMillis() - inicio;
            System.out.println("El tiempo transcurrido es " + fin + " milisegundos");


        }
        catch (Exception e) {
            depura("Error en la aplicación " + e.toString());
            e.printStackTrace();
        }
        finally {
            if (conexion != null) {
                try {
                    conexion.close();
                }
                catch (JMSException e)
                {}
            }
        }

        depura("Salimos del programa");
    }
}
```

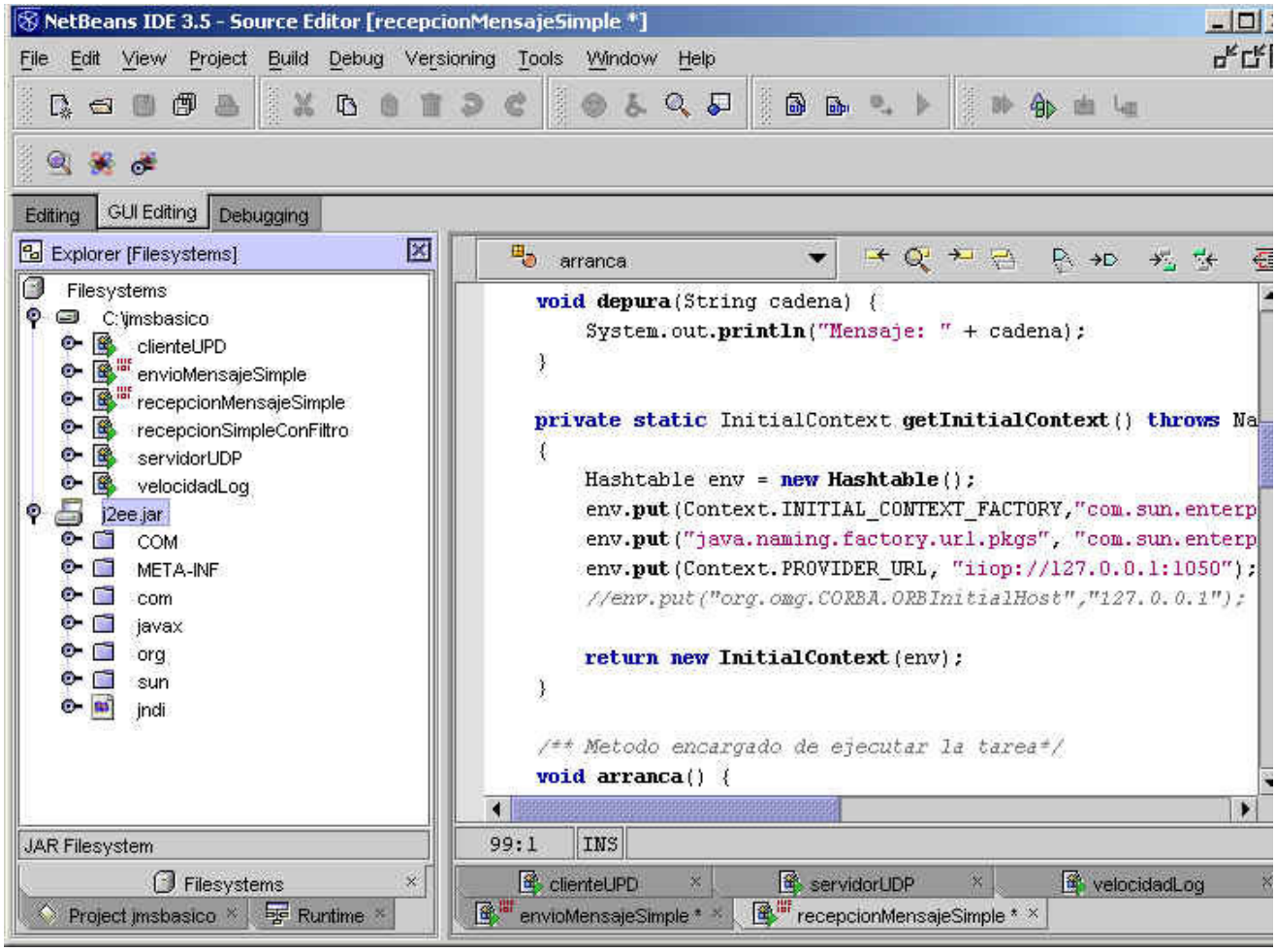
Ahora el código del cliente (que se diferencia muy poquito del anterior)

```
/*
 * recuperacionPrecio.java
 *
 * Created on 20 de agosto de 2003, 20:22
 */
```

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

/**
 *
 * @author  Roberto Canales
 */
public class recepcionMensajeSimple {

    /** Creates a new instance of recuperacionPrecio */
    public recepcionMensajeSimple() {
    }

    /**
     * @param args the command line arguments
     */
    /** Punto de entrada a la aplicacion*/
    public static void main(String[] args) {
        recepcionMensajeSimple programa = new recepcionMensajeSimple();
        programa.arranca();
    }

    /** Funcion para centralizar mensajes*/
    void depura(String cadena) {
        System.out.println("Mensaje: " + cadena);
    }

    private static InitialContext getInitialContext() throws NamingException
    {
        Hashtable env = new Hashtable();
        env.put(Context.INITIAL_CONTEXT_FACTORY,"com.sun.enterprise.naming.SerialInitContextFactory");
        env.put("java.naming.factory.url.pkgs", "com.sun.enterprise.naming");
        env.put(Context.PROVIDER_URL, "iiop://127.0.0.1:1050");

        return new InitialContext(env);
    }

    /** Metodo encargado de ejecutar la tarea*/
    void arranca() {
        // inicializamos las variables
        Context jndiContext = null;
        Queue origen = null;
        QueueConnection  conexion = null;
        QueueSession  sesionActual = null;
        QueueReceiver  receptorMensajes = null;
        TextMessage mensaje = null;


        // identificamos en nombre de la cola
        String colaorigen = "ColaRoberto";

        try {
            depura("Creamos un contexto");
             //jndiContext = new InitialContext();
            jndiContext = getInitialContext();
        }
        catch (NamingException e) {
            System.out.println("Imposible crear el contexto  " + "contexto: " + e.toString());
            System.exit(1);
        }

        try {
            // creamos el objeto para construir la conexion
            depura("Buscamos la factoria");
            QueueConnectionFactory queueConnectionFactory = (QueueConnectionFactory)jndiContext.lookup("QueueConnectionFactory'

            // localizamos la cola origen
            depura("Buscamos la cola");
            origen = (Queue) jndiContext.lookup(colaorigen);

            // creamos una conexión
            depura("Creamos la conexion");
            conexion = queueConnectionFactory.createQueueConnection();

            depura("Creamos la sesion");
            sesionActual = conexion.createQueueSession(false, Session.AUTO_ACKNOWLEDGE);

            // Creamos una factoria de mensaje y un mensaje
            depura("Creamos el Receptor de Mensajes");

            receptorMensajes = sesionActual.createReceiver(origen);

            depura("Arrancamos la conexion");
```

```
            conexion.start();

            depura("Leemos el mensajes");
            mensaje = (TextMessage)receptorMensajes.receive();

            depura("El mensaje recibido es " + mensaje.getText());

            receptorMensajes.close();
        }
        catch (Exception e) {
            depura("Error en la aplicación " + e.toString());
            e.printStackTrace();
        }
        finally {
            if (conexion != null) {
                try {
                    conexion.close();
                }
                catch (JMSException e)
                {}
            }
        }

        depura("Cerramos el lector");
    }
}
```

Si usais NetBeans (o cualquier otro entorno gráfico) **aseguraos que encuentra el fichero j2ee.jar**



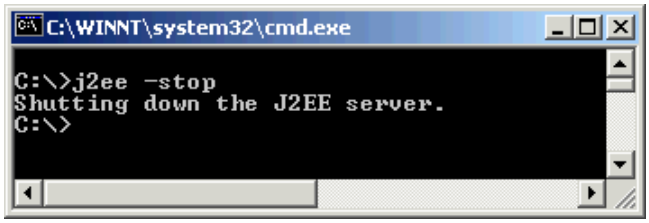
Si arrancamos ahora nuestro programa para enviar mensajes a la cola

```
Mensaje: Creamos un contexto
Mensaje: Buscamos la factoria
Mensaje: Buscamos la cola
Mensaje: Creamos la conexion
Java(TM) Message Service 1.0.2 Reference Implementation (build b14)
Mensaje: Creamos la sesion
Mensaje: Creamos el productor de mensajes
El tiempo transcurrido es 43142 milisegundos
```

```
Mensaje: Salimos del programa
```

El tiempo es un pelín alto.. aunque puede deberse a distintas cosas (como el lookup de nombre de máquina)

Si paramos el servidor de aplicaciones



Y lo volvemos a arrancar ..... y lanzamos el programa que lee mensajes de la cola ... **los mensajes siguen**

```
Mensaje: Creamos un contexto
Mensaje: Buscamos la factoria
Mensaje: Buscamos la cola
Mensaje: Creamos la conexion
Java(TM) Message Service 1.0.2 Reference Implementation (build b14)
Mensaje: Creamos la sesion
Mensaje: Creamos el Receptor de Mensajes
Mensaje: Arrancamos la conexion
Mensaje: Leemos el mensajes
Mensaje: El mensaje recibido es Nuevo mensaje Sat Aug 23 10:38:54 CEST 2003
Mensaje: Cerramos el lector
```

Hay un linea que marcamos en el programa enviador de mensajes

mensaje.setStringProperty("TipoMensaje", "1");

Una de las gracias que tiene el uso de JMS, es que te permite conectar un programa cliente y filtrar solo los mensajes que deseas .....

Si modificamos un poquito el programa receptor (ojo que ahora leemos 1000 mensajes) .. podemos establecer el filtro a través de propiedades, con una sintaxis parecida a la de SQL

```
/*
 * recepcionSimpleConFiltro .java
 *
 * Created on 20 de agosto de 2003, 20:22
 */
import javax.naming.*;
import javax.jms.*;
import java.util.*;

/**
 *
 * @author  Roberto Canales
 */
public class recepcionSimpleConFiltro {

    /** Creates a new instance of recuperacionPrecio */
    public recepcionSimpleConFiltro() {
    }

    /** Punto de entrada a la aplicacion*/
    public static void main(String[] args) {
        recepcionSimpleConFiltro programa = new recepcionSimpleConFiltro();
        programa.arranca();
    }

    /** Funcion para centralizar mensajes*/
    void depura(String cadena) {
        System.out.println("Mensaje: " + cadena);
    }

    private static InitialContext getInitialContext() throws NamingException
    {
        Hashtable env = new Hashtable();
        env.put(Context.INITIAL_CONTEXT_FACTORY,"com.sun.enterprise.naming.SerialInitContextFactory");
        env.put("java.naming.factory.url.pkgs", "com.sun.enterprise.naming");
        env.put(Context.PROVIDER_URL, "iiop://127.0.0.1:1050");
        //env.put("org.omg.CORBA.ORBInitialHost","127.0.0.1");
```

```
        return new InitialContext(env);
    }

    /** Metodo encargado de ejecutar la tarea*/
    void arranca() {
        // inicializamos las variables
        Context jndiContext = null;
        Queue origen = null;
        QueueConnection  conexion = null;
        QueueSession  sesionActual = null;
        QueueReceiver  receptorMensajes = null;
        TextMessage mensaje = null;


        // identificamos en nombre de la cola
        String colaorigen = "ColaRoberto";

        try {
            depura("Creamos un contexto");
             //jndiContext = new InitialContext();
            jndiContext = getInitialContext();
        }
        catch (NamingException e) {
            System.out.println("Imposible crear el contexto  " + "contexto: " + e.toString());
            System.exit(1);
        }

        try {
            // creamos el objeto para construir la conexion
            depura("Buscamos la factoria");
            QueueConnectionFactory queueConnectionFactory = (QueueConnectionFactory)jndiContext.lookup("QueueConnectionFactory'

            // localizamos la cola origen
            depura("Buscamos la cola");
            origen = (Queue) jndiContext.lookup(colaorigen);

            // creamos una conexión
            depura("Creamos la conexion");
            conexion = queueConnectionFactory.createQueueConnection();

            depura("Creamos la sesion");
            sesionActual = conexion.createQueueSession(false, Session.AUTO_ACKNOWLEDGE);

            // Creamos una factoria de mensaje y un mensaje
            depura("Creamos el Receptor de Mensajes");

         receptorMensajes = sesionActual.createReceiver(origen,"TipoMensaje = 1");

            depura("Arrancamos la conexion");
            conexion.start();

            depura("Leemos el mensajes");
            for(int i=0;i<1000;i++)
            mensaje = (TextMessage)receptorMensajes.receive();

            depura("El mensaje recibido es " + mensaje.getText());

            receptorMensajes.close();
        }
        catch (Exception e) {
            depura("Error en la aplicación " + e.toString());
            e.printStackTrace();
        }
        finally {
            if (conexion != null) {
                try {
                    conexion.close();
                }
                catch (JMSException e)
                {}
            }
        }

        depura("Cerramos el lector");
    }
}
```

Como conclusión .... la utilización de colas de mensajes..... penalizando en tiempo ... nos proporciona mayores garantias de entrga y mejores capacidades de gestión, desacomplando la generación y la captura..

Conclusión General

En función del tipo de aplicación .... debemos decidir cual es el sistema que más nos interesa

... aunque con este ejemplo si hemos ganado algo claro .... la posibilidad de tener ejemplos a mano, para hacer nuestras propias pruebas.

Sobre el Autor ..

Nuevo servicio de notificaciones

Si deseas que te enviemos un correo electrónico cuando introduzcamos nuevos tutoriales, inserta tu dirección de correo en el siguiente formulario.

Subscribirse a Novedades	
e-mail	
	Enviar

Otros Tutoriales Recomendados (También ver todos)

Nombre Corto	Descripción
Rendimiento de aplicaciones Web	En este tutorial veremos una introducción al funcionamiento de la Suite e-Test de Empirix.
Cachear porciones de JSPs	En este tutorial os ensañamos como incrementar increiblemente el rendimiento de vuestro Web basado en tecnología JSP con el FrameWork de cache OSCACHE
Test con JUnit	Cuando se hacen desarrollo profesionales, no basta con hacer los programas, hay que asegurarse de que van a funcionar. Una de las técnicas más seguras es crear aplicaciones que incluyan el código para autoprobarse. Os mostramos como usar JUnit
Construir un Servidor Web en Java	En este tutorial os enseñamos los principios de las aplicaciones multi-hilo a través de la creación de un servidor web básico en Java. Podremos ver en un ejemplo real el uso de sockets, threads, excepciones, etc.
Introducción a log4j	En un desarrollo Java es vital normalizar los logs para posteriormente poder depurar el funcionamiento de nuestra aplicación. Os mostramos como usar Log4J.
Message-Driven Beans al instante	Os mostramos como crear un EJB que consuma los mensajes JMS de una cola