Objetivos	Desarrollar una aplicación que utilice una arquitectura de 3 capas (presentación, aplicación y datos). La capa de presentación se desarrolla sobre J2ME. La capa de presentación y la de aplicación se deben comunicar mediante RMI.

Índice

1. Introducción

En esta práctica se debe realizar una aplicación formada por 3 capas:

- La capa de presentación es una aplicación que se debe ejecutar en la PDA
- La capa de aplicación se debe ejecutar en la máquina donde se desarrolla la práctica.
- La tercera es la base de datos que estará en la máquina `epi.uv.es`. La base de datos se llama `Libros` su estructura es la que aparece en los apuntes de teoría (tema JDBC).

La comunicación entre la PDA y la capa intermedia se realiza mediante el envío de mensajes a un objeto remoto mediante RMI.

La comunicación entre la capa intermedia y la base de datos se realiza mediante JDBC.

2. La Plataforma J2ME

Existen diferentes plataformas Java. La elección de una de ellas depende del aparato donde se va a ejecutar y del tipo de funcionalidad requerida. Las plataformas son:

- La plataforma Java 2 *Standard Edition* (J2SE) proporciona un entorno para el desarrollo de aplicaciones para ordenadores de sobremesa y servidores. Esta es la plataforma que hemos estado utilizando en todas las prácticas. Además constituye la base para la plataforma J2EE.
- La plataforma Java 2 *Enterprise Edition* (J2EE) define el estándar para el desarrollo de aplicaciones multicapa empresariales basadas en componentes. Incluye soporte para servicios Web.
- La plataforma Java 2 *Mobile Edition* (J2ME) proporciona un entorno orientado a aparatos con menor capacidad de procesamiento y memoria como teléfonos móviles, PDAs, sistemas telemáticos para vehículos,...

2.1. Configuraciones y perfiles de la plataforma J2ME

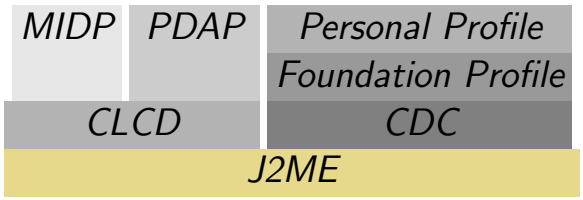
Debido a la variabilidad en cuanto a la capacidad de procesamiento, de memoria,... de los dispositivos (móviles, PDAs, ...) J2ME se divide en configuraciones, perfiles y paquetes opcionales. En función de las capacidades del dispositivo se debe elegir una configuración y un perfil a la hora de desarrollar aplicaciones J2ME.

Una configuración está diseñada para un tipo específico de dispositivo con unas restricciones de memoria y capacidad de procesamiento. Los perfiles son más específicos que las configuraciones. Un perfil está basado en una configuración y añade APIs para interfaces gráficas de usuario, almacenamiento persistente, etc. Las APIs opcionales definen funcionalidades específicas añadidas que pueden ser incluidas en una configuración particular.

La configuración básica se conoce como CLDC (*Connected Limited Device Configuration*) y es apropiada para móviles (cuyo perfil se conoce como MIDP *Mobile Information Device Profile*) y PDAs con pocas capacidades (cuyo perfil se conoce como PDAP *Personal Digital Assistant Profile*).

La configuración siguiente se conoce como CDC (em Connected Device Configuration) y es apropiada para aparatos con más capacidad de procesamiento y de memoria. Esta configuración proporciona una máquina virtual y bibliotecas básicas de clases para soportar aplicaciones en aparatos como PDAs o *set-top boxes* (aparato electrónico conectado a un canal de comunicación que produce su salida en una pantalla de televisión) para televisión digital interactiva. Se divide en diversos perfiles cada uno de los cuales añade funcionalidad sobre el anterior estos son el *Foundation Profile* y el *Personal Profile*.

La siguiente tabla muestra esta organización de configuraciones y perfiles de J2ME.



2.2. El Personal Profile

Para esta práctica vamos a utilizar el Personal Profile (este perfil es el más parecido al J2SE) al que hemos añadido el paquete RMIOP (*Remote Invocation Method Optional Package*) para poder desarrollar una aplicación que utilice RMI.

Los paquetes que incluye el Personal Profile son los siguientes:

java.applet
java.awt
java.awt.color
java.awt.datatransfer
java.awt.event
java.awt.image
java.beans
java.rmi
java.rmi.registry
javax.microedition.xlet
javax.microedition.xlet.ixc

Estos paquetes se añaden a los que ofrece el Foundation Profile que son:

java.io
java.lang
java.lang.ref
java.lang.reflect
java.math
java.net
java.security
java.security.acl
java.security.cert
java.security.interfaces
java.security.spec
java.text
java.util
java.util.jar
java.util.zip
javax.microedition.io

El Personal Profile admite tres modelos de aplicaciones:

1. Una aplicación que contiene el método

   ```
   public static void main(String[] args){...}
   ```

   La aplicación es básicamente como una aplicación típica desarrollada para la plataforma J2SE, la diferencia consiste en que utiliza las clases definidas para el Personal Profile.

2. Un applet que se ejecuta en el Applet Viewer o en un navegador.
3. Un Xlet que se ejecutan en un Xlet Manager.

En esta práctica se desarrollará la aplicación para la PDA utilizando la primera opción.

3. Descripción de las tareas a realizar

Es **crítico** realizar las tareas en el orden descrito en este enunciado.

Los comentarios en color rojo indican las tareas a realizar.

3.1. Realización de las clases para el servidor

3.1.1. Interface que describe el servicio

La interface que describe el servicio es la siguiente:

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

/** Interface que describe el servicio RMI
 * Define los métodos que se pueden llamar desde el cliente
 */
public interface ConsultaBD extends Remote {

   /** Metodo se debe implementar el objeto remoto
    * @param int codigo de autor
    * @return una LinkedList de String con los titulos de los libros del autor buscado
    * @throws RemoteException
    */
   public LinkedList listadoLibros(int codigoAutor) throws RemoteException;
}
```

3.1.2. Clase que implementa la interface

Esta clase debe tener un único método (el definido en la interface). En este método se debe realizar una consulta en la base de datos y almacenar el resultado en un objeto del tipo LinkedList.

La clase que implementa a la interface **debe** tener la siguiente estructura:

```
import java.rmi.server.*;
import java.rmi.*;
import java.sql.*;
import java.rmi.*;
import java.sql.*;
import java.util.*;
```

```
/** Clase que implementa los métodos definidos en la interface
 */
public class ConsultaBDImpl extends UnicastRemoteObject implements ConsultaBD {
   PreparedStatement consultaPlantilla;

   /** Constructor de la clase ConsultaDBImpl
    *@param url url a la base de datos (incluye host, puerto, BD, usuario y clave)
    */
   public ConsultaBDImpl(String url) throws RemoteException{
      try{
         // Conexion con la base de datos (se puede utilizar la clase Conexion)

         // Definicion de la plantilla con la consulta SQL

         // Obtencion del PreparedStatement con la consulta

      }catch(SQLException ex){ex.printStackTrace();}
   }

   public LinkedList listadoLibros(int codigoAutor) throws RemoteException{
      System.out.println("Llamada al metodo remoto");
      LinkedList titulos = new LinkedList();

      try{
         // Ejecucion de la consulta en la base de datos

         // Recorrer el ResultSet y almacenar cada uno de los titulos en una LinkedList
      }catch(SQLException ex){
         ex.printStackTrace();
      }
      return titulos;

   }

}
```

3.1.3. Clase que actúa de servidor

Las tareas que debe realizar esta clase son:

- Crear una instancia de la clase que ofrece el servicio.
- Registrar la referencia asignándole un nombre que identifique al servicio.

En este caso (para simplificar el problema) no vamos a instalar un administrador de seguridad.

La estructura que **debe** tener esta clase es la siguiente:

```
import java.sql.*;
import java.net.MalformedURLException;
import java.rmi.Naming;
import java.rmi.RMISecurityManager;
import java.rmi.RemoteException;

/**
 * Servidor RMI
 */
public class ConsultaBDServidor {
```

```
   public static void main(String[] args) {
      String urlBD = "....";
      try{

         // Creación del objeto que se ofrece como servicio pasando
         // como argumento la URL a la BD

         // Registro del servicio asignandole un nombre

      }catch(RemoteException e){
         e.printStackTrace();
      }catch(MalformedURLException e){
         e.printStackTrace();
      }
   }
}
```

3.2. Realización de las clases para el cliente

El cliente debe mostrar una interfaz gráfica de usuario que contenga un campo de texto, un botón y un área de texto. En el campo de texto el usuario debe introducir el código de autor y cuando se pulse sobre el botón debe enviar un mensaje al objeto remoto. El resultado se debe mostrar en el área de texto.

Nótese que esta clase se va a ejecutar en la PDA por lo tanto hay que utilizar las clases definidas en el Personal Profile.

La estructura que **debe** tener esta clase es

```
import java.net.MalformedURLException;
import java.rmi.Naming;
import java.rmi.NotBoundException;
import java.rmi.RMISecurityManager;
import java.rmi.RemoteException;
import java.awt.*;
import java.awt.event.*;
import java.util.*;

/**
 * Cliente RMI
 */
public class Consulta extends Frame{
   private TextField campoTexto;
   private Button boton;
   private TextArea areaTexto;

   // La referencia al objeto remoto
   private ConsultaBD con=null;

   public Consulta(String urlRMI){

      // Obtención de la referencia al objeto remoto
      try {

         con = (ConsultaBD) Naming.lookup(urlRMI);
      } catch (MalformedURLException e) {
         System.out.println("URL Incorrecta");
      } catch (RemoteException e) {
         System.out.println("Excepcion remota");
```

```
        } catch (NotBoundException e) {
            System.out.println("No asociado al servicio");
        }

        // Construcción de la GUI

        // Tratamiento de eventos en el boton

        boton.addMouseListener(new MouseAdapter(){
            public void mousePressed(MouseEvent e){
                // Enviar mensaje al objeto remoto y mostrar el
                // resultado en el área de texto
            }
        });

        // Tratamiento de eventos en la ventana
        addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent e){
                System.exit(0);
            }
        });

        setSize(200,200);
        show();

    }
    public static void main(String[] args){
        // Substituir host y servicio por lo que corresponda
        new Consulta("rmi://host/ servicio");
    }
}
```

3.3. Compilación, generación del stub y organización de las clases

☐ Compilar las clases correspondientes al servidor.

☐ Generar el *stub* utilizando `rmic`:

```
rmic -v1.2 ConsultaBDImpl
```

☐ Las clases compiladas correspondientes al servidor se colocan en un directorio con nombre `Servidor` estas clases son:

```
practica12
    |
    Servidor
        |   Conexion.class
        |   ConsultaBD.class
        |   ConsultaBDImpl.class
        |   ConsultaBDImpl_Stub.class
```

☐ Compilar las clases correspondientes al cliente. En este caso se trata de una única clase `Consulta.java`. Esta clase hay que compilarla de forma que se compruebe que únicamente se están utilizando características propias del *Personal Profile*. Para ello compilar esta clase utilizando la siguiente instrucción:

```
javac -bootclasspath classes.zip;ppro-ui.jar;rmip.jar Consulta.java
```

La opción -bootclasspath sirve para indicar que la compilación no se realice utilizando las bibliotecas de clases estándar sino que se realice utilizando las bibliotecas que se pasan como argumento (en este caso: `classes.zip` son las clases que definen el Personal Profile, `ppro-ui.jar` son las clases que permiten utilizar AWT y `rmip.jar` constituyen el RMI Optional Package).

☐ Las clases compiladas correspondientes al cliente se colocan en un directorio con nombre `Cliente`.

☐ Empaquetar las clases del cliente en un fichero llamado `Consulta.jar`:

```
practica12
    |
    |-- Cliente
            |
            |-- Consulta.class
            |-- Consulta$1.class
            |-- Consulta$2.class
            |-- ConsultaBDImpl_Stub.class
            |
            |-- Consulta.jar
```

```
jar -cvf Consulta.jar *.class
```

☐ Crear un directorio llamado `PDA` y copiar en él el fichero `Consulta.jar`

```
practica12
    |
    |-- PDA
            |
            |-- Consulta.jar
```

3.4. Ejecución del servidor

En el directorio `Servidor` se lanza:

```
rmiregistry
```

En el directorio `Servidor` se lanza:

```
java -classpath ruta/mysql-connector-java.jar;. ConsultaBDServidor
```

3.5. Ejecución del cliente en la misma máquina que el servidor

Antes de copiar el cliente a la PDA hay que probarlo en la misma máquina en la que está ejecutándose el servidor.

En el directorio `Cliente` se lanza:

```
java -classpath Consulta.jar Consulta
```

3.6. Ejecución del cliente en la PDA

Una vez que se ha comprobado que el cliente funciona correctamente, se debe copiar el fichero `Consulta.jar` y el fichero `Consulta.lnk` a la PDA, el contenido de este último fichero se detallará en la sesión de laboratorio.

La aplicación se ejecuta pulsando sobre el fichero `Consulta.lnk` y si se han seguido los pasos indicados nos aparecerá la interfaz gráfica.