UNIVERSIDAD DE MÁLAGA
Departamento de Lenguajes y Ciencias de la Computación
UNIVERSIDAD DE MÁLAGA

Gestión de la Información
Práctica 3

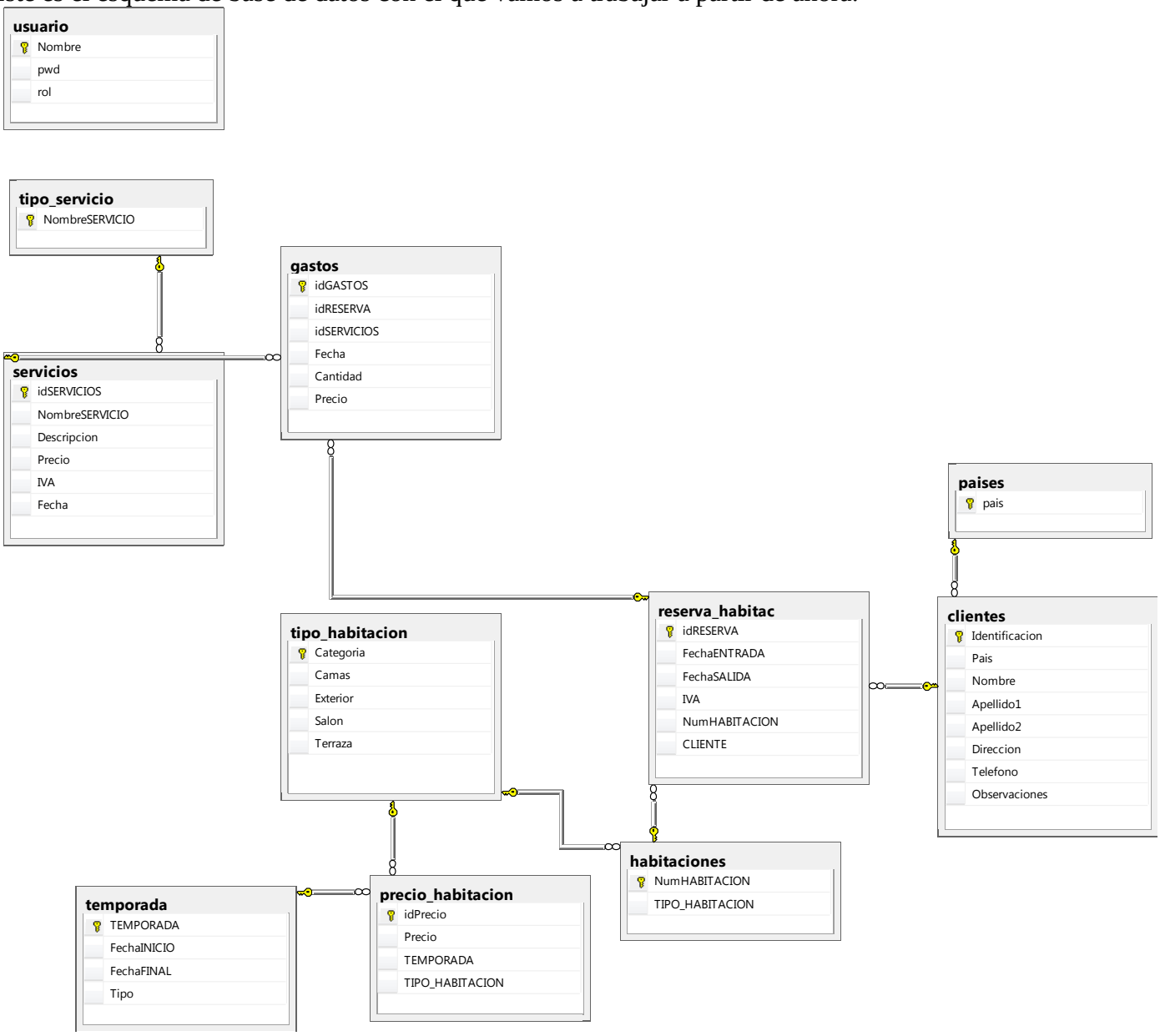
JDBC- SQL - Server

Vamos a crear y a trabajar con la base de datos de reservas de habitaciones de un hotel.

El objetivo de esta práctica empezar a familiarizarse con el manejo de conexiones a bases de datos usando Java + JDBC.

Lo primero que vamos a hacer es desde una consola de comandos `sqlcmd` añadir a la base de datos "Hotel" la tabla de gastos de una reserva. Dicha tabla tendrá los campos `idGASTOS int identity NOT NULL , idRESERVA numeric NOT NULL, idSERVICIOS int NOT NULL, Fecha datetime NOT NULL, Cantidad int NOT NULL, Precio money not null`. La clave primaria será `idGASTOS` y `idreserva` será clave externa que referencia al campo `idreserva` de la tabla `reserva_habitac` y `idservicios` será clave externa que referencia al campo `idSERVICIOS` de la tabla `SERVICIOS`.

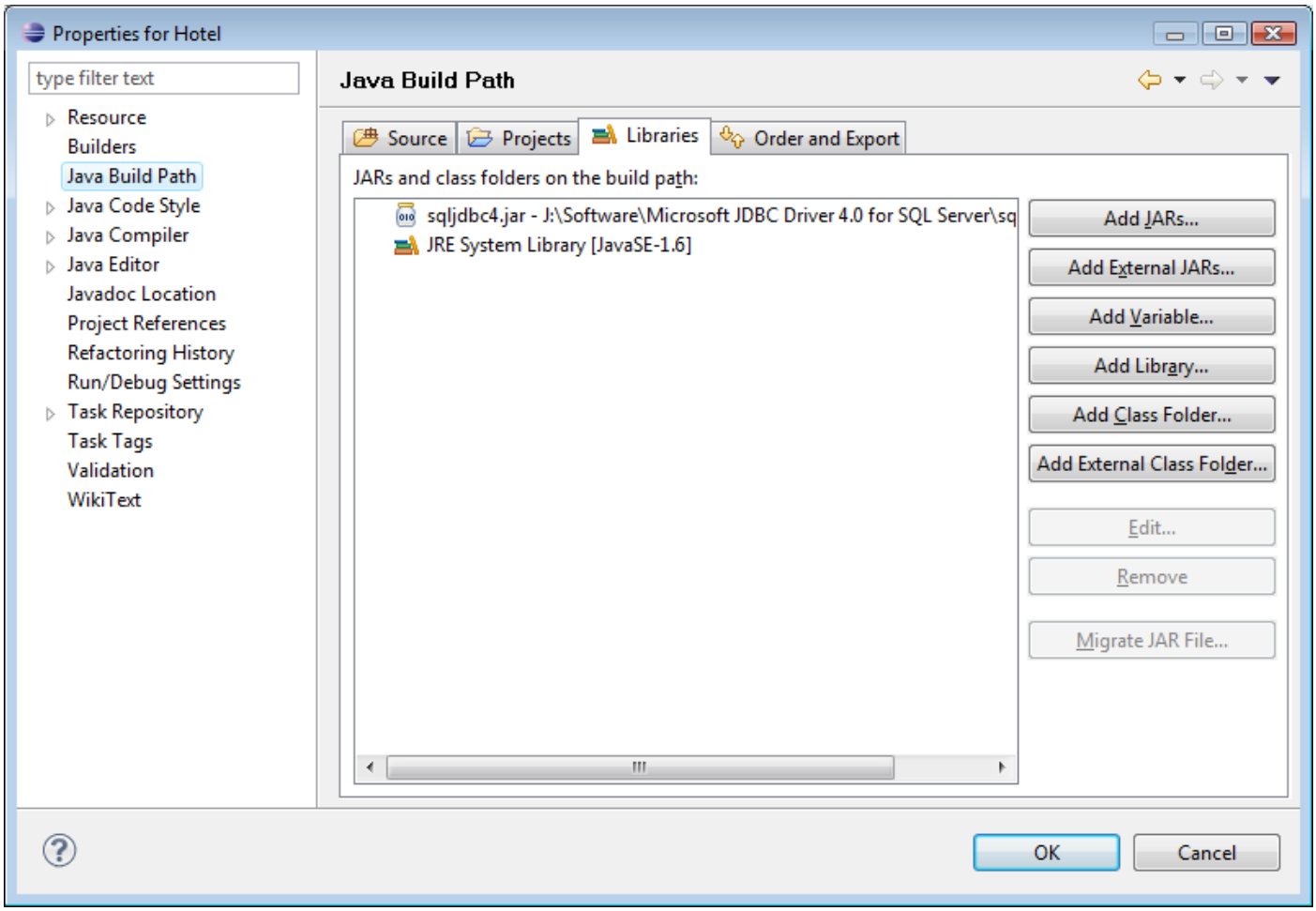
Este es el esquema de base de datos con el que vamos a trabajar a partir de ahora:



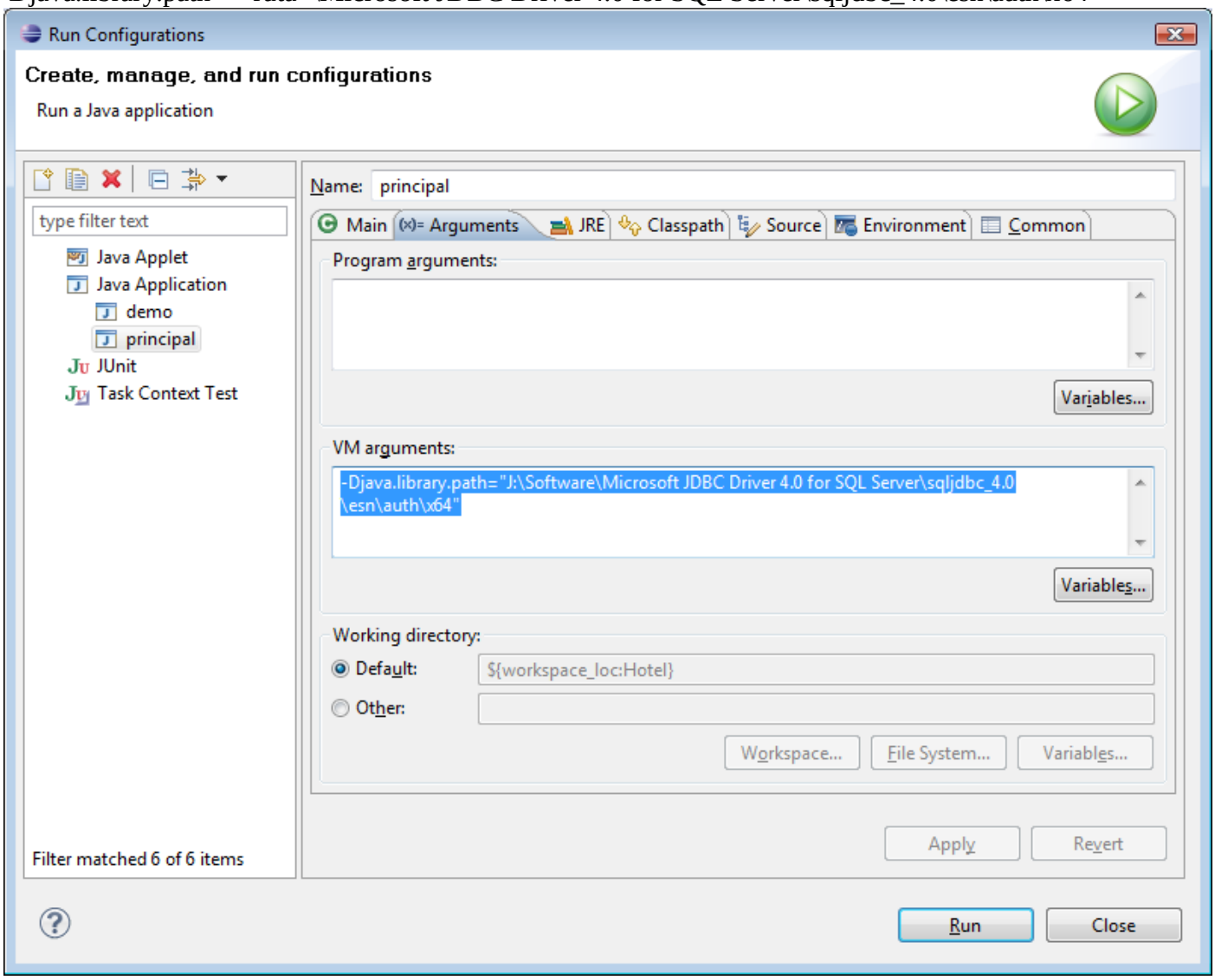
Posteriormente vamos a habilitar la escucha de peticiones a través de TCP/IP de nuestro servidor de SQL Server. Para ello usaremos el "*Administrador de configuración de SQL Server*". En la pestaña de Configuración de Red de SQL Server -> Protocolos de SQLEXPRESS habilitaremos TCP/IP.
A continuación pincharemos con el botón derecho sobre TCP/IP y accederemos al menú de propiedades.
En la pestaña de "Direcciones IP" establecernos como puerto de escucha el 1433 para todos los protocolos: IP1, IP2, …

Una vez realizada esta tarea deberemos reiniciar la máquina y tendremos el servidor SQL Server preparado para recibir peticiones por el puerto 1433 vía TCP/IP.

A continuación descargaremos (y descomprimiremos) del campus virtual el driver de JDBC para SQL Server en la carpeta que deseemos. Entraremos en eclipse y crearemos el proyecto java llamado Hotel.
Añadiremos en las propiedades del proyecto Java Build Path una librería externa que será la `sqljdbc4.jar`



Posteriormente el la configuración del Run para dicho proyecto añadiremos como variable de entorno para la máquina virtual de java el path de la librería de autentificación de Windows (estas máquinas son de 64 bits).
-Djava.library.path="<ruta>\Microsoft JDBC Driver 4.0 for SQL Server\sqljdbc_4.0\esn\auth\x64"



¡¡ Y ya podemos empezar a codificar en java ☺ !!

Cualquier error que se produzca deberá resultar en el lanzamiento de una excepción no comprobada `HotelException`, incluyendo un mensaje descriptivo del error.

Crear la Clase `Usuario`:

- Define las constantes:
  - INVITADO = 0;
  - USUARIO = 1;
  - ADMINISTRADOR = 2;
- Tiene los siguientes atributos:
  - `private String Nombre;`
  - `private String pwd;`
  - `private int rol;`
- Y Ofrece los siguientes métodos públicos:
  - `public Usuario(String name):` Constructor que rellena los atributos leyéndolos de la tabla correspondiente de la base de datos.
  - `public void setNombre(String nombre):` Modifica el atributo y actualiza la base de datos.

- `public String getNombre()`: Retorna el valor del atributo.
- `public void setPwd(String pwd)`: Modifica el atributo y actualiza la base de datos.
- `public String getPwd():` Retorna el valor del atributo.
- `public void setRol(int rol)` : Eleva una excepción indicando
- `public int getRol()` : Retorna el valor del atributo.
- `public String toString()`:Retorna la cadena "<nombre>:<pwd>:<rol>".
- `public void add(String n, String p, int r)`: Si el usuario que la invoca es administrador añadirá el nuevo usuario en la tabla. En caso contrario lanzará una excepción.
- `public void del(Usuario u)`: Si el usuario que la invoca es administrador borrará el usuario de la tabla. En caso contrario lanzará una excepción.
- `public void modiRol(Usuario u, int newRol)`: Si el usuario que la invoca es administrador modificará el rol del usuario en la tabla. En caso contrario lanzará una excepción.

Crear la Clase `Pais`:

- Ofrece los siguientes métodos públicos:
  - `public static List<String> ListadoPaises()`: Nos retorna la lista de países almacenados en la base de datos.

Crear la Clase `tipo_servicio`:

- Ofrece los siguientes métodos públicos:
  - `public static List<String> ListadoServicios():` Nos retorna la lista de servicios disponibles almacenados en la base de datos.

Crear la Clase `clientes`:

- Tiene los siguientes atributos:
  - `private String Identificacion;`
  - `private String Pais;`
  - `private String Nombre;`
  - `private String Apellido1;`
  - `private String Apellido2;`
  - `private String Direccion;`
  - `private String Telefono;`
  - `private String Observaciones;`
- Y Ofrece los siguientes métodos públicos:
  - `public Cliente(String id)`: Constructor que rellena los atributos leyéndolos de la tabla correspondiente de la base de datos.
  - `public static List<String> ListaIDClientes()`: Función que nos devuelve la lista con todos los identificadores de clients que hay en la base de datos.

Para Probar las clases desarrolladas podemos usar en siguiente programa principal:

```
package Hotel;

public class principal
{
	public static void main(String[] args)
	{
		Usuario u1,u2,adm;

		adm = new Usuario("admin");

		try
		{
			u1 = new Usuario("user2");
			System.out.println(u1.toString());
		}
		catch(Exception x)
		{
			System.out.println(x.getMessage());
			x.printStackTrace();
		}
		try
		{
			u1 = new Usuario("user2");
			u1.setNombre("pepe");
			System.out.println(u1.toString());
		}
		catch(Exception x)
		{
			System.out.println(x.getMessage());
			x.printStackTrace();
		}
		try
		{
			u1 = new Usuario("user2");
			u1.setNombre("user2");
			System.out.println(u1.toString());
		}
		catch(Exception x)
		{
			System.out.println(x.getMessage());
			x.printStackTrace();
		}
		try
		{
			u1 = new Usuario("user2");
			u1.setPwd("pwd");
			System.out.println(u1.toString());
		}
		catch(Exception x)
		{
			System.out.println(x.getMessage());
			x.printStackTrace();
		}
		try
		{
			u1 = new Usuario("user2");
			u1.setPwd("user2");
			System.out.println(u1.toString());
		}
		catch(Exception x)
		{
			System.out.println(x.getMessage());
			x.printStackTrace();
		}
```

```
try
{
	u1 = new Usuario("user2");
	u1.setRol(0);
	System.out.println(u1.toString());
}
catch(Exception x)
{
	System.out.println(x.getMessage());
	x.printStackTrace();
}
try
{
	u1 = new Usuario("user2");
	u1.setRol(1);
	System.out.println(u1.toString());
}
catch(Exception x)
{
	System.out.println(x.getMessage());
	x.printStackTrace();
}
try
{
	adm.add("pepe", "pepe", 1);
	u2 = new Usuario("pepe");
	System.out.println(u2.toString());
}
catch(Exception x)
{
	System.out.println(x.getMessage());
	x.printStackTrace();
}
try
{
	u2 = new Usuario("pepe");
	u2.add("u2", "u2", 2);
	u2 = new Usuario("u2");
	System.out.println(u2.toString());
}
catch(Exception x)
{
	System.out.println(x.getMessage());
	x.printStackTrace();
}
try
{
	u2 = new Usuario("pepe");
	adm.modiRol(u2,0);
	System.out.println(u2.toString());
}
catch(Exception x)
{
	System.out.println(x.getMessage());
	x.printStackTrace();
}
try
{
	u2 = new Usuario("pepe");
	adm.del(u2);
	u2 = new Usuario("pepe");
	System.out.println(u2.toString());
}
catch(Exception x)
{
	System.out.println(x.getMessage());
	x.printStackTrace();
```

```
		}

		try
		{
			System.out.println("Servicios");
			for(String t: tipo_servicio.ListadoServicios())
			{
				System.out.println(t);
			}
		}
		catch(Exception x)
		{
			System.out.println(x.getMessage());
			x.printStackTrace();
		}

		try
		{
			System.out.println("Paises");
			for(String p: Pais.ListadoPaises())
			{
				System.out.println(p);
			}
		}
		catch(Exception x)
		{
			System.out.println(x.getMessage());
			x.printStackTrace();
		}

		try
		{
			System.out.println("Clientes");
			for(String id: Cliente.ListaIDClientes())
			{
				Cliente c = new Cliente(id);
				System.out.println(c.toString());
			}
		}
		catch(Exception x)
		{
			System.out.println(x.getMessage());
			x.printStackTrace();
		}

	}

}
```