

## Diseño y Desarrollo de Sistemas de Información.

### Sesión Práctica 4. Creación con JDeveloper de Aplicaciones de Acceso a Bases de Datos Usando JDBC

#### Objetivo

Este tutorial muestra cómo crear aplicaciones usando JDeveloper que accedan a una base de datos empleando JDBC.

#### Contenido

En este tutorial se tratarán los siguientes puntos:

- Introducción.
- Preparar el entorno de trabajo.
- Establecer parámetros para una aplicación
- Conectarse a una base de datos.
- Ejecutar consultas.
- Insertar filas en una tabla.
- Actualizar datos en una base de datos.
- Resumen.

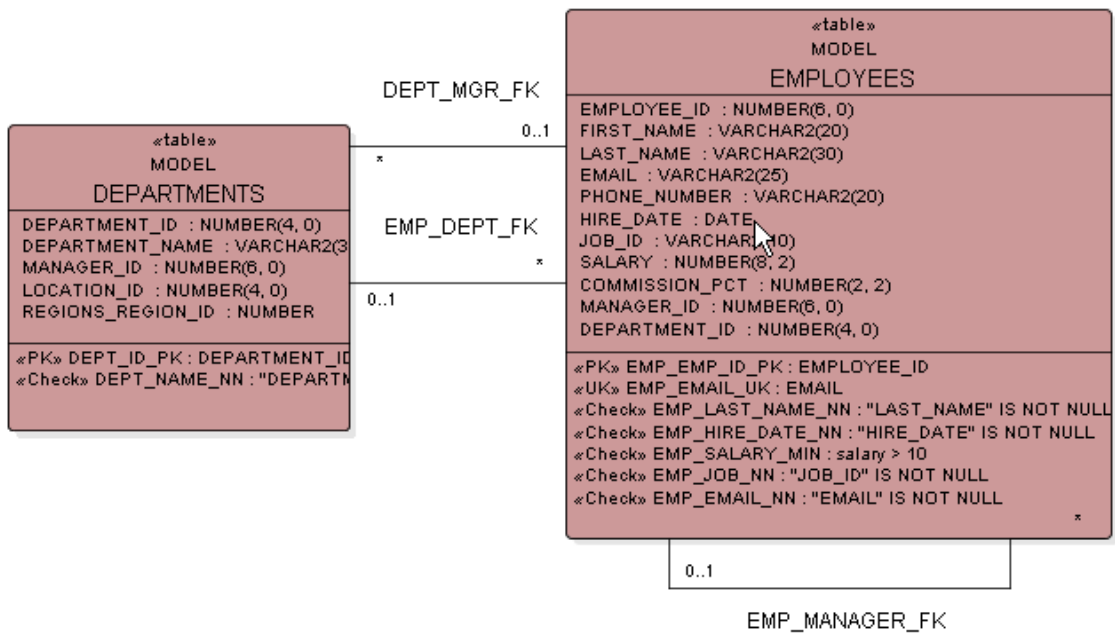
#### Introducción

Este tutorial muestra como acceder y manipular datos en una base de datos Oracle desde un programa escrito en Java empleando el entorno JDeveloper. Se creará un programa que ejecutará consultas sencillas, insertará filas en una tabla, y y actualizará datos de una tabla. Finalmente, el programa volverá a consultar la tabla para mostrar los cambios realizados.

El programa de ejemplo está diseñado para realizar sus operaciones sobre algunas tablas del esquema HR. El esquema HR es un esquema de ejemplo que suele venir instalado en las bases de datos Oracle.

El objetivo del programa será crear un nuevo departamento llamado “Facilities Management”, en la localización “1700”, y dirigido por “Laura Bissot” una empleada ya existente. Será necesario insertar el nuevo departamento en la Tabla *Departments* y posteriormente actualizar el registro de dicha empleada para indicar que dirige el nuevo departamento.

El diagrama de las tablas implicadas en este proceso es el siguiente:

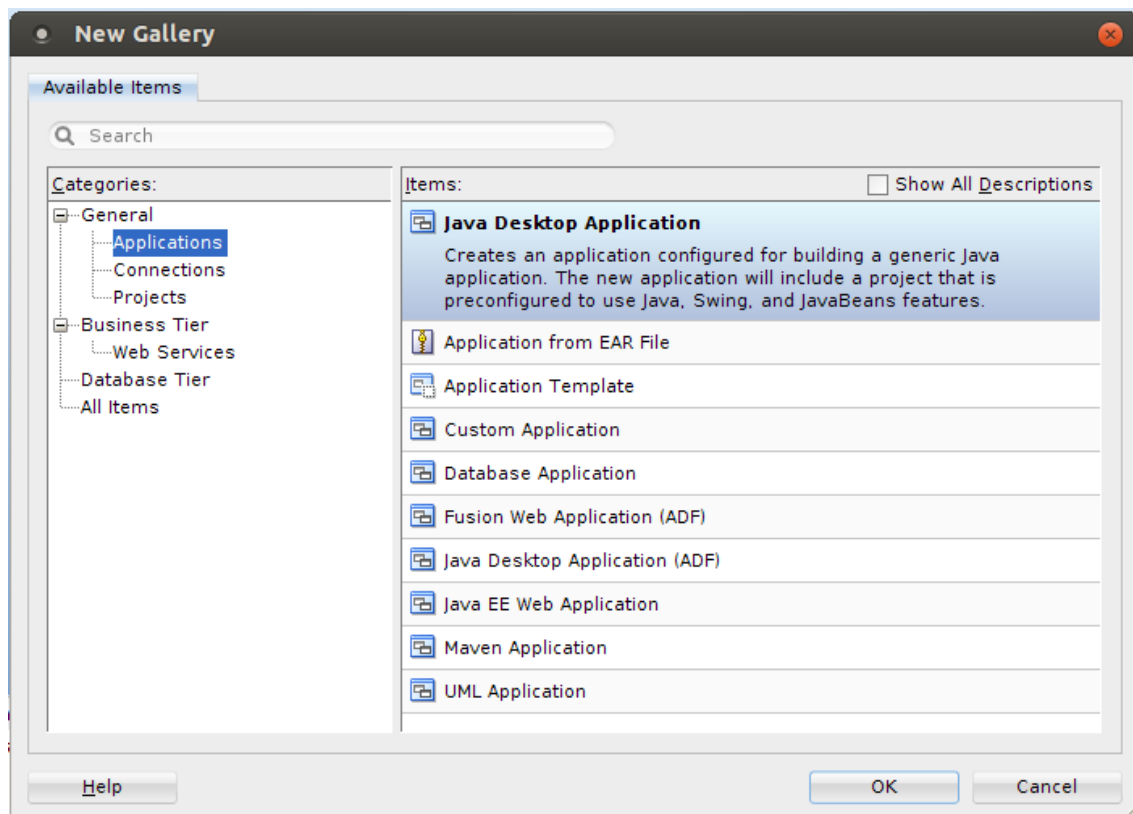


## Preparar el Entorno de Trabajo

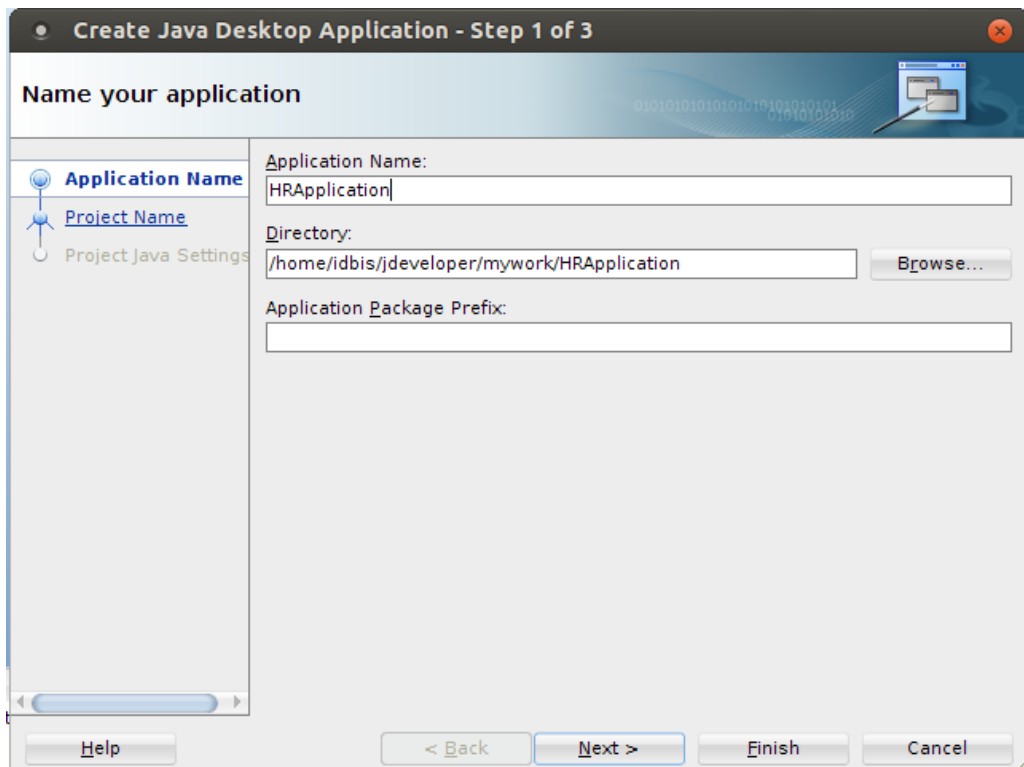
Antes de iniciar la implementación de la aplicación es necesario realizar algunas tareas para configurar adecuadamente el entorno de trabajo de tal forma que estén disponibles las tecnologías que deseamos emplear, en este caso JDBC. El primer paso será crear un espacio de trabajo y una aplicación dentro de dicho espacio que contendrá el trabajo que realizaremos. Posteriormente configuraremos la aplicación para que importe el paquete de clases que contienen el driver JDBC para acceder a una base de datos Oracle. Para ello, realice las siguientes tareas:

### Creamos un nuevo proyecto en el espacio de trabajo

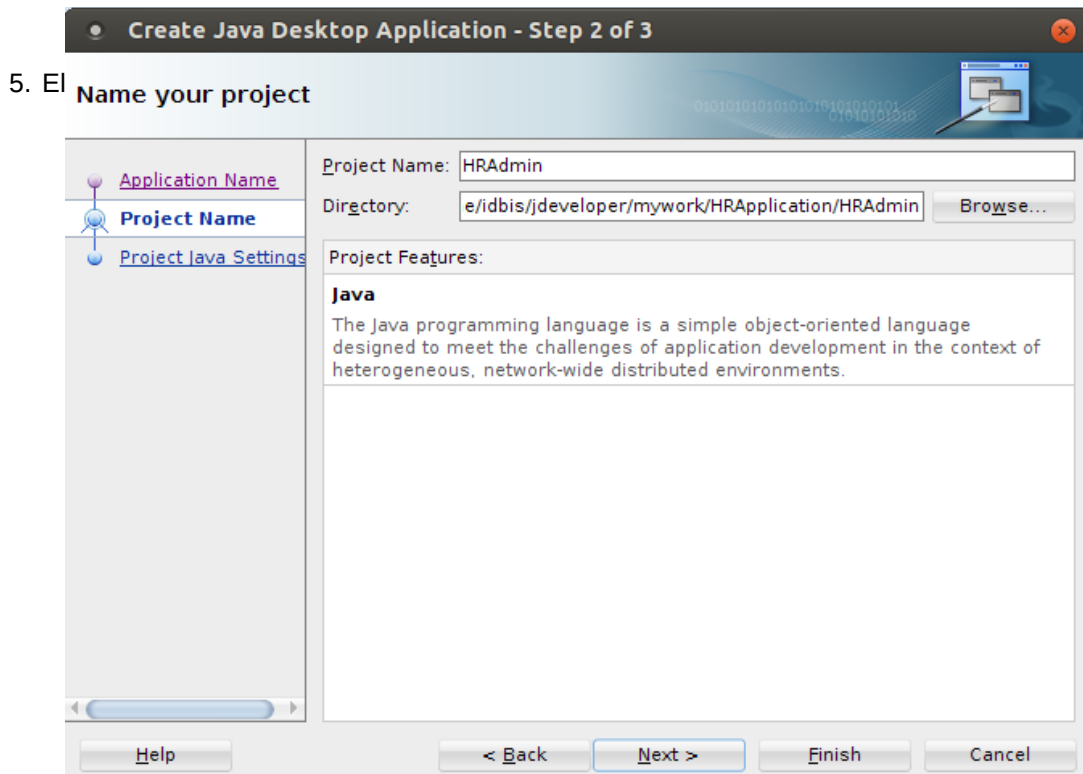
1. Seleccione las opciones **File** → **New** del menú para abrir la “**New Gallery**”.
2. En el panel de categorías “**Categories**” seleccione la opción “**Applications**” dentro de “**General**”, y en el panel de elementos “**Items**” seleccione la opción “**Java Desktop Application**”.



3. En el cuadro de diálogo de creación de aplicaciones, damos a nuestra aplicación el nombre de **HRApplication**.



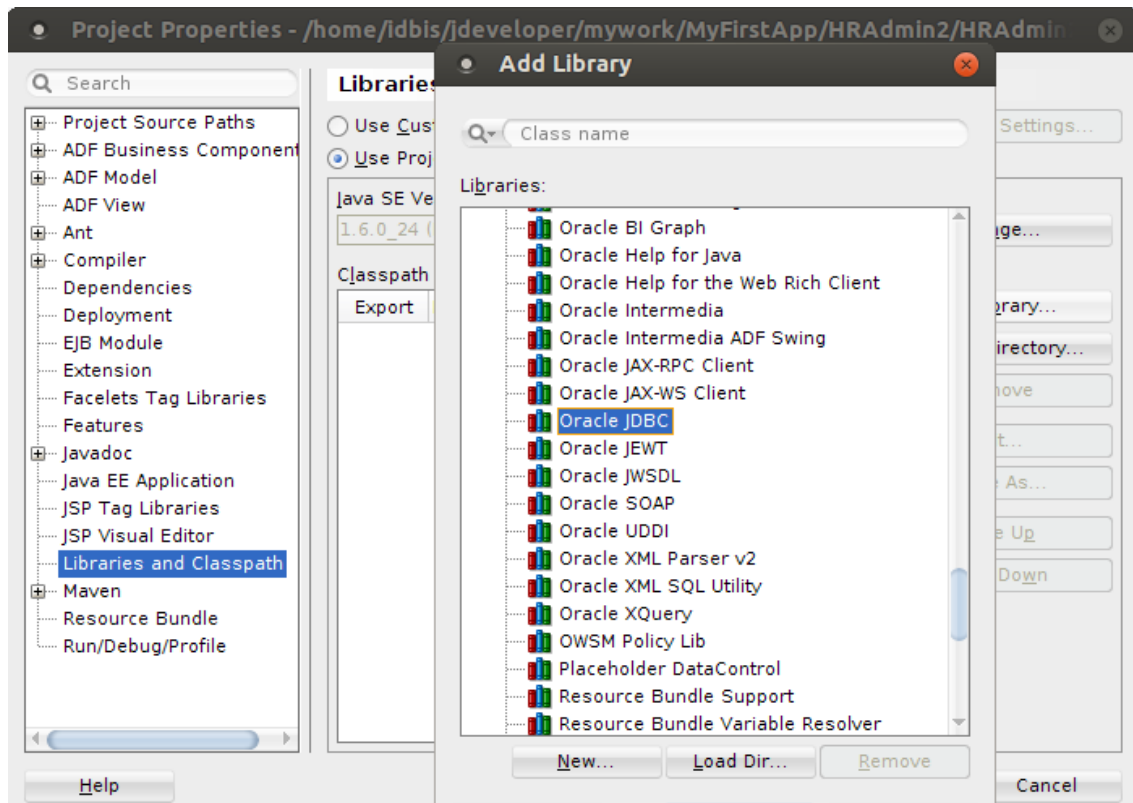
cuadro de diálogo de creación de proyectos, damos un nombre al proyecto, use **HRAdmin** como en el ejemplo u otro que usted elija. Pulse **Finish** para terminar.



nuevo proyecto que hemos creado aparecerán en el navegador de aplicaciones. Seleccione el proyecto y guarde todo pulsando el botón **Save All**.

## Añadir las librerías de JDBC

1. Seleccione el proyecto en el navegador de aplicaciones, pulse botón derecho y seleccione la opción **Project Properties** del menú.
2. Se habrá abierto el cuadro de diálogo de las propiedades del proyecto. Pulse el nodo **Libraries and Classpath**, y pulse el botón **Add Library**.



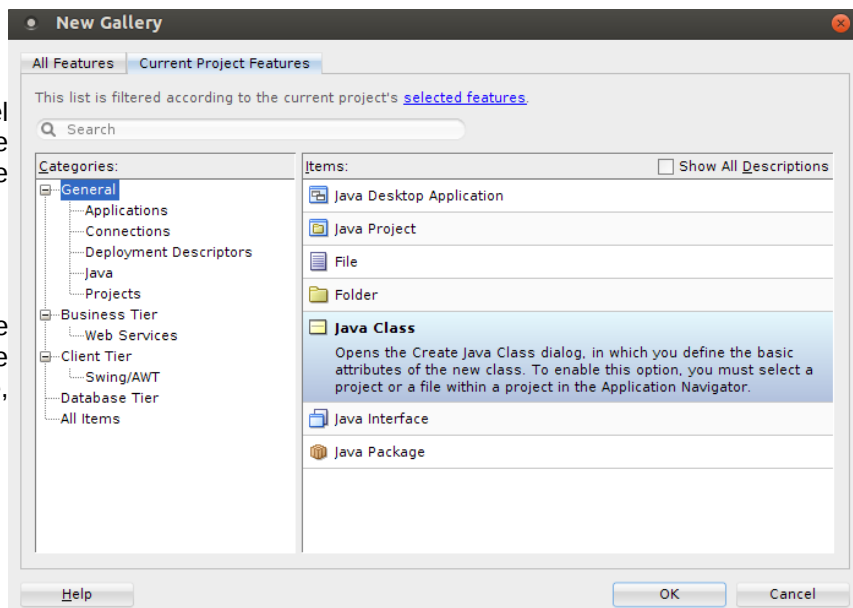
3. Viendo las librerías disponibles, busque en la lista la librería **Oracle JDBC**. Selecciónela y pulse en **OK** para terminar.

### Crear una clase Java para el programa

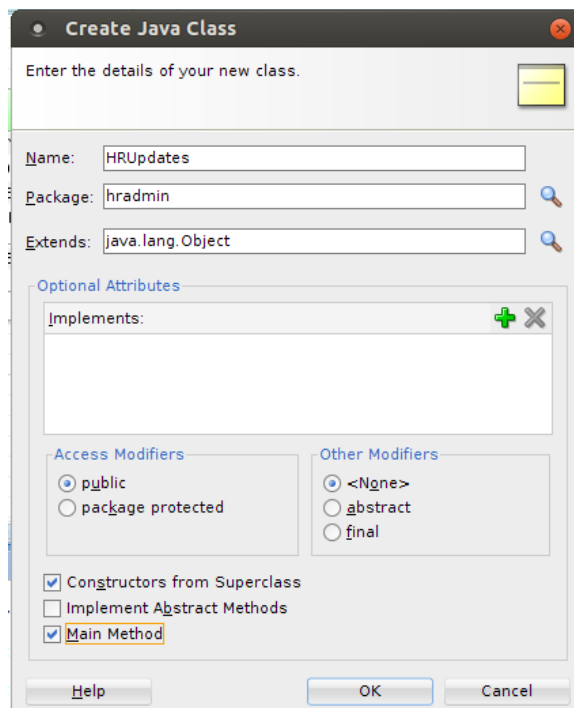
1. Seleccione el proyecto en el navegador y posteriormente haga clic con el botón derecho. Seleccione la opción **New...** del menú contextual.

2. En la “**New Gallery**”, mantenga la categoría por defecto “General”, y seleccione “**Java Class**” de la lista de elementos. Posteriormente, haga clic en **OK**.

3. En el diálogo de creación de clases Java (**Create Java Class**), de un nombre a la clase, por ejemplo



**HRUpdates**. Seleccione los atributos adicionales **Public**, **Generate Default Constructor** y **Generate Main Method**. Pulse **OK** para terminar.



## Importar los paquetes necesarios para usar JDBC

Para poder usar JDBC en una clase es necesario importar los siguientes paquetes:

```
java.sql.*  
java.sql.SQLException.*  
oracle.jdbc.driver.*
```

Incorpore las sentencias **import** correspondientes. El código resultante queda como sigue:

A screenshot of a Java IDE window titled 'JDeveloper Welcome' and 'HRUpdates.java'. The code is as follows:

```
package mypackage;  
import java.sql.*;  
import java.sql.SQLException.*;  
import oracle.jdbc.driver.*;  
  
public class HRUpdates  
{  
    public HRUpdates()  
    {  
    }  
    /**  
     *  
     * @param args  
     */  
    public static void main(String[] args)  
    {  
        HRUpdates hRUpdates = new HRUpdates();  
    }  
}
```

The line `import oracle.jdbc.driver.*;` is highlighted in yellow.

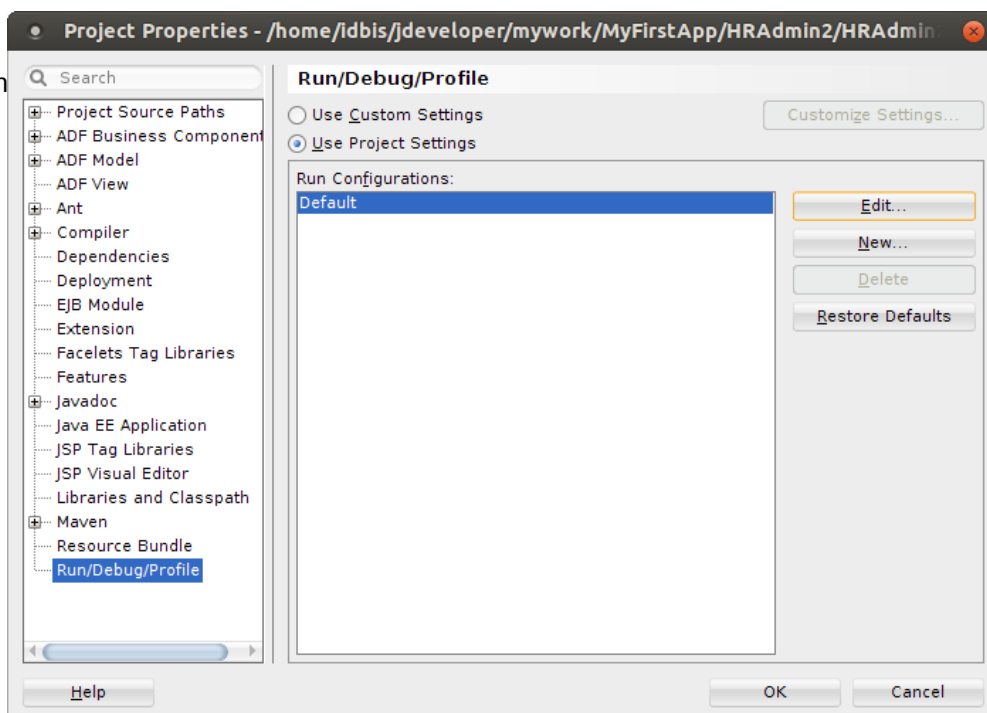
## Establecer Parámetros para una Aplicación

La aplicación permitirá a sus usuarios especificar como parámetros para la aplicación los valores para el nuevo departamento que se va a crear, el identificador del departamento, su nombre y su código de localización, así como los datos del empleado que dirigirá el nuevo departamento.

Para que JDeveloper pase, cada vez que ejecutemos la aplicación desde el entorno de desarrollo, unos parámetros específicos, deberemos de realizar las siguientes acciones.

1. Haga doble clic en el nodo del proyecto en el navegador. De esta forma abrirá el diálogo de propiedades del proyecto. Seleccione el nodo **Run/Debug/Profile**. Seleccionamos la configuración por defecto y le damos a **Edit...**

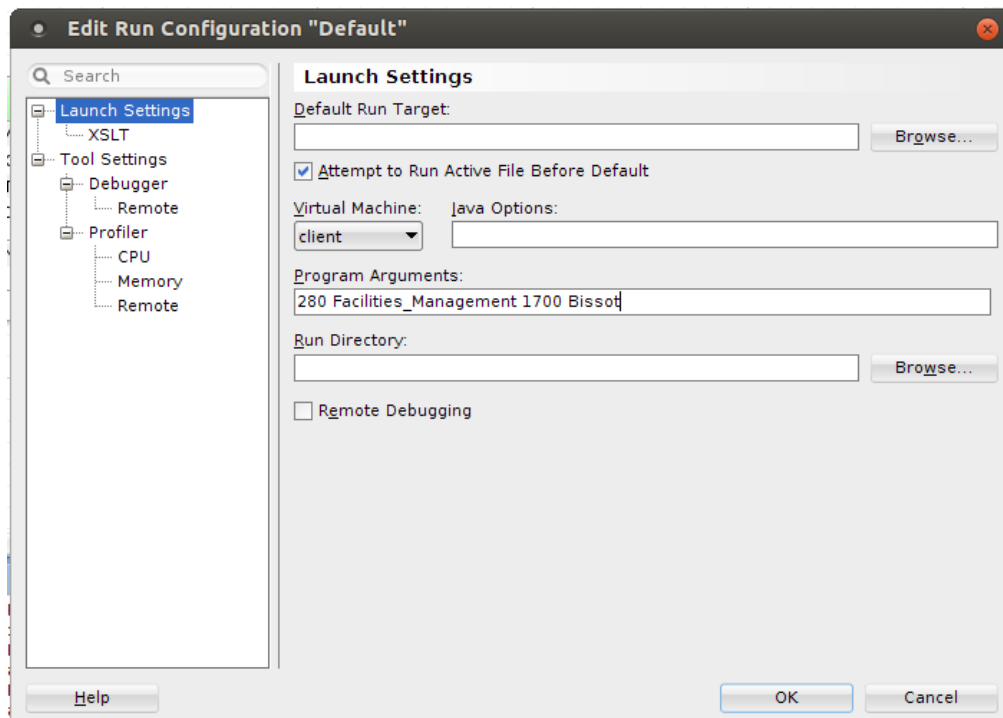
2. En



**Launch Settings**, en el campo de texto **Program Arguments**, que se emplea para indicar los argumentos que se pasarán por línea de comando al programa, escriba lo siguientes valores, que representan los datos del nuevo departamento (Identificador, nombre, localización y director): **280 Facilities\_Management 1700 Bissot**



3.



Modifique el método principal (**main**) de la clase **HRUpdates**, de tal forma que éste propague cualquier excepción. Para ello añada después de la cabecera de dicho método y antes de la apertura de la llave del cuerpo del método **throws Exception**.

```

JDeveloper Welcome | HRUpdates.java
package mypackage;
import java.sql.*;
import java.sql.SQLException.*;
import oracle.jdbc.driver.*;

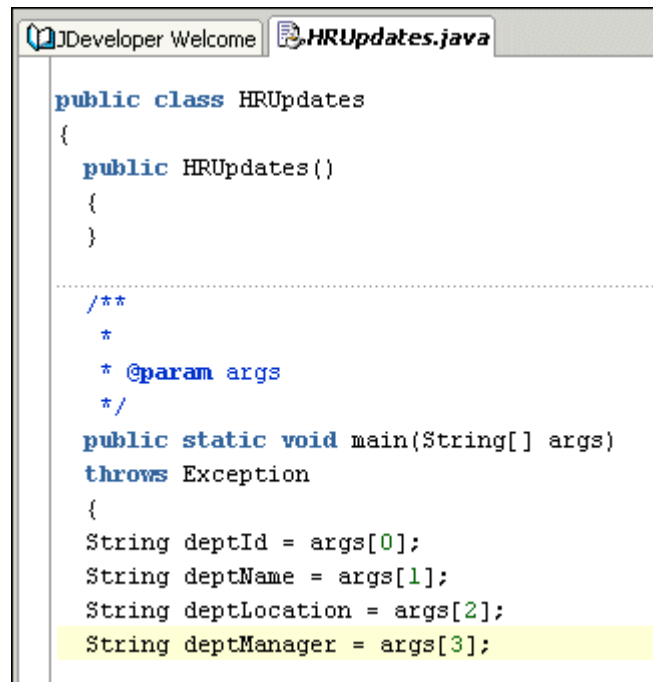
public class HRUpdates
{
    public HRUpdates()
    {
    }

    /**
     *
     * @param args
     */
    public static void main(String[] args)
    throws Exception
    {
        HRUpdates hRUpdates = new HRUpdates();
    }
}

```

4. Declare las siguientes variables dentro del método principal (main). Estas variables estarán inicializadas según diferentes datos que se pasan como argumento al programa desde la línea de comandos de la forma que se muestra a continuación:

```
String deptId = args[0];  
String deptName = args[1];  
String deptLocation = args[2];  
String deptManager = args[3];
```



```
IDE Developer Welcome | HRUpdates.java  
  
public class HRUpdates  
{  
    public HRUpdates()  
    {  
    }  
}  
  
/**  
 *  
 * @param args  
 */  
public static void main(String[] args)  
throws Exception  
{  
    String deptId = args[0];  
    String deptName = args[1];  
    String deptLocation = args[2];  
    String deptManager = args[3];  
}
```

5. Después de la declaración e inicialización de las variables anteriores añada una referencia a la clase HRUpdates e inicialícela con una nueva instancia de la misma. Para ello incorpore el siguiente código.

```
HRUpdates hrUpdates = new HRUpdates();
```



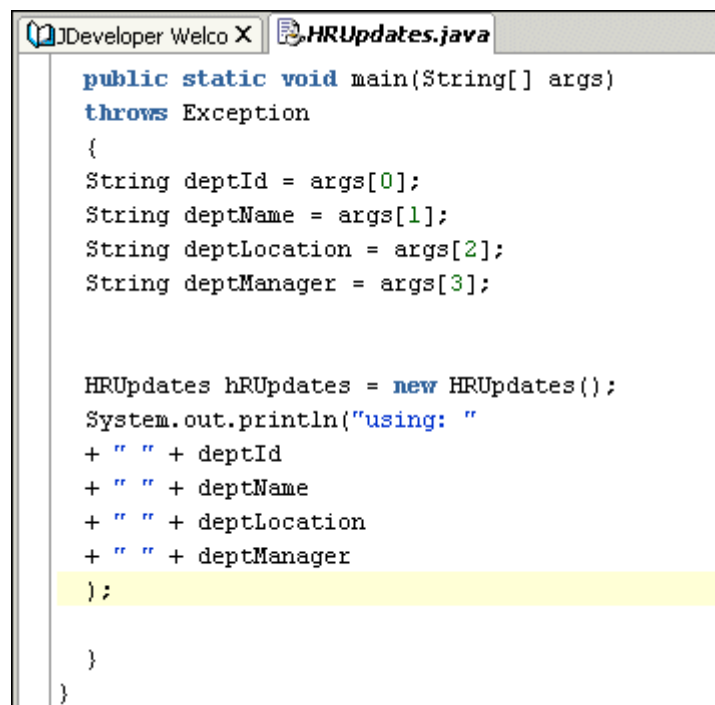
```
public class HRUpdates
{
    public HRUpdates()
    {
    }

    /**
     *
     * @param args
     */
    public static void main(String[] args)
    throws Exception
    {
        String deptId = args[0];
        String deptName = args[1];
        String deptLocation = args[2];
        String deptManager = args[3];

        HRUpdates hRUpdates = new HRUpdates();
    }
}
```

6. Añada a continuación código para mostrar por pantalla el valor de los argumentos que hemos recibido.

***System.out.println("using: " + " " + deptId + " " + deptName + " " + deptLocation + " " + deptManager);***

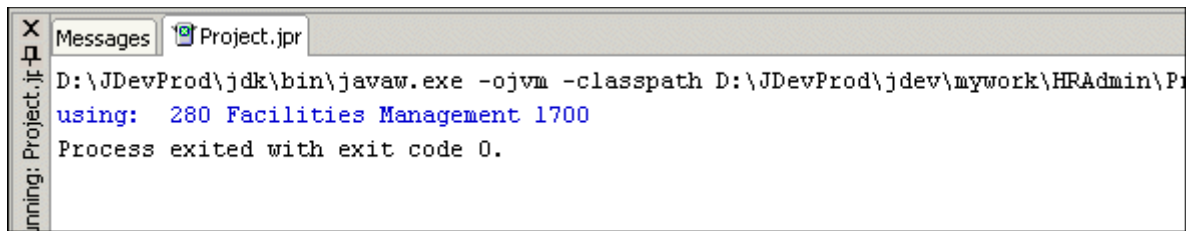


```
public static void main(String[] args)
    throws Exception
{
    String deptId = args[0];
    String deptName = args[1];
    String deptLocation = args[2];
    String deptManager = args[3];

    HRUpdates hRUpdates = new HRUpdates();
    System.out.println("using: "
        + " " + deptId
        + " " + deptName
        + " " + deptLocation
        + " " + deptManager
    );

}
}
```

7. Ejecute el programa. Haga clic con el botón derecho sobre el editor de código y seleccione la opción **Run** del menú contextual. Si se han seguido los pasos anteriores correctamente, el programa deberá de compilar sin problemas y se mostrará el valor de los parámetros de entrada en la ventana de mensajes (**Log Window**).



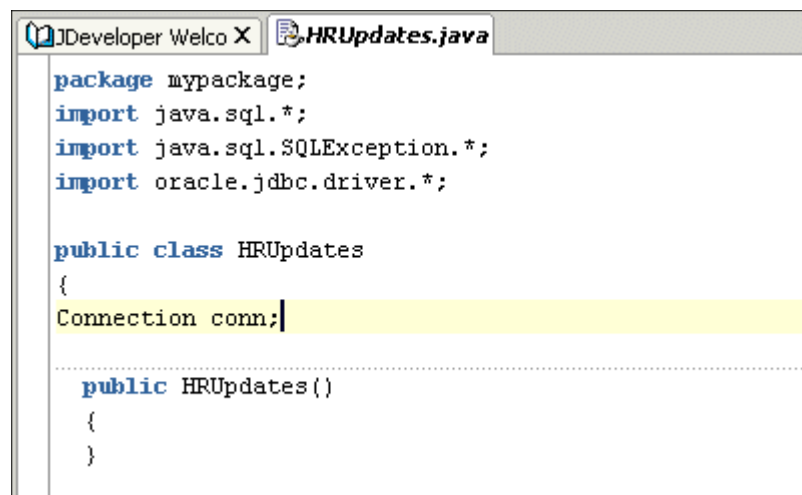
```
D:\JDevProd\jdk\bin\javaw.exe -ojvm -classpath D:\JDevProd\jdev\mywork\HRAdmin\Pr
using: 280 Facilities Management 1700
Process exited with exit code 0.
```

## Conectándose a una base de datos

Para poder conectarse a una base de datos usando JDBC, es necesario registrar el controlador (**driver**) que usaremos para la conexión (en nuestro caso el controlador JDBC para base de datos Oracle) en el gestor de controladores de JDBC (**driver manager**). Existen diversas maneras de llevar esta tarea a cabo, en este ejemplo emplearemos el método **register.Driver()** de la clase **java.sql.DriverManager**. Una vez registrado el controlador, invocaremos el método **getConnection()** para establecer una conexión con la base de datos. Cuando se establece una conexión a una base de datos esta tiene el modo **autocommit** activado, lo que significa que después de cada sentencia SQL se envía automáticamente una validación de los cambios (**commit**). Este modo se puede desactivar, de tal forma que será necesario realizar de forma que cada grupo de sentencias que conformen una transacción será necesario validar explícitamente los cambios llamando al método **commit()** o deshacer los cambios llamando al método **rollback()**. Este ejemplo muestra como conectar a una base de datos y deshabilitar el modo **autocommit**.

1. Añada a la clase HRUpdates un atributo de tipo **Connection**, usando la declaración que se muestra a continuación. Este atributo contendrá una referencia a la conexión a la base de datos que estableceremos.

**Connection conn;**



```
package mypackage;
import java.sql.*;
import java.sql.SQLException.*;
import oracle.jdbc.driver.*;

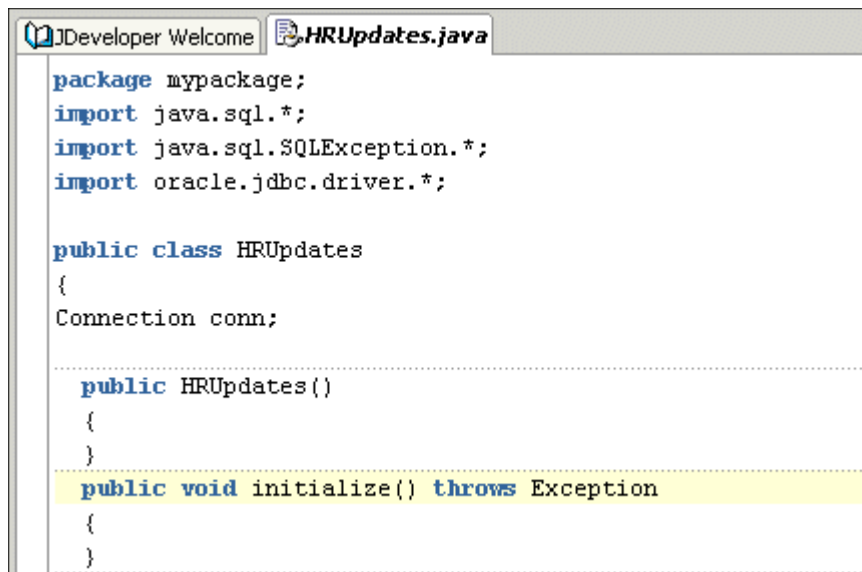
public class HRUpdates
{
    Connection conn;

    .....

    public HRUpdates()
    {
    }
}
```

2. Cree un método llamado initialize que tendrá como objetivo establecer la conexión a la base de datos. El método deberá de propagar cualquier excepción que se produzca durante su ejecución. Podrá añadir este método incluyendo este método en la clase HRUpdates:

```
public void initialize() throws Exception
{
}
}
```



```
package mypackage;
import java.sql.*;
import java.sql.SQLException.*;
import oracle.jdbc.driver.*;

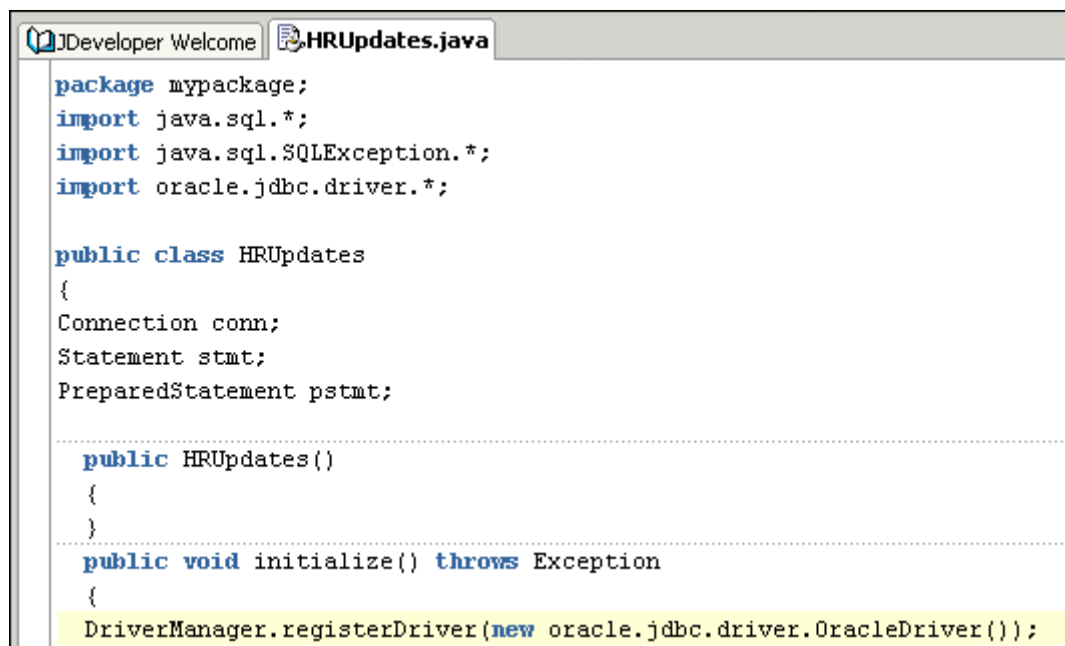
public class HRUpdates
{
    Connection conn;

    public HRUpdates()
    {
    }

    public void initialize() throws Exception
    {
    }
}
```

3. Añada al método anterior el código para registrar el controlador en el gestor de controladores JDBC:

***DriverManager.registerDriver(  
new oracle.jdbc.driver.OracleDriver());***



```
package mypackage;
import java.sql.*;
import java.sql.SQLException.*;
import oracle.jdbc.driver.*;

public class HRUpdates
{
    Connection conn;
    Statement stmt;
    PreparedStatement pstmt;


    public HRUpdates()
    {
    }

    public void initialize() throws Exception
    {
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
    }
}
```

4. Posteriormente añade el siguiente código para establecer una conexión con la base de datos. Modifique dicho código para adaptarlo a los datos de conexión de su base de datos. Sustituyendo **hostname** por el nombre de la máquina en la que está instalado el servidor de bases de datos, **dbname** por el nombre de la base de datos a la que se desea conectar, la primera cadena que contiene **hr** por el nombre de su usuario en la base de datos, y la segunda

por su contraseña. Si fuera necesario también puede modificar el puerto en el que escucha el servidor de base de datos, en el ejemplo el puerto **1521**.

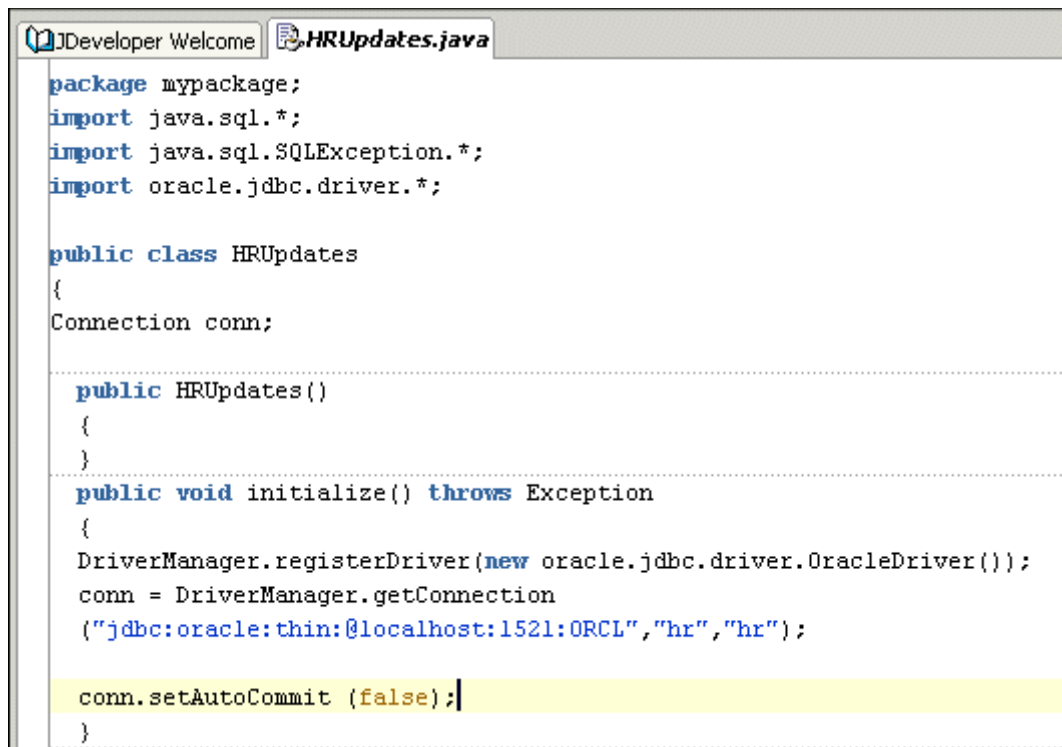
```
conn = DriverManager.getConnection  
("jdbc:oracle:thin:@hostname:1521:dbname",  
"hr","hr");
```



```
package mypackage;  
import java.sql.*;  
import java.sql.SQLException.*;  
import oracle.jdbc.driver.*;  
  
public class HRUpdates  
{  
    Connection conn;  
  
    public HRUpdates()  
    {  
    }  
  
    public void initialize() throws Exception  
    {  
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());  
        conn = DriverManager.getConnection  
        ("jdbc:oracle:thin:@localhost:1521:ORCL","hr","hr");  
    }  
}
```

5. Para deshabilitar la autovalición, añade la siguiente línea de código:

```
conn.setAutoCommit (false);
```



```
package mypackage;
import java.sql.*;
import java.sql.SQLException.*;
import oracle.jdbc.driver.*;

public class HRUpdates
{
    Connection conn;

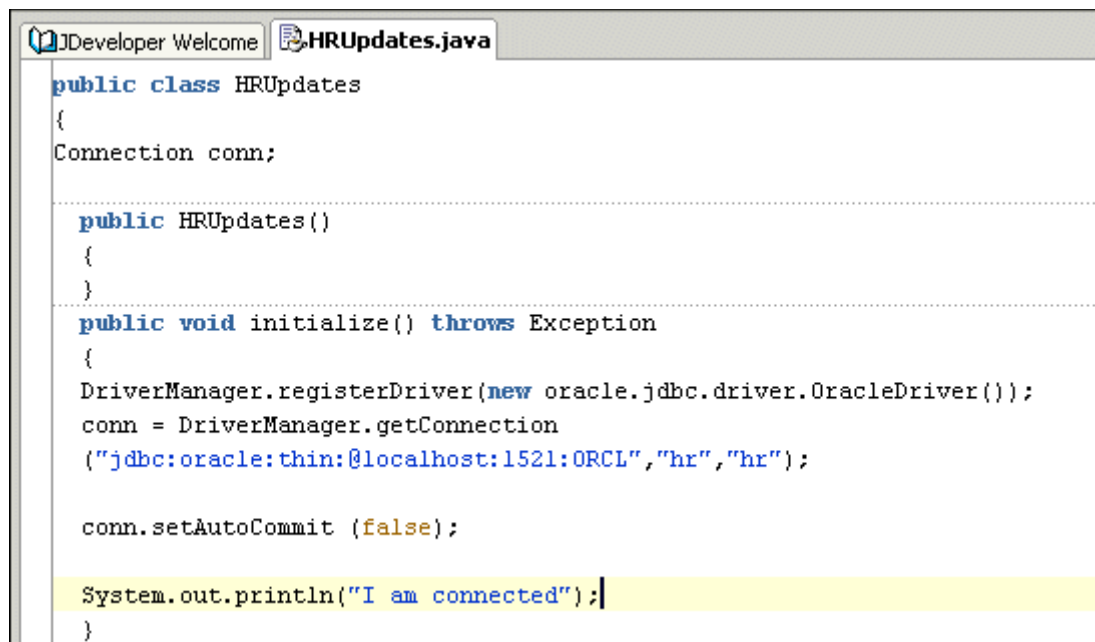
    public HRUpdates()
    {
    }

    public void initialize() throws Exception
    {
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
        conn = DriverManager.getConnection
            ("jdbc:oracle:thin:@localhost:1521:ORCL","hr","hr");

        conn.setAutoCommit (false);
    }
}
```

6. Añada un mensaje para mostrar indicando que se ha establecido la conexión. Puede usar la siguiente línea de código:

***System.out.println("I am connected");***



```
public class HRUpdates
{
    Connection conn;

    public HRUpdates()
    {
    }

    public void initialize() throws Exception
    {
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
        conn = DriverManager.getConnection
            ("jdbc:oracle:thin:@localhost:1521:ORCL","hr","hr");

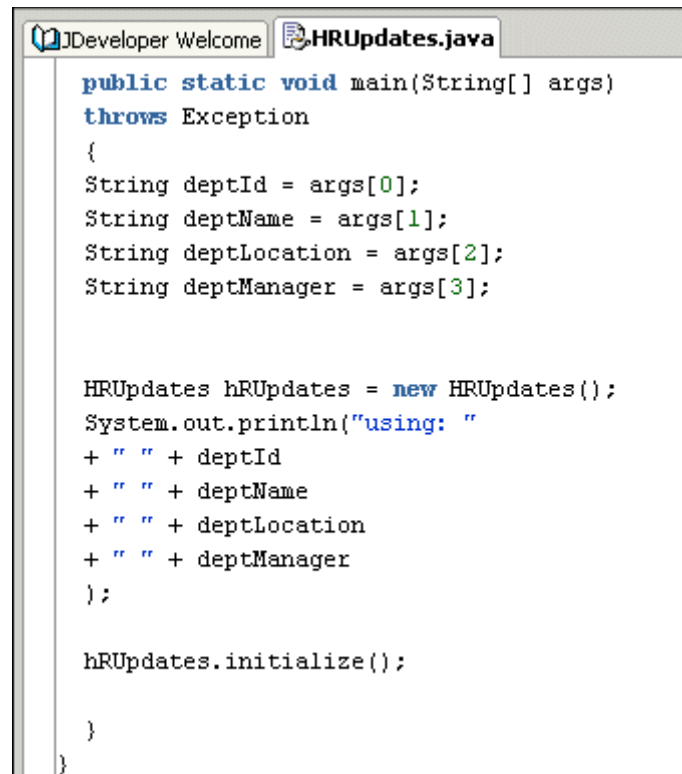
        conn.setAutoCommit (false);

        System.out.println("I am connected");
    }
}
```



7. Incluya en el método principal una llamada al método anterior para que se establezca una conexión a la base de datos al ejecutar el programa. El código de esta llamada es el siguiente:

***hRUpdates.initialize();***

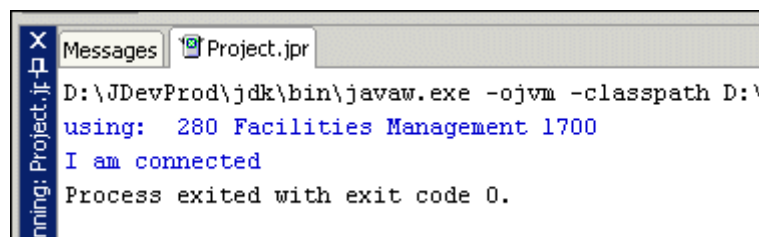


```
public static void main(String[] args)
throws Exception
{
    String deptId = args[0];
    String deptName = args[1];
    String deptLocation = args[2];
    String deptManager = args[3];

    HRUpdates hRUpdates = new HRUpdates();
    System.out.println("using: "
+ " " + deptId
+ " " + deptName
+ " " + deptLocation
+ " " + deptManager
);

    hRUpdates.initialize();
}
```

8. Ejecute el programa para comprobar que se establece satisfactoriamente una conexión a la base de datos. En caso de que haya algún problema se obtendrá una excepción con un mensaje descriptivo.



```
Messages Project.jpr
D:\JDevProd\jdk\bin\javaw.exe -ojvm -classpath D:\...
using: 280 Facilities Management 1700
I am connected
Process exited with exit code 0.
```

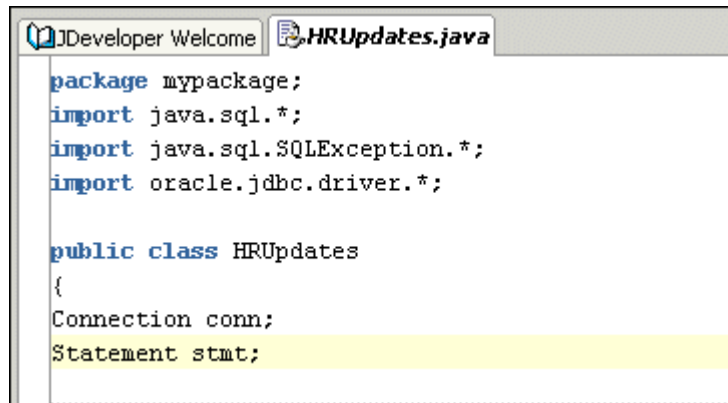
## Ejecutar consultas

Una vez registrado el controlador y establecida la conexión con la base de datos el programa ya está listo para lanzar consultas sobre los datos. Antes de añadir el departamento comprobaremos que el nombre del empleado que lo dirigirá es válido. En este programa eso se realizará obteniendo la lista de empleados cuyo apellido coincide con el que se incluyó como parámetro de entrada al programa.

Para ejecutar consultas sobre una base de datos es necesario crear un objeto **Statement** empleando el método `createStatement()`. Este tipo de objetos representarán una consulta, y se indicará que se desea ejecutar dicha consulta empleando su método **`executeQuery()`**. Este método recibe una sentencia SQL como entrada y devuelve un objeto **ResultSet** que representa el resultado de la ejecución de dicha sentencia. Para iterar sobre las filas resultantes de la ejecución de la sentencia emplearemos el método `next()`. Siga los siguientes pasos para ejecutar la consulta que determinará si el empleado que dirigirá el departamento es válido:

1. Incluya un atributo de tipo `Statement` en la clase `HRUpdates`. Emplee el siguiente código:

**`Statement stmt;`**



```
package mypackage;
import java.sql.*;
import java.sql.SQLException.*;
import oracle.jdbc.driver.*;

public class HRUpdates
{
    Connection conn;
    Statement stmt;
```

2. Cree un método llamado `listEmp` que contendrá el código para la ejecución de la consulta que obtiene los empleados según sus apellidos. Este método devolverá un número entero indicando el número de empleados encontrados. Emplee el siguiente código para incorporar dicho método.

```
public int listEmp(String lastName) throws Exception
{
}
```



```
public HRUpdates()
{
}

public void initialize() throws Exception
{
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
    conn = DriverManager.getConnection
    ("jdbc:oracle:thin:@localhost:1521:ORCL","hr","hr");

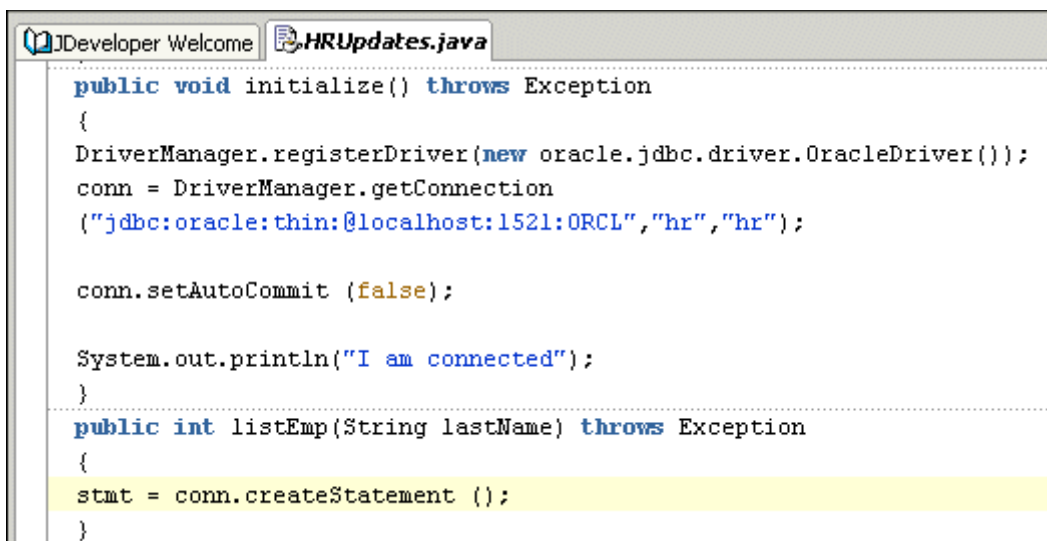
    conn.setAutoCommit (false);

    System.out.println("I am connected");
}

public int listEmp(String lastName) throws Exception
{
}
```

3. Añada dentro del método que ha creado el código necesario para obtener un objeto Statement que nos permita ejecutar la consulta. El código es el siguiente:

***stmt = conn.createStatement ();***



```
public void initialize() throws Exception
{
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
    conn = DriverManager.getConnection
    ("jdbc:oracle:thin:@localhost:1521:ORCL","hr","hr");

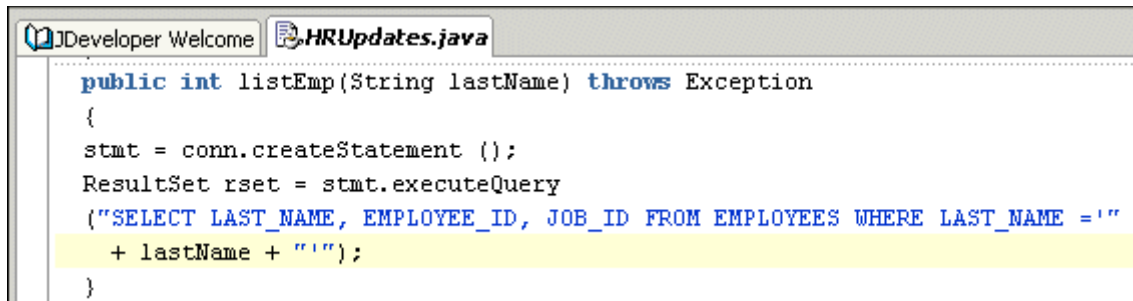
    conn.setAutoCommit (false);

    System.out.println("I am connected");
}

public int listEmp(String lastName) throws Exception
{
    stmt = conn.createStatement ();
}
```

4. Emplee el método ***executeQuery()*** para representar la consulta que comprobará la validez del empleado cuyos apellidos se han pasado como argumento al método. Para ello, añada el siguiente código:

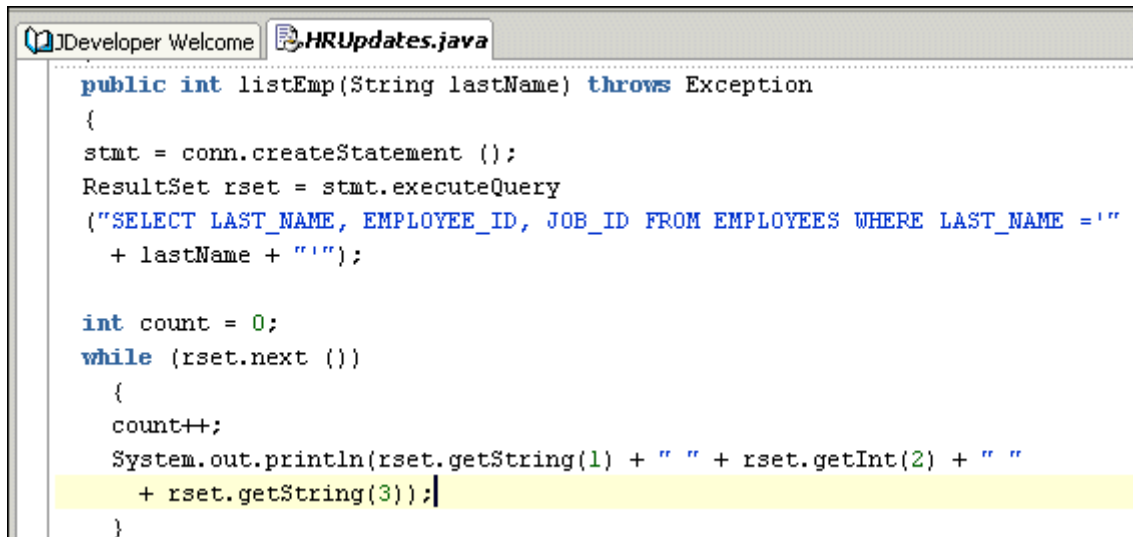
***ResultSet rset = stmt.executeQuery ("SELECT LAST\_NAME, EMPLOYEE\_ID, JOB\_ID  
FROM EMPLOYEES where LAST\_NAME = '' + lastName + '');***



```
public int listEmp(String lastName) throws Exception
{
    stmt = conn.createStatement ();
    ResultSet rset = stmt.executeQuery
    ("SELECT LAST_NAME, EMPLOYEE_ID, JOB_ID FROM EMPLOYEES WHERE LAST_NAME =" +
    + lastName + "'");
}
```

5. Obtenga las filas resultantes de la consulta anterior, contando cuantas ha obtenido e imprimiendo por pantalla los datos de los empleados obtenidos. Incorpore al método el siguiente código:

```
int count = 0;
while (rset.next ())
{
    count++;
    System.out.println(rset.getString(1) + " " + rset.getInt(2) + " " + rset.getString(3));
}
```

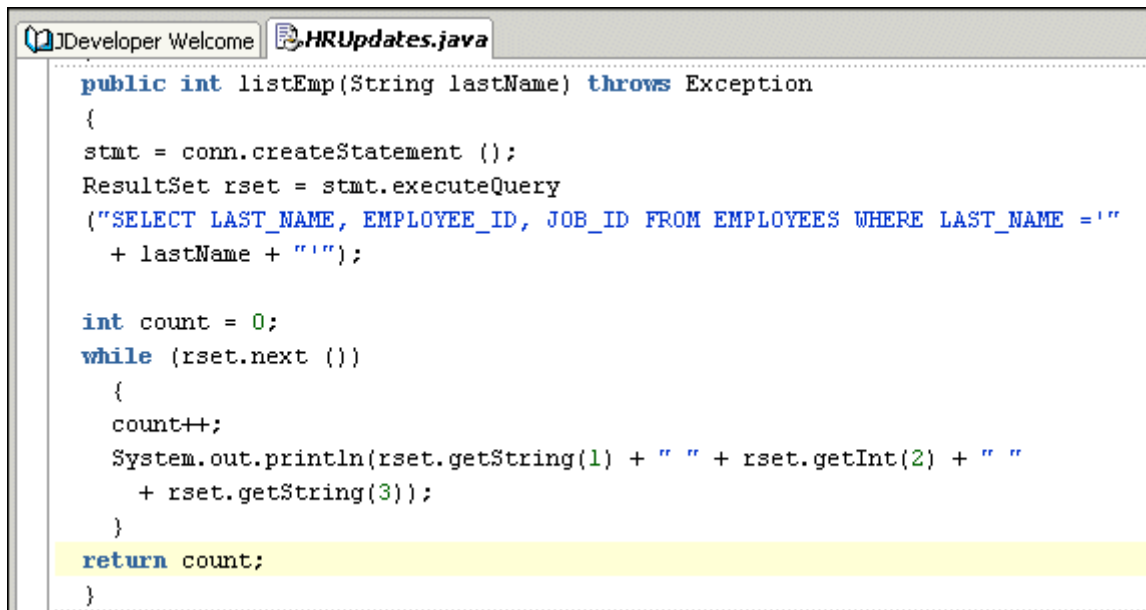


```
public int listEmp(String lastName) throws Exception
{
    stmt = conn.createStatement ();
    ResultSet rset = stmt.executeQuery
    ("SELECT LAST_NAME, EMPLOYEE_ID, JOB_ID FROM EMPLOYEES WHERE LAST_NAME =" +
    + lastName + "'");

    int count = 0;
    while (rset.next ())
    {
        count++;
        System.out.println(rset.getString(1) + " " + rset.getInt(2) + " " +
        + rset.getString(3));
    }
}
```

6. Devuelva el número de filas obtenidas como resultado de la ejecución de la consulta. Para ello, añada el siguiente código:

```
return count;
```

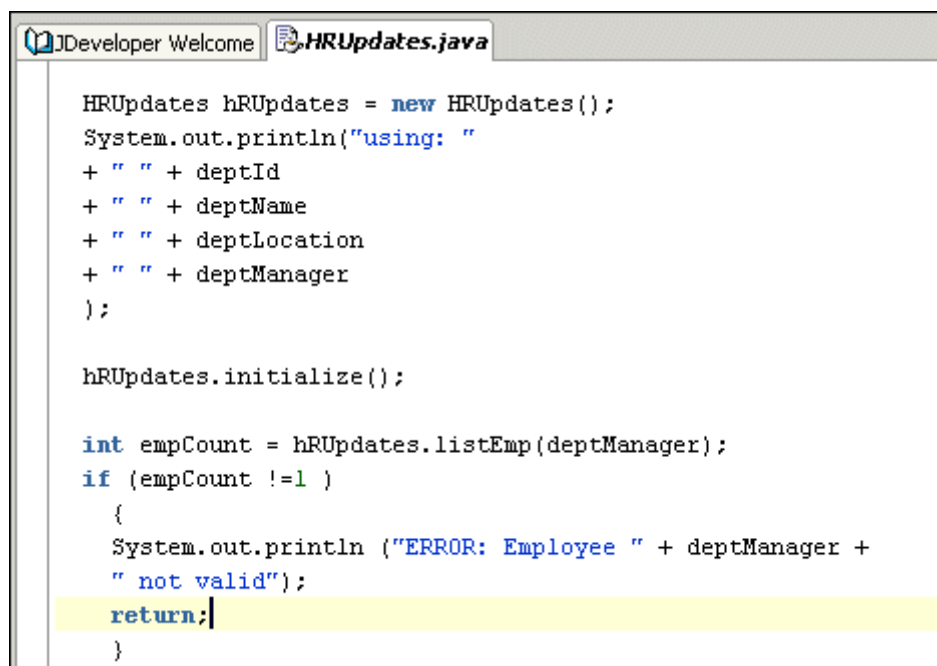
A screenshot of a Java IDE window titled 'HRUpdates.java'. The code defines a public method 'listEmp' that takes a 'lastName' string and throws an 'Exception'. Inside the method, a 'stmt' object is created from a 'conn', and a 'ResultSet' 'rset' is obtained by executing a SQL query: 'SELECT LAST\_NAME, EMPLOYEE\_ID, JOB\_ID FROM EMPLOYEES WHERE LAST\_NAME = ' + lastName + '''. A 'count' variable is initialized to 0, and a 'while' loop iterates through the 'rset' using 'rset.next()'. Inside the loop, 'count' is incremented, and the first three columns of the result are printed: 'rset.getString(1)', 'rset.getInt(2)', and 'rset.getString(3)'. The method returns the 'count' value.

```
public int listEmp(String lastName) throws Exception
{
    stmt = conn.createStatement ();
    ResultSet rset = stmt.executeQuery
    ("SELECT LAST_NAME, EMPLOYEE_ID, JOB_ID FROM EMPLOYEES WHERE LAST_NAME ='"
     + lastName + "'");

    int count = 0;
    while (rset.next ())
    {
        count++;
        System.out.println(rset.getString(1) + " " + rset.getInt(2) + " "
        + rset.getString(3));
    }
    return count;
}
```

7. Añada el siguiente código al método principal del programa. Este código ejecuta la consulta llamando al método que acabamos de crear y comprueba si el número de empleados obtenido es 1. En caso de que el número de empleados no sea el esperado, imprimirá un mensaje de error y terminará la ejecución del programa.

```
int empCount = hRUpdates.listEmp(deptManager);
if (empCount !=1 )
{
    System.out.println ("ERROR: Employee " + deptManager + " not valid");
    return;
}
```

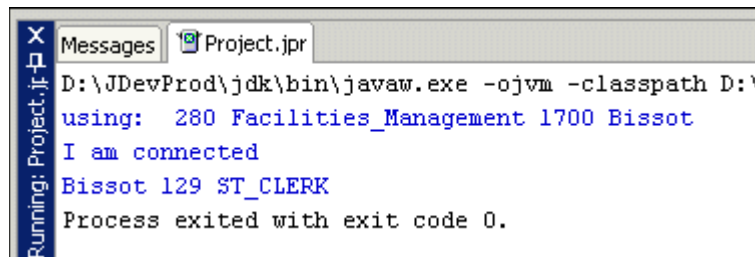
A screenshot of a Java IDE window titled 'HRUpdates.java'. The code shows the initialization of an 'hRUpdates' object, printing department information, and calling the 'listEmp' method. It then checks if the returned 'empCount' is not equal to 1. If true, it prints an error message and returns. The 'return;' statement is highlighted in yellow.

```
HRUpdates hRUpdates = new HRUpdates();
System.out.println("using: "
+ " " + deptId
+ " " + deptName
+ " " + deptLocation
+ " " + deptManager
);

hRUpdates.initialize();

int empCount = hRUpdates.listEmp(deptManager);
if (empCount !=1 )
{
    System.out.println ("ERROR: Employee " + deptManager +
    " not valid");
    return;
}
```

8. Ejecute el programa para comprobar que el empleado cuyo apellido se incluye como argumento del programa es válido.



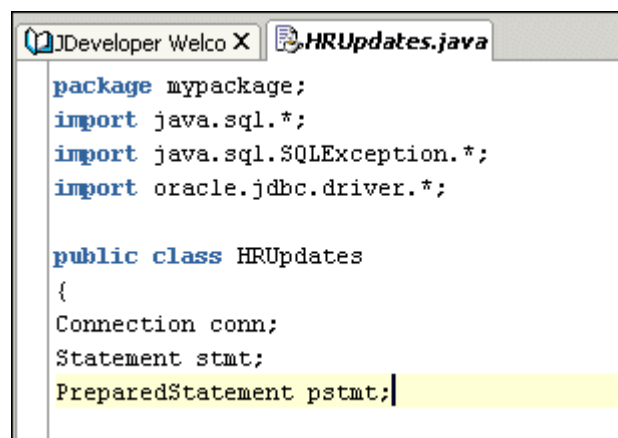
```
Messages | Project.jpr
D:\JDevProd\jdk\bin\javaw.exe -ojvm -classpath D:\
using: 280 Facilities_Management 1700 Bissot
I am connected
Bissot 129 ST_CLERK
Process exited with exit code 0.
```

## Insertar filas en una tabla

Una vez que hemos comprobado que es válido el empleado que dirigirá el nuevo departamento, pasaremos a insertar dicho departamento en la base de datos. La tabla **Departments** posee las columnas **Department\_Id**, **Department\_Name**, **Manager\_Id** (opcional) y **Location\_Id** en las que realizaremos la inserción de los datos que se pasan como argumento por línea de comandos. Debido a que el programa cada vez que crear una conexión desactiva la autovalidación, esta ha de hacerse de forma explícita en el código, aunque para realizar pruebas hasta que el programa sea definitivo es recomendable no realizar dichas validaciones. En este apartado se incluirá código para el tratamiento de excepciones provocadas por errores de SQL. El objetivo de este tratamiento es detectar la violaciones de clave primaria que pueden provocar la inserción del mismo departamento dos veces. Finalmente, una vez que se haya realizado la inserción, confirmaremos ésta consultando el contenido de la tabla **Departments**.

1. Añada a la clase HRUpdates un atributo de tipo PreparedStatement. Para ello, use le siguiente línea de código:

**PreparedStatement pstmt;**



```
JDeveloper Welco X | HRUpdates.java
package mypackage;
import java.sql.*;
import java.sql.SQLException.*;
import oracle.jdbc.driver.*;

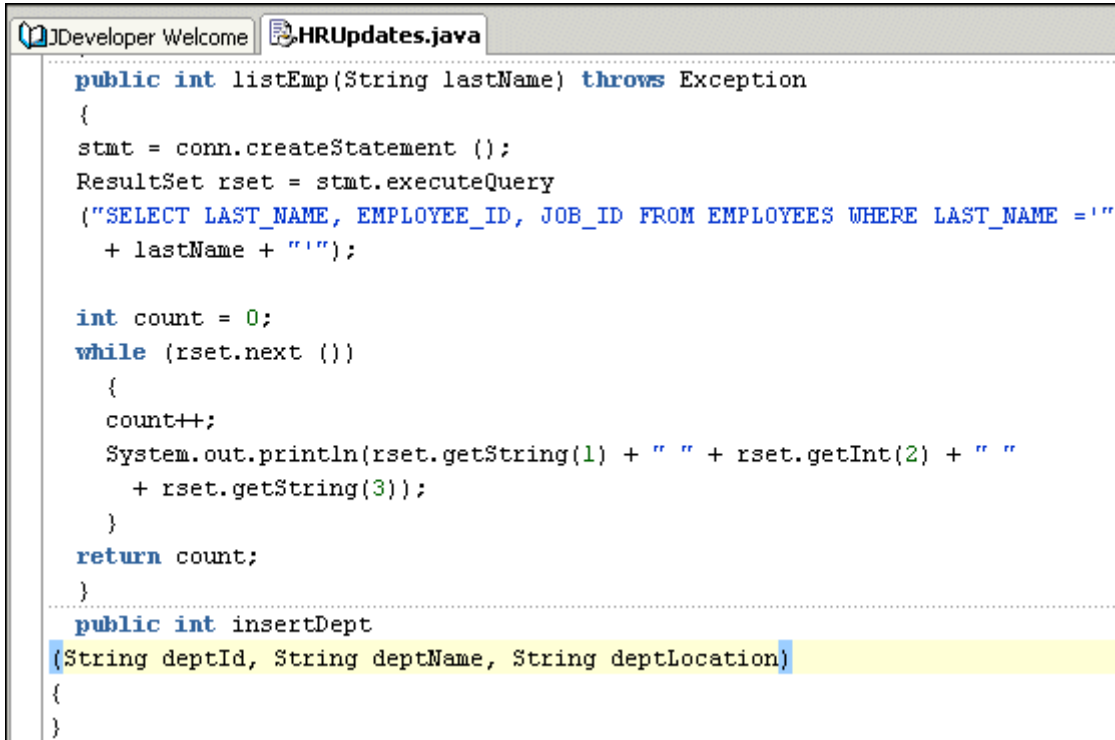
public class HRUpdates
{
    Connection conn;
    Statement stmt;
    PreparedStatement pstmt;
```

2. Cree un método para contener el código para realizar la inserción del departamento. Este método se llamará **insertDept** y recibirá como argumento los datos de dicho departamento. Devolviendo un número entero que será 0 cuando todo haya ido bien y 0 en caso contrario. Emplee una declaración similar a la siguiente:

```

public int insertDept (String deptId, String deptName, String deptLocation)
{
}

```



```

Developer Welcome HRUpdates.java
public int listEmp(String lastName) throws Exception
{
    stmt = conn.createStatement ();
    ResultSet rset = stmt.executeQuery
    ("SELECT LAST_NAME, EMPLOYEE_ID, JOB_ID FROM EMPLOYEES WHERE LAST_NAME =" +
    + lastName + "'");

    int count = 0;
    while (rset.next ())
    {
        count++;
        System.out.println(rset.getString(1) + " " + rset.getInt(2) + " "
        + rset.getString(3));
    }
    return count;
}

public int insertDept
(String deptId, String deptName, String deptLocation)
{
}

```

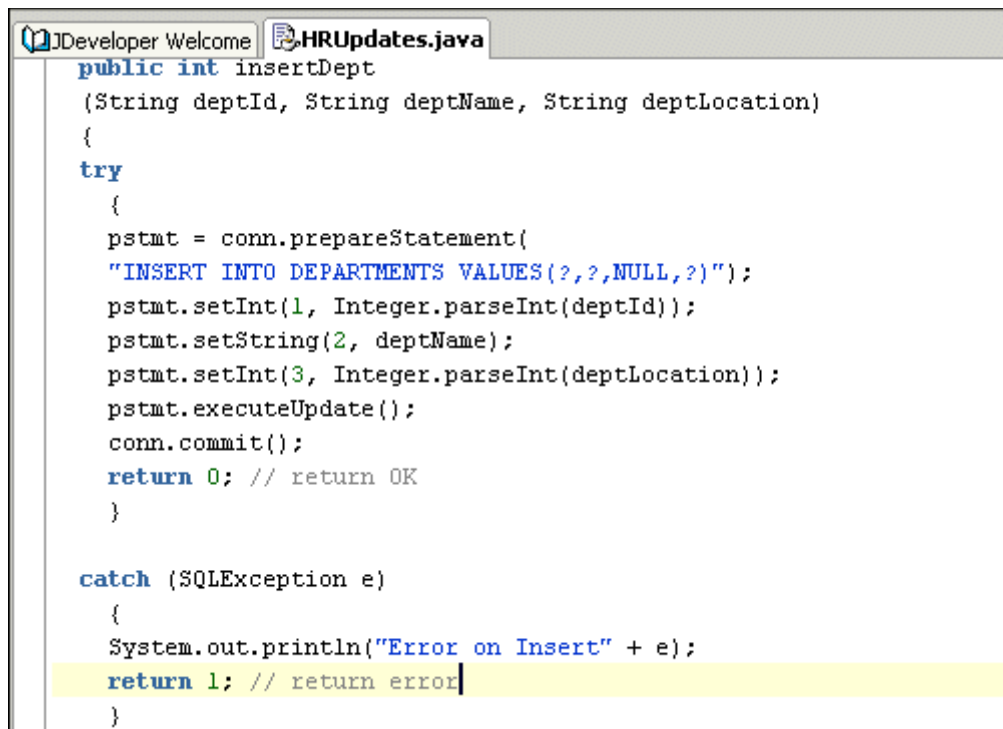
3. Ahora añade el código para ejecutar la inserción y tratar las posibles excepciones que se produzcan a realizar la misma. El código a insertar dentro del método es el siguiente:

```

try
{
    pstmt = conn.prepareStatement("INSERT INTO DEPARTMENTS VALUES(?,?,NULL,?)");
    pstmt.setInt(1, Integer.parseInt(deptId));
    pstmt.setString(2, deptName);
    pstmt.setInt(3, Integer.parseInt(deptLocation));
    pstmt.executeUpdate();
    conn.commit();
    return 0; // return OK
}
catch (SQLException e)
{
    System.out.println("Error on Insert" + e);
    return 1; // return error
}

```

Si realizará pruebas del funcionamiento de la clase antes de terminarla es recomendable que comente la línea **conn.commit();** para evitar que las modificaciones se validen y así poder ejecutar varias veces el programa obteniendo los mismos resultados.

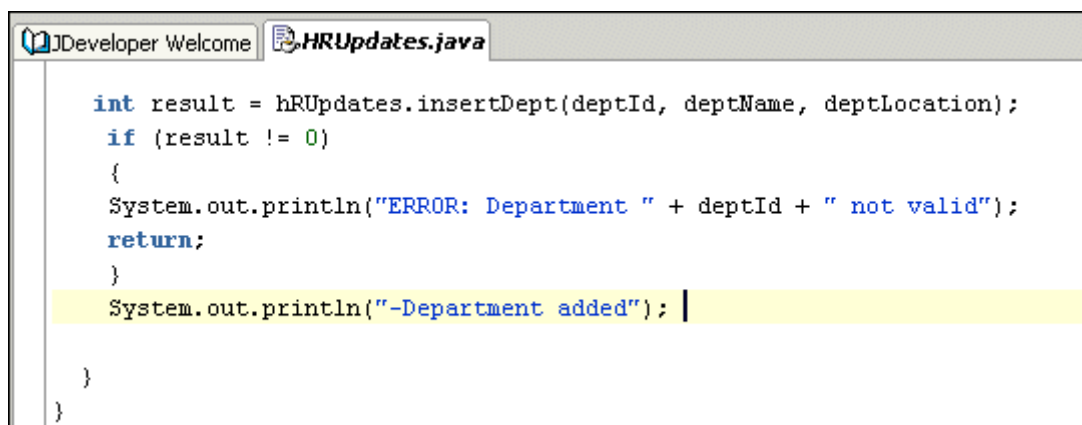


```
Developer Welcome HRUpdates.java
public int insertDept
(String deptId, String deptName, String deptLocation)
{
    try
    {
        pstmt = conn.prepareStatement(
            "INSERT INTO DEPARTMENTS VALUES(?,?,NULL,?)");
        pstmt.setInt(1, Integer.parseInt(deptId));
        pstmt.setString(2, deptName);
        pstmt.setInt(3, Integer.parseInt(deptLocation));
        pstmt.executeUpdate();
        conn.commit();
        return 0; // return OK
    }

    catch (SQLException e)
    {
        System.out.println("Error on Insert" + e);
        return 1; // return error
    }
}
```

4. Añada al código del método principal las sentencias necesarias para invocar al anterior método y comprobar si su ejecución ha sido satisfactoria. El código puede ser el siguiente:

```
int result = hRUpdates.insertDept(deptId, deptName, deptLocation);
if (result != 0){
    System.out.println("ERROR: Department " + deptId + " not valid");
    return;
}
System.out.println("-Department added");
```



```
Developer Welcome HRUpdates.java

int result = hRUpdates.insertDept(deptId, deptName, deptLocation);
if (result != 0)
{
    System.out.println("ERROR: Department " + deptId + " not valid");
    return;
}
System.out.println("-Department added");

}

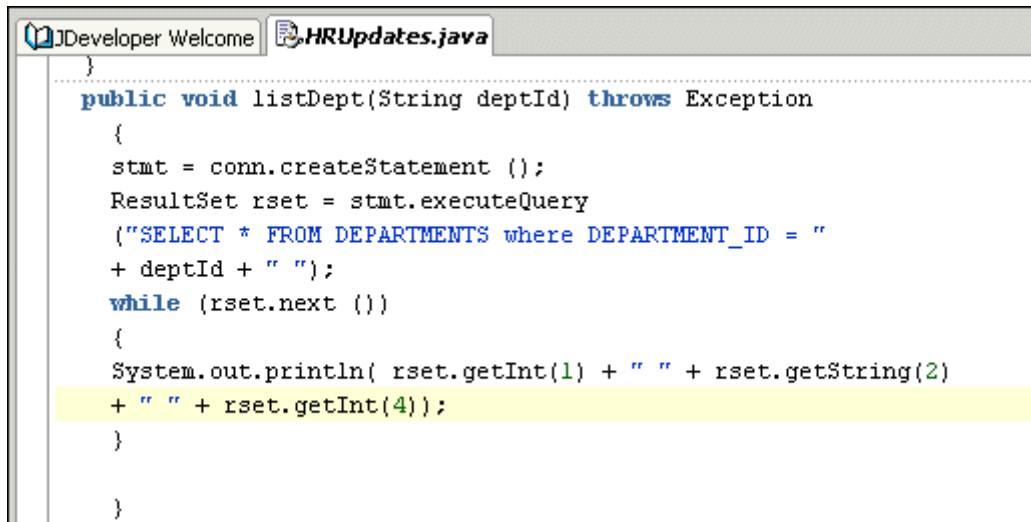
}
```

5. Posteriormente a la inserción comprobaremos que esta se ha realizado correctamente. Para ello, crearemos un método llamado listDept que realizará una consulta para obtener la fila que



se acaba de insertar. Este método funciona de forma similar al método `listEmployee` y se define según el siguiente código:

```
public void listDept(String deptId) throws Exception{  
    stmt = conn.createStatement ();  
    ResultSet rset = stmt.executeQuery ("SELECT * FROM DEPARTMENTS where  
DEPARTMENT_ID = " + deptId + " ");  
  
    while (rset.next ())  
        System.out.println( rset.getInt(1) + " " + rset.getString(2)+ " " + rset.getInt(4));  
    }  
}
```

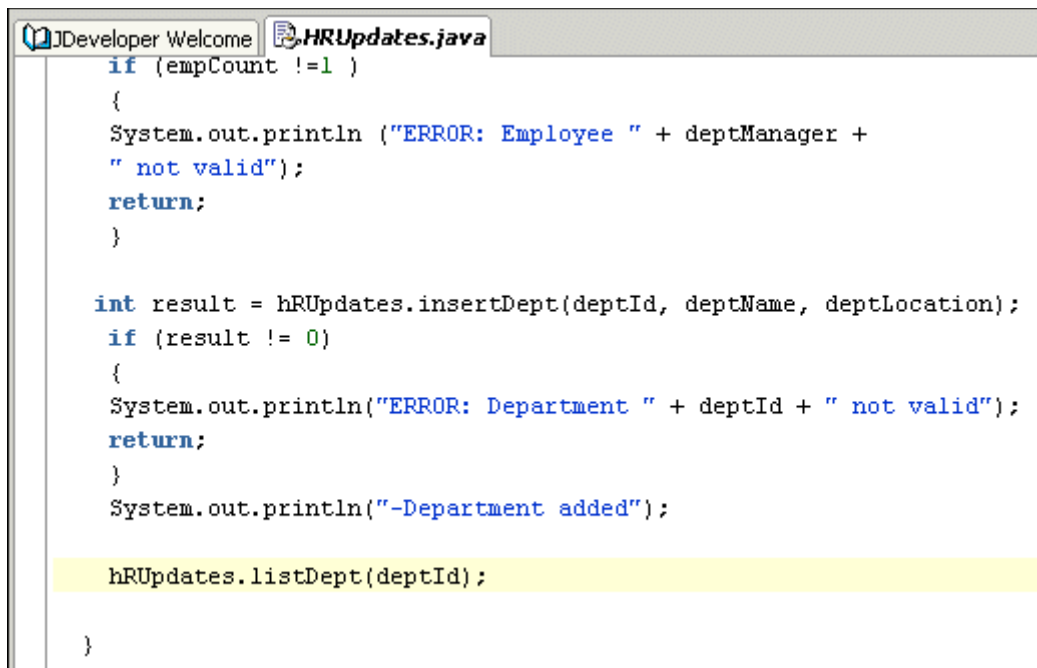


The screenshot shows a code editor window titled "Developer Welcome" and "HRUpdates.java". The code inside the editor is the same as the one in the previous block, but with some formatting changes: the `throws Exception` is in blue, and the `while` loop body is highlighted in yellow. The code is as follows:

```
public void listDept(String deptId) throws Exception  
{  
    stmt = conn.createStatement ();  
    ResultSet rset = stmt.executeQuery  
    ("SELECT * FROM DEPARTMENTS where DEPARTMENT_ID = "  
    + deptId + " ");  
    while (rset.next ())  
    {  
        System.out.println( rset.getInt(1) + " " + rset.getString(2)  
        + " " + rset.getInt(4));  
    }  
  
}
```

6. Añada al método principal una línea de código para invocar el método que acaba de crear recientemente. Para ello, use la línea:

```
hRUpdates.listDept(deptId);
```



```

JJDeveloper Welcome HRUpdates.java
    if (empCount !=1 )
    {
        System.out.println ("ERROR: Employee " + deptManager +
        " not valid");
        return;
    }

    int result = hRUpdates.insertDept(deptId, deptName, deptLocation);
    if (result != 0)
    {
        System.out.println("ERROR: Department " + deptId + " not valid");
        return;
    }
    System.out.println("-Department added");

    hRUpdates.listDept(deptId);

}

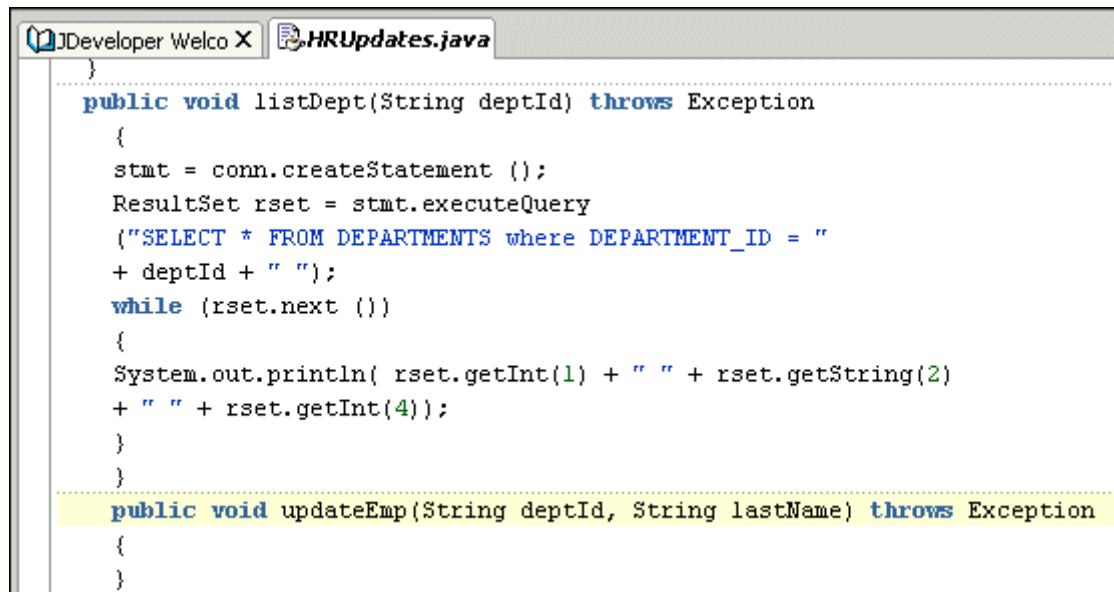
```

### Actualizar datos en una base de datos

Hasta este punto, el programa ya ha insertado en la base de datos el departamento que se deseaba incluir. Solo resta actualizar los datos del empleado que gestionará dicho departamento, de tal forma que se refleje en la base de datos su nueva posición y asignación al departamento. Para ello, siga los siguientes pasos:

1. Cree un método llamado **updateEmp** que contendrá el código para realizar la actualización. Este método recibirá el código del departamento y los apellidos del empleado que lo gestionará. Emplee una estructura similar a la siguiente:

```
public void updateEmp(String deptId, String lastName) throws Exception
{
}
```



```

}

public void listDept(String deptId) throws Exception
{
    stmt = conn.createStatement ();
    ResultSet rset = stmt.executeQuery
    ("SELECT * FROM DEPARTMENTS where DEPARTMENT_ID = "
    + deptId + " ");
    while (rset.next ())
    {
        System.out.println( rset.getInt(1) + " " + rset.getString(2)
        + " " + rset.getInt(4));
    }
}

public void updateEmp(String deptId, String lastName) throws Exception
{
}

```

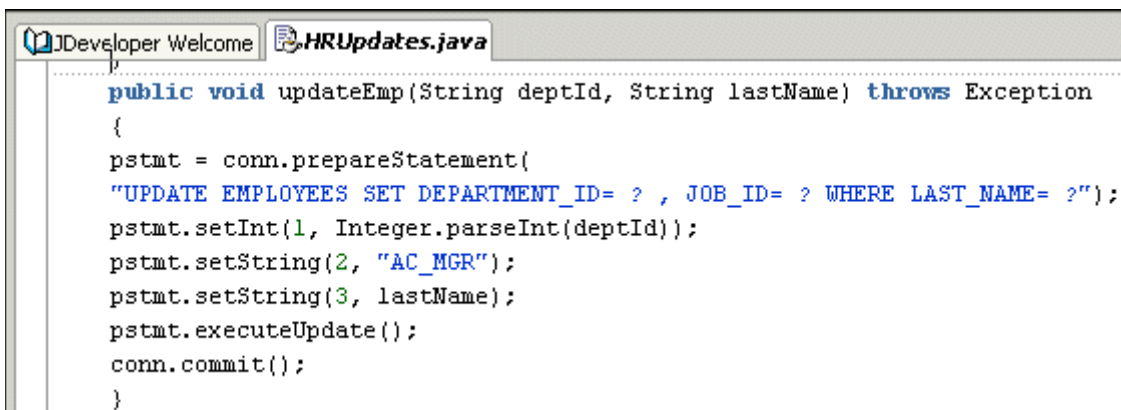
2. Añada al método anterior el código necesario para realizar convenientemente la actualización, Emplee el siguiente código:

```

pstmt = conn.prepareStatement("UPDATE EMPLOYEES SET DEPARTMENT_ID= ? ,
JOB_ID= ? WHERE LAST_NAME= ?");
pstmt.setInt(1, Integer.parseInt(deptId));
pstmt.setString(2, "AC_MGR");
pstmt.setString(3, lastName);
pstmt.executeUpdate();
conn.commit();

```

Si va a realizar varias ejecuciones de prueba es recomendable que comente la línea de código en la que se valida la actualización.



```

}

public void updateEmp(String deptId, String lastName) throws Exception
{
    pstmt = conn.prepareStatement(
    "UPDATE EMPLOYEES SET DEPARTMENT_ID= ? , JOB_ID= ? WHERE LAST_NAME= ?");
    pstmt.setInt(1, Integer.parseInt(deptId));
    pstmt.setString(2, "AC_MGR");
    pstmt.setString(3, lastName);
    pstmt.executeUpdate();
    conn.commit();
}

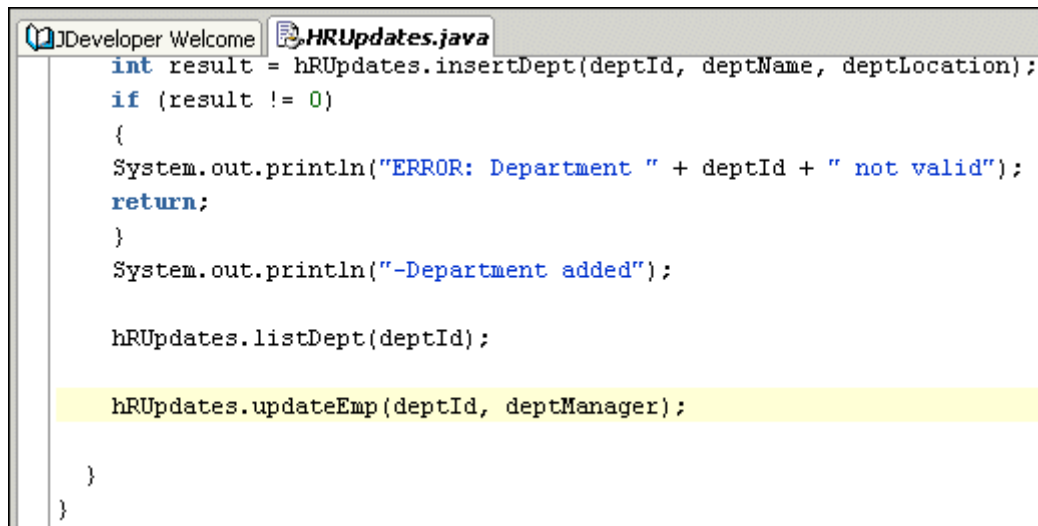
```

3. Añada el código necesario para invocar este nuevo método desde el método principal. Emplee la línea siguiente para invocarlo:

```

hrUpdates.updateEmp(deptId, deptManager);

```



```
int result = hRUpdates.insertDept(deptId, deptName, deptLocation);
if (result != 0)
{
    System.out.println("ERROR: Department " + deptId + " not valid");
    return;
}
System.out.println("-Department added");

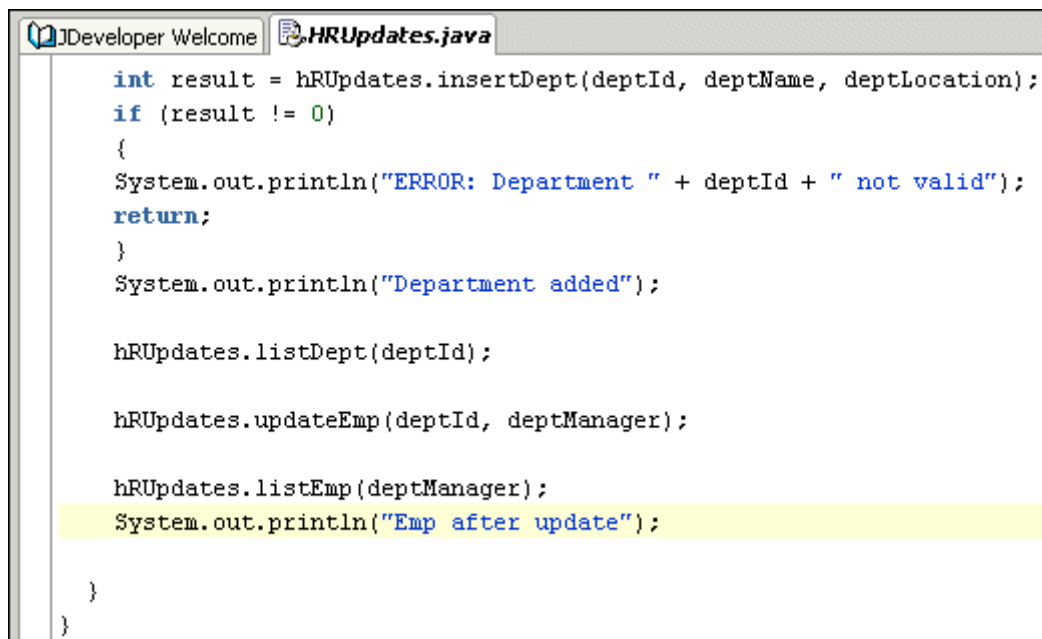
hRUpdates.listDept(deptId);

hRUpdates.updateEmp(deptId, deptManager);

}
```

4. Añada código para volver a realizar la consulta sobre la tabla de empleados para comprobar que la actualización se ha llevado a acabo. Para ello incluya el siguiente código en el método principal:

***hRUpdates.listEmp(deptManager);***  
***System.out.println("Emp after update");***



```
int result = hRUpdates.insertDept(deptId, deptName, deptLocation);
if (result != 0)
{
    System.out.println("ERROR: Department " + deptId + " not valid");
    return;
}
System.out.println("Department added");

hRUpdates.listDept(deptId);

hRUpdates.updateEmp(deptId, deptManager);

hRUpdates.listEmp(deptManager);
System.out.println("Emp after update");

}
```

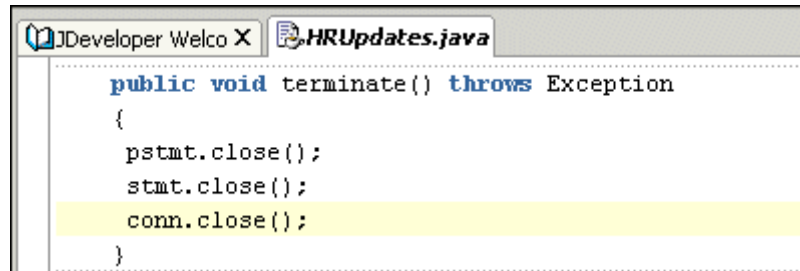
5. Para finalizar el programa deberemos de crear un método que se encargue de cerrar la conexión, así como los objetos que representan las sentencias que hemos ejecutado. Cree un método llamado ***terminate*** definido según el siguiente código:

***public void terminate() throws Exception{***  
 ***pstmt.close();***  
 ***stmt.close();***

```

    conn.close();
}

```

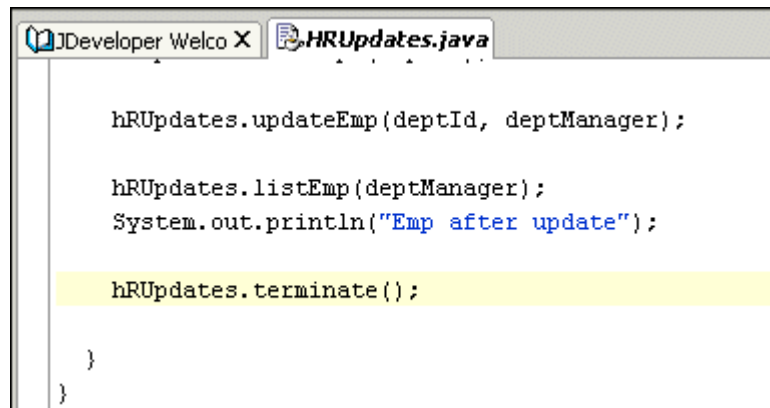


6. Finalmente, incluya en al final del código del método principal una llamada al método **terminate** para que se cierre ordenadamente la conexión a la base de datos. La línea de código necesaria es la siguiente:

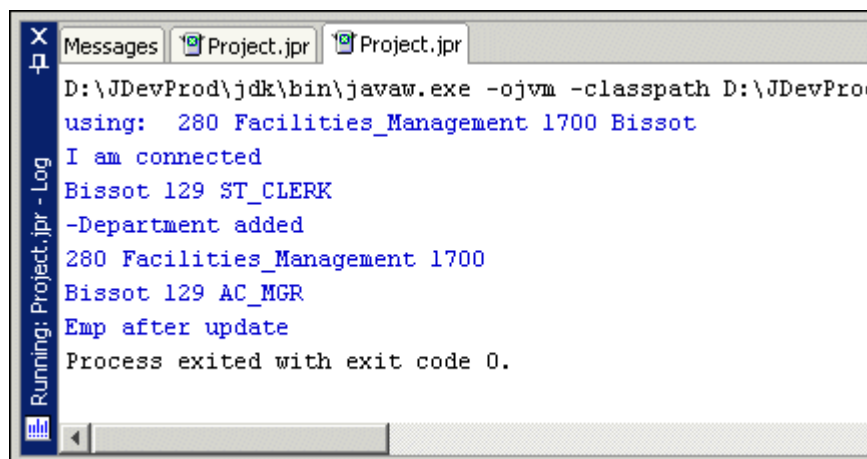
```

hRUpdates.terminate();

```

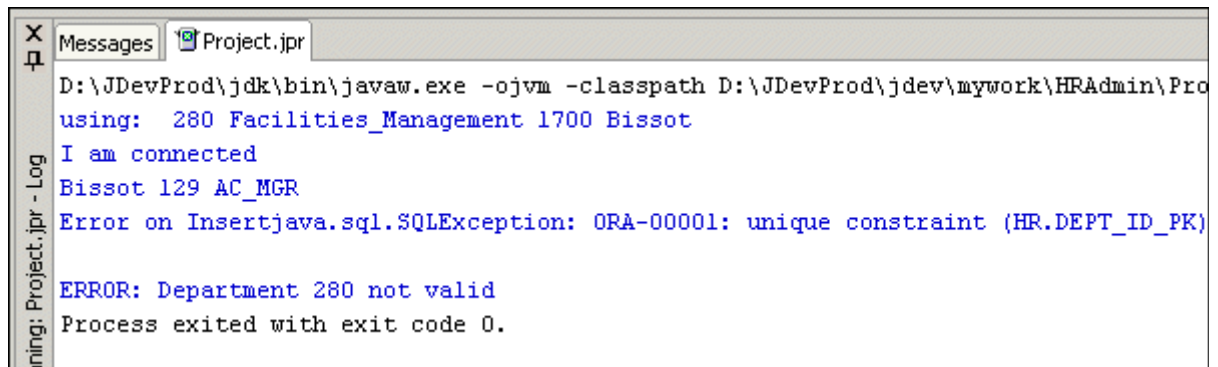


7. Ejecute el programa y observe sus resultados del mismo en la ventana de mensajes (**Log window**).



8. Realice una nueva ejecución del programa para confirmar el funcionamiento de código de tratamiento de excepciones incluido en el Método **insertDepartment**. Recuerde que este código controla que se haya producido una excepción debido a una violación de la clave primaria al intentar insertar dos veces el mismo departamento. Por tanto, si se vuelve a ejecutar el programa se producirá este error y se mostrará en la salida del mismo el mensaje "Error on

Insert". Si comentó el código de las confirmaciones (**commit**) de la inserción y de la actualización, descoméntelo para que los cambios realizados en la base de datos sean permanentes, y así se pueda producir el error que provoque la excepción.



The screenshot shows a Java IDE window titled 'Messages' with a sub-tab 'Project.jpr'. The output text is as follows:

```
D:\JDevProd\jdk\bin\javaw.exe -ojvm -classpath D:\JDevProd\jdev\mywork\HRAdmin\Pro
using: 280 Facilities_Management 1700 Bissot
I am connected
Bissot 129 AC_MGR
Error on Insertjava.sql.SQLException: ORA-00001: unique constraint (HR.DEPT_ID_PK)
ERROR: Department 280 not valid
Process exited with exit code 0.
```

## Resumen

En este tutorial se muestra como se pueden crear aplicaciones en JDeveloper que accedan a bases de datos empleando JDBC. Se han ejecutado consultas sobre tablas y se han insertado y actualizado datos.

## Ejercicios

1. Cree un método para listar todos los integrantes de un departamento. Puede encontrar detalles sobre la API JDBC en <http://docs.oracle.com/javase/1.5.0/docs/guide/jdbc/getstart/GettingStartedTOC.fm.html>
2. Cree un método para borrar un empleado, trate las posibles excepciones que pudieran darse.
3. Cree un método genérico que describa la estructura de una tabla. El nombre de la tabla se le pasa como argumento y debe mostrar por pantalla una descripción (nombre, tipo y longitud de columnas, número de tuplas que contiene, etc...). Puede obtener información sobre los métodos que necesita en la siguiente dirección: <http://docs.oracle.com/javase/1.5.0/docs/api/java/sql/ResultSetMetaData.html>