

# Calidad de los Sistemas Informáticos, 2018-2019

## Práctica 0 **Instalación y configuración**

- ▶ Aplicaciones requeridas
- ▶ Recursos requeridos y versiones
- ▶ Instalación
- ▶ Prueba de funcionamiento
- ▶ Buenas prácticas

# Aplicaciones requeridas

---

## Entorno Eclipse

- ▶ Lenguaje de programación Java
- ▶ Pruebas unitarias
- ▶ Cobertura de código
- ▶ IDE para interfaces de usuario



## Paquete de servicios AppServer (o alternativa)

- ▶ Sistema de gestión de bases de datos MySQL
- ▶ Aplicación web phpMyAdmin



# Recursos requeridos y versiones

---



- ▶ Eclipse IDE for Java Developers (4.8, Photon)



- ▶ AppServer 8.6.0 (o similar) para servicio MySQL y phpMyAdmin



- ▶ Conector de MySQL 5.1.34 para Java



- ▶ WindowBuilder 1.9.1 para IDE de interfaces en Swing



- ▶ JUnit 5.1 para pruebas unitarias (incluido en Eclipse 4.8)



- ▶ EclEmma 3.1.0 para cobertura de código (incluido en Eclipse 4.8)

## Eclipse IDE for Java Developers (4.8, Photon)

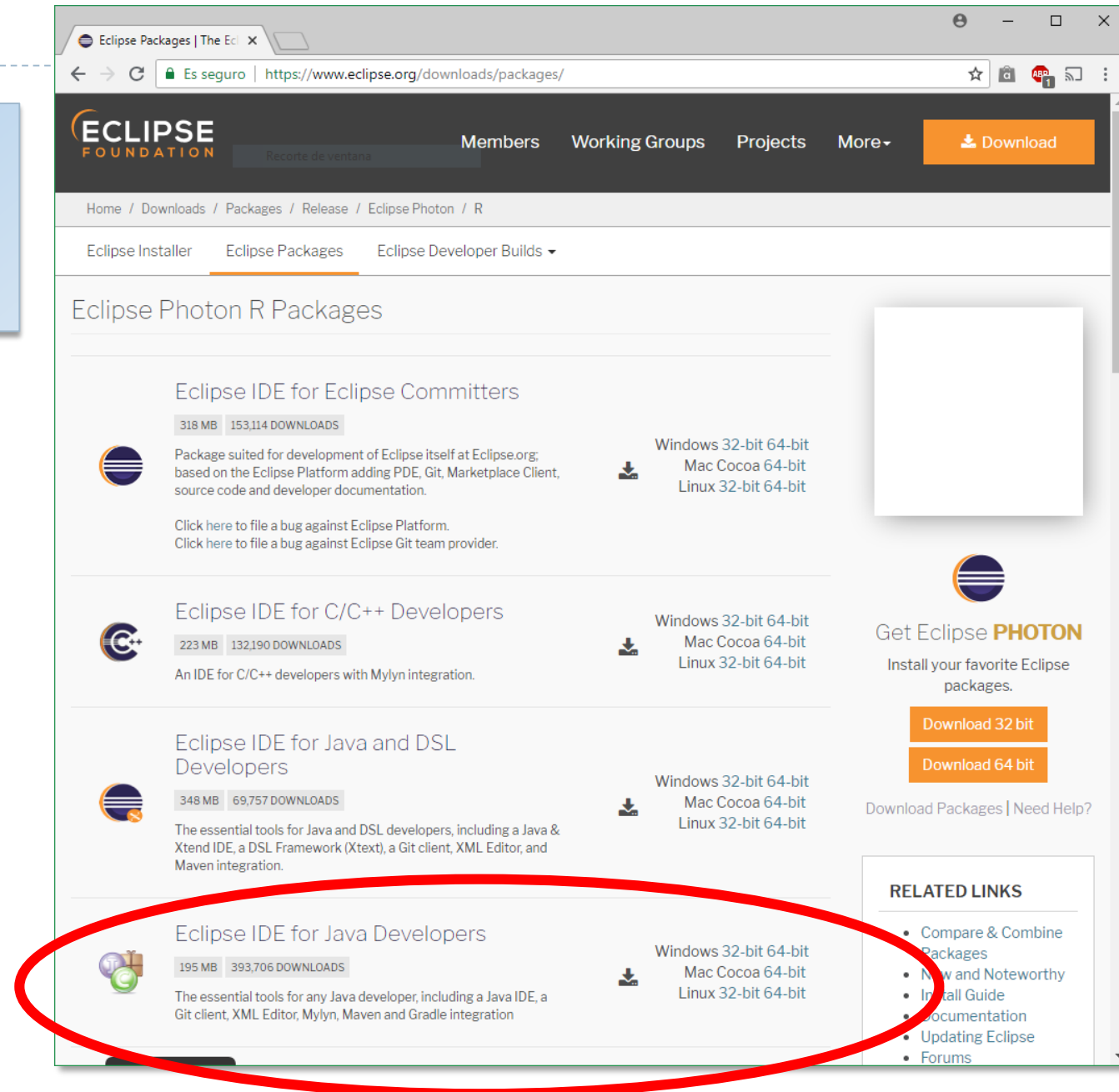
<http://www.eclipse.org/downloads/eclipse-packages/>

### Instalación en el equipo

1. Descargar la misma versión (32 ó 64) que la versión Java del equipo
2. Descomprimir y copiar en una carpeta
3. Ejecutar Eclipse.exe (o similar en otros sistemas)

### Configuración de MarketPlace de Eclipse

1. Ir a *Help > Install New Software*
2. En *Work with*,  
<http://download.eclipse.org/mpc/photon>
3. Marcar *EPP Marketplace Client*
4. Continuar y reiniciar si se requiere



# Instalación (2 de 4)

## AppServ 8.6.0 (sólo Windows)

<https://www.appserv.org/download/>

### Instalación en el equipo

- ▶ Servidor: myserver.com (o cualquier otro)
- ▶ Puerto Apache: 89 (o cualquier otro)
- ▶ Puerto MySQL: 3306 (o cualquier otro)

### Prueba de acceso (hacer en un navegador)

- ▶ `http://localhost{:puerto}/`
- ▶ `http://localhost{:puerto}/phpmyadmin`

usuario: *root*, contraseña: *{:la de instalación}*



Otras alternativas como WampServer son válidas. Consultar en caso de no encontrarlas para Mac y/o Linux.

## Conector MySQL para Java

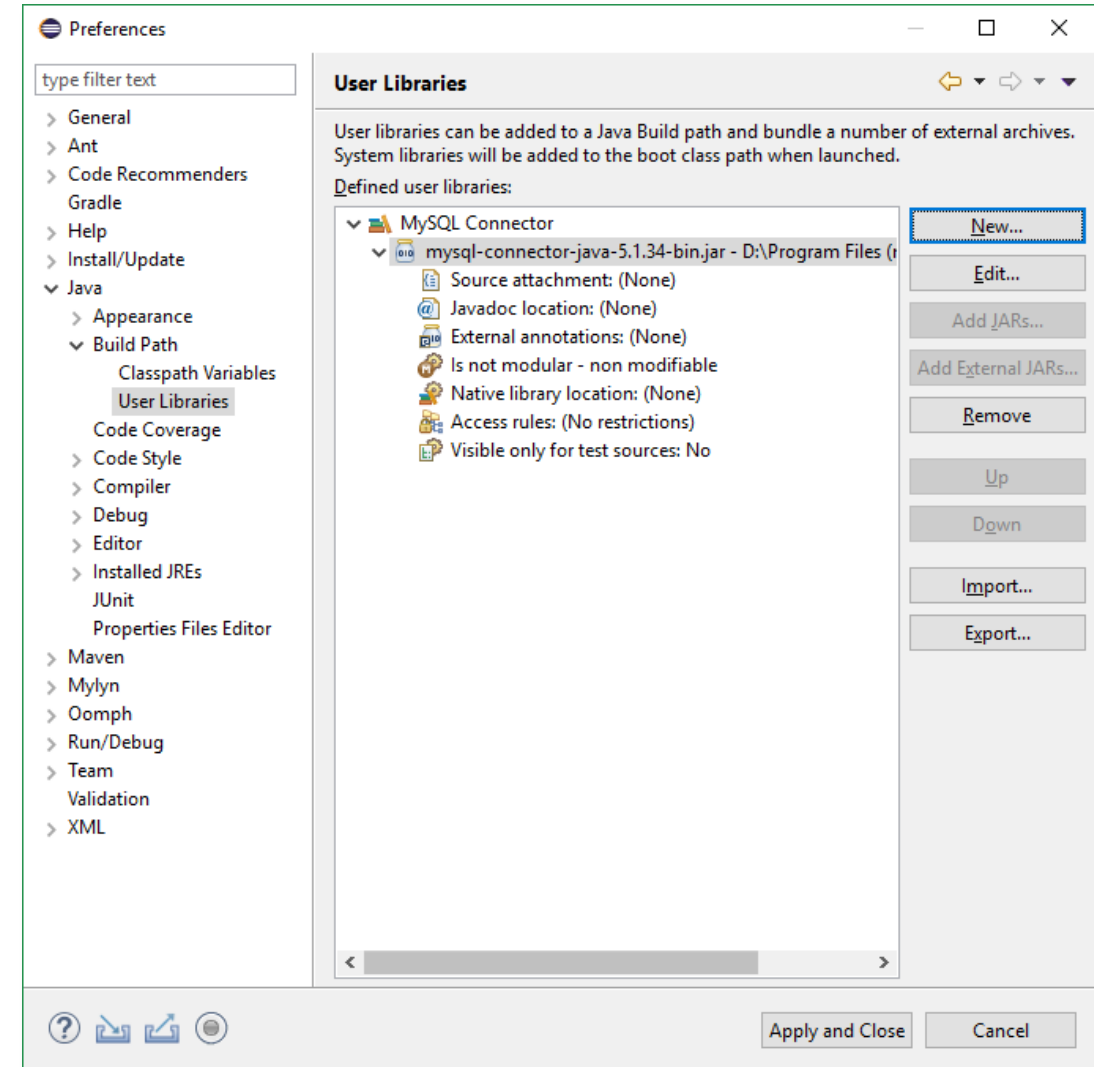
<http://mirror.cogentco.com/pub/mysql/Connector-J/mysql-connector-java-5.1.34.zip>

### Extracción en el equipo

1. Creación –donde se desee– de carpeta `javaliib/csi` para las bibliotecas externas
2. Descompresión del ZIP del enlace
3. Copia del `.JAR` existente en el recién creado `javaliib/csi`

### Configuración en Eclipse

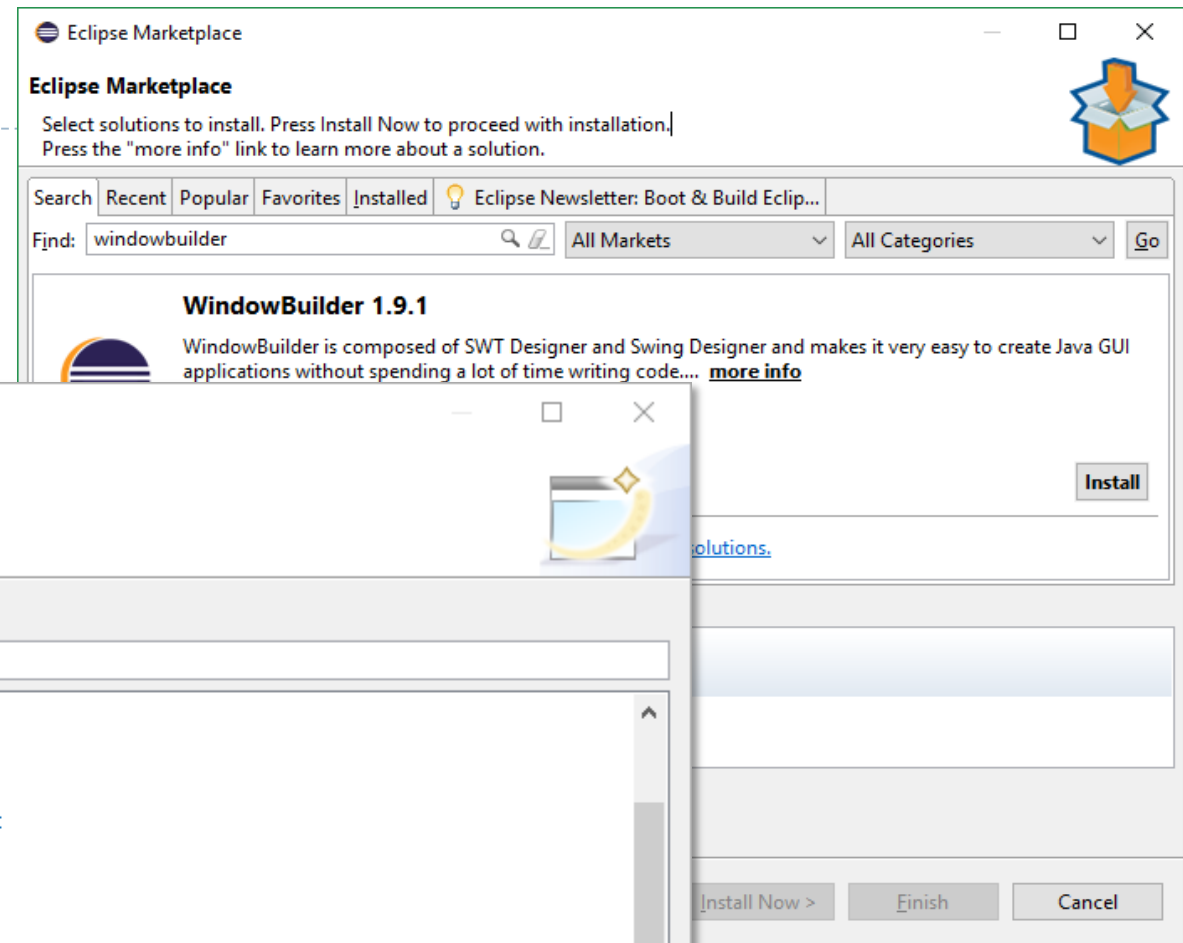
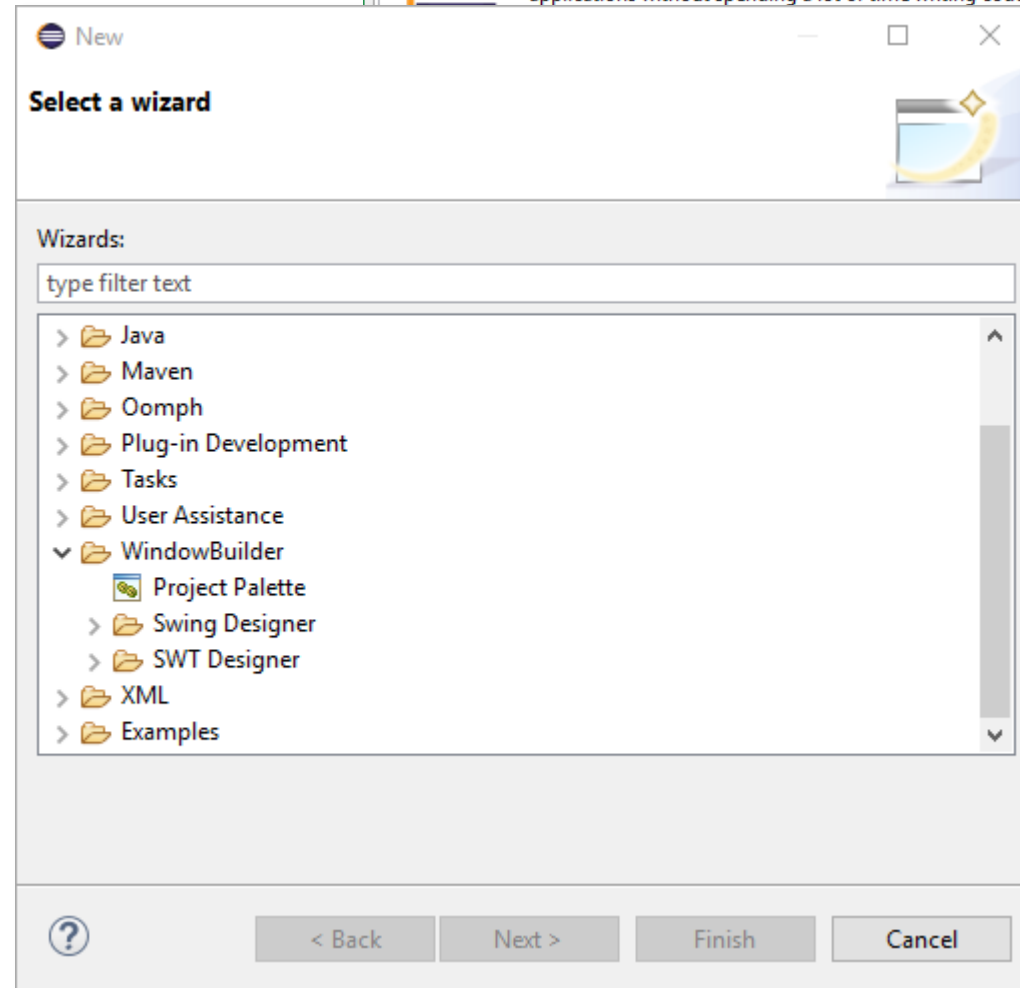
1. Ir a *Window > Preferences*
2. Ir a *Java > Build Path > User Libraries*
3. *New...*
4. Escribir *MySql Connector*
5. *Add External JARs...*
6. Seleccionar el `.JAR` de `javaliib/csi`
7. *Apply and Close*



# Instalación (4 de 4)

## WindowBuilder 1.9.1

1. Ir a *Help > Eclipse Marketplace*
2. Buscar *WindowBuilder*
3. Instalar y reiniciar
4. Comprobar que en  
*File > New... > Other...*  
aparece la carpeta  
*WindowBuilder*



# Prueba de funcionamiento (1 de 9)

---

1. Abrir Eclipse e indicar carpeta para proyectos
2. Crear un nuevo proyecto de Java (*File > New > Java Project*)
  - ▶ Project name: `es.uca.gii.csi18.check`
3. Incluir biblioteca de acceso a MySQL creada antes, marcando (*Next > Libraries > Add Library... > User Library > MySQL Connector*)
4. Crear un archivo de acceso a la base de datos
  - ▶ Botón derecho sobre el nombre del proyecto, *New > File*
  - ▶ File name: `db.properties`
  - ▶ Incluir contenido de la diapositiva siguiente



## Prueba de funcionamiento (2 de 9)

### db.properties

```
jdbc.driverClassName=com.mysql.jdbc.Driver  
jdbc.url=jdbc:mysql://localhost/mysql  
jdbc.username={ :usuario MySQL}  
jdbc.password={ :contraseña}
```

Nombre la base de datos –para la prueba se usará *mysql*, creada por defecto–.

Si el puerto no es 3306, debe indicarse tras *localhost*.

Debe modificarse el usuario y la contraseña por las correctas, sin incluirse las llaves ni comillas.

### 5. Crear paquete de utilidades

- ▶ Botón derecho sobre `src`, *New > Package*
- ▶ Name: `es.uca.gii.csi18.check.util`

### 6. Crear clase de acceso a la configuración

- ▶ Botón derecho sobre `es.uca.gii.csi18.check.util`, *New > Class*
- ▶ Name: `Config`
- ▶ Incluir código fuente de la imagen de la diapositiva siguiente

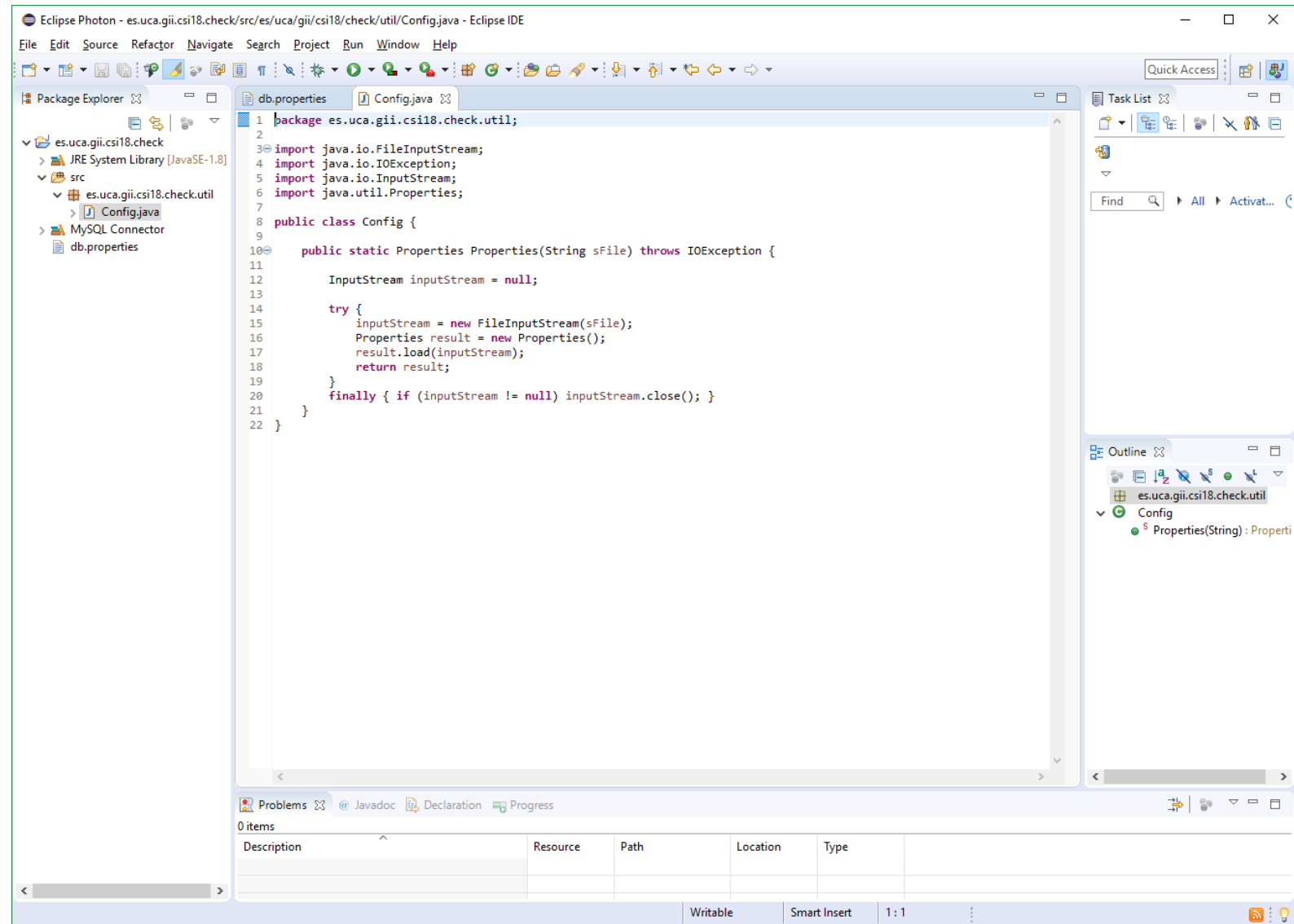
# Prueba de funcionamiento (4 de 9)

## util/Config.java

La clase `Config` proporciona un método estático `Properties` que, dado un archivo de propiedades – como `db.properties` –, devuelve una instancia de la clase `Properties` de Java.

A su vez, la clase `Properties` de Java. proporciona el método `getProperty` para acceder al valor de la propiedad correspondiente. Ejemplo de uso:

```
Properties properties =  
    Config.Properties("db.properties");  
String sUrl =  
    properties.getProperty("jdbc.url");
```



### 7. Crear paquete de acceso a datos

- ▶ Botón derecho sobre `src`, *New > Package*
- ▶ Name: `es.uca.gii.csi18.check.data`

### 8. Crear clase para acceso a datos

- ▶ Botón derecho sobre `es.uca.gii.csi18.check.data`, *New > Class*
- ▶ Name: `Data`
- ▶ Incluir código fuente de la imagen de la diapositiva siguiente

# Prueba de funcionamiento (6 de 9)

## data/Data.java

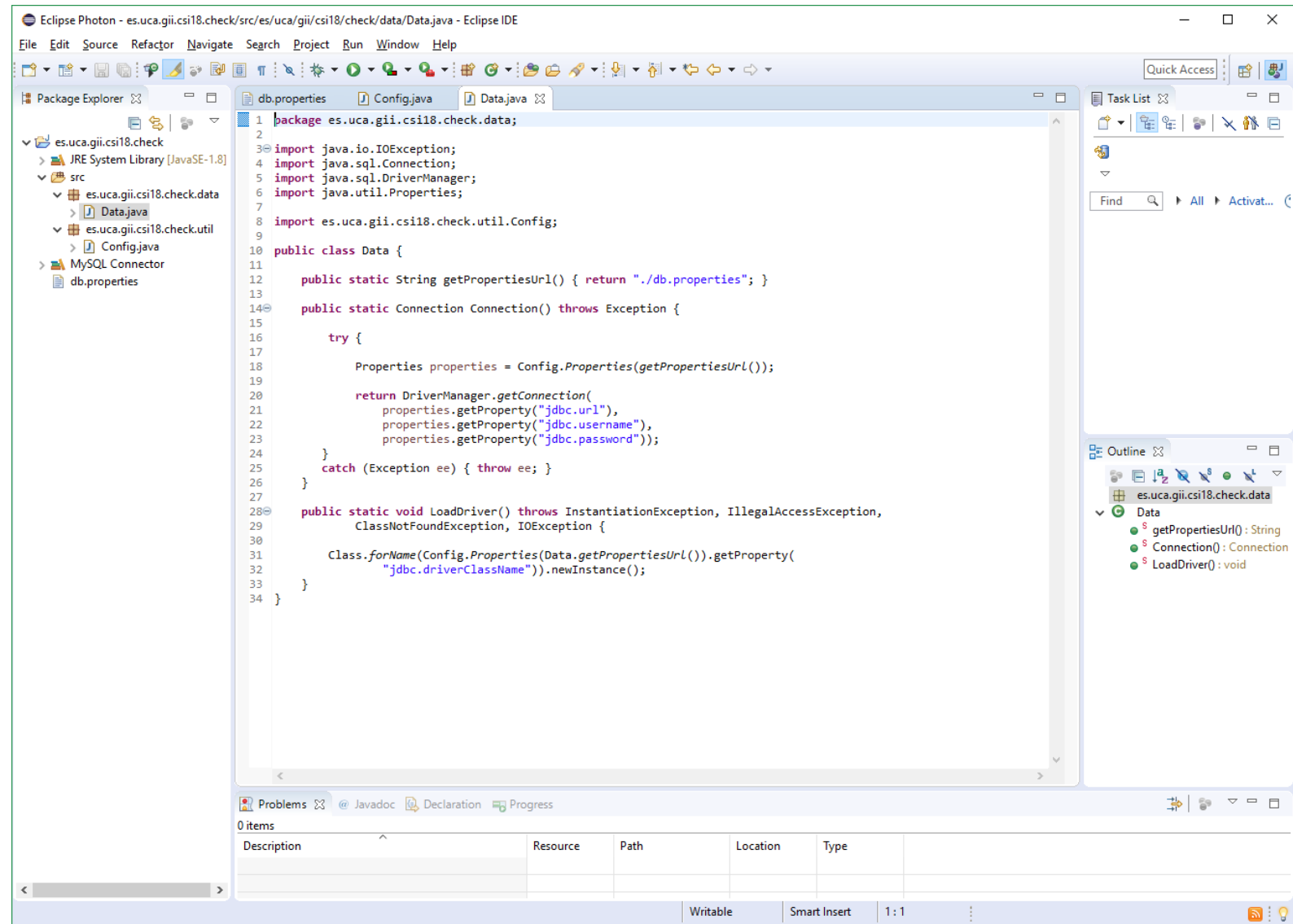
Data aporta los métodos estáticos:

a) `LoadDrive`, que debe invocarse una sola vez y que carga el drive de conexión correspondiente, a partir del archivo de propiedades.

b) `Connection()`, que devuelve una conexión a la base de datos – instancia de la clase `Connection` de Java–, a partir del archivo de propiedades.

c) `getPropertiesUrl()`, que devuelve la URL del archivo de propiedades.

```
Data.LoadDriver();  
Connection con = Data.Connection();
```



### 9. Crear paquete de pruebas unitarias

- ▶ Botón derecho sobre `src`, *New > Package*
- ▶ Name: `es.uca.gii.csi18.check.test`

### 10. Crear clase para pruebas de la clase *Data*

- ▶ Botón derecho sobre `es.uca.gii.csi18.check.test`, *New > JUnit Test Case*
- ▶ Name: `DataTest`
- ▶ Seleccionar la inclusión del método `setUpBeforeClass`
- ▶ Indicar cuando se pida la inclusión de JUnit 5 en las bibliotecas
- ▶ Incluir código fuente de la imagen de la diapositiva siguiente

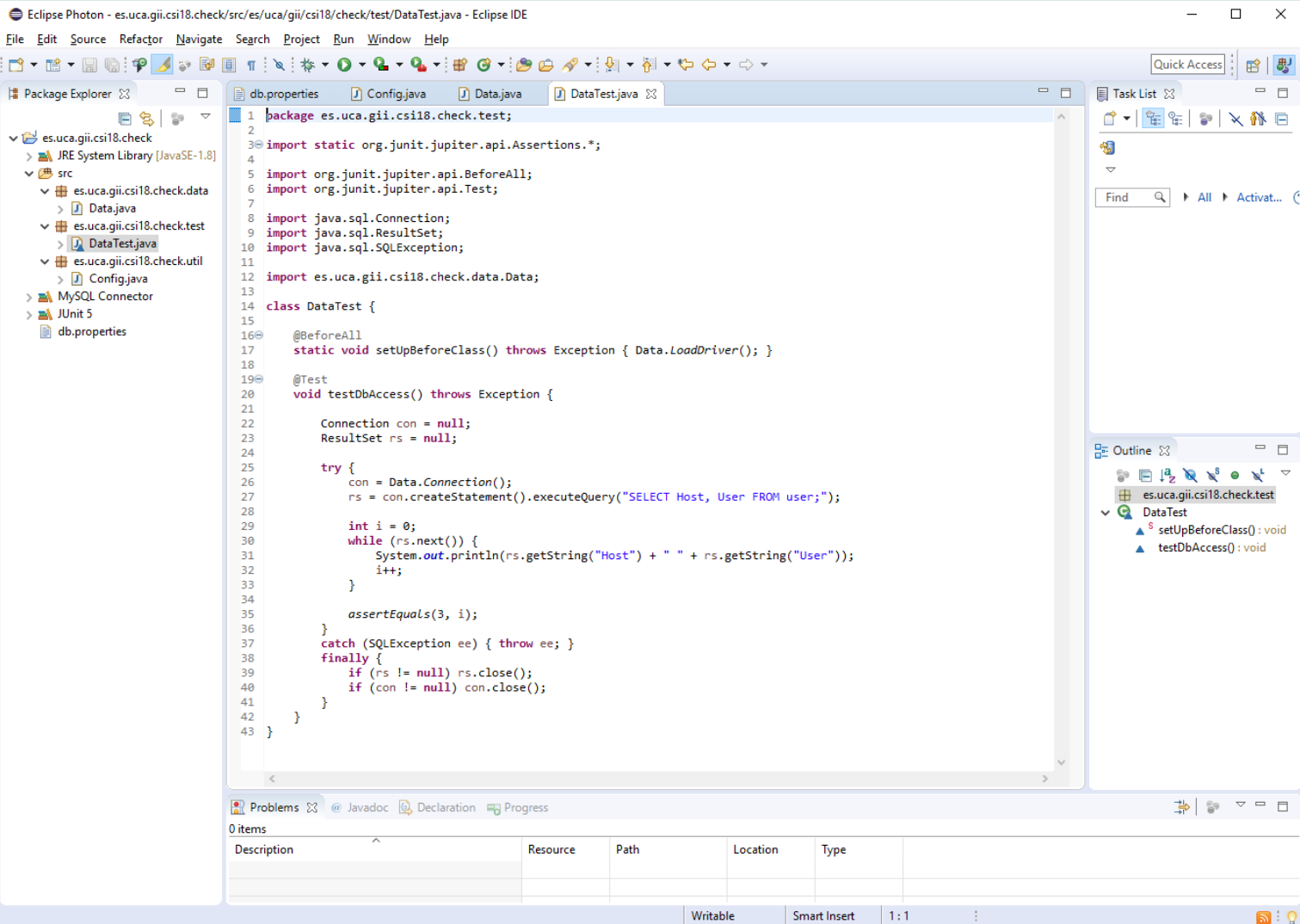
# Prueba de funcionamiento (8 de 9)

## test/DataTest.java

DataTest prueba el funcionamiento de la clase Data:

a) `setUpBeforeClass` se ejecuta una vez al principio y carga el driver MySQL.

b) `testDbAccess()`, que realiza una consulta a MySQL, imprime los registros —ilustrativamente, no es necesario en una prueba unitaria— y comprueba vía `assertEquals` que el número de elementos devuelve es el que se espera —esto sí se requiere—.



```
1 package es.uca.gii.csi18.check.test;
2
3 import static org.junit.jupiter.api.Assertions.*;
4
5 import org.junit.jupiter.api.BeforeAll;
6 import org.junit.jupiter.api.Test;
7
8 import java.sql.Connection;
9 import java.sql.ResultSet;
10 import java.sql.SQLException;
11
12 import es.uca.gii.csi18.check.data.Data;
13
14 class DataTest {
15
16     @BeforeAll
17     static void setUpBeforeClass() throws Exception { Data.LoadDriver(); }
18
19     @Test
20     void testDbAccess() throws Exception {
21
22         Connection con = null;
23         ResultSet rs = null;
24
25         try {
26             con = Data.Connection();
27             rs = con.createStatement().executeQuery("SELECT Host, User FROM user;");
28
29             int i = 0;
30             while (rs.next()) {
31                 System.out.println(rs.getString("Host") + " " + rs.getString("User"));
32                 i++;
33             }
34
35             assertEquals(3, i);
36         }
37         catch (SQLException ee) { throw ee; }
38         finally {
39             if (rs != null) rs.close();
40             if (con != null) con.close();
41         }
42     }
43 }
```

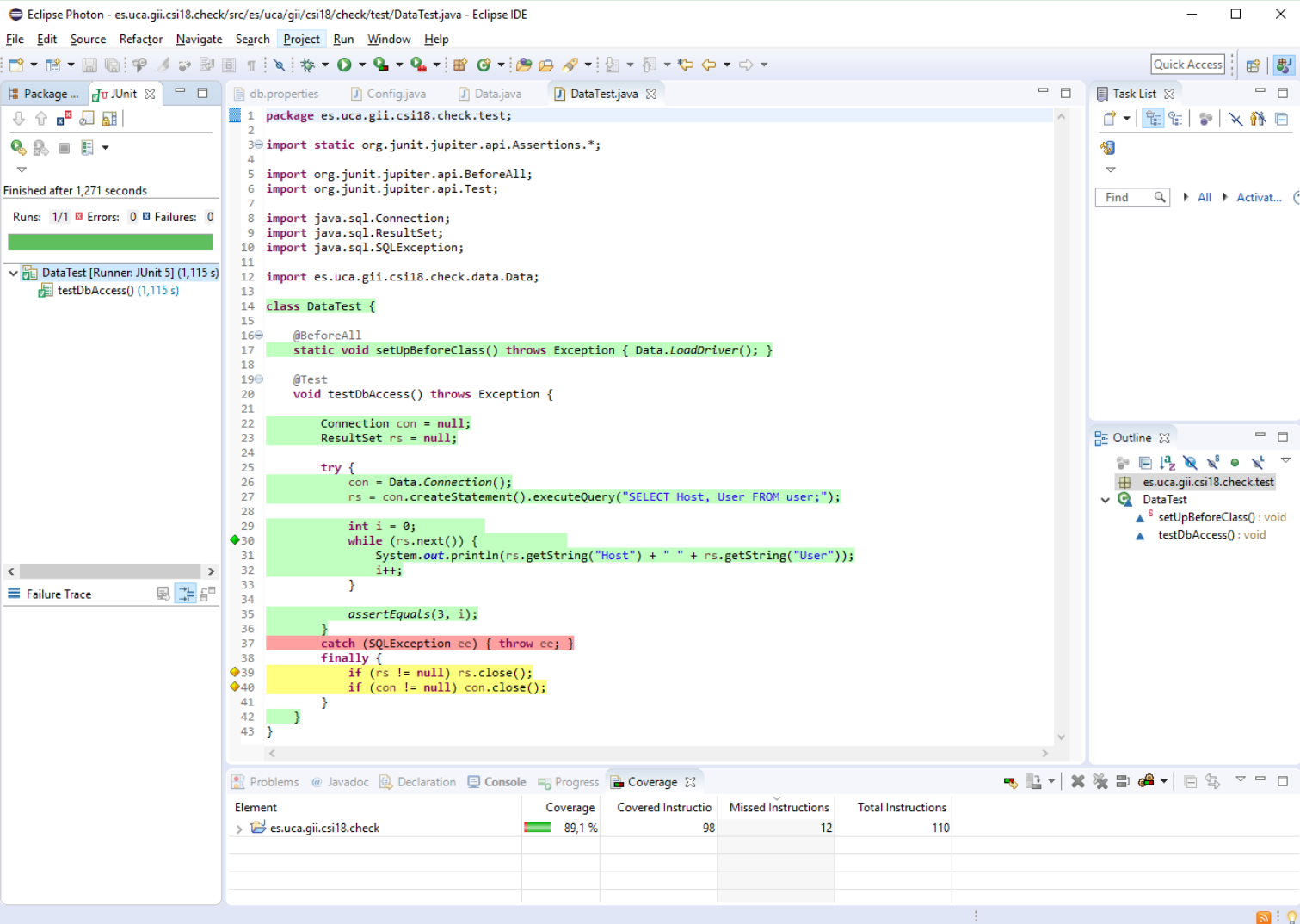
## Ejecución del test

**Ejecutar como JUnit Test con cobertura de código** (botón derecho sobre el nombre del proyecto, Coverage As > JUnit Test). Una vez ejecutado:

a) A la izquierda, **JUnit mostrará el resultado de la prueba**, verde si ha pasado el `assertEquals` y rojo si no.

b) En la pestaña inferior Coverage aparecerá el **código total que se ha ejecutado**. El árbol permite expansión y muestra cuánto se ha ejecutado de cada archivo. Pinchando dos veces en un archivo se accede a éste.

c) En la pestaña inferior Console se mostrarán los **resultados impresos con `System.out.println`** dentro del bucle, que son todos los registros de la tabla `User`.



The screenshot shows the Eclipse Photon IDE with the following components:

- Main Editor:** Displays the source code of `DataTest.java`. The code includes imports for JUnit and JDBC, and a test method `testDbAccess()` that connects to a database, executes a query, and prints the results.
- JUnit View (Left):** Shows the test results for `DataTest`. It indicates that the test passed (green bar) and provides a summary of runs, errors, and failures.
- Coverage View (Bottom):** Shows the coverage statistics for the project. The table below summarizes the data:
- Console View (Bottom Right):** Displays the output of the test, showing the contents of the `User` table.

Element	Coverage	Covered Instructio	Missed Instructions	Total Instructions
es.uca.gii.csi18.check	89,1 %	98	12	110

En caso de no pasar el test de JUnit, debe actuarse poniendo en el primer parámetro de `assertEquals` lo que realmente se espera.



## Buenas prácticas de aplicación para todas las prácticas de la asignatura

---

- ▶ Estudiar el código del método `testDbAccess()` de *DataTest.java*. Se usa el **mismo esquema en todas las consultas de selección**.
- ▶ Todas las conexiones `Connection` y cursores `ResultSet` creados en un método `no` recibidos como parámetros **deben ser cerrados al final** del método, haya habido error en éste o no. El modo de asegurarse es hacerlo en el `finally` de un `try`. Primero se cierran los cursores y, luego, la conexión.
- ▶ El objetivo de los métodos de prueba no es imprimir. Puede hacerse para comprobar visualmente algunos datos en consola, pero no es necesario. El objetivo de los métodos de prueba es tener **descritos adecuadamente sus correspondientes asserts** para que JUnit pueda validarlos, comparando lo que se espera `primer parámetro` con lo que recibe `segundo parámetro`.
- ▶ La cobertura indica qué porcentaje de código se ejecuta: **un porcentaje de código bajo (< 75%) no justificado puede significar código innecesario** en la aplicación.