UNIVERSIDAD DEL QUINDÍO
FACULTAD DE INGENIERÍA
PROGRAMA DE INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

Información general	
Actualizada por:	Carlos Andrés Flórez Villarraga
Duración estimada en minutos:	60
Docente:	Christian Andrés Candela y Einer Zapata G.
Guía no.	7
Nombre de la guía:	Pruebas

Información de la Guía

OBJETIVOS

Aprender a realizar pruebas haciendo uso del framework JUnit y Arquilliam.

CONCEPTOS BÁSICOS

Manejo de Eclipse, Java, Bases de Datos, JDBC, DataSource, Entidades y Glassfish.

PRUEBAS

Dentro del desarrollo de software las pruebas de programador cumplen con una importante función a la hora de comprobar el correcto funcionamiento de la aplicación, así como de encontrar y eliminar los defectos que se puedan haber generado durante la etapa de codificación. Entre más grande sea la aplicación, más modificaciones deben hacerse a funcionalidades ya codificadas y probadas, así como el trabajo en equipo sobre una misma funcionalidad hace necesario formas más rápidas y eficientes que permitan probar el correcto funcionamiento de la aplicación.

Java provee mecanismos que permiten realizar verificaciones dentro del código conocidos como assertions.

Assert: Es una sentencia Java usada para expresar y verificar una afirmación. El assert se puede usar con uno o dos parámetros.

***assert expresion_booleana*;**

```
assert 7 < 10;
```

```
int a = 7;
assert a < 10;
```

La expresión booleana representa la afirmación que se desea verificar, si el resultado de la expresión es verdadero, la ejecución del programa continuará de forma normal, en caso de que el resultado de la expresión sea falso, la ejecución se interrumpirá para generar un AssertionError.

assert expresion_booleana : expresion_resultante;

```
assert 7 < 10 : “Afirmación invalida”;
```

```
int a = 7;
String mensaje = “Afirmación invalida”;
….
assert a < 10 : mensaje;
```

Esta forma funciona igual a la anterior, se diferencia por el uso de una expresión resultante, la cual debe ser un valor diferente de **void**. Bien puede ser una variable, expresión booleana, matemática o incluso el llamado a un método que retorne un valor. En caso de que la expresión booleana sea falsa, el valor resultante de evaluar la segunda expresión es convertido a String, y pasado por parámetro al AssertionError generado por el assert.

JUNIT

JUnit es un framework basado en los assert creado por Erich Gamma y Kent Beck. JUnit es usado para la automatización de las pruebas unitarias. Permite la ejecución y comprobación de las funcionalidades de las clases de un proyecto de manera controlada. Todo ello con el fin de evaluar si los métodos de la clase funcionan correctamente. Para evaluar el funcionamiento del código se brindan datos de entrada a cada uno de los métodos para los cuales se conoce la respuesta correcta, se compara el resultado de la ejecución con el resultado esperado, en caso de que los resultados coincidan la prueba es marcada como exitosa, en caso contrario se dice que la prueba fallo.

A partir de la versión 4 de JUnit se incluyen un conjunto de anotaciones basadas en Java 5 que facilitan mucho el desarrollo de las pruebas.

- **@Test**: Permite marcar los métodos que serán usados como pruebas. Por medio de parámetros es posible indicar a la prueba una duración máxima, en cuyo caso si la prueba dura más del tiempo establecido ésta fallará. También es posible determinar un resultado específico por ejemplo una excepción, en cuyo caso si la prueba no arroja dicha excepción fallará.

```
@Test public void method() {
        ...
}

@Test(timeout=100) public void method2() {
        ...
}

@Test(expected = Exception.class) public void method3() {
        ...
}
```

- **@Before**: Permite marcar los métodos que deberán ser ejecutados antes de cada uno de los métodos de prueba `@Test`. Generalmente los métodos `@Before` son usados para la inicialización de datos a ser usados en la prueba.

```
@Before public void method() {
        ...
}
```

- **@After**: Permite marcar los métodos que deberán ser ejecutados después de cada uno de los métodos de prueba `@Test`. Generalmente los métodos `@After` son usados para liberar los recursos usados en la prueba.

```
@After public void method() {
	...
}
```

- **@BeforeClass**: Permite marcar los métodos estáticos que deberán ser ejecutados antes de iniciar las pruebas de una clase. El método `@BeforeClass` es ejecutado una única vez antes de todas las pruebas de una clase. Generalmente los métodos `@BeforeClass` son usados para la inicialización de datos comunes a ser usados por las pruebas.

```
@BeforeClass public void method() {
	...
}
```

- **@AfterClass**: Permite marcar los métodos estáticos que deberán ser ejecutados después de ejecutar las pruebas de una clase. El método `@AfterClass` es ejecutado una única vez después de todas las pruebas de una clase. Generalmente los métodos `@AfterClass` son usados para liberar los recursos comunes usados por las pruebas de una clase.

```
@AfterClass public static void method() {
	...
}
```

- **@Ignore**: Permite marcar los métodos de prueba `@Test` que por alguna razón no se desean ejecutar.

```
@Ignore @Test public static void method() {
	...
}
```

SENTENCIAS ASSERT EN JUNIT

JUnit proporciona un conjunto de sentencias `assert` encaminadas a la declaración de diferentes tipos de afirmaciones. Estas sentencias están disponibles como métodos de la clase Assert.

- **Fail**: Provoca que la prueba falle. Recibe por parámetro el mensaje de error que se debe generar.

  ```
  Assert.fail("Error al ejecutar la prueba");
  ```

- **assertTrue**: Verifica si la expresión booleana proporcionada es verdadera, en caso de ser falsa la prueba falla. Además de la expresión booleana también es posible proporcionar un mensaje de error.

  ```
  Assert.assertTrue(expresionBooleana);

  Assert.assertTrue("Prueba fallida", expresionBooleana);
  ```

- **assertEquals**: Verifica que dos objetos dados sean iguales, en caso de no serlos la prueba falla. Además de los objetos también es posible proporcionar un mensaje de error.

```
Assert.assertEquals(resultadoEsperado, resultadoReal);

Assert.assertEquals("No coinciden los resultados", resultadoEsperado, resultadoReal);
```

- **assertNull**: Verifica que el objeto dado sea null, en caso de no serlo la prueba falla. Además del objeto también es posible proporcionar un mensaje de error.

```
Assert.assertNull(objeto);

Assert.assertEquals("El objeto no es nulo", objeto);
```

Como los métodos anteriores, la clase `Assert` posee otros tantos métodos assert encaminados a verificar la validez de las afirmaciones propuestas. Puede encontrar un listado de estos métodos en la Api de JUnit en: https://junit.org/junit4/javadoc/4.8/org/junit/Assert.html

ARQUILLIAN

Arquillian es un framework open source que facilita la elaboración de pruebas. Permite a los desarrolladores verificar el comportamiento del código brindando un ambiente de ejecución, dando la oportunidad al desarrollador de centrarse en las pruebas de comportamiento de la aplicación.

Un caso de prueba de arquilliam debe tener 3 elementos:

- La clase de prueba debe tener la anotación `@RunWith(Arquillian.class)`
- Un método estático anotado con `@Deployment` que retorne un elemento de tipo `Archive` (`ShirnkWrap`).
- La clase debe tener al menos un método marcado con la anotación `@Test`

Adicionalmente, los métodos que requieran el uso de transacciones deberán ser anotados con `@Transactional(value=TransactionMode.ROLLBACK)`, el atributo value puede tomar uno de los siguientes valores:

- `COMMIT`: Cada prueba finalizará con la operación de commit. Este es el comportamiento predeterminado.
- `DISABLED`: Si ha habilitado la compatibilidad transaccional en el nivel de clase de prueba, marcar la prueba dada con este modo simplemente lo ejecutará sin la transacción.
- `ROLLBACK`: al final de la ejecución de la prueba se llevará a cabo la reversión

Puede buscar más información en la página http://arquillian.org.

PRECAUCIONES Y RECOMENDACIONES

En caso de hacer uso de clases por fuera del paquete donde se encuentran las entidades debe recordar incluir dichas clases en la generación de la prueba, específicamente en el método deploy.

ARTEFACTOS

Se requiere tener instalado el JDK y un IDE para el desarrollo de aplicaciones Eclipse JEE en su última versión, un servidor de aplicaciones que cumpla con las especificaciones de JEE, para esta práctica Glassfish y el motor

de base de datos MySQL.

EVALUACIÓN O RESULTADO

Se espera que el alumno pueda hacer uso de Junit y Arquillian para la realización de las pruebas unitarias.

Procedimiento

1. Para el desarrollo de esta guía necesitara una base de datos en MySQL, un proyecto de tipo Maven con soporte para el uso de JPA. Y una conexión a dicha base de datos para ser usada en la generación de las tablas y un conjunto de entidades ya desarrolladas. Se sugiere hacer uso del proyecto creado anteriormente.

2. Cree un nuevo Junit Test Case. Para ello acceda al menú new – other – Junit . Asignele al Junit el nombre de su preferencia.

3. Adicione a su clase Test la anotación `@RunWith(Arquillian.class)`

```
@RunWith(Arquillian.class)
public class EntidadTest {
	…
}
```

4. Adicione a su clase Test una instancia de la clase `EntityManager` con la anotación `@PersistenceContext`.

```
@RunWith(Arquillian.class)
public class EntidadTest {

	@PersistenceContext
	private EntityManager entityManager;

}
```

5. Cree un método estático marcado con la anotación `@Deployment` que retorne un elemento de tipo `Archive<?>`

```
@Deployment
public static Archive<?> createTestArchive() {
    return ShrinkWrap
        .create(WebArchive.class, "test.war")
        .addPackage(ENTIDAD.class.getPackage())
        .addAsResource("persistenceForTest.xml", "META-INF/persistence.xml")
        .addAsWebInfResource(EmptyAsset.INSTANCE, "beans.xml");
}
```

   Como podrá ver el método addPackage le permite determinar que clases usará para la ejecución de las pruebas. En este punto remplace `ENTIDAD.class` por las clases que correspondan a su proyecto.

6. Las pruebas sobre las entidades pueden facilitarse si se tiene como punto de partida un conjunto de datos

conocido. Para construir dicho conjunto de datos se usará archivos JSON. Para construir los archivos JSON será necesario identificar la estructura de cada una de las tablas que componen su proyecto en la base de datos usando la instrucción: `describe nombreTabla;`

Si por ejemplo al solicitar la estructura de la tabla `Empleado` obtiene algo como:

```
cedula varchar(15)
nombre varchar(50)
email  varchar(50)
clave  varchar(50)
```

Puede crear un conjunto de datos de prueba creando un archivo `empleado.json` en la carpeta `src/main/resources` así:

```
{
    "EMPLEADO": [
        {
        "cedula":"123456789",
        "nombre":"Pedro Perez",
        "email":"pperez@mail.com",
        "clave":"12345",
        },
        {
        "cedula":"223456789",
        "nombre":"Maria Martinez",
        "email":"mmartinez@mail.com",
        "clave":"12345",
        },
        {
        "cedula":"323456789",
        "nombre":"Hernan Hernandez",
        "email":"hhernandez@mail.com",
        "clave":"12345",
        }
    ]
}
```

Identifique la estructura de las tablas que generó su proyecto y cree archivos JSON para cada una de ellas con al menos un registro.

7. Para iniciar las pruebas cree un método `test` que permita evaluar la búsqueda de registros.

```
@Test
public void buscarTest() {

}
```

8. Adicione a su método Test la anotación `@Transactional(value=TransactionMode.ROLLBACK)`, esto permitirá que al finalizar la transacción los cambios sean reversados.

```
@Test
```

```
@Transactional(value=TransactionMode.ROLLBACK)
public void buscarTest() {

}
```

9. Ahora procederemos a cargar uno de los archivos JSON para verificar se estructura (ejemplo para el json empleado).

```
@Test
@Transactional(value=TransactionMode.ROLLBACK)
@UsingDataSet({"empleado.json"})
public void buscarTest(){

}
```

10. De clic derecho sobre la prueba y ejecútela. En caso de error realice las correcciones necesarias y vuelva a probar.

11. Adicione al método otro de los archivos JSON, y ejecute las pruebas nuevamente. Continúe adicionando uno a uno los archivos json y ejecutando las pruebas hasta haber incluido todos los archivos.

12. Haga uso del `entityManager` para buscar uno de los registros creados en los archivos JSON. Ejemplo: Su

```
@Test
@Transactional(value=TransactionMode.ROLLBACK)
@UsingDataSet({"empleado.json"})
public void buscarTest(){
	Empleado empleado = entityManager.find(Empleado.class, "123456789");
}
```

13. Adicione a su método de prueba las afirmaciones (***assert***) que le permitan verificar de forma automática que efectivamente se obtuvo el registro en la base de datos.

```
@Test
@Transactional(value=TransactionMode.ROLLBACK)
@UsingDataSet({"empleado.json"})
public void buscarTest(){
	Empleado empleado = entityManager.find(Empleado.class, "123456789");
	Assert.assertEquals("pperez@mail.com", empleado.getEmail());
}
```

14. Después de haber probado el correcto funcionamiento de la búsqueda, pruebe el funcionamiento de la inserción de datos. Para ello cree un método `registrarTest` y en él cree una instancia de la entidad (Declare una variable de tipo `Entidad` y cree la instancia con **new**) y asigne a los atributos de la instancia los valores que desea almacenar en la base de datos (Para esto puede hacer uso de los métodos set de su entidad).

```
@Test
@Transactional(value=TransactionMode.ROLLBACK)
@UsingDataSet({"empleado.json"})
```

```
public void registrarTest(){
	Empleado empleado = new Empleado();
	empleado.setCedula("423456789");
	empleado.setNombre("Carlos Mendez");
	empleado.setEmail("cmendez@mail.com");
	empleado.setClave("12345");
}
```

15. Si suponemos que existe entidades que tienen relaciones "muchos a muchos", como sería el caso en que una entidad `Empleado` esté relacionada con muchos Servicios, siendo `Empleado` el propietario de la relación.

Por lo cual se debe asociar al empleado los servicios con los que se relaciona.

```
@Test
@Transactional(value=TransactionMode.ROLLBACK)
@UsingDataSet({"empleado.json","servicio.json"})
public void registrarTest(){
	Empleado empleado = new Empleado();
	empleado.setCedula("423456789");
	empleado.setNombre("Carlos Mendez");
	empleado.setEmail("cmendez@mail.com");
	empleado.setClave("12345");

	Servicio servicio = entityManager.find(Servicio.class,1);
	List<Servicio> servicios = new ArrayList<Servicio> ();
	servicios.add(servicio);
	empleado.setServicios( servicios );
}
```

16. Asigne una lista o instancia en todas las entidades propietarias que tenga en su solución (diagrama de entidades).

17. Invoque el método `persist` del `entityManager` enviando como parámetro la instancia de la entidad creada previamente. (También dentro del método test)

```
@Test
@Transactional(value=TransactionMode.ROLLBACK)
@UsingDataSet({"empleado.json","servicio.json"})
public void registrarTest(){
	Empleado empleado = new Empleado();
	empleado.setCedula("423456789");
	empleado.setNombre("Carlos Mendez");
	empleado.setEmail("cmendez@mail.com");
	empleado.setClave("12345");

	Servicio servicio = entityManager.find(Servicio.class,1);
	List<Servicio> servicios = new ArrayList<Servicio> ();
	servicios.add(servicio);
	empleado.setServicios( servicios );

	entityManager.persist(empleado);
}
```

El método `persist` permite registrar en el sistema una instancia de una entidad.

18. Para probar el funcionamiento del Junit de clic derecho sobre la clase, acceda al menú Run As y seleccione la opción Junit Test.

19. Ahora se debe verificar por medio de sentencias de tipo Assert que el registro se haya almacenado. Adicione a su método de prueba las afirmaciones (***assert***) que le permitan verificar de forma automática que efectivamente se creó el registro en la base de datos.

```
@Test
@Transactional(value=TransactionMode.ROLLBACK)
@UsingDataSet({"empleado.json","servicio.json"})
public void registrarTest(){
        Empleado empleado = new Empleado();
        empleado.setCedula("423456789");
        empleado.setNombre("Carlos Mendez");
        empleado.setEmail("cmendez@mail.com");
        empleado.setClave("12345");

        Servicio servicio = entityManager.find(Servicio.class,1);
        List<Servicio> servicios = new ArrayList<Servicio> ();
        servicios.add(servicio);
        empleado.setServicios( servicios );

        entityManager.persist(empleado);

        Empleado registrado = entityManager.find (Empleado.class, "423456789");
        Assert.assertEquals(empleado , registrado );
}
```

NOTA: Para poder comparar dos entidades es **necesario** que la entidad Empleado tenga reescritos los métodos `equals`.

Para la próxima clase

20. Cree los métodos de prueba que le permitan verificar no solo la inserción de datos, sino también la actualización, el borrado y la consulta de datos.

Use los métodos `remove` y `merge` de `EntityManager` y el método *`assertNull`* de `Assert`.