Universidad Nacional Autónoma de México

Dirección General de Cómputo de Tecnologías de Información y Comunicación

Diplomado

Desarrollo de Sistemas con Tecnología Java

Ejercicio Catorce

“JUnit 5”

Mtro. ISC. Miguel Ángel Sánchez Hernández

Tabla de contenido

1. Copiar un proyecto en IntelliJ IDEA

Para copiar un proyecto, hacemos lo siguiente:

1. Señalamos el proyecto spring-core-pojodao
2. Apretamos Ctrl + C
3. Luego Ctrl + V
4. Project Name: spring-core-pojodaotest
5. En el archivo pom.xml cambiar las siguientes líneas
    - <artifactId>spring-core-javabean</artifactId> por lo siguiente
        - <artifactId>spring-core-pojodaotest</artifactId>
    - Cambiar las etiquetas <name> con
        - <name> spring-core-pojodaotest </name>
    - Cambiar la etiqueta <description> con
        - <description> Ejemplo de configuración JUnit en Spring </description>
6. Actualizar maven

2. Archivo pow.xml

Modificar el Archivo pow.xml como sigue:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
		xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
		xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
		<modelVersion>4.0.0</modelVersion>
		<groupId>dgtic</groupId>
		<artifactId>spring-core-pojodaotestcero</artifactId>
		<version>0.0.1-SNAPSHOT</version>
		<name>Ejemplo de configuracion JUnit en Spring</name>
		<description>Ejemplo de configuracion JUnit en Spring</description>
		<parent>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-parent</artifactId>
			<version>2.7.1</version>
			<relativePath />
		</parent>
		<properties>
			<java.version>17</java.version>
			<spring.version>5.3.18</spring.version>
		</properties>
		<dependencies>
			<dependency>
				<groupId>org.springframework</groupId>
				<artifactId>spring-context</artifactId>
			</dependency>
			<dependency>
				<groupId>org.springframework</groupId>
				<artifactId>spring-context-support</artifactId>
			</dependency>

			<dependency>
				<groupId>org.springframework.boot</groupId>
				<artifactId>spring-boot-starter-test</artifactId>
				<scope>test</scope>
			</dependency>
		</dependencies>

		<build>
```

```
                        <plugins>
                                <plugin>
                                        <!-- Build an executable JAR -->
                                        <groupId>org.apache.maven.plugins</groupId>
                                        <artifactId>maven-jar-plugin</artifactId>
                                        <version>3.1.0</version><!--$NO-MVN-MAN-VER$ -->
                                        <configuration>
                                                <archive>
                                                        <manifest>
                                                                <addClasspath>true</addClasspath>
                                                                <classpathPrefix>lib/</classpathPrefix>
                                                                <mainClass>dgtic.core.Inicio</mainClass>
                                                        </manifest>
                                                </archive>
                                        </configuration>
                                </plugin>
                                <plugin>
                                        <groupId>org.springframework.boot</groupId>
                                        <artifactId>spring-boot-maven-plugin</artifactId>
                                </plugin>
                        </plugins>
                </build>
</project>
```

3. Clase Estudiante

Crear la clase Estudiante con las siguientes características:

- Nombre de la clase: Estudiante
- Paquete: dgtic.core.modelo

```
package dgtic.core.modelo;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Objects;
public class Estudiante {
        private String matricula;
        private String nombre;
        private int edad;
        private List<Materia> materias=new ArrayList<>();
        public Estudiante(String matricula, String nombre, int edad) {
                super();
                this.matricula = matricula;
                this.nombre = nombre;
                this.edad = edad;
        }
        public String getMatricula() {
                return matricula;
        }
        public void setMatricula(String matricula) {
                this.matricula = matricula;
        }
        public String getNombre() {
                return nombre;
        }
        public void setNombre(String nombre) {
                this.nombre = nombre;
        }
```

```
	public int getEdad() {
		return edad;
	}
	public void setEdad(int edad) {
		this.edad = edad;
	}
	public List<Materia> getMaterias() {
		return materias;
	}
	public void setMaterias(Materia ...materias ) {
		this.materias.addAll(Arrays.asList(materias));
	}
	@Override
	public String toString() {
		return "Estudiante [matricula=" + matricula + ", nombre=" + nombre + ", edad=" + edad + ", materias=" + materias
				+ "]";
	}


	@Override
	public int hashCode() {
		return Objects.hash(edad, materias, matricula, nombre);
	}
	@Override
	public boolean equals(Object obj) {
		if (this == obj)
			return true;
		if (obj == null)
			return false;
		if (getClass() != obj.getClass())
			return false;
		Estudiante other = (Estudiante) obj;
		return edad == other.edad && Objects.equals(materias, other.materias)
				&& Objects.equals(matricula, other.matricula) && Objects.equals(nombre, other.nombre);
	}



}
```

4. Clase CreditosMenores

Crear la clase CreditosMenores con las siguientes características:

- Nombre de la clase: CreditosMenores
- Paquete: dgtic.core.excepsiones

```
package dgtic.core.excepciones;
public class CreditosMenores extends RuntimeException{
		private static final long serialVersionUID = 1L;
		public CreditosMenores(String message) {
			super(message);
		}
}
```

5. Clase Materia

Crear la clase Materia con las siguientes características:

- Nombre de la clase: Materia
- Paquete: dgtic.core.modelo

```
package dgtic.core.modelo;
import java.util.Objects;
import dgtic.core.excepciones.CreditosMenores;
public class Materia {
		private String nombre;
		private Integer creditos;
		public Materia(String nombre, Integer creditos) {
			super();
			this.nombre = nombre;
			this.creditos = creditos;
		}
		public String getNombre() {
			return nombre;
		}
		public void setNombre(String nombre) {
			this.nombre = nombre;
		}
		public Integer getCreditos() {
			return creditos;
		}
		public void setCreditos(Integer creditos) {
			if(creditos<0) {
				throw new CreditosMenores("No creditos negativos");
			}else {
				this.creditos = creditos;
			}

		}
		@Override
		public String toString() {
			return "Materia [nombre=" + nombre + ", creditos=" + creditos + "]";
		}
		@Override
		public int hashCode() {
			return Objects.hash(creditos, nombre);
```

```
	}
	@Override
	public boolean equals(Object obj) {
		if (this == obj)
			return true;
		if (obj == null)
			return false;
		if (getClass() != obj.getClass())
			return false;
		Materia other = (Materia) obj;
		return Objects.equals(creditos, other.creditos) && Objects.equals(nombre, other.nombre);
	}

}
```

6. @Test

Un caso de prueba será solamente un método público con la anotación @Test

7. @assertEquals

Verificara que los valores esperados y actuales puedan ser iguales.

8. Clase ServicioDAOTest

Crear la clase ServicioDAOTest con los siguientes datos:

- Nombre de la clase: ServicioDAOTest
- Paquete (test/java): dgtic.core.servicio

```
package dgtic.core.servicio;
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;
@SpringBootTest(classes = {ServicioDAOTest.class})
public class ServicioDAOTest {
	@Test
	public void testUno(){
		String esperado="Spring";
		String actual="spring";
		assertEquals(esperado, actual,"Cadenas no iguales");
	}

}
```

Nota: para correr la prueba unitaria completa, presionar (ctrl+shift+F10)

9. @BeforeEach,@AfterEach

@BeforeEach se ejecuta en cada prueba unitaria (@Test), mientras que @AfterEach se ejecutara después de terminada cada prueba unitaria.

10. Anotaciones @BeforeEach,@AfterEach

Agregar la siguiente línea código en la clase ServicioDAOTest

```
package dgtic.core.servicio;
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;
@SpringBootTest(classes = {ServicioDAOTest.class})
public class ServicioDAOTest {
		@BeforeEach
		public void inicio(){
				System.out.println("Antes de cada método");
		}

		@AfterEach
		public  void despues(){
				System.out.println("Despues de cada método");
		}
		@Test
		public void testUno(){
				System.out.println("Prueba unitaria");
				String esperado="Spring";
				String actual="spring";
				assertEquals(esperado, actual,"Cadenas no iguales");
		}

}
```

Nota: para correr la prueba unitaria completa, presionar (ctrl+shift+F10)

11. @BeforeAll,@AfterAll

@BeforeAll se ejecuta una sola vez durante todas las pruebas, mientras que @AfterAll se ejecutara una sola vez cuando termine todas las pruebas.

12. Anotaciones @BeforeAll,@AfterAll

Agregar la siguiente línea código en la clase ServicioDAOTest

```
package dgtic.core.servicio;
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;
@SpringBootTest(classes = {ServicioDAOTest.class})
public class ServicioDAOTest {
		@BeforeEach
		public void inicio() {
			System.out.println("Antes de cada método");
		}
		@AfterEach
		public void despues() {
			System.out.println("Despues de cada método");
		}
		@BeforeAll
		public static void unicoInicio() {
			System.out.println("Unica vez al inicio");
		}
		@AfterAll
		public static void unicoFinal() {
			System.out.println("Unica vez al final");
		}
		@Test
		public void testUno(){
			System.out.println("Prueba unitaria");
			String esperado="Spring";
			String actual="spring";
			assertEquals(esperado, actual,"Cadenas no iguales");
		}
}
```

13. @assertArrayEquals

Es para comprobar que dos arreglos son iguales.

14. Anotaciones @assertArrayEquals

Agregar la siguiente línea código en la clase ServicioDAOTest

```
package dgtic.core.servicio;
import static org.junit.jupiter.api.Assertions.assertArrayEquals;
import static org.junit.jupiter.api.Assertions.assertEquals;
import java.util.Random;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;
@SpringBootTest(classes = {ServicioDAOTest.class})
public class ServicioDAOTest {
		private int lista[];

		@BeforeEach
		public void inicio() {
				System.out.println("Antes de cada método");
				Random rd=new Random();
				int limite=rd.nextInt(3)+1;
				lista=new int[limite];
				for(int i=0;i<limite;i++) {
					lista[i]=i;
				}
		}
		@AfterEach
		public void despues() {
				System.out.println("Despues de cada método");
		}
		@Test
		public void testUno(){
				System.out.println("Prueba unitaria");
				String esperado="Spring";
				String actual="spring";
				assertEquals(esperado, actual,"Cadenas no iguales");
		}
		@Test
		public void testDos(){
				int[] esperado= {0,1,2};
				assertArrayEquals(esperado, lista);
		}


}
```

Nota: para correr la prueba unitaria completa, presionar (ctrl+shift+F10)

15. @assertNotNull,@assertNull

Comparar si un objeto es nulo o no.

16. Anotaciones @assertNotNull,@assertNull

Agregar la siguiente línea código en la clase ServicioDAOTest, abajo del último método

```
@Test
public void testTres(){
		Estudiante est=null;
		assertNull(est);
		est=new Estudiante("123", "Raul", 20);
		assertNotNull(est);
}
```

17. @assertSame,@assertNotSame

Verificar si dos variables hacen referencia o no referencia al mismo objeto.

18. Anotaciones @assertSame,@assertNotSame

Agregar la siguiente línea código en la clase ServicioDAOTest, abajo del último método

```
@Test
public void testCuatro() {
		Estudiante valorActual = new Estudiante("234", "Raul", 18);
		Estudiante valorEsperado = new Estudiante("234", "Raul", 18);
		Estudiante valorEsperadoTmp = valorEsperado;
		// dos variables se refieren al mismo objeto
		//assertSame(valorEsperado, valorActual);
		//assertSame(valorEsperado, valorEsperadoTmp);

		//dos variables no se refieren al mismo objeto
		//assertNotSame(valorEsperado, valorActual);

		// Se quiere verificar sin son iguales, respecto a su estado
		// implementar equals
		//assertEquals(valorEsperado, valorActual);

}
```

19. Implementar el método equals en la clase Estudiante

En la parte de abajo del código implementa la clase equals de la clase Estudiante.

```
@Override
public boolean equals(Object obj) {
        if (this == obj)
                return true;
        if (obj == null)
                return false;
        if (getClass() != obj.getClass())
                return false;
        Estudiante other = (Estudiante) obj;
        return edad == other.edad && Objects.equals(materias, other.materias)
                        && Objects.equals(matricula, other.matricula) && Objects.equals(nombre, other.nombre);
}
```

20. Modificar la clase

```
package dgtic.core.servicio;
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;
@SpringBootTest(classes = {ServicioDAOTest.class})
@ComponentScan("dgtic.core")
public class ServicioDAOTest {
@Autowired
   private ServicioDAO servicio;
.
.
.
}
```

21. @assertTrue,@assertFalse

Su objetivo es verificar si una condición es verdadera o falsa.

22. Anotaciones @assertTrue,@assertFalse

Agregar la siguiente línea código en la clase ServicioDAOTest, abajo del último método

```
@Test
public void testCinco() {
		Estudiante est = servicio.getServicioDAO().getEstudiante("ico", "123");
		String esperado="Lógica";
		assertEquals(esperado, est.getMaterias().stream()
					.filter(dato->dato.getNombre().equals("Lógica"))
					.findFirst().get().getNombre(),"Primero");
		assertTrue(est.getMaterias().stream()
					.filter(dato->dato.getNombre().equals("Lógica"))
					.findFirst().isPresent(),"Log");
		assertTrue(est.getMaterias().stream()
					.anyMatch(dato->dato.getNombre().equals("Lógica")));
		assertFalse(est.getMaterias().stream()
					.anyMatch(dato->dato.getNombre().equals("Lógica")));

}
```

23. @assertAll

Agrupar pruebas unitarias las sigue ejecutando hasta que una falle.

24. Anotaciones @assertAll

Agregar la siguiente línea código en la clase ServicioDAOTest, abajo del último método

```
@Test
public void testSeis() {
		Estudiante est = servicio.getServicioDAO().getEstudiante("ico", "123");
		String esperado="Lógica";
		//assertAll(()->{},()->{},()->{});
		assertAll(()->{assertEquals(esperado, est.getMaterias().stream()
					.filter(dato->dato.getNombre().equals("Lógica"))
					.findFirst().get().getNombre());},
					()->{assertTrue(est.getMaterias().stream()
								.filter(dato->dato.getNombre().equals("Lógic"))
								.findFirst().isPresent());},

					()->{assertTrue(est.getMaterias().stream()
								.anyMatch(dato->dato.getNombre().equals("Lógica")));}
					);
}
```

25. @assertThrows

Nos permite en una forma sencilla afirmar si la línea ejecutada manda una excepción especifica.

26. Anotaciones @assertThrows

Agregar la siguiente línea código en la clase ServicioDAOTest, abajo del último método

```
@Test
public void testSiete() {
		Materia materia = new Materia("Algebra", 23);
		Exception exp = assertThrows(CreditosMenores.class, () -> {
			materia.setCreditos(-20);
		});
		String actual = exp.getMessage();
		String esperado = "No creditos negativos";
		// verdadero, se prueba el error
		assertEquals(esperado, actual);
}
```

27. @assertIterableEquals, @DisplayName

Revisa si los iterables son iguales, es decir en el mismo orden y se requiere que sean del mismo tipo. Con @DisplayName cambiamos el que se despliega en lugar del nombre del método.

28. Anotaciones @assertIterableEquals

Agregar la siguiente línea código en la clase ServicioDAOTest, abajo del último método

```
@Test
@DisplayName("assertIterableEquals sin equals y con equals")
public void testOcho() {
		Estudiante estUno = servicio.getServicioDAO().getEstudiante("ico", "123");
		Estudiante estDos = servicio.getServicioDAO().getEstudiante("ico", "124");
		//Sin implementar equals en Materia
		//assertIterableEquals(estUno.getMaterias(), estDos.getMaterias());

		//sin implementar equals en Materia
		//estDos.getMaterias().add(0, new Materia("Cálculo",9));
		//assertIterableEquals(estUno.getMaterias(), estDos.getMaterias());

		//implementado equals
		//estDos.getMaterias().add(0, new Materia("Cálculo",9));
		//assertIterableEquals(estUno.getMaterias(), estDos.getMaterias());
}
```

29. @assertTimeout

Nos permite hacer pruebas donde nos interese probar un proceso cuanto tiempo se tarda.

30. Anotaciones @assertTimeout

Agregar la siguiente línea código en la clase ServicioDAOTest, abajo del último método

```
@Test
public void testNueve() {
		assertTimeout(java.time.Duration.ofSeconds(2), ()->{
			Thread.sleep(1000);
		});
}
```