EXAMEN PPSS JUNIO 2022

1. A partir del proyecto multimódulo matriculación visto en prácticas, en el que parte de sus coordenadas son las siguientes: (1p)

```
<artifactId>matriculacion</artifactId>
<groupId>ppss</groupId>
<version>1.0-SNAPSHOT</version>
```

Se pide:

1.a) Indica qué tenemos que añadir en el pom.xml para completar las coordenadas de dicho proyecto:

1.b) Indica qué tenemos que añadir en el pom.xml para poder compilar todos sus módulos con un único comando maven (debes indicar sólo lo que es necesario para lo que se pide).

1.c) Indica qué se debe añadir al pom.xml de cualquiera de sus módulos hijo para que puedan heredar los elementos de su configuración.

1.d) Tenemos la configuración del siguiente plugin en el pom.xml. Suponiendo que algún test de integración falla al ejecutar el comando "mvn install", indica si el proceso de construcción se interrumpirá. En caso afirmativo indica en qué fase se interrumpe y en caso negativo justifica por qué no se interrumpe.

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-failsafe-plugin</artifactId>
    <version>3.0.0-M5</version>
    <executions>
        <execution>
            <goals>
                <goal>integration-test</goal
                 <goal>verify</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

a) A este pom no tenemos que añadirle nada, ya esta completo

b) Para compilar todos los modulos de golpe debemos añadir lo siguiente:

```
<packaging> POM </packaging>
<modules>
    <module> matriculacion-dao </module>
    <module> matriculacion-comun </module>
    <module> matriculacion-dao </module>
    <module> matriculacion-proxy </module>
</modules>
```

c) Para que los elementos hijos puedan usar la configuración deben añadir lo siguiente

```
<parent>
    <artifactId> matriculacion </artifactId>
    <groupId> ppss </groupId>
    <version> 1.0-SNAPSHOT </version>
</parent>
```

d) Se interrumpiría en la fase verify si fallara un test de integración.

e) Apartado extra: para configurar las dependencias entre modulos se añade lo siguiente en los modulos

```
<dependencies>
    <dependency>
        <groupId> ${project.groupId} </groupId>
        <artifactId> module </artifactId>
        <version> ${project.version} </version>
    </dependency>
</dependencies>
```

2. Dado el proyecto matriculacion-dao visto en prácticas, completa los siguientes tests de integración para los métodos AlumnoDAO.addAlumno y AlumnoDAO.delAlumno, teniendo en cuenta lo siguiente: (1,5p)

Disponemos de los siguientes ficheros.xml:

tabla0.xml (Base de datos inicial en cada test)

```
<?xml version='1.0' encoding='UTF-8'?>
<dataset>
  <alumnos nif="11111111A" nombre="Alfonso Ruiz" direccion="Rambla, 22" email="alfonso@ua.es" fechaNac="1982-02-22"/>
  <alumnos nif="22222222B" nombre="Laura Perez" direccion="Maisonnave, 5" email="laura@ua.es" fechaNac="1980-02-22"/>
</dataset>
```

tabla1.xml (Base de datos esperada como salida en testDelete)

```
<?xml version='1.0' encoding='UTF-8'?>
<dataset>
  <alumnos nif="22222222B" nombre="Laura Perez" direccion="Maisonnave, 5" email="laura@ua.es" fechaNac="1980-02-22"/>
</dataset>
```

Los casos de prueba cuya implementación has de completar son los siguientes

ID	método a probar	entrada	salida esperada
testAdd	void addAlumno(AlumnoTO alumno)	alumno = null	DAOException
testDelete	void delAlumno(String nif)	nif = "11111111A"	tabla1

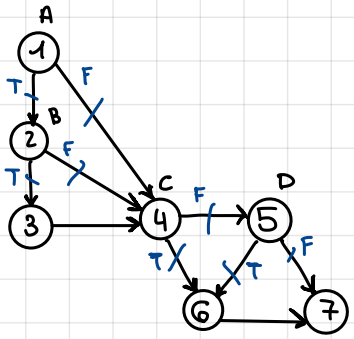
Y la implementación a completar (rellenando los huecos correspondientes) se muestra a continuación, teniendo en cuenta que cada hueco acabado en ';' debe rellenarse con una única sentencia:

```
public class TestIT {

IDatabaseTester databaseTester;
IDatabaseConnection connection;
AlumnoDAO alumnoDAO;

@BeforeEach
public void setup() throws Exception {
    databaseTester = New JDecDatabaseTester("com.mysql.jdbc.driver", "jdbc:mysql://localhost:3306/DBUNIT?
    useSSL=false", "root", "ppss");
    connection = databaseTester.getConnectiont();
    IDataset inject = new FlatxmlFileLoader().load("tabla0.xml");
    databaseTerster.setDataset(inject);
    databaseTester.onSetup();
    alumnoDAO = New FactoriaDAO.getAlumnoDAO();
}

@Test
public void testAdd throws Exception {
    Assertions.assertthrows(DAOException.class, ()->alumnoDAO.addAlumno(null));
}

@Test
public void testDel throws Exception {
    Assertions.assertDoesNotThrow(()->alumnoDAO.delAlumno("11111111A"));
    IDataset actual = connection.createDataset();
    ITable actualTable = actual.getTable("alumnos");
    IDataset expected = new FlatxmlFileLoader().load("tabla1.xml");
    ITable expectedTable = expected.getTable("alumnos");
    Assertion.assertEquals(expectedTable, actualTable);
}

}
```

3. Responde a las siguientes cuestiones acerca del análisis de cobertura de código y de la herramienta JaCoCo: (1,5p)

3.a) Dado el siguiente fragmento de código donde 'A', 'B', 'C' y 'D' representan condiciones booleanas simples y los operadores 'and' y 'or' se evalúan con cortocircuito, indica con el menor número posible de casos de prueba el valor booleano que deben tener cada una de las condiciones simples en cada caso de prueba para obtener los niveles de cobertura indicados:

```
if (A and B) {
  x = x+1;
}
if (C or D) {
  x = x+2;
}
else {
  x = x+3;
}
```

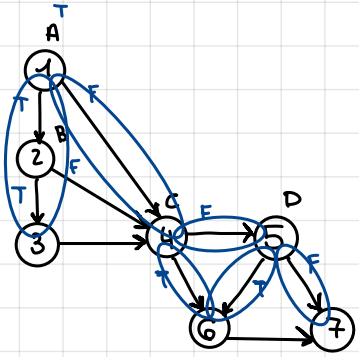


3.a.1) Una cobertura de ramas (decisiones) del 100% (nivel 2) (completa las filas (casos de prueba) que consideres necesarias)

Id	A	B	C	D
C1	True	True	True	True
C2	False	True	False	False
C3	True	False	False	True
C4				
C5				

3.a.2) Una cobertura de condiciones del 100% (nivel 3) (completa las filas (casos de prueba) que consideres necesarias)

Id	A	B	C	D
C1	True	True	True	True
C2	True	False	False	False
C3				
C4				
C5				



3.b) Completa la siguiente configuración del plugin jacoco que provoca el fallo en el proceso de construcción indicado por los siguientes mensajes:

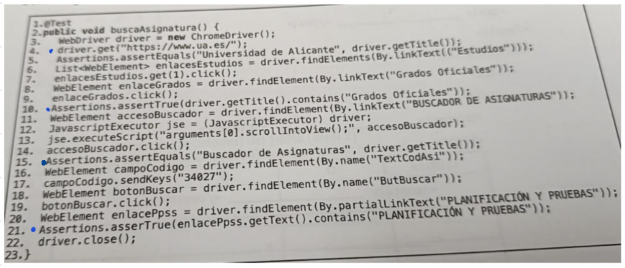
[WARNING] Rule violated for bundle cobertura: methods missed count is 4, but expected maximum is 2
[WARNING] Rule violated for class ejercicio2.MyClass: lines covered ratio is 0.71, but expected minimum is 0.75

```
<execution>
  <goals>
    <goal> check </goal>
  </goals>
  <configuration>
    <rules>
      <rule>
        <element> Bundle </element>
        <limits>
          <limit>
            <counter> Method </counter>
            <value> COVEREDRATIO </value>
            < maximum > 2 </ maximum >
          </limit>
        </limits>
      </rule>
      <rule>
        <element> CLASS </element>
        <limits>
          <limit>
            <counter> LINE </counter>
            <value> COVEREDRATIO </value>
            < minium > 0.75 </ minium >
          </limit>
        </limits>
      </rule>
</configuration>
```

4. Queremos realizar pruebas de propiedades emergentes funcionales sobre la aplicación web de la UA. En concreto, el siguiente caso de prueba: (2,5p)

Accedemos a la página principal de la UA, y desde ahí a la opción de menú "Estudios" y la opción del submenú "Grados Oficiales". Desde una nueva página pulsamos el botón "BUSCADOR DE ASIGNATURAS" que accede a otra página en la que se introduce el código de asignatura "34027" y después de pulsar el botón "Buscar" aparece el nombre parcial de la asignatura "PLANIFICACIÓN Y PRUEBAS"

Dado el siguiente test que implementa el caso de prueba anterior con selenium WebDriver:



```
1.@Test
2.public void buscaAsignatura() {
3.   WebDriver driver = new ChromeDriver();
4.   driver.get("https://www.ua.es/");
5.   Assertions.assertEquals("Universidad de Alicante", driver.getTitle());
6.   List<WebElement> enlacesEstudios = driver.findElements(By.linkText("Estudios"));
7.   enlacesEstudios.get(1).click();
8.   WebElement enlaceGrados = driver.findElement(By.linkText("Grados Oficiales"));
9.   enlaceGrados.click();
10.  Assertions.assertTrue(driver.getTitle().contains("Grados Oficiales"));
11.  WebElement accesoBuscador = driver.findElement(By.linkText("BUSCADOR DE ASIGNATURAS"));
12.  JavascriptExecutor jse = (JavascriptExecutor) driver;
13.  jse.executeScript("arguments[0].scrollIntoView();", accesoBuscador);
14.  accesoBuscador.click();
15.  Assertions.assertEquals("Buscador de Asignaturas", driver.getTitle());
16.  WebElement campoCodigo = driver.findElement(By.name("TextCodAsi"));
17.  campoCodigo.sendKeys("34027");
18.  WebElement botonBuscar = driver.findElement(By.name("ButBuscar"));
19.  botonBuscar.click();
20.  WebElement enlacePpss = driver.findElement(By.partialLinkText("PLANIFICACIÓN Y PRUEBAS"));
21.  Assertions.assertTrue(enlacePpss.getText().contains("PLANIFICACIÓN Y PRUEBAS"));
22.  driver.close();
23.}
```

4.a) implementa el test equivalente usando el patrón Page Object y la clase PageFactory.

4.b) Indica cuántas Page Object necesitas (indica también sus nombres) e implementa únicamente la que primero se acceda desde el test.

```
public class Test1 {
    PrincipalPage poPrincipal;
    GradosOficialesPage poGradosOficiales;
    BuscadorPage poBuscador;
    AsignaturaPage poAsignatura
    WebDriver driver;

    @Test
    public void Test1() {
        driver = New ChromeDriver();
        driver.get("https://ua.es/");
        poPrincipalPage = PageFactory.initElements(drive, PrincipalPage.class);
        Assertions.assertEquals("Universidad de Alicante", driver.getTitle());
        poGradosOficiales = poPrincipalPage.enterGradosOficiales();
        Assertions.assertTrue(driver.getTitle().contains("Grados Oficiales"));
        poBuscador = poGradosOficiales.enterBuscador();
        Assertions.assertEquals("Buscador de Asignaturas", driver.getTitle());
        poAsignatura = poBuscador.buscar("34027");
        AssertTrue(poAsignatura.getEnlace().contains("PLANIFICACIÓN Y PRUEBAS");
}
```

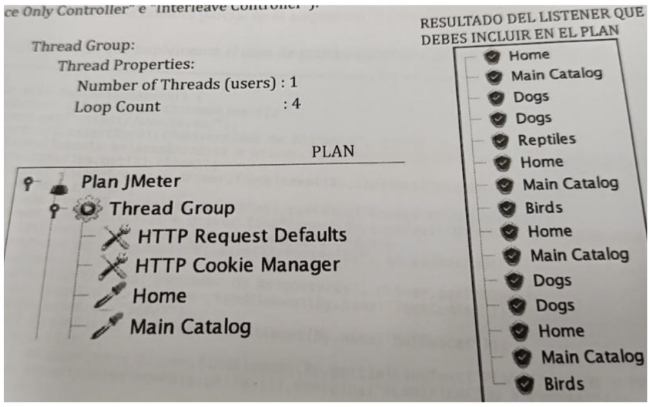
4) Necesitamos 4 page Objects PrincipalPage, GradosOficialesPage, BuscadorPage, AsignaturaPage.

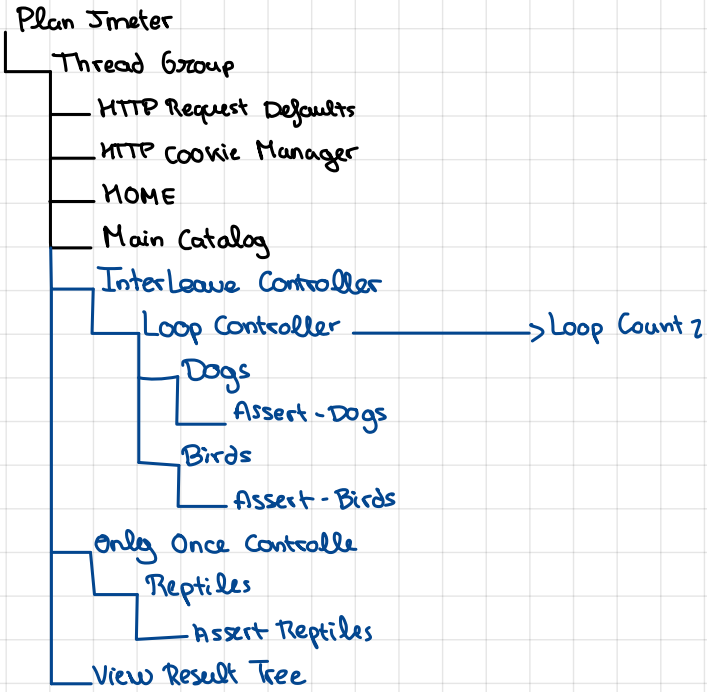
```
public class PrincipalPage {
    Webdriver driver;
    @FindBy(linkText = Estudios) List<WebElement> enlacesEstudios;

    public PrincipalPage(Webdriver driver){
        this.driver = driver;
    }

    public GradosOficialesPage enterGradosOficiales(){
        enlacesEstudios.get(1).click();
        return PageFactory.initElements(driver, GradosOficialesPage.class);
    }
}
```

5. Completa la implementación del siguiente plan JMeter. Para ello proporcionamos la configuración de grupo de hilos y el resultado del listener de dicho plan.

Debes tener en cuenta que cada una de las peticiones http sólo debe aparecer una única vez en el plan, y que debes usar los controladores lógicos "Loop Controller" (indicando también el valor de loop count),

"Once Only Controller" e "Interleave Controller"). (2p)





```
<exeutions>
    <execution>
        <id> default-prepare-agent </id>
        <goals>
            <goal> prepare-agent </goal>
        </goals>
    </execution>
    <execution>
        <id> default-check </id>
        <goals>
            <goal> check </goal>
        </goals>
        <configurations>
            <rules>
                <rule>
                    <element> ——— </element>
                    <limits>
                        <limit>
                            <counter> ——— </counter>
                            <value> ——— </value>
                            <maximum> ——— </maximum>
                            minimum                 minimum
                        </limit>
                </rule>
            </rules>
        </configuration>
```

```
public class test{
  IDatabasetester databaseTester;
  IDatabaseConnection connection;
  @BeforeEach
  public void setup() throws Exception{
    databaseTester = new JDBCDatabasetester(——— - —)
    connection= databaseTester.getConnection();
    IDataset init = new flatxmlfileloader().load("init.xml");
```