**Indica cuál de las siguientes afirmaciones es falsa:**

a) Aunque probemos un código utilizando un conjunto de pruebas eficiente y efectivo y todos los test pases, no podemos garantizar que el código no presenta ningún defecto
b) Las pruebas pueden demostrar la ausencia de defectos
c) Durante el proceso de desarrollo de un producto, cuanto antes se detecte un defecto menos costosos será repararlo
d) Aunque un producto funcione de acuerdo con los requerimientos especificados puede que no satisfaga las expectativas del cliente

**Cualquier librería que sea requerida en el proceso de construcción de un proyecto Maven**

a) si se encuentra en un repositorio remoto, se descarga en el directorio target
b) si no se encuentra en un repositorio remoto, se busca en el repositorio local
c) si no se encuentra en un repositorio remoto, se busca en el directorio .m2
d) todas las afirmaciones anteriores son falsas

**El método de diseño de transición de estados:**

a) Requiere que hayamos implementado todas las unidades para poder aplicarlo
b) Al igual que el método del camino básico se aplica en pruebas dinámicas
c) Es un método funcional y por lo tanto podremos aplicar en cualquier nivel de pruebas
d) Representa las transiciones a partir de entradas y acciones

**Dado el siguiente driver para probar metodoSUT():**

```
@Test (expected =MiExcepcion.class)
Public void testC1(){
   Int resultadoEsperado =0;
   Int resultadoReal = metodoSut();
   AssertEquals(resultadoEsperado, resultadoReal);
}
```

En el informe de pruebas para dicho test aparecerá:

a) Passed, si el metodoSUT() devuelve 0
b) Error, si el metodoSUT() provoca la excepción MiExecpcion.class
c) Failed, si el metodoSUT() provoca la excepción MiExecpcion.class
d) Passed, si el metodoSUT() provoca la excepción MiExecpcion.class

**Indica cual de las siguientes afirmaciones es falsa acerca del uso de categorías en jUnit**

a) Se anota como categoría una clase que contiene drivers
b) Se anota como categoría una interfaz
c) Se anota como categoría varios drivers de una clase
d) Se anota como categoría un método anotado como @test

**Con cual de los siguientes comando de Maven no se ejecutarían las pruebas unitarias:**

a) Mvn clean compile surefire:test
b) Mvn test-compile surefire:test
c) Mvn clean test-compile surefire:test
d) Mvn test

**Indica cual de las siguientes afirmaciones es cierta con respecto a estos 3 tipos de objeto**
**Podemos crear una librería EasyMock: NiceMock, mock y StrictMock**

a) Con los tipos Mock y StrictMock se verifica el orden en el que se realizan las invocaiones al doble
b) Con los tipos Mock y NiceMock se lanza un AssertionError si no hemos programado las expect.. de algún método
c) Con los tres tipos se verifica si todas las llamadas esperadas a métodos realizadas por el doble .realizan con los argumentos especificados
d) Con los tres tipos siempre hay que invocar al método Verify() de la librería.

**Si al aplicar el método de caja negra de particiones equivalentes obtendremos las siguientes particiones de entrada, validas y no validas, teniendo en cuenta la siguiente codificación para identificar las particiones "E" denota entrada, "V" denota valida y "nV" denota no valida.**

Entrada 1 : "E1V1, E1V2, E1nV1"
Entrada 2 : "E2V1, E2nV1, E2nV2"
Indica cual es la cardinalidad del conjunto de casos de prueba eficiente y efectivo al aplicar dicho método:

a) 5
b) 6
c) 4
d) No se puedo obtener si no se conocen las particiones de salida validas y no válidas.

**El comando nvm de Maven**

a) Sirve para ejecutar goals pero no fases
b) Sirve para ejecutar fases, pero no goals
c) Se ejecuta desde el directorio src para compilar los ficheros fuentes y desde el directorio test para compilar los ficheros con las pruebas
d) Se ejecuta desde el directorio donde se encuentra el fichero de configuración del proyecto Maven

**Con respecto al método de caja negra de particiones equivalente, indica cual de las siguientes afirmaciones es cierta:**

a) Dos elementos de la misma partición de entrada no se pueden corresponder con dos elementos de particiones de salida diferentes
b) Dejo la pregunta en blanco
c) Todas las entradas y salidas se corresponden con parámetros de la SUT
d) Se deben probar todas las posibles combinaciones de las particiones
e) Las particiones pueden compartir elementos para mantener algunos datos de prueba redundantes

**Indica las líneas en las que identificamos las dependencias externas de la SUT calculaConsumo():**

```
1.//paquete ppss.ejercicio2
2.public class GestorLlamadas {
3.  static double TARIFA_NOCTURNA=10.5;
4.  static double TARIFA_DIURNA=20.8;
5.
6.  public Calendario getCalendario() {
7.    Calendario c = new Calendario();
8.    return c;
9.  }
10.
11.  public double calculaConsumo(int minutos) {
12.    Calendario c = getCalendario();
13.    int hora = c.getHoraActual();
14.    if(hora < 8 || hora > 20) {
15.      return minutos * TARIFA_NOCTURNA;
16.    } else {
17.       return minutos * TARIFA_DIURNA;
18.    }
19.  }
20.}
```

a) En las líneas 7 y 12
b) En las líneas 12 y 13
c) Dejo la pregunta en blanco
d) En las líneas 7, 12, y 13
e) Solo en la línea 13

**Indica la línea o líneas en las que tenemos puntos de inyección de seams para la SUT calculaPrecio()**

```
1.public class AlquilaCoches {
2.  protected Calendario calendario = new Calendario();
3.
4.  public Ticket calculaPrecio(TipoCoche tipo, LocalDate inicio, int ndias)
5.                                                    throws MensajeException {
6.    Ticket ticket = new Ticket();
7.    float precioDia,precioTotal =0.0f;
8.    float porcentaje = 0.25f;
9.
10.   String observaciones = "";
11.   IService servicio = new Servicio();
12.   precioDia = servicio.consultaPrecio(tipo);
13.   for (int i=0; i<ndias;i++) {
14.      LocalDate otroDia = inicio.plusDays((long)i);
15.      try {
16.         if (calendario.es_festivo(otroDia)) {
17.           precioTotal += (1+ porcentaje)*precioDia;
18.         } else {
19.           precioTotal += (1- porcentaje)*precioDia;
20.         }
21.      } catch (CalendarioException ex) {
22.          observaciones += "Error en dia: "+otroDia+"; ";
23.      }
24.   }
25.
26.   if (observaciones.length()>0) {
27.      throw new MensajeException(observaciones);
28.   }
29.
30.   ticket.setPrecio_final(precioTotal);
31.   return ticket;
32.  }
33.}
```

```
public class Ticket {
  private float precio_final;
  //getters y setters
}
```

LINEA 2

**Utilizando un método de caja negra de particiones equivalentes si tenemos una entrada asociada a un tipo enumerado con 3 valores, indica cual de las siguientes afirmaciones es falsa**

a) Podemos tener dos particiones validas de dicha entrada
b) Dejo la pregunta en blanco
c) Podemos tener tres particiones no validas de dicha entrada
d) Podemos tener una sola partición valida de dicha entrada
e) Podemos tener tres particiones validas de dicha entrada

Si en el pom.xml de nuestro proyecto añadimos la siguiente propiedad:

```
<properties>
     <miPropiedad>misTests</miPropiedad>
</properties>
```

y la siguiente configuración del plugin maven-surefire-plugin:

```
<configuration>
          <groups>${miPropiedad}</groups>
 </configuration>
```

y tiene la siguiente clase para los tests con 3 drivers:

```
class MiClaseTest {
      @Tag("misTests")
      @Test void test1() {
            // aquí vendría el código del test
       }

      @Tag("otroTest")
      @Test void test2() {
            // aquí vendría el código del test
       }

      @Test void test3() {
            // aquí vendría el código del test
       }
}
```

Si desde línea de comandos ejecutamos la orden

```
mvn test -DmiPropiedad=""
```

**Seleccione una**

a) Se ejecuta solo test1()
b) Se ejecuta solo test3()
c) Dejo la pregunta en blanco
d) Se ejecutan los 3 drivers
e) No se ejecuta ningún driver por que en la orden no se indica ninguna etiqueta

**Seleccione una:**

Indica las líneas en las que identificamos las dependencias externas de la SUT realizaReserva():

```
//paquete ppss
1. public class Reserva {
2.
3.    public boolean compruebaPermisos(String login, String password, Usuario tipoUsu) {
4.       throw new UnsupportedOperationException("Not yet implemented");
5.    }
6.
7.    public void realizaReserva(String login, String password,
8.                  String socio, String [] isbns) throws Exception {
9.
10.    ArrayList<String> errores = new ArrayList<>();
11.    if(!compruebaPermisos(login, password, Usuario.BIBLIOTECARIO)) {
12.        errores.add("ERROR de permisos");
13.    } else {
14.        IOperacionBO io = new Operacion();
15.      try {
16.         for(String isbn: isbns) {
17.            try {
18.              io.operacionReserva(socio, isbn);
19.            } catch (IsbnInvalidoException iie) {
20.               errores.add("ISBN invalido" + ":" +  isbn);
21.            }
22.         }
23.      } catch (SocioInvalidoException sie) {
24.            errores.add("SOCIO invalido");
25.      } catch (JDBCException je) {
26.            errores.add("CONEXION invalida");
27.      }
28.    }
29.    if (errores.size() > 0) {
30.       String mensajeError = "";
31.       for(String error: errores) {
32.         mensajeError += error + "; ";
33.       }
34.       throw new ReservaException(mensajeError);
35.    }
36. }
37.}
```

```
//paquete ppss
public enum Usuario {
  BIBLIOTECARIO, ALUMNO, PROFESOR
}
```

a) En las lineras 14 y 18
b) Dejo la pregunta en blanco
c) En las líneas 10 y 14
d) En las líneas 11 y 18

Cuando refactorizamos la SUT para poder inyectar el doble:

Seleccione una:

- si añadimos un método setter estamos obligando a cualquier código cliente de la SUT a conocer dicha dependencia antes de invocarla
- si añadimos un parámetro a la SUT estamos obligados a declarar la dependencia como un atributo de la clase que contiene la SUT
- si añadimos una factoría local, alteramos el comportamiento de la clase que contiene la SUT
- si añadimos una clase factoría no se altera el código de producción
- Dejo la pregunta en blanco

En un método de diseño de pruebas basado en el flujo de control del código, un camino imposible detectado:

Seleccione una:

- no se asocia a ningún caso de prueba
- se asocia a un caso de prueba con un resultado esperado desconocido
- se asocia a un caso de prueba con valores de entrada desconocidos
- Dejo la pregunta en blanco
- todas las afirmaciones del resto de opciones son falsas

Indica cuál de las siguientes afirmaciones es cierta con respecto a estos 3 tipos de objeto que podemos crear con la librería EasyMock: NiceMock, Mock y StrictMock:

Seleccione una:

- con los 3 tipos siempre hay que invocar al método replay() de la librería
- con los tipos Mock y StrictMock se verifica el orden en el que se realizan las invocaciones al doble
- con los 3 tipos hay que invocar al método verify() de la librería antes de invocar a la sut
- con los tipos Mock y NiceMock se lanza un AssertionError si no hemos programado las expectativas de algún método
- Dejo la pregunta en blanco

Al ejecutar una clase de test que contiene n drivers:

Seleccione una:

- el método anotado con @AfterEach se ejecuta n+1 veces
- el método anotado con @BeforeAll se ejecuta una vez
- el método anotado con @AfterAll se ejecuta n veces
- el método anotado con @BeforeEach se ejecuta n-1 veces
- Dejo la pregunta en blanco

De los siguientes comandos de maven, ¿cuáles generan el .jar del proyecto?

Comando 1: mvn clean compiler:testCompile package
Comando 2: mvn clean compile package
Comando 3: mvn clean test-compile package
Comando 4: mvn clean verify
Comando 5: mvn clean compiler:compile verify

Seleccione una:

- Dejo la pregunta en blanco
- todos los comandos
- los comandos 1, 2, y 3
- los comandos 4 y 5
- los comandos 2, 3, 4 y 5

Dado el siguiente método, que devuelve el sumatorio desde 1 hasta un determinado valor n:

```
int sumatorio(int n) {
      int suma = 0;
      for (int i=1; i <= n; i++) {
            suma += i;
      }
      return suma;
}
```

Se puede aplicar el método del camino básico

Seleccione una:

- con uno o 2 caminos
- al contener un bucle cuyo número de iteraciones depende del valor de entrada n, no podemos saber el número de caminos
- con n caminos
- Dejo la pregunta en blanco
- con más de 2 caminos

Cualquier librería que sea requerida en el proceso de construcción de un proyecto Maven

Seleccione una:

- si se encuentra en un repositorio remoto, se descarga en el directorio target
- si no se encuentra en un repositorio remoto, se busca en el repositorio local
- Dejo la pregunta en blanco
- si no se encuentra en un repositorio remoto, se busca en el directorio $HOME/.m2
- todas las afirmaciones del resto de opciones son falsas

De los siguientes comandos de maven, ¿cuáles ejecutarán los tests uni

Comando 1: mvn clean surefire:test
Comando 2: mvn clean test
Comando 3: mvn test
Comando 4: mvn clean compile surefire:test
Comando 5: mvn clean test-compile surefire:test

Seleccione una:

- los comandos 2, 3 y 5
- Dejo la pregunta en blanco
- los comandos 1, 2, 4 y 5
- los comandos 1, 4 y 5
- los comandos 4 y 5

Para realizar pruebas unitarias utilizando verificación basada en el comportamiento de la SUT compruebaPremio(), indica los tipos y número de mocks nece

```
public class Premio {
    private static final float PROBABILIDAD_PREMIO = 0.1f;
    public Random generador = new Random(System.currentTimeMillis());
    public ClienteWebService cliente = new ClienteWebService();

    public String compruebaPremio() {
        if(generaNumero() <  PROBABILIDAD_PREMIO) {
            try {
                String premio = cliente.obtenerPremio();
                return "Premiado con " + premio;
            } catch(ClienteWebServiceException e) {
                return "No se ha podido obtener el premio";
            }
        } else {
            return "Sin premio";
        }
    }

    // Genera numero aleatorio entre 0 y 1
    public float generaNumero() {
        return generador.nextFloat();
    }
}
```

Seleccione una:

- un PartialMock y un Mock
- un StrictControl, un PartialMock y un Mock
- un PartialMock y un StrictMock
- Dejo la pregunta en blanco
- un StrictControl y dos StrictMock

Indica cuál de las siguientes afirmaciones es cierta:

Seleccione una:

- si es necesario, se puede alterar temporalmente el código de la SUT para realizar las pruebas
- si es necesario, refactorizamos la SUT para eliminar sus dependencias externas
- Dejo la pregunta en blanco
- un doble reemplaza al código real de una dependencia externa durante las pruebas
- un SEAM es un punto de inyección del doble

Utilizando el método de caja negra de particiones equivalentes, si tenemos una entrada asociada a un tipo enumerado con 3 valores, indica cuál de las siguientes afirmaciones es falsa:

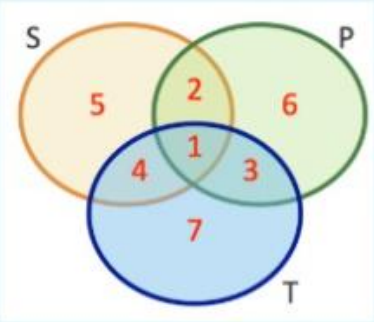
Seleccione una:

- podemos tener dos particiones válidas de dicha entrada
- Dejo la pregunta en blanco
- podemos tener tres particiones no válidas de dicha entrada
- podemos tener una sola partición válida de dicha entrada
- podemos tener tres particiones válidas de dicha entrada

Los enum nunca tienen particiones inválidas

---

Dado el siguiente diagrama de Venn que hemos trabajado en clase:




Indica cuál de las siguientes afirmaciones es cierta:

Seleccione una:

- Dejo la pregunta en blanco
- Un tester debe intentar que el subconjunto 7 sea lo más grande posible
- Con métodos de caja negra y de caja blanca, se pueden alcanzar comportamientos de los subconjuntos 1 y 2
- Con un método de caja negra se pueden alcanzar comportamientos del subconjunto 3
- Con un método de caja blanca se pueden alcanzar comportamientos del subconjunto 4

El artefacto maven con las siguientes coordenadas:

`ppss.examen:ejemplo:war:1.0-SNAPSHOT`

representa el fichero:

Seleccione una:

- $HOME/.m2/repository/ppss/examen/ejemplo/1.0-SNAPSHOT/ejemplo-1.0-SNAPSHOT.war
- $HOME/.m2/repository/ppss/examen/ejemplo-1.0-SNAPSHOT.war
- $HOME/.m2/repository/ppss/examen/ejemplo/ejemplo-1.0-SNAPSHOT.war
- $HOME/.m2/repository/ppss/examen/ejemplo/war/1.0-SNAPSHOT/ejemplo-1.0-SNAPSHOT.war
- Dejo la pregunta en blanco

Si al aplicar el método de caja negra de particiones equivalentes, obtenemos las siguientes particiones de entrada, válidas y no válidas, teniendo en cuenta la siguiente codificación para identificar las particiones: 'E' denota entrada; 'V' denota válida; 'nV' denota no válida:

Entrada 1: E1V1, E1nV1
Entrada 2: E2V1, E2nV1, E2nV2

Indica cuál es la cardinalidad del conjunto de casos de prueba eficiente y efectivo obtenido al aplicar dicho método

Seleccione una:

- 4
- Dejo la pregunta en blanco
- 5
- No se puede obtener si no se conocen las particiones de salida válidas y no válidas
- 3

Con el método del camino básico de McCabe:

Seleccione una:

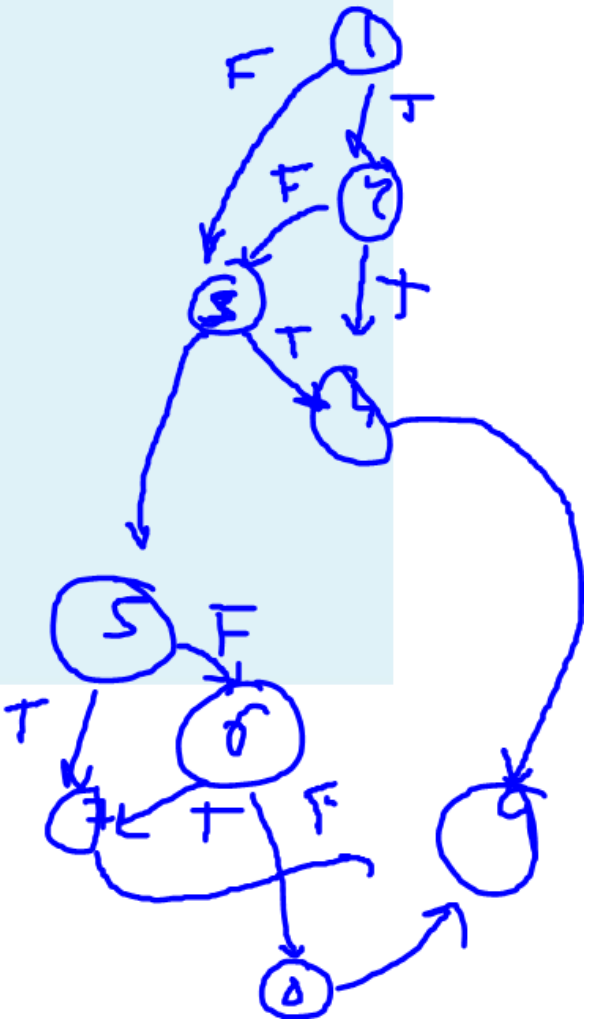
- debemos elegir el conjunto mínimo de caminos para conseguir que todas las sentencias se ejecuten al menos una vez en cada caso de prueba
- Dejo la pregunta en blanco
- todas las afirmaciones del resto de opciones son falsas
- debemos elegir el conjunto mínimo de caminos para conseguir ejecutar todas las condiciones al menos una vez en cada caso de prueba
- debemos elegir todos los caminos del grafo

Indica cuál es la complejidad ciclomática del siguiente código:

```
if (a > b && a < c || c < b)
        a = b;
else if (a < b || c > a)
        c = a;
else
        b = c;
```

Seleccione una:

- 6
- 4
- 3
- Dejo la pregunta en blanco
- 5




17 − 9 + 2 = 6