parcial1PPSS_2024.pdf

gigantetitan

Planificación y Pruebas de Sistemas Software

3º Grado en Ingeniería Informática

Escuela Politécnica Superior
Universidad de Alicante

1. Dados los siguientes ficheros de un proyecto Maven creado en $HOME
   - ExamenPPSS-1.0-SNAPSHOT.jar
   - Parcial1.java
   - Parcial1Test.java
   - Parcial1.class
   - Parcial1Test.class
   - ppss.Parcial1Test.txt
   - TEST-ppss.Parcial1Test.xml
   - pom.xml

   a) El artifactId del proyecto es _____

   b) Indica la ruta relativa desde $HOME de cada uno de los ficheros, representando gráficamente la estructura jerárquica de directorios

c) En este proyecto, cualquier comando maven se tiene que ejecutar desde la carpeta _____

d) Indica los nombres de ficheros que se crean, si es que se crea alguno, después de ejecutar cada uno de los siguientes comandos **secuencialmente** y teniendo en cuenta las suposiciones que se mencionan:

d1) Suponemos que Parcial1Test.java contiene errores de compilación y ejecutamos el comando "mvn clean test"

d2) Suponemos que corregimos los errores de compilación de Parcial1Test.java y ejecutamos el mismo comando

d3) Ejecutamos el comando "mvn package"

d4) Ejecutamos el comando "mvn install"

2. *Queremos implementar un buscador de cines en un rango de kms, a partir de un código postal, con el siguiente prototipo:*

```
Mensaje buscador(String cod_postal, float dmin, float dmax)
```

Si el rango de distancias es incorrecto, se generará el mensaje de error: "cambie el rango de búsqueda" y el proceso termina. Una vez validados todos los datos de entrada, el sistema iniciará la búsqueda y devolverá un objeto de tipo **Mensaje** formado por dos campos: el primero es la lista de objetos Cine (con su nombre y distancia) de acuerdo con los criterios de búsqueda solicitados (por ejemplo {("abc",34.05), ("Yelmo",5.00)}). Si no se han encontrado cines, o no se ha realizado la búsqueda, este campo tendrá el valor null. El segundo campo del objeto Mensaje tiene como valor por defecto una cadena de caracteres vacía .En caso de que no se encuentre ningún cine que cumpla los criterios de búsqueda, el segundo campo del mensaje de salida contendrá el texto "no se ha encontrado nada". Si no se realizado la búsqueda, el segundo campo del mensaje de salida contendrá un texto formado por el mensaje de error generado.

Suponiendo que aplicamos el método de particiones equivalentes, ¿**Cuántos casos de prueba** necesitamos para garantizar que nuestras pruebas son eficientes y efectivas? Es **imprescindible** que apliques el método tal y como hemos explicado en clase. (2p)

Examen PPSS (Cód. 34027) (18 de marzo de 2024)
Departamento Ciencia de la Computación e Inteligencia Artificial

3. Disponemos del siguiente código del método ***calculaPrecio()***., el cual calcula el precio de alquiler de un coche durante un determinado número de días, a partir de una fecha que se pasa también como parámetro. El precio de alquiler diario depende de dicha fecha y se incrementará en 25 euros si el día es festivo. La comprobación de si es festivo (método *es_festivo()*) o no puede lanzar una excepción, en cuyo caso, ese día no se contabilizará en el precio."

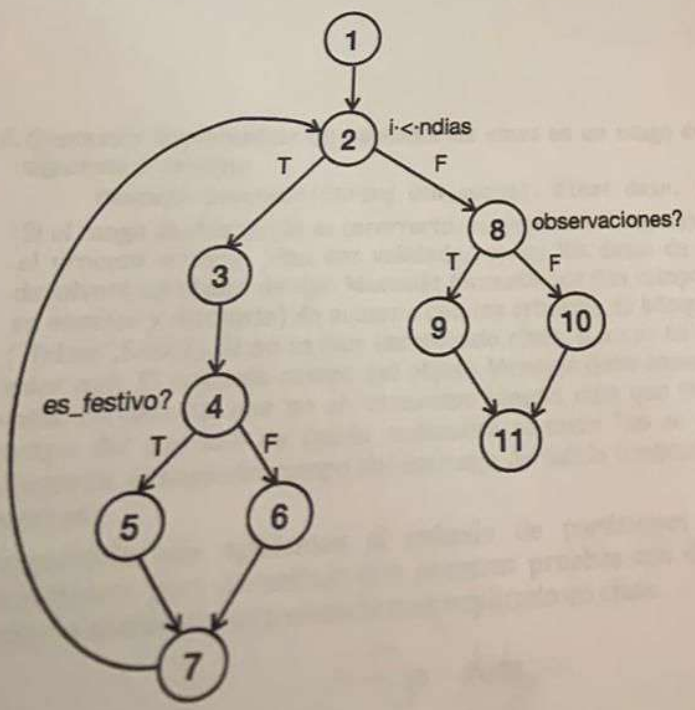
```
1. public class AlquilaCoches {
2.   protected Calendario calendario = new Calendario();
3.
4.   public Ticket calculaPrecio(LocalDate inicio, int ndias {
5.     Ticket ticket = new Ticket();
6.     float precioDia,precioTotal =0.0f;
7.
8.     String observaciones = "";
9.     IService servicio = new Servicio();
10.    precioDia = servicio.consultaPrecio(inicio);
11.    for (int i=0; i<ndias;i++) {
12.      LocalDate otroDia = inicio.plusDays((long)i);
13.
14.      try {
15.        if (calendario.es_festivo(otroDia)) {
16.            precioTotal += precioDia+ 25;
17.        }  else {
18.            precioTotal += precioDia;
19.        }
20.      } catch (CalendarioException ex) {
21.         observaciones += "dia "+otroDia.toString()+"
   no contabilizado ;";
22.      }
23.    }
24.
25.    if (observaciones.isEmpty()) {
26.        ticket.setObservaciones(null);
27.    } else {
28.        ticket.setObservaciones(observaciones);
29.    }
30.
31.    ticket.setPrecio_final(precioTotal);
32.    return ticket;
33.  }
34.}
```

```
public class Ticket {
  private float precio_final;
  private String observaciones;

  //getters y setters
}
```

El método devuelve un ticket con el precio total calculado y unas observaciones, que serán null si no se ha producido error al comprobar si un día es festivo, o un mensaje indicando los días en los que se han producido errores. Por ejemplo si los días X e Y provocan excepciones, el mensaje sería: "día X no contabilizado;día Y no contabilizado

Proporcionamos un CFG para el método anterior, el cual está incompleto. Complétalo sobre el propio grafo proporcionado. Diseña una tabla de casos de prueba aplicando el método del camino básico. (1,5p)



Examen PPSS (Cód. 34027) (18 de marzo de 2024)
Departamento Ciencia de la Computación e Inteligencia Artificial

4. Disponemos de la siguiente implementación del método Cadenas.subCadenaAleatoria() que a partir de un String y un número, devuelve otro String de longitud dada por el número de entrada, y cuyos caracteres se obtienen de forma aleatoria seleccionándolos del String de entrada.

Es decir, si la cadena de entrada es "Ejercicio" y el número es 4, el método devuelve una subcadena de longitud 4, seleccionando 4 caracteres aleatoriamente de la cadena de entrada.

```
public String subCadenaAleatoria(String cadena, int lon_subcadena)
                                                  throws LongitudException {

   int indice, longitud;
   Random random = new Random();
   longitud = cadena.length();
   if (lon_subcadena > longitud) {
      throw new LongitudException();
   }
   StringBuilder buffer = new StringBuilder();
   for (int i=0; i < lon_subcadena; i++) {
     indice = random.nextInt(longitud);
     buffer.append(cadena.charAt(indice));
   }
   return buffer.toString();
}
```

Implementa un driver para automatizar el siguiente caso de prueba usando verificación basada en el comportamiento. Debes implementar toda la clase. Si necesistas refactorizar, debes tener en cuenta que no puedes añadir ninguna clase adicional en producción ni alterar en modo alguno la invocación a nuestra unidad desde otras unidades, ni tampoco podemos añadir ningún atributo en la clase de nuestra SUT.

Para implementar el driver debes seguir todas las **normas** explicadas en clase. Es **imprescindible** que indiques los pasos que vas siguiendo. Debes tener en cuenta que la tabla tiene más casos de prueba además del mostrado

(2,5p)

	cadena	lon_subcadena	indices aleatorios	Resultado esperado
C1	"Ejercicio"	4	{3,5,4,8}	"rico"

5. Disponemos de la siguiente implementación del método MatriculaAlumno.validaAsignaturas(), que recibe el dni de un alumno y una lista de códigos de asignatura de las que se quiere matricular, y devuelve un justificante de la matricula, que incluye el dni, la lista de asignaturas de las que se puede matricular y la lista de asignaturas de las que no se puede matricular (por haber sido ya cursada o por ser un código incorrecto).

La comprobación de si un alumno se puede matricular en una asignatura se lleva a cabo en el método Operacion.compruebaMatricula() que devuelve las excepciones AsignaturaIncorrectaException o AsignaturaCursadaException, cuando no se puede matricular de una determinada asignatura. Siempre se comprueban todas las asignaturas de la lista, independientemente de que algunas no sean válidas.

```
public class MatriculaAlumno {
   protected Operacion getOperacion() {
      return new Operacion();
   }

   public JustificanteMatricula validaAsignaturas(String dni, String[] asignaturas) {
     JustificanteMatricula justificante = new JustificanteMatricula();
     ArrayList<String> validas = new ArrayList<>();
     ArrayList<String> listaErrores = new ArrayList<>();

     Operacion op = getOperacion();
     for (String asignatura: asignaturas) {
        try {
           op.compruebaMatricula(dni, asignatura);
           validas.add(asignatura);
        } catch (AsignaturaIncorrectaException ex) {
           listaErrores.add("Asignatura " + asignatura + " no existe");
        } catch (AsignaturaCursadaException ex) {
           listaErrores.add("Asignatura " + asignatura + " ya cursada");
        }
     }

     justificante.setDni(dni);
     justificante.setAsignaturas(validas);
     justificante.setErrorres(listaErrores);

     return justificante;
   }
}
```

```
public class JustificanteMatricula {
  private String dni;
  private ArrayList<String> asignaturas;
  private ArrayList<String> errores;
  //getters y setters
}
```

Implementa un driver para automatizar el siguiente caso de prueba usando verificación basada en el estado. Tienes que implementar toda la clase del driver, así como cualquier el código adicional para ejecutarlo. Si necesistas refactorizar, debes tener en cuenta que no puedes añadir ninguna clase adicional en producción ni ningún atributo. No puedes usar EasyMock.

Para implementar el driver debes seguir todas las **normas** explicadas en clase. Es **imprescindible** que indiques los pasos que vas siguiendo. También debes tener en cuenta que la tabla tiene más casos de prueba además del mostrado. (2,5p)

	dni	asignaturas	JustificanteMatricula (dni, asignaturas, errores)
C1	"00000000T"	{"MD", "ZZ", "FBD","P1"}	("00000000T", {"MD", "FBD"}, {"Asignatura ZZ no existe", "Asignatura P1 ya cursada"}

Suponemos que el alumno con dni "0000000T" ya ha cursado las asignaturas "P1", "FC" y "FFI" ,y que los únicos códigos de asignaturas que no existen son "YYY" y "ZZ".