Convocatoria Extraordinaria

Análisis y Diseño de Software (2015/2016)

Responde a cada apartado en hojas separadas

Apartado 1 (2,5 puntos)

Debes diseñar una aplicación para **realizar automáticamente el análisis de calidad de preguntas creadas por los profesores de un centro educativo.**

Los *profesores* (identificados por su nombre) se encargan de crear *preguntas* y añadirlas a la aplicación. Así mismo, para cada profesor, la aplicación almacena los *informes* que los *analizadores* de preguntas van generando sobre las preguntas creadas por ese profesor. El profesor puede borrar cualquiera de esos informes una vez que lo haya leído. Un profesor, tras leer el informe sobre una pregunta, puede mejorarla cambiando cualquier dato de ella, lo cual hace que automáticamente la pregunta modificada solicite un nuevo análisis completo a todos los analizadores que dicha pregunta tenga asignados previamente. Dicho análisis realizado por cada analizador asignado, genera un nuevo informe para el profesor.

Una *pregunta* se caracteriza por el texto de la pregunta propiamente dicha, el epígrafe del temario (un número entero) con el que está asociada la pregunta, y el profesor que la creó. Además, una vez creada, a la pregunta se le pueden añadir (uno a uno) un número variable de analizadores, y también se le pueden eliminar analizadores.

Los *analizadores* de preguntas son objetos cuyo principal objetivo es analizar las preguntas que los profesores van añadiendo a la aplicación. Cada tipo concreto de analizador se distingue de otros por el criterio que utiliza para analizar la pregunta. Por ejemplo, un *analizador gramatical* se centra en la corrección y complejidad sintáctica de las frases. Otro ejemplo es el *analizador de vocabulario* que basa su análisis en el uso de términos poco recomendables (vulgares, extremadamente raros o incorrectos). También tenemos un tipo de *analizador conceptual* al comparar la pregunta con los conceptos básicos de su epígrafe en el temario. Actualmente se usan esos tres tipos de analizadores, pero la aplicación debe diseñarse para admitir fácilmente otros tipos de analizadores cuyo comportamiento difiere sólo en la forma en que cada uno realiza su análisis de la pregunta.

Cada analizador se crea con su pregunta asignada para analizar y recibe un identificador numérico generado automáticamente por la aplicación. Cuando termina de analizar su pregunta, el analizador genera un informe, con el análisis textual y la pregunta analizada, que añade a la colección de informes correspondiente al profesor que creó esa pregunta.

Se pide:

(a) Representar el diagrama de clases en UML para la aplicación descrita, sin las clases relacionadas con una posible interfaz de usuario. Respecto a los constructores, los métodos *getters* o *setters*, y otros métodos, **debes incluir los que intervengan directamente en la funcionalidad descrita arriba** [1,7 puntos]

(b) Para cada uno de los siguientes requisitos de la aplicación, describe en detalle (con su nombre, argumentos, tipo de resultado, y demás características) el método correspondiente y justifica su inclusión en la clase más apropiada para contenerlo:

1. Añadir una pregunta [0,1 punto]
2. Solicitar el análisis completo de una pregunta [0,1 puntos]
3. Añadir un nuevo informe de análisis sobre una pregunta [0,1 puntos]

(c) Implementa, en pseudocódigo o código java, el método para solicitar el análisis completo de una pregunta. [0,2 puntos]

(d) ¿Qué patrón de diseño has utilizado y cómo participan en él las clases de tu diseño? [0,3 puntos]

Nota: No es necesario detallar los algoritmos específicos para los diversos analizadores. En el enunciado se han mencionado solamente para ilustrar, de forma general, diversas alternativas posibles para analizar preguntas.

Convocatoria Extraordinaria

Análisis y Diseño de Software (2015/2016)

Responde a cada apartado en hojas separadas

Apartado 2 (2,5 puntos)

Se quiere construir una aplicación para gestionar las solicitudes informáticas de los usuarios de una empresa. Las solicitudes podrán ser de dos tipos: reparación de una avería de hardware, o instalación de software. Todas las solicitudes tienen una descripción y una prioridad. La prioridad es un entero entre 0 y 5, siendo 5 la prioridad más alta y 0 la más baja. En el caso de instalación de software, el valor máximo de la prioridad será 3, y en caso de no especificarse, se tomará como 0. Todos los tipos de solicitud deberán responder a un método `esUrgente`, que indicará si la solicitud se considera urgente. En el caso de instalaciones software esto será si su prioridad es la máxima (es decir, 3), mientras que las averías hardware serán urgentes si además de tener su prioridad máxima (es decir, 5) tienen asignado un técnico.

Las averías hardware estarán descritas además por el tipo de hardware (por ejemplo "Monitor HP 2311x"). La solicitud de instalación de software por el software a instalar (por ejemplo "JDK 1.8"). Todas las solicitudes pueden resolverse, y se llevará la cuenta del número de solicitudes reparadas (método `numResueltas`). Ten en cuenta que si una solicitud ya resuelta se trata de resolver de nuevo (llamando al método `resolver`), no se ha de contar dos veces como resuelta. Las averías hardware necesitan de la asignación de un técnico para poder resolverse. Si no tiene técnico asignado, el método `resolver` no tiene ningún efecto.

Se pide: Completa el siguiente programa con las clases necesarias para que produzca la salida de más abajo. Se te da el código de la clase `Tecnico`, que no debes implementar. Debes utilizar adecuadamente los conceptos de orientación a objetos en tu código de la mejor manera posible, y asegurarte de que el programa sea fácilmente extensible y modificable. Asegúrate también de realizar un control básico de errores respecto a la prioridad asignada a cada solicitud (por ejemplo, asignando un valor 0 si se le trata de dar un valor negativo). No incluyas getters o setters en tus clases si no son necesarios.

```
class Tecnico {
        private String nombre;
        public Tecnico(String n) { this.nombre = n; }
}
public class Ejercicio2 {
  public static void main(String[] args) {
        Solicitud [] solicitudes = {
                new InstalacionSoftware("Necesito instalar Java", "JDK 1.8"), // prioridad 0
                new AveriaHardware("El monitor no enciende", 5, "Monitor HP 2311x"),
                new InstalacionSoftware("Necesito migrar a Windows 10", 3, "Windows 10")};
        Tecnico tecnico = new Tecnico("Antonio Lopez");
        for (Solicitud s : solicitudes)
                System.out.println(s);

        System.out.println("Número de solicitudes resueltas: "+Solicitud.numResueltas());
        System.out.println("Urgente? "+solicitudes[0].esUrgente()); // no urgente porque por defecto prioridad 0
        solicitudes[0].resolver(); // podemos resolver porque es software y no necesita técnico

        System.out.println("Número de solicitudes resueltas: "+Solicitud.numResueltas()); // 1 solicitud resuelta
        System.out.println("Urgente? "+solicitudes[1].esUrgente()); // no urgente porque no tiene técnico
        solicitudes[1].resolver(); // no se resolverá porque no tiene técnico
        System.out.println("Número de solicitudes resueltas: "+Solicitud.numResueltas()); // 1 solicitud resuelta
        ((AveriaHardware)solicitudes[1]).asignarTecnico(tecnico);
        System.out.println("Urgente? "+solicitudes[1].esUrgente()); // Es urgente al tener técnico y prioridad 5
        solicitudes[1].resolver(); // Como tiene técnico sí se resuelve

        System.out.println("Número de solicitudes resueltas: "+Solicitud.numResueltas());//2 solicitudes resueltas
  }
}
```

Convocatoria Extraordinaria

Análisis y Diseño de Software (2015/2016)

Responde a cada apartado en hojas separadas

Salida esperada:

```
Instalacion SW:
 - JDK 1.8
 - descripcion: Necesito instalar Java
 - prioridad: 0
 - no resuelta
Avería HW:
 - Marca: Monitor HP 2311x
 - descripcion: El monitor no enciende
 - prioridad: 5
 - no resuelta
Instalacion SW:
 - Windows 10
 - descripcion: Necesito migrar a Windows 10
 - prioridad: 3
 - no resuelta
Número de solicitudes resueltas: 0
Urgente? false
Número de solicitudes resueltas: 1
Urgente? false
Número de solicitudes resueltas: 1
Urgente? true
Número de solicitudes resueltas: 2
```

Convocatoria Extraordinaria

Análisis y Diseño de Software (2015/2016)

Responde a cada apartado en hojas separadas

Apartado 3 (2,5 puntos)

Se quiere gestionar la asignación de *tareas de programación* a los distintos programadores que participan en un proyecto. Al crear un proyecto se indican los nombres de los programadores que participan en él, y sólo a esos se les podrá asignar tareas. Se lanzará una excepción si se intenta asignar una tarea a un programador desconocido para ese proyecto. A cada programador del proyecto se le puede asignar un número variable de tareas sin duplicados. Cada tarea de programación se identifica mediante un nombre de paquete, el tipo de *módulo* a programar (que puede ser interfaz, enumerado, clase o excepción), y el nombre del módulo. En un mismo paquete no pueden existir dos módulos con el mismo nombre (aunque sean de distinto tipo), es decir, si se intenta asignar a un programador una tarea con el mismo nombre de paquete y módulo que otra que ya tiene asignada ese programador, se ignora ese intento de asignación y se imprime un aviso con la tarea duplicada no añadida.

Se pide: Completa el siguiente programa con las clases, enumerados y excepciones necesarias para que produzca la salida de más abajo, donde las asignaciones del proyecto se imprimen ordenadas, en primer lugar, por nombre de programador, y para cada programador sus tareas asignadas aparecen ordenadas por nombre de paquete y dentro de cada paquete por nombre de módulo. Además, nótese que al imprimir cada tarea, la última letra indica el tipo de módulo: I para interfaz, C para clase, X para excepción, y E para enumerado.

```
public class Ejercicio3 {
  public static void main(String[] args) {
    Proyecto p = new Proyecto("Javier", "Ana", "Luis");
    try {
      p.asignaTarea("Luis", new Tarea("paqueteA", Tarea.Tipo.CLASS, "Arbol"));
      p.asignaTarea("Luis", new Tarea("paqueteA", Tarea.Tipo.INTERFACE, "Arbol")); // tarea duplicada
      p.asignaTarea("Ana",  new Tarea("paqueteB", Tarea.Tipo.EXCEPTION, "Error"));
      p.asignaTarea("Ana",  new Tarea("paqueteB", Tarea.Tipo.CLASS, "Data"));
      p.asignaTarea("Ana",  new Tarea("paqueteA", Tarea.Tipo.CLASS, "Hoja"));
      p.asignaTarea("Jose", new Tarea("paqueteA", Tarea.Tipo.CLASS, "Test"));
      p.asignaTarea("Luis", new Tarea("paqueteC", Tarea.Tipo.CLASS, "Raiz"));  // no se ejecuta
    } catch (ProgramadorDesconocido e) {
      System.out.println(e);
    }
    System.out.println("Proyecto: " +  p);
  }
}
```

Salida esperada:

```
Duplicado no añadido: Luis:paqueteA.Arbol:I
Programador desconocido: Jose
Proyecto: {Ana=[paqueteA.Hoja:C, paqueteB.Data:C, paqueteB.Error:X], Luis=[paqueteA.Arbol:C]}
```

Convocatoria Extraordinaria

Análisis y Diseño de Software (2015/2016)

Responde a cada apartado en hojas separadas

Apartado 4 (2,5 puntos)

Se quiere implementar una calculadora sencilla, que permita interpretar expresiones aritméticas sin tener en cuenta paréntesis ni precedencia de operadores. Las expresiones tendran la forma "*valor1 op1 valor2 op2 valor3 op3… valorN*".
La calculadora ha de permitir añadir nuevos operadores de forma dinámica, con el método "addOperator", mediante un nombre y un objeto de tipo Operator.

Para simplificar el diseño nos dan las siguientes interfaces:

```
@FunctionalInterface
public interface Operator {
   double apply(double a, double b);
}

public interface Calculator {
   Calculator addOperator(String name, Operator op);
   double calc(String operation, double left, double right);

   default double calc(String exp){
       //Expresión aritmética simple, espacios como separador y sin precedencia
       String[] values= exp.split("\\s");
       double result=Double.parseDouble(values[0]);
       for (int i=1; i<values.length; i+=2)
           result= calc(values[i], result, Double.parseDouble(values[i+1]));
       return result;
   }
}
```

En un futuro es posible que queramos implementar una calculadora más compleja, y que admita por ejemplo paréntesis o precedencia de operadores. Para conseguir esto, en lugar de crear directamente la calculadora, usaremos la clase CalculatorMaker. El método de clase CalculatorMaker.getInstance() nos dará una instancia que nos permite crear calculadoras, por ahora sencillas, mediante el método makeCalculator().
El método "addBasicOperators" añade a una calculadora, previamente creada con el mismo CalculatorMaker, los operadores básicos. En nuestro caso, la suma (+), resta (-), multiplicación (*) y división (/).

Queremos poder evaluar expresiones aritméticas sencillas, de forma que el siguiente fragmento de código:

```
public class BasicCalc {
   public static void main(String ...args){
       CalculatorMaker maker=CalculatorMaker.getInstance();

       //addBasicOperators añade los operadores aritméticos básicos a la calculadora:
       //  suma (+), resta(-), multiplicación (*) y división (/)
       Calculator calc=maker.addBasicOperators(maker.makeCalculator());
       //Añadimos manualmente x elevado a y, usando el símbolo ^
       calc.addOperator("^", Math::pow);
       //Calculamos la expresión, separada por espacios, y sin precedencia de operadores
       System.out.println(calc.calc("10 + 8 / 2 - 4 * 0.1")); //Resultado 0.5
       System.out.println(calc.calc("8 ^ 2 / 2 * 0.25"));     //Resultado 8.0
   }
}
```

Produzca la salida siguiente al ejecutar el método main:

```
0.5
8.0
```

Se pide:

(a) Utilizando **patrones** de diseño, y expresiones **Lambda** (siempre que sea posible), crear la clase CalculatorMaker, y las clases o interfaces adicionales necesarias. [2 puntos]

(b) Indica qué **patrón o patrones** de diseño has utilizado, **y** qué **roles** en el patrón juegan las clases e interfaces que has implementado. [0.5 puntos]