Convocatoria extraordinaria de septiembre

Nombre: 5-09-2014

I- Cada respuesta incorrecta restará $1/2$ de la puntuación obtenida por cada respuesta correcta.

1. (**0,5**) Seleccionar la única respuesta correcta a la siguiente cuestión sobre el concepto de *patrón de diseño* en el desarrollo de software:
   (a) ✓ Las clases de un patrón de diseño se pueden modificar para adaptarse a las condiciones particulares de un problema que se repite.
   (b) Un patrón de diseño es básicamente un esquema algorítmico (como "divide y vencerás", "programación dinámica", etc.) pero expresado con clases
   (c) Cada patrón es una estructura de clases fija que se adapta al desarrollo de una determinada aplicación o sistema–software.
   (d) Un patrón propone una solución a un tipo de problema para cualquier plataforma de ejecución.
2. (**0,5**)Seleccionar la única respuesta correcta a la siguiente cuestión sobre las diferencias entre patrones de diseño y marcos de trabajo (*frameworks*) en sistemas software:
   (a) Los patrones de diseño son menos abstractos que los marcos de trabajo.
   (b) Los marcos de trabajo son un conjunto de clases que se han de utilizar sin modificación.
   (c) ✓ Un marco de trabajo es el esqueleto de una aplicación que ha de ser adaptado a las necesidades concretas por el programador de la aplicación.
   (d) Un patrón de diseño puede estar compuesto por varios marcos de trabajo.
3. (**0,5**) Indicar para cada uno de los siguientes ejemplos de sistemas a qué tipo de sistemas (S, P o E) pertenece:
   (a) Un sistema de reservas de vuelos y hoteles. E
   (b) Un sistema de prevención sísmica basado en las lecturas de una red de sismógrafos. P
   (c) Un programa para jugar al ajedrez. P
   (d) Un sistema operativo. P
   (e) Un sistema para control automático de vuelo de una aeronave. P
   (f) Un sistema informático para modelar la Economía Mundial.E
4. (**0,5**)Seleccionar cuál de los elementos siguientes no tiene influencia importante en los costos de mantenimiento de un sistema software :
   (a) Tipo de aplicación que hay que mantener (transformacional, reactiva, tiempo–real, etc.)
   (b) Tipo de sistema software: S, P o E
   (c) Duración prevista del ciclo de mantenimiento y evolución
   (d) ✓ Sistema operativo y software de red en que está instalado el sistema
   (e) Calidad de la documentación del software
   (f) Diseño del sistema

5. (**0,5**) De las siguientes afirmaciones relacionadas con la aplicación de MDA al desarrollo de software, seleccionar la única incorrecta:
   (a) De acuerdo con el modelo de desarrollo por transformación de modelos MDA, pueden existir desarrollos de un sistema software en los que el mismo tipo de diagrama (Clases, Estados-UML, Secuencia, Actividad, etc.) pertenece a distintos tipos de modelos intermedios (PIM, PSM, etc.)
   (b) Un mismo modelo expresado en una notación determinada puede ser considerado un modelo MDA diferente dependiendo de la plataforma objetivo final.
   (c) ✓ Si realizamos pruebas unitarias de software necesitamos tener desarrollados todos los componentes del sistema antes de comenzar a probarlo.
   (d) No se puede, en general, obtener automáticamente un modelo PIM a partir de un modelo CIM de MDA.
   (e) Las pruebas que hay que desarrollar para validar un sistema software son: unitarias, integración, validación y del sistema.

6. (**0,5**)Seleccionar la única alternativa correcta sobre evolución del software:
   (a) El mantenimiento del software posee un ciclo de proceso independiente del ciclo de desarrollo del software
   (b) El modelo predictivo de Belady–Lehman expresa la relación entre el coste en desarrollar el sistema (p), la complejidad del software (c) y la buena documentación (d) del mismo, que se resume con la siguiente ecuación $M = p + K \times c - d$. La constante empírica K se calcula después de la instalación y configuración para una plataforma software concreta
   (c) El factor (SU) de valoración de la comprensión del código para el correcto mantenimiento del software según el modelo COCOMO depende fundamentalmente de la alta cohesión y bajo acoplamiento entre componentes (módulos, clases, paquetes) del sistema
   (d) ✓ Los mantenimientos perfectivo y correctivo del software *deterioran* a un sistema software

7. (**0,5**)¿Cuáles de los siguientes pasos del proceso sistemático de prueba de una aplicación Web no pertenecen al protocolo explicado?:
   (a) Revisar el modelo de contenidos
   (b) Comprobar el modelo de interfaz respecto de los casos de uso
   (c) ✓ Detectar errores en el modelo de requisitos
   (d) Ejercitar la navegación del modelo de interfaz de usuario
   (e) Detectar errores de navegación en el modelo de interfaz
   (f) Pruebas unitarias de componentes Web
   (g) Revisar la facilidad de navegación a través de toda la arquitectura software
   (h) ✓ Buscar enlaces rotos antes de instalar las páginas
   (i) Comprobar compatibilidad con plataformas y sus configuraciones
   (j) Pruebas de seguridad, robustez, stress, carga y rendimiento

II- Ejercicios sobre la teoría impartida

1. (**2,5**) Utilizar el patrón de diseño *factoría abstracta* para realizar el diagrama de clases de una aplicación que simula 2 carreras de bicicletas: montaña y carretera:

   - `Carrera` es una interfaz de Java, que representa a la *factoría* del patrón, que declara los métodos abstractos de fabricación: `crearCuadro()`, `crearManillar()`, `crearRuedas()`.
   - `CarreraMontaña` y `CarreraCarretera` son las clases factoría específicas o concretas que se utilizan para crear los objetos específicos: *cuadro_motaña*, *cuadro_carretera*, `manillar_montaña`, `manillar_carretera`, `ruedas_montaña` y `ruedas_carretera`
   - La clase genérica del patrón que se pide se llamará `Bicicleta` (clase abstracta) e incluirá los métodos:
     - Método constructor: `+Bicicleta (TC b);`
     - `public enum TC{ MONTANA,CARRETERA}`
     - el método `-crearBicicleta()`, que se encarga de construir una bicicleta como una agregación de objetos de `Cuadro`, `Manillar` y `Ruedas`
     - Esta clase usa los métodos fabricación declarados en la interfaz factoría abstracta `Carrera` del patrón.
   - Las clases específicas `CuadroMontaña`, `ManillarMotaña` y `RuedasMontaña`; y `CuadroCarretera`, `ManillarCarretera` y `RuedasCarretera` son todas ellas subclases de las clases genéricas: `Manillar`, `Cuadro` y `Ruedas`

   Por último, el programa principal incluye las clases `Montaña` y `Carretera` que se encargan de obtener las bicicletas necesarias para montar una carrera de montaña o de carretera, respectivamente.

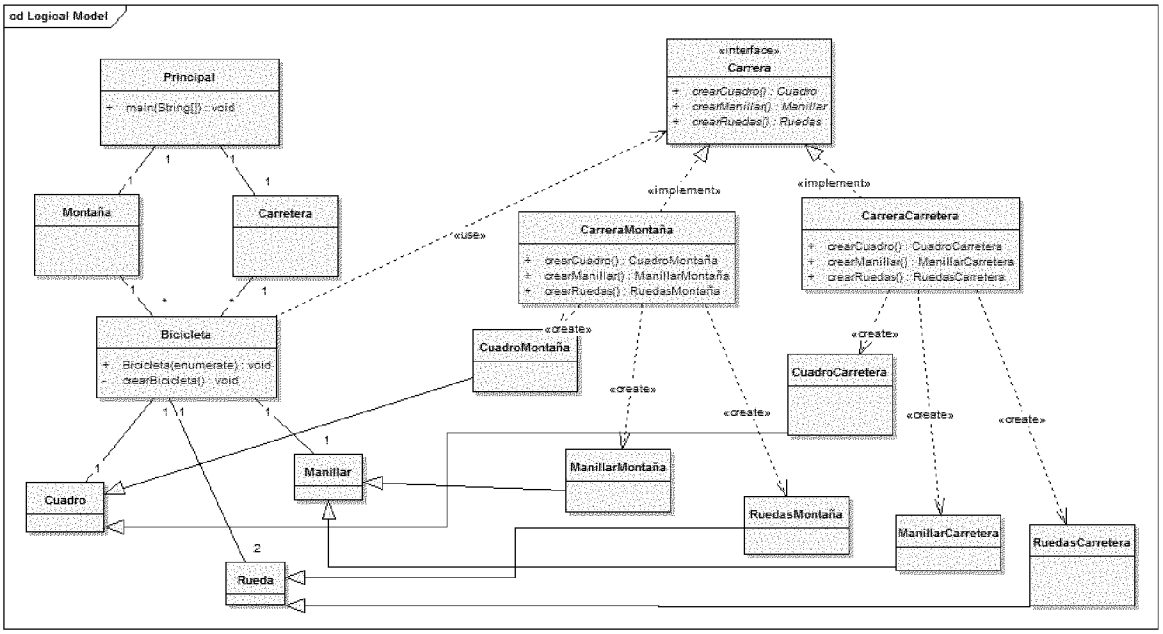


Figura 1: Diagrama de clases utilizando el patrón *factoría abstracta*

2. (**1,5**) Para el siguiente código, determinar su número ciclomático utilizando los tres métodos diferentes estudiados en clase: (a) construir el grafo de flujo y contar nodos y arcos; (b) obteniendo las sentencias condicionales y (c) regiones espaciales en que se divide al plano.

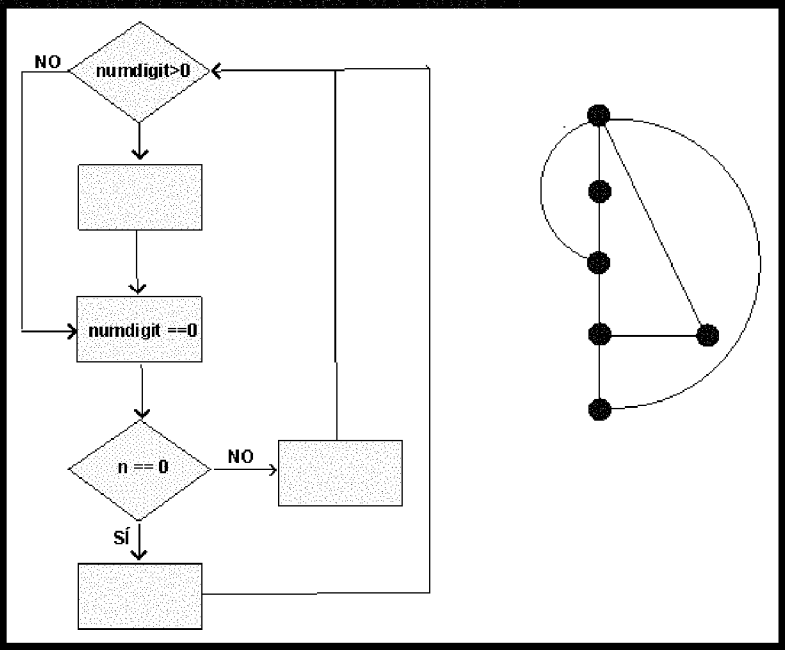
```
public void drawScore( int n
  while numdigitos
    score numdigitos  erase

  numdigitos
  if n
     //liberar memoria del objeto
    delete score numdigitos
    score numdigitos  new Displayable digitos
    score numdigitos  move Point      numdigitos
    score numdigitos  draw
    numdigitos

  while n
    int resto  n
    delete score numdigitos
    score numdigitos  new Displayable digitos resto
    score numdigitos  move Point      numdigitos
    score numdigitos  draw
    n  n
    numdigitos
```

Solución:

(a) Número de nodes (N) = 6; Número de arcos (A) = 8; Número McCabe = A – N + 2 = 4

(b) 1 sentencia if y 2 while; Número McCabe = sentencias condicionales + 1 = 4

(c) El plano se divide en 4 subregiones (ver figura 2)



3. (**1,0**) Una expresión de un lenguaje de programación se puede descomponer en una jerarquía de subexpresiones. Por ejemplo, `(a/(b+c))` es una de las subexpresiones contenidas en la expresión terminal: `(a/(b+c))+(b-func(d)*(e/(f+g)))`. Utilizando uno de los patrones `Jerarquía General` o `Composite`, crear un diagrama de clases para representar expresiones sintácticamente correctas para un lenguaje de programación en particular, por ejemplo, Java.

Solución:

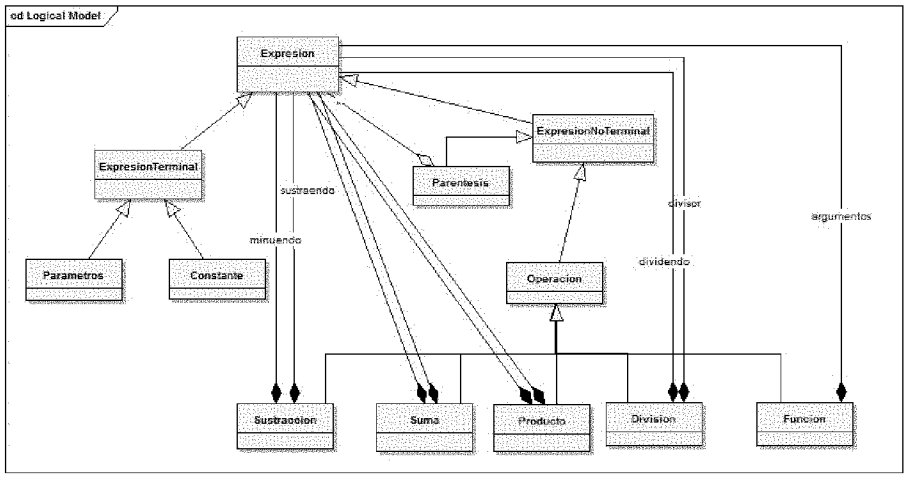


Figura 3: Diagrama de clases para expresiones Java correctas sintácticamente

IV- Supuesto Práctico

1. (**1,5**) Considerando el siguiente programa `Java` cuando se ejecuta: (a)sin argumentos, Salida:"Debes especificar un argumento". En este caso se produce una excepción que es capturada en el programa principal `main(...)`, en la línea (101).
(b) con `'-'` como argumento Salida:"El valor numerico de -MiOtraExcepcion:la cadena solo contiene '-' Manejado en el punto 2". En este caso se produce una excepción en la línea (50) del método estático `c(String s, int base)` que es capturada en la línea (64) del método `b(String s, int base)`.
y (c) con :

   - `'675'`: Salida:"El valor numerico de 675 en base 10 es 675". En este caso no se produce ninguna excepción, ya que se ejecutan las instrucciones(60–62) del método `b(String s, int base)`, que llama al `c(String s, int base)`; después de ejecutar el bloque (21–41) devolverá el valor de `resultado` tras asignar correctamente el signo,
   - `'-67'`, Idem que el anterior. Después de cambiar el signo de `resultado`, en las líneas (60) y (62) se escribirá: "El valor numerico de -67 en base 10 es -67"
   - `'A'` Salida:"El valor numérico de A MiExcepcion:el caracter no es un dígito:'0'..'9' Manejado en el punto 1". En este caso se produce una excepción en la línea (30) del método estático `c(String s, int base)` que es capturada en la línea (79) del método `a(String s, int base)`, escribiéndose la parte final del mensaje con las instrucciones del bloque (83–85). y

- `'7C8'`. Se trata de un caso similar al anterior, cuando se lea el carácter 'C' de la hilera de entrada, se levantará `MiExcepcion` en la línea (38) del método estático `c(String s, int base)`; continuando el resto de la ejecución como en el caso anterior.

Indicar la salida que producirá en cada uno de los casos (a)-(c) anteriores y explicar cómo funciona la gestión de las excepciones que se han implementado (indicar el orden de ejecución de las instrucciones del programa cuando se produzca cada una de las excepciones programadas).

```
1  class MiExcepcion extends Exception{
2    public MiExcepcion(){ super();}
3    public MiExcepcion(String s){ super(s);}
4  }
5  class MiOtraExcepcion extends Exception{
6    public MiOtraExcepcion(){ super();}
7    public MiOtraExcepcion(String s){ super(s);}
8  }
9  class MiSubExcepcion extends MiExcepcion{
10   public MiSubExcepcion(){ super();}
11   public MiSubExcepcion(String s){ super(s);}
12 }
13
14 public class ParseIntegerDemo {
15         public static int c(String s, int base) throws MiExcepcion, MiOtraExcepcion{
16                 int resultado = 0;
17                 boolean negativo = false;
18                 int i = 0;
19                 int digito;
20                 int max=s.length();
21
22                 if (max > 0) {
23                     if (s.charAt(0) == '-') {
24                           negativo = true;
25                           i++;
26                     }
27                     if (i < max) {
28                           digito = Character.digit(s.charAt(i++),base);
29                           if (digito < 0) {
30                           throw new MiExcepcion("el caracter no es un digito:'0'..'9'");
31                           } else {
32                               resultado = -digito;
33                           }
34                     }
35                     while (i < max) {
36                           digito = Character.digit(s.charAt(i++),base);
37                           if (digito < 0) {
38                             throw new MiExcepcion("el caracter no es un digito:'0'..'9'");
39                           }
40                           resultado *= base;
41                           resultado -= digito;
42                     }
43                 } else {
44                   throw new MiSubExcepcion("la cadena de entrada 's' esta vacia");
45                   }
46                 if (negativo) {
47                     if (i > 1) {
48                           return resultado;
49                     } else {
50                           throw new MiOtraExcepcion("la cadena solo contiene '-'");
51                       }
52                 } else {
53                      return -resultado;
54                   }
55                 }
56
57         public static void b(String s, int base) throws MiExcepcion{
58                 int resultado;
59                 try{
60                   System.out.print("El valor numerico de "+s);
61                   resultado= c(s,base);
```

```
62                      System.out.print("en␣base␣10␣es␣"+resultado);
63                    }
64                    catch(MiOtraExcepcion e){   //Punto 2
65                      System.out.println("MiOtraExcepcion:"+e.getMessage());
66                      System.out.println("Manejado␣en␣el␣punto␣2");
67                    }
68                    finally{
69                      System.out.print("\n");
70                    }
71
72
73  }
74
75          public static void a(String s, int base){
76                    try{
77                      b(s,base);
78                    }
79                    catch(MiExcepcion e){//Punto 1
80                      if(e instanceof MiSubExcepcion)
81                        System.out.print("MiSubExcepcion:");
82                      else
83                        System.out.print("MiExcepcion:");
84                      System.out.println(e.getMessage());
85                      System.out.println("Manejado␣en␣el␣punto␣1");
86                    }
87                  }
88
89
90           public static void main(String[] args) throws MiExcepcion{
91                      int base=10;
92                      int max;
93                      String s;
94                      try{
95                         s=args[0];
96                      max= s.length();
97                         if (max == 0) {
98                            throw new NumberFormatException("null");
99                          }
100                     }
101                     catch(ArrayIndexOutOfBoundsException e){
102                        System.out.println("Debes␣especificar␣un␣argumento");
103                        return;
104                     }
105                 a(s,base);
106   }
107 }
```