Examen final

Nombre:

22-06-2015

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

1. (**0,5**) Seleccionar la única respuesta incorrecta a la siguiente cuestión sobre el concepto de *patrón de diseño* en el desarrollo de software.
   (a) Sirve para resolver un problema concreto en un determinado contexto.
   (b) La factibilidad de la solución que se propone ha de estar comprobada.
   (c) ✓ Un patrón de diseño no se cambia, sino que se adapta mediante extensión al sistema–software concreto.
   (d) Generan indirectamente la solución al problema que se proponen resolver.
2. (**0,5**)Seleccionar la única respuesta correcta a la siguiente cuestión sobre el patrón de diseño *Abstracción / Caso*. Suponer que se va a crear un catálogo de piezas de coche:
   (a) Crear cada pieza disponible como una instancia de subclase de Pieza ={Nombre, Fabricante, Precio, Código-Almacén, Fecha-Entrada, Disponibles } con su código de barras.
   (b) Crear clases-Pieza específicas y diferentes por cada fabricante, que a su vez son subclases de la clase Pieza.
   (c) Crear una clase Pieza como una agregación de todas las piezas disponibles, que tienen como atributo sólo su código de barras.
   (d) ✓ Mediante una asociación, la clase Pieza se materializa en las piezas de ese tipo que quedan en el almacén.
3. (**0,5**) Seleccionar la única respuesta correcta a la siguiente cuestión sobre el patrón de diseño *Jerarquía General*. Suponer que se va a crear un catálogo de piezas de coche que pueden tener partes. La utilización más correcta del patrón anterior supone que:
   (a) Para representar la jerarquía de piezas del catálogo sólo necesitaremos utilizar subclasificación (herencia entre clases de piezas).
   (b) Se ha definir una Pieza-compuesta y una Pieza-simple, ambas como agregaciones de la clase Pieza.
   (c) ✓ Se establece sólo una asociación entre Pieza y Pieza-compuesta.
   (d) Se establecen asociaciones entre Pieza, Pieza-compuesta y Pieza-simple
4. (**0,5**)Seleccionar la única respuesta incorrecta a la siguiente cuestión sobre el patrón de diseño *Inmutable*.
   (a) Los valores de las variables de instancia de los objetos sólo se pueden asignar o modificar en el método constructor de la clase.
   (b) Un método que necesite modificar el estado de la instancia ha de llamar necesariamente al método constructor.
   (c) ✓ Los objetos inmutables no pueden tener variables públicas.
   (d) La ejecución de cualquier método de instancia no puede cambiar el valor de las variables propias del objeto.

Nota: Mutable Objects/Clases don't provide "setter"methods — methods that modify fields or objects referred to by fields
-Make all fields final and private
-Don't allow subclasses to override methods. The simplest way to do this is to declare the class as final. A more sophisticated approach is to make the constructor private and construct instances in factory methods.
-If the instance fields include references to mutable objects, don't allow those objects to be changed. (https://docs.oracle.com/javase/tutorial/essential/concurrency/imstrat.html)

5. (**0,5**)Seleccionar la única respuesta correcta a la siguiente cuestión sobre el patrón de diseño *Factoría Abstracta*.
   (a) ✓ La ejecución del único método del marco de trabajo devuelve a la entidad demandante de creación de instancias una referencia de objeto.
   (b) *Factoría* ha de ser siempre una clase abstracta.
   (c) El método *crearInstancia()* de las factorías siempre devuelve lo mismo.
   (d) La clase de objetos específica del marco de trabajo implementa la interfaz *ClaseGenerica*

6. (**0,5**)Seleccionar la respuesta incorrecta a la siguiente cuestión sobre el patrón de diseño *Visitante*.
   (a) Los visitantes concretos guardan el estado durante el recorrido de la estructura de objetos.
   (b) A la operación *visitar()* de un visitante concreto hay que pasarle la referencia de objeto del elemento que ha llamado al método *aceptar(...)*.
   (c) ✓ *Elemento* no posee subclases.
   (d) Los objetos atienden llamadas a su operación *aceptar()* pero no pueden resolver por sí solos cuál es el visitante concreto que ha de actuar a continuación.

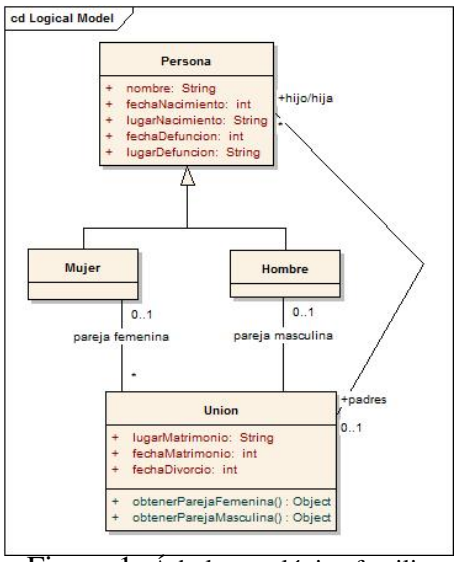

Figura 1: Árbol genealógico familiar

II- Ejercicios sobre la teoría impartida

1. (**0,5**) Encontrar todas las situaciones en las que el patrón `Delegation` podría ser aplicado en el diagrama de clases de la figura 1 que representa un árbol genealógico.

   Solución: `obtenerMadre()/obtenerPadre():Persona`; la clase de los objetos que se devuelven programa estos métodos utilizando el mecanismo de *delegación* –a través de la asociación `hijo-hija/padres`– para lo cual delega en los métodos `obtenerParejaFemenina()/Masculina()` de la clase `Union`.

2. (**1,0**) Calcular el número de McCabe o ciclomático por los tres métodos estudiados: (a) construyendo un grafo, (b) geometría del plano, (c) a partir del número de condicionales para el siguiente programa simple:

```
1  #define N 200
2  int main(){
3    int i , j;
4    int M[N][N];
5    bool v[N];
6    for (i=0; i<N; ++i)
7      for (j=0; j<N; ++j)
8        M[i][j]= i + j;
9      for (i=0; i<N; ++i)
10       if(M[i][j]%2 == 0)
11         v[i]= true;
12         else
13           v[i]= false;
14   return 0;
15 }
```

Solución:

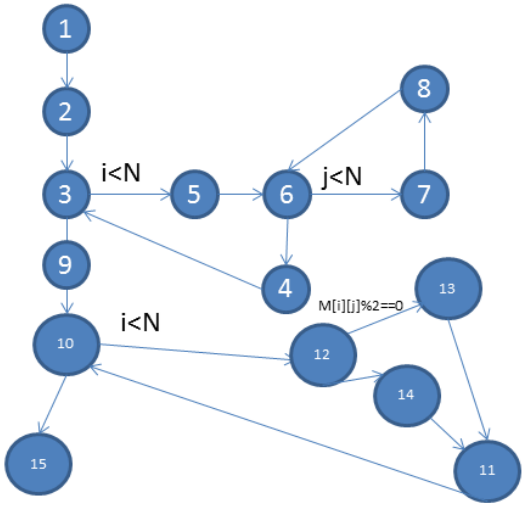

Figura 2: Grafo de flujo equivalente

a aristas= 18; nodos= 15; N= 5

b 5 regiones en el plano

c condicionales= 4; N= 5

3. (**1,0**)Una clase *singleton* que se denomina `Quiz` ha de *provocar* la creación de *preguntas* (objetos) de varios tipos (sólo texto, con sonidos o imágenes) para un aplicación de móvil. La idea para implementar el juego consiste en que, dependiendo del tipo de pregunta que se ha generado en la actividad principal del juego, se crearán las instancias de `PreguntaTexto`, `PreguntaSonidos`, o `PreguntaImagenes` apropiada.

   Utilizando el patrón de diseño factoría abstracta dibujar un diagrama de clases en el que han de aparecer, al menos, las siguientes clases: *ActividadPrincipal*, *Pregunta*, *PreguntaTexto*, *PreguntaSonidos*, *PreguntaImagenes*, *FactoriaPreguntas*, *FactoriaPreguntasTexto*, *FactoriaPreguntasSonidos*, y *FactoriaPreguntasImagenes*. Para que se considere correctamente realizado

hay que programar además la selección de la factoría correcta en la actividad principal dependiendo del tipo de pregunta. Los métodos de creación de las factorías concretas no devolverán el mismo tipo que el método de creación de *FactoriaPreguntas*.

Solución:

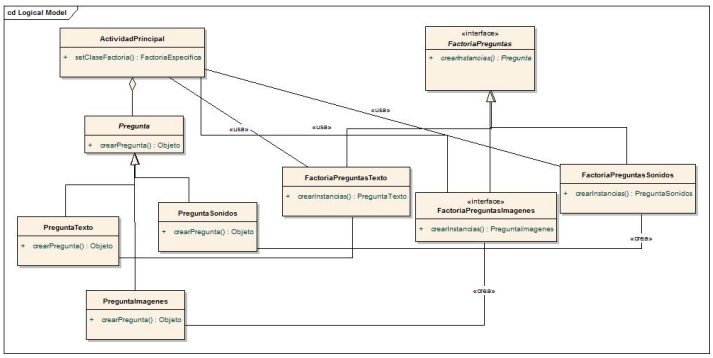

Figura 3: Diagrama de clases del patrón *Factoría abstracta*

4. (**1,0**) Un programador afirma que sus programas están libres de defectos con un nivel de certeza del 80 %. ¿ Con cuántos defectos debe sembrarse el programa que ha realizado antes de la prueba a fin de probar la afirmación anterior del programador? Nota: el plan de prueba establece en este caso que ha que continuar las pruebas del programa hasta encontrar la totalidad de los defectos sembrados.

$$C = \frac{S}{S - N + 1}$$
$$C = 0,80$$
$$S = sembrados$$
$$N = reales = 0$$

Solución: S= 4 defectos

III- Preguntas cortas

1. (**0,5**) Indicar tres posibles usos de los patrones en el diseño orientado a objetos de un sistema software. Solución:
   (a) Modelado: sirve para identificar y representar entidades del mundo real como objetos informáticos
   (b) Estructuración: sirve para descomponer un objeto complejo en otros más simples
   (c) Interfaz: sirve para determinar qué atributos no es necesario que sean visibles desde donde se vayan a utilizar esos objetos

2. (**1,0**) Justificar el patrón de diseño que se elegiría para diseñar un sistema software en el cual un conjunto de entidades (objetos en el diseño OO) comparten información pero se diferencian en aspectos importantes. Queremos evitar a toda costa replicar información común a las mencionadas entidades, prever inconsistencias de información y poder cambiar características particulares en los objetos. Representar la solución propuesta con un diagrama de clases UML. Solución: Pueden servir los patrones `Abstracción/Caso o Flyweight`, en este segundo caso, la mayor parte del estado de los objetos se puede convertir en *extrínseco* de tal forma que muchos de los objetos pueden ser reemplazados por unos pocos objetos compartidos.

3. (**0,5**) ¿Cuáles de los siguientes elementos no tiene una influencia importante en los costos de mantenimiento de un sistema software? Justificar la respuesta.
   (a) Tipo de aplicación (transformacional, reactiva, tiempo–real, etc.)
   (b) ✓ Instalación de una nueva aplicación del sistema
   (c) Tipo de sistema (S, P o E)
   (d) Duración prevista del ciclo de mantenimiento y evolución
   (e) ✓ Actualización con el hardware más novedoso y de mayor rendimiento
   (f) ✓ Plataforma (sistema operativo y software de red) en que está instalado
   (g) Calidad de diseño del sistema
   (h) Calidad de la documentación

IV- Supuesto Práctico

(a) (**1,5**) Explicar brevemente la ecuación fundamental del modelo predictivo de costes de mantenimiento de Belady-Lehman $M = P + K \times c - d$ y contestar a la siguiente cuestión justificándola

(b) ¿ Cuál de las siguientes decisiones eliges como más costo–efectiva para el mantenimiento futuro de un sistema software, que acaba de ser entregado, que posee una documentación poco clara y alta complejidad ?($c = 48$, ver Tabla 1) :

i. Pedir una nueva documentación (10,000 EUR)
ii. Devolver el software para que lo refactoricen y bajen la complejidad ciclomática un 50 % (hora programador= 80 EUR, 10,000 líneas de código, 40 paquetes y 120 componentes)
iii. Cambiar el sistema operativo, la constante K disminuye su valor el 20 %.

complejidad (c)	riesgo	tiempo prueba (horas)
1–10	bajo	10/1000 líneas
11–20	moderado	50/1000 líneas
21–50	alto	100/1000 líneas
51+	imposible tests	—-

Tabla 1: Valoración de riesgos según el número ciclomático

Solución:

a Indica una predicción de los costes de mantenimiento en función de tres factores: P se refiere al esfuerzo necesario para desarrollar ese software; c a la complejidad ciclomática del producto software, que viene multiplicada por una cosntante de tipo empírico; d a la familiaridad que tienen los responsables de mantenimiento con el software en cuestión.

b La opción (i) aumentaría la familiaridad con el software de los miembros del equipo responsable y, por consiguiente, podría llegar a reducir considerablemente el coste de mantenimiento y no es muy cara considerando los beneficios potenciales; (ii) puede resultar muy cara ya que se puede observar claramente que se trata de una aplicación muy compleja y extensa, puede resultar muy caro reducir su complejidad en un 50 % y puede ocasionar replantearse incluso el diseño del software, además después de reducir la complejidad nada nos garantiza que la documentación siga siendo mala e insuficiente; (iii) es incluso menos efectiva que la opción (ii) ya que el término de la complejidad sólo se ve reducido en un 20 %, además el esfuerzo de mantenimiento y la falta de experiencia con ese software se mantendrán inalteradas.