Examen final

Nombre:

13-06-2014

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) Pueden ser un esquema algorítmico para resolver un problema que se repite.
   (b) Las clases de un patrón de diseño no se pueden modificar (p.e.: necesito pasar un parámetro extra cuando invoco ciertos métodos de una clase del patrón) si lo hago, estaría proponiendo un patrón diferente.
   (c) ✓ Un patrón propone una solución a un tipo de problema en un determinado contexto.
   (d) Cada patrón es una estructura de clases fija que se adapta al desarrollo de una determinada aplicación o sistema–software.
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 poseen un menor grado de abstracción que los marcos de trabajo.
   (b) ✓ Los patrones se pueden modificar para adaptarlos a las condiciones de problemas concretos, mientras que los marcos de trabajo se han de utilizar sin modificación.
   (c) Los marcos de trabajo son menos especializados que los patrones de diseño.
   (d) En general, los marcos de trabajo son elementos arquitectónicos más pequeños que los patrones.
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 predicciones meteorológicas. P
   (b) Un sistema de reserva de vuelos y hoteles. E
   (c) Un programa ara jugar al ajedrez.P
   (d) Un programa que utiliza el polimorfismo para calcular la inversa de matrices para diferentes tipos de elementos: reales, complejos, enteros de diferentes precisiones.S
   (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 la única alternativa incorrecta a la siguiente cuestión sobre el proceso sistemático de prueba de una aplicación Web:
   (a) Revisar el modelo de contenidos
   (b) Comprobar el modelo de interfaz respecto de la inclusión de todos 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 diseño
   (f) Pruebas unitarias de componentes funcionales

(g) Revisar la facilidad de navegación a través de toda la arquitectura software de la aplicación

(h) Pruebas de seguridad y robustez de la aplicación o de su entorno (p.e.: buscar enlaces rotos antes de instalar las páginas)

(i) Comprobar compatibilidad con plataformas y sus configuraciones

(j) Pruebas de rendimiento

5. (**0,5**)Seleccionar la única alternativa correcta a la siguiente cuestión sobre matenimiento del software:
   (a) ✓ El mantenimiento *perfectivo* no necesariamente se realiza para detectar posibles fallos en el software
   (b) Los costes de desarrollo (implementación) de un sistema software serán siempre superiores al del mantenimiento del software
   (c) El que un software sea más o menos mantenible depende de factores internos y sólo se puede determinar con el uso prolongado del sistema
   (d) Según el estudio de Lienz y Swanson, el esfuerzo mayor de mantenimiento de un sistema software se emplea en realizar *mantenimiento correctivo*

6. (**0,5**)Seleccionar la única alternativa correcta a la siguiente cuestión sobre evolución del software:
   (a) El mantenimiento del software posee un ciclo de proceso independiente del ciclo de desarrollo del software
   (b) ✓ Los mantenimientos perfectivo y correctivo del software *deterioran* a un sistema software
   (c) El modelo predictivo de Belady–Lehman expresa la relación entre el coste de 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. Tener en cuenta que el valor de K depende fundamentalmente del entorno. Se determina comparando el esfuerzo de desarrollo que presenta nuestro modelo de sistema software con las relaciones de esfuerzo que se dan en varios proyectos de desarrollo reales similares. Por consiguiente, K no se calcula para una plataforma concreta sólo.
   (d) 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

7. (**0,5**)Seleccionar la única respuesta correcta a la siguiente cuestión sobre factores internos que dificultan el mantenimiento de un sistema software:
   (a) El paradigma de programación utilizado (PDO, imperativo, concurrente, etc.) no afecta a la dificultad de mantenimiento del software
   (b) Lo único importante para que un sistema software se pueda mantener sin grandes dificultades es que el número de McCabe o *ciclomático* sea lo más bajo posible
   (c) ✓ La herencia y el polimorfismo podrían ayudar a deteriorar el sistema tras la realización de acciones de mantenimiento (correctivo, adaptativo, etc.)
   (d) El número ciclomático se puede determinar también mediante inspección del código, ya que es igual al número de bucles + 2

II- Ejercicios sobre la teoría impartida

1. (**1,0**) Rehacer el diagrama de clases de la figura 1 para que se adapte al patrón explicado *Jerarquía General*

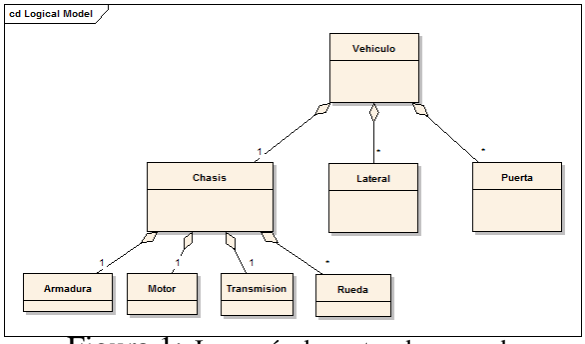

Figura 1: Jerarquía de partes de un coche

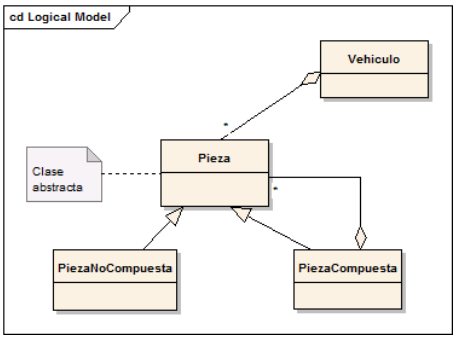

Figura 1: Solución utilizando el patrón Jerarquia General

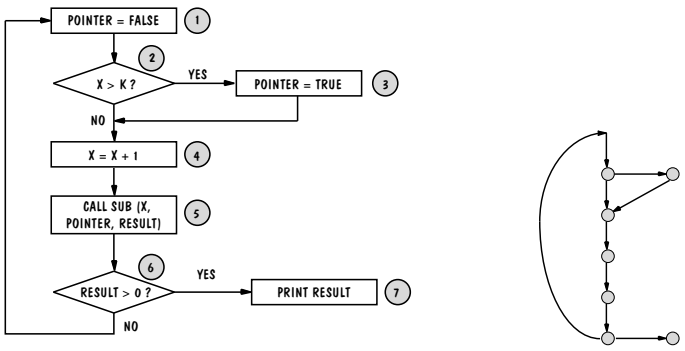

Figura 2: (a) Diagrama de flujo de un programa simple (b) Grafo orientado equivalente

2. (**1,0**) Considerar un diagrama de flujo de un programa como un grafo dirigido en el cual los rombos y cajas del programa se consideran como nodos y las flechas de flujo lógico entre ellos como aristas orientadas del grafo. Por ejemplo el programa de la figura 2 siguiente puede graficarse como se muestra al lado. Calcular el número de McCabe o ciclomático que se infiere para el programa representado por el diagrama de flujo.

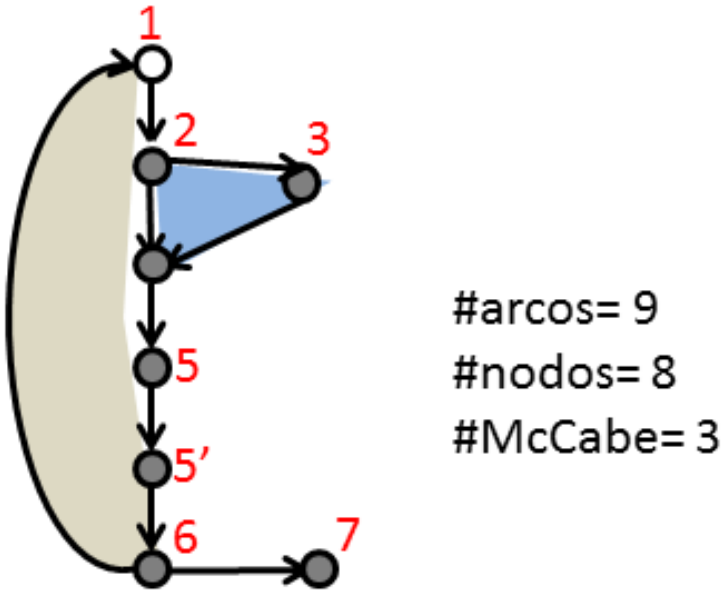

Figura 3: Solución

Por último, probar que la prueba de caminos es equivalente a encontrar todos los caminos posibles a través del grafo. Para demostrar la fórmula de McCabe: $M = E - N + 2$ se puede utilizar un grafo en el cual cada nodo de salida vuelve con un arco hacia el nodo de entrada del componente. En tal caso el grafo será ahora fuertemente conexo y la complejidad ciclomática del programa es igual al número ciclomático del grafo (conocido como el primer número de Betti), que viene dado por la ecuación: (1) $M = E - N + 2 * P$, donde P es su número de componentes conexos. Esto se podría ver como el cálculo del número de ciclos linealmente independientes que existen en el grafo (aquellos ciclos que no contienen a otros ciclos dentro) fuertemente conexo, ya que tiene que existir al menos 1 ciclo por cada punto de salida porque existe el arco hacia atrás hasta el punto de entrada del componente. Evidentemente si eliminamos el arco desde el nodo de salida al de entrada de cada componente, los ciclos se convierten en caminos linealmente independientes dentro del grafo total.

Concluyendo que, para un programa simple (método o subrutina), el parámetro P siempre es igual a 1 y se obtiene la fórmula de McCabe:(2)$M = E - N + 2$ a partir de (1).

3. (**1,0**)Una clase *singleton* que se denomina `MotorJuego` ha de *provocar* la creación de animales (objetos) que pertenezcan a distintos continentes en un juego de ordenador. La idea para implementar el juego consiste en que, dependiendo del continente al que pertenezca el país seleccionado por el usuario, se crean los animales apropiados (por ejemplo: leones y gacelas en Tanzania; pumas y castores en Canada). Utilizando el patrón de diseño *factoría* dibujar un diagrama de clases en el que han de aparecer, al menos, las siguientes clases: `MotorJuego`, `Animales`, `AnimalesAfrica`, `AnimalesAmerica`, `CreadorAnimales` (incluye el método factoría `crearAnimales(Pais XXX)`), `CreadorAnimalesPaisXXX` (que implementa el método factoría).

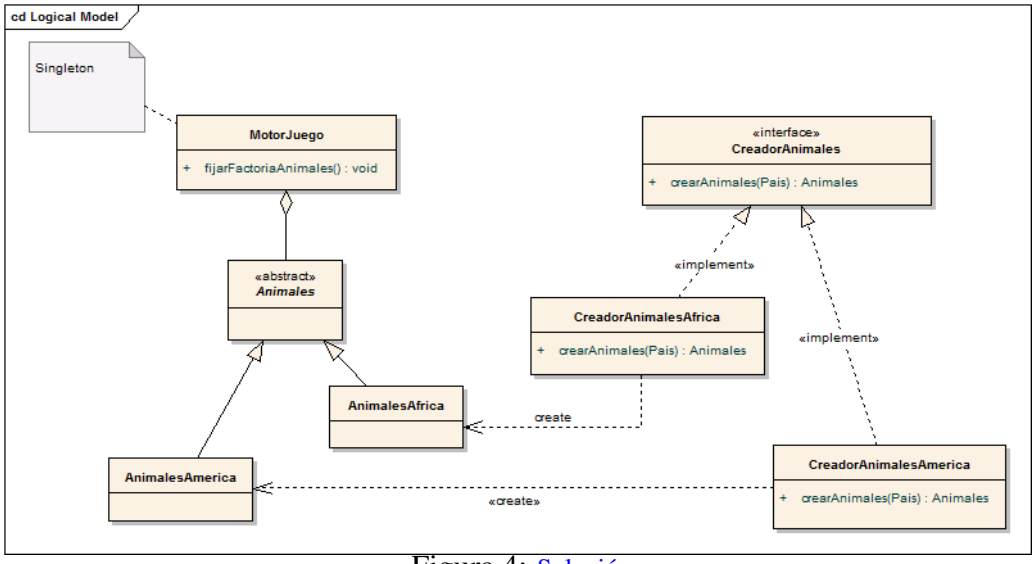

Figura 4: Solución

III- Preguntas cortas

1. (**0,5**) Describir el papel de las *metodologías de diseño OO*, *marcos de trabajo* y *patrones de diseño* en el análisis y diseño de un sistema software. Las metodologías de diseño orientadas a objetos (OO) facilitan el análisis y diseño de un sistema software y mejoran su calidad final gracias a los principios de alta cohesión y bajo acoplamiento que propician las citadas metodologías (Grady Booch, 1987) en el desarrollo de componentes software.

   Los patrones de diseño refuerzan las metodologías y técnicas OO ya que proporcionan posibles soluciones a problemas comunes de diseño de software, lo que permite recurrir a dichas soluciones "prefabricadas" para ahorrar tiempo en la búsqueda de una solución a un problema específico.

   Los marcos de trabajo (*frameworks*), por su parte, nos permiten definir un esqueleto (conjunto de clases) que podemos adaptar para obtener el software concreto que necesitamos construir.

2. (**1,0**) Indicar el patrón de diseño más apropiado para ser aplicado a la solución de los siguientes problemas y justificar dicha elección brevemente:
   (a) Estamos desarrollando una marco de trabajo para usarlo en la presentación de una interfaz de usuario con datos de cotizaciones de bolsa. Queremos obtener las cotizaciones tan pronto estén disponibles. También que nuevas cotizaciones activen determinados cálculos financieros y ambas cosas a la vez, además de hacer que las cotizaciones sean transmitidas sin cables a una red de *pagers*. ¿Cómo diseñar el marco de trabajo de tal forma que varias partes diferentes del código de la aplicación pudieran reaccionar de manera independiente a la llegada de nuevas cotizaciones?

Patrón de diseño: Observador

   Justificación:*CotizarAcciones()* sería el observable y las otras clases (cálculos finanacieros, etc.) serán sus observadores.

   (b) Queremos incluir nuevas operaciones de objetos `PoligonoRegular`, que distorsionen los objetos–polígonos, de tal forma que dejen de ser regulares ¿Cómo hacemos para permitir la ejecución de dichas operaciones sin que se levanten excepciones en el programa?

Patrón de diseño: Inmutable

Justificación: Las operaciones que distorsionan un polígono regular devuelven instancias de otra clase, que no serían polígonos regulares.

(c) Se está construyendo un catálogo de productos que vende la empresa en la que trabajamos. Queremos reutilizar algunas clases proporcionadas por los suministradores, que no podemos modificar. ¿Cómo podíamos asegurarnos que las clases suministradoras puedan seguir utilizándose y mantener el polimorfismo de los objetos de nuestra aplicación?

Patrón de diseño: Adaptador

Justificación: Se puede crear una clase de tipo "adaptador", tal que sus métodos deleguen en los de las clases de los suministradores.

(d) Nuestro programa manipula imágenes muy pesadas. ¿Cómo podemos diseñar un programa que sólo cargue las imágenes en memoria cuando las necesite?

Patrón de diseño: Proxy

Justificación:La mayoría del tiempo se mantendrá una instancia del *proxy–imagen* en memoria. Proxy–imagen proporciona una interfaz idéntica a la de un objeto imagen de la aplicación, pero difiere del objeto real sólo si se ejecutan operaciones que no modifiquen el estado del objeto.

(e) Hemos creado un subsistema con 25 clases. Sabemos que los componentes del resto del sistema accederán, en promedio, a sólo 5 métodos de nuestras clases¿Cómo podemos simplificar la vista que posee el resto del sistema de nuestro subsistema?

Patrón de diseño:Façade

Justificación:Se programa una clase de este tipo con una interfaz pública de sólo 5 métodos.

3. (**1,0**) (a) 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. color blue La ecuación anterior indica que la predicción de costes debe tener en cuenta 3 factores fundamentales: una estimación sobre el esfuerzo necesario para el desarrollo (análisis,evaluación,diseño,codificación y pruebas) del sistema (P); la complejidad ciclomática (c) del propio producto software, multiplicado por una constante empírica que depende del entorno del sistema (K); la familiaridad que poseen los técnicos de mantenimiento con el software que han de mantener (d). (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) Elegir la (a) :
   (a) Pedir una nueva documentación (10,000 EUR) aumentará la familiaridad con el software que se ha de mantener y, por tanto, podría reducir considerablemente los costes derivados de su mantenimiento. No es la inversión más cara, considerando los beneficios que reportaría.
   (b) 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) Puede resultar muy cara de llevar a cabo: 80,000 EUR para revisar un código de 10,000 líneas con el número ciclomático= 48. Los datos que nos están dando demuestran que se trata de un software muy complejo (riesgo "alto" según la tabla) y extenso. Se requieren

muchas horas de trabajo de programadores para reducir su complejidad en un 50%. Es posible que también haya que modificar el diseño del software con lo que los costes se dispararían. Además, incluso después de esta modificación, la documentación seguiría siendo deficiente, por lo que persistirían las dificultades para conseguir un buen mantenimiento del sistema.

(c) Cambiar el sistema operativo, la constante `K` disminuye su valor el 20%. Es menos efectiva que la opción (b) anterior, ya que la complejidad se reduce sólo en un 20% en lugar del 50% si se adopta la solución anterior. Esta solución no tendría ningún impacto en el esfuerzo (P) de desarrollo ni en la familiaridad (d) con el sistema, por consiguiente los costes de mantenimiento se verían poco rebajados.

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

IV- Supuesto Práctico

1. (**1,0**) Considerando el siguiente programa `Java` cuando se ejecuta: (a)sin argumentos, (b) con un objeto *String* como argumento y (c) con argumentos enteros: `0, 1, 2` y `99`. Explicar cómo funciona la gestión de las excepciones que se han implementado (indicar el orden de ejecución de los bloques) en dicho programa y la salida que producirá en cada caso. Solución en la relación de ejercicios resueltos del Tema 4: ejercicio #63.

```
class MiExcepcion extends Exception{
  public MiExcepcion(){ super();}
  public MiExcepcion(String s){ super(s);}
}
class MiOtraExcepcion extends Exception{
  public MiOtraExcepcion(){ super();}
  public MiOtraExcepcion(String s){ super(s);}
}
class MiSubExcepcion extends MiExcepcion{
  public MiSubExcepcion(){ super();}
  public MiSubExcepcion(String s){ super(s);}
}
public class throwtest{
   public static void main(String argv[]){
     int i;
     try{
       i= Integer.parseInt(argv[0]);
     }
     catch(ArrayIndexOutOfBoundsException e){
       System.out.println("Debes especificar un argumento");
       return;
     }
     catch(NumberFormatException e){
       System.out.println("Debes especificar un argumento entero");
       return;
     }
     a(i);
   }
public static void a(int i){
  try{
    b(i);
  }
  catch(MiExcepcion e){//Punto 1
    if(e instanceOfMiSubExcepcion)
      System.out.print("MiSubEXcepcion:");
    else
      System.out.print("MiExcepcion:");
    System.out.println(e.getMessage());
    System.out.println("Manejado en el punto 1");
  }
}
public static void b(int i) throws MiExcepcion{
  int result;
  try{
    System.out.print("i="+i);
    resultado= c(i);
    System.out.print("c(i)="+resultado);
  }
  catch(MiOtraExcepcion e){  //Punto 2
    System.out.println("MiOtraExcepcion:"+e.getMessage());
    System.out.println("Manejado en el punto 2");
  }
  finally{
    System.out.print("\n");
  }
public static int c(int i) throws MiExcepcion, MiOtraExcepcion{
  switch(i){
    case 0: throw new MiExcepcion("entrada demasiado baja");
    case 1: throw new MiSubExcepcion("entrada todavia muy baja");
    case 99: throw new MiOtraExcepcion("entrada muy alta");
    default: return i*i;
  }
```

```
63 }
64 }
```