PROGRAMACIÓN ORIENTADA A OBJETOS **30/06/2006**

NOMBRE:_____Titulación:_____

1. (**0'75 ptos**) ¿Qué significan los conceptos de ***alta cohesión*** y ***bajo acoplamiento*** y qué tienen que ver con el modelo de objetos?

2. Los elementos que constituyen un Sistema de Ficheros son: Ficheros y Carpetas. Una carpeta puede estar formada por un número indeterminado de ficheros y carpetas. Todos los elementos del Sistema de Ficheros pueden comprimirse, para ello disponen de un método `comprimir(Compresor c)`, que recibe como argumento un objeto compresor que es el encargado de realizar la compresión. A su vez, todo Compresor tiene disponible el método `comprimir` con dos argumentos: el Fichero que se va a comprimir y un `String` con el nombre del fichero comprimido. En consecuencia, para comprimir una carpeta habrá que comprimir uno a uno los ficheros que contiene.

a) (**1 pto**) Representa gráficamente la jerarquía que modela los elementos del sistema de ficheros e implementa en cada caso el método `comprimir`.

b) (**1 pto**) Dado que existen diferentes algoritmos de compresión, explica detalladamente cómo sería posible, manteniendo la signatura actual del método comprimir en el sistema de ficheros, que en el momento de invocar dicho método sobre un fichero o una carpeta se le pasara como argumento el algoritmo de compresión que debe utilizar.

3. El método `comprimir` de la clase `Compresor` lanzará la excepción `OutOfMemoryException` si no hay memoria suficiente para almacenar el fichero comprimido y `NotAvailableFileException` si el fichero que se va a comprimir está abierto.

a) (**0'5 ptos**) Justifica el tipo (comprobada o no comprobada) de cada una de las excepciones.

b) (**1 pto**) El método comprimir de las clases que modelan los elementos del sistema de ficheros deben informar a los clientes que lo invoquen de que se ha producido un fallo en la compresión del fichero detallando, mediante un `String`, el tipo concreto del error que se ha producido. Modifica el método `comprimir` de la clase que representa un `Fichero` para que se ajuste a esta especificación. **Justifica el mecanismo de control de errores elegido**.

4. Sea la clase `Plotter` la encargada de dibujar funciones, en concreto se ha programado para dibujar la función seno y la función coseno (cada una de las funciones modelada en una clase diferente, `PlotSeno` y `PlotCoseno` respectivamente) de la siguiente manera:

```
class Plotter{
      Dimension d; //dimensión del área de dibujo
      …
      public void dibujaFuncion (Graphics g){
            for (int px=0; px<d.width; px++){
                  double x = (double)(px – xorigin)/(double)xratio;
                  double y;
                  if (this instanceof PlotSeno)
                        y = Math.sin(x);
                  else if (this instanceof PlotCoseno)
                        y = Math.cos(x);
                  int py = yorigin – (int)(y*yratio);
                  g.fillOval(px-1, py-1, 3, 3);
            }
      }
}
```

a) (**0'75 ptos**) Justifica la incorrección del código anterior de acuerdo con los elementos del modelo del objetos y los principios de diseño de Abierto-Cerrado y Elección Única.

b) (**1 pto**) Corrige el código anterior aplicando el patrón de Diseño del Método Plantilla, modificando todo lo que consideres necesario.

5. Supuesta la declaración de una variable `p` de tipo `Plotter`:

a) (0'75 ptos) Compara cómo Eiffel, Java y C++ proporcionan semántica valor (almacenamiento) y referencia para sus variables.

b) (0'5 ptos) ¿Sería correcto el siguiente código Java? Justifica la respuesta.

```
Plotter p;
p.dibujaFuncion(areaGrafica);  //areaGrafica es una referencia a un objeto gráfico
```

6. Sea la clase Eiffel `ControlRemoto` la encargada de encender y apagar todos los dispositivos que tiene registrados. Estos dispositivos pueden ser de diferentes tipos: TV, Alarma, Luces, … con la única relación entre ellos de que pueden encenderse y apagarse.

```
class ControlRemoto feature
      dispositivos: LIST [?]
      ...
      encender is do
            from dispositivos.start
            until dispositivos.after
            do
                  dispositivos.item.on
                  dispositivos.forth
            end
      end
      apagar is do
            from dispositivos.start
            until dispositivos.after
            do
                  dispositivos.item.off
                  dispositivos.forth
            end
      end
end
```

a) (0'5 ptos) Implementa la clase por la que habría que sustituir el **?** del código anterior.

b) (0'5 ptos) Explica por qué son necesarias las clases diferidas en un lenguaje con comprobación estática de tipos. Utiliza el código del ejercicio para ilustrar la explicación.

c) (1 pto) Dado que los esquemas de recorrido de los métodos encender y apagar son idénticos, cambia la implementación dada por otra en la que se haga uso de los iteradores vistos en clase para el lenguaje Eiffel. Implementa todo lo que sea necesario para resolver el ejercicio.

d) (0'25 ptos) ¿Los iteradores utilizados en el apartado anterior son iteradores internos o externos? Justifica la respuesta.

e) (0'5 ptos) Utiliza la implementación de los iteradores para explicar el significado de código genérico.