Nombre: ..... Fecha: / 11 / 2008

HOJA DE EJERCICIOS 2
RELACIONES ENTRE CLASES. HERENCIA ENTRE CLASES

Esta hoja de ejercicios nos servirá para repasar las nociones más importantes del Tema 2.

1. Señala los distintos tipos de relaciones que se pueden producir entre dos clases, así como la forma de caracterizar dichas relaciones.

2. Explica cómo se implementan generalmente las relaciones de agregación y de asociación en Java y en C++.

3. Señala algunas de las diferencias del trabajo con “arrays” en Java y C++.

4. Comenta algunas de las ventajas principales de utilizar relaciones de herencia a la hora de implementar clases.

5. ¿Cuál es la palabra clave en Java que permite definir una relación de herencia entre dos clases en Java? Escribe un ejemplo de cabecera de una clase que tenga una relación de herencia.

6. ¿Cuál es la sintaxis en C++ para definir relaciones de herencia? Escribe un ejemplo de cabecera de una clase que contenga una relación de herencia.

7. ¿Cuál debe ser la primera operación a realizar siempre en un constructor de una clase derivada?

8. ¿Para qué se utiliza la redefinición de métodos?

9. Explica la diferencia entre los dos casos de herencia que hemos visto en el tema (que dimos en llamar “hijos buenos” e “hijos malos”).

10. Define el ámbito de visibilidad del modificador de acceso “protected” y comenta alguno de sus posibles usos en Java y C++.

11. Representa en UML, Java y C++ el siguiente sistema de información:

Queremos crear una aplicación para calcular el tipo de IVA que deben pagar los artículos de un comercio. El tipo del IVA puede ser del 4%, del 7% o del 16%, dependiendo del tipo de artículo. Definiremos una clase “Articulo” que tendrá como atributos un nombre identificador del mismo y el precio correspondiente (sin IVA); los atributos serán privados. Asimismo, la clase contendrá métodos de acceso y modificación de dicho nombre, y un método de acceso al precio. Sólo incorporará un constructor que reciba como parámetros el nombre del artículo y su precio.

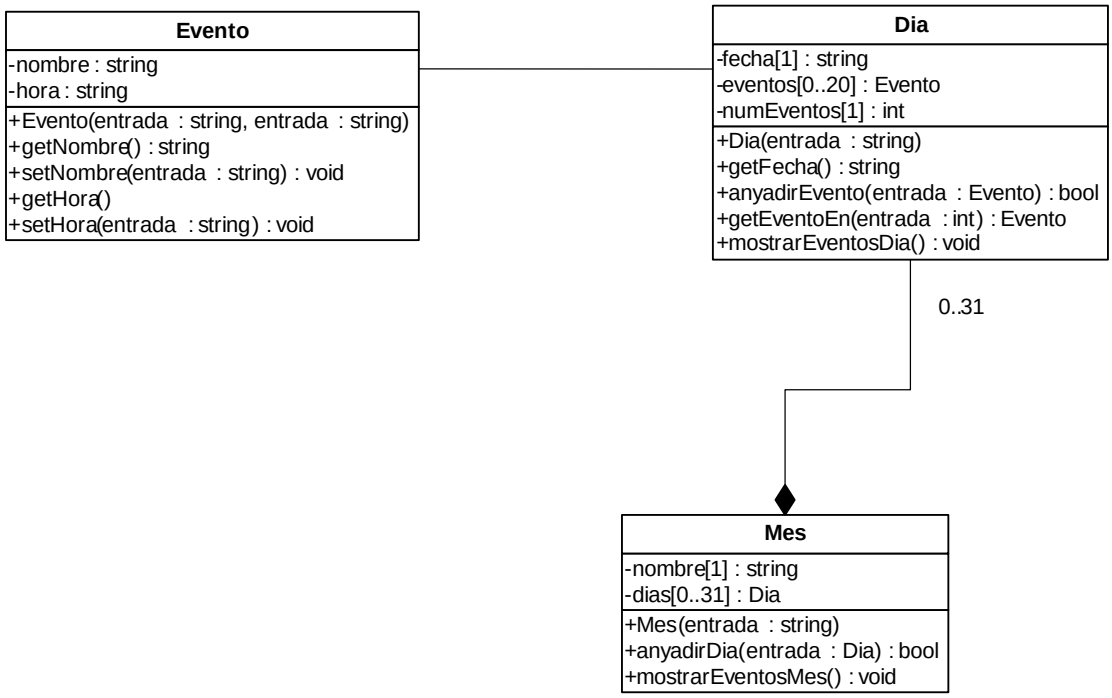
Definimos tres clases que hereden de la misma, de nombres Tipo4, Tipo7 y Tipo16. Las mismas contendrán una constante de clase, llamada TIPO, con el valor del IVA correspondiente (4%, 7%, 16%). Estas clases incorporarán un método llamado “getParteIVA(): double” que debe calcular, no el precio, sino la cantidad que debe ser abonada de IVA por un artículo (es decir, el resultado de multiplicar su precio por la tasa de IVA y dividirlo por 100). Estas clases también deben redefinir el método “getPrecio(): double”, de forma que éste devuelva el precio más el IVA.

12. Escribe un programa principal en Java y C++ que realice las siguientes acciones:

a) Declara un objeto art1 de la clase Tipo4 y créalo con los atributos “Harina”, y “0.5” (euros)
b) Declara un objeto art2 de la clase Tipo7 y créalo con los atributos “Lentillas”, y “50”
c) Declara un objeto art3 de la clase Tipo16 y créalo con los atributos “Armario”, y “300”
d) Muestra el valor del IVA (sobre el precio) para cada uno de esos artículos así como su precio total ¿Qué método “getPrecio(): double” está siendo invocado? ¿El de la clase “Articulo” o el de la clase derivada?

e) Define una estructura genérica (un “array”) “array_articulos” en el cual puedas introducir los tres artículos creados
f) Crea un iterador sobre ese “array” e invoca de nuevo al método “getParteIVA(): double” sobre cada uno de los elementos del “array”. Observa y comenta el resultado
g) Crea un iterador sobre ese “array” e invoca al método “getPrecio(): double” sobre cada uno de los elementos del “array”. Observa el resultado y compáralo con el obtenido en d)

13. Programa el siguiente diagrama UML en Java y C++:



Debes tener en cuenta los siguientes requisitos:

a) La hora será un atributo de tipo “string” que tomará valores de la forma “00:00” a “23:59”
b) Los objetos de la clase “evento” no tienen por qué estar ordenados dentro de un “Dia”, simplemente se van introduciendo en el “array” correspondiente
c) El atributo “fecha” está representado por medio de un “string” cuyo valor será de la forma “aaaammdd”
d) El constructor “Dia(string)” recibe como único parámetro la fecha del día que debe ser creado. No inicializará ningún evento.
e) El método “anyadirEvento(Evento): bool” devolverá verdadero si hemos podido añadir el objeto de la clase “Evento” a nuestro “array”, o falso en caso contrario
f) El método “getEventoEn (int): Evento” devuelve el evento en la posición señalada por “int” del “array”. Si el índice es mayor que el número de componentes en el “array”, devolverá el objeto “null” en Java, y un objeto por defecto en C++
g) El método “mostrarEventosDia(): void” debe mostrar únicamente los eventos que hayan sido inicializados; caso de que un día no contenga ningún evento, mostrará un mensaje advirtiendo de tal situación: “El dia ” ... “no contiene eventos”
h) El método “mostrarEventosMes(): void” debe mostrar los eventos de todos los día del mes, incluidos aquellos días que no contengan ningún evento

Nota: no incluyas, por el momento, ningún método o constructor adicional al diagrama anterior.

14. ¿Qué problemas encuentras al implementar el anterior diagrama en Java y C++? ¿Qué soluciones se te ocurren? ¿Qué cambios requieren sobre el diagrama UML anterior?

15. Produce ahora un programa cliente de la anterior implementación que contenga las siguientes operaciones:

a) Declara un objeto event1 de la clase “Evento” que contenga los valores iniciales “Reunion consejo direccion” y “10:30”
b) Declara un objeto event2 de la clase “Evento” que contenga los valores iniciales “Visita medico” y “13:45”
c) Declara un objeto event3 de la clase “Evento” que contenga los valores iniciales “Recoger correo” y “17:50”
d) Declara un objeto de la clase “Dia” con valor “20080614”
e) Añádele los tres eventos anteriores
f) Declara un objeto de la clase “Mes” de nombre “Junio”
f) Comprueba el resultado de ejecutar el método “mostrarDia(): void” (debería mostrar los tres eventos)
g) Comprueba el valor de “mostrarMes(): void” (debería mostrar todos los días del mes con sus correspondientes eventos)

16. Aprovecha los resultados de los ejercicios anteriores para comentar brevemente las peculiaridades de la inicialización de “arrays” de objetos en constructores en Java y C++ y de punteros a objetos en C++.

17. Explica el comportamiento del siguiente fragmento de código en Java:

```
int [] array1 = new int [50];
array1 [0] = 1;
array1 [1] = 2;
array1 [2] = 3;
array1 [3] = 4;
array1 [4] = 5;
int [] array2  new int [50];

array2 = array1;
array2 [0] = 25;

System.out.println (“El valor del array2 en su posicion 0 es ” + array2 [0]);
System.out.println (“El valor del array1 en su posicion 0 es ” + array1 [0]);
```

Relaciona tu respuesta con el modelo de memoria de Java.

18. Trata de representar el anterior fragmento de código en C++. Comenta las diversas opciones que consideres.

19. Comenta el comportamiento del siguiente fragmento de código en Java:

```
//Fichero Principal_AB.java

class A{
        private int atributo;

        public A (){
                this.atributo = 0;
        }

        public A (int atributo){
                this.atributo = atributo;
        }

        public void metodoAux(){
                System.out.println (“Llamando al metodoAux de la clase A”);
        }

        public void metodoAux2(){
                System.out.println (“Llamando al metodoAux2 de la clase A”);
                this.metodoAux();
        }
}
```

```
class B extends A{
	public B (){
		super();
	}

	public void metodoAux2 (){
		System.out.println (“Llamando al metodoAux de la clase B”);
	}

}

public class Principal_AB {
	public static void main (String [] args){
		A objeto1 = new A();
		A objeto3 = new B();
		B objeto2 = new B();

		objeto1.metodoAux2();
		objeto2.metodoAux2();
		objeto3.metodoAux2();
	}
}
```

Escribe el resultado de la ejecución del código anterior (relaciónalo con la idea de tipado dinámico).

Traduce el ejemplo a C++ y escribe el resultado de ejecutar el mismo. Recuerda que en C++, por defecto, el tipado es estático.