PRIMER EXAMEN PARCIAL

ESTRUCTURA DE DATOS Y ALGORITMOS

27 FEBRERO 2014

Apellidos y Nombre:	
Grupo	

Algunas reglas:

- Antes de comenzar el examen, escribe tu nombre y grupo.
- Lee atentamente el enunciado de cada ejercicio.
- Utiliza las hojas de examen (hojas de cuadros) para preparar tu respuesta. Una vez que estés segur@ de la solución de un ejercicio, escribe la respuesta en el hueco que le corresponde. Los profesores solo recogerán este cuadernillo.

Dado el siguiente código:

```
public class Alumno {
    private String nombre;

    /** método getNombre */
    public String getNombre(){
        return nombre;
    }

    /** método seExamina, que indica cuando un alumno
     * se examina de un Curso dado - como parámetro
     * @return true o false de forma aleatoria
     */
    public boolean seExamina(CursoExtraescolar curso){
        if (Math.random()>0.5) return true;
        else return false;
    }
}
```

Elige la/s respuesta/s correcta/s para cada pregunta:

1. **(0,5 puntos) Para poder utilizar el método *seExamina* de la clase Alumno desde el método *main* de una nueva clase llamada *Test*, y suponiendo que se ha implementado previamente una clase *CursoExtraescolar*:**

   a) Basta con definir una variable de tipo Alumno (*a*) y otra de tipo CursoExtraescolar (*curso*) antes de hacer la llamada al método mediante el siguiente código: *a.seExamina(curso);*

   b) Basta con definir una variable de tipo Alumno (*a*) y otra de tipo CursoExtraescolar (*curso*) e instanciar los objetos antes de hacer la llamada al método mediante el siguiente código: *a.seExamina(curso);*

   c) Basta con definir una variable de tipo CursoExtraescolar (*curso*) y hacer una llamada al método mediante el siguiente código: *Alumno.seExamina(curso);*

d) Basta con hacer una llamada al método mediante el siguiente código:
Alumno.seExamina(CursoExtraescolar);

2. (0,5 puntos) Relacionado con los constructores de la clase Alumno:

a) El código es incorrecto, porque para cada clase es necesario crear un método constructor. Si no se hace, la clase no se podría instanciar.
b) El código es correcto, pero faltaría incluir un constructor para poder instanciar objetos de la clase Alumno.
c) El código es correcto, y para instanciar un objeto de la clase Alumno bastaría con llamar a su constructor por defecto. Un ejemplo es:
Alumno a = new Alumno("Daniel");
d) El código es correcto, y para instanciar un objeto de la clase Alumno bastaría con llamar a su constructor por defecto. Un ejemplo es
Alumno a = new Alumno();

3. (0,5 puntos) Relacionado con los constructores de la clase Alumno. ¿Se pueden crear dos constructores en la misma clase?

a) Sí y solo si los nombres de los constructores son diferentes.
b) Sí y solo si el número de parámetros de los constructores son diferentes.
c) Sí y solo si el número o tipo de datos de los parámetros de los constructores son diferentes.
d) No, nunca, los constructores no admiten sobrecarga.

4. (0,5 puntos) Dada la definición de las siguientes interfaces y clases, marca cuáles de las siguientes líneas de código relativas a instanciación de objetos es correcta:

```
public interface IPersona {...}
public abstract class Empleado implements IPersona {...}
public class Profesor extends Empleado {...}
```

a) IPersona p = new Empleado();
b) IPersona p = new Profesor();
c) Empleado e = new Profesor();
d) Profesor p = new Empleado();

5. **(0,5 puntos) Dado el siguiente código y suponiendo que se ha implementado anteriormente una clase Profesor, señala las opciones correctas:**

```
Profesor p1 = new Profesor();
Profesor p2 = p1;
```

a) Se ha creado un objeto de tipo Profesor y tanto la variable p1 como la variable p2 hacen referencia a este objeto.

b) Se han creado dos objetos de tipo Profesor, uno referenciado por la variable p1 y otro referenciado por la variable p2.

c) El código es incorrecto (error de compilación), porque no se está creando bien ningún objeto de tipo Profesor.

d) El código es incorrecto, ya que en la segunda línea de código la asignación debería ser al revés (p1 = p2).

6. **(0,5 puntos) Relacionado con los modificadores de acceso a atributos y métodos de las clases y suponiendo el siguiente código, marca las opciones correctas:**

```
public abstract class Empleado{
        protected int v1;
        …
}
public class Profesor extends Empleado {...}
```

a) Desde la clase *Profesor* se podrá acceder directamente a la variable v1 para consultar o modificar. No será necesario utilizar los métodos get/set de acceso a la variable.

b) Desde la clase *Empleado* se podrá acceder directamente a la variable v1 para consultar o modificar. No será necesario utilizar los métodos get/set de acceso a la variable.

c) Solo se podrá acceder desde la clase *Profesor* directamente a la variable v1 para consultar o modificar si ésta se hubiera definido como *private*.

d) Desde la clase *Profesor* NO se podrá acceder directamente a la variable v1 para consultar o modificar. Será necesario utilizar los métodos get/set de acceso a la variable (si existiesen).

PRIMER EXAMEN PARCIAL

ESTRUCTURA DE DATOS Y ALGORITMOS

27 FEBRERO 2014

Apellidos y Nombre:	
Grupo	

Ejercicio 2 (2 puntos): En el Colegio "El Greco" de Leganés se desea desarrollar un software que lleve el control de los cursos extraescolares que se imparten. Estos cursos se imparten en horarios de tarde, se puede inscribir cualquier alumno del colegio y hay cursos de diversos tipos.

Teniendo en cuenta el código de la clase Alumno proporcionado, señala la/s respuesta/s correcta/s a los siguientes puntos:

1. **(0,5 puntos) ¿Cómo se implementarían los Cursos Extraescolares? Teniendo en cuenta que de ellos solo se sabe que se comportarán de la siguiente forma: todos ellos permitirán nuevas inscripciones de alumnos en cualquier momento del curso (*addAlumno)*; permiten mostrar el nombre de todos sus alumnos (*mostrarAlumnos*); permiten mostrar todas las características del Curso Extraescolar específico (*mostrarCurso*). Además, es imprescindible tener en cuenta que, en principio, no existen restricciones aún a la hora de implementar, por ejemplo, el conjunto de alumnos, que podría ser implementado mediante listas, arrays, conjuntos, etc.**

   a. Mediante una Clase Abstracta *CursoExtraescolar* donde todos sus métodos se implementan.
   public abstract class CursoExtraescolar

   b. Mediante una Clase Abstracta *CursoExtraescolar* donde todos sus métodos son abstractos.
   public abstract class CursoExtraescolar

   c. Mediante una Clase *CursoExtraescolar* donde ninguno de sus métodos se implementan.
   public class CursoExtraescolar

   d. Mediante una Interfaz *ICursoExtraescolar* donde todos sus métodos son abstractos.
   public interface ICursoExtraescolar

2. **(0,5 puntos) Teniendo en cuenta que existe una clase abstracta *CursoExtraescolar* con una variable** *listaAlumnos* implementada mediante un array**, es buena práctica en el paradigma Orientado a Objetos:**

   a. Definir la variable como *public* para no poner ningún límite de acceso a la misma.

   b. Definir la variable como *private* e implementar los métodos privados get/set de acceso a la variable.

   c. Definir la variable como *private* o *protected* e implementar los métodos get/set de acceso a la variable (con el mismo modificador de acceso que la variable).

   d. Definir la variable como *private* o *protected* e implementar los métodos get/set de acceso a la variable como públicos/protegidos, dependiendo del caso.

3. **(0,5 puntos) Suponiendo que existe una clase abstracta** *CursoExtraescolar***, y que se ha implementado la subclase** *CursoIngles* **con la siguiente cabecera:**

   public class CursoIngles extends CursoExtraescolar

   a. *CursoIngles* hereda todos los elementos de *CursoExtraescolar* y además tiene la obligación de implementar todos sus métodos (sean o no abstractos).

   b. *CursoIngles* hereda todos los elementos de *CursoExtraescolar* y además tiene la obligación de implementar todos sus métodos abstractos.

   c. *CursoExtraescolar* hereda todos los elementos de *CursoIngles* y además tiene la obligación de implementar todos sus métodos (sean o no abstractos).

   d. *CursoExtraescolar* hereda todos los elementos de *CursoIngles* y además tiene la obligación de implementar todos sus métodos abstractos.

4. **(0,5 puntos) Teniendo en cuenta la clase abstracta *CursoExtraescolar* con un método implementado denominado *mostrar()* y su subclase *CursoIngles* que sobreescribe este método. Sea así mismo el código de la clase Test donde se instancian diferentes objetos:**

```
public abstract class CursoExtraescolar{

    public void mostrar(){
        System.out.println("Soy CursoExtraescolar");
    }

}
```

```
public class CursoIngles extends CursoExtraescolar {
    int nivel;

    @Override
    public void mostrar(){
        System.out.println("Soy CursoIngles");
    }
}
```

```
public class Test {
    public static void main(String[] args) {
        CursoExtraescolar v1 = new CursoIngles();
        v1.mostrar();

        CursoIngles v2 = new CursoIngles();
        v2.mostrar();
    }
}
```

a. *v1.mostrar()* imprimirá el mensaje por pantalla “Soy CursoIngles”.
b. *v1.mostrar()* imprimirá el mensaje por pantalla “Soy CursoExtraescolar”.
c. *v2.mostar()* imprimirá el mensaje por pantalla “Soy CursoIngles”.
d. *v2.mostrar()* imprimirá el mensaje por pantalla “Soy CursoExtraescolar”.