


Apellidos		Nombre	
Firma		NIA	Grupo

<p>EXAMEN PROGRAMACIÓN 21 de Septiembre de 2007 INGENIERÍA INFORMÁTICA Primera parte: Cuestiones 1,5 horas</p>	 <p>UNIVERSIDAD CARLOS III DE MADRID</p>
---	---

Pregunta 1 (0,5 puntos).- Indicar si las siguientes afirmaciones son ciertas, y **explicar** brevemente por qué.

- Al crear objetos de una clase abstracta, sólo podremos usar los métodos que no sean abstractos.*
- Una clase derivada sólo puede acceder a un método privado de la superclase si y sólo si ambas pertenecen al mismo paquete.*

Pregunta 2 (0,5 puntos).- Indicar si las siguientes afirmaciones son ciertas, y **explicar** brevemente por qué.

- a) *Al igual que es posible que distintas clases implementen una única interfaz, una única clase puede implementar dos o más interfaces.*
- b) *Para implementar una interfaz, la clase debe proporcionar la implementación de al menos un método definido en la interfaz.*

Pregunta 3 (0,5 puntos).- Dada la siguiente lista: (21, 5, 34, 6, 17, 11). Escribir cómo evolucionaría dicha lista hasta quedar completamente ordenada mediante el método Rápido (QuickSort)

Pregunta 4 (0,5 puntos).- Dado el siguiente método:

```
public static void leer(String nombre){
    String linea;
    int x = 0;
    int y = 0;
    try {
        BufferedReader br = new BufferedReader(new
                                                FileReader(nombre));

        linea = br.readLine();
        while (linea!=null){
            linea = br.readLine();
            System.out.println(linea);
        }
        System.out.println("leido");
        br.close();
    }catch (IOException e){
        System.out.println("Exception");
    }finally {
        System.out.println("---");
    }
    System.out.println("adios");
}
```

Y el fichero prueba.txt con el siguiente contenido:

a b
c
d
e f
g
h

Explicar qué sale por pantalla al invocar leer("prueba.txt"). Modificar el método leer para que imprima sólo las líneas impares.

Pregunta 5 (0,5 Puntos).- Dadas las siguientes clases e interfaces:

```
public abstract class PrimerPlato {
    public void PedirAgua(){
        System.out.println(" Agua, por favor...");
    }
    public void PedirPan(){
        System.out.println(" Pan, por favor...");
    }
}

public abstract class SegundoPlato {
    public abstract void PedirVino();
}

public interface Postre {
    public void PedirDulce();
    public void PedirAmargo();
}
```

Implementar, si es posible, la clase `MenuDelDia` de tal forma que herede los métodos de `PrimerPlato` y `SegundoPlato`, e implemente a su vez todos los métodos contenidos en la clase `Postre`. Si no es posible, explicar por qué, y modificar el código de tal forma que permita crear dicha clase `MenuDelDia`.

Pregunta 6 (0,5 puntos).- Dado el siguiente método:

```
public static int recursivo (int[] vector, int j){
    if (j > 0) {
        int aux = 0;
        for(int i=0; i<j; i++) {
            aux += vector[i];
        }
        return aux - recursivo(vector, j-1) + recursivo(vector,
j-2);
    }
    else return 0;
}
```

Explicar cual sería el resultado por pantalla de la siguiente invocación:

```
int[] vector = {3, 5, -2, 1};
System.out.println(recursivo(vector, 3));
```

Pregunta 7 (0,5 puntos).- Dadas las siguientes clases:

```
public class Mamifero {
    private String habitat;
    public Mamifero (String h){
        habitat=h;
    }
    public Mamifero(){
        this("terrestre");
    }
    public String tipo (){
        return "mamífero "+habitat;
    }
}

public class Cetaceo extends Mamifero{
    public Cetaceo (String h){
        super(h);
    }
    public Cetaceo(){
        this("acuático");
    }
    public String tipo(){
        return super.tipo()+ " (cetáceo)";
    }
}
```

Y el método:

```
public static void imprimirTipo(Mamifero m){
    System.out.println(m.tipo());
}
```

Explicar cuál será el resultado por pantalla de la siguiente secuencia de invocaciones:

```
Mamifero elefante = new Mamifero();
Cetaceo delfin = new Cetaceo();
imprimirTipo(elefante);
imprimirTipo(delfin);
```

Pregunta 8 (0,5 puntos).- Dado el siguiente método:

```
public static void metodo1 (){
    String [] MensCifrado = new String[11];
    MensCifrado[1] = "excepciones";
    MensCifrado[2] = "interfaces";
    MensCifrado[3] = "de";
    MensCifrado[4] = "error";
    MensCifrado[5] = "ejemplo";
    MensCifrado[6] = "otro";
    MensCifrado[7] = "un";
    MensCifrado[8] = "es";
    MensCifrado[9] = "Aquel";
    MensCifrado[10] = "Este";
    try{
        System.out.println("El mensaje cifrado es: ");
        for (int i=8; i<MensCifrado.length; i=i-2){
            System.out.println(MensCifrado[i-1]);
        }
        System.out.println("For your eyes only");
    }catch (NullPointerException npe){
        System.out.println(" null pointer ");
    }catch (ArrayIndexOutOfBoundsException aiobe){
        System.out.println(" array index out ");
    }catch (Exception e){
        System.out.println(" exception ");
    }finally{
        System.out.println("Fin del comunicado");
    }
}
```

Explicar cual sería la salida por pantalla al invocar dicho método

EXAMEN PROGRAMACIÓN
21 de Septiembre de 2007
INGENIERÍA INFORMÁTICA
Segunda parte: problemas
2,5 horas



UNIVERSIDAD CARLOS III DE MADRID

Nos encontramos en algún momento entre los años 1.100 y 1.300 a.C. El rapto de Helena de Esparta por Paris de Troya ha desencadenado la guerra. Para vengar la afrenta, los príncipes griegos se disponen a asaltar la ciudad de Troya. El objetivo de los problemas es crear los elementos que van a tomar parte en la batalla, incluyendo el famoso caballo de Troya.

Problema 1 (2 puntos)

En este problema se crearán la clase `Guerrero` y las clases derivadas `Troyano` y `Griego`.

- a) **(0,1 punto)** Crear la clase abstracta `Guerrero`, que deberá poder serializarse. Deberá contar con los siguientes atributos, no accesibles por ninguna otra clase:
- nombre, de tipo `String`, que no podrá cambiar una vez le hemos dado valor.
 - edad, fuerza, de tipo `int`.
 - herido, muerto de tipo `boolean`.
- b) **(0,1 punto)** Crear dos métodos públicos, uno para acceder al atributo `edad` y otro para darle valor. Suponer que existen métodos equivalentes para el resto de atributos.
- c) **(0,3 puntos)** Crear dos métodos, accesibles solamente por sus clases derivadas, `static boolean comprobarEdad (int e)` y `static boolean comprobarFuerza (int f)` que calculen si `edad` y `fuerza` están dentro de los siguientes rangos: `edad` entre 15 y 60 (ambos inclusive), `fuerza` entre 1 y 10 (ambos inclusive). Devolverán `true` si son correctos y `false` en caso contrario.
- d) **(0,8 puntos)** Crear tres constructores para la clase `Guerrero`:
- El primero recibirá valores para todos los atributos, excepto `herido` y `muerto`, que obviamente serán falsos. Deberá comprobar que los valores dados son válidos y en caso contrario poner como `edad` 25 y como `fuerza` 5.
 - El segundo, que deberá utilizar el primero, no recibirá ningún valor y creará un guerrero cuyo nombre sea `guerreroX` y `edad` y `fuerza` valgan 15 y 1 respectivamente.
 - El tercero, recibirá un objeto de tipo `Guerrero` y un nombre y creará una copia del guerrero pasado con el nuevo nombre.
- e) **(0,2 puntos)** Crear el siguiente método público y abstracto:
- `boolean retirarse ()`
- f) **(0,2 puntos)** Crear la clase `Troyano`, que hereda de `Guerrero` y que no tiene ningún nuevo atributo. Crear un constructor para esta clase que reciba el nombre, `edad` y `fuerza` del `Troyano` y utilice alguno de los constructores de la clase `Guerrero`.
- g) **(0,3 puntos)** Implementar los métodos necesarios para que la clase `Troyano` no sea abstracta, teniendo en cuenta que:
- Los métodos devolverán `true` si se ha podido realizar la acción y `false` si no.
 - Los troyanos no pueden retirarse.
- h) **(0,2 Puntos)** Crear la clase `Griego`, que hereda de `Guerrero` y que no tiene ningún nuevo atributo. Los métodos y constructores de esta clase serán similares a los de `Troyano` (no hace falta repetirlos), salvo el método `retirarse`, ya que los griegos sí lo pueden hacer, siempre que estén heridos y, evidentemente, no estén muertos. Implementar este método.

Problema 2 (4 puntos)

Este problema creará la clase para representar al Caballo de Troya. Aunque es continuación lógica del anterior se puede realizar independientemente.

- a) **(0,2 puntos)** Crear la clase `Caballo` que tendrá como atributos públicos:
- `capacidad`, de tipo `int`, representa el número de guerreros griegos que puede haber dentro del caballo. No se podrá cambiar una vez le hemos dado valor.
 - `ocupacion`, de tipo `int`, representa el número actual de griegos en el caballo.
 - `ocupantes`, array de objetos de la clase `Guerrero`.
- b) **(0,8 puntos)** Crear dos constructores para la clase `Caballo`:
- el primero recibirá un array de `Guerreros`, deberá comprobar que todos los ocupantes son `Griegos`, en caso contrario se creará un `Caballo` sin ocupantes con capacidad 100 (en caso positivo, suponer que la capacidad es la del número de `Guerreros` que se ha pasado como parámetro)
 - el segundo recibirá un único `Guerrero` y la capacidad. También deberá comprobar que el `Guerrero` es `Griego`, en caso contrario se creará un `Caballo` sin ocupantes.
- c) **(0,4 puntos)** Crear el método `void ordenar()` que ordene descendentemente el array de ocupantes según su fuerza utilizando el algoritmo de selección directa.
- d) **(0,4 puntos)** Crear un método `int buscar (String nombre)` que busque por su nombre un guerrero dentro del array `ocupantes` ordenado según el método anterior y devuelva la posición en que está ó -1 si no está.
- e) **(0,4 puntos)** Definir la interfaz `PuedeMontarse` que especifica que se puede montar en los objetos de tipo `Caballo`. Tendrá dos métodos:
- `int montar (Guerrero g)` que monta un `Guerrero` en el `Caballo` y devuelve el número de ocupantes que hay actualmente ó -1 si el `Caballo` ya está lleno. Deberá comprobar que sólo los griegos puedan montar en el `Caballo` (si se intenta montar un `Troyano` devolverá -2).
 - `void desmontar ()` que desmonta todos los `Guerreros` del `Caballo`.
- f) **(0,9 puntos)** Realizar los cambios oportunos en la clase `Caballo` para que implemente la interfaz `PuedeMontarse`.
- g) **(0,3 puntos)** Crear un método `void guardar (String fichero)` que guarde en un fichero los nombres de todos los `Guerreros` montados en el `Caballo`.
- h) **(0,3 puntos)** Crear un método `String [] leer (File fichero)` que lea los nombres de los `Guerreros` guardados en un fichero y los devuelva en un array de `String`.
- i) **(0,3 puntos)** Crear una clase `Usuaría`, que en su método `main` cree un `Troyano`, un `Caballo` y un `Griego`, y monte al soldado `Griego` en el `Caballo`.