Apellidos	*Nombre*	
Firma	*NIA*	*Grupo*

EXAMEN PROGRAMACIÓN **15 de Junio de 2007** **INGENIERÍA INFORMÁTICA** **Primera parte: Cuestiones** **1,5 horas**	UNIVERSIDAD CARLOS III DE MADRID

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

a) Un objeto de una clase A que hereda de otra clase B siempre puede acceder a todos los atributos heredados de B.
b) Una clase abstracta puede no tener ningún método abstracto.

a) Falso. Dependerá de la visibilidad de los atributos de la clase B, si por ejemplo son privados, no podrá acceder.
b) Verdadero. Es posible declarar una clase como abstracta aunque no tenga ningún método abstracto, pero no al revés, es decir, si una clase tiene al menos un método abstracto tendrá que ser abstracta.

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

a) Si utilizamos un objeto de la clase FileOutputStream para escribir datos en un fichero y el fichero no existe, se crea automáticamente.
b) Una clase puede tener varios métodos con el mismo nombre.

a) Verdadero. El constructor de la clase FileOutputStream recibe como parámetro un fichero sobre el que escribir y si no existe, intenta crearlo.
b) Verdadero. A esto se le conoce como métodos sobrecargados y es válido siempre que al menos uno de los parámetros de entrada de cada método sea de distinto tipo o su número sea distinto. Ejemplo, podríamos tener los siguientes métodos, todos con el mismo nombre:

```
int metodo1(int a, int b) {…}
int metodo1(int a, float b) {…}
int metodo1(int a) {…}
int metodo1 (float a) {…}
int metodo1 (String a, double b, short c) {…}
```

Pregunta 3 (0,5 puntos).- Explicar en qué consiste el método de ordenación por selección directa y mostrar paso a paso cómo se ordenaría la lista (8,3,5,2,1,9,7,6) usando este método.

Para ordenar de menor a mayor por el método de selección directa buscamos el elemento de menor valor y lo intercambiamos con el primero de la lista. A continuación buscamos el segundo de menor valor y lo intercambiamos con el segundo de la lista y así sucesivamente.

El proceso sería así:
(8,3,5,2,1,9,7,6)
(1,3,5,2,8,9,7,6)
(1,2,5,3,8,9,7,6)
(1,2,3,5,8,9,7,6)
(1,2,3,5,6,9,7,8)
(1,2,3,5,6,7,9,8)
(1,2,3,5,6,7,8,9)

Pregunta 4 (0,5 puntos).- Dadas las siguientes clases:

```
public class ClaseA {
      private int atributo1;
      public ClaseA (int a){
                  atributo1=a;          }
      public ClaseA()
            {this(0);}
      public String toString (){
            return "objeto A "+atributo1;
      }
}
public class ClaseB extends ClaseA{
      public ClaseB (int a){
            super(a);  }
      public ClaseB(){}
      public String toString(){
            return "objeto B "+super.toString();
      }
}
public class ClaseC extends ClaseA{
      public ClaseC (int a){
            super(a);  }
      public ClaseC (){
            super();
      }
}
```

Y el método:

```
public static void metodo1(ClaseA objeto1){
            System.out.println(objeto1.toString());
}
```

Explicar cuál será el resultado de las siguientes invocaciones:

```
ClaseA objeto1 = new ClaseA(1);
ClaseB objeto2 = new ClaseB(2);
ClaseC objeto3 = new ClaseC(3);

metodo1(objeto1);
metodo1(objeto2);
metodo1(objeto3);
```

El método1 recibe como parámetro un objeto de la `ClaseA`, pero gracias al polimorfismo es compatible con objetos de la `ClaseB` y `ClaseC` ya que ambas heredan de A. El método llama al `toString()` del objeto y lo imprime por pantalla.
En la primera llamada invocamos al método con un objeto de la `ClaseA`, por lo que se llama al `toString()` y se imprime "`objetoA 1`"
En la segunda llamada lo invocamos con un objeto de la `ClaseB` (que gracias al polimorfismo es también un objeto de la `ClaseA`) y se llama otra vez al método `toString()`. Pero como el objeto es realmente de la `ClaseB` y ha sobrescrito el método, es al nuevo método al que se llama, por lo que se imprime "`objeto B objeto A 2`" (ya que el método sobrescrito hace una llamada al del padre)
En el tercer caso, se llama al método con un objeto de la `ClaseC`. Como `ClaseC` no sobrescribe el método `toString()`, utiliza el del padre, por lo que imprime "`objeto A 3`"

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

```
public class Clase1 {
      private int atr1;
      private String atr2;

      public Clase1 (int a, String b){
            atr1=a;
            atr2=b;
      }
      public Clase1(){
            this(-1,"vacío");
      }
}

public class Clase2 extends Clase1 {
      boolean atr3;
}
```

Añadir los métodos y constructores necesarios en `Clase1` y `Clase2` para que el resultado por pantalla de ejecutar el siguiente código:

```
Clase2 objeto = new Clase2(3,”hola”,true);
System.out.println(objeto.toString());
```

Sea:

```
Atributo 1 = 3
Atributo 2 = hola
Atributo 3 = true
```

En primer lugar hay que crear un constructor para `Clase2` que reciba valores para los tres parámetros de esta clase (2 heredados y 1 nuevo). Un ejemplo sería:

```
      public Clase2 (int a, String b, boolean c){
            super(a,b);
            atr3 = c;
      }
```

Hay que usar `super()` porque los atributos heredados se han definido como privados y no tenemos acceso directo a ellos, hay que darles por lo tanto valor con el constructor del padre.
También sería conveniente incluir un constructor por defecto:

```
      public Clase2 (){}
```

Ahora hay que hacer un método `toString`. No se puede hacer que se imprima todo directamente en `Clase2`, porque los atributos heredados son privados y no podemos acceder a ellos. La opción más elegante es crear un método `toString` en `Clase1`:

```
public String toString(){
      return "Atributo 1 = "+atr1+'\n'+"Atributo 2 = "+atr2+'\n' }
```

Y luego sobrescribirlo en `Clase2`, llamando al padre:

```
public String toString (){
            return super.toString()+"Atributo 3 = "+atr3;    }
```

También se podrían añadir métodos `getAtr1` y `getAtr2` en `Clase1` y llamarlos desde el `toString` de `Clase2`.

Pegunta 6 (0,5 puntos).- Dado el siguiente código java:

```
public static void metodo1(){
    String [] palabras = new String[4];
    palabras[1] = "palabra1";
    palabras[2] = "palabra2";
    palabras[3] = "palabra3";
    try{
        System.out.println(" antes del for ");
        for (int i=1; i<palabras.length; i++){
            System.out.println(palabras[i%3]);
        }
        System.out.println(" despues del for ");
    }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(" todo bien? ");
    }
}
```

Explicar cuál sería la salida por pantalla al invocar dicho método.

Tras inicializar la variable palabras, entraríamos en el try. Lo primero que mostraría sería " antes del for ". Dentro del for, el cual va de 1 a 3 inclusive, mostraríamos consecutivamente el valor de palabras[1], palabras[2] y palabras[0], lo que se corresponde con "palabra1", "palabra2" y null. Al salir del for se mostraría "despues del for " y como no se ha lanzado ninguna excepción (imprimir null no produce excepciones) , sólo quedaría mostrar " todo bien? " del apartado finally.
El resultado por pantalla es por lo tanto:

```
 antes del for
palabra1
palabra2
null
 despues del for
 todo bien?
```

Pregunta 7 (0,5 puntos).- Dado el siguiente código java:

```
public static void leer(String nombre){
        String linea;
        int x = 0;
        int y = 0;
        try {
            FileReader fr = new FileReader(nombre);
            BufferedReader br = new BufferedReader(fr);
            linea = br.readLine();
            x = Integer.parseInt(linea);
            for (int i=0; i<= x; i++){
                linea = br.readLine();
                System.out.println(y+=Integer.parseInt(linea));
            }
             System.out.println(“leido”);
             br.close();
        }catch (NumberFormatException e){
            System.out.println("Number Format Exception");
        }catch (IOException e){
            System.out.println("Exception");
        }finally {
            System.out.println("y = "+y);
        }
        System.out.println("adios");
    }
```

Y siendo el contenido del fichero “prueba.txt” el siguiente:

```
4
0
6
1
3
hola
5
```

<u>Explicar</u> cuál sería la salida por pantalla al invocar `leer(“prueba.txt”)`

Se toma el primer entero del fichero y se leen siguientes (4+1) líneas. Lee y muestra una a una (“0”, “6”, “7”, 10”) hasta el “hola” que no puede pasar a entero por lo que lanza una excepción. La excepción es de tipo NumberFormatException por lo que se muestra por pantalla “Number Format Exception”, seguido por el mensaje que aparece en finally: “y = 10”. Por último se muestra el “adios”.

0
6
7
10
Number Format Exception
y = 10
adios

Pregunta 8 (0,5 puntos).- Dado las siguientes clases Java:

```
public abstract class ClaseUno {
      public int atr1;
      private int atr2;
      public ClaseUno (int a, int b){
            atr1= a;
            atr2= b;
      }
      public ClaseUno (){}
      //método que suma los dos atributos
      public abstract int suma();
      //método que resta los dos atributos
      public abstract int resta();
}

public abstract class ClaseDos {
      public abstract int multiplica ();
      public abstract int divide ();
}
```

a) En qué deberíamos convertir ClaseDos para que fuera posible que otra clase denominada ClaseTres, heredara los métodos de ClaseUno y ClaseDos (sin que ClaseDos sea hija de ClaseUno o viceversa)

b) Escriba la cabecera de ClaseTres una vez hecho este cambio.

c) ¿Qué se debería hacer en ClaseTres para que se puedan crear objetos de dicha clase?

a) `ClaseDos` no puede ser una Clase, porque en Java no se permite la herencia múltiple, sino que tiene que ser una interfaz. De este modo su cabecera quedaría:

```
public interface ClaseDos() {…}
```

b) La cabecera de `ClaseTres`, una vez convertida `ClaseDos` en una interfaz sería:

```
 public abstract ClaseTres extends ClaseUno implements
ClaseDos
```

c) `ClaseTres` al heredar métodos abstractos de `ClaseUno` e implementar la interfaz `ClaseDos`, que también tiene métodos abstractos, debe implementar esos métodos o declararse como abstracta (tal y como se muestra en la cabecera de b). Como no se pueden crear objetos de una clase abstracta, la única solución sería implementar los 4 métodos heredados. También sería conveniente, aunque no obligatorio, crear constructores para la `ClaseTres`..

EXAMEN PROGRAMACIÓN **15 de Junio de 2007** **INGENIERÍA INFORMÁTICA** **Segunda parte: problemas** **2,5 horas**	UNIVERSIDAD CARLOS III DE MADRID

Se pretende construir un programa que sirva para gestionar una biblioteca. Para ello, a lo largo de tres problemas se crearán las clases necesarias para representar los libros y la biblioteca, ordenarlos, buscarlos, guardarlos en un archivo y restaurarlos.

Problema 1 (2 puntos)

En este problema se implementará la clase `Libro`. Para ello, se pide lo siguiente:

a) **(0,75 puntos)** Crear una clase `Libro` para representar cualquier libro existente en la biblioteca. La clase debe cumplir las siguientes características:
- Pertenecer al paquete `examenjunio`.
- Dar la posibilidad (sin implementarla) de que se puedan serializar objetos de esta clase.
- Tener los siguientes atributos, que no deben ser accesibles por ninguna otra clase:
  - `titulo`: Indica el título del libro
  - `descripcion`: Contiene una breve descripción del libro
  - `prestado`: Indica si el libro actualmente se encuentra prestado o está disponible en la biblioteca.
- Tener un atributo visible por todas las clases llamado `tipoDeLibro`, que contiene un número indicativo del tipo del libro. Los posibles tipos para un libro estarán definidos como constantes públicas comunes para todos los objetos de la clase, y son los siguientes:
  - `TIPO_NOVELA`: Valor 1
  - `TIPO_RELATOS`: Valor 2
  - `TIPO_POESIA`: Valor 3
- Tener un atributo visible por cualquier otra clase y común a todos los objetos `Libro`, denominado `contador`, que cuente cuantos libros se han creado.
- Tener un atributo visible por cualquier otra clase llamado `identificador`, que es un número único para cada libro. Al crear un libro se le asignará un identificador consecutivo partiendo del número 100 (el primer libro que se cree tendrá el número 100, el siguiente el 101, y así sucesivamente).

b) **(0,75 puntos)** Crear los siguientes métodos de la clase Libro
- Crear un constructor que reciba como parámetros el título, la descripción y el tipo del libro. Al crearse un libro nuevo no se encuentra prestado.
- Crear un constructor (que será utilizado para los libros de poesía), que utilice el constructor anterior y que sólo reciba el título y la descripción del libro.
- Crear métodos que devuelvan el valor de los atributos `titulo`, `descripción`, y `prestado`.
- Crear un método que permita modificar el valor del atributo `prestado`, recibiendo como parámetro el nuevo valor.

c) (**0,25 puntos)** Crear una interfaz llamada `Ordenable`, con las siguientes características:
- Pertenecer al paquete `examenjunio`.
- Tener un método público denominado `getOrden` que no recibirá parámetros y devolverá un número.

d) (**0,25 puntos)** Realizar las modificaciones necesarias a la clase `Libro` para que cumpla la interfaz `Ordenable`. El orden de un libro será igual a su identificador.

Problema 2 (2,5 puntos)

Aunque este problema es continuación lógica del primer problema, es posible realizarlo sin haber hecho el problema anterior.

a) **(1 puntos)** Crear una clase denominada `Biblioteca`, que cumpla las siguientes características.
   - Pertenecer al paquete `examenjunio`.
   - Tener un atributo denominado `listaLibros` que contendrá un array de objetos `Libro` representando los libros de la biblioteca.
   - Tener un constructor que recibe como parámetro la lista de libros inicial de la biblioteca.
   - Tener un método añadir`Libro` que recibe como parámetro un libro, y lo añade al final de la lista de libros actuales.

b) **(0,5 puntos)** Modificar la clase `Biblioteca` para incluir un método denominado `ordenarLibros` que ordenará utilizando el método de selección directa la lista de libros contenida en el atributo `listaLibros` según el valor devuelto por su método `getOrden`. Si no se ha hecho el problema anterior, asumir que en la clase `Libro` existe un método llamado `getOrden` que no recibe parámetros y devuelve un número por el que ordenar los libros.

c) **(0,75 puntos)** Dado el siguiente método de la clase `Biblioteca`:

```
public Libro busquedaBinaria(int id) {
 return busquedaBinariaRecursiva(id, 0, listaLibros.length-1);
    }
```

Asumiendo que la lista de libros está ordenada por su identificador, modificar la clase `Biblioteca` para implementar el método denominado `busquedaBinariaRecursiva` que permite buscar un libro dado su identificador, según el método de búsqueda binaria. Es necesario que este método sea implementado de manera recursiva.

d) **(0,25 puntos)** Modificar la clase `Biblioteca` para incluir un método público denominado `estaPrestado`. Dicho método recibirá como parámetro un identificador de libro, y devolverá `true` si el libro se encuentra prestado, o `false` si el libro no se encuentra prestado o no pertenece a la biblioteca. Para buscar el libro se utilizará el método `busquedaBinaria` del apartado anterior.

Problema 3 (1,5 puntos)

Aunque este problema es continuación lógica del primer problema, es posible realizarlo sin haber hecho el problema anterior.

a) **(0,5 puntos)** Modificar la clase `Biblioteca` para incluir un método `guardarLibros` que reciba como parámetro el nombre de un fichero, y escriba la lista de libros de la biblioteca a dicho fichero (serializándola). El método deberá gestionar las posibles excepciones que se produzcan.

b) **(1 puntos)** Modificar la clase Biblioteca para incluir un método `leerLibros` que reciba como parámetro el nombre de un fichero y lea la lista de libros de la biblioteca de dicho fichero. Asegurarse de que una vez leída la lista de libros del fichero, si quisiésemos crear nuevos libros éstos tomarían los valores consecutivos correctos para su campo `identificador` (Ej: Si el mayor de los identificadores de los libros leídos del fichero es el 154, el siguiente libro que se creará tendrá el identificador 155. La lista de libros no tiene por qué estar ordenada). El método deberá gestionar las posibles excepciones que se produzcan.

Si no se ha hecho el problema anterior, asumir que en la clase `Biblioteca` existe un atributo denominado `listaLibros` que contiene un array de objetos `Libro`, que los libros tienen un atributo entero denominado `identificador` y otro que lleva la cuenta de libros creados, denominado `contador`, y existe el método `ordenarLibros` creado en el apartado b) del problema anterior.

Problema 1

Clase Libro

```
package examenjunio;
import java.io.*;
public class Libro implements Serializable,Ordenable {
      private String titulo,descripcion;
      private boolean prestado;
      public int tipoDeLibro;
      public  static  final  int  TIPO_NOVELA  =  1,  TIPO_RELATOS=2,
TIPO_POESIA = 3;
      public int identificador;
      public static int contador;

      public Libro (String t, String d, int tipo){
            titulo= t;
            descripcion = d;
            prestado = false; // no hace falta, es el valor por
                              defecto
            tipoDeLibro = tipo;
            identificador = id+100;
            id++;
      }

      public Libro (String t, String d){
            this(t,d,TIPO_POESIA);
      }

      public String getTitulo (){
            return titulo;
      }
      public String getDescripcion (){
            return descripcion;
      }
      public boolean getPrestado (){
            return prestado;
      }
      public void setPrestado (boolean p){
            prestado=p;
      }

      public int getOrden (){
            return identificador;
      }

}
```

Interfaz Ordenable

```
package examenjunio;

public interface Ordenable {
      public abstract int getOrden();
}
```

Problema 2

```
package examenjunio;
import java.io.*;
public class Biblioteca {
      private Libro listaLibros [];

      public Biblioteca (Libro [] lista){
            listaLibros = lista;
      }

      public void añadirLibro (Libro l){
            Libro [] aux = new Libro [listaLibros.length+1];
            for (int i=0; i<listaLibros.length;i++) {
                  aux[i]=listaLibros[i];
            }
            aux[listaLibros.length]=l;
            listaLibros = aux;
      }

      public void ordenarLibros (){
            //definimos las variables auxiliares
            int posicionMenor;
            Libro valorMenor;
            int ultimo = listaLibros.length-1;
            //buscamos sucesivamente el menor elemento de la lista
            for (int i=0; i<=ultimo; i++) {
                //el menor empieza siendo el valor de la posición i
                posicionMenor=i;
                valorMenor=listaLibros[i];
                //Buscamos el menor de los restantes recorriendo la
                  //lista con el índice j
                for (int j=i+1; j<=ultimo; j++) {
                  //Si el elemento lista[j] es menor que el
                  //valorMenor, pasa a ser el valorMenor
                  if
                  (listaLibros[j].getOrden()<valorMenor.getOrden()) {
                      valorMenor=listaLibros[j];
                      posicionMenor=j;
                  }// fin if
                }// fin for (j)
                //intercambiamos el menor por el de la posición i
                listaLibros[posicionMenor]=listaLibros[i];
                listaLibros[i]=valorMenor;
                //y seguimos con el bucle
            } //fin for (i)
      }
      public Libro busquedaBinaria(int identificador) {
         return busquedaBinariaRecursiva(identificador, 0,
                                              listaLibros.length-1);}
```

```
     public Libro busquedaBinariaRecursiva(int id, int prim, int
ult){
          int central = (prim+ult)/2;
          if (primero>ultimo)  return null;
          if (id==listaLibros[central].getOrden()) return
               listaLibros[central];
          else if (id>listaLibros[central].getOrden())
               busquedaBinariaRecursiva(id,central+1,ult);
          else return busquedaBinariaRecursiva(id,prim,central-1);
          return null;
     }

     public boolean estaPrestado (int id){
          Libro aux =busquedaBinaria(id);
          if (aux==null) return false;
          return aux.getPrestado();
     }
```

Problema 3

```
public void guardarLibros(String nombreFichero) throws Exception {
    ObjectOutputStream oos = null;
    FileOutputStream fos = null;
    try {
        fos = new FileOutputStream(nombreFichero);
        oos = new ObjectOutputStream(fos);
        oos.writeObject(listaLibros);
    } catch(Exception e) {
        System.out.println("Excepción intentando guardar libros");
    } finally {
        if(fos != null) {
            fos.close();
        }
    }
}

public void cargarLibros(String nombreFichero) {
    ObjectInputStream ois = null;
    FileInputStream fis = null;
    try {
        fis = new FileInputStream(nombreFichero);
        ois = new ObjectInputStream(fis);
        listaLibros = (Libro[]) ois.readObject();
        // Recorre la lista de libros para buscar el último
           identificador
        int max = 0;
        for(int i=0; i<listaLibros.length; i++) {
            if(listaLibros[i].identificador > max) {
                max = listaLibros[i].identificador;
```

```
            }
        }
        Libro.contador = max – 99;
    } catch(IOException ioe) {
        System.out.println("Excepción intentando restaurar
libros");
    } catch(ClassNotFoundException e) {
        System.out.println("Excepción intentando restaurar
libros");
    } finally {
        if(fis != null) {
            fis.close();
        }
    }
}
```