2º EXAMEN DE PRÁCTICAS DE PROGRAMACIÓN 11/06/2015 Duración: 1 hora

(Nota: El examen se evalúa sobre 10 puntos, si bien su peso específico en la nota final de la asignatura es de 1,20 puntos)

A rellenar por el profesor

Escribe tu nombre y apellidos, y el código de tu grupo de prácticas, aquí:

Nombre y apellidos:		Grupo:	

Pregunta 1. **(3 puntos)**

En la clase **GestorBanco** de la práctica 4 se ha implementado el siguiente método:

```
public static double saldoTrasRetirarFondo(Cuenta c, double cantidad) {
      c.retirar(cantidad);
      return c.getSaldo();
}
```

Sin embargo, el compilador da el siguiente mensaje de error:
(unreported exception SaldoInsuficienteException; must be caught or declared to be thrown)

Se pide: Modificar el método **saldoTrasRetirarFondo** para realizar el tratamiento de excepciones (la captura) tanto si la cuenta es **null** como si no hay saldo suficiente. En ambos casos, se deberá mostrar un mensaje de error indicando el motivo, y devolver como resultado 0.

```
public static double saldoTrasRetirarFondo(Cuenta c, double cantidad) {
   try {
      c.retirar(cantidad);
      return c.getSaldo();
   } catch (NullPointerException e) {
      System.out.println("No existe la cuenta");
   } catch (SaldoInsuficienteException e) {
      System.out.println("No hay suficiente saldo");
   }
   return 0;
}
```

Pregunta 2. **(2 puntos)**

En el proyecto de la práctica 4, se ha añadido el siguiente método, parcialmente implementado, para leer un fichero binario de cuentas (argumento **a**) y generar otro fichero binario de cuentas (argumento **b**) que contenga solamente las cuentas, leídas del primer fichero, de saldo superior a un millón de euros:

```
public void generarFicheroMillonarios(String a, String b) {
   ObjectInputStream f = new ObjectInputStream(new FileInputStream(a));
   ObjectOutputStream g = new ObjectOutputStream(new FileOutputStream(b));
   try {
      while (true) {
         // bucle: lectura de fichero a, escritura en fichero b
         // bucle "infinito", concluye al generarse EOFException
      }
   } catch (EOFException e) {
      f.close(); g.close();
   }
}
```

Se desea completar el método teniendo en cuenta que:

- Se sabe que el fichero de entrada contiene solo objetos de la clase **`Cuenta`**, que corresponden a cuentas válidas (por tanto, no será necesario verificarlos).
- Se desconoce el número de objetos a leer: en cada iteración, se debe leer un objeto del fichero de entrada y, si cumple la condición relativa al saldo, escribir el objeto en el fichero de salida.
- Se deben propagar las excepciones **`IOException`** y **`ClassNotFoundException`**.

Se pide: Especificar qué código añadir, y dónde añadirlo, para cumplir con la funcionalidad descrita.

- 1°) Añadir, en el perfil, la cláusula throws:

```
throws IOException, ClassNotFoundException
```

- 2°) Especificar el bloque de código del bucle while (true):

```
Cuenta c = (Cuenta) f.readObject();
if (c.getSaldo() > 1e+6)
     g.writeObject(c);
```

A rellenar por el profesor

Pregunta 3. **(2 puntos)**

A continuación, se muestra una implementación del método **`insOrd(String, int)`** de la clase **`Concordancia`** que contiene errores de ejecución:

```
 0   private void insOrd(String pal, int numLin) {
 1      NodoCnc aux = prim, ant = null;
 2      while (aux != null && aux.palabra.compareTo(pal) > 0) {
 3         aux = aux.siguiente;
 4         ant = aux;
 5      }
 6      if (aux != null && aux.palabra.compareTo(pal) == 0)
 7         aux.numLins.encolar(numLin);
 8      else {
 9         NodoCnc nuevo = new NodoCnc(pal, numLin, aux);
10         talla++;
11         ant.siguiente = nuevo;
12      }
13   }
```

Las estructuras de datos **`Concordancia`** y **`NodoCnc`**, vistas en prácticas, tienen los atributos que se indican en la pregunta 4 (véase página siguiente).

Se pide: Identificar cada error y dar el código correspondiente a su solución.

- 1°) En la línea 2, el operador de la comparación de Strings debe ser <

```
2      while (aux != null && aux.palabra.compareTo(pal) < 0) {
```

- 2°) Las líneas 3 y 4 deben intercambiarse

```
3      ant = aux;
4      aux = aux.siguiente;
```

- 3°) La línea 11 debe sustituirse por el siguiente código:

```
11      if (ant == null) prim = nuevo else ant.siguiente = nuevo;
```

A rellenar por el profesor

Pregunta 4. **(3 puntos)**

En el proyecto de la práctica 5, se desea poder eliminar de la concordancia toda la información relativa a una línea dada del texto.

Supongamos ya implementado, en la clase **ColaIntEnla**, un método de instancia con el siguiente perfil: **public void eliminar(int n)**, que borra todas las apariciones de **n** en la cola de enteros que lo invoque.

Se recuerda que las estructuras de datos **Concordancia** y **NodoCnc**, vistas en prácticas, tienen los atributos siguientes:

Concordancia	**NodoCnc**
private NodoCnc prim;	**String palabra;**
private int talla;	**ColaIntEnla numLins;**
private boolean esOrd;	**NodoCnc siguiente;**
private String separadores;	

Se pide: Implementar, en la clase **Concordancia**, un método que reciba un entero **n** que representa un número de línea y borre, en cada uno de sus **NodoCnc**, todas las apariciones del valor **n** de su cola de números de línea y si, como resultado de este borrado, el número de líneas en el **NodoCnc** es 0, borre también dicho **NodoCnc**, modificando adecuadamente los atributos del objeto **Concordancia**.

```
public void eliminar(int n) {
   NodoCnc aux = prim, ant = null;
   while (aux != null) {
      aux.numLins.eliminar(n);
      if (aux.numLins.talla() == 0) {
         if (ant != null) ant.siguiente = aux.siguiente;
         else prim = prim.siguiente;
         talla--;
      } else ant = aux;
      aux = aux.siguiente;
   }
}
```

A rellenar por el profesor