EXAMEN DE PRÁCTICAS DE PROGRAMACIÓN 23/06/2015 Duración: 1 Hora

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. **(2 puntos)**

Se quiere completar la implementación de un método recursivo, llamado **`esInicioYFin`**. Este método, dados **a** y **b** de la clase **`String`**, determina si **a** es prefijo directo y sufijo inverso de **b** (es decir, si los caracteres de **a** se pueden leer, de izquierda a derecha, en las posiciones iniciales de **b**, y también se pueden leer, de derecha a izquierda, en las posiciones finales de **b**). Como ejemplos de uso de dicho método, considérense las siguientes líneas de código:

```
boolean x = esInicioYFin("lasa","lasaladelasal");   // x ← true
boolean y = esInicioYFin("des","desiertodelased");  // y ← true
boolean z = esInicioYFin("los","losdiasdesol");     // z ← true
boolean v = esInicioYFin("susi","susaludasusta");   // v ← false
boolean w = esInicioYFin("dos","dosdiasmalos");     // w ← false
```

Se dispone de la siguiente implementación <u>parcial</u> del método:

```
0     /** precondición: 2 * a.length() <= b.length() */
1     public static boolean esInicioYFin(String a, String b) {
2       if (a.length() == 0) return true;
4       else return a.charAt(0) == ...
5            && a.charAt(0) == ...
6            && esInicioYFin( ...
7     }
```

<u>Se pide</u>: Completar la implementación, añadiendo el código necesario en el caso general (en donde se indican los puntos suspensivos).

```
public static boolean esInicioYFin(String a, String b) {
   if (a.length() == 0) return true;
   else return a.charAt(0) == b.charAt(0)
        && a.charAt(0) == b.charAt(b.length()-1)
        && esInicioYFin(a.substring(1), b.substring(1,b.length()-1));
}
```

Pregunta 2. **(2 puntos)**

Considera la siguiente tabla con los tiempos de ejecución (en milisegundos) de dos algoritmos que resuelven el mismo problema:

# Talla n	Algoritmo A TA(n)	Algoritmo B TB(n)
1000	10148.00	397.81
2000	19753.00	1501.83
3000	25327.00	3312.83
4000	34552.00	5820.27
5000	44145.00	9029.67
6000	51902.00	12936.26
7000	61002.00	17543.27
8000	69827.00	22835.75
9000	77687.00	30385.04
10000	86409.00	36873.96
11000	94799.00	43157.29
12000	103808.00	51282.19
13000	112750.00	59839.54
14000	124301.00	69399.81
15000	131841.00	81848.08
16000	143048.00	90444.16
17000	154218.00	102002.64
18000	166435.00	116535.31
19000	173328.00	127247.52
20000	182737.00	143252.33

Se pide: Contestad a las siguientes cuestiones justificando la respuesta:

a) ¿Cuál es el coste asintótico que mejor se ajusta a los datos de la columna **Algoritmo A**?
b) ¿Cuál es el coste asintótico que mejor se ajusta a los datos de la columna **Algoritmo B**?
c) ¿Qué algoritmo tiene un comportamiento asintótico mejor?

a) El coste asintótico que mejor se ajusta a los datos de la columna **Algoritmo A** es un coste lineal, pues a medida que la talla del problema se multiplica por dos el coste temporal también se multiplica por dos. Si f(2n)/f(n) = 2n/n = 2 entonces f(n) es lineal.

b) El coste asintótico que mejor se ajusta a los datos de la columna **Algoritmo B** és un coste cuadrático, dado que el ratio f(2n)/f(n) da un valor muy próximo a 4. Por tanto, si una función f(n) es cuadrática se cumple que f(2n) = 4 f(n), multiplicando por 2 la talla se multiplica por 4 el valor de la función, si por ejemplo la función es f(x)=x*x, se ve claramente que f(2*x) = 4*f(x), dado que f(2*x) = (2*x)*(2*x) = 2*2*x*x = 4*x*x = 4 * f(x).

c) Evidentemente, el algoritmo A tiene un mejor comportamiento asintótico, aunque para valores pequeños de la talla del problema sea más lento que el algoritmo B.

A rellenar por el profesor

Pregunta 3. **(3 puntos)**

A rellenar por el profesor

Considérese el proyecto de la práctica 4, que se ha modificado para permitir un descubierto en cuenta (es decir, que el saldo de una cuenta bancaria pueda ser negativo).

Para ello, en la clase **Cuenta**, se ha añadido un atributo, llamado **admiteDescubierto**, que indica si se autoriza un saldo negativo, y un atributo estático, llamado **DESCUBIERTO_MAX**, con la cantidad máxima de descubierto permitido. Por tanto, los atributos de esta clase ahora son:

```
private static final double DESCUBIERTO_MAX = 1000.0;
private double saldo;
private int numCuenta;
private boolean admiteDescubierto;
```

Además, se ha implementado una clase **DescubiertoExcedidoException**, subclase de **Exception**, para gestionar las situaciones en las que el saldo negativo de la cuenta pudiera exceder el máximo descubierto permitido.

Se pide: Implementar, en la clase **Cuenta**, un método de instancia, llamado **pagarRecibo**, tal que, dado un número real (importe del recibo), realice el cargo del mismo en la cuenta excepto si se dan las siguientes circunstancias:

- Si la cuenta no admite descubierto y el importe del recibo excede el saldo en cuenta, se lanzará la excepción **SaldoInsuficienteException** (con un mensaje adecuado), y ésta se propagará.

- Si la cuenta admite descubierto y el importe del recibo excede el saldo y el descubierto máximo autorizado, se lanzará la excepción **DescubiertoExcedidoException** (con un mensaje adecuado), y ésta se propagará.

```
public void pagarRecibo(double cantidad)
   throws SaldoInsuficienteException, DescubiertoExcedidoException {
   if (!admiteDescubierto && cantidad > saldo)
      throw new SaldoInsuficienteException(
      "¡Recibo impagado porque no hay suficiente dinero en la cuenta!");
   if (admiteDescubierto && cantidad > saldo + DESCUBIERTO_MAX)
      throw new DescubiertoExcedidoException(
      "¡Recibo impagado porque excedería el descubierto autorizado!");
   saldo -= cantidad;
}
```

Pregunta 4. **(3 puntos)**

En el proyecto de la práctica 5, se desea poder obtener, de la concordancia, todas las palabras que aparecen en una línea dada del texto.

Supondremos que en la clase **`ColaIntEnla`**, además de todos los métodos conocidos, se ha implementado un método con el siguiente perfil: **`public boolean contiene(int n)`**, que devuelve verdadero si el valor **`n`** está en la cola de enteros que lo invoque, y devuelve falso en caso contrario.

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

`Concordancia`
```
private NodoCnc prim;
private int talla;
private boolean esOrd;
private String separadores;
```

`NodoCnc`
```
String palabra;
ColaIntEnla numLins;
NodoCnc siguiente;
```

Se pide: Implementar, en la clase **`Concordancia`**, un método que reciba un entero **`n`** que representa un número de línea, y devuelva una cadena de caracteres formada por todas las palabras que aparecen en la línea **`n`** del texto, separadas por comas. El método debe funcionar tanto si la concordancia está ordenada como si no lo está. No importa el orden en el que las palabras aparezcan en la **`String`** resultado.

Ejemplo: dada la concordancia que se muestra en el recuadro de la derecha, si el método es invocado por ese objeto **`Concordancia`**, y recibe el valor 2, devuelve la cadena **"Más, alto, afirmar, "**; y si recibe el valor 4, devuelve **"el, sí, y, "**.

```
Más (2):      1  2
azul (2):     1  1
que (2):      1  3
el (4):       1  3  4  4
alto (1):     2
afirmar (1):  2
amor (1):     3
querer (1):   3
sí (3):       3  4  4
y (2):        4  4
```

```
public String palabrasEnLinea(int n) {
   String s = "";
   for (NodoCnc p = prim; p!=null; p = p.siguiente)
      if (p.numLins.contiene(n)) s += p.palabra + ", ";
   return s;
}
```

A rellenar por el profesor