PRG - Programació. Tema 1 - Recursió

Autoavaluació T1: 50 punts

* Descarrega els fitxers `AutoAvaluacioT1.java` i `TestAAT1.class` al directori `PRG/Tema 1/exercicisT1` del teu disc `W` d'`UPVNET` i obri en *BlueJ* el projecte `exercicisT1`.
* Completa, al fitxer `AutoAvaluacioT1.java`, els mètodes l'enunciat dels quals apareix en aquest document.
* Escriu el nom del/s autor/s en la documentació de la classe.
* Escriu la precondició de cada mètode.
* Comprova que el codi no té errors de compilació i que segueix l'estil de codificació recomanat en Java segons el *Checkstyle* de BlueJ.
* Una vegada resolts els mètodes, executa el `main` de `TestAAT1.class` per tal de comprovar si són correctes.

1. El següent mètode iteratiu, donat un enter $m > 0$, torna una `String` amb la seqüència `1 2 ... m-1 m` , acabada per un caràcter de canvi de línia.

```
/** Precondició: m > 0. */
public static String sequencia(int m) {
    String res = "1 ";
    for (int i = 2; i <= m; i++) { res += i + " "; }
    return res + "\n";
}
```

   Per exemple, si s'executa la crida `sequencia(5)`, torna la `String` següent (finalitzada amb un canvi de línia): "`1 2 3 4 5 `".

   (a) [10 punts] Fent servir el mètode `sequencia(int)`, escriu un mètode recursiu `trgSuperior(int)` que torne una `String` de n línies ($n > 0$) amb seqüències decreixents, formant un triangle. La capçalera del mètode serà la següent:

```
public static String trgSuperior(int n)
```

   Per exemple, si s'executa la crida `trgSuperior(6)`, la `String` resultant serà:

```
1 2 3 4 5 6
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
```

   Nota que un *triangle* com l'anterior, de certa amplària n, pot definir-se recursivament, per a valors de $n > 1$, com:

   - Una seqüència d'amplària n, seguida
   - d'un *triangle* d'amplària $n - 1$.

Solució:

```
/** Torna una String de n línies amb seqüències decreixents,
 *  formant un triangle.
 *  Precondició: n > 0.
 */
public static String trgSuperior(int n) {
    if (n == 1) { return sequencia(1); }
    else { return sequencia(n) + trgSuperior(n - 1); }
}
```

(b) [10 punts] Emprant el mètode `sequencia(int)`, escriu un mètode recursiu `trgInferior(int)` que torne una `String` de n línies ($n > 0$) amb seqüències creixents, formant un triangle. La capçalera del mètode serà la següent:

```
public static String trgInferior(int n)
```

Per exemple, si s'executa la crida `trgInferior(6)`, la `String` resultant serà:

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
1 2 3 4 5 6
```

A l'igual que abans, **nota que** un *triangle* així, de certa amplària n, pot definir-se recursivament, per a valors de $n > 1$, com:

- Un *triangle* d'amplària $n - 1$ tot seguit
- d'una seqüència d'amplària n.

Solució:

```
/** Torna una String de n línies amb seqüències creixents,
 *  formant un triangle.
 *  Precondició: n > 0.
 */
public static String trgInferior(int n) {
    if (n == 1) { return sequencia(1); }
    else { return trgInferior(n - 1) + sequencia(n); }
}
```

2. [15 punts] Siga a un array $\{a_0, a_1, a_2, \ldots, a_{n-1}\}$ de `double` (n = `a.length` > 0), que representa els coeficients d'un polinomi $a(x) = a_0 + a_1 \cdot x + a_2 \cdot x^2 + \cdots + a_{n-2} \cdot x^{n-2} + a_{n-1} \cdot x^{n-1}$.

Escriu un mètode recursiu privat que, donats l'array `a` i un `x` de tipus `double`, calcule $a(x)$. L'estructuració recursiva del problema pot tenir en compte que l'avaluació d'un polinomi $a_0 + a_1 \cdot x + a_2 \cdot x^2 + \cdots + a_{n-2} \cdot x^{n-2} + a_{n-1} \cdot x^{n-1}$ es pot reduir a l'avaluació d'un polinomi de menor grau $a_0 + a_1 \cdot x + a_2 \cdot x^2 + \cdots + a_{n-2} \cdot x^{n-2}$ al que se li ha de sumar el terme $a_{n-1} \cdot x^{n-1}$.

Escriu un mètode públic homònim (**mètode guia o llançadora**) que realitze la **crida inicial** al mètode recursiu anterior.

Solució: l'avaluació d'un polinomi $a(x) = a_0 + a_1 \cdot x + a_2 \cdot x^2 + \cdots + a_{n-2} \cdot x^{n-2} + a_{n-1} \cdot x^{n-1}$ es pot reduir a l'avaluació d'un polinomi de menor grau $a_0 + a_1 \cdot x + a_2 \cdot x^2 + \cdots + a_{n-2} \cdot x^{n-2}$ al que se li ha de sumar el terme $a_{n-1} \cdot x^{n-1}$.

```
/** Calcula a[0] + a[1]*x + ... + a[i]*x^i.
 *  Precondició: a.length > 0, 0 <= i < a.length.
 *  És un recorregut recursiu descendent.
 */
private static double polinomi(double[] a, double x, int i) {
    if (i == 0) { return a[0]; }
    else { return polinomi(a, x, i - 1) + a[i] * Math.pow(x, i); }
}

/** Calcula a[0] + a[1]*x + ... + a[a.length-1]*x^(a.length-1).
 *  Precondició: a.length > 0.
 */
public static double polinomi(double[] a, double x) {
    return polinomi(a, x, a.length - 1);
}
```

Solució: Per evitar calcular potències, es pot usar la regla de Horner.

Donat el polinomi $a(x) = a_0 + a_1 \cdot x + a_2 \cdot x^2 + \cdots + a_{n-2} \cdot x^{n-2} + a_{n-1} \cdot x^{n-1}$ es pot escriure com:

$a(x) = a_0 + x \cdot (a_1 + x \cdot (a_2 + x \cdot (\cdots + x \cdot (a_{n-2} + x \cdot a_{n-1}) \cdots)))$

Es van calculant les successives avaluacions parcials que condueixen al valor de $a(x)$.

$b_{n-1} = a_{n-1}$

$b_{n-2} = a_{n-2} + x \cdot b_{n-1} = a_{n-2} + x \cdot a_{n-1}$

$b_{n-3} = a_{n-3} + x \cdot b_{n-2} = a_{n-3} + x \cdot a_{n-2} + x^2 \cdot a_{n-1}$

...

$b_1 = a_1 + x \cdot b_2 = a_1 + x \cdot a_2 + x^2 \cdot a_3 + \cdots + x^{n-2} \cdot a_{n-1}$

$b_0 = a_0 + x \cdot b_1 = a_0 + x \cdot a_1 + x^2 \cdot a_2 + x^3 \cdot a_3 + \cdots + x^{n-1} \cdot a_{n-1} = a(x)$

```
/** Avaluació parcial i-èsima de la regla de Horner per a a(x):
 *  a[i] + a[i+1]*x + ... + a[n-1]*x^(n-1-i), on n=a.length.
 *  Precondició: a.length > 0, 0 <= i < a.length.
 *  És un recorregut recursiu ascendent.
 */
private static double horner(double[] a, double x, int i) {
    if (i == a.length - 1) { return a[i]; }
    else { return a[i] + x * horner(a, x, i + 1); }
}

/** Calcula a[0] + a[1]*x + ... + a[a.length-1]*x^(a.length-1).
 *  Precondició: a.length > 0.
 */
public static double horner(double[] a, double x) {
    return horner(a, x, 0);
}
```

3. [15 punts] Escriu un mètode recursiu que comprove si dues `String` a i b que tenen la mateixa longitud són simètriques. Dues `String` són simètriques quan el primer element de la primera és igual a l'últim de la segona, i així successivament. Per exemple, les `String` "`HOLA`" i "`ALOH`" són simètriques, mentre que "`HOLA`" i "`ALHA`" no ho són. La capçalera del mètode serà la següent:

```
public static boolean simetriques(String a, String b)
```

Recorda el significat dels següents mètodes de la classe `String` de Java:

- `s.charAt(i)` torna el `char` que ocupa la posició `i` de `s`.
- `s.substring(i)` torna un objecte `String` que representa la substring de `s` formada pels caràcters compresos entre el `i` i el `s.length() - 1`.
- `s.substring(i, j)` torna un objecte `String` que representa la substring de `s` formada pels caràcters compresos entre el `i` i el `j - 1`.

Solució:

```
/** Determina si a i b són simètriques.
 *  Precondició: a.length() == b.length().
 *  És una cerca lineal recursiva combinada.
 */
public static boolean simetriques(String a, String b) {
    if (a.length() == 0) { return true; }
    else {
        int ult = b.length() - 1;
        return a.charAt(0) == b.charAt(ult)
            && simetriques(a.substring(1), b.substring(0, ult));
    }
}
```