Programación I

Recuperatorio Parcial (turno mañana) – 27/11/2019

Ejercicio 1 *(20pts. - 5pts. cada item.)*

Discutir la veracidad de las siguientes afirmaciones, justificando su respuesta.

a) Para invocar un método de instancia desde otra clase se debe mencionar la clase donde fue creado el método.
b) Para que un algoritmo recursivo termine tiene que tener definido al menos un caso base y una llamada recursiva.
c) Un tipo abstracto de datos (TAD) se define especificando cómo se implementan sus operaciones.
d) En una lista doblemente enlazada agregar un elemento al final tiene complejidad O(n), donde n es la longitud de la lista.

Ejercicio 2 *(20pts.)*

Escribir una función recursiva **public static** String intercalarDespuesDe(String s, **char** c, String t) que devuelve un String igual a s en el cual por cada aparición de la letra c se incorporó una letra del String t. Por ejemplo:

- intercalarDespuesDe("banana", 'a', "ooo") debe devolver "baonaonao".
- intercalarDespuesDe("aula", 'l', "hola") debe devolver "aulha".
- intercalarDespuesDe("perro", 'r', "x") debe devolver "perxro".
- intercalarDespuesDe("", 'h', "luz") debe devolver "".
- intercalarDespuesDe("sol", 'o', "") debe devolver "sol".

Se pide resolver **utilizando recursión**. Se pueden dar por hecha la función String resto(String s) que devuelve una cadena igual a s pero sin su primer caracter.

Ejercicio 3 *(35 pts. - item a. 15 pts. - item b. 20 pts.)*

Consideremos las siguientes clases:

```
public class Tienda {
        public Venta[] ventas;
        ...
}

public class Venta {
        public Ticket ticket;
        public Producto[] productos;
        public AsesorDeVenta[] asesores;
        ...
}

public class Producto {
        public String marca;
        public double precio;
        ...
}
```

```
public class Ticket {
        public String nombreCliente;
        public String apellidoCliente;
        public String direccionCliente;
        public double iva;
        ...
}

public class AsesorDeVenta {
        public int legajo;
        public String nombre;
        public String apellido;
        public double promedioDeVentasMensuales;
        ...
}
```

Estas clases modelan una Tienda que comercializa productos químicos. En la tienda tenemos un arreglo de ventas. Cada venta tiene un ticket con los datos del cliente, un arreglo con los productos vendidos en esa transacción, y dada la peligrosidad de los productos químicos, podemos tener uno o varios asesores de venta, almacenados en un arreglo, que brindaron asesoramiento en dicha venta. Para la clase Tienda, se pide:

a) Escribir el método **public** Set<Venta> mejoresVentas(**double** montoMin) que devuelve el conjunto de ventas que tuvieron un monto total más de montoMin de pesos.
b) Escribir el método **public int** ventasConAsesoresExistosos(**int** n) que calcula la cantidad de ventas que contaron con al menos un asesor de ventas con un promedio de n o más ventas mensuales.
c) (*bonus track 20 pts.*) Escribir el método **public** String marcaMasVendida() que devuelve la marca más vendida en la tienda. En caso de existir más de una marca con estas características se puede devolver cualquiera de ellas.

Ejercicio 4 *(25pts.)*

Dadas las clases

```
public class NodoInt {
    int elemento;
    NodoInt siguiente;
}
```

```
public class ListaInt {
    NodoInt primero;
    ...
}
```

Se pide para esta clase escribir el método de instancia **public void** escalerasHastaCero() que modifica la lista agregando nodos para que los elementos queden en escaleras descendentes de números consecutivos hasta 0.
Por ejemplo:

- Si la lista es [3,2,4], la lista debe quedar [3,2,1,0,4,3,2,1,0].
- Si la lista es [1,2,3], la lista debe quedar [1,0,2,1,0,3,2,1,0].
- Si la lista es [5,4,3,2,1], la lista debe quedar [5,4,3,2,1,0].
- Si la lista es [], la lista debe quedar [].

Se pide además que el método implementado **sea de orden lineal**, es decir, $O(n)$ donde n es la cantidad de elementos de la lista. Justificar la complejidad del mismo.

Observación: Se puede suponer que la lista original tiene todos sus elementos mayores a 0 y menores a 10.