Programación I

Parcial (turno mañana) – 2022-11-09

Resolver en tinta, no usar lápiz, ni color rojo. Resolver cada ejercicio en hojas separadas. El examen se aprueba con 40 puntos, y se debe contar con, al menos, 10 puntos en el ejercicio 3 y 10 puntos en el ejercicio 4.

Ejercicio 1 *(15 pts. - 5 pts. cada item.)*

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

a) Los objetos se mantienen en memoria dinámica mientras haya alguna variable que los referencie.
b) El constructor devuelve un boolean que indica si el objeto fue creado exitosamente.
c) Si duplicamos el tamaño de la entrada de un algoritmo de búsqueda binaria el tiempo de ejecución también se duplicará.

Ejercicio 2 *(25 pts.)*

Escribir una función recursiva `public static String camelCase(String s)` que devuelva un `String` similar a `s` pero en formato camelCase. Es decir, debe suprimir los espacios y pasar a mayúscula todos los caracteres precedidos del carácter espacio.

Por ejemplo:

- `camelCase("carpinteria de aluminio")` debe devolver `"carpinteriaDeAluminio"`.
- `camelCase("archivos de texto")` debe devolver `"archivosDeTexto"`.
- `camelCase("el primer dato de la tabla")` debe devolver `"elPrimerDatoDeLaTabla"`.
- `camelCase("calabaza")` debe devolver `"calabaza"`.
- `camelCase("")` debe devolver `""`.

Se pide resolver **utilizando recursión**. Se puede dar por hecha la función `public static String resto(String s)` que devuelve un String 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 Universo {
        public Cuadrante[] cuadrantes;
}

public class Cuadrante {
        public String nombre;
        public Planeta[] planetas;
        public boolean estáEnGuerra;
        public Nave[] naves;
}

public class Planeta {
        public Raza[] razas;
        public int porcentajeDeOxígeno;
        public int porcentajeDeNitrógeno;
        public int temperaturaPromedio;
}
```

```
public class Raza {
        public String nombre;
        public boolean perteneceFederación;
        public int nivelTecnológico;
        public Raza razaEnemiga;
}

public class Nave {
        public int númeroDeRegistro;
        public String nombre;
        public Capitán[] capitanes;
        public boolean estáEnServicio;
}

public class Capitán {
        public int númeroDeIdentificación;
        public String nombre;
        public String apellido;
}
```

Estas clases modelan el universo de la Federación Unida de Planetas.
En el universo tenemos un array con los cuadrantes, donde cada cuadrante es un sector del universo que tiene nombre, un array de planetas que pertenecen a ese cuadrante, un boolean que indica si hay guerras en el sector, y un array con las naves espaciales que pertenecen al cuadrante (una nave pertenece a un único cuadrante).
En cada planeta tenemos un array de razas, ya que en un mismo planeta pueden habitar muchas razas distintas, además tenemos el porcentaje de oxígeno y nitrógeno, y la temperatura promedio del planeta.
Cada raza alienígena, tiene un nombre, un boolean que indica si esa raza pertenece a la Federación, y su nivel tecnológico. Además, cada raza tiene una variable con una raza enemiga (todas las razas en el universo tienen una raza enemiga).
Por último, tenemos naves y capitanes. Cada nave tiene un número de registro, un nombre, un boolean que indica si está en servicio, y de un array de capitanes que incluye todos los capitanes que tuvo cada nave a lo largo de sus años de servicio. Cada capitán tiene un número de identificación, y su nombre y apellido.

Para la clase Universo, se pide:

a) Escribir el método **public** ArrayList<Planeta> planetasConRazasDeNivelSuperior(**int** r) que devuelve la lista de planetas que tienen más de r razas con un nivel tecnológico mayor a 9. La lista no debe tener planetas repetidos.
b) Escribir el método **public int** cantidadDePlanetasConEnemigos() que devuelve la cantidad de planetas que tienen entre sus razas a al menos dos que son enemigas.
c) (*bonus track* 20 pts.) Escribir el método **public** Capitán capitánMásTrabajador() que devuelve el capitán que comandó la mayor cantidad de naves en todos los cuadrantes del universo. Tener en cuenta que un capitán puede haber comandado muchas naves a lo largo de su carrera. En caso de que exista más de un capitán con estas características, se puede devolver cualquiera de ellos.

Se pueden dar por hechos los métodos .equals() que sean necesarios.

Ejercicio 4 *(25 pts.)*

Dadas las clases:

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

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

Escribir el método **public** ListaInt extraerListaCada(**int** p) que modifica la lista dejando p nodos y eliminando un nodo de manera alternada hasta el final de la lista. Es decir que si p=2 se dejan los primeros 2 nodos, se elimina el tercero, se dejan 2 nodos, se elimina el siguiente, y así siguiendo hasta el final de la lista. Cada número eliminado debe agregarse a una nueva lista, sin importar el orden, y el método devuelve la lista de números eliminados de la lista original. (Aclaración: p>=1)

Por ejemplo:

- Si la lista es [1,8,5,8,7,6] y p = 2, debe quedar [1,8,8,7] y devolver [5,6].
- Si la lista es [1,8,5,8,7,6] y p = 1, debe quedar [1,5,7] y devolver [8,8,6].
- Si la lista es [1,8,5,8,5,6] y p = 10, debe quedar [1,8,5,8,5,6] y devolver [].
- Si la lista es [] y p = 10, debe quedar [] y devolver [].

Además, se pide 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.