Algoritmos y Programación II
Tercer Examen Práctico

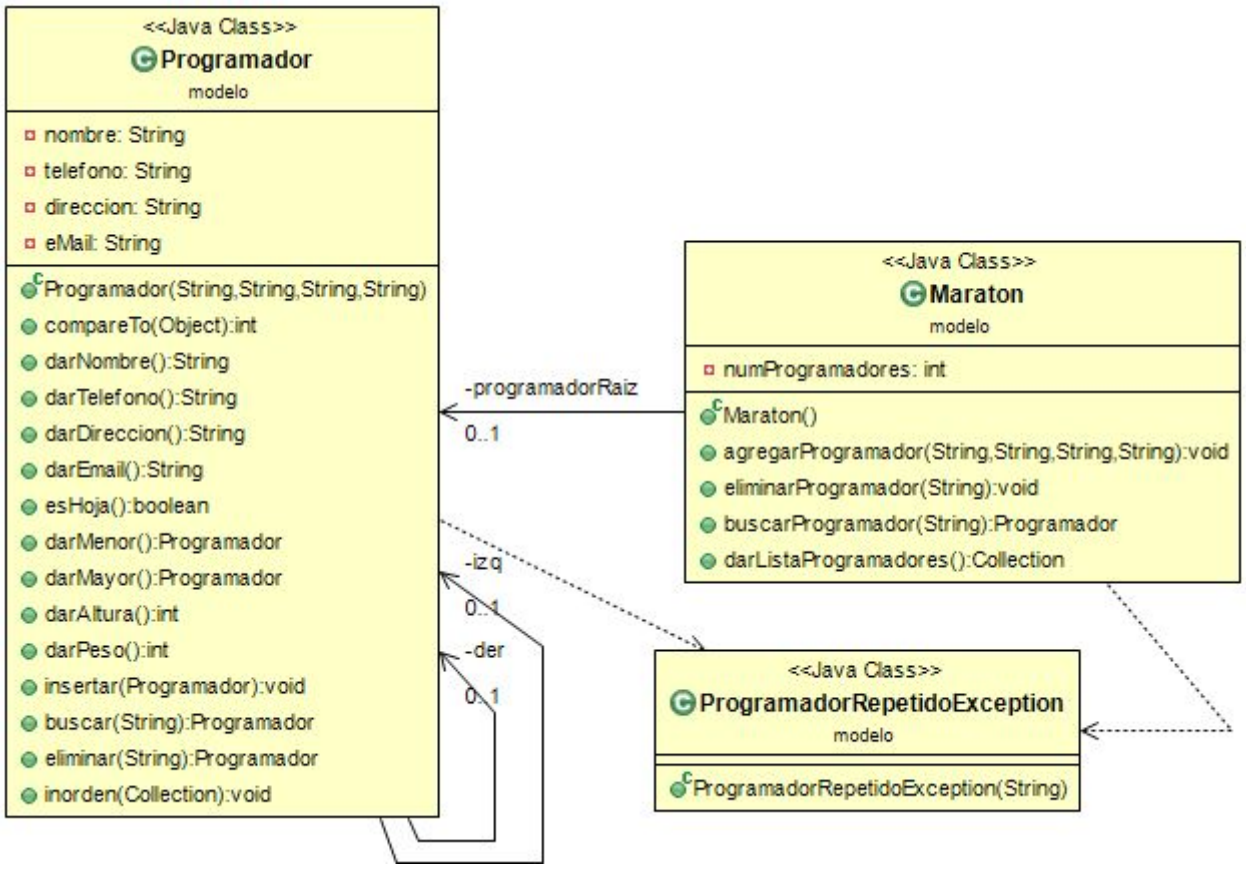
Nombre:	
Código:	Fecha: 22 de noviembre de 2018

Unidad 6: Estructuras y Algoritmos Recursivos
Al finalizar esta unidad, el estudiante estará en capacidad de:
OE6.1 Emplear el concepto de recursividad como una alternativa a la estructura de control iterativa.
OE6.2 Aplicar la computación recursiva en la solución de problemas de naturaleza inherentemente autocontenida.
OE6.3 Utilizar árboles binarios de búsqueda para representar grupos de objetos que mantienen entre ellos una relación de orden.
OE6.4 Escribir algoritmos recursivos para manipular estructuras de información recursivas y explicar las ventajas que, en este caso, estos algoritmos tienen sobre los algoritmos iterativos.

Enunciado

La Competición Internacional Universitaria ACM de Programación (en inglés ACM International Collegiate Programming Contest, abreviado ACM-ICPC o simplemente ICPC) es una competición anual de programación y algorítmica entre universidades de todo el mundo patrocinada por IBM. La ACM-ICPC es una competición que se inició en la Universidad A&M de Texas en 1970. Pasó a ser una competición con varias rondas clasificatorias en 1977 y la final mundial se organizó en colaboración con la ACM Computer Science Conference. Durante la competición, los equipos tienen alrededor de 5 horas para resolver entre 9 y 12 problemas (lo normal es 10 para las competiciones regionales y 12 para la final). Se deben programar las soluciones con C, C++, Java o Python 2 o 3. En la competición prima el trabajo en equipo, el análisis de problemas y el desarrollo rápido de software. ICPC es un evento auspiciado por la Association for Computing Machinery (ACM).

La Universidad ICESI quiere construir un programa que le permita registrar todos los programadores de la maratón para el control de la asistencia del evento, pero la universidad tiene la necesidad que todos los programadores sean registrados en un árbol binario de búsqueda. Un estudiante inició el programa hace ya un tiempo pero por problemas personales no pudo terminarlo y por está razón se le ha solicitado a usted que complete unas funcionalidades de este programa, la especificación de estas funcionalidades las encontrará a continuación.



Al construir cada uno de los métodos siguientes de forma recursiva, verifique su funcionalidad con las pruebas ya construidas. Dichas pruebas se encuentran en el paquete test del proyecto suministrado.

1. [25%]**[**OE6.1, OE6.2, OE6.3, OE6.4**] Agregar programador.**
Mediante el siguiente contrato y teniendo en cuenta el diagrama de clases suministrado, implemente el siguiente método.

```
/**
 * Inserta un nuevo programador al árbol que comienza en este nodo de forma recursiva.
 * @param nuevo - el nuevo programador que se va a insertar - nuevo != null
 * @throws ProgramadorRepetidoException se lanza esta excepción si el programador que se
 * quiere agregar ya está en la maratón
 */
public void insertar( Programador nuevo ) throws ProgramadorRepetidoException
{
    // TODO
}
```

2. [25%]**[**OE6.1, OE6.2, OE6.3, OE6.4**] Buscar programador.**
Mediante el siguiente contrato y teniendo en cuenta el diagrama de clases suministrado, implemente el siguiente método.

```
/**
 * Implementación recursiva para localizar un programador en el árbol que comienza en este nodo
 * @param unNombre nombre que se va a buscar - unNombre != null
 * @return programador asociado al nombre. Si no lo encuentra retorna null;
 */
public Programador buscar( String unNombre )
{
    // TODO
   return null;
}
```

3. [25%]**[**OE6.1, OE6.2, OE6.3, OE6.4**] Eliminar programador.**
Mediante el siguiente contrato y teniendo en cuenta el diagrama de clases suministrado, implemente el siguiente método.

```
/**
 * Elimina un programador del árbol que comienza en este nodo de forma recursiva.
 * @param unNombre nombre del programador que se va a eliminar.
 * @return el árbol de programadores después de eliminar el programador indicado.
 */
public Programador eliminar( String unNombre )
{
    // TODO
    return null;
}
```

4. [25%]**[**OE6.1, OE6.2, OE6.3, OE6.4**] Retorna una colección de programadores.**
Mediante el siguiente contrato y teniendo en cuenta el diagrama de clases suministrado, implemente el siguiente método.

```
/**
 * Retorna colección con los nombres de los programadores, ordenados alfabéticamente y ascendentemente
 * de forma recursiva.
 * @param acumulado colección donde se van agregando los nombres de los programadores ordenadamente
 */
public void inorden( Collection acumulado )
{
    // TODO
}
```