Universidad de Costa Rica
Escuela de Ciencias de la Computación e Informática
CI-0112 Programación 1

Profesores:

- Francisco Arroyo
- Braulio Solano
- Javier Vásquez
- Maureen Murillo (coordinadora)

# Examen #2 de Cátedra
# II-2018

**Observaciones**

1. Cuenta con **3.5** horas para entregar su solución del examen.
2. El examen es individual. Es prohibido utilizar herramientas digitales para intercambio de documentos. Guarde su celular. Si lo utiliza para lo que sea se considerará fraude.
3. Puede utilizar cualquier material propio (prácticas, tareas, libros, apuntes) que haya traído a la prueba.
4. No puede consultar internet, excepto la documentación de Java del sitio de Oracle: https://docs.oracle.com/javase/8/docs/
5. Debe realizar el examen utilizando una computadora de los laboratorios de la ECCI, por lo que no puede utilizar una portátil.
6. Como nombre de su proyecto indique su carne, nombreApellido, junto con ex2. Por ejemplo: B82345_JuanPerez_ex2.
7. En el comentario principal de cada clase, como autor indique su número de carné y su nombre.
8. En cada clase, incluya comentarios con el número de pregunta que está respondiendo (marque el inicio y el fin de cada respuesta). Ej:
   //inicio respuesta a pregunta #1
9. Cada 20 minutos suba su solución en un archivo comprimido que contenga el código fuente al sitio de entrega del examen de su grupo.
10. En caso de fallo de energía eléctrica deben continuar su examen en papel. Se calificará la versión subida hasta ese momento a la página del grupo, así como la documentación que aporte en su cuaderno de examen
11. La solución del examen debe subirse dentro del tiempo asignado a la página usada en su grupo. Al finalizar su examen y por motivos de seguridad guarde en memoria USB una copia de la solución entregada.
12. Firme la hoja de asistencia al finalizar el examen.

**Preguntas**

Construya un solo proyecto que contenga todas las clases necesarias para implementar las siguientes preguntas:

1. 10%. Como parte de una clase llamada "***Lista***" que representa una lista enlazada simple de números enteros, programe un método **iterativo** llamado "***int sume()***" que sume todos los valores que sean mayores que 10 y menores que 20 contenidos en la lista, y que retorne el resultado. La clase *Lista* deberá tener los métodos necesarios para poder probar el método *sume*, específicamente el *constructor* y un método *agregar*. Sólo se evaluará el método *sume*, el cual deberá colocarlo como primer método dentro de la clase.

2. 35%. Como parte de una clase llamada "***Arbol***" que representa un árbol binario ordenado que contiene números enteros positivos, programe el método "***int getElemento(int i)***" que devuelva el elemento en la i-ésima posición, según un ordenamiento ascendente. Si el árbol está vacío devuelve el valor -1.

   Por ejemplo, si se insertan en el árbol los números: 5, 2, 10, 4, 8, 16, 7, y luego se llama al método *getElemento(4)*, éste deberá devolver el 8, ya que según un orden ascendente el 8 es el cuarto elemento.

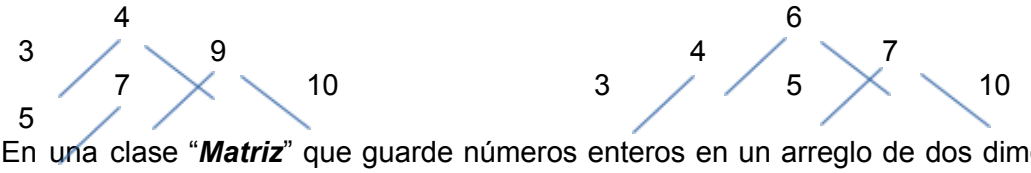
   La clase *Arbol* deberá contener los métodos *constructor* y *agregar* para poder probar el método *getElemento*. Solo se evaluará *getElemento* y deberá colocarlo como primer método de la clase. El método *getElemento* debe programarlo según el siguiente algoritmo:

   **Algoritmo:**

   Parámetros necesarios:
   la posición *i*-ésima que se desea
   la cantidad de elementos que ya se contaron en el momento en que el método inicia
   Retorna: el entero positivo en la posición *i*-ésima o -1 si no existe un elemento en esa posición

   1. Contar cuántos elementos tiene el subárbol izquierdo.
   2. Si la cantidad de elementos del subárbol izquierdo más la cantidad de elementos que ya se contaron (segundo parámetro) es mayor o igual que la posición *i*-ésima que se desea, entonces el elemento buscado está en el subárbol izquierdo, por lo que deberá solicitarse al subárbol izquierdo que ejecute *getElemento*, mandándole como parámetros los mismos valores recibidos.
   3. Si no se cumple lo anterior, determinar si el elemento deseado es el de la raíz. Si lo es, se retorna. Note que para saber si la raíz contiene el elemento que se desea, hay que determinar qué número de elemento es. Tome en cuenta la información que se recibió en los parámetros.
   4. Si no se cumple lo anterior, el elemento deseado probablemente esté en el subárbol derecho, por lo que deberá solicitarse al subárbol derecho que ejecute *getElemento* mandándole como parámetros los valores adecuados.

3. 25%. En la misma clase "***Arbol***" que programó en la pregunta anterior, programe el método "***boolean tienenFondoIgual(Arbol otroArbol)***" que devuelva verdadero si el fondo del primer árbol (el que ejecuta el método) es igual al fondo del segundo árbol (recibido por parámetro) y false en caso contrario. Dos árboles binarios tienen fondos iguales si tienen exactamente las mismas hojas leídas de izquierda a derecha, independientemente de los nodos interiores. Por ejemplo, los siguientes árboles tienen fondo igual:




4. 25%. En una clase "***Matriz***" que guarde números enteros en un arreglo de dos dimensiones, programe el método constructor "***Matriz(int numero)***" que construya una matriz cuadrada con el número de filas y de columnas indicado por el parámetro. La matriz puede ser de cualquier tamaño y deberá tener un triángulo inferior de números, en donde la primera fila tiene un 1 y la segunda fila del triángulo dos 1. Luego, cada elemento del triángulo, de la tercera fila en adelante, es la suma del elemento que está encima de él y del elemento que está a la izquierda del elemento encima de él. Si el elemento no tiene nada a la izquierda o encima se asume 0. Por ejemplo: ante la instrucción ***new Matriz(5)***, deberá generarse la siguiente matriz:

```
1 0 0 0 0
1 1 0 0 0
1 2 1 0 0
1 3 3 1 0
1 4 6 4 1
```

5. 5%. En la clase "***Matriz***" anterior, programe el método "***String toString***" que retorne únicamente el triángulo inferior de la matriz.

Principales aspectos a evaluar dentro del puntaje de cada pregunta:

1. Cumplimiento de buenas prácticas de programación.
2. Separación de responsabilidades de los objetos.
3. Correcta especificación de clases.
4. Uso correcto de constructores.
5. Uso correcto de la programación orientada a objetos.
6. Manipulación correcta de matrices.
7. Programación correcta de estructuras dinámicas.
8. Uso correcto de estructuras de control.
9. Entrada/salida.