Segundo Parcial de Estructura de Datos y Algoritmos

1	2	3	Nota
/4	/4	/2	/10

Duración: 1 hora y 45 minutos

Condición Mínima de Aprobación. Deben cumplir estas condiciones:

- Sumar **no menos de cuatro** puntos
- Sumar al menos 3 puntos entre el ejercicio 1 y 2

Muy Importante

Al terminar el examen deberían subir los siguientes 2 grupos de archivos, según lo explicado en los ejercicios:

1) Para ejer 1: CorporateHierarchy.java y Employee.java según lo pedido.

2) Para ejer 2: Amigos.java según lo pedido.

3) Para todos los ejercicios que no consistan en implementar código Java y pidan calcular complejidades, dibujar matrices, completar cuadros, hacer seguimientos, etc. pueden optar por alguna de estas estrategias:
   a. O completarlo en un documento, imprimirlo en pdf y subirlo también
   b. O directamente resolverlo en hojas de papel y sacarle fotos (formato jpg, png o pdf) y subir todas las imágenes.

Se considera una entrega aquella que corresponda a "un upload a Campus" para esta actividad, donde pondrán todos los archivos que deseen entregar.

Si bien pueden subir múltiples veces, sólo se corrige la última entrega realizada dentro del horario que dura la actividad.

Ejercicio 1

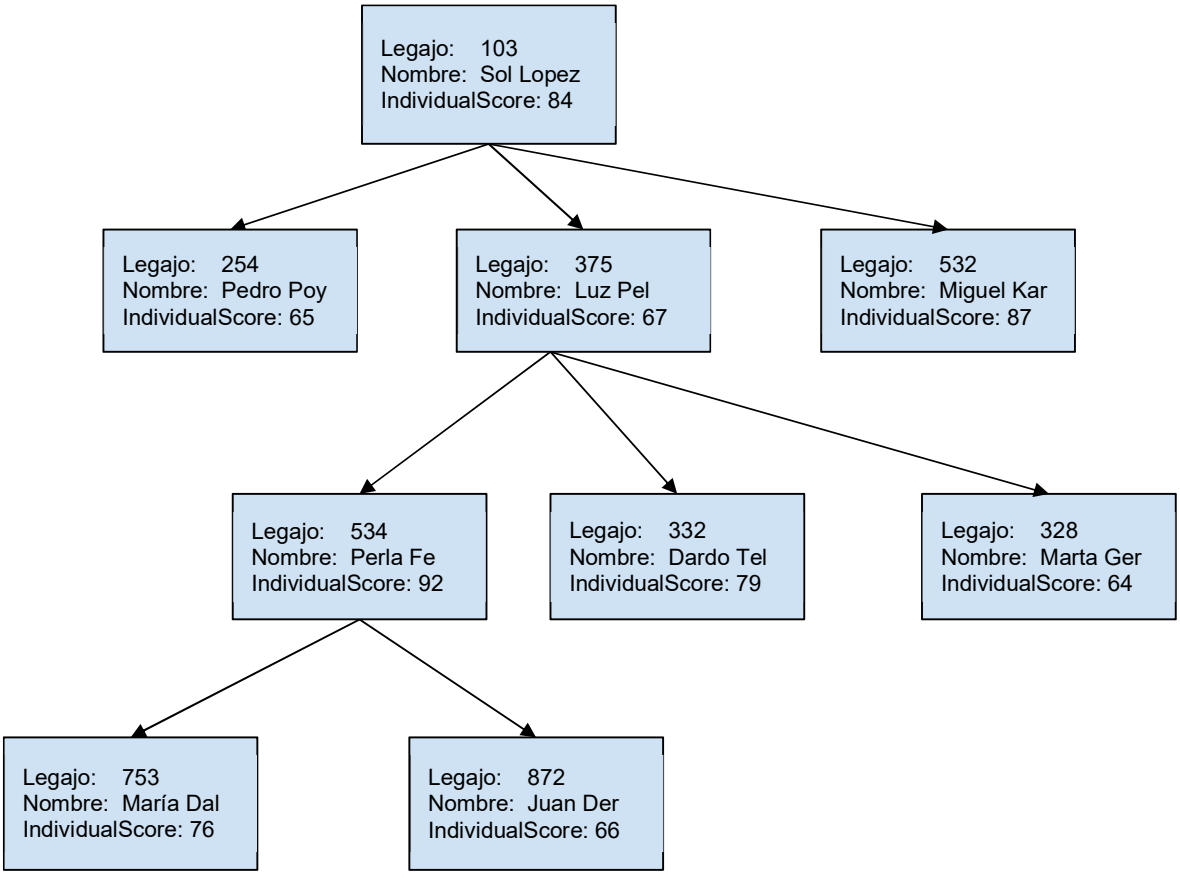
Se tiene el árbol con la **estructura jerárquica** de una empresa donde cada persona tiene como padre a su jefe y como hijos a las personas que reportan directamente hacia él. De esta manera **la raíz del árbol es el CEO de la empresa**.

La empresa realiza evaluaciones de personal anualmente y califica a cada empleado con una calificación individual en base a su rendimiento personal entre 0 y 100. La evaluación del CEO la realiza la Junta Directiva.

Para cada persona se almacena en la clase Empleado su número de legajo, su nombre y su **Calificación individual** (Individual Score)

Además, cada empleado tiene una **Calificación Colectiva** (Collective Score) que se calcula en un **50% como la calificación individual propia** y en el otro **50% como el promedio de la Calificación Colectiva de sus reportes directos (hijos directos de él en el árbol**). Si no tuviera reportes directos (en el caso de ser hoja del árbol) su calificación colectiva sería su calificación individual. Esa **Calificación Colectiva NO SE ALMACENA**, sino que se calcula cada vez con la fórmula explicada.

Ejemplo:



En el ejemplo:

getCollectiveScore(332) = 79

Debe retornar 79 por ser un nodo hoja.

getCollectiveScore(753) = 76

Debe retornar 76 por ser un nodo hoja.

getCollectiveScore(872) = 66

Debe retornar 86 por ser un nodo hoja.

getCollectiveScore(534) = 81.5

Calcula el promedio entre los **Scores Colectivos** de los empleados con legajo 753 y 872 (que coinciden con sus individuales por ser hoja)

(76+66)/2 = 71

y devuelve el promedio entre dicho valor y **su Score Individual**, o sea, retorna
92*0.5 + 71*0.5 = 81.5

getCollectiveScore(375) = 70.91666

Calcula el promedio entre los Scores Colectivos de 534, 332 y 328. Es decir,

(81.5 + 79 + 64)/3 = 74.8333

y devuelve el promedio entre dicho valor y su **Score Individual**, o sea, retorna
67 * 0.5+ 74.8333 * 0.5 = 70.91666

Y así siguiendo.

Bajar las clases **CorporateHierarchy** y **Employee** de campus.

Se pide implementar:

- el método **double getCollectiveScore(int legajo)** que devuelve la calificación colectiva dado un número de legajo.

- el método find **Employee findEmployee(int recordNum)** que devuelve un empleado dado su número de legajo. Este método es usado en la inserción de empleados y puede usarse también en getCollectiveScore.

Ejercicio 2

Preguntamos a los alumnos del ITBA sobre sus amigos y obtuvimos la lista de amigos de cada uno.

Almacenamos en un HashMap las listas usando como clave el nombre del alumno y como valor la lista de sus amigos. **No hay alumnos que tengan el mismo nombre**.

Por ejemplo:

```
HashMap<String, List<String>> amigos = new HashMap<>();
amigos.put("A", Arrays.asList("B", "C"));
amigos.put("B", Arrays.asList("D", "F", "G"));
amigos.put("C", Collections.emptyList());
amigos.put("D", Arrays.asList("A", "B"));
amigos.put("F", Arrays.asList("A", "D"));
amigos.put("G", Arrays.asList("B", "C", "H"));
amigos.put("H", Collections.emptyList());
```

Este ejemplo representaría que el alumno "A" considera amigos a "B" y "C", el alumno "B" considera amigos a "D", "F", y "G", etc. **Notar que la relación de amistad no es simétrica**, es decir, "A" puede considerar como amigo a "B" y "B" no necesariamente considera como amigo a "A".

Queremos conocer a los amigos indirectos de algunos alumnos hasta cierto nivel. Para esto hablaremos de amigos de grado N.

Los amigos de grado 1 son tus amigos directos.
Los amigos de grado 2 son los amigos de tus amigos grado 1 que no sean amigos de grado 1 tuyos.
Los amigos de grado 3 son los amigos de tus amigos de grado 2 que no sean amigos de grado 1 o 2 tuyos.
Etc…

En general, los amigos de grado N son los amigos de tus amigos de grado N-1 que no sean amigos tuyos de grado K, para todo K < N.

Por ejemplo, para el caso de "A" en el ejemplo:

- Amigos de grado 1: "B" y "C"
- Amigos de grado 2: "D", "F", "G"

- Amigos de grado 3: "H"
- Amigos de grado 4: no tiene

Si alguien no tiene amigos de grado N entonces tampoco tendrá amigos de grado mayor a N.

Queremos implementar el método
List<String> amigos(String alumno, HashMap<String, List<String>> amigos, Integer grado)

que dado un alumno (primer parámetro), el mapa de amigos y un grado, devuelva todos los amigos de ese grado del alumno en cuestión (primer parámetro). Si el grado es menor o igual a cero debe devolver la lista vacía. Si no tiene amigos de ese grado debe devolver la lista vacía también.

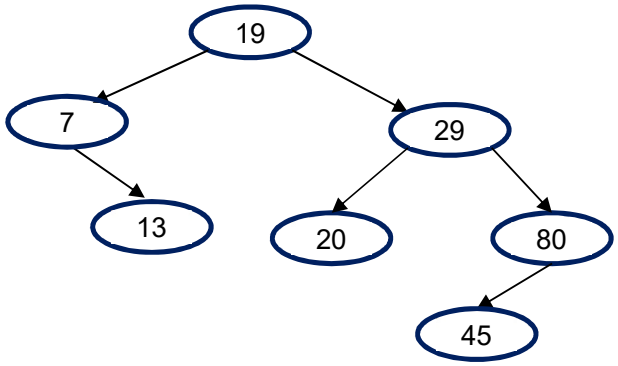
Descargar de campus la clase Amigos.

Se pide:

- Implementar en la clase Amigos el método
  List<String> amigos(String alumno, HashMap<String, List<String>> amigos, Integer grado) que cumpla con lo pedido.

Ejercicio 3

Tomando como comienzo el siguiente árbol AVL que **no acepta repetidos** (los ignora).



Mostrar **gráficamente** cómo va quedando si se le aplican las operaciones solicitadas en secuencia.
Para cada operación se pide:

- Mostrar primero gráficamente dónde se inserta el valor.
- Analizar si algún nodo queda desbalanceado. En caso de que haya que solucionar, indicar claramente con qué tipo de rotación se soluciona.
    - Si fuera simple, no omitir indicar cuál es el pivote y hacia donde se rota. Mostrar cómo queda el gráfico luego de la rotación
    - Si fuera doble, desdoblar el análisis en 2 casos. Primero indicar cuál es el primer pivote y hacia dónde se rota, mostrando cómo queda el gráfico luego de dicha rotación intermedia. Luego, indicar cuál es el segundo pivote y hacia dónde se rota, mostrando cómo queda el gráfico luego de dicha última rotación. O sea, mostrar 2 rotaciones sucesivas e indicar pivote usado en cada caso.

3.1) Inserción del valor 77

Rta 3.1

3.2) Inserción del valor 10 respecto al AVL Tree obtenido en 3.1

Rta 3.2

3.3) Inserción del valor 82 respecto al AVL Tree obtenido en 3.2

Rta 3.3

3.4) Inserción del valor 99 respecto al AVL Tree obtenido en 3.3

Rta 3.4