Informática II Nivel Superior - Prepa Tec Campus Eugenio Garza Lagüera
Laboratorio Primer Parcial

Nombre: _____ Matrícula: _____

Sección 1: Diseña una clase llamada CreditCard que sirva para modelar una tarjeta de crédito. Piensa en los atributos de una tarjeta de crédito, ¿qué información está en ella? ¿Qué acciones puedes realizar? Utiliza las respuestas a estas preguntas para crear un diagrama UML de dicha clase. Debe contener como mínimo 5 atributos y 4 métodos. Posteriormente, dibuja tres ejemplos de objetos (con sus respectivos atributos).

Sección 2: Ordena los siguientes arreglos a mano ascendentemente, mostrando cada uno de los pasos, utilizando los algoritmos de Selection Sort y Bubble Sort. Indica la cantidad de comparaciones e intercambios que tuvo que realizarse en cada algoritmo.

45	1	88	0	0	35	99

5	4	3	2

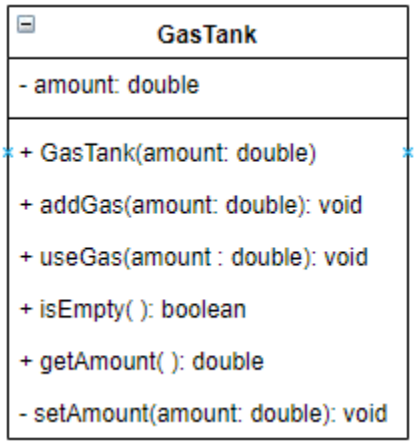
3	9	7	5	2	-1	46

1	3	3	4	5

Sección 3: Resuelve los siguientes programas en tu computadora. Al finalizar, sube a Blackboard los archivos de código fuente (.java).

Problema 1: Crea una clase llamada **`SortingAlgorithms`** que contenga los métodos de ordenamiento Bubble Sort y Selection Sort, modificados para que calcule y almacene la cantidad de intercambios y comparaciones realizadas. Utiliza esta clase para comprobar tus respuestas de la Sección 2.

Problema 2: Diseña una clase **`GasTank`** que sirva para representar un tanque de gasolina. Diseña la clase de acuerdo con el siguiente diagrama UML:



- **Constructor:** Este método debe recibir como parámetro de entrada una cantidad **amount**, y llame al método **setAmount** para actualizar la variable de instancia **amount.**
- **addGas**: deberá incrementar la cantidad de gasolina en el tanque en la cantidad recibida como parámetro**.** Asegúrate de validar que sólo se procesen valores positivos.

- **useGas:** deberá reducir la cantidad de gasolina en el tanque en la cantidad recibida como parámetro. Asegúrate de que sólo se procesen valores positivos.
- **isEmpty:** Deberá devolver **true** cuando la cantidad de gasolina en el tanque sea menor a 0.1. De lo contrario, deberá retornar **false.**
- **getAmount:** Getter para la variable amount.
- **setAmount:** Método privado (sólo será usado por el constructor) que actualice la variable **amount** siempre y cuando el parámetro recibido sea mayor o igual a 0.

Problema 3: Diseña una clase **`Movie`** que sirva representar la información de una película. La clase debe tener los siguientes atributos:

- Nombre de la película
- Clasificación (AA, A, B, B15, C, D)
- La cantidad de personas que le dieron una reseña de 1 estrella (Muy mala).
- La cantidad de personas que le dieron una reseña de 2 estrellas (Mala).
- La cantidad de personas que le dieron una reseña de 3 estrellas (OK).
- La cantidad de personas que le dieron una reseña de 4 estrellas (Buena).
- La cantidad de personas que le dieron una reseña de 5 estrellas (Muy buena).

¿Puedes evitar definir 1 variable diferente para cada calificación?

Adicionalmente, agrega los siguientes métodos:

- Método constructor que reciba dos parámetros de entrada: El nombre de la película y su clasificación inicial.
- Métodos **accesors** y **mutators** para las variables Nombre de la película y la clasificación.
- Método **void addReview(int rating)** que reciba como parámetro un entero **rating**. Valida que el entero recibido sea un número entre 1 y 5. De ser así, incrementará en 1 la cantidad de personas que le dieron a la película dicha calificación.
- Método **double getAverage( )** que retorne la calificación promedio otorgada a la película.
- Método **String toString()** que imprima el contenido del objeto.

Al terminar, prueba tu clase escribiendo un método main que instancie por lo menos 2 objetos de la clase **Movie**. Agrega 5 calificaciones a cada objeto, e imprime el nombre de la película, su clasificación y calificación promedio.

Problema 4: Diseña una clase llamada **`TicTacToe`** que sirva para modelar un juego de Tic-Tac-Toe (gato) para dos jugadores. El juego deberá validar las entradas y mostrar en consola el ganador del juego.

Problema Reto (Máximo 10 puntos extras!): Diseña una clase llamada **`Connect4`** que sirva para modelar y jugar el juego de mesa Connect 4. Si quieres revisar las reglas, utiliza la siguiente liga: https://www.mathsisfun.com/games/connect4.html

---

Problema 5: Escribe un método estático **char[ ] removeDuplicates(char[ ] in)** que retorne un nuevo arreglo de caracteres sin caracteres duplicados. Siempre deberás mantener el primer elemento encontrado y eliminar los elementos subsecuentes.
Ejemplo:
`removeDuplicates(new char[]{'b','d','a','b','f','a','g','a','a'} )` → {'b','d','a','f','g'}.

Problema 6: Escribe un método estático **int[ ][ ] removeSmaller(int v, int[ ][ ] in)** que retorne una nueva matriz de enteros a partir del arreglo recibido **in**, pero con los valores *menores a v* eliminados. El nuevo arreglo retornado deberá tener la misma cantidad de filas que el arreglo `in`, pero cada fila deberá tener sólo las columnas necesarias.

`removeSmaller(5, in)`

`in =`

5	3	6	5
1	2	3	5
8	8	6	5
−1	5	2	15
0	0	0	0

`out =`

5	6	5	
5			
8	8	6	5
5	15		

Prueba el método anterior diseñando 3 casos de prueba adicionales a los siguientes:

1. in es un arreglo no inicializado (null)
2. todos los valores de in son menores a v
3. <caso de prueba 3>
4. <caso de prueba 4>
5. <caso de prueba 5>