Universidad Nacional de San Agustín
Escuela Profesional de Ingeniería de Sistemas
Fundamentos de Programación II
Practica de Laboratorio 13:
Definición de Clases de Usuario
Clase Soldado – Miembros de Clase

I

OBJETIVOS

- Que el alumno demuestre poder crear "clases definidas por el programador"
- Implementar métodos para las clases definidas por el programador
- Utilizar miembros de Clase

II

Marco Teórico

Clase: Una Fábrica de objetos. Una clase está conformada por dos partes: datos miembro (variables de instancia / atributos) y los métodos.

Los métodos nos permiten acceder y/o modificar los datos miembros (atributos) de los objetos.

Toda clase necesita al menos un Constructor. El constructor lleva el mismo nombre de la clase. El constructor es un método especial que siempre se llama con la palabra reservada **new()**

III

EJERCICIOS PROPUESTO

1. Introducción

Este laboratorio requiere que usted escriba un programa utilizando clases definidas por el programador. No deberá utilizar sintaxis o constructores que no han sido cubiertos durante las clases teóricas. Será penalizado por esta falta. A menos que una plantilla sea dada, deberá utilizar cada programa desde cero de manera que obtenga suficiente práctica en la escritura de programas en Java.

Un consejo: **Programe incrementalmente.** No trate de terminar todas las partes del programa y luego compilarlo. Escriba sus programas en partes y compílelo de forma frecuente. Trate de mantener un programa compilable aun cuando esté trabajando en él. Presentar un programa compilable que funcione parcialmente es mejor que presentar un programa no-compilable. EN SERIO, programe incrementalmente.

Los objetivos de este laboratorio son:

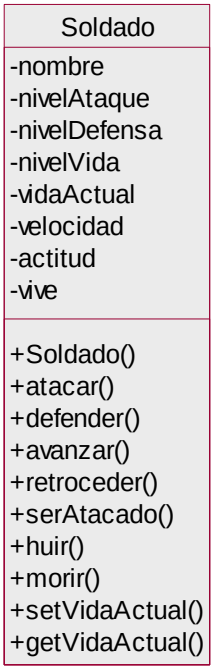
Deberá asumir que todos los **datos de ingreso son correctos.**

Deberá utilizar la clase Scanner en System.in para ingresos de datos y System.out para salida de datos en sus programas, a menos que se indique lo contrario.

Pruebe sus programas con sus propios datos de prueba antes de presentarlos.

Evitar **duplicación de código**.

Usar como base el diagrama de clases UML siguiente:



- Crear 3 constructores sobrecargados.
- La actitud puede ser defensiva, ofensiva, fuga. Dicha actitud varía cuando el soldado defiende, ataca o huye respectivamente.
- Al atacar el soldado avanza, al avanzar aumenta su velocidad en 1. Al defender el soldado se para. Al huir aumenta su velocidad en 2. Al retroceder, si su velocidad es mayor que 0, entonces primero para y su actitud es defensiva, y si su velocidad es 0 entonces disminuirá a valores negativos. Al ser atacado su nivel de vida disminuye y puede llegar incluso a morir.
- Crear los atributos y métodos extra que considere necesarios.
- Tendrá 2 Ejércitos. Usar la estructura de datos que considere más adecuada. Inicializar el tablero con n soldados aleatorios entre 1 y 10 para cada Ejército.

Cada soldado tendrá un nombre autogenerado: Soldado0X1, Soldado1X1, etc., un valor de puntos de vida autogenerado aleatoriamente [1..5], la fila y columna también autogenerados aleatoriamente (no puede haber 2 soldados en el mismo cuadrado). Nivel de ataque y de defensa son aleatorios [1..5]. Se debe mostrar el tablero con todos los soldados creados (usar caracteres como | _ y otros) y distinguir los de un ejército de los del otro ejército. Además de los datos del Soldado con mayor vida de cada ejército, el promedio de puntos de vida de todos los soldados creados por ejército, los datos de todos los soldados por ejército en el orden que fueron creados y un ranking de poder de todos los soldados creados por ejército (del que tiene más nivel de vida al que tiene menos) usando 2 diferentes algoritmos de ordenamiento.

- El juego es humano contra humano y consistirá en mover un soldado por cada turno de cada jugador. Se puede mover en cualquier dirección, Ud. deberá darle la coordenada del soldado a mover y la dirección de movimiento, el programa deberá verificar que hay un soldado del ejército que corresponda en dicha posición y que el movimiento es válido (no puede haber 2 soldados del mismo ejército en el cuadrado y no se puede ordenar moverse a una posición fuera del tablero), pidiendo ingresar nuevos datos si no es así. Cuando un soldado se mueve a una posición donde hay un soldado rival, se produce una batalla y gana el soldado basado en la siguiente métrica: la suma de los 2 niveles de vida actual de los soldados que luchan son el 100% y se le debe dar la probabilidad correspondiente de vencer para cada soldado (ejemplo S1:5 S2:3, las probabilidades de vencer serían S1:62.5% S2:37.5%) y de acuerdo a dichas probabilidades se decidirá el ganador aleatoriamente. El ganador ocupará dicho cuadrado (se le aumentará su nivel de vida actual en 1) y el perdedor desaparecerá. Para cada batalla se deberá explicar por qué ganó uno de los soldados. Gana el juego quien deje al otro ejército vacío. Después de cada movimiento se deberá mostrar el tablero con su estado actual. Hacer un programa iterativo.
- Realizar el diagrama de clases de UML completo.
- Al empezar el juego mostrar la cantidad total de objetos Soldado creados y la cantidad de objetos Soldado creados por cada ejército. (Usar miembros de clase).
- Considerar la cantidad máxima de soldados por ejército como una constante de clase y usarla en todo el programa, no usar "número mágico" para eso.

- Las opciones de menú de la práctica anterior deberán usar los miembros de clase.
- Al acabar cada turno mostrar la cantidad de objetos Soldado que quedan por cada ejército. (Usar miembros de clase).
- La finalización del juego será determinada por la verificación de dichos miembros de clase.