Modulo	Nivel de Dificultad
Fundamentos de Programación Java	Medio
Tema: Programación Orientada a Objetos	Challenge “Consolidando POO”

Intención del aprendizaje o aprendizaje esperado:

- Aplicar programación orienta a objetos.
- Se debe emplear código Java, basado en conceptos y buenas prácticas de la industria

Planteamiento del Problema:

1.- CASO: “CONSECUENCIAS”

Hace unos meses, la “Cámara Secreta” de *Oscurilandia* decidió implementar un nuevo sistema de votación para sus leyes, dado el aumento en la cantidad de representantes. Esto generó vicios en el proceso, ya que muchos representantes, a fin de evitar aprobar una ley que no fuera de su gusto, se restaban de la votación, aumentando la cantidad en el universo de votos y dificultando con ello obtener el margen de aprobación necesario.

Esto generó molestia en los ciudadanos de *Oscurilandia*, y están considerando seriamente invadir la “Cámara Secreta”, gracias a la ayuda de un grupo de superhéroes anónimos autodenominados “FirstLine”.
En razón a la contingencia actual, los representantes han llamado a su grupo de acción, la élite de seguridad PKS, para restaurar el orden. Este grupo cuenta con tres tipos de vehículos para realizar operaciones:

- **Kromis:** Tienen un largo de tres metros cada una, y son utilizadas para transportar efectivosde la PKS.
- **Caguanos:** Cada uno mide dos metros de largo, y su misión es lanzar confetis y dulces a quienes protestan.
- **Trupallas:** Corresponden a efectivos cibernéticos con tecnología de punta, cada uno ocupando en total un metro cuadrado de espacio, preparados para hacer entrar en razón a los manifestantes con la fuerza de sus ideas.

Como respuesta a las acciones de la PKS, el comando FirstLine solicita que se desarrolle un programa que los ayude a simular las posiciones de los efectivos alrededor de la Cámara Secreta, y de esta forma coordinar las acciones previas a la invasión.

2.- PROBLEMA
Desarrolle un programa que realice lo siguiente:

- Para efectos de simulación, el lugar de acción se simulará como una matriz compuesta de caracteres con 15 filas y 15 columnas, en los que se dispondrán de forma aleatoria:

- 3 Kromis, donde cada una utiliza tres celdas de la matriz y se ubican siempre de forma vertical. Cada celda ocupada por un vehículo de este tipo se define con una letra “K”.
- 5 Caguanos, ubicados siempre horizontalmente en la matriz, definidos en ésta con una letra “C” y haciendo uso de dos espacios.
- 10 Trupallas, los cuales se representan como una celda en la matriz con la letra “T”.

- El programa debe situar de forma aleatoria la cantidad de vehículos antes indicados. Se recomienda hacerlo en el orden antes mencionado, pero no es obligación. Considere que cada tipo de vehículo ocupará una determinada cantidad de celdas en la matriz, y que no pueden existir vehículos encima de otros.
- Por cada vehículo ingresado, entonces, se debe hacer lo siguiente:

- El programa calcula una fila y columna de forma aleatoria (el número debe estar en el rangodefinido).
- Si está agregando una Kromi, debe escribir tres celdas hacia abajo contando desde la celda obtenida, si es un Caguano debe rellenar dos celdas hacia el lado (de izquierda a derecha), mientras que, si el vehículo a registrar es un Trupalla, se debe agregar en la coordenada obtenida ya que solo usa una celda.
- Antes de agregar un vehículo, se debe validar que no exista ningún otro en algunas de las posiciones que se considerará. De no ser así, se debe obtener nuevamente otro par de coordenadas de forma aleatoria hasta que sea posible ubicar el vehículo en cuestión sin inconvenientes.
- Además, recuerde que todo vehículo debe quedar siempre situado dentro de la matriz, lo cual también se debe validar. Si las coordenadas obtenidas provocan que el vehículo no cumpla con esta condición, se debe solicitar otra fila y columna.
- Un ejemplo de matriz generada de forma posterior a este proceso es la que se muestra a continuación:

T														
		K				T							C	C
		K												
		K							C	C	T			
		T												
				C	C							T		
		T					T				K			
											K			
			K								K			
			K					T				T		
			K											
	C	C						C	C					
														T

- Los manifestantes, como método de defensa frente a la acción de la PKS, tienen un arsenal ilimitado de huevos, los que pueden lanzar libremente dentro del espacio considerado. Para representar esto, se debe solicitar al usuario que ingrese un número de fila y de columna, ambos válidos, y verificar si en la matriz está ubicado un vehículo o no:
- Si el proyectil da sobre una Kromi, se asignan tres puntos, si da sobre un Caguano se asignan dos puntos, y si da sobre un Trupalla se asigna un punto.

- Si una Kromi ha sido inutilizada completamente, esto es, que sus tres celdas han sido atacadas por huevos, se asignan 10 puntos adicionales. Si se inutiliza un Caguano se asignan 7 puntos adicionales. Un Trupalla queda inutilizado automáticamente al recibir un huevo, por tanto, no existe puntaje adicional por estos efectos.
- No hay una cantidad límite de huevos a lanzar, y un huevo puede caer más de una vez sobre una misma posición. Sin embargo, cada vez que una posición sea atacada por un huevo, el valor de la celda en la matriz original debe ser expresado como "H". Si la celda sobre la que se lanza un huevo ya había sido atacada previamente, no se debe asignar puntaje por este concepto.
- Finalmente, al terminar el programa se debe indicar el puntaje total obtenido por el usuario.

3.- Clases

Para resolver este problema se pide crear como mínimo las siguientes estructuras:

- Clase "Carro": es la clase padre que definirá los distintos vehículos que administra la PKS. De cada carro se interesa saber la cantidad de ocupantes, la fecha de ingreso a la institución y su ubicación en el tablero virtual de 15x15 (se recomienda registrar la fila y columna en atributos independientes). Debe tener un método que permita desplegar sus datos básicos,y sus coordenadas en el tablero.
- Clase "Kromi": clase hija que representa una Kromi; se requiere saber su año de fabricación y su marca.
- Clase "Caguano": clase hija que representa un Caguano; de esta clase se interesa saber su alcance de tiro, y color de confeti que arroja.
- Clase "Trupalla": clase hija que representa un Trupalla; de este tipo de elemento interesa conocer su nivel de armadura (entero entre 1 a 5) y el nombre de la persona que lo manipula.
- Clase "Huevo": clase que define cada lanzamiento realizado dentro del tablero. Por cada instancia de esta clase se desea conocer la fila donde cayó el proyectil, la columna y el puntaje obtenido en el lanzamiento.
- Clase "Tablero": clase que representa el terreno en el que se ubica cada carro y en el cual se sitúan proyectiles. Esta clase debe contener dos atributos: una lista o arreglo de instancias de la clase "Carro" y una lista de instancias de la clase "Huevo"; se sabe que la primera lista no tendrá más de 18 elementos, mientras que en el segundo no hay límite de instancias a crear. Esta clase, además, debe tener definidos los siguientes métodos:
  - Crear Carro: crea una subclase de la clase Carro y la asigna a la lista respectiva. Recuerde que las coordenadas del carro se calculan de forma aleatoria, y no se puede traslapar un carro con otro.
  - Lanzar Huevo: crea una instancia de la clase "Huevo", solicita la coordenada de lanzamiento, asigna el puntaje al movimiento y la almacena en el listado correspondiente.
  - Mostrar Matriz: este método debe mostrar en forma de matriz cada uno de los carros existentes, y los lanzamientos que se han registrado hasta el momento. Recuerde que donde hubo un lanzamiento debe haber una letra "H", independiente de si acierta a un carro o no. Una vez que realiza la acción, debe calcular el puntaje obtenido hasta el momento.
  - Calcular puntaje: suma los puntajes asignados a cada lanzamiento y los entrega como resultado. Este método debe ser visible solo dentro de la clase, y es utilizado en los métodos de la misma clase. Clase "Huevo": clase que define cada lanzamiento realizado dentro del tablero.
- Algunas acotaciones de índole general
  - Todas las clases deben estar acompañadas de sus métodos set, get y toString respectivos.
  - Toda clase debe tener declarado un constructor apropiado.
  - Para el desarrollo del problema puede usar matrices. Las puede declarar como atributos de alguna clase de las anteriores.

- Puede considerar otras clases además de las antes mencionadas.
- El programa debe contar con un menú que despliegue las acciones que contempla el programa, y debe mostrarse tantas veces sea necesario hasta el usuario indique lo contrario.

4.- Consideraciones

- Se debe programar bajo el concepto de programación orientada a objetos, por ende, debe tener estructura de clases y superclases, conceptos de herencia y polimorfismos.
- Se debe implementar la mayor cantidad de conceptos vistos en cada una de las clases, sin ser el total de ellos de carácter obligatorio.
- Se valorará la participación y el trabajo colaborativo. (Commits)
- La revisión se realizará durante clases.
- Tendrá una duración máxima de 3 clases. Durante el último plazo, se revisará en cada uno de los
- equipos de los participantes.
- Cada equipo del proyecto tendrá como máximo 2 personas.

Datos de apoyo al planteamiento

Ejecución en Pares (equipo de 2 personas)
Componentes para evaluar: Debe entregar su respuesta en un archivo de extensión *.rar o *.zip.

Recursos Bibliográficos:

Tutorial de Java
https://docs.oracle.com/javase/tutorial/

Atributos y métodos de clase
http://www.it.uc3m.es/java/gitt/resources/static/index_es.html

Diagramas de clases
https://www.lucidchart.com/pages/es/tutorial-de-diagrama-de-clases-uml

Java: Polimorfismo, herencia y simplicidad
https://www.arquitecturajava.com/java-polimorfismo-herencia-y-simplicidad/

SOLID: los 5 principios que te ayudarán a desarrollar software de calidad
https://profile.es/blog/principios-solid-desarrollo-software-calidad/