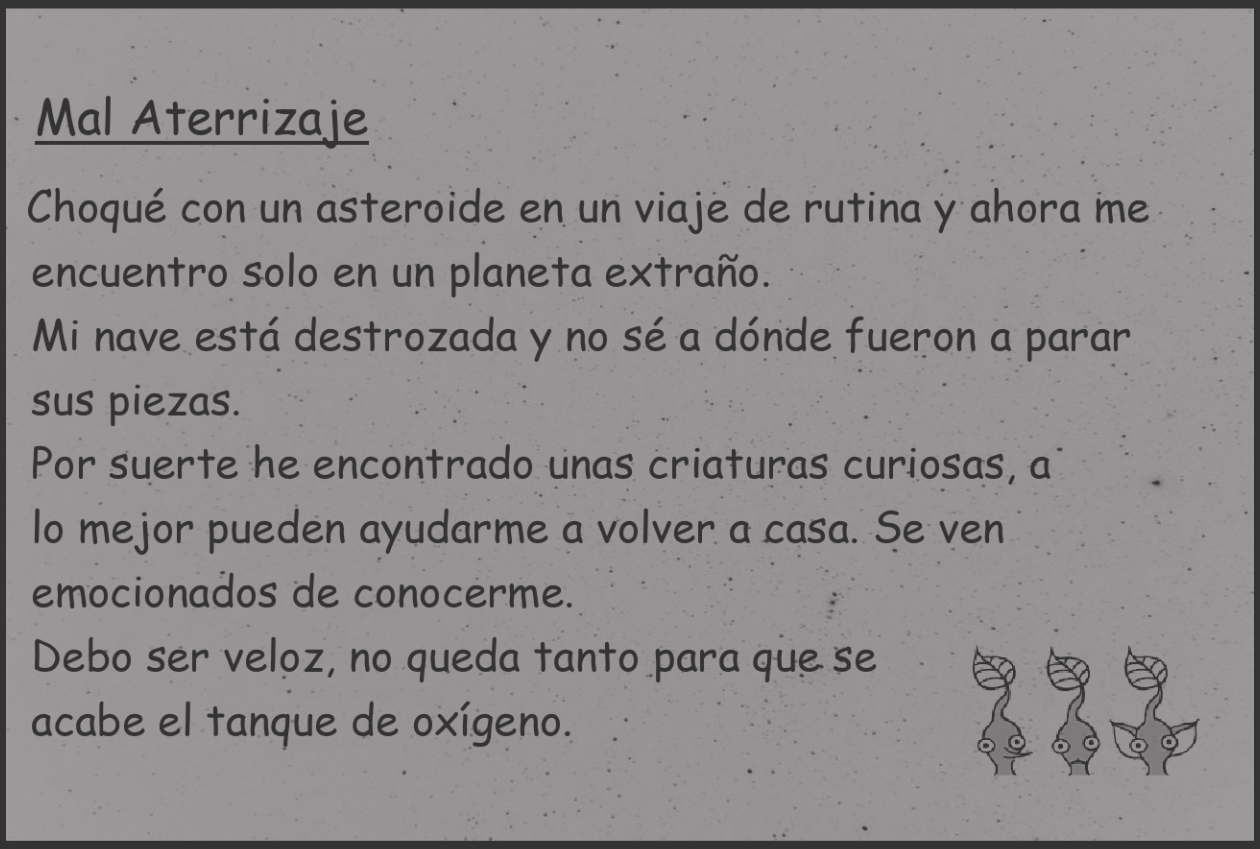
INF-253 Lenguajes de Programación

Tarea 3: Java

12 de septiembre de 2023

1. Pikinim



En menos de 30 horas, el capitán Lomiar debe recuperar las 3 piezas de su nave para escapar del planeta antes de que se le acaben sus soportes vitales. Para esto, investigará cada zona del planeta con la ayuda de las extrañas criaturas, los *Pikinim*.

2. Vista General del Juego

El juego de estrategia consiste en investigar un mapa, donde en cada turno el jugador podrá ver el estado de sus *Pikinim*, las piezas que lleva, y elegir si ir a la zona de la izquierda y la derecha. El jugador tiene 30 turnos para encontrar todas las piezas, y cada vez que se mueve a una zona del mapa (celda del arreglo) pasa 1 turno.

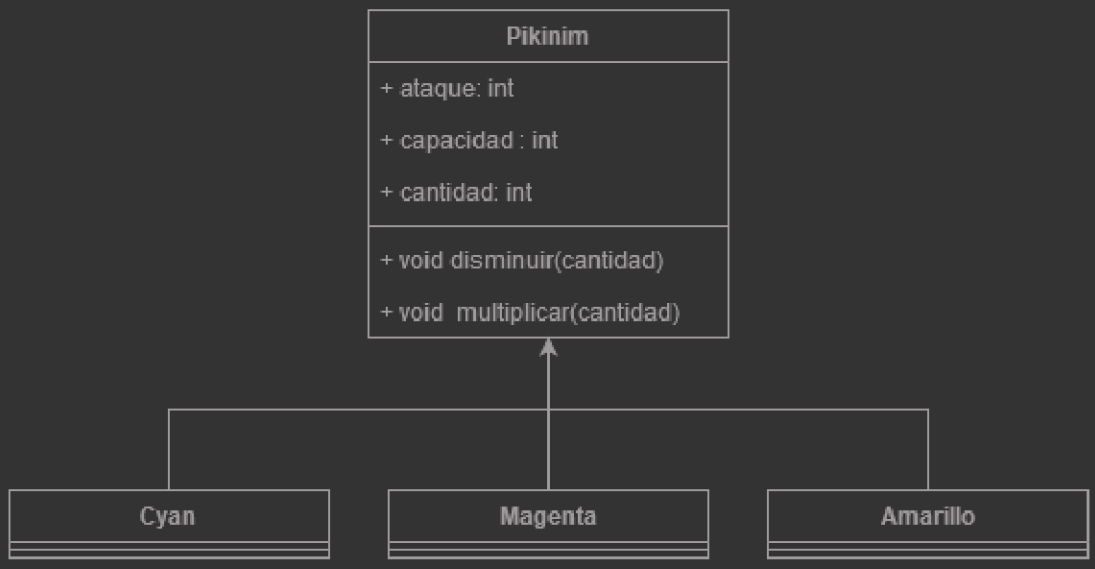
Dentro de cada zona, el jugador puede encontrar enemigos, murallas destructibles, una pieza de la nave, o una píldora. Cada zona interactúa de forma distinta con los *Pikinim* del jugador: pueden interrumpir su camino, pelear contra ellos o aumentar la cantidad de *Pikinim* que acompañan al jugador.

Si el jugador recupera todas las piezas de la nave dentro de los 30 turnos, gana el juego, de lo contrario, pierde y el capitán Lomiar no podrá volver a su hogar. También puede perder si en alguna pelea se queda sin ningún *Pikinim*.

3. Pikinim

Los *Pikinim* son una de las criaturas nativas del planeta, son muy pequeñas y no muy fuertes individualmente, pero en conjunto pueden hacer muchas cosas (como una hormiga). Por esto, los *Pikinim* llevan píldoras y enemigos a su casa para multiplicarse, así, de esta forma el tamaño de la tropa aumenta y así pueden recuperar las piezas de la nave de Lomiar.

En el programa se debe seguir la estructura del siguiente UML:



Donde *Pikinim* es una clase **abstracta** que incluye las variables *ataque*, *capacidad, cantidad*, y las funciónes *disminuir* y *multiplicar*. Las subclases *Cyan*, *Magenta* y *Amarillo* heredan de *Pikinim* y simbolizan un **grupo de *Pikinim* de ese color** (por eso la variable *cantidad*), cada color tiene características propias e implementan su propia versión de *multiplicar*. A continuación, se explica qué hace cada elemento de la clase:

- *ataque – int*: Determina cuánto daño hace cada *Pikinim* cuando pelea contra un enemigo.
- *capacidad – int*: Determina cuánto peso levanta cada *Pikinim* cuando intenta llevarse una pieza de la nave.
- *cantidad – int*: Cantidad de *Pikinim* del color de la clase. Se multiplica por *ataque* y *capacidad* en los cálculos que los requieran.
- *disminuir(cantidad) – void:* Recibe un entero *cantidad* y disminuye la *cantidad* de *Pikinim* por el parámetro *cantidad*.
- *multiplicar(cantidad) – void:* Recibe un entero *cantidad* y aumenta la *cantidad* de *Pikinim* según la fórmula de cada color (**cantidad en estas fórmulas se refiere al parámetro de cantidad, no el campo de la clase *Pikinim***):
  - *Cyan*: $cantidad * 3$
  - *Magenta*: $cantidad * ataque$
  - *Amarillo*: $ceil(cantidad * 1.5)$

Los valores iniciales a considerar en el constructor de cada color son:

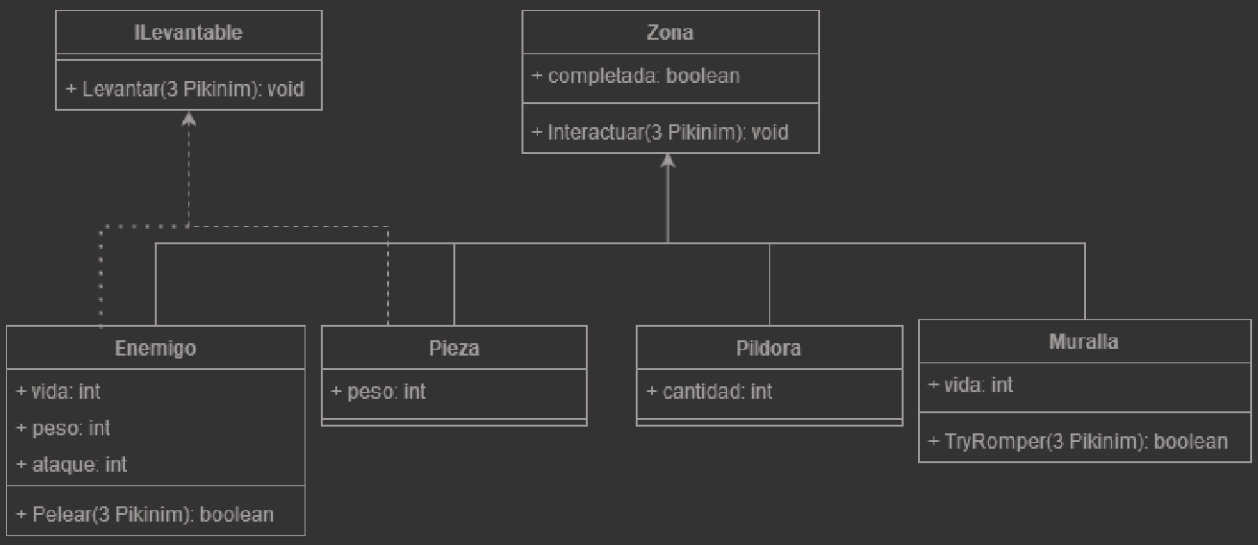
Cyan: ataque = 1; capacidad = 1

Magenta: ataque = 2; capacidad = 1

Amarillo: ataque = 1; capacidad = 3

4. Zonas

El mapa del juego se compone por *Zonas*, donde cada una interactúa con el jugador de forma distinta. La arquitectura de las clases se puede ver mejor en el siguiente diagrama:



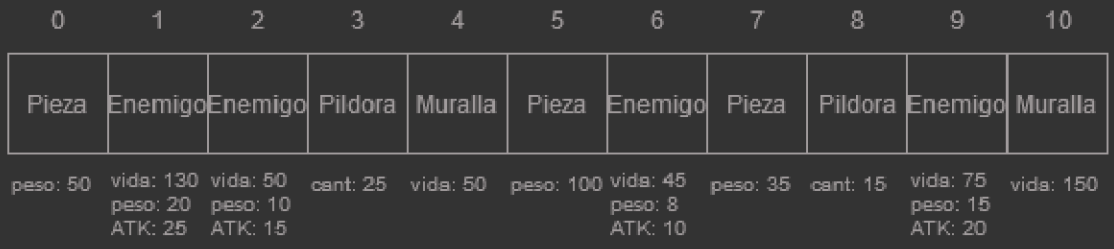
Para todas las clases, *Interactuar* es el método que se llama cuando el jugador llega a la *Zona* (recibe cada color de Pikinim). En el caso de la super clase *Zona,* implementa *Interactuar* solo para el caso base: donde ya se encuentra *completada*, por ejemplo, que diga "No queda nada que hacer aquí". Además, *ILevantable* es una interfaz implementada solo por *Enemigo* y *Pieza*. A continuación, se explica en qué consiste cada subclase de *Zona:*

- *Pieza: Zona* donde se encuentra una pieza de la nave.
  - *Interactuar* (sobrecarga)*:* Intenta levantar la pieza para llevarla a la nave.
  - *Levantar:* Se lleva la pieza si la suma de $cantidad * capacidad$ de los 3 colores de *Pikinim* es mayor o igual que el *peso* de la pieza. Si es así, debe registrar que se ha recuperado un tesoro. Hacer esto exitosamente deja *completada* la *Zona.*
- *Pildora: Zona* que aumenta la cantidad de un tipo de *Pikinim.*
  - *Interactuar* (sobrecarga)*:* Llama *multiplicar* de 1 color de *Pikinim* que elige el jugador, entregando *cantidad* como parámetro. Apenas termina esto se debe declarar como *completada.*
- *Muralla:* Mientras *vida* sea mayor a 0, el jugador no puede atravesar esa zona del mapa.
  - *Interactuar* (sobrecarga)*:* Intenta romper la muralla, si lo logra, queda *completada* la *Zona*.
  - *TryRomper:* Resta de *vida* la suma de $cantidad * ataque$ de los *Pikinim.*
- *Enemigo:* Zona donde se encuentra un enemigo.
  - *Interactuar* (sobrecarga)*:* Realiza una pelea entre los *Pikinim* y el enemigo, e intenta *levantar* al enemigo si es derrotado.

- *Pelear:* Resta de su *vida* la suma de $cantidad * ataque$ de los *Pikinim,* luego de esto, el enemigo ataca a los *Pikinim* (sin importar si su vida llegó a 0), reduciendo en *ataque* la *cantidad* de 1 color al azar. Si su *vida* llegó a 0, intenta levantarlo inmediatamente.
- *Levantar:* Se lleva al enemigo si la suma de $cantidad * capacidad$ de los 3 colores de *Pikinim* es mayor o igual que el *peso* del enemigo, si es así, debe *multiplicar* 1 color de *Pikinim* en *peso* (elegido por el jugador). Independiente de si se levanta o no, la *Zona* queda como *completada* una vez se derrota el enemigo.

5. Inicialización del juego

Cuando empieza el programa, se debe generar el mapa del juego, el cual es un arreglo del tipo *Zona* de tamaño 11. Cuando termina de generarse el mapa, se debe colocar el jugador en la posición 5 del mapa, con 10 *Pikinim* de cada color para iniciar el juego. Cada partida presenta el mismo mapa, donde cada celda de *Zona* sigue los valores que se presentan en el siguiente dibujo:



Las variables cant y ATK son versiones más cortas de cantidad y ataque.

6. Turnos del juego

En cada turno el jugador puede escoger entre moverse 1 celda a la izquierda o a la derecha, o quedarse en el mismo lugar. Cuando selecciona qué hacer, el programa debe revisar si la *Zona* a la que fue tiene acciones por hacer o no; si aún puede hacer algo con la *Zona*, debe llamar su método *Interactuar*, que mostrará por pantalla qué es lo que ocurre en ese turno.

Ejemplo del caso donde el jugador elige ir a una *Zona* con una píldora:

```
Lomiar llegó a un lugar lleno de unas figuras con forma de píldoras, los 30 Pikinim
se llevan las píldoras.

Qué color de pikinim desea que se multiplique? (cantidad a multiplicar: 10)
1.Cyan  2.Magenta  3.Amarillo

Input: 3
Los Pikinim amarillos han aumentado su cantidad en 15.
```

Al terminar de interactuar con una *Zona* puede ocurrir que ya no queden más acciones disponibles en ella (por ejemplo: derrotar un enemigo o romper una muralla), esto quiere decir que queda *completada*; en tal caso, el jugador no realizará ninguna acción al volver a esa *Zona* en cualquier turno posterior. Por otro lado, si una *Zona* no queda como *completada*, volver a ella continuará la acción que quedó pendiente.

7. Detalles Importantes

- El jugador **siempre** se mueve a la celda que indicó, si es que no termina con el uso de la *Zona*, puede reintentar interactuar con ella en el siguiente turno con la opción de quedarse en el mismo lugar.
- *Muralla* es la única *Zona* que detiene el paso del jugador, bloqueando el paso hacia la siguiente celda. Es importante notar que esto es situacional: si el jugador viene por la derecha, no puede avanzar a la celda a la izquierda de la *Muralla,* y viceversa.
- Es importante que el jugador sepa todos los datos necesarios para que pueda tomar decisiones, como lo pueden ser las zonas a las que puede ir, el estado de sus *Pikinim,* vida de los enemigos, etc.
- El formato en el que muestra la información queda a decisión del alumno, pero procure que la interfaz sea clara.
- Todas las variables de las clases son *private* o a lo más *protected*, por lo que requiere hacer *getters* y *setters* para cada campo.

8. Sobre la Entrega

- El código debe venir indentado y ordenado.
- Se deberá entregar un .tar.gz con los siguientes archivos:
    - Pikinim.java, Cyan.java, Magenta.java, Amarillo.java, Zona.java, Pieza.java, Enemigo.java, Pildora.java, Muralla.java, Juego.java, ILevantar.java.
    - makefile
    - README.txt
- **Todas las clases a realizar deben seguir fielmente lo que se enseña en los diagramas de clases, además de lo solicitado textualmente en este enunciado.**
- Las funciones deberán ir comentadas, explicando clara y brevemente lo que realiza, los parámetros que recibe y los que devuelve (en caso de que devuelva algo). Se deja libertad al formato del comentario.
- Los ayudantes correctores pueden realizar descuentos en caso de que el código se encuentre muy desordenado.
- Debe estar presente el archivo MAKEFILE para que se efectúe la revisión, este debe compilar TODOS los archivos.
- Se debe trabajar de forma individual.

- La entrega debe realizarse en tar.gz y debe llevar el nombre: Tarea3LP_RolAlumno.tar.gz. Ejemplo: *Tarea3LP_202200000-0.tar.gz*
- El archivo README.txt debe contener nombre y rol del alumno e instrucciones detalladas para la compilación y utilización de su programa.
- La entrega será vía aula y el plazo máximo de entrega es hasta el 06 de octubre de 2023 a las 23:55 hora aula.
- Las copias serán evaluadas con nota 0 y se informarán a las respectivas autoridades.
- Si el makefile no está bien realizado, la tarea no se revisará.
- Se utilizará el compilador javac versión 11.
- Solo se pueden hacer consultas sobre la Tarea hasta 1 día antes de la fecha de entrega.

9. Calificación

9.1. Entrega

Para la calificación de su tarea, debe realizar una entrega con requerimientos mínimos que otorgarán 30 pts base, luego se entregará puntaje dependiendo de los otros requerimientos que llegue a cumplir.

9.1.1. Entrega Mínima

Para obtener el puntaje mínimo, el programa debe crear el mapa solicitado en el inciso **Inicialización del juego**, permitiendo al jugador recorrerlo sin interactuar con él, pero mostrando qué *Zona* está pisando. (30 pts. Total)

Para recorrer el mapa, la línea de comandos **podría** verse como en el siguiente ejemplo:

```
Turno 1 (Cyan – 10, Amarillo – 10, Magenta – 10)

Zona Actual: Pieza (peso - 100)

Opciones:

1. Ir a derecha (Enemigo) 2. Ir a la izquierda (Muralla) 3. Quedarse aquí
```

9.1.2. Entrega

Luego de cumplir con la Entrega Mínima, puede obtener más puntaje cumpliendo con los siguientes puntos (puede haber puntaje parcial por cada punto):

- Implementar la clase *Pikinim* (total 20 pts)
  - Método *multiplicar()* para cada subclase. (10 pts)
  - Método *disminuir()*. (5 pts)
  - Constructores, getters y setters de cada clase y subclase. (5 pts)
- Implementar Zona (total 10pts)
  - Constructores, getters y setters (2 pts)
  - Uso correcto de *completado*. Bloquear o permitir las acciones de las *Zonas* según la variable. (5 pts)
  - Implementar y usar la versión básica de *Interactuar* mediante *super*. (3 pts)
- Implementar Enemigo (total 10 pts)
  - Constructores, getters y setters (2 pts)
  - Implementar *Interactuar* + *Pelea* (5 pts)
  - Implementar *Interactuar* + *Pelea* + *Levantar* (3 pts)
- Implementar Píldora (total 5 pts)
  - Constructores, getter y setter (1 pts)
  - Implementar *Interactuar* (4 pts)
- Implementar Muralla (total 10 pts)
  - Constructores, getters y setters (2 pts)
  - Implementar *Interactuar* (3 pts)
  - Bloquear y desbloquear el camino. (5 pts)
- Implementar Pieza (total 5 pts)
  - Constructores, getter y setter (1 pts)
  - Implementar *Interactuar* y *Levantar* (4 pts)
- Condición de Victoria / Pérdida. (total 10 pts)
- **Puntaje extra:** Creatividad en la interfaz de usuario (formato bonito para los turnos, texto que haga más amena la experiencia, etc). (10 pts)

9.2 Descuentos

Luego de cumplir con la Entrega Mínima, puede sufrir descuentos por los siguientes motivos:

- Falta de comentarios (-10 pts c/u Max 30 pts)
- Código no compila (-100 pts)
- Warning (c/u -5 pts)
- Falta de README (-20 pts)
- Falta de alguna información obligatoria en el README (-5 pts c/u)
- Falta de orden (entre -5 y -20 pts dependiendo de que tan desordenado)
- Día de atraso (-20 pts por día, -10 pts dentro de la primera hora)
- Mal nombre en algún archivo entregado (-5 pts c/u)