Diseño y Programación Orientada a Objetos - ELO329

Documento Tarea 1: Simulación de una mascota virtual en java

Hector Cepeda Bleck (ROL: 201921025-4)
Martín Alonso Karle (ROL: 201921021-1)
Bastián Rivas Cordero (ROL: 201921003-3)
Claudio Zanetta Penna (ROL: 202021030-6)

Introducción:

La tarea consistió en desarrollar un programa en Java que simule el cuidado de una mascota virtual, que puede interactuar con diferentes tipos de ítems almacenados en un inventario. El programa se divide en varias etapas, siendo la última etapa la implementación del inventario y la interacción de la mascota con los ítems del inventario.

Clases implementedas

Para la solución de esta tarea se implementaron 8 clases:

1. Main
2. Mascota
3. Estado
4. Inventario
5. Item
6. Juguete
7. Comida
8. Medicina

Etapas:

3.1 Etapa 1

En esta etapa se realiza la inicialización de la clase `Main` y se crea la clase `Mascota`. La clase `Mascota` representará a la mascota virtual del programa, conteniendo los atributos y métodos necesarios para modelar su comportamiento y estado.
Implementación de la clase `Mascota`:

- **Atributos:**
  - `edad`: Edad de la mascota (inicializada en 0).
  - `salud`: Salud de la mascota (inicializada en 100).
  - `energía`: Energía de la mascota (inicializada en 100).
  - `felicidad`: Felicidad de la mascota (inicializada en 50).

3.2 Etapa 2

En esta etapa, se diseñaron y crearon las clases que representan los diferentes ítems que pueden ser utilizados por la mascota virtual: Comida, Medicina y Juguete. Estas clases heredan de la clase abstracta `Item`, que define los atributos comunes a todos los ítems, como el ID, el nombre y la cantidad disponible.
Implementación de las clases:

- **Clase abstracta `Item`:**
  - Define los atributos comunes a todos los ítems, como el ID, el nombre y la cantidad disponible.
  - Define un método abstracto `usarItem` que debe ser implementado por las subclases para especificar el efecto del ítem sobre la mascota.
  - Implementa métodos `getters` y `setters` para acceder y modificar los atributos.
- **Subclases `Comida`, `Medicina` y `Juguete`:**
  - Representan los diferentes tipos de ítems que pueden ser utilizados por la mascota.
  - Implementan el método `usarItem` de acuerdo a su función específica.
  - Constructor: Permite inicializar los atributos de un ítem cuando se crea una instancia.

La implementación de estas clases proporciona la base para la interacción entre la mascota virtual y los diferentes tipos de ítems disponibles en el programa.

3.3 Etapa 3

En esta etapa, se desarrolló la clase `Inventario`, encargada de almacenar instancias de ítems correspondientes a las clases `Comida`, `Medicina` y `Juguete`. Además, se implementó la funcionalidad para interactuar con el inventario, seleccionar ítems y usarlos en la mascota virtual.
Implementación de la clase `Inventario`:

- **Atributos:**
  - `items`: ArrayList que almacena las instancias de ítems disponibles en el inventario.
- **Métodos:**
  - `agregarItem(Item item)`: Permite agregar un ítem al inventario.
  - `existeItem(int id)`: Verifica si un ítem con el ID especificado ya existe en el inventario.
  - `usarItem_Inventario(Mascota mascota, int id)`: Permite que la mascota virtual use un ítem del inventario. Si se usa un ítem, se consume y se aplica su efecto en la mascota. Si la cantidad de un ítem llega a cero, se elimina del inventario.
  - `borrarItem(Item item)`: Elimina un ítem específico del inventario.
  - `mostrarInventario()`: Muestra por consola los ítems disponibles en el inventario, junto con su ID y cantidad.

La implementación de la clase `Inventario` proporciona la funcionalidad necesaria para gestionar los ítems disponibles y su interacción con la mascota virtual.

3.4 Etapa 4

En esta etapa, se implementa la lectura de un archivo de configuración `config.csv` para inicializar el inventario y el nombre de la mascota. Además, se completa el menú de acciones para permitir nuevas funcionalidades como hacer que la mascota duerma en un momento determinado, continuar la simulación sin realizar ninguna acción y terminar la simulación en cualquier momento.

Implementación

(a) **Archivo de Configuración:**
- Se espera un archivo CSV (`config.csv`) como entrada del programa.
- El archivo debe tener la siguiente estructura:
  - La primera línea contiene el nombre de la mascota.

  - Las líneas siguientes contienen la información de los ítems del inventario en el formato: `ID;Tipo;Nombre;Cantidad`.

(b) **Proceso de Lectura:**

- Se lee el archivo línea por línea.
- Se divide cada línea en campos usando el delimitador `;`.
- Se crea un nuevo ítem de acuerdo al tipo especificado y se agrega al inventario.

(c) **Menú de Acciones:**

- Se muestra el estado actual del tiempo simulado, los atributos de la mascota y las acciones disponibles.
- Se permite al usuario seleccionar un elemento del inventario para usarlo en la mascota, dormir, continuar la simulación o salir.
- La simulación se detiene si la mascota muere o si el usuario elige salir.

Ejemplo de Ejecución:

```
$ java Main config.csv
tiempo simulado: 0.5
Atributos
---------
Nombre: Garfield
Edad: 0.5
Salud: 95
Energía: 95
Felicidad: 45
Estado: (-_-) Meh....
Acciones
--------
0: dormir
1: Pelota, cantidad 4
2: Queso, cantidad 5
3: Pan, cantidad 3
4: Jarabe, cantidad 4
Seleccione un elemento del inventario, 'c' para continuar, y 'x' para salir: 1
Usando Juguete Pelota
...
```

El código del `Main` proporcionado se encarga de realizar estas operaciones de manera ordenada y eficiente.

3.5 Etapa Puntaje Extra

Para mantener un registro de las estadísticas a lo largo del tiempo de juego se importaron las librerías `BufferedWriter` y `FileWriter` para ir generando un archivo .csv llamado `ultima_partida.csv`. Dada la implementación actual, este archivo retiene únicamente lo de la última partida hecha y se sobreescribe al jugar de nuevo.

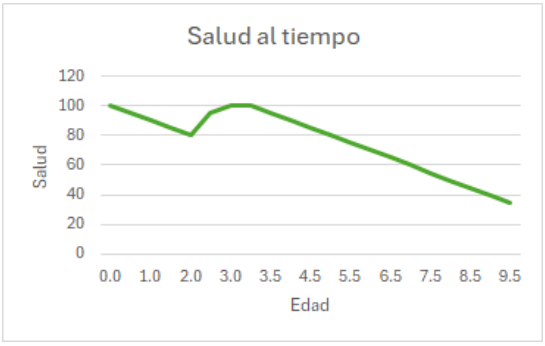
Toda la parte extra fue implementada dentro de la etapa 4, distinguiendo las partes correspondientes con comentarios en el mismo código.

Un ejemplo de lo generado se presenta en el siguiente cuadro de texto:

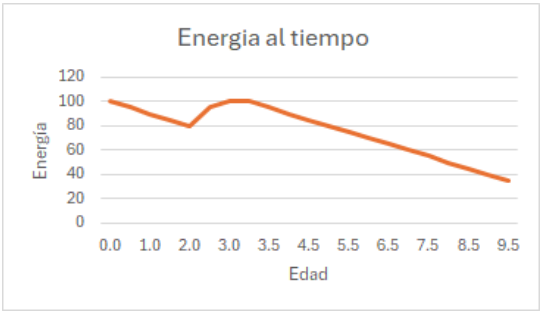
```
Edad; Salud; Energia; Felicidad
0.0;100;100;50
0.5;95;95;45
1.0;90;90;40
1.5;85;85;35
2.0;80;80;30
2.5;95;95;25
3.0;100;100;20
3.0;100;100;20
3.5;95;95;45
4.0;90;90;40
```

```
4.5;85;85;35
5.0;80;80;30
5.5;75;75;25
6.0;70;70;20
6.5;65;65;15
7.0;60;60;10
7.5;55;55;5
8.0;50;50;0
8.5;45;45;0
9.0;40;40;0
9.5;35;35;0
```

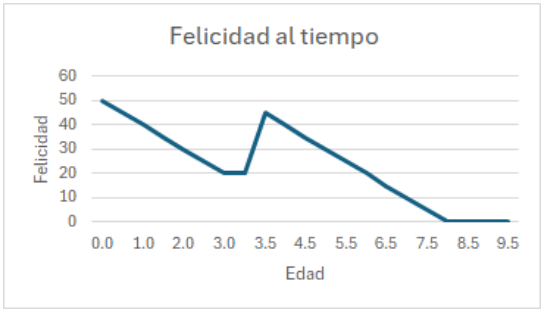
Usando los datos anteriores, se obtuvieron los gráficos de la figura 1.



(a) Salud a lo largo de la partida



(b) Energía a lo largo de la partida



(c) Felicidad a lo largo de la partida

Figure 1: Gráficos generados luego de una partida

Dificultades:

Gran parte de las dificultades a la hora de elaborar la tarea radicaron en la familiarización con el lenguaje y el paradigma inherente a este, se comprendían los objetivos y la manera en que se debían realizar ciertas acciones, pero a la hora de implementarlas no resultaba tan sencillo al no conocer con certeza las herramientas propias de Java.

En el contexto de la implementación del inventario y su interacción con la mascota: Se tuvo que diseñar una estructura de datos adecuada (empleando la clase **ArrayList**) para almacenar los ítems del inventario, fue necesario investigar y comprender el correcto uso de los métodos de dicha clase para así implementar eficientemente métodos que permitirían agregar, eliminar y seleccionar ítems del inventario. Fue necesario además hacer uso de otras clases existentes en Java que solucionan ciertas limitaciones de la propia clase **ArrayList** para finalmente lograr una adecuada interacción entre el inventario, usuario y mascota.

La lectura del archivo **CSV** requirió un mayor tiempo de estudio de la documentación con el cual finalmente se logró leer, extraer y asignar correctamente los datos entregados, logrando así inicializar tanto el inventario como el nombre de la mascota de manera más libre y abierta. En síntesis, el estudio de la documentación permitió un manejo adecuado de la lectura de archivos y un adecuado procesamiento de los datos para configurar el estado inicial del programa.

Diagrama UML:

Se muestra el diagrama UML generado con `IntelliJ` en la figura 2.

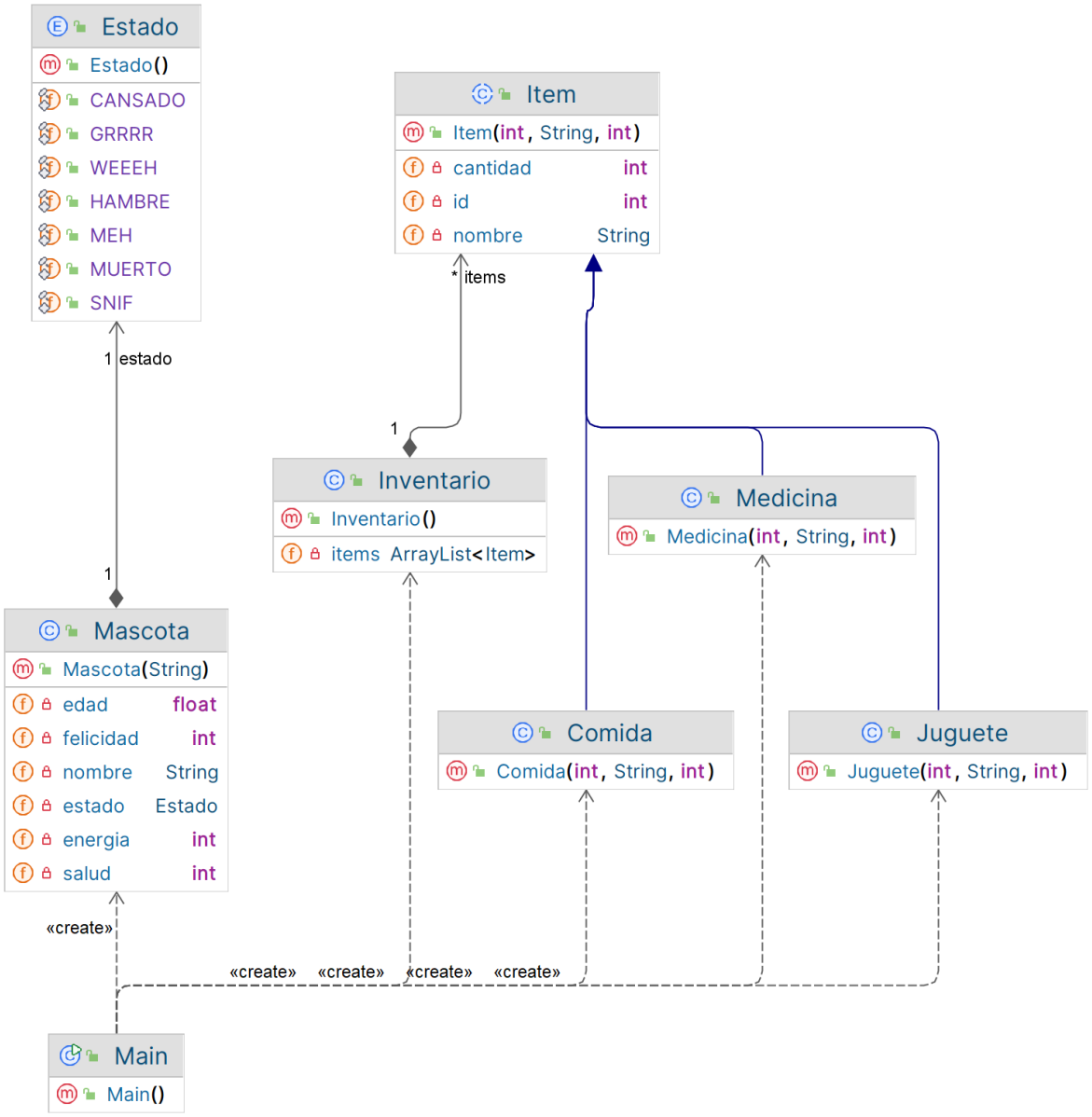


Figure 2: Diagrama UML generado