Diseño y Programación Orientados a Objetos

1er. Semestre 2024

Tarea 1: Simulación de una mascota virtual en Java

Lea detenidamente la tarea. **Si algo no lo entiende, consulte. Si es preciso, se incorporarán aclaraciones al final**. Esta interacción se asemeja a la interacción entre desarrolladores y clientes cuando algo no está del todo especificado.

1. Objetivos de la tarea:

- Modelar objetos reales como objetos de software.
- Ejercitar la creación y extensión de clases dadas para satisfacer nuevos requerimientos.
- Reconocer clases y relaciones entre ellas en código fuente Java.
- Ejercitar la compilación y ejecución de programas en lenguaje Java desde una consola por línea de comandos.
- Ejercitar la configuración de un ambiente de trabajo para desarrollar aplicaciones en lenguaje Java. Se puede trabajar con un editor tipo "Sublime" o con un IDE. IntelliJ y Visual Studio Code son los IDE sugeridos.
- Ejercitar la entrada y salida de datos en Java.
- Manejar proyectos vía Git (voluntario para esta tarea, pero recomendado).
- Conocer el formato .csv y su importación desde una planilla electrónica.
- Ejercitar la preparación y entrega de resultados de software (creación de `Makefile`, `Readme`, documentación).
- Familiarización con desarrollos "iterativos" e "incrementales" (o crecientes).

2. Descripción General

En esta tarea se implementará un programa bajo el paradigma de Programación Orientada a Objetos utilizando el lenguaje de programación Java. Este simulara el comportamiento de una **mascota virtual**. Para ello, tendrá que modelar tanto la mascota, como los diferentes elementos con los cuales puede interactuar.

Pueden ver un ejemplo de funcionamiento de mascota virtual en el siguiente enlace: Juego Pou

2.1 Mascota virtual

El modelo para una mascota virtual en esta tarea contiene las siguientes características e indicadores:

- **Nombre** de la mascota
- **Edad**, la cual está en *unidades de tiempo* de la simulación
- **Salud**, la cual toma valores entre 0 y 100 puntos
- **Energía**, la cual toma valores entre 0 y 100 puntos
- **Felicidad**, la cual toma valores entre 0 y 100 puntos

Algunos indicadores de la mascota pueden verse afectados entre sí:

- Si `edad` <= 5 y `salud`<=10 puntos, la `felicidad` bajará 20 puntos por cada incremento de tiempo de simulación.
- Si 5 < `edad` <= 10 y `salud`<=50, la `felicidad` bajará 20 puntos y la `energía` bajará 10 puntos por cada incremento de tiempo de simulación.
- Si `edad`>10 y `salud`<=50, la `felicidad` bajará 30 puntos y la `energía` bajará 20 puntos por cada incremento de tiempo de simulación.

En base a los indicadores de edad, salud, energía y felicidad, es posible determinar el **estado** en el que se encuentra la mascota. Estos estados están definidos como un dato tipo `enum` en los códigos de ayuda. A continuación se presentan la lista de estados posibles según orden de prioridad (de menor a mayor), así como sus condiciones específicas:

1. *(-_-) Meh....* **(Neutro)**: Estado por omisión de la mascota.
2. *\(^_^)/ Weeeeeh!* **(Feliz)** : La felicidad de la mascota es mayor o igual a 60 puntos.
3. *(._.) snif....* **(Triste)**: La felicidad de la mascota es menor o igual a 20 puntos.
4. *(0o0) hambre hambre!* **(Hambriento)**:
    - Si la salud es menor o igual a 20 puntos para una edad menor o igual a 5.
    - Si la salud es menor o igual a 50 puntos para una edad entre 5 y 10
5. *(ôwô) grrrr....* **(Enojado)**: La edad es mayor a 5, mientras la salud y energía son menores o iguales a 30 puntos
6. *(=_=) zzzz....* **(Cansado)**: La energía de la mascota es menor o igual a 15 puntos.
7. *(x_x) fin del juego* **(Muerto)**: La salud o la energía de la mascota ha alcanzado el valor mínimo de 0, o la edad es mayor o igual a 15.

En caso de verificarse las condiciones para más de un estado a la vez, se privilegia el estado de mayor prioridad.

La mascota, al interactuar con alguna instancia de un ítem, puede modificar sus indicadores y, en consecuencia, su estado.

También pueden modificarse los indicadores y estado de la mascota cuando se duerme. Al dormir, la mascota aumenta su energía al máximo (100 puntos), mientras que su felicidad y salud aumentan en 15 puntos respectivamente.

Tanto los indicadores como el estado de la mascota resultante se visualizan en cada tiempo de simulación tal como se ve en el siguiente ejemplo

```
Atributos
---------
Edad: 7.5
Salud: 75
Energía: 65
Felicidad: 35
Estado: (-_-) Meh....
```

El paso tiempo para la simulación de vida de la mascota virtual se simula a través de una variable `float`, cuyo incremento se establece en aumentos de 0.5 unidades de tiempo por cada paso de tiempo. Cada incremento de tiempo produce:

- La aparición de un menú, el cual permite interactuar con la mascota, continuar el paso del tiempo, o terminar el programa.
- Un aumento en la edad de la mascota, la cual aumenta en **0.5 unidades de tiempo**.
- La modificación de la salud, la energía y la felicidad de la mascota, las cuales disminuyan en **5 puntos cada una**.

2.2 Ítems

Los ítems corresponde a diferentes elementos con los que puede interactuar la mascota virtual. Estos se pueden agrupar en las siguientes categorías/clases:

- **Comida**, la cual, al ser utilizada en la mascota, le aumenta 20 puntos de energía y salud.
- **Medicina**, la cual, al ser utilizada en la mascota, le aumenta 40 puntos de salud.
- **Juguete**, el cual, al ser utilizado en la mascota, le aumenta 30 puntos de felicidad.

Cada instancia de un ítem podrá ser usado en la mascota **una única vez**. Esto incluye a los juguetes, los cuales deben ser considerados como *desechables*.

Adicionalmente, cada una de estas tres clases corresponde a sub clases de la clase base `Item`. Esta contiene todos los elementos comunes de los ítems. Esto incluye los siguientes atributos:

- `id`, el cual se utilizará para poder acceder a cada ítem.
- `cantidad`, correspondiente a la cantidad de elementos existente para dicho elemento.
- `nombre`, correspondiente al nombre de dicho ítem.

2.3 Inventario

El inventario contendrá todas las instancias de ítems disponibles. Este permitirá la adición o eliminación de ítems según sea necesario, garantizando una actualización constante de los elementos almacenados.

2.4 Configuración inicial

El programa contará con un archivo de configuración `config.csv` (referencia a archivos .csv), cuyos elementos se separan por `;` en este ejemplo. Este archivo contiene tanto el nombre de su mascota, así como los ítems que tendrá a disposición en su inventario siguiendo el siguiente formato:

```
Garfield
1;Juguete;Pelota;4
2;Comida;Queso;5
3;Comida;Pan;3
4;Medicina;Jarabe;4
```

Las líneas posteriores al nombre de la mascota contienen la siguiente información sobre cada ítem:

```
<identificador_ítem>;<tipo_item>;<nombre_item>;<cantidad_inicial>
```

3. Etapas del Desarrollo

3.1 Etapa 1: Inicialización de la clase Main y creación de la clase Mascota

En esta etapa se desarrollará la clase `Mascota` que representará a la mascota virtual del programa. Esta clase contendrá los atributos y métodos necesarios para modelar el comportamiento y estado de la mascota. Se enfocará en la definición de los atributos básicos como nombre, edad, salud, energía, felicidad y estado. No necesita el archivo `config.csv` en esta etapa.

Los valores iniciales para los indicadores de `Mascota` serán los siguientes

- `edad = 0`
- `salud = 100`
- `energia = 100`
- `felicidad = 50`

Puede utilizar opcionalmente el código de ayuda publicado en Aula como base para su desarrollo.

Ejemplo de salida esperada

```
$ java Main

Mascota Virtual

Atributos
---------
Nombre: Garfield
Edad: 0
Salud: 100
Energía: 100
Felicidad: 50
Estado: (-_-) Meh....
```

3.2 Etapa 2: Diseño e implementación de Ítems

En esta etapa se procederá a la creación tanto de la clase `Item` como de sus clases derivadas `Comida`, `Medicina` y `Juguete`. Estas clases estarán destinadas a representar los diferentes tipos de ítems que serán almacenados en el inventario más adelante. Las clases se deben dotar de los atributos y métodos necesarios para interactuar adecuadamente con la mascota. No necesita el archivo `config.csv` en esta etapa.

Para verificar la correcta implementación de esta etapa, se solicita lo siguiente:

1. Dentro del código de la clase `Main`, crear una instancia de cada una de las clases de ítems mencionadas anteriormente.
2. Aplicar cada instancia en la mascota, simulando una interacción con la misma.
3. Mostrar los resultados de estas interacciones por medio de la terminal.

También debe implementar y probar en esta etapa la acción `dormir` de la mascota. Muestre también el resultado de esta acción por la terminal.

Se proporcionan valores iniciales para los indicadores de la mascota:

- `edad = 0`
- `salud = 50`
- `energia = 50`
- `felicidad = 50`

Ejemplo de salida esperada

```
$ java Main

Atributos
---------
Nombre: Garfield
Edad: 0.0
Salud: 50
Energía: 50
Felicidad: 50
Estado: (-_-) Meh....

Dando de comer Queso...
Atributos
---------
Nombre: Garfield
Edad: 0.0
Salud: 70
Energía: 70
Felicidad: 50
Estado: (-_-) Meh....

Aplicando medicamento Jarabe...
Atributos
---------
Nombre: Garfield
Edad: 0.0
Salud: 100
Energía: 70
Felicidad: 50
Estado: (-_-) Meh....

Usando Juguete Pelota...
Atributos
---------
Nombre: Garfield
Edad: 0.0
Salud: 100
Energía: 70
Felicidad: 80
Estado: \(^_^)/ Weeeeeh!

Garfield ha dormido como un tronco!
Atributos
---------
Nombre: Garfield
```

```
Edad: 0.0
Salud: 100
Energía: 100
Felicidad: 95
Estado: \(^_^)/ Weeeeeh!
```

3.3 Etapa 3: Creación del inventario con ítems

En esta etapa se creará la clase `Inventario`,donde se almacenarán diversas instancias de ítems correspondientes a las clases `Comida`, `Medicina` y `Juguete`. Estas instancias deben ser almacenadas en un `ArrayList` de tipo `Item` dentro de la clase `Inventario`. No necesita el archivo `config.csv` en esta etapa.

También debe implementar el paso del tiempo para la simulación de la mascota. Cada acción de la mascota o interacción con un ítem, produce que el tiempo avance 0.5 unidades.

Adicionalmente, cree un menú que permita seleccionar alguno de los ítems presentes en el inventario utilizando su `id`. Una vez seleccionado el ítem, este debe ser utilizado por la mascota, y luego se incrementará el tiempo de simulación según lo indicado. Es importante recordar que cada ítem contiene el atributo `cantidad`, por lo tanto, cada vez que se utilice un ítem, la cantidad de dicho ítem debe ser actualizada. Cuando esa cantidad llega a cero, el ítem **debe ser borrado del inventario.**

Los valores iniciales para los indicadores de `Mascota` en esta etapa serán los siguientes

- `edad = 0`
- `salud = 100`
- `energia = 100`
- `felicidad = 50`

Para verificar la correcta implementación de esta etapa, se solicita lo siguiente:

- Dentro del código de la clase `Main`, se deberá crear un inventario con los siguientes ítems
  - id 1, Clase Juguete, Nombre Pelota, cantidad 4
  - id 2, Clase Comida, Nombre Queso, cantidad 5
  - id 3, Clase Comida, Nombre Pan, cantidad 3
  - id 4, Clase Medicina, Nombre Jarabe, cantidad 4
- Mostrar el tiempo de simulación a medida que este avanza. Es importante recordar que el tiempo afectará la edad de la mascota y sus atributos según lo explicado.
- Mostrar el estado de la mascota, atributos e indicadores en cada incremento de tiempo.
- Mostrar el estado del inventario en cada incremento de tiempo como un menú que permita seleccionar un ítem del mismo.
- Utilice el ítem seleccionado en la mascota y vuelva a mostrar el nuevo tiempo, el estado de la mascota y el estado del inventario.

Ejemplo de salida esperada

```
$ java Main

tiempo simulado: 0.5
Atributos
```

```
---------
Nombre: Garfield
Edad: 0.5
Salud: 95
Energía: 95
Felicidad: 45
Estado: (-_-) Meh....

Acciones
--------
1: Pelota, cantidad 4
2: Queso, cantidad 5
3: Pan, cantidad 3
4: Jarabe, cantidad 4
Seleccione un elemento del inventario: 1

Usando Juguete Pelota
tiempo simulado: 1.0
Atributos
---------
Nombre: Garfield
Edad: 1.0
Salud: 90
Energía: 90
Felicidad: 70
Estado: \(^_^)/ Weeeeeh!

Acciones
--------
1: Pelota, cantidad 3
2: Queso, cantidad 5
3: Pan, cantidad 3
4: Jarabe, cantidad 4
Seleccione un elemento del inventario: 2

Dando de comer Queso
tiempo simulado: 1.5
Atributos
---------
Nombre: Garfield
Edad: 1.5
Salud: 95
Energía: 95
Felicidad: 65
Estado: \(^_^)/ Weeeeeh!

Acciones
--------
1: Pelota, cantidad 3
2: Queso, cantidad 4
3: Pan, cantidad 3
4: Jarabe, cantidad 4
Seleccione un elemento del inventario:
...
```

3.4 Etapa 4: Inicialización de inventario a partir de archivo `config.csv`

En esta etapa se inicializará el inventario y el nombre de la mascota utilizando un archivo de entrada llamado `config.csv`. Este archivo debe ser pasado como parámetro del programa al ejecutarlo.

Ejemplo de ejecución:

```
java Main config.csv
```

Adicionalmente, complete el menú de la etapa anterior para que permita acceder las siguientes acciones adicionales:

- Hacer que la mascota pueda dormir en un cierto momento.
- Permitir continuar la simulación sin realizar ninguna acción.
- Terminar la simulación en cualquier instante.

Ejemplo de salida esperada

```
$ 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
tiempo simulado: 1.0
Atributos
---------
Nombre: Garfield
Edad: 1.0
Salud: 90
Energía: 90
Felicidad: 70
```

```
Estado: \(^_^)/ Weeeeeh!

Acciones
--------
0: dormir
1: Pelota, cantidad 3
2: Queso, cantidad 5
3: Pan, cantidad 3
4: Jarabe, cantidad 4
Seleccione un elemento del inventario, 'c' para continuar, y 'x' para salir: c

tiempo simulado: 1.5
Atributos
---------
Nombre: Garfield
Edad: 1.5
Salud: 85
Energía: 85
Felicidad: 65
Estado: \(^_^)/ Weeeeeh!

Acciones
--------
0: dormir
1: Pelota, cantidad 3
2: Queso, cantidad 5
3: Pan, cantidad 3
4: Jarabe, cantidad 4
Seleccione un elemento del inventario, 'c' para continuar, y 'x' para salir: 2

Dando de comer Queso
tiempo simulado: 2.0
Atributos
---------
Nombre: Garfield
Edad: 2.0
Salud: 95
Energía: 95
Felicidad: 60
Estado: \(^_^)/ Weeeeeh!

Acciones
--------
0: dormir
1: Pelota, cantidad 3
2: Queso, cantidad 4
3: Pan, cantidad 3
4: Jarabe, cantidad 4
Seleccione un elemento del inventario, 'c' para continuar, y 'x' para salir: c

tiempo simulado: 2.5
Atributos
---------
Nombre: Garfield
```

```
Edad: 2.5
Salud: 90
Energía: 90
Felicidad: 55
Estado: (-_-) Meh....

Acciones
--------
0: dormir
1: Pelota, cantidad 3
2: Queso, cantidad 4
3: Pan, cantidad 3
4: Jarabe, cantidad 4
Seleccione un elemento del inventario, 'c' para continuar, y 'x' para salir: 4

Aplicando medicamento Jarabe
tiempo simulado: 3.0
Atributos
---------
Nombre: Garfield
Edad: 3.0
Salud: 95
Energía: 85
Felicidad: 50
Estado: (-_-) Meh....

Acciones
--------
0: dormir
1: Pelota, cantidad 3
2: Queso, cantidad 4
3: Pan, cantidad 3
4: Jarabe, cantidad 3
Seleccione un elemento del inventario, 'c' para continuar, y 'x' para salir: x

Fin de la simulación
```

3.5 Extra crédito: Salida de datos y utilización

Para esta etapa opcional, se solicita imprimir en un archivo con formato `.csv` el valor de los atributos de la mascota en cada instante de tiempo durante la simulación. Posteriormente, se debe importar este archivo en un programa de hojas de cálculo (MS Excel, Google Sheets, Libre Office Calc, etc.) y generar uno o múltiples gráficos que muestren la evolución de dichos atributos en función del tiempo.

Esta parte es voluntaria, su desarrollo otorga 10 puntos adicionales (la nota final se satura en 100). Si desarrolla esta parte, indíquelo en su documentación agregando también los gráficos obtenidos.

4. Elementos a considerar en su documentación

Entregue todo lo indicado en "Normas de Entrega de Tareas" (entrega por github/gitlab **no aplica en esta tarea**).

Prepare un archivo makefile para compilar y ejecutar su tarea en aragorn con rótulo `run`. Además, incluya rótulos `clean` para borrar todos los `.class` generados. Los comandos a usar en cada caso son:

```
$ make    /* para compilar */
$ make run  /* para ejecutar la tarea */
$ make clean  /* para borrar archivos .class */
```

En su archivo de documentación (`pdf` o `html`) incorpore el diagrama de clases de la aplicación (Etapa 4).

Entregue su desarrollo en un archivo comprimido (`.zip`, `.rar`, `.tar.gz`, etc.), que incluya su desarrollo por cada etapa separado por carpetas.