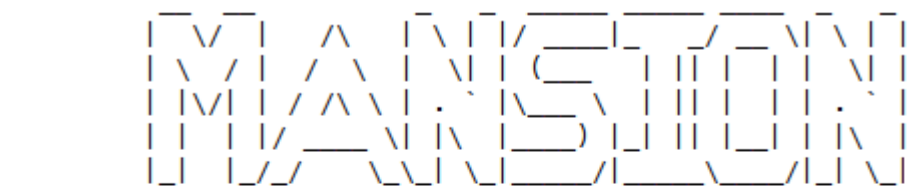
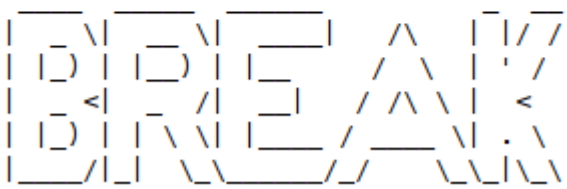
Programación (DM1D2)

Práctica 1





Álvaro de la Vega Olmedilla

Práctica 1

MansionBreak

Índice de contenido

1. Planificación

En un principio, la práctica se diseñó teiendo en cuenta que iba a tener una especie de interfaz gráfica con caracteres ASCII; pero se descartó una vez empezado el proyecto por la complejidad, la falta de compatiblidad entre sistemas operativos, uno de los pilares del Ciclo Formativo de Grado Superior que estamos cursando actualmente, y la cantidad de tiempo necesario para llevarlo acabo, que se alargaría más allá de la fecha de entrega del proyecto.

También se pensó hacer un menú en el que el usuario elegiría el nivel de dificultad del juego, lo mismo que el argumento que se le pasa al programa al ejecutarlo (1 para el nivel facil, 0 para el nivel normal) y un posible tercer tipo de argumento que hubiera sido el nivel dificil en el que el jugador podría romper jarrones y mover objetos como las mesas y las sillas y tener un nivel de vida él y los monstruos, tener varios objetos del mismo tipo, etc. Esto también se descartó por las mismas razones que lo anterior, demasido tiempo para llevarlo a cabo, incompatiblidad de codificaciones entre sistemas operativos no UNIX y la extremada complejidad que alcanzaría el proyecto.

Finalmente, el proyecto se planteó de una forma más sencilla y sin añadidos al producto encargado. Lo únco que se puede decir que no está dentro de lo normal sería que el código está escrito en su totalidad en inglés, por la razón de que así sería más ampliable en el futuro si se convierte en un proyecto más "global" o se decide subir el código del proyecto a Internet, además de ahorrar problemas por la codificación diferente en distintos sistemas operativos.

2. Estructura del proyecto

El proyecto internamente se ha dividido en tres "paquetes": characters, mansion y things; los cuales se explican a continuación detalladamente. También se han creado varias carpetas en la raíz del proyecto: src, donde está todo el código fuente; doc, donde está toda la documentación necesaria, incluyendo este documento; y dentro de doc un carpeta llamada javadoc, que como su nombre indica allí se encuentra el javadoc generado a partir del código fuente.

2.1. Paquete characters

El paquete characters contiene todas las clases y las interfaces que tienen relación con los personajes, las cuales se detallan a continuación.

2.1.1. Interfaz CharactersInterface

Esta interfaz define todos los métodos básicos que debe tener un personaje, los cuales son:

- **int[] getLocation()**. Es un método que devuelve un array unidimensional con tres valores, que se corresponden así: 0 con la componente X, 1 con la componente Y y 2 con la componenete Z. Sirve para poder saber la localización de un personaje dentro de la mansión.
- **void setLocation(int[] location)**. Es un método con el cual se guarda una nueva localización al personaje, dicha localización se le pasa al método como parámetro, siendo un array unidimensional con tres valores.
- **boolean isAlive()**. Es un método que sirve para indicar si el persoanje está vivo o muerto. Si devuelve *True*, el personaje está vivo; si devuelve *False*, está muerto.
- **void setAlive(boolean alive)**. Es un método para decir si un personaje está vivo, si se introduce *True*; o si está muerto, introduciendo *False*.

2.1.2. Clases

2.1.2.1. Clase Character

Implementa de la interfaz CharactersInterface por lo que tiene los métodos anteriormente descritos. De ella extentienden todas las clases de los monstruos y del jugador.

2.1.2.2. Clase Player

Extiende de la clase madre Character, por lo que hereda de ella todos sus métodos. Define los métodos que exclusivamente tiene el jugador. Tiene un constructor con parametro de tipo Mansion y Bag. El de tipo de Mansion indica la mansión en la que estará el jugador, el de tipo Bag indica la mochila que llevará el jugador. Se definen los siguientes métodos adicionales:

- **public Bag getBag()**. Método que devuelve un objeto de tipo Bag, que es la mochila que está usando el jugador.
- **public void setBag(Bag bag)**. Método que al introducir un objeto de tipo Bag, se cambia la mochila que está usando el jugador.
- **public void move(String movement)**. Método que interpreta los comandos de movimiento que puede realizar el jugador. Estos comandos se pasan al método en forma de String, pudiendo contener los valores: *mn*, *ms*, *mo*, *me*, *s* y *b*.

2.1.2.3. Clase Ghost

Extiende de la clase madre Character, por lo que hereda de ella todos sus métodos. Define los métodos que exclusivamente tiene el fantasma. Tiene un constructor al que se le pasa un parametro que es un array unidimensional con tres valores que indica la localización dentro de la mansión. Se define el siguiente método adicional:

- **public void kill()**. Es un método que mata al fantasma.

2.1.2.4. Clase Vampire

Extiende de la clase madre Character, por lo que hereda de ella todos sus métodos. Define los métodos que exclusivamente tiene el vampiro. Tiene un constructor al que se le pasa un parametro que es un array unidimensional con tres valores que indica la localización dentro de la mansión. Se definen los siguientes métodos adicionales:

- **public void kill()**. Es un método que mata al vampiro si está previamente debilitado.
- **public boolean isWeakened()**. Método si que indica si el vampiro ha sido debilitado o no previamente.
- **public void setWeakened(boolean weakened)**. Método para decir si el vampiro ha sido debilitado o no.

2.1.2.5. Clase Zombie

Extiende de la clase madre Character, por lo que hereda de ella todos sus métodos. Define los métodos que exclusivamente tiene el zombie. Tiene un constructor al que se le pasa un parametro que es un array unidimensional con tres valores que indica la localización dentro de la mansión.Se define el siguientes método adicional:

- **public void kill()**. Es un método que mata al zombie.

2.2. Paquete mansion

2.2.1. Clases

2.2.1.1. Clase Room

Crea todo lo necesario para el funcionamiento de una habitación.Tiene un constructor con los siguientes parámetros:

- **ArrayList<Character> characters**. ArrayList que guarda todos los personajes que pueda haber en la habitación.
- **ArrayList<Thing> things**. ArrayList que guarda todos los objetos que pueda haber en la habitación.

También contiene los siguientes métodos:

- **public ArrayList<Character> getCharacters()**. Devuelve un ArrayList que contiene todos los personajes que pueda haber en la habitación.
- **public ArrayList<Thing> getThings()**. Devuelve un ArrayList que contiene todos los objetos que pueda haber en la habitación.
- **public void addCharacter(Character character)**. Añade un personaje a la habitación.
- **public void addThing(Thing thing)**. Añade un objeto a la habitación.
- **public int getTablesNumber()**. Devuelve el número de mesas que hay en la habitación.
- **public int getChairsNumber()**. Devuelve el número de sillas que hay en la habitación.
- **public int getVasesNumber()**. Devuelve el número de jarrones que hay en la habitación.

2.2.1.2. Clase Mansion

Es la clase que crea todas las habitaciones, introduce en ellas los personajes y objetos de forma aleatoria. Su constructor, solamente llama al método `reallocate()` porque este método es el que genera todo facilmente si te encuentras al fantasma. Tiene los siguientes métodos:

- `public Ghost getGhost()`. Devuelve el fantasma.
- `public Vampire getVampire()`. Devuelve el vampiro.
- `public Zombie getZombie()`. Devuelve el zombie.
- `public Ax getAx()`. Devuelve el hacha.
- `public BasicVampireKillingKit getBasicVampireKillingKit()`. Devuelve el Kit Básico Mata-Vampiros.
- `public Cross getCross()`. Devuelve la cruz.
- `public FruitJuice getFruitJuice()`. Devuelve el zumo de frutas.
- `public Vacuum getVacuum()`. Devuelve la aspiradora.
- `public void reallocate()`. Resdistribuye toda la mansión.
- `public int[] getCurrentRoomLocation()`. Devuelve las coordenadas de la habitación en la que se encuentra el jugador.

- `public void setCurrentRoomLocation(int[] currentRoomLocation)`. Cambia la habitación en la que se encuentra el jugador
- `public Room getCurrentRoom()`. Devuelve la habitación de la habitación en la que está el jugador.

2.3. Paquete things

2.3.1. Interfaz ThingsInterface

Define todos los métodos que deben tener todos los objetos, los cuales son:

- `int[] getLocation()`. Devuelve la localización del objeto dentro de la mansión.
- `void setLocation(int[] location)`. Introduce una nueva localización para el objeto.
- `boolean isUsable()`. Define si se puede usar el objeto o no.
- `void setUsable(boolean usable)`. Cambie si el objeto se puede usar o no.

2.3.2. Clases

2.3.2.1. Clase Thing

Clase que implementa la interfaz `ThingsInterface`, por lo que tiene los métodos anteriormente descritos. De ella extienden todas las clases de los objetos. Y tiene un cosntructor al que hay que indicarle la localización del objeto y si se puede usar o no el objeto.

2.3.2.2. Clase Ax

Extiende de `Thing`. Clase que define el hacha. Tiene un constructor al que se le pasa la localización del hacha y se asume que se puede usar. Tiene un método `public void use(Zombie zombie)` con el que se mata al zombie.

2.3.2.3. Clase BasicVampireKillingKit

Extiende de `Thing`. Clase que define el Kit Básico Mata-Vampiros. Tiene un constructor al que se le pasa la localización del Kit Básico Mata-Vampiros y se asume que se puede usar. Tiene un método `public void use(Vampire vampire)` con el que se mata al vampiro.

2.3.2.4. Clase Cross

Extiende de `Thing`. Clase que define la cruz. Tiene un constructor al que se le pasa la localización del cruz y se asume que se puede usar. Tiene un método `public void use(Zombie zombie)` con el que se debilita al vampiro.

2.3.2.5. Clase FruitJuice

Extiende de `Thing`. Clase que define el zumo de frutas. Tiene un constructor al que se le pasa la localización del zumo de frutas y se asume que se puede usar.

2.3.2.6 Clase Vacuum

Extiende de `Thing`. Clase que define el fantasma. Tiene un constructor al que se le pasa la localización del fantasma y se asume que se puede usar. Tiene un método `public void use(Vacuum vacuum)` con el que se mata al fantasma.

2.3.2.7. Clase Bag

Define la mochila. El límite de la mochila se define en la constante `DEFAULT_BAG_LIMIT`, la cual vale `5`. Tiene un cosntructor al que se le pasa un array de `Thing` que contendrá los objetos, otro al que se le pasa el límite que se quiera y otro que crea el límite por defecto (`5`).Tiene los siguientes métodos:

- `public Thing[] getThings()`. Devuelve toadas las cosas de la mochila
- `public void setThings(Thing[] things)`. Cambia todas las cosas de la mochila.
- `public int getThingsNumber()`. Devuelve la cantidad de cosas que hay en la mochila.
- `public static int getDefaultBagLimit()`. Devuelve el número límite de la mochila por defecto.
- `public boolean addThing(Thing thing)`. Añade un objeto a la mochila.
- `public boolean deleteThing(Thing thing)`. Elimina un objeto de la mochila.

2.3.2.8. Clase Chair

Extiende de `Thing`. Clase que crea las sillas. Tiene un constructor al que se le pasa la localización de la silla en la mansión. Se asume que no se puede usar.

2.3.2.9. Clase Door

Extiende de `Thing`. Clase que crea las puertas. Tiene un constructor al que se le pasa la localización de la puerta en la mansión. Se asume que no se puede usar.

2.3.2.10. Clase Stairway

Extiende de `Thing`. Clase que crea las escaleras. Tiene un constructor al que se le pasa la localización de las escaleras en la mansión. Se asume que no se puede usar aunque en realidad a través de comandos se pueden subir y bajar.

2.3.2.11. Clase Table

Extiende de `Thing`. Clase que crea las mesas. Tiene un constructor al que se le pasa la localización de la mesa en la mansión. Se asume que no se puede usar.

2.3.2.12. Clase Vase

Extiende de `Thing`. Clase que crea los jarrones. Tiene un constructor al que se le pasa la localización del jarrón en la mansión. Se asume que no se puede usar.

2.3.2.13 Clase Window

Extiende de `Thing`. Clase que crea las ventanas. Tiene un constructor al que se le pasa la localización de la ventana en la mansión. Se asume que no se puede usar aunque en realidad a través de los comandos solamente se puede utilizar una.

2.4. Clases sin paquete

2.4.1 Clase Main

Es la clase pricipal del proyecto, desde ella se llama a la mansión, a la mochila, al jugador, se representa todo en pantalla y se recogen los comandos del teclado y se interpretan.

2.4.2. Clase Util

Es la clase que contiene algunos métodos y constantes que se usan mucho en todo el proyecto y en varias clases. Contiene las siguientes constantes:

- Las que defienen los límites máximos de la mansión
  - `public static final int MANSION_MAX_X = 4`
  - `public static final int MANSION_MAX_Y = 3`
  - `public static final int MANSION_MAX_Z = 3`
- Las que definen las dimensiones máximas de las habitaciones (pensado para futuras ampliaciones):
  - `public static final int ROOM_MAX_X = 5`
  - `public static final int ROOM_MAX_Y = 7`
- Las que por comodidad definen las compnentes de las coordenadas en los arrays:
  - `public static final int X = 0`
  - `public static final int Y = 1`
  - `public static final int Z = 2`

Y los siguientes métodos estáticos:

- `public static int[] randomCoordinates()`. Genera unas coordeandas aletorias para meter objetos dentro de una habitación y representarlos. Actualmente sin uso, pero sirvee para una futura ampliación.
- `public static int[] randomLocation()`. Genera una localización aleatoria.
- `public static int randomNumber(int min, int max)`. Genera un número aleatorio entre los dos números que se indican
- `public static boolean compareLocations(int[] location1, int[] location2)`. Compara dos array de localización para saber si apuntan a la misma localización.

3. Comentarios

Todo el proyecto ha estado en un repositorio del sistema de control de versiones Git.