Un personaje y su inventario de objetos.

Introducción y objetivo

Las colecciones acotadas de objetos de distinto tipo son muy frecuentes en todo tipo de programas y sistemas. Una aplicación de estas colecciones son las mochilas y objetos equipados de los personajes de algunos videojuegos de tipo RPG o similares. Básicamente, en estos juegos, el jugador recoge objetos por el mundo virtual y los almacena en su mochila o los *equipa*, es decir, se los pone para poderlos usar a través de la pulsación de una letra del teclado o de un botón. Desde el punto de vista de programación, una solución para implementar esta funcionalidad consiste en guardar los objetos en dos colecciones acotadas de objetos: una será la mochila y la otra el conjunto de objetos equipados.

El objetivo de esta práctica es que el alumno se familiarice con la manipulación de estas estructuras de datos, para ello deberá implementar las operaciones necesarias para poder manipular los objetos del inventario de acuerdo con las reglas que se describen en este enunciado. Este objetivo se considera cumplido si las clases desarrolladas por el alumno, junto con las adjuntas con este enunciado, logran mostrar el inventario del personaje, con su mochila y objetos equipados y todo funciona de acuerdo a lo previsto.

No es necesario escribir el código fuente en un orden determinado, el alumno puede escribir un pequeño esquema de las clases e ir completando cada una de las funcionalidad en el orden que considere más adecuado, aunque es conveniente escribirlas en el orden en que aparecen en este enunciado.

1. Descripción

El personaje de nuestro particular módulo de videojuego se maneja a través de una interfaz gráfica que debe compilar y arrancar sin problemas antes de que el alumno escriba nada de código. Al arrancar debe observarse que el personaje tiene un nombre (pendiente de rellenar), un retrato (en azul) y 14 huecos entre mochila (*backpack*), 10 huecos, y equipados (*equipped*) otros 4. En la parte inferior hay una zona con fondo negro donde se mostrarán mensajes de texto.

Tanto el retrato como cada hueco de objeto tienen un botón que se puede pulsar para hacer una acción, en la implementación proporcionada al pulsar los botones se puede ver un mensaje de texto con la información del hueco que se ha pulsado. Además, en cada hueco se muestra un texto que marca una zona de *"drag and drop"*, es decir, pinchar y arrastrar, al pinchar en un hueco y, sin saltar, arrastrar a otra zona se obtiene otro mensaje diferente con la información de los huecos de origen y destino.

1.1. Estructura del código

El código fuente está dividido en 2 paquetes: `model` y `gui` y una carpeta adicional: *assets* con iconos.

La mayor parte del trabajo del alumno debe hacerse en el paquete `model`. En este paquete deben añadirse las clases que el alumno decida implementar para representar los objetos del inventario, las colecciones que guardan esos objetos y, en la clase `Player`, implementar las operaciones que aparecen definidas mediante los métodos marcados con TODO y que se llaman desde la interfaz gráfica.

El paquete `gui` contiene las clases que generan la interfaz gráfica de usuario, en general el alumno no necesita tocar o leer estas clases.

1.2. Mensajes en la interfaz gráfica

Como no tenemos el resto del videojuego, para simular que realizamos las acciones asociadas a los objetos vamos a utilizar la zona de mensajes de la interfaz gráfica de usuario. En el paquete `gui` tenemos una clase `Report` que aparece gráficamente como una zona con fondo negro donde se pueden mostrar mensajes escritos desde cualquier clase de las que implemente el alumno, para ello será necesario usar el método de clase:

```
public static void print(String msg);
```

de dicha clase `gui.Report`, por ejemplo, la sentencia:

```
gui.Report.print("Begining program");
```

mostrará *Begining program* en la zona de mensajes. Cada mensaje está precedido por un índice creciente que se genera automáticamente y que sirve para comprobar que cada mensaje se ha ido generando en el orden esperado. Esta zona solo guardará los últimos

> En algunos apartados de este enunciado se dirá que se "*debe mostrar un mensaje*", esta es la forma de hacerlo.

2. Se pide

Para que la interfaz gráfica de usuario, *GUI*, muestre correctamente lo que se espera se tienen que tener en cuenta los siguientes puntos:

1. Se debe mostrar un nombre en la parte superior de la GUI.
2. Al pulsar el icono con el retrato se debe alternar entre los distintos aspectos del personaje. Estos aspectos vienen dados por los valores de `enum PlayerAspect`.
   - Cuando se llame al método para cambiar entre aspectos, se debe pasar al aspecto siguiente, cuando llega al último se debe volver a seleccionar el primero.
3. Se recomienda implementar una clase `Item` para representar los objetos de inventario y un constructor de la clase `Player` para generar un inventario inicial, el alumno es libre de elegir cómo quiere hacerlo. Este inventario inicial no se puede alterar, es decir, no están previstas ni son necesarias las operaciones que permite recoger nuevos *items* o soltar los que se tienen.
4. Se supone que el personaje solo puede tener un máximo de 10 objetos en cualquier momento. Estos 10 objetos se pueden guardar entre la mochila (máximo 10) y equipados (máximo 4).
5. Se puede *activar* un objeto entre los equipados, el objeto a activar se indica mediante un índice del 0 al 3 (inclusive) que se corresponde con los 4 huecos disponibles para equipar objetos. Si no existe el objeto correspondiente la activación no debe hacer nada, pero se debe mostrar el mensaje: *"No hay un objeto para activar"* o un mensaje similar. Al activar un objeto se debe mostrar un mensaje característico de ese objeto. Por ejemplo: "Usando llave".
6. Se puede mostrar una descripción de un objeto cuando está en la mochila y se pulsa el botón correspondiente. Por ejemplo: "Una llave herrumbrosa". Si no hay un objeto en ese hueco se debe mostrar un mensaje, por ejemplo: "Ese hueco está libre".
7. Se puede *mover* un objeto equipado a la mochila o viceversa. Al hacerlo se libera el hueco de origen y se ocupa el hueco destino. Si el hueco de origen está vacío debe mostrarse un mensaje y no hacer nada. Si el hueco de destino está en la mochila, pero está ocupado debe buscarse otro hueco y colocar el objeto en ese hueco. Si el hueco de destino está en los objetos equipados debe mostrarse un mensaje que diga que ya hay un objeto equipo y no hacer nada más.

3. Funcionalidades adicionales

Este programa se puede completar con algunas funcionalidades, se invita al alumno a que piense cómo y a intentarlo o bien que elija implementar algunas de las que se sugieren a continuación:

- Poner en los huecos texto (fácil) o iconos (más difícil) relacionados con los objetos que se guarden.

  Notas:

  - La GUI ya tiene previsto que se puede cambiar el texto de los botones mediante el método `gui.CharacterWrappe`
  - Se puede modificar libremente la clase CharacterWrapper para lograr esta funcionalidad. En la propia gui se manejan iconos se puede consultar el código fuente para ver cómo se hace.

- Hacer que algunos objetos puedan estar activados o desactivados. Para reflejar esto se puede usar el texto del ítem (fácil) o su color de fondo (más difícil). Se puede usar el método `gui.CharacterWrapper.getColor`. Pero es necesario completar también el método `gui.ItemPanel.setColor` para decidir cuál es el color que se modifica.

- Hacer que las activaciones tengan "un tiempo de enfriamiento". La gui tiene activado un temporizador que, básicamente, llama periódicamente al método Player.tick. El alumno puede utilizar el tiempo transcurrido para deshabilitar el uso del objeto durante un cierto intervalo de tiempo.

- [...]

4. Evaluación y calificación

La forma de evaluar este trabajo va a reflejar que los apartados obligatorios tienen cierta dificultad, pero siguen siendo obligatorios. Por ello, no se podrá obtener un 5 como nota final del ejercicio sino que pasaremos directamente del 4 al 6.

- Las soluciones que realicen correctamente los puntos 1 y 2 obtendrán una calificación de 2 puntos.

- Las soluciones que realicen correctamente los puntos 3, 4 y 5 obtendrán una calificación de 4 puntos.

- Las soluciones que cumplan todos los puntos hasta el 7 obtendrán 6 puntos.

- Las notas del 7 en adelante se obtendrán completando funcionalidades adicionales y por "buen" estilo en la implementación de los puntos obligatorios.