Segunda Práctica
Implementación del diseño de una estructura de clases

Competencias específicas de la segunda práctica

- Interpretar correctamente diagramas de clases del diseño en UML.
- Implementar el esqueleto de clases (cabecera de la clase, declaración de variables y declaración de métodos) en Java y en Ruby, y relacionarlas adecuadamente a nivel de implementación.
- Implementar algunos de los métodos simples de las clases.

Programación de la segunda práctica

Tiempo requerido: Dos sesiones (cuatro horas).

Planificación y objetivos:

<table>
<tr><th>Sesión</th><th>Semana</th><th>Objetivos</th></tr>
<tr><td>Primera</td><td>Del 27 al 31 de octubre</td><td rowspan="2">• Ubicar las clases y relaciones implementadas en la primera práctica dentro del diagrama de clases UML dado.
• Identificar nuevas clases.
• Declarar las clases nuevas.
• Declarar las variables básicas de cada clase.
• Declarar las variables de referencia (implementan asociaciones entre clases).
• Declarar los métodos constructores, consultores y modificadores de cada clase.
• Declarar otros métodos que aparezcan en el diagrama de clases UML.
• Implementar todos los métodos constructores.
• Conocer y utilizar el patrón de diseño Singleton.
• Implementar métodos con funcionalidad simple.</td></tr>
<tr><td>Segunda</td><td>Del 3 al 7 de noviembre</td></tr>
<tr><td colspan="3">* El trabajo se realizará primero en Java y una vez depurado se trasladará a Ruby.
* La práctica se desarrollará en grupo, con la misma composición que para la primera práctica.</td></tr>
</table>

Entrega y **examen** de la segunda práctica en la semana del 10 al 14 de noviembre.

- **Lugar:** Aula en la que de desarrolla la correspondiente sesión de prácticas.
- **Día:** El correspondiente a la sesión de cada uno de la semana indicada.
- **Duración**: 20 minutos al comienzo de la sesión.
- **Tipo examen:** Realizar pequeñas modificaciones sobre el propio código.
- Se pedirá entregar el código de la segunda práctica con las modificaciones realizadas.

A) Descripción del problema y dinámica del juego

Se desea implementar un juego de rol llamado Napakalaki, el cual se basa en el juego *Munchkin, diseñado por Steve Jackson, bajo el lema* "*Mata a los monstruos, roba el tesoro, apuñala a tus amigos"*. El lema de Napakalaki es "*Trabaja en pareja, gana puntos y aprende Ruby".*

En Napakalaki, al igual que en *Munchkin*, cada jugador comienza siendo un humano de nivel 1 y el reto consiste en alcanzar el nivel 10 antes que los demás. Para ello, cada jugador debe enfrentarse a monstruos y superar sus terribles maldiciones. Como ayuda, contará con armas y armaduras (sus tesoros).

En nuestro juego podrán participar 3 jugadores, identificados por un nombre único. Como ya hemos dicho, su nivel inicial es 1 y gana la partida si llega a 10.

El juego contiene dos barajas de cartas: tesoros y monstruos. Los Monstruos ya los conoces. En cuanto a los Tesoros, a continuación podrás ver cómo ayudarán a luchar contra los monstruos.

Tesoros

Los tesoros poseen un nombre, valen una cierta cantidad de piezas de oro, y suponen al jugador que los tiene poder enfrentarse a un monstruo con un cierto número de niveles adicionales a los que él tiene como jugador (lo que se denomina un bonus). Hay varios tipos de tesoros: de mano (una o dos manos), casco, calzado, armadura y collar.

Puede haber tesoros sin bonus, cuya utilidad es simplemente la de su valor en piezas de oro. Los tesoros que tienen 2 bonus distintos (como en el ejemplo de la Figura 1), suman el valor más bajo si el jugador no dispone del tesoro "collar" y suman el nivel más alto si se dispone del tesoro "collar".

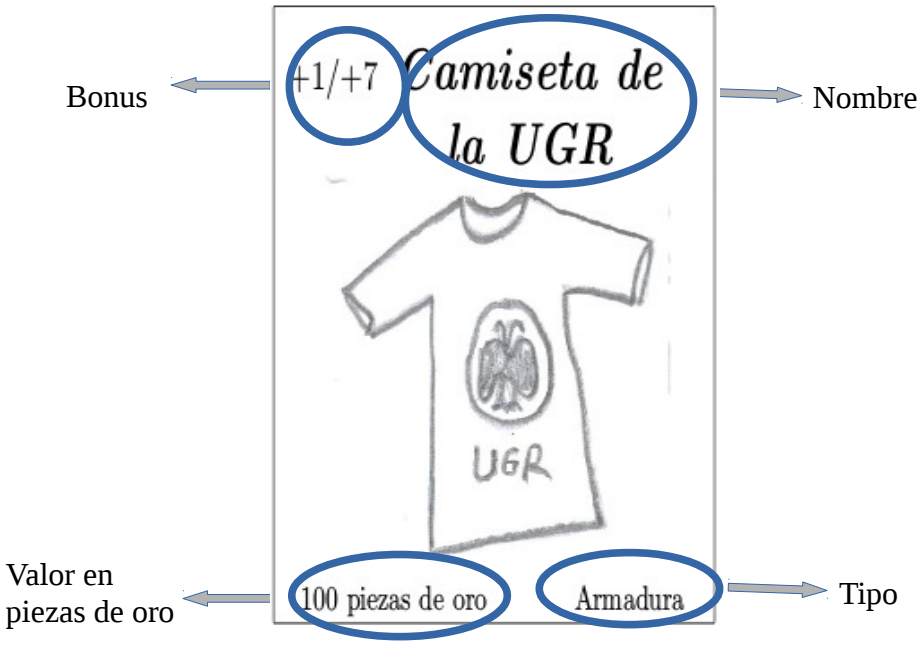

Figura 1. Ejemplo de carta tesoro

Por ejemplo, la carta de la Figura 1 representa un tesoro de tipo armadura con nombre "Camiseta de la UGR" y cuyo valor son 100 piezas de oro. El jugador que la posea, podrá luchar con un nivel más que el suyo si no tiene collar, y con siete niveles más si posee una carta tesoro de tipo collar.

A las cartas de tesoro de las que dispone un jugador para combatir se les denomina cartas "equipadas" o "visibles" y se muestran a todos los jugadores. Cada jugador puede *equiparse* con un único tesoro de cada tipo (salvo los tesoros que sólo ocupan una mano, en cuyo caso puede tener dos). Así, cada jugador podría equiparse en el mejor de los casos con una armadura, un tesoro de dos manos, un casco, un calzado y un collar; o con una armadura, dos tesoros de una mano, un casco, un calzado y un collar.

Además de los tesoros equipados, cada jugador podrá tener hasta cuatro tesoros "ocultos", que como es lógico los demás jugadores no pueden ver. En el combate contra un monstruo, al nivel del jugador se sumará el bonus de todos sus tesoros equipados, no así el de sus tesoros ocultos. Al final de cada turno, el jugador podrá equiparse de nuevos tesoros (haciendo visible una o varias cartas que tenía ocultas en la mano), respetando las restricciones comentadas anteriormente.

Si un jugador se ha equipado con el collar, el resto de tesoros actúan con el bonus más alto. Pero tras un combate contra un monstruo en el que ha intervenido el collar y el jugador ha ganado, éste debe descartarse obligatoriamente del collar. No lo puede volver a usar hasta que le vuelva a salir al robar un tesoro del mazo.

Juego

El jugador que comienza el juego se decide al azar, y después se sigue el orden de las agujas del reloj. En el primer turno que tiene cada jugador, lanza el dado para saber con cuantos tesoros comienza: si obtiene un 1 en el dado, comienza con 1 tesoro, si obtiene entre un 2 y un 5 en el dado, comienza con 2 tesoros, y si obtiene un 6, comienza con 3 tesoros.

En su turno, el jugador debe sacar una carta del mazo de monstruos y combatir contra el monstruo que salga. Para el combate, el jugador tendrá un nivel total calculado como sigue: nivel total del jugador = nivel del jugador + bonus de todos los tesoros que tenga equipados (teniendo en cuenta que si tiene el collar equipado, todos los tesoros suman el bonus más alto).

A continuación se detallan los distintos escenarios posibles durante el combate:

<table>
<tr><td rowspan="6">ESCENARIOS</td><td rowspan="2">Si el monstruo tiene un nivel inferior al nivel total del jugador</td><td rowspan="2">El jugador gana y se aplica el “buen rollo” definido por el monstruo</td><td>Sube los niveles indicados (si no se dice nada es uno)</td><td colspan="3" rowspan="2">Si tiene equipado el collar debe descartarse de él</td></tr>
<tr><td>Roba el número de tesoros indicados</td></tr>
<tr><td rowspan="4">Si el monstruo tiene un nivel mayor o igual al nivel total del jugador</td><td rowspan="4">El jugador pierde pero tiene la posibilidad de intentar huir tirando un dado</td><td>Si el resultado es 5 o 6</td><td colspan="3">El jugador huye y se queda tal y como estaba</td></tr>
<tr><td rowspan="3">Si el resultado es menor que 5</td><td rowspan="3">El jugador se aplica el “mal rollo” indicado por el monstruo</td><td colspan="2">Si el mal rollo es “muerte”, el jugador pierde todos sus tesoros (visibles y ocultos) y vuelve a quedar con nivel 1</td></tr>
<tr><td rowspan="2">Si el mal rollo no es “muerte”</td><td>Baja los niveles indicados</td></tr>
<tr><td>Se descarta del número tesoros indicados (ocultos y/o visibles) o del tipo de tesoros indicados (ocultos y/o visibles)</td></tr>
</table>

El nivel mínimo de un jugador es 1, es decir, que si un jugador tiene nivel 2 y sucumbe ante un monstruo cuyo mal rollo es bajar 3 niveles, se queda con nivel 1 (nunca se puede quedar en negativo o cero).

Cuando un jugador se queda en situación de muerte, lanzará el dado para equiparse con algún tesoro antes del siguiente combate. Se equipará con 1, 2 ó 3 tesoros según lo que obtenga en el dado, aplicando la misma correspondencia que al principio de la partida.

Los tesoros recibidos por el jugador siempre pasarán a formar parte de su lista de tesoros ocultos. Después del combate, el jugador puede equiparse las cartas que considere oportuno. En caso de quedarse con más de 4 cartas ocultas, deberá descartar las que juzgue que le serán menos útiles hasta quedarse con un máximo de 4 cartas ocultas.

Las cartas visibles no pueden pasar nunca a ser ocultas. Por ejemplo, un jugador no

puede cambiar un casco que se haya equipado por otro con un bonus mayor que tenga oculto. Sin embargo, sí podrán ser susceptibles de ser elegidas como descartes cuando el mal rollo del monstruo no indique explícitamente qué tipo de tesoros se pierden.

Finalmente, es posible comprar un nivel con 1.000 piezas de oro. La compra se realizará al principio del turno del jugador, antes de sacar el monstruo con el que debe combatir. Para ello, el jugador deberá descartarse de tantos tesoros como sea necesario para sumar dicha cantidad. Es posible comprar tantos niveles como se crea conveniente (1 nivel – 1000 piezas de oro, 2 niveles – 2000 piezas de oro), y es posible usar tesoros tanto visibles como ocultos. El juego no devuelve el cambio, si un jugador emplea tesoros por valor de 1900 piezas de oro para comprar niveles, el sistema sólo le dará un nivel y perderá todos los tesoros empleados.

Tras el combate, el monstruo en juego se descarta y el jugador podrá equiparse algunos de sus tesoros ocultos, respetando las restricciones comentadas anteriormente.

B) Tareas a realizar en la primera sesión

B.1) Implementa en Java el diagrama de clases del diseño, proporcionado en swad como DClasesNapakalaki.pdf, siguiendo el siguiente esquema:

1) Declara todas las variables básicas, teniendo en cuenta, además de su tipo primitivo clase, su alcance (si son private, package, protected o public), y su ámbito (si son de instancia o de clase (static)).

2) Declara todos los métodos, teniendo en cuenta, parámetros, valor de retorno, ámbito de acceso y alcance.

3) Identifica las variables de referencia a partir de las asociaciones existentes entre las clases. Estudia cuál sería la colección de objetos más adecuada para la implementación de estos atributos de referencia (HashMap, ArrayList, o LinkedList), y úsala en la declaración de la variable.

4) Implementa los constructores de todas las clases, para que los objetos queden correctamente inicializados.

5) Declara cada una de las nuevas clases que aparecen en el diagrama, teniendo en cuenta cuales de ellas son **<<singleton>> o <<enumeration>>** .

   Se dice que una clase es un singleton cuando sólo puede tener una instancia. Para conseguir esto utilizamos el código que nos indica el patrón de diseño singleton, que es el siguiente (suponiendo que Miclase es la clase singleton):

```
public class MiClase {

   private static final MiClase instance = new MiClase();

  // El constructor privado asegura que no se puede instanciar
  // desde otras clases
  private MiClase() { }
```

```
  public static MiClase getInstance() {
     return instance;
  }
}
```

Más información: http://en.wikipedia.org/wiki/Singleton_pattern

B.2) Siguiendo lo dispuesto en la tarea B.1 implementa en Ruby el diagrama de clases del diseño. Ten en cuenta que en Ruby:

- No hay declaración explícita de tipos,
- Las variables y métodos solo admiten tres controles de acceso: public, protected y private,
- Las variables de instancia se declaran con @ y las variables de clase con @@,
- Los métodos de clase se declaran usando el nombre de la clase o self para indicar que solo ella puede ejecutar el método,

```
class Ejemplo
def Ejemplo.métodoDeClase
o
def self.métodoDeClase
```

- Para implementar el patrón Singleton en Ruby se puede usar el modulo Singleton,

```
include Singleton
```

- Para trabajar con colecciones puedes usar Array o Hash.