Primera Práctica
Lenguajes de Programación Orientada a Objetos: Java y Ruby

Competencias específicas de la segunda práctica

- Trabajar con entornos de desarrollo.
- Tomar contacto con los lenguajes OO Java y Ruby.
- Desarrollar pequeños ejemplos siguiendo el paradigma OO.
- Interpretar los resultados obtenidos tras ejecutar un determinado código.

Programación de la primera práctica

Tiempo requerido: Tres sesiones (seis horas).

Planificación y objetivos:

Comienzo: **6 de octubre**

Sesión	Semana	Objetivos
Primera (Java)	6-10 al 10-10	• Familiarizarse con el entorno de desarrollo. • Conocer las características del lenguaje de programación Java. En especial que es interpretado, que todas las variables son referencias a objetos excepto los tipos primitivos y que posee recolección automática de basura. • Conocer el contenido básico de una clase.
Segunda (Java)	13-10 al 17-10	• Saber cómo implementar las relaciones entre objetos. • Familiarizarse con los aspectos básicos de las colecciones de objetos y con algunas de las clases que los manejan en Java. • Familiarizarse con el concepto de excepción en Java. • Conocer el ámbito de las variables y métodos.
Tercera (Ruby)	20-10 al 24-10	• Conocer las características del lenguaje de programación Ruby • Familiarizarse con las clases que manejan las colecciones de objetos en Ruby. • Conocer el ámbito de las variables y métodos. • En general, conocer los aspectos mas relevantes de la Orientación a Objetos en Ruby.
La práctica se desarrollará en grupo de 2 componentes.		

Entrega (finalizada las tres sesiones):

- **Fecha**: Semana del 27 al 31 de octubre de 2013. Cada uno en su sesión.
- **Entregables**: Un archivo zip denominado P1.zip, con el siguiente contenido:
  - Una carpeta denominada P1Java el proyecto desarrollado, exporta el proyecto desde Netbeans a un zip, llamado P1Java..
  - Otra carpeta llamada P1Ruby, con todos los ficheros .rb o el proyecto netbeans generado, en este caso hacerlo de la misma forma que se hizo con Java.
- **Forma de entrega:** Sólo un miembro del grupo sube a SWAD en archivo P1.zip (pestaña "Evaluación", enlace "Mis trabajos", zona de "Actividades"), antes de que termine el plazo indicado por el profesor para cada grupo.

Evaluación

Se realizará un control sobre el contenido de la práctica el día de la entrega. La evaluación se realizará teniendo en cuenta tanto la entrega como el resultado del control, tal como se indica en la guía de la asignatura.

Enlaces interesantes

http://groups.diigo.com/group/pdoo_ugr/content/tag/Java

https://groups.diigo.com/group/pdoo_ugr/content/tag/ruby

Primera sesión (Java)

1) Descripción del problema: El ejemplo que se va a desarrollar es una baraja de cartas de monstruos, esta baraja va a servir para desarrollar, a partir de la segunda práctica, un juego en el que los jugadores tendrán que enfrentarse a los monstruos para conseguir tesoros y niveles de poder para ganar la partida. Una carta de **Monstruo** contiene lo siguiente (figura 2):

Un **nombre** que lo identifica y un cierto **nivel de combate**, además indica cuántos niveles y/o tesoros gana el jugador que le venza (“**buen rollo**”). Sin embargo, también indica un “**mal rollo**”, que determina cuántos niveles y/o tesoros (ocultos y/o visibles y de qué tipo) pierde el jugador que sucumba ante él. El mal rollo también puede ser “**muerte**”, en este caso el jugador pierde todos sus tesoros (visibles y ocultos) y vuelve a quedar con nivel 1.

Por defecto, al vencer a un monstruo se sube un nivel, a no ser que el monstruo indique explícitamente que se suben más, como en el ejemplo mostrado en la figura 2.

En el ejemplo de la figura 2, se muestra un monstruo de nombre “el rey de rosa”, cuyo nivel es 13. Si un jugador logra vencerle, entonces subirá 2 niveles y podrá robar 4 tesoros. Si por el contrario el jugador perdiese, perdería 5 niveles.

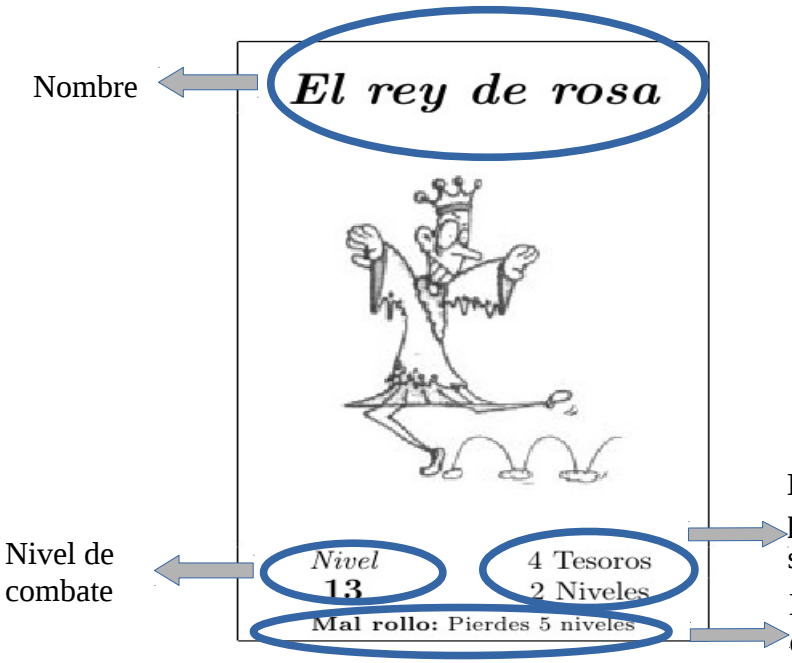

Figura 2. Ejemplo de carta monstruo

2) Tareas para hacer

Recomendación: Planteate todo lo que vas haciendo, si algo no se entiende pregúntale al profesor.

A) Entorno de desarrollo: Instala y/o ejecuta NetBeans

1. Si trabajas con los ordenadores del aula, ejecuta NetBeans (desde Ubuntu 14.04).
2. Si trabajas con portátil propio, instala Java y NetBeans desde: http://www.oracle.com/technetwork/java/javase/downloads/jdk-netbeans-jsp-142931.html
3. Una vez instalado el entorno, ejecútalo.

B) **Crea un proyecto** de tipo *aplicación Java*. Para ello, una vez abierto NetBeans, crea un nuevo proyecto denominado **Napakalaki** en tu directorio que consideres apropiado para guardarlo, indicando que se desea crear un clase que incluya el método *main()* llamada **napakalaki.PruebaNapakalaki**.
Además de crearse el proyecto también se ha creado un paquete (elemento que aglutina clases) denominado *napakalaki* y una clase, que incluye el método *main(),* denominada *PruebaNapakalaki*.

C) En el paquete napakalaki **crea una clase** denominada *Prize.* Para ello, pulsa el botón derecho del ratón sobre el paquete y selecciona *Nuevo / Clase Java*. Para esa clase define:

1) Los siguientes atributos: *treasures* y *level,* los dos de tipo *int* y visibilidad privada.
2) El constructor de los objetos de esa clase: *Prize(int treasures, int level),* su funcionalidad es construir objetos de la clase y darle valor a los atributos *definidos.*
3) *L*os consultores básicos de los atributos definidos anteriormente: getTreasures() y getLevel(). Su funcionalidad es devolver el valor de dichas variables.

D) **Crea un enumerado** denominado T*reasureKind.* Para ello, pulsa botón derecho del ratón sobre el paquete y selecciona *Nuevo / Java Enum* y define los siguientes valores *{armor, oneHand, bothHand, helmet, shoe, necklace}* (investiga cómo definir un enumerado y sus valores en Java).

E) Crea una clase denominada *BadConsequence (*igual que en el punto anterior). Para esta clase define:

1. Los siguientes atributos con visibilidad privada:
   - *text,* de tipo *String*, para representar lo que nos dice un mal rollo
   - *levels*, de tipo *int*, para representar los niveles que se pierden.
   - *nVisibleTreasures*, de tipo *int*, para representar el numero de tesoros visibles que se pierden.
   - *nHiddenTreasures*, de tipo *int*, para representar el número de tesoros ocultos que se pierden.
   - death, de tipo boolean, para representar un mal rollo de tipo muerte.
2. Los siguientes constructores con visibilidad pública:
   - *BadConsequence(String text, int levels, int nVisible, int nHidden)*
   - *BadConsequence(String text, boolean death)*
3. *L*os consultores básicos de los atributos definidos anteriormente, *getTtext()*, *getLevels()*... con visibilidad de pública. Su funcionalidad es devolver el valor de dichas variables.

F) Crea la clase *Monster*

1. Los siguientes atributos con visibilidad privada
   - *name*, de tipo *String*, para representar el nombre del monstruo.
   - *combatLevel*, de tipo *int*, para representar el nivel de combate del monstruo
2. *L*os consultores básicos de los atributos definidos anteriormente, cuya funcionalidad es devolver el valor de dichas variables.

G) En la clase *BadConsequence* define dos atributos de tipo lista de *TreasureKind*, de la siguiente forma:

- *private ArrayList<TreasureKind> specificHiddenTreasures = new ArrayList();*
- Igual para *specificVisibleTreasures*

H) En la clase *BadConsequence* define un nuevo constructor para dar valor a estos nuevos atributos:

- *BadConsequence(String text, int levels, ArrayList<TreasureKind> tVisible, ArrayList<TreasureKind> tHidden)*

I) En clase *Monster* define un atributo de tipo *Prize*, otro de tipo *BadConsequence* y el constructor de la clase:

- *Monster(String name, int level, BadConsequence bc, Prize price)*

J) Define el método toString() en las tres clases definidas (*Monster, BadConsequence y Prize*), este método nos devuelve un String con el estado del objeto correspondiente, por ejemplo en la clase Prize, el método toString() tedrá la siguiente implementación:

```
public String toString(){
    return “Treasures = “ + Integer.toString(treasures) + “ levels = ” + Integer.toString(level)
}
```

K) Crea un pequeño main para probar lo que has realizado, creando objetos y consultando su estado con los correspondientes métodos toString() que has implementado.

Apéndice: Plantilla para la definición de una clase en Java.

```
//Cabecera de la clase
[public] [abstract|final] class nombreDeLaClase [extends NomSuperClase]
                                                 [implements NombreInterface, ...]

//Cuerpo de la clase
{
    //Definición de variables que almacenan el estado de objetos los de esta clase
      [publi|protected|private] [static] [final] tipo nomVar [=inicialización];
        ...

    //Definición de los métodos que definan la funcionalidad de los objetos

    //Cabecera del método
    [publi|protected|private] [static] [abstract|final] returnType
            nombreMetodo ([listParametros]) [throws listaExcepciones]

   //Cuerpo del método
    {//Código java que implemente el método}
        ...
}
```