Segunda Práctica (P2)

Implementación del diseño de una estructura de clases

Competencias específicas de la segunda práctica

- Interpretar diagramas de clases del diseño en UML.
- Implementar el esqueleto de clases (cabecera de la clase, declaración de atributos 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.

A) Programación y objetivos

Tiempo requerido: Dos sesiones, S1 y S2 (cuatro horas).

Comienzo: **semana del 9 de octubre excepto los grupos A1, C2, D1 que comenzarán la siguiente semana (19 de octubre).**

Planificación y objetivos:

Sesión	Semana	Objetivos
S1	9-13 octubre o 19 de octubre	• Ubicar las clases y asociaciones implementadas en la primera práctica dentro del diagrama de clases UML dado. • Identificar y definir las nuevas clases. • Declarar los atributos básicos de cada clase en Java. • Declarar los atributos de referencia (implementan asociaciones entre clases) de cada clase en Java. • Declarar e implementar los métodos constructores y consultores de cada clase en Java. • Declarar en Java otros métodos que aparezcan en el diagrama de clases UML. • Conocer y utilizar el patrón de diseño Singleton.
S2	16-20 de octubre o 26 de octubre	• Mismos objetivos anteriores en lenguaje Ruby
La práctica se desarrollará tanto en Java como en Ruby en equipos de 2 componentes. El examen es individual.		

Evaluación:

El examen de las prácticas 1 y 2 será en la semana del 23 al 27 de octubre (cada grupo en su sesión) para todos los grupos excepto para los grupos A1, C2 y D1 que lo tendrán el 2 de noviembre.

El examen lo realizará **cada grupo en su hora** y aula de prácticas y se resolverá sobre el código que se haya desarrollado para las prácticas 1 y 2 **en el ordenador del aula**. Para ello, cada estudiante deberá acudir a clase con su proyectos Java y Ruby en un pen drive o haberlos dejado

en su cuenta de usuario de los ordenadores del aula.

No habrá entrega del código previa al examen, sino que se pedirá que cada estudiante suba a una tarea creada en PRADO a tal efecto los proyectos Java y Ruby con los cambios solicitados durante el examen. .

Enlaces interesantes

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

SESIÓN 1

B) S1. Tareas en Java

1) **Implementa el diagrama de clases *DCQytetet*** proporcionado en el fichero DiagramaClases.pdf, siguiendo las siguientes directrices para cada clase:

   a) Incluye todas las clases en el paquete ModeloQytetet

   b) Declara todos los atributos básicos teniendo en cuenta, además de su tipo primitivo, su visibilidad (si son *private*, *package*, *protected* o *public*), y su ámbito (si son de instancia o de clase - *static* -).

   c) Declara todos los métodos teniendo en cuenta: parámetros, valor de retorno, visibilidad y ámbito de acceso. Evita errores de ejecución en los métodos que devuelven algún valor, comentando el método o bien lanzando una excepción con el siguiente código:

```
throw new UnsupportedOperationException("Sin implementar");
```

   d) Identifica los atributos de referencia a partir de las asociaciones existentes entre las clases y decláralos.

   e) Implementa los constructores de todas las clases prestando atención a que los objetos queden correctamente inicializados con valores en todos sus atributos.

   f) Implementa consultores básicos para todos los atributos, así como el método *toString* para todas las clases.

   g) Ten en cuenta en la implementación si aparece en el diagrama la palabra ***<<singleton>>*** o ***<<enumeration>>*** .

   h) Se dice que una clase es un *singleton* cuando sólo puede tener una instancia. Para conseguirlo utilizamos el patrón de diseño *singleton*. Una de las formas de implementarlo es la 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;
}
```

```
}
```

Cuando necesitemos acceder a la instancia de `MiClase:`

```
MiClase mc= MiClase.getInstance();
```

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

Por último, revisa las clases *Sorpresa*, *TipoSorpresa*, *Tablero*, *Casilla*, *TipoCasilla* y *TituloPropiedad* desarrolladas en la práctica anterior, y asegúrate de que concuerdan con lo que se indica en el diagrama de clases proporcionado.

2) **Pruébalo todo:** utiliza el método *main* de la clase *PruebaQytetet* para crear y mostrar *jugadores, tablero, cartas sorpresa* y a la única instancia de la clase *Qytetet*.

3) Realiza en Java los **ejercicios de autoevaluación** que se proponen al final del guión.

SESIÓN 2

C) S2. Tareas en Ruby

1) Sigue lo dispuesto para Java implementando el diagrama de clases, teniendo en cuenta que en Ruby:

- Todo lo realizado en Ruby deberá seguir formando parte del **módulo *ModeloQytetet***
- No hay declaración explícita de tipos ni hay tipos primitivos.
- Los atributos son privados en Ruby.
- Los atributos de instancia se pueden usar en cualquier método de instancia anteponiendo a su nombre @. Lo recomendable es crearlos e inicializarlos dentro del método *initialize*.
- Los atributos de clase se pueden crear en cualquier lugar del código asociado a la clase y anteponiendo a su nombre @@.
- Los métodos de instancia de la clase se declaran usando el nombre de la clase o *self* para indicar que sólo ella puede ejecutar el método,

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

- El equivalente del método *toString* en Ruby es *to_s*.
- El patrón *Singleton* en Ruby ya está implementado en el módulo *Singleton* y lo único que tenemos que hacer es usarlo, por ejemplo, si la clase *MiClase* es un singleton tenemos que añadir:
  - Al fichero donde está definida la clase singleton:

```
require "singleton"
```

- En *MiClase* incluir el módulo *Singleton* de la siguiente forma:
  ```
  include Singleton
  ```

- Donde necesitemos acceder a la instancia de *MiClase*:
  ```
  mc = MiClase.instance
  ```

- Para trabajar con colecciones puedes usar la clase *Array*.

2) **Pruébalo todo**: utiliza el método *main* de la clase *PruebaQytetet* para crear y mostrar *jugadores, tablero, cartas sorpresa* y a la única instancia de la clase *Qytetet*.

3) Realiza en Ruby los **ejercicios de autoevaluación** que se proponen a continuación.

Ejercicios de autoevaluación

1. Haz una copia de los proyectos antes de introducir los cambios que se proponen, ya que la práctica siguiente partirá de lo realizado hasta ahora. Trabaja sobre estas copias.
2. Realiza los cambios que consideres oportunos en el código para incluir las siguientes supuestas modificaciones en el diagrama de clases:
   a) La clase Dado pasa a ser *singleton*.
   b) El número mínimo de jugadores será 3.
   c) Se añade "*Escapandose*" como otra forma de salir de la cárcel.
   d) La composición de Tablero a Casilla pasa a ser simplemente una agregación.
   e) Se desea añadir al juego las fichas, de tal forma que cada jugador tiene asociada una ficha para jugar. Las fichas tienen una forma y un color, y estarán colocadas en una casilla en cada momento. En una casilla puede haber varias fichas. Modifica el diagrama de clases y el código para incluir este nuevo elemento y sus relaciones con los demás.

NOTA: Asegúrate de que todos los métodos resaltados en color rojo en el diagrama de clases están implementados (en Java y en Ruby) al finalizar la práctica.