Programación Modular y Orientación a Objetos - Laboratorio 3

Requisitos previos

El alumnado se desenvuelve en el entorno Eclipse y maneja adecuadamente la perspectiva Java; es capaz de crear y representar (mediante diagramas UML) clases sencillas y relaciones simples entre clases; y sabe realizar casos de prueba con el framework JUnit.

Objetivos

Este laboratorio se centrará en afianzar los conocimientos adquiridos en los laboratorios anteriores, así como en profundizar en la visión de la POO frente a la perspectiva de la programación imperativa. Además, se presentará el concepto de interfaz en Java.

Al finalizar este laboratorio, el alumnado deberá ser capaz de:

- Desenvolverse en el uso del entorno Eclipse para la realización de ejercicios sencillos, que se irán complicando en sucesivos laboratorios, desde una perspectiva orientada a objetos.
- Comprender el concepto de interfaz, y ser capaz de crear una clase que implemente una interfaz dada.
- Comprender los diagramas de clase UML, y utilizar JUnit para crear pruebas unitarias.

Motivación

Antes de comenzar a trabajar con elementos más avanzados, como los TAD y las MAE, o los aspectos relativos a la herencia (que de momento solamente se han dejado entrever), es conveniente afianzar todo lo visto hasta ahora. Este laboratorio consiste en la realización de un par de ejercicios sencillos que el alumnado debería poder completar sin mayores problemas.

Tarea 1: la interfaz IFecha

En este primer ejercicio se pide desarrollar la clase Fecha, que implementa la siguiente interfaz IFecha, así como la correspondiente clase FechaTest que contendrá **todos los casos de prueba necesarios** para comprobar la corrección de los métodos de la clase Fecha.

```
package org.pmoo.packlaboratorio3;
public interface IFecha
{
	public abstract void incrementar();
	public abstract void decrementar();
}
```

Figura 1 - Interfaz IFecha que se pide implementar.

El método *incrementar()* suma un día al objeto (this) que lo ejecuta, y el método decrementar() resta un día. En ambos casos, la fecha resultante debe ser la fecha válida correspondiente.

Para este ejercicio se considerará que una fecha es válida cuando el año es un número positivo (es decir, >0), el mes es un número comprendido entre 1 y 12, y el día es un número comprendido entre 1 y...

- 31 si el mes es 1, 3, 5, 7, 8, 10 ó 12.
- 30 si el mes es 4, 6, 9 u 11.
- 29 si el mes es 2 y el año es bisiesto.
- 28 si el mes es 2 y el año no es bisiesto.

Un año es bisiesto cuando es múltiplo de 4 pero no de 100. Los múltiplos de 400 son una excepción, ya que aun siendo múltiplos de 100 son bisiestos.

La plantilla *Fecha.java* que se facilita junto a este enunciado (y que el alumnado deberá completar) incluye la implementación de la constructora *Fecha()*, que deberá recibir como parámetro los valores enteros que representan el día, el mes y el año de la nueva fecha que se quiere construir. Lo primero que hace esta constructora es comprobar que estos tres valores representan una fecha válida, ya que si no es el caso, lo que hace es inicializar los atributos correspondientes con la fecha actual del sistema.

En el fichero *Fecha.java* también se incluye la implementación del método *toString()*, que devuelve la representación textual de la fecha. Este método, que no hay que modificar, resultará de mucha utilidad de cara a homogeneizar las pruebas que se realicen.

Y en relación a las pruebas que hay que realizar en la clase FechaTest, será necesario verificar la correcta implementación de la constructora *Fecha()*, así como de los métodos *incrementar()* y *decrementar()*. Algunos de los casos de prueba de la constructora consistirán en intentar construir una fecha que no es válida para comprobar que lo que se construye es en realidad un objeto con una fecha válida, a saber, la fecha actual del sistema. Para ello, se

recomienda utilizar la misma estrategia que se utiliza en la constructora *Fecha()* para generar la fecha de hoy, y después compararla con el resultado devuelto por la constructora. Por ejemplo:

```
Calendar c = new GregorianCalendar();
int dia = c.get(Calendar.DATE);
int mes = c.get(Calendar.MONTH) +1;
int annio = c.get(Calendar.YEAR);
Fecha fechaDelSistema = new Fecha(dia, mes, annio);
...
Fecha f1 = new Fecha(0,0,0); // fecha no válida
asssertEquals(f1.toString(), fechaDelSistema.toString());
...
```

Figura 2 – Ejemplo de JUnit para verificar la (no) construcción de fechas que no son válidas.

Tarea 2: la interfaz IFraccion

Se pide implementar la clase Fraccion, que tendrá dos atributos (el numerador y el denominador), sus correspondientes *getters* y los métodos que permitan realizar operaciones aritméticas (suma, resta, producto, cociente y simplificación de una fracción) y de comparación entre fracciones (igualdad, mayor o menor que). También habrá que implementar la clase FraccionTest, en la que se desarrollarán **todos los casos de prueba necesarios** para verificar la corrección de los métodos de la clase Fraccion.

La clase Fraccion deberá implementar la siguiente interfaz:

```
package org.pmoo.packlaboratorio3;


public interface IFraccion
{
  public abstract int getNumerador();
  public abstract int getDenominador();
  public abstract void simplificar();
  public abstract IFraccion sumar (IFraccion pFraccion);
  public abstract IFraccion restar (IFraccion pFraccion);
  public abstract IFraccion multiplicar (IFraccion pFraccion);
  public abstract IFraccion dividir (IFraccion pFraccion);
  public abstract boolean esIgualQue(IFraccion pFraccion);
  public abstract boolean esMayorQue(IFraccion pFraccion);
  public abstract boolean esMenorQue(IFraccion pFraccion);
}
```

Figura 3 - Interfaz IFraccion que se pide implementar.

La plantilla *Fraccion.java* que se facilita junto a este enunciado (y que el alumnado deberá completar) incluye la cabecera de la constructora *Fraccion()*, que recibe como parámetro los valores del numerador y del denominador. Esta constructora NO simplifica el objeto recién creado, pudiendo construirse, por ejemplo, las fracciones 8/6, -12/-12 o 2/-3. La única restricción en este sentido es que la constructora deberá impedir que se le asigne al atributo denominador el valor cero. Cuando esto ocurra, es decir, cuando el parámetro *pDenominador* valga cero, se le asignará el valor 1 al denominador, y además se mostrará un aviso por la consola del sistema.

El método *simplificar()* simplifica el objeto fracción (this), utilizando para ello (opcionalmente) el método auxiliar *mcd()*, que obtiene el máximo común divisor de dos números enteros. Aunque podría hacerse de otra manera, en esta tarea se va a suponer que el método *simplificar()* siempre deja el signo de la fracción en el numerador, de manera que al simplificar, por ejemplo, la fracción 4/-6 el resultado será la fracción -2/3, y no 2/-3.

Por su parte, los métodos *sumar()*, *restar()*, *multiplicar()* y *dividir()*, una vez realizada la operación correspondiente, devolverán la fracción resultado simplificada según lo comentado antes. Así, por ejemplo, el resultado de sumar las fracciones 2/-3 y 2/6 será -1/3 y no -2/6 ni tampoco 1/-3.

Los métodos *sumar()*, *restar()*, *multiplicar()* y *dividir()* utilizan dos operadores de tipo Fraccion: el que se encarga de realizar la operación correspondiente (esto es, this) y el que se recibe como parámetro (esto es, pFraccion). Como resultado de la ejecución de estos métodos deberá ocurrir lo siguiente:

1. Se ha generado un NUEVO objeto de tipo Fraccion, cuyos numerador y denominador se corresponden con el resultado de la operación (según sea el caso, la suma, la resta, la multiplicación, o la división) entre el objeto this y el objeto pFraccion (por ejemplo, this *menos* pFraccion). Este nuevo objeto es el que, una vez simplificado, se devuelve vía return.
2. ¡Importante! En el proceso de realizar la operación, no se ha modificado ninguno de los operadores. Es decir, el numerador y el denominador de los objetos this y pFraccion siguen valiendo lo mismo que valían al principio.

Los métodos *esIgualQue()*, *esMayorQue()*, y *esMenorQue()* tampoco deben modificar los operadores this y pFraccion. El booleano que devuelven indica si el objeto encargado de realizar la operación (es decir, this) es igual, es mayor, o, respectivamente, es menor que el objeto pFraccion que reciben como parámetro.