UNIVERSIDADE DA CORUÑA

Diseño Software
Curso 2016-2017

Boletín de Ejercicios 1

NOTA: Lee las instrucciones de los ejercicios antes de empezar a realizarlos. Os proporcionaremos algunos ejemplos de los tests JUnit que deben superar los ejercicios.
Fecha límite de entrega: viernes 14 de octubre de 2016

1. Considera el siguiente algoritmo definido para números enteros positivos:

   - Elige un número
     - Si el número es par divídelo por la mitad
     - Si el número es impar multiplícalo por 3 y súmale 1
   - Repite el proceso anterior hasta obtener un 1

   Por ejemplo, si comenzamos con el número 6 el proceso es como sigue:

   - 6, es par luego se divide por 2 $\Rightarrow$ 3
   - 3, es impar luego se multiplica por 3 y se le suma 1 $\Rightarrow$ 10
   - 10, es par luego se divide por 2 $\Rightarrow$ 5
   - 5, es impar luego se multiplica por 3 y se le suma 1 $\Rightarrow$ 16
   - 16, es par luego se divide por 2 $\Rightarrow$ 8
   - 8, es par luego se divide por 2 $\Rightarrow$ 4
   - 4, es par luego se divide por 2 $\Rightarrow$ 2
   - 2, es par luego se divide por 2 $\Rightarrow$ 1

   La conjetura de Collatz (debida a Lothar Collatz que fue el primero en establecer el problema en 1937) especifica que, independientemente del número con el que empecemos, siempre acabaremos en 1. La conjetura no ha podido ser probada formalmente, pero no se ha encontrado ningún número que la invalide.

   En base a este problema crea una clase Collatz que tenga los siguientes métodos estáticos:

- `int[] collatz(int n)`: Dado un número devuelve un array con la serie de números de Collatz (si le pasamos el 6 devolverá ⇒ 6, 3, 10, 5, 16, 8, 4, 2, 1). Para no tener que trabajar con un array de tamaño variable supondremos que el array tiene un tamaño fijo suficiente (p.ej. 300 elementos) para albergar la serie y que los elementos no utilizados se fijan a cero. Si `n` no es un entero positivo el método deberá lanzar la excepción `IllegalArgumentException`
- `int lengthCollatz(int[] list)`: Dado un array que contiene una serie de números de Collatz devuelve la longitud de la serie (recordar que el array tendrá siempre mas elementos que la serie que están inicializados a cero)

Tema: Bucles, arrays.

2. Crea una clase `UtilidadesFecha` que tenga los siguientes métodos estáticos:

   - `boolean esBisiesto(int anho)`. Indica si un año es bisiesto o no. Un año es bisiesto si es divisible entre 4, a no ser que que sea divisible entre 100, en cuyo caso deberá ser divisible entre 400 para ser bisiesto (1900 no es bisiesto pero el año 2000 sí) ⇒ Ver seminario JUnit.
   - `int diasMes(int mes, int anho)`. Indica el número de días de un mes. Como el número de días de febrero depende del año habrá que pasarle también el año.
   - `String convertirISO(String textoFecha)`. El formato de fecha ISO es un formato estándar que representa las fechas situando primero el año, luego el mes y por último el día: "*AAAA/MM/DD*". En dicho formato, por ejemplo, el "*28 de julio de 2006*" se representaría como "*2006-07-28*". El método `convertirISO` convierte una fecha en formato habitual de "*DD de MM de AAAA*" a su representación en formato ISO. Para simplificar las cosas podemos suponer que el formato habitual que le vamos a pasar es correcto, aunque la fecha puede ser incorrecta, por ejemplo, podemos pasarle el 31 de Febrero.
   - `boolean comprobarFechaISO(String fechaISO)`. Dado un `String` que representa una fecha en formato ISO comprueba que es una fecha correcta. Por fecha correcta entendemos que no existan letras ni caracteres extraños en la fecha, que tenga la longitud adecuada y que la fecha representada sea válida, es decir, que se tenga en cuenta el número de días en cada mes.

   Tema: Sentencias condicionales, cláusula `switch`, manejo de Strings

   De utilidad: Métodos de la clase `String` del API de Java

3. Crea una clase denominada `Racional` que permita representar y manipular números racionales. Los números racionales se representan normalmente de la forma a/b, dónde a es el numerador y b el denominador (que es distinto de cero). Las operaciones que pueden realizarse con un número racional son:

   - Suma: $a/b + c/d = (ad + bc)/bd$
   - Resta: $a/b - c/d = (ad–bc)/bd$
   - Multiplicación: $a/b \times c/d = ac/bd$
   - División: $(a/b)/(c/d) = ad/bc$

La clase debería incluir:

- Métodos constructores (por defecto, especificando numerador y denominador, y a partir de otro número racional – constructor de copia –).
- Métodos para realizar las operaciones de suma, resta, multiplicación y división (decide cómo tienen que ser los parámetros y el tipo de retorno).
- Un método `toString` para obtener una representación textual del número racional.
- Métodos de lectura del numerador y el denominador y ¿debería haber métodos de escritura para estos valores? dicho de otro modo ¿debería ser la clase Racional inmutable? Realiza la implementación escogiendo la opción que te parezca más adecuada.
- Método `equals` que permita comparar dos números racionales y decidir si son iguales. Consideramos que dos números racionales son iguales cuando son equivalentes, es decir, `1/2` sería igual a `2/4`. Dos fracciones son equivalentes los productos cruzados entre el numerador de una y el denominador de la otra son iguales: a/b es equivalente a c/d si $ad = bc$.
- Método `hashCode` cuyo comportamiento sea consecuente con el `equals` (siempre que dos objetos sean iguales su código hash debería ser el mismo).

Tema: Objetos, encapsulación, métodos, métodos `equals` y `hashCode`

4. Crea un tipo enumerado para representar las constantes de las monedas de Euro. Estas constantes deberán ser: `EURO2`, `EURO1`, `CENT50`, `CENT20`, `CENT10`, `CENT5`, `CENT2`, `CENT1`. Cada constante debe además almacenar el valor numérico que le pertenece (podemos usar como unidades los céntimos de Euro para evitar valores decimales) y deberá incluir un método de lectura de dicho valor.

   Crea una clase `Monedero` que permita contener un número indeterminado de monedas de euro. La clase `Monedero` debe incluir los siguientes métodos:

   - Introducir una moneda dada pasada por parámetro en el monedero
   - Extraer una moneda dada pasada por parámetro del monedero (si la moneda no existe el monedero no se modifica)
   - Devolver el número de monedas del monedero
   - Devolver el valor numérico (en céntimos de euro) de las monedas contenidas en el monedero
   - Contar el número de monedas que hay en el monedero de un determinado tipo (`EURO2`, `EURO1`, etc.) pasado por parámetro.

   Tema: Tipos enumerados, colecciones de objetos

   De utilidad: Clases como "ArrayList" son la mejor alternativa a los arrays para representar colecciones de objetos. Un inconveniente importante es que al extraer un objeto de la colección debe convertirse al dato adecuado de forma explícita. Por ejemplo: `Euro e = (Euro)monedas.get(i);`

5. El objetivo del ejercicio es implementar las clases básicas para utilizar en un supermercado. Necesitaremos por tanto almacenar información de Clientes, Dependientes y Reponedores. De todos ellos necesitamos saber su nombre, apellidos, DNI, dirección y teléfono. A parte de eso necesitamos la siguiente información específica de cada clase:

   - De los clientes también nos interesa su código de cliente y el número de compras realizadas, esta última característica es necesaria para obtener el descuento que se le debe aplicar al cliente (cada 100 compras se le aplica un 1 % de descuento), debemos por lo tanto implementar un método compra que anote que realizó una compra.
   - De los dependientes y reponedores debemos saber su número de la seguridad social, su salario y el turno al que pertenecen (mañana, tarde o noche). Para obtener el salario se tendrá en cuenta que si trabaja en turno de noche tiene un extra de 150 euros.
   - De los dependientes almacenaremos su especialidad (carnicería, frutería, caja, etc.) y de los reponedores almacenaremos información de la empresa de la que proceden ya que se suelen subcontratar.

   Todos los atributos se definirán privados y deberán disponer de métodos de lectura y escritura. Se deberá incluir un método `toString` que sobrescriba el método `toString` de `Object` y que permita imprimir toda la información de cada clase. Emplea clases abstractas para generalizar características comunes entre distintas subclases.

   Tema: Encapsulación de atributos y clases abstractas.