neveras.pdf

Cerebrandex

Programación Orientada a Objetos

1º Grado en Ingeniería Informática

Escuela Técnica Superior de Ingeniería Informática
Universidad de Málaga

Programación Orientada a Objetos: Control 26 de Mayo de 2020

En este ejercicio vamos a crear una aplicación para gestionar los productos de una nevera. Para ello se creará un paquete `neveras` con clases `Fecha`, `Producto`, `Nevera` y `NeveraInteligente`.

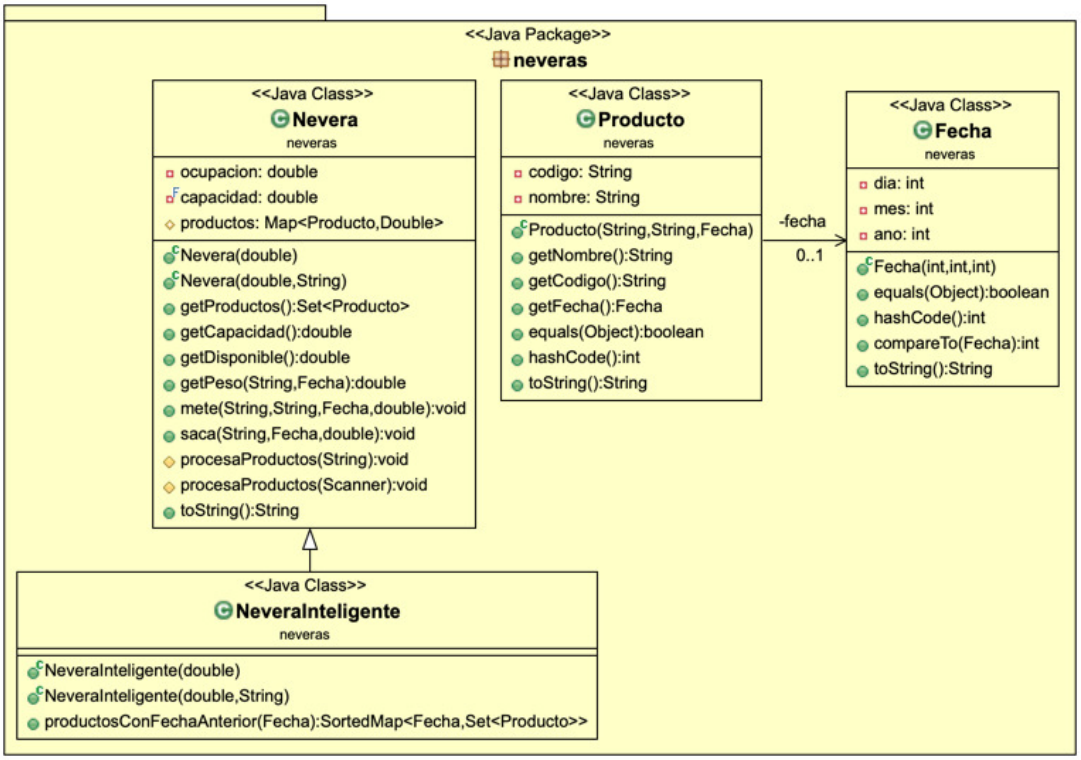


Figura 1: Diagrama de clases UML

Clase `Fecha`

Una fecha tiene un día, un mes y un año. Para simplificar, supondremos que los días vienen representados por un valor entero entre 1 y 31, los meses por un entero entre 1 y 12 y los años por un valor entero. La clase tendrá un único constructor `Fecha(int d, int m, int a)` al que se se propocionará como argumento el día `d`, mes `m` y año `a` con los que crear la fecha. Argumentos no válidos producirán una excepción del tipo `IllegalArgumentException`. Las fechas se representarán en el formato día/mes/año. Así, por ejemplo, el 26 de mayo de 2020 se representará como `"26/5/2020"`.

Clase `Producto`

Un producto tiene un código, un nombre y una fecha de caducidad. Así, por ejemplo, podemos tener un producto con nombre `"zumo"` y código `"Z123"` que caduca el 26/5/2020. Ninguno de los tres componentes de un producto pueden ser `null`. Un código vendrá dado por una cadena de caracteres de longitud 4, y el nombre de un producto por una cadena de caracteres no vacía.

1. La clase `Producto` tendrá un único constructor, `Producto(String c, String n, Fecha f)`, en el que se proporcionarán como argumentos el código `c`, el nombre `n` y la fecha de caducidad `f` del producto. Argumentos no válidos producirán una excepción del tipo `IllegalArgumentException`.

2. La clase proporciona métodos para consultar el código, el nombre y la fecha de caducidad del producto.
3. Dos objetos de la clase `Producto` se considerarán iguales si tienen el mismo código (sin distinguir mayúsculas y minúsculas) y la misma fecha de caducidad.
4. La representación como cadena de caracteres de un objeto de tipo `Producto` debe mostrar el nombre del producto, su código y su fecha de caducidad con el formato utilizado en el siguiente ejemplo: `"zumo: Z123: 26/5/2020"`.

Aplicación `MainProducto`

La ejecución de la clase `MainProducto` proporcionada debe dar la siguiente salida como resultado:[1]

```
Los productos queso: A123: 26/5/2020 y tomate: A124: 26/5/2020 no son iguales
Los productos tomate: A124: 26/5/2020 y tomate rallado: A124: 26/5/2020 son iguales
Los productos queso: A123: 26/5/2020 y tomate rallado: A124: 26/5/2020 no son iguales
Los productos tomate: A124: 26/5/2020 y tomate rallado: A124: 20/5/2020 no son iguales
java.lang.IllegalArgumentException
 at neveras.Producto.<init>(Producto.java:10)
 at pruebas.MainProducto.main(MainProducto.java:17)
java.lang.IllegalArgumentException
 at neveras.Producto.<init>(Producto.java:10)
 at pruebas.MainProducto.main(MainProducto.java:22)
java.lang.IllegalArgumentException
 at neveras.Producto.<init>(Producto.java:10)
 at pruebas.MainProducto.main(MainProducto.java:27)
java.lang.IllegalArgumentException
 at neveras.Producto.<init>(Producto.java:10)
 at pruebas.MainProducto.main(MainProducto.java:32)
java.lang.IllegalArgumentException
 at neveras.Fecha.<init>(Fecha.java:8)
 at pruebas.MainProducto.main(MainProducto.java:37)
java.lang.IllegalArgumentException
 at neveras.Fecha.<init>(Fecha.java:8)
 at pruebas.MainProducto.main(MainProducto.java:42)
java.lang.IllegalArgumentException
 at neveras.Fecha.<init>(Fecha.java:8)
 at pruebas.MainProducto.main(MainProducto.java:47)
```

Clase `Nevera`

Crea una clase `Nevera` en la que almacenar productos. En concreto, las instancias de la clase `Nevera` almacenan los productos que contienen y los pesos de cada uno de ellos en una correspondencia de tipo `Map<Producto, Double>`, de forma que asociamos a cada producto la cantidad del mismo en la nevera. Consideraremos iguales a productos con el mismo código y la misma fecha, de forma que podemos tener en la nevera, por ejemplo, productos con el mismo código y fechas distintas. Por ejemplo, podemos tener 1000 gr. de leche que caduca el 30/6/2020 y 1000 gr. de leche que caduca el 30/7/2020 como productos diferenciados y asociados a ellos sus pesos correspondientes.

Una nevera admite un peso máximo determinado, y no podrán almacenarse en ella más productos que los posibles según la capacidad de la misma. Intentos de meter en una nevera cantidades de productos que superen su capacidad resultarán en inserciones parciales, es decir, se insertará la parte del producto que quepa y se descartará el resto. Una variable `ocupacion` mantendrá el peso total de los productos en la nevera en cada momento.

1. La clase `Nevera` proporcionará dos constructores, uno al que se le pasa la capacidad de la nevera, y un segundo constructor que recibirá además un string con el nombre del fichero del que se leerá

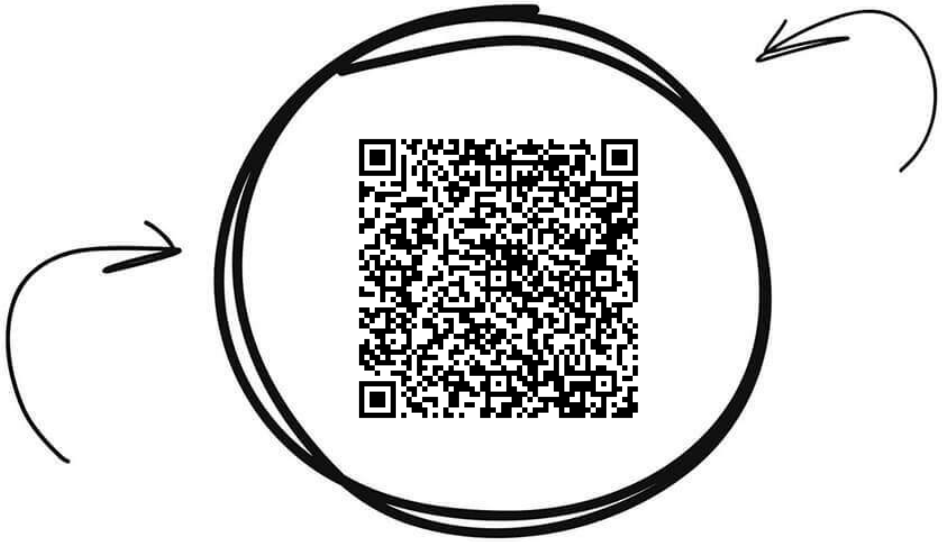
[1] Los números de línea de las excepciones pueden no coincidir con las indicadas en las salidas.

¿DÍA DE CLASES infinitas?

WRIGLEY'S Orbit

masca y fluye

Programación Orientada a Obj...



Banco de apuntes de la

WUOLAH

Comparte estos flyers en tu clase y consigue más dinero y recompensas

1. Imprime esta hoja
2. Recorta por la mitad
3. Coloca en un lugar visible para que tus compis puedan escanar y acceder a apuntes
4. Llévate dinero por cada descarga de los documentos descargados a través de tu QR

la información sobre los productos con los que se inicializará la nevera. Véase el formato en los ficheros de entrada ejemplo proporcionados.

Supondremos que los códigos y nombres de productos no contienen retornos de carro (`\n`) ni los caracteres `:` o `;`. La información de cada producto vendrá separada por el carácter `;` (posiblemente con espacios en blanco y retornos de carro alrededor). La información sobre un producto (el código, el nombre, el peso y la fecha de caducidad del producto vendrán separados por el carácter `':'`). Los componentes de las fechas vendrán en el mismo formato con el que los representamos. Así, por ejemplo, la información de uno de estos productos en uno de los ficheros proporcionados viene dada como `A123:tomate:200.2:20/5/2020`, donde en primer lugar nos encontramos el código `A123`, a continuación el nombre del producto `tomate`, la cantidad de producto `200.2` y por último su fecha de caducidad `20/5/2020`.

Si cualquiera de los datos proporcionados no se ajusta a las suposiciones establecidas el producto en cuestión se descartará. Como resultado de crear una nevera de capacidad 800 con el contenido del fichero `productos3.txt` proporcionado tendríamos 3 productos (200 gr. de tomate, 180 de lechuga y 420 de leche). Como hemos dicho anteriormente, se meterán productos mientras estos quepan, por lo que no es posible meter los 500 gr. de leche especifidos, sino tan solo los 420 hasta ocupar el total de la capacidad de la nevera.

2. La clase proporcionará métodos `getProductos()`, `getCapacidad()` y `getDisponible()` que devolverán, respectivamente, un conjunto con los productos en la nevera, su capacidad y el espacio disponible.

3. El método `getPeso(String c, Fecha f)` devolverá el peso de todos los productos en la nevera con código `c` cuyas fechas de caducidad sean anteriores a `f`. Si no hubiera ningún producto con el código especificado y fecha de caducidad anterior se devolverá `0`.

4. La clase proporcionará métodos `mete(String c, String n, Fecha f, double p)` y `saca(String c, Fecha f, double p)` para, respectivamente, meter y sacar una cierta cantidad `p` del producto con código `c` y fecha de caducidad `f` en / de la nevera. Si el código, el nombre, la fecha o el peso son no válidos se lanzará una excepción `IllegalArgumentException`.

   Al meter un producto en la nevera se tendrá en cuenta que si el producto ya está (mismo código y misma fecha de caducidad) simplemente se incrementará la cantidad de dicho producto. Igualmente, recuérdese que no se puede superar la capacidad máxima de la nevera, y que por tanto si no cabe un producto completo se meterá solo la parte que quepa hasta alcanzar el máximo de la capacidad de la nevera.

   Al extraer una cantidad de producto con el método `saca(String, Fecha, double)` se decrementará el peso en la nevera del producto especificado en la cantidad indicada. Si la cantidad fuese negativa o superior al peso disponible en la nevera se lanzará un excepción `IllegalArgumentException`. Si la extracción resultara en un peso 0 del producto este será extraído de la nevera, eliminándolo de la correspondencia `productos`.

5. La clase dispondrá de una representación de los objetos como cadenas de caracteres como la que se muestra en el siguiente ejemplo:

   ```
   "500.0 / 500.0: [[tomate: A123: 20/5/2020 (200.2)], [maiz: A126: 20/5/2020 (299.8)]]"
   ```

   En esta representación se muestra el peso de todos los productos, seguido de la capacidad de la nevera y del contenido de la nevera, donde para cada producto se especifica la cantidad del mismo en la nevera. Como puede verse en la salida del progama `MainNevera` a continuación, productos distintos (incluidos productos con el mismo código y fecha de caducidad distinta) se mostrarán por separado.

Aplicación `MainNevera`

La ejecución de la clase `MainNevera` proporcionada debe dar la siguiente salida como resultado:[2]

```
2000.0 / 2000.0: [[tomate: A123: 20/5/2020 (200.2)], [lechuga: A124: 20/5/2020 (150.0)], @
                  [lechuga: A124: 26/5/2020 (180.0)], [leche: A126: 24/5/2020  (1119.8)], @
                  [maiz: A126: 20/5/2020 (350.0)]]
1900.0 / 2000.0: [[tomate: A123: 20/5/2020 (200.2)], [lechuga: A124: 20/5/2020 (150.0)], @
                  [lechuga: A124: 26/5/2020 (80.0)], [leche: A126: 24/5/2020 (1119.8)], @
                  [maiz: A126: 20/5/2020 (350.0)]]
1750.0 / 2000.0: [[tomate: A123: 20/5/2020 (200.2)], [lechuga: A124: 26/5/2020 (80.0)], @
                  [leche: A126: 24/5/2020 (1119.8)], [maiz: A126: 20/5/2020 (350.0)]]
2000.0 / 2000.0: [[tomate: A123: 20/5/2020 (200.2)], [lechuga: A124: 30/5/2020 (260.0)], @
                  [lechuga: A124: 26/5/2020 (80.0)], [leche: A126: 24/5/2020 (1119.8)], @
                  [maiz: A126: 20/5/2020 (350.0)]]
A124 anterior a 1/6/2020: 330.0
500.0 / 500.0: [[maiz: A126: 24/5/2020 (299.8)], [tomate: A123: 24/5/2020  (200.2)]]
880.0 / 2000.0: [[leche: A126: 24/5/2020 (500.0)], [tomate: A123: 24/5/2020 (200.0)], @
                 [lechuga: A124: 24/5/2020 (180.0)]]
```

Clase `NeveraInteligente`

La clase `NeveraInteligente` representa neveras con una funcionalidad adicional consistente en poder informarnos de los productos con fecha de caducidad anterior a una fecha dada. En concreto, la clase proporciona un método `productosConFechaAnterior(Fecha)` que devuelve aquellos productos de la nevera con fecha anterior a la especificada agrupados por fecha como una correspondencia ordenada de tipo `SortedMap<Fecha, Set<Producto>>`. La clase proporciona constructores con los mismos argumentos que los de la clase `Nevera`.

Aplicación `MainNeveraInteligente`

La ejecución de la clase `MainNeveraInteligente` proporcionada debe dar la siguiente salida como resultado:

```
2000.0 / 2000.0: [[tomate: A123: 20/5/2020 (200.2)], [lechuga: A124: 20/5/2020 (150.0)], @
                  [lechuga: A124: 26/5/2020 (180.0)], [leche: A126: 24/5/2020 (1119.8)], @
                  [maiz: A126: 20/5/2020 (350.0)]]
1900.0 / 2000.0: [[tomate: A123: 20/5/2020 (200.2)], [lechuga: A124: 20/5/2020 (150.0)], @
                  [lechuga: A124: 26/5/2020 (80.0)], [leche: A126: 24/5/2020 (1119.8)], @
                  [maiz: A126: 20/5/2020 (350.0)]]
1750.0 / 2000.0: [[tomate: A123: 20/5/2020 (200.2)], [lechuga: A124: 26/5/2020 (80.0)], @
                  [leche: A126: 24/5/2020 (1119.8)], [maiz: A126: 20/5/2020 (350.0)]]
2000.0 / 2000.0: [[tomate: A123: 20/5/2020 (200.2)], [lechuga: A124: 30/5/2020 (260.0)], @
                  [lechuga: A124: 26/5/2020 (80.0)], [leche: A126: 24/5/2020 (1119.8)], @
                  [maiz: A126: 20/5/2020 (350.0)]]
A124 anterior a 1/6/2020: 330.0
Productos con fecha anterior a 26/5/2020: @
  {20/5/2020=[tomate: A123: 20/5/2020, maiz: A126: 20/5/2020], 24/5/2020=[leche: A126: 24/5/2020]}
```

[2] Para mostrar la salida se han introducido retornos de carro en varios lugares. Estos vienen indicados con el símbolo `@`.