Programación Orientada a Objetos – Curso 2024-25

Práctica 5 – Marketplace. La clase Basket.

En moodle encontrarás los siguientes ficheros que tendrás que descargar al directorio marketplace/tests:

- basket-test.cc
- Nuevo fichero CMakeLists.txt para los tests.

Crea el directorio poo/p5 con una copia del directorio poo/p4.

EJERCICIO 1. La clase *Basket*
La clase Basket representa la cesta de productos de un cliente del marketplace. Codifica la clase Basket en los ficheros basket.h y basket.cc en el directorio marketplace/src/basket y añade la siguiente funcionalidad.

1. Basket tiene una lista de productos '`product_list_`' (una std::list de objetos de tipo Product) a la que el cliente irá añadiendo productos.
2. La clase Basket debe definir un mapa '`product_quantity`' o diccionario para contabilizar la cantidad de cada producto que hay en la cesta. Para ello usaremos std::map de la STL para lo que debemos usar el siguiente include:

   #include <map>

   Podemos definir un mapa de la siguiente manera:

   std::map<std::string,int> product_quantity_;

   En este mapa, en cada par key-value, 'key' será el id del producto y 'value' la cantidad de ese producto en la cesta.

   Cuando se añada por primera vez un producto a la cesta se hará;

   product_quantity_[id] =1;

   Y cuando se añada de nuevo el mismo producto se incrementará en 1.

   Puedes comprobar si una clave está en un mapa con la función:

   product_quantity_.count(id)

   que devuelve 1 si existe el 'id' y 0 si no existe dicho 'id'

3. Basket también tiene un atributo de tipo float '`total_`' con la suma total del coste de los productos de la cesta.
4. Constructor que inicializa a 0.0 el valor de '`total_`'
5. AddProduct(): de tipo void que recibe un producto y lo añade a la cesta. Ten en cuenta lo siguiente:
   ◦ Si no existe en la cesta (en la lista product_list_) un producto con ese id, se añade a

product_list_ normalmente y se pone a 1 la entrada del mapa con su id.
◦ Si ya existe en la cesta un producto con ese id, debe incrementarse la cantidad de ese producto en el mapa, pero no añadirse a la lista de productos.
◦ Cada vez que se añade un producto en la cesta o se incrementa su cantidad, se debe actualizar el precio total de la cesta (atributo total_ de la clase Basket).

6. DeleteProduct(): Recibe un producto a borrar. Ten en cuenta lo siguiente:
◦ Si no existe un producto en la cesta con ese id, no se hace nada y se devuelve false.
◦ Si ya existe en la cesta un producto con ese id debe decrementarse la cantidad de ese producto en la cesta. Si queda igual a 0, se elimina ese producto de la lista y su correspondiente entrada en el mapa.
◦ Cada vez que se borra un producto en la cesta o se decrementa su cantidad, se debe actualizar el precio total de la cesta (atributo total_ de la clase Basket).
7. DeleteProduct(): Recibe el id de un producto a borrar. Ten en cuenta lo mismo que en la versión anterior de esta función.
8. DeleteBasket(): que borra todos los elementos de la cesta y deja todo en su estado inicial.
9. GetSize(): devuelve el número de productos en la cesta (número de elementos de la lista).
10. GetTotal(): devuelve el valor del dato de tipo float ‘total’.
11. GetIds(): devuelve un vector de std::string con los ids de todos los productos de la cesta. Si la cesta está vacía, devuelve un vector vacío.
12. GetQs(): devuelve un vector de int con las cantidades todos los productos de la cesta, de modo que el primer elemento del vector sea la cantidad del primer producto de la cesta, el segundo la cantidad del segundo producto, etc. Si la cesta está vacía, devuelve un vector vacío.

Añadir un sencillo programa principal para probar la clase Basket que se llame ‘basket-main.cc’ que declare un objeto de tipo Basket y muestre un pequeño menú para probar cada función.

No olvides añadir el CMakeLists.txt con las modificaciones necesarias en el directorio basket y actualizar el del directorio src para poder compilar con la implementación de la nueva clase.

Pasar los tests: basket_test.cc