TNG033/2017: Dugga 2

Attention

This is an **individual** dugga. Thus, it strictly forbidden to use email, chat forums, or any form of exchanging information about the dugga through the internet, until the deadline closes (19:00).

Books, notes, and internet material can be used.

If you find any open questions in some of the presented exercises then feel free to make your own decisions. However, your decisions should be reasonable and must not contradict any of the conditions explicitly written for the exercise. Please, write comments in the programs that clarify your assumptions/decisions, if any.

If you use Code::Blocks then follow the instructions given here. Remember that you should not ignore the compiler warnings, since they are usually an indications of serious problems in the code.

You are not allowed to post any information about the dugga in any website until the deadline expires.

Course goals

This is an assessment of whether you can define hierarchies of classes satisfying given requirements and use the standard template library (STL) components and algorithms.

Requirements for grade G

- Readable and well indented code.
- Use of good programming practices.
- Global variables cannot be used[1].
- Respect for submission instructions.
- Code copied from a web page must be clearly indicated through a commented line containing the web page address used as source, with exception for the material posted on the course website.
- Use of statements that are not part of the ISO C++ leads to automatically failing the dugga.
- Your programs must pass all test examples shown in this paper.
- Other criteria are listed in the feedback report document available from course website.

Note that there is no guarantee that your code is correct just because it passes all given test examples. Thus, you should always test your code with extra examples, you come up with.

Deadline

15 of December, 19:00.

[1] All variable must be declared inside a function of your program or as class members.

Submission instructions

1. Create **one** source (`.cpp`) file for the exercise. The file must be named with your LiU login (e.g. `ninek007.cpp`).
2. Write your name and personal number at the beginning of the source code file.
3. Submit only the source code file (`.cpp`) through Lisam, without compressing it (i.e. neither `.zip` nor `.rar` files are accepted).

Remember that you should deliver only the source code file.

It is the last submission made in Lisam that is graded. All other submissions you may have uploaded in Lisam are not considered.

Duggor solutions submitted not accordingly the submission instructions are ignored.

Login and password to access the course material

All course material, like lecture slides and code examples, are available through the course website (**`http://weber.itn.liu.se/~aidvi05/courses/12/index.html`**). However, the material is password protected. Use the following login and password to access the material.

login: **`TNG033`** password: **`TNG033ht13`**

Lycka till!!

Exercise

In this exercise, you are requested to define first a polymorphic class hierarchy of shops and then define a class that represents companies that manage a product's brand.

Start by designing a polymorphic class hierarchy for representing different types of shops that sell a product. The base class of this hierarchy is the class **`Shop`**. This class should store the name of the shop (**`string`**), the price of the product sold by the shop (**`double`**), and (a reference to) an output stream (like **`std::cout`**) where information about the shops can be logged. Additionally, the following (basic) functionality should also be provided.

- A constructor. When a shop is created, its name, the product's price, and the log output stream must be explicitly initialized and, thereafter it should not be possible to modify the shop's name. In addition, the constructor must guarantee that the type of shop being created (e.g. "*Outlet shop*"), shop's name, and price are written to the log output stream (e.g. a message like "*Outlet shop created: Vero Moda, start price = 200*").
- A function **`get_name`** that returns the shop's name.
- A function **`update`** that updates the product's price and logs the price change. This function has as argument a reference price (**`double`**). The type of shop updating the price (e.g. "*Vip shop*"), shop's name, current product's price, and the new updated price are written to the log output stream. How prices are updated depends on the type of shop.
- The output stream for logging purposes should not be available to the sub-classes. In other words, it should not be possible for the sub-classes to access directly the output stream (i.e. only class **`Shop`** should be able to write in the log output stream).
- It should not be possible to create copies of `Shop` objects.
- It should not be possible to created instances of class **`Shop`** that are not instances of one of its subclasses (like **`Vip_Shop`** or **`Outlet_Shop`**).

`Vip_Shop`, a sub-class of **`Shop`**, should also store the price factor (**`int`**). When a VIP shop is created its name and product's price shall always be given. The log output stream is optional, and if not given then it should be initialized to **`std::cout`**. The price factor should always be initialized to one.

Vip shops always update the price of the product by multiplying the price factor by the given reference price. Additionally, it should be possible to modify the price factor through a function **`set_price_factor`**.

`Outlet_Shop`, another sub-class of **`Shop`**, should also store the discount (**`double`**) to be applied when the product's price is updated. When an outlet shop is created its name and product's price shall always be given. The log output stream is optional, and if not given then it should be initialized to **`std::cout`**. The discount should always be initialized to zero.

Outlet shops always update the price of the product by applying the discount to the given reference price. For instance, if the discount is 0.2 then the price should be updated to

$$reference\ price - 0.2 \times reference\ price.$$

Additionally, it should be possible to modify the discount through a function **`set_discount`**.

Finally, define a class **`Brand_Manager`** that represents companies that manage a product sold by several shops. This class should store the name of the company (**`string`**), the reference price of the product (**`double`**), and a "list of the shops"[2] selling the product managed by the company. Additionally, the following (basic) functionality should also be provided.

- A constructor. When a brand manager is created, its name must be explicitly initialized and, thereafter it should not be possible to modify it. The reference price of the product is set to zero.
- A function **`get_name`** that returns the company's name.
- Function **`set_reference_price`** that modifies the reference price to a new given price.
- Function **`register_shop`** that allows a shop to register itself as selling the company's product (i.e. the shop should be added to the brand manager's "list of the shops"). Make sure that the same shop (i.e. an instance of class `Shop`) is not registered twice.
- Function **`display_registered_shops`** that displays the name of all shops registered at the company.
- Function **`notify`** that requests all registered shops to update their selling prices, by calling function **`Shop::update`** and passing to the shops the reference price of the product.
- It should not be possible to create copies of **`Brand_Manager`** objects.

No other public member functions other than those described above are allowed[3].

Design your classes with care and follow the given specifications. Keep in mind that other types of shops, using different strategies to update the prices, can be added to the shop's hierarchy in the future.

Make use of good programming practices in your code.

Add your code to the file `dugga2.cpp` that contains a simple `main` function to test incrementally your code. The `main` should be delivered without modifications.

The expected output is available in the file `dugga2_out.txt`. Note that we cannot guarantee that your solution is correct just because it passes the given tests.

Finally, remember to rename the file `dugga2.cpp` with your LiU id.

[2] Note that in this sentence, "*a list of the shops*" is just an informal expression. It is you who decides how to implement in C++ the "*list of shops*".
[3] You can add non-public member functions, though.