Object-Oriented Programming with C++ *#1*

Lab session 1 – Part 2

Please consider visiting also the following two pages, or any other document you may find useful, whenever you wish: http://en.cppreference.com/w/cpp , http://www.cplusplus.com/reference .

Do not rely solely on the lecture's content!

11. A database of products

Move to folder `11.database`.

You are going to model a simple database to store and retrieve products to be sold. The database will also contain customers, sellers and product shippers. Later, you will have to handle product orders.

Of course, you are going to model it using an Object-Oriented approach.

First, before starting to code, you will have to sketch the object model. To achieve that, please consider using the UML Class Diagram (https://en.wikipedia.org/wiki/Class_diagram). There is no need for something complex: just sketch the classes, the most important relations (as overviewed in the first lecture) and the most relevant methods and attributes.

Please discuss the model with your supervisor. Do not start to code immediately! Also, if something is unclear or ambiguous, do not hesitate to discuss and clarify it!

The database contains products. There are different categories of products: electronics, clothing… A given product may have multiple categories (e.g. a smart watch is both clothing and electronics). All the products have a price, a weight and a description. Also, each product has its own seller and its own shipper. Finally, each product is identified by an ID which is unique among all the products of the database.

There are different types of products: TShirt, Shoes, Laptop, SmartWatch, CPU, Memory, Disk, Graphics. The first two products are clothing. A SmartWatch is clothing and electronics. The others are electronics. Other product types and categories may be added in the future.

A TShirt has a brand (just a name). Shoes have a color. Currently only the following colors are sold: black, white, red, green, blue. Other colors may be added in the future.

A CPU has a number of cores and a frequency. A Memory has a size in bytes. The Disk has a size too and can be SSD or not. A Graphics (card) can be PCIexpress or not.

A Laptop is made of a CPU, a Memory, a Disk and a Graphics card. A SmartWatch has a CPU and a Memory too, as well as a color. Mind that the CPU of a Laptop or SmartWatch is a product itself (and could be sold alone). The same applies to the other components.

A Customer has a first name, a last name and an address (just a text). A Customer also has an ID which is unique among all the customers of the database.

Both Seller and Shipper have a name. They also have unique IDs.

About the persistence of the data, the Database will load from a file and store its content to a file. The file format is made of rows of text. The order is not important. Each row identifies a specific product, or a customer, or a seller or a shipper. It is important to understand that a row only contains the items it describes. For instance, a row will not contain the TShirt and its Seller and its Shipper. However, the TShirt row will reference the seller and the shipper using their IDs.

Thus, after each load from a file, a database will automatically perform an indexing: it will visit all the loaded products, customers, sellers and shippers, and it will "link" them. For instance, when you get a Laptop from the database, the Laptop instance knows its Seller.

The Database knows how to load from a file. It loops through each line, identifies the type to load (a Laptop product, a TShirt, a Customer…), creates a default instance for that type, then asks the instance to deserialize itself from the given line of text. Thus, each type knows how to deserialize (or unmarshal) itself. The deserialization must have a minimal level of robustness: if an error is encountered when parsing the line, the object instance must not be overwritten with faulty data. But if everything is ok, all the instance's fields will take the read values. It is possible that a value is missing (e.g. a product description for a Disk): in that case, the instance will take the default value ("" for strings, 0 for numbers). IDs cannot be missing.

The database also knows how to store to a file. It asks each instance to serialize (or marshal) itself.

The format of the database file is a follows

- Each line represents an instance.
- Instances are made of records. Each record is separated from the next using the RS (record separator) character.
- Records are made of a record identifier and a value. The record identifier is a very short text made of capital letters. It denotes the type of the instance or the instance's field. The value is simply the printed value, that is, the one you would get by sending it to the standard output (cout << value). It may also be any text value automatically parsable. The record identifier and the value are separated by the US (unit separator) field.

  Example:
  KI***US***CPU***RS***ID***US***1***RS***PR***US***45.9***RS***WE***US***34.2***RS***DE***US***Dual   CORE***RS***CA***US***1***RS***CO***US***2***RS***HZ***US***2.4e+10
  where ***US*** is the unit separator character and ***RS*** the record separator, KI denotes the kind, ID is the unique id field, PR is the price field…

Your mission:

a. Sketch the diagram of the object model.
b. Implement each class. Test their marshaling and unmarshaling capabilities. Nice to have: products, customers, sellers and shippers know how to print in human-readable text.
c. Implement the database class.

I. Create a few instances of products and other objects, add them to the database, then store the database to a file.
II. Load the database `db.txt` (provided file).

d. Implement a search of the product the most expensive in the database.
e. Add a class Order. An order involves a customer and one or more products. Mind that a product may change after an order instance has been made: the order should not change.
f. Create an order with the first three products in the database and the first customer. Then modify one of the involved products in the database. Check that the order is unchanged.
g. Create a collection of Order instances, filled with all the products and the customers of the database (do a “circular” loop on the products and customers, and add four products and one customer per Order instance).
   Search for the most expensive order.