Homework 2 for CS202
Bradley Fallon
bfallon@pdx.edu
5/8/2019

DESIGN ANALYSIS

-Overview
The Shipper class is the root base class for all shipper classes. The shipper is not abstract, so it is ok to create an instance of a shipper. A base shipper will just be a standard package that gets left at the door. There are two directly derived classes, they are the gift, which comes with return instructions, and the signed package, which tracks whether the package has been signed for. Derived from signed shipper, we have the installed shipper. This tracks if the delivery has been signed for and also tracks if the product has been installed at the delivery location.

-RTTI with dynamic casting
The classes each have a copy constructors that work by taking a constant object by reference.

This assignment involves RTTI run time type identification, because it was part of the assignment to make use of this. I may have attempted to find a way to avoid using RTTI via dynamic casting if I didn't think it was the intended strategy for this assignment. I made use of dynamic casting RTTI in a copy function and in a display function. I think the display function could have achieved the same behavior by using a virtual display function on the root base class. The copy function seems to actually be a somewhat reasonable application for dynamic casting RTTI. The copy takes in a pointer of type root base, but the actual data it points to might be an instance of any of the derived classes. This will then create a new instance of the correct type and return a pointer to it.

This means that there is new memory allocated in the copy function which is used by the queue class. This is problematic, because the queue is doing nothing to manage whether the client deallocates the memory. I don't know an easy way to solve this, because I cannot copy to a reference object provided by the client, because the client may not know what type to use. My solution was to have the client deallocate memory when finished. A more advanced approach could be to somehow track if the memory still points to the same objects by having constant unique id properties. The destructor of the queue would go through and check if the memory is still allocated with the same unique constant id, and clean up if needed. This is not something that I felt was in scope for this assignment. Another approach would be to use more abstraction, having anything that deals with different shipper types be dealt with by a queue manager class, which is derived from queue. The manager class would ask the client what type of package they want and then it would handle the rest. This seems like it would be a suitable approach for this course, but given the time I am willing to spend on this, and the fact that this was no included in the design we were given, I am instead just programming my test client to manage its own memory.