C++ 2 wk 2

Khaula Molapo

Year 2 Semester 2

1.

Imagine a modern inventory system for a large retail store, built using advanced classes in C++. Each item in the store is represented as an object with attributes like item ID, name, price, quantity, and supplier information. An advanced class design allows inheritance, so special categories like perishable goods or electronics can extend the base class and add unique properties such as expiry dates or warranty details. Operator overloading can make updating stock as simple as using "+" or "-" operators to increase or decrease inventory. Encapsulation keeps sensitive data like supplier pricing private, while public methods provide controlled access. Polymorphism allows flexible handling of different item types through a single interface, making the system scalable and efficient. This advanced class approach not only makes the code easier to maintain but also provides the store with a reliable, flexible, and secure way to track its vast inventory.

2.

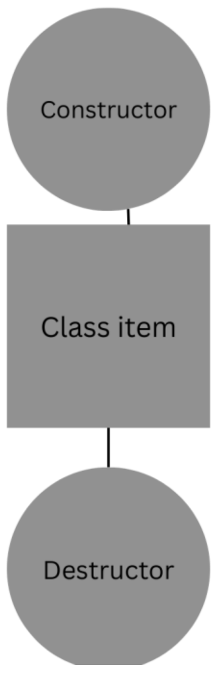
Operator overloading in C++ is an advanced concept that allows developers to redefine how operators such as +, -, =, or [] behave for user-defined classes. Instead of limiting operators to built-in types, operator overloading makes them usable with custom objects. For example, in an inventory system, "+" could be overloaded to add stock, while "-" reduces it. This improves readability and makes the code more intuitive, since the operations resemble natural arithmetic. While powerful, operator overloading should be applied carefully to avoid confusing results. Its main role is to enhance clarity, maintainability, and consistency when working with complex classes.

3.

Writing and running this C++ class in Visual Studio Code gave me hands-on practice with constructors and destructors. The constructor automatically initializes the object with a name and quantity, while the destructor shows when the object is destroyed. This helped me understand how C++ manages object lifecycle, especially memory management and clean-up. Running the program made me see the order of execution clearly, since the constructor message appeared first and the destructor last. Visual Studio Code also made it easier to debug and compile. Overall, this exercise improved my understanding of how classes manage data safely and efficiently.

4.

In Canva, I created a simple advanced class diagram to represent the lifecycle of a class from constructor to destructor. The diagram uses a rectangle for the class, with arrows pointing to the constructor at the start and the destructor at the end, showing the flow of object creation and destruction. Additional shapes represent member variables and functions. This visual design makes it easier to understand how objects are initialized, used, and then cleaned up. By using arrows and labels, the diagram clearly shows how constructors set up data while destructors free resources once the object’s job is complete.


Constructor
Class item
Destructor