COSE213 Data Structures

Programming Assignment #2: Smart Device Management System

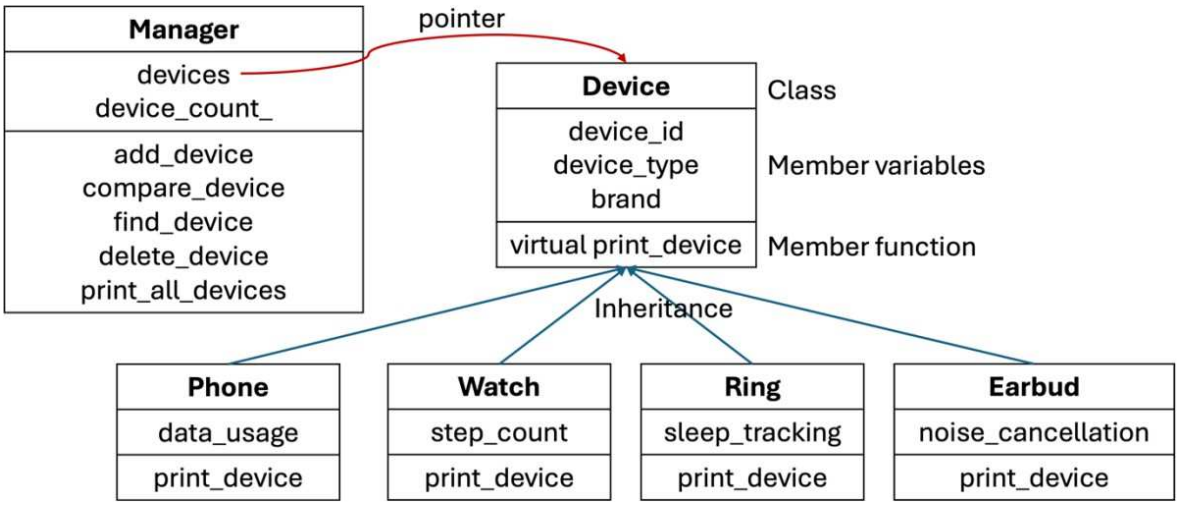
(Deadline: 3.24 (Mon) ~ 4.7 (Mon))

Introduction

In this programming assignment, you will develop a device management system that builds on the foundational C++ programming concepts introduced in our class. This assignment is designed to reinforce your understanding of key object-oriented programming principles, including inheritance, operator overloading, and virtual functions, which are essential for understanding the upcoming data structures in this class.

We are living in the era of multiple smart devices! Your task is to create a system that efficiently manages various types of devices, providing functionality such as adding, deleting, and searching for devices.

Class Structure



Device: The Device class needs to have the following members: device_id, device_type, and brand. These members should be accessible to derived classes but not to external classes. The Device class has the following methods:

1. print_device(): a pure virtual function in this class and prints all the information about the device,. This method will be implemented in detail in the derived classes.
2. operator==: This is **operator overloading** we learned in the class. This compares two Device objects and return true if they are considered equal, and false otherwise.
    a. Attributes to Compare:
        i. device_id: The unique identifier for the device.
        ii. device_type: The type of device (e.g., "Phone", "Watch", "Ring", "Earbud").
        iii. brand: The brand of the device (e.g., "Apple", "Samsung", "Sony").
    b. Equality Criteria: Two Device objects should be considered equal if **all the following conditions are met**:
        i. The device_id of both devices is the same.
        ii. The device_type of both devices is the same.
        iii. The brand of both devices is the same.

Phone, Watch, Ring, Earbud: The Phone, Watch, Ring, and Earbud classes inherit from the Device class. Each of these derived classes may have additional members specific to the type of device they represent.

1. The print_device() method should be overridden in each derived class to print all relevant information about the specific device type.

Manager: Manager class manages all devices within the system and handles the main functions. The Manager class maintains an array of Device class pointers. This array should be capable of handling up to 100 devices. The Manager class will contain methods to manage the devices in the system, including add_device, compare_device, find_device, delete_device, and print_all_devices. The Manager class needs to include the following functionalities:

1. add_device(): This method adds a device object to the device array. (1) It should be capable of adding any of the device types (Watch, Phone, Ring, or Earbud). (2) It should throw a DuplicateDevice exception (see below) if the device is already in the manager, by using operator "==" defined in the Device class.

2. compare_device(): This method compares whether the device at a specified index in the device array matches the device specified by the given arguments. Use the operator "==" defined in the Device class.
3. find_device(): This method finds the device that matches the given criteria. Use the operator "==" defined in the Device class.
4. delete_device(): This method deletes the device that matches the given criteria. You should be careful about handling available devices in the device array.
5. print_all_devices(): This method prints all information for all devices stored in the array. Use the print_device() method

DuplicateDevice: DuplicateDevice is special class to **handle exceptions** in add_device() in the Manager class. You should use this class to implement the add_device() method.

All classes: Note that all classes have proper implementations of constructors and destructors.

Skeleton Code

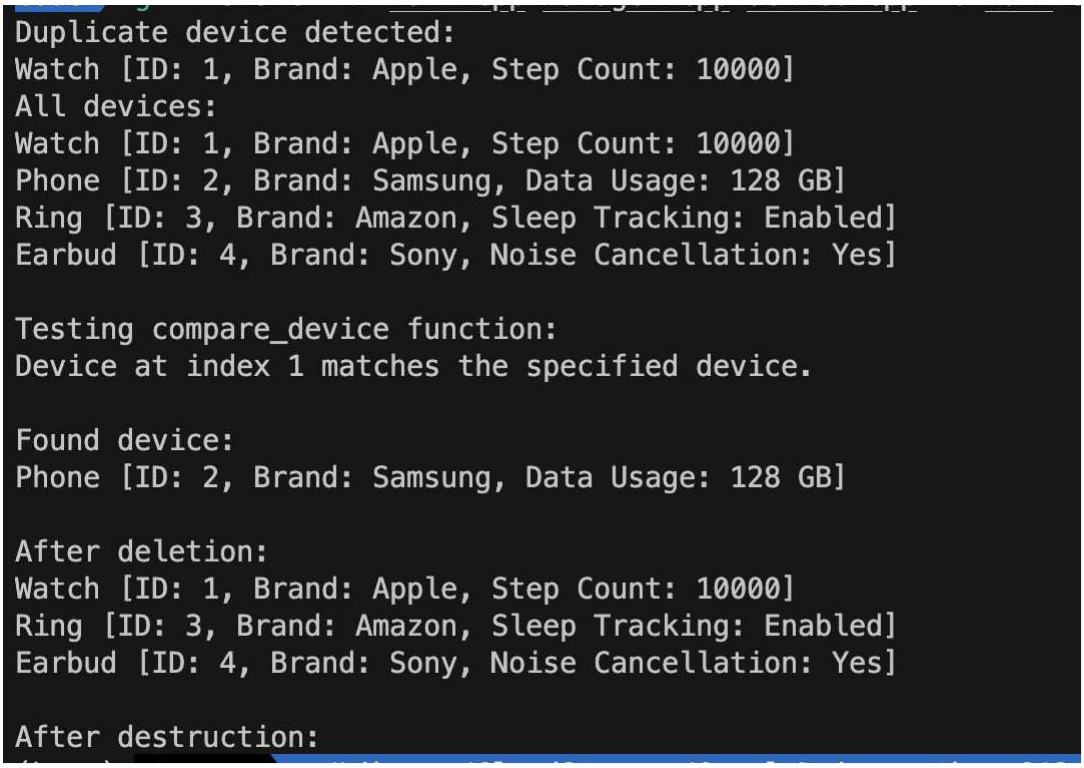
We provide main.cpp, manager.h, manager.cpp, device.cpp, and device.h files for you to start with. **Your job is to complete the implementation of manager.cpp and device.cpp**. **You cannot change manager.h and device.h**. You can test your code with main.cpp we provide, but the code written in main.cpp does not cover all test cases in the actual grading.

Restrictions

- **Allowed Libraries**: You are permitted to use only the following C++ standard libraries: <iostream>, <string>, and <exception>.
- **C++ Version**: It is recommended that you use a recent version of C++ for this assignment (Minimum Version: C++11; Recommended Version: C++17)

Example Output

Below is the console output of running main.cpp with correct implementation of manager.cpp and device.cpp.



```
Duplicate device detected:
Watch [ID: 1, Brand: Apple, Step Count: 10000]
All devices:
Watch [ID: 1, Brand: Apple, Step Count: 10000]
Phone [ID: 2, Brand: Samsung, Data Usage: 128 GB]
Ring [ID: 3, Brand: Amazon, Sleep Tracking: Enabled]
Earbud [ID: 4, Brand: Sony, Noise Cancellation: Yes]

Testing compare_device function:
Device at index 1 matches the specified device.

Found device:
Phone [ID: 2, Brand: Samsung, Data Usage: 128 GB]

After deletion:
Watch [ID: 1, Brand: Apple, Step Count: 10000]
Ring [ID: 3, Brand: Amazon, Sleep Tracking: Enabled]
Earbud [ID: 4, Brand: Sony, Noise Cancellation: Yes]

After destruction:
```

Submission

- **Makefile**: Create your own Makefile to compile your project. This will be part of your grade, so make sure it compiles your project correctly when typing make in the terminal.
- **ReadMe.txt**: Include a Readme file explaining any specific considerations for grading your project or any deviations from the instructions.
- **Format**: Compress all the relevant files (device.cpp, manager.cpp, Makefile, and ReadMe.txt) into a single zip file named STUDENT_ID.zip and submit it.

Grading Criteria

- **Submission format**
- **Compilation**
- **Virtual Function (print_device)**
- **Constructor**

- **Destructor**
- **Operator Overloading**
- **Add Device**
- **Compare Device**
- **Find Device**
- **Delete Device**
- **Exception**
- **Print All Devices**