[2024 Fall] UNIST CSE221 Data Structures
Programming Assignment #2: Smart Device Management System

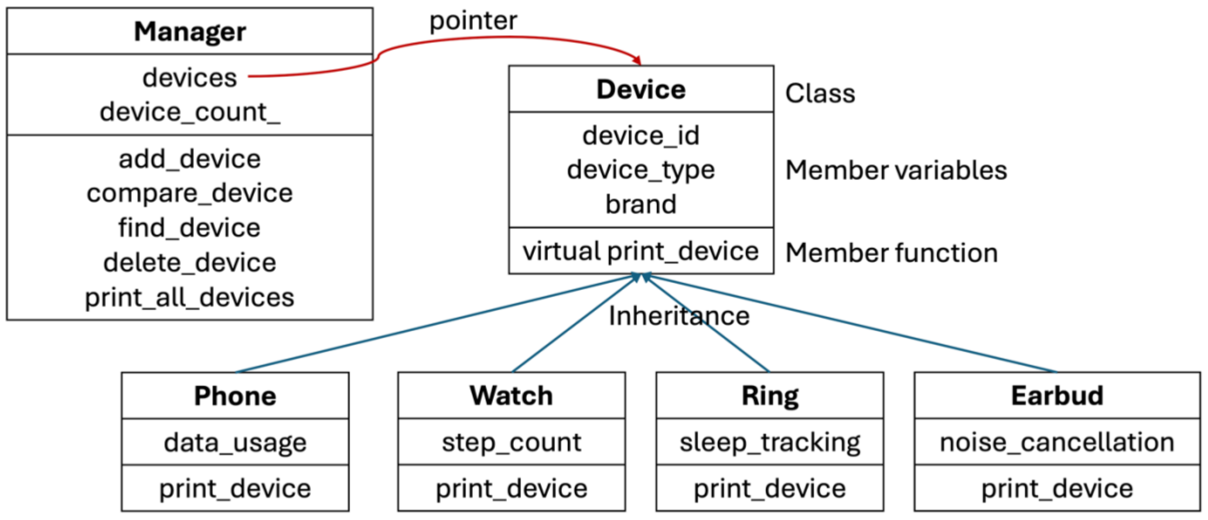
Prof: Taesik Gong (taesik.gong@unist.ac.kr) TA: Minseong Choi (liberty@unist.ac.kr)

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.

In today's world of multiple smart devices, your task is to create a system that efficiently manages different types of devices, providing functionality to add, delete, and search 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.
3. operator<, operator>, operator<= , operator>=: Similar to ‘operator==’, this compares the device_id of two Device objects and return true if they satisfy the specified comparison operator (i.e., >, <, >=, <=), and false otherwise.

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. You must follow the printing output format as shown in the following example outputs for each class.
    a. **Phone**: *Phone [ID: 1, Brand: Samsung, Data Usage: 10 GB]*
    b. **Watch**: *Watch [ID: 1, Brand: Apple, Step Count: 10000]*
    c. **Ring** (sleep_tracking is true): *Ring [ID: 1, Brand: Amazon, Sleep Tracking: Enabled]*
    d. **Ring** (sleep_tracking is false): *Ring [ID: 1, Brand: Amazon, Sleep Tracking: Disabled]*
    e. **Earbud** (noise_cancellation is true): *Earbud [ID: 1, Brand: Sony, Noise Cancellation: Enabled]*
    f. **Earbud** (noise_cancellation is false): *Earbud [ID: 1, Brand: Sony, Noise Cancellation: Disabled]*

2. (Only for **Phone** and **Watch**) operator<, operator>, operator<= , operator>=: This compares their own member variable (e.g., data_usage, step_count) of two objects and return true if they satisfy the specified comparison operator (e.g., >, <, >=, <=), and false otherwise. You should override the operators >,<,>=,<= in the base class. Note that Ring and Earbud simply inherit the operators >,<,>=,<= from their base class.

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 first matched device object from the device array that matches the given criteria. Use the operator "==" defined in the Device class.
4. delete_device(): This method deletes the first matched device object from the device array that matches the given criteria. If multiple matches are found, only the first matching device should be deleted. 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**. We also provide an example **Makefile** for compilation. You can add your own files if necessary, but note that you should explain them in ReadMe.txt and address them in your Makefile. You can test your code with main.cpp we provide, but the code written in main.cpp does not cover all the test cases in grading.

Submission

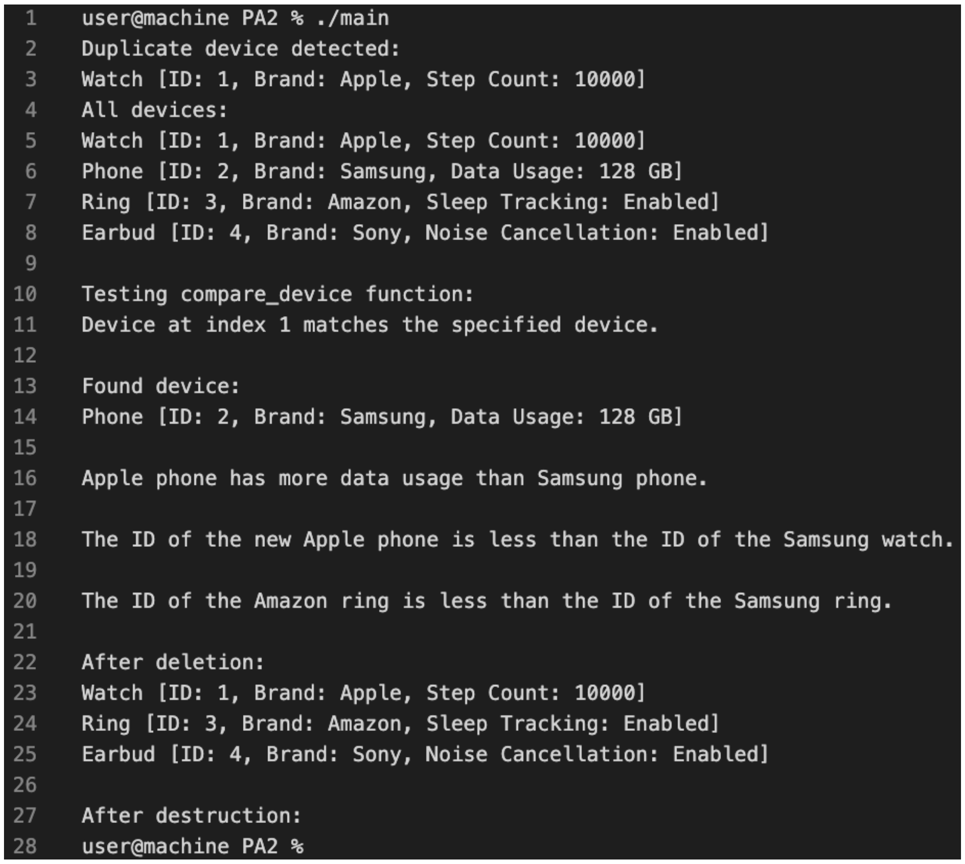
- **Submission Format**: Compress all the relevant files (device.cpp, manager.cpp, Makefile, ReadMe.txt, etc.) into a single **zip** file named **STUDENT_ID.zip** and submit it.
    a. **device.cpp**, **manager.cpp**, and **optional files** that you personally added
    b. **Makefile**: Create your own Makefile to compile your project. You can use the Makefile we provided. This will be part of your grade, so make sure it compiles your project correctly when typing make in the terminal. See the "compilation" section in the grading criteria below for details.
    c. **ReadMe.txt**: Include a Readme file explaining any specific considerations for grading your project or any deviations from the instructions.
- **Deadline: 27th September 11:59PM**
    a. **Late Submission**: The late submission follows the policy that was explained in the course overview lecture (see lecture notes in Blackboard).

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 or higher**.

Example Output

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



```
user@machine PA2 % ./main
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: Enabled]

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

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

Apple phone has more data usage than Samsung phone.

The ID of the new Apple phone is less than the ID of the Samsung watch.

The ID of the Amazon ring is less than the ID of the Samsung ring.

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: Enabled]

After destruction:
user@machine PA2 %
```

Grading Criteria

- **Submission format**
  - It is mandatory to ensure the integrity of header files (i.e., device.h and manager.h). Your submission should only include device.cpp and manager.cpp files (and any other .h and .cpp files you personally added) for the source code.
- **Compilation**
  - The "**make**" command MUST work to create the "**main**" program (see the example Makefile provided).
  - While you only need to submit part of the source files, assume that the provided *.h files and main.cpp will be included during compilation, so you don't need to worry about compilation errors due to missing those files.
  - If compilation fails, the score will be zero.
  - If a runtime exception halts the program (e.g., a crash due to a segmentation fault), the score will be based on progress up to the point of termination.
  - Scoring will be based on each correctly printed element of each class.

- **Constructors**
  - Scoring will be based on whether each class's constructor correctly stores data.
- **Destructors**
  - Scoring will be based on whether objects are properly deleted after invoking the destructor (includes memory release).
- **Operator Overloading**
  - Scoring will be based on whether size comparison (i.e., '>', '<', '>=', and '<=') and equality (i.e., '==') operations are processed correctly.
- **Add Device**
  - Scoring will be based on whether an object of each class is added.
  - Scoring will be based on adding objects up to the maximum number.
- **Compare Device**
  - The same criteria as for Operator Overloading.
- **Find Device**
  - Scoring will be based on correctly finding an object of the same class type.
- **Delete Device**
  - Scoring will be based on whether the Device object is released and no longer accessible from the Manager object, and releasing the actual memory is also completed.
- **Print Device / Print All Devices**
  - Test if the objects are printed in the order they were entered.
  - Follow the printing output format (case insensitive, but underscore ('_') or other unnecessary characters may lead to misjudgment and scores will proceed as graded).
  - Scoring will be based on whether the printed output shows correctly even if multiple additions and deletions are performed.
- **Exception**
  - Scoring will be based on whether an exception is thrown when it is Duplicated.