OOP Final

> This final was created by 黃漢軒（109590031）, please feel free to ask any problem during the exam.

Folder Structure Tree

- You should finish the unit test written by you.
- You can split the unit test into multiple files, just remember to include all of it into `ut_main.cpp` (see course repo).

While your project has been built by `makefile`, the structure tree should be the same as the following section.

```
bin/
├─ ut_all
src/
├─ bike_factory.h
├─ bike_rms.h
├─ bike.h
├─ electric_bike.h
├─ mybike.h
test/
├─ <some test file>
├─ ut_main.cpp
makefile
```

Problem Content

It's time to say goodbye to Uriah, hope you have fun in the pervious homework and already improve the OOP concept to beat the final. GL&HF.

In the Iepiat City, there are two type of bike available to borrow:

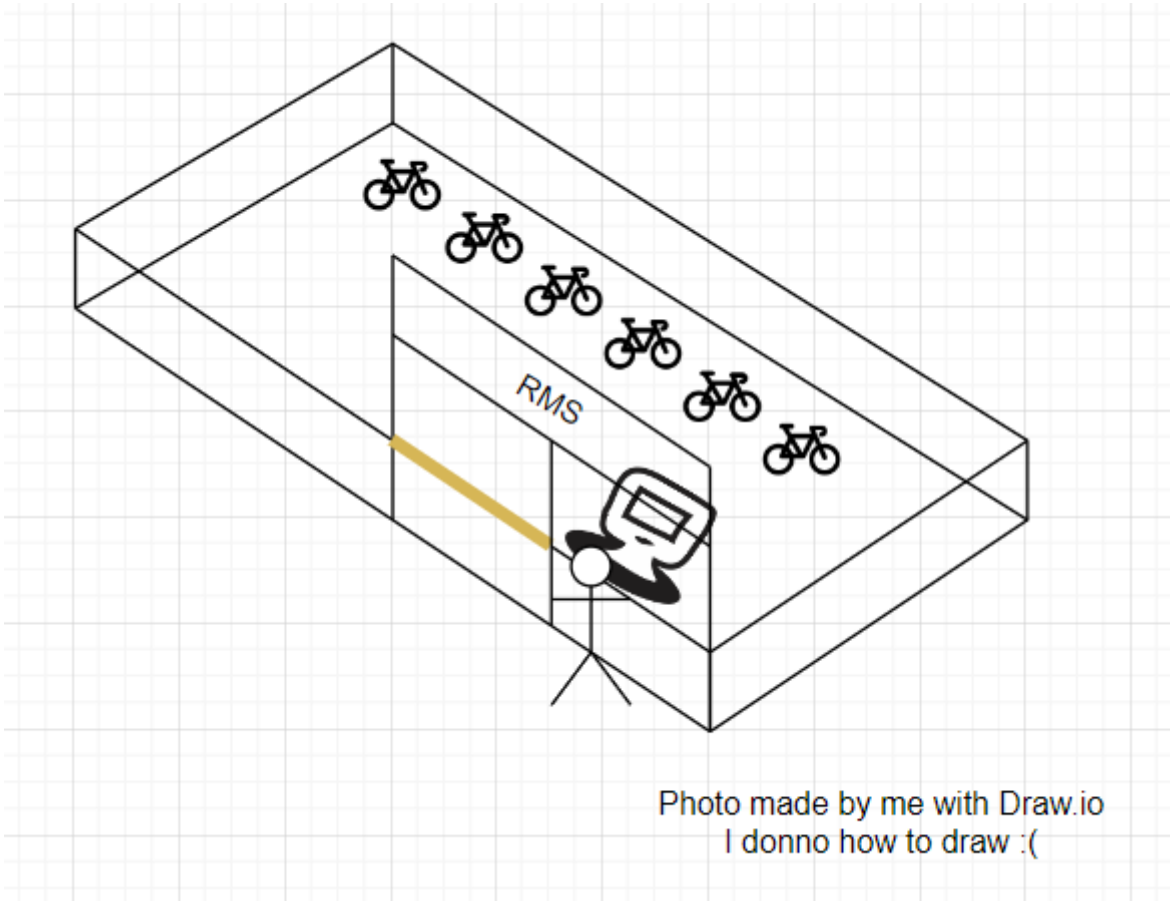
- Electric Bike: The bike with electric support, the electric bike have `ID`, `power` and `rental_price`.
- My-Bike: The normal bike, the bike have `ID`, and `rental_price`.

Hairu is going to make a bike rental management system (bikeRMS).

There will be multiple bikeRMS, and each bikeRMS have a yard to park the bike.

The system can rent the bike to people, and record the available bike.

The income of all bikeRMS should be calculated as a total value.



Since Hairu may have a lot of bikeRMS, Hairu want to have a program to check all bikes in bikeRMS, so you should help her record it.

Hairu want to make a bike factory also, the bike factory can product the bike or the bike array with specific type.

In this task, you should make 5 class: `ElectricBike`, `MyBike`, `Bike`, `BikeRMS` and `BikeFactory`

- The `Bike` can get the `ID` and the `rent_price`.
- The `ElectricBike` and `MyBike` will inheritance the `Bike` class.
- The `ElectricBike` can set the `power` and get the `power`.
- The `BikeRMS` can record available bikes and manage the bikes.
- The `BikeRMS` can record the total income. (Use static variable to record it!)
- The `BikeFactory` can product the specific type of bikes.

Task

In this task, you should make 5 class: `ElectricBike`, `MyBike`, `Bike`, `BikeRMS` and `BikeFactory`

- class `Bike`
  - `Bike(int ID, int rent_price)`
  - The parameterized constructor to initialize bike.
    - Throw `std::invalid_arguments` if `rent_price` is negative.
  - `int get_id()`
    - Return the ID of the bike.

    - `int get_rental_price()`
        - Return the rental price of the bike.
    - `virtual std::string to_string() = 0`
        - Return the format string to describe bike.

- class `ElectricBike`, inheritance `Bike`
    - `ElectricBike(int ID, int rent_price, double power = 0)`
        - The parameterized constructor to initialize electric bike.
        - Throw `std::invalid_arguments` if the power is not in range $[0, 1]$ or the `rent_price` is negative.
    - `int get_id()`
        - Return the ID of the electric bike.
    - `int get_rental_price()`
        - Return the rental price of the electric bike.
    - `double get_power()`
        - Return the power of the electric bike.
    - `void set_power(double power)`
        - Set the power to the specific value.
        - Throw `std::invalid_arguments` if the power is not in range.
    - `std::string to_string()`
        - Return the format string to describe bike.
        - If we have a electric bike with ID `15`, price `30` and power `0.3`, the `to_string` function should return the string by the format below:

          ```
          15-ElectricBike price=30 power=0.30
          ```

          Notice that you should output power round off to the 2nd decimal place.
        - The format string is `%d-ElectricBike price=%d power=%.2f`

- class `MyBike`, inheritance `Bike`
    - `MyBike(int ID, int rent_price)`
        - The parameterized constructor to initialize MyBike.
        - Throw `std::invalid_arguments` if price is negative.
    - `int get_id()`
        - Return the ID of the MyBike.
    - `int get_rental_price()`
        - Return the rental price of the MyBike.
    - `std::string to_string()`
        - Return the format string to describe MyBike.
        - If we have a electric bike with ID `15`, price `30` and power `0.3`, the `to_string` function should return the string by the format below:

          ```
          15-MyBike price=30
          ```

        - The format string is `%d-MyBike price=%d`

- class `BikeRMS`
    - `BikeRMS(std::vector<Bike*> bikes)`
        - The parameterized constructor to initialize BikeRMS.
    - `int get_count_of_available_bike()`
        - Return the count of available bike.
    - `Bike* rent_bike(int ID)`
        - Rent a bike with specific ID from bikeRMS.
        - Throw `std::invalid_arguments` if bike of specific ID is unavailable or not exist in bikeRMS.

          i.e. Assume our bikeRMS have 3 bikes with ID `{1, 2, 3}`, if 3 already rented by other, then `rent_bike(3)` should throw the exception.

          If someone want to rent bike with ID 4, because bike with ID 4 not in the bikeRMS, so you should throw the exception also.
    - `void return_bike(Bike* bike)`
        - Return the bike to bikeRMS.
        - Throw `std::invalid_arguments` if bike of specific ID is not belong or already exists in bikeRMS.

          i.e. Assume our bikeRMS have 3 bikes with ID `{1, 2, 3}`, and someone restore bike with ID 4,

          you should throw the exception since bike with ID 4 is not belong of or bikeRMS.

          If the bike with ID 3 already in the bikeRMS and someone return the bike with ID 3, you should throw the exception also.
    - `void append_bike(Bike* bike)`
        - Add a new bike to bikeRMS.
        - Throw `std::invalid_arguments` if bike of specific ID is already exists in bikeRMS.

          i.e. Assume our bikeRMS have 3 bikes with ID `{1, 2, 3}`, and someone append bike with ID 3,

          you should throw the exception since bike with ID 3 is already exists in bikeRMS.
    - `static int get_total_rental_income()`
        - Return the total of the rental income in all of the bikeRMS.

          i.e. Assume we have bikeRMS1 have 2 bike with ID and price `{1: 150, 2: 130}` and bike RMS2 have 2 bike with ID and price `{3: 150, 4: 190}`, if the bike 1 and bike 3 are rented, the `get_total_rental_income()` should return 300.
    - `static void reset_total_rental_income()`
        - Reset the record of the rental income to zero.

- class `BikeFactory<T>`, with template to specific type of bike.
    - `static T* create_bike(int ID, int rent_price)`
        - Create bike with specific `ID` and `price`.

- `static std::vector<T*> create_bike_array(std::vector<int> IDs, std::vector<int> rent_prices)`
  - Create specific bike array with specific `IDs` and specific `rent_prices`.

Some important notice:

- You should delete the default constructor and write the destructor.
- You can assume it won't happen two different bikeRMS have the same ID of bike.

Score

The score will calculate by the percentage of passed test in each test suite.

In this exam, we have 6 tests suite.

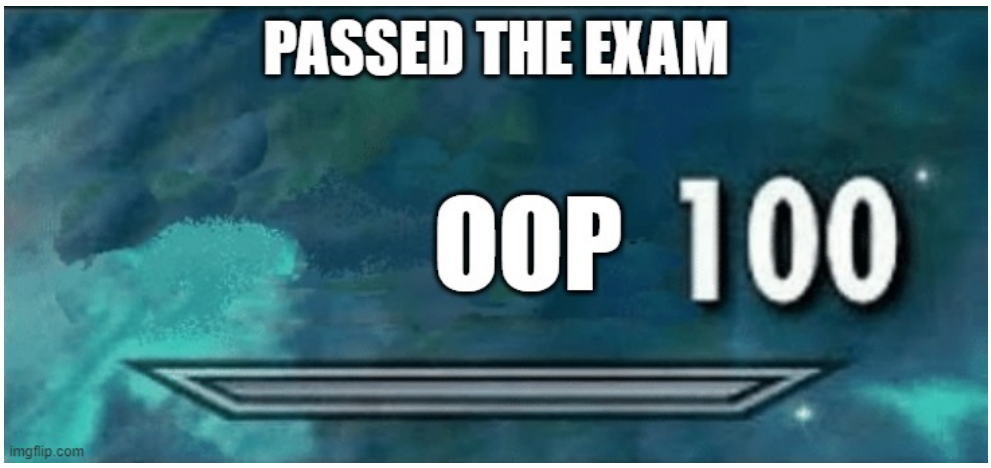
- \[0%]\ SampleTestWithFixture
- \[15%]\ BikeTest (or named TypeTraitsTest)
- \[15%]\ ElectricBikeTest
- \[15%]\ MyBikeTest
- \[15%]\ BikeFactoryTest
- \[40%]\ BikeRMSTest

Notice

- Use nullptr if you want to have a null pointer, which is a special pointer that doesn't point to anything.
- Use `ASSERT_EQ` to test integers, `ASSERT_NEAR` to test floating-point numbers, and `ASSERT_THROW` to test exceptions.
- You should neither add a bin folder to your git nor add a file with the name '.gitignore' in the bin folder (see our class repo).
- In some situations you will lose score:
  - **You lose 5 points for each test that has a memory leaks. You can check memory leak with `valgrind` cmd.**

    ```
    valgrind --track-origins=yes --leak-check=all <executable_file>
    ```
  - **You will lose 10% if your bin folder contains compiled ut_all in the git repo.**

Meme


PASSED THE EXAM
OOP 100
imgflip.com