COMP 2012 Object-Oriented Programming and Data Structures

Assignment 2 Harvest Moon

Introduction

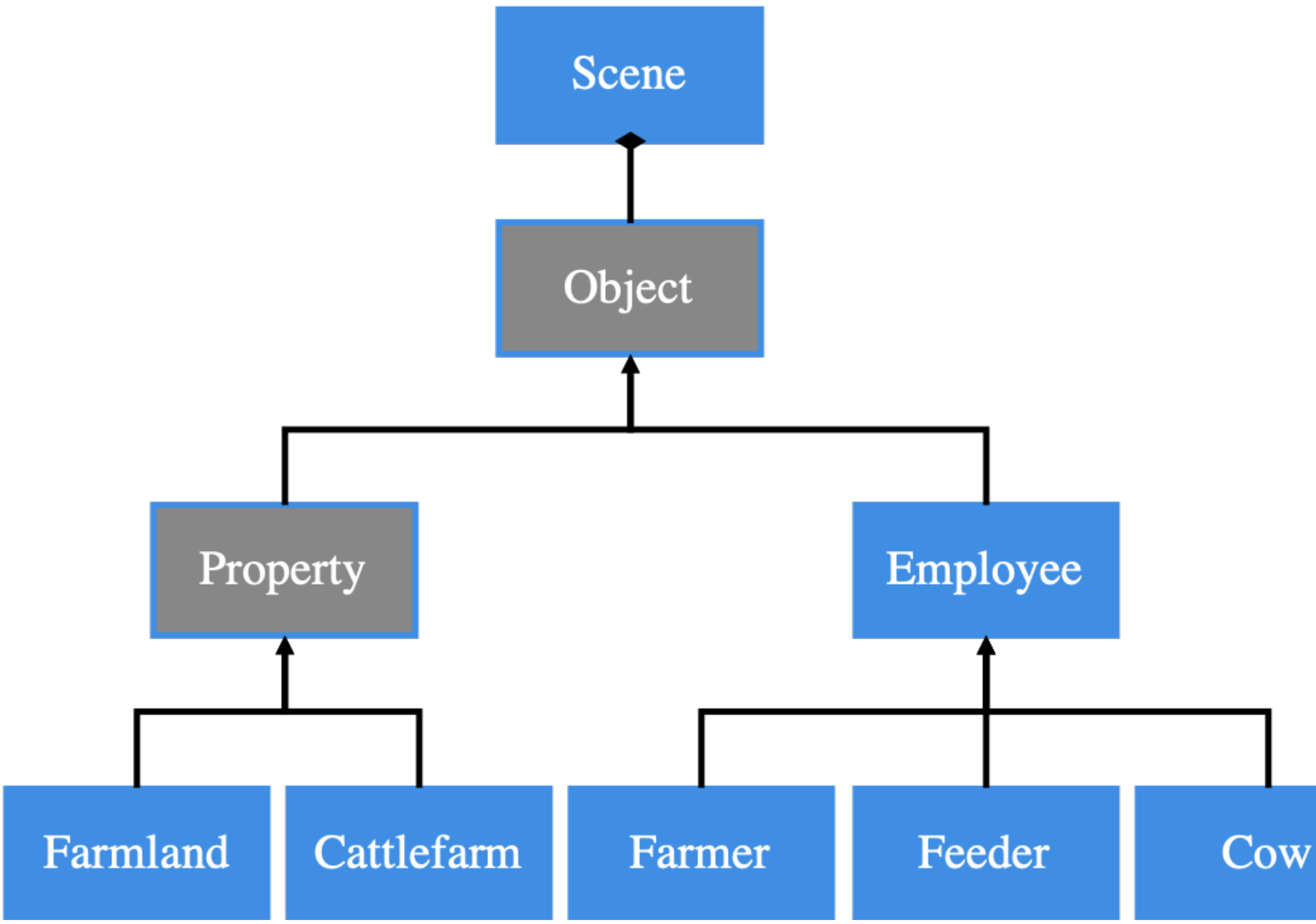
In this programming assignment, you are going to familiarize yourself with the concepts of inheritance and polymorphism the same time, you will have the opportunity to work with the ncurse library.

Before you work on this assignment, **read the description part carefully** and make sure you understand the connection between all classes.

We value academic integrity very highly. Please read the Honor Code section on our course web page to make sure you understand what is considered as plagiarism and what the penalties are. The following are some of the highlights:

- Do NOT try your "luck" - we use some sophisticated plagiarism detection software to find cheaters. It is much bett most students think. It has been proven times and times again tricks didn't work. We also review codes for potentia manually.
- The penalty (for **BOTH** the copier and the copiee) is not just getting a zero in your assignment - it is much more th It is simply not worth to cheat at all. You would hurt your friend and yourself by doing that. It is obvious that a real f won't ask you to get involved in a plagiarism act in any way due to the consequences. Read the Honor Code again you even try to think about cheating.
- Serious offenders will fail the course immediately and there may be additional disciplinary actions from the departr and university.

Description



You must finish this assignment with the provided files. Download them in the Download section.

Your task is to complete all the missing function implementations in all `Todo*.cpp` and `Todo*.h` according to the instruct below.

About the Game

Harvest Moon is a turn-based game that you act as a manager to run a farm. With a smart investment strategy, you may properties and employ staff to make money. You may have a risk of bankruptcy due to bad management. Various emplo are capable of making money in different types of properties, which may give you different rewards at the end of each t Both properties and employees are modeled by an abstract base class `Object`.

At each turn, the player can buy and build some properties (`Farmland`, `Cattlefarm`) on the screen. Different types of properties cost different amount of money to build. He can buy as many properties as he wishes at each turn as long as enough money. The properties have to be built on the screen without overlapping with each other. The player can also e staff (`Farmer`, `Feeder`, even `Cow`) to make money in properties for him. Please note that he can assign an employee to a t property only if the property is suitable for that kind of employee. He may also upgrade properties he owns to make mor money or to accommodate more employees.

Different Types of Properties

1. `Property`: This is an abstract base class of property.

2. `Farmland`: This models a farmland in the game (represented with 'R' on the screen; 'R' for Rice). It accommodates `Employee` and `Farmer`, and returns a random reward in the range [0, `number of employees in work x 2 + numbe farmers in work x 6 + level x 3`) at the end of each turn. In this formula, `number of employees` counts only th number of `Employee` objects. For instance, if the 1-level farmland owns 2 `Employee` objects in `WORK` and 1 `Farmer` ob `WORK`, the formula should be [0, `2 x 2 + 1 x 6 + 1 x 3`).

3. `Cattlefarm`: This models a cattle farm in the game (represented with 'C' on the screen; 'C' for Cow). It accommoda `Feeder` and `Cow`. At the end of each turn, it returns a reward equal to `min(number of cows, number of feeders i work) x level x 10`. It also checks the state of each cow, and removes those cows that have died.

   `void Cattlefarm::removeDiedCow()`: It completely removes all the cows it owned if they are not alive.

All properties can be constructed by a constructor (int, int) where the first int refers to the x-coordinate of the property second int refers to the y-coordinate of the property. The specifications of each property, like cost, size, and range of th maximum number of employees are written inside the cpp file of different properties. The `ObjectState` of properties is a `NORMAL`.

You may find the members and their details in `Property.h` before implementing your property-related classes.

Different Types of Employees

1. `Employee`: This is a basic type of employees (represented with "e" on the screen; 'e' for employee). He can handle jobs from `Farmland`. He costs $5 and has a daily salary of $1 with the employment terms of 1-day work and 1-day r

2. `Farmer`: This models a farmer (represented with "r" on the screen; 'r' for rice). He can handle only jobs from `Farmla` costs $10 and has a daily salary of $2 with the employment terms of 3-day work and 1-day rest.

3. `Feeder`: This models a feeder (represented with "d" on the screen; 'd' for dairy). He can handle only jobs from `Cattlefarm`. He costs $20 and has a daily salary of $5 with the employment terms of 6-day work and 1-day rest.

4. `Cow`: This models a cow (represented with "c" on the screen; 'c' for cow). It can handle only jobs from `Cattlefarm`. $15, has no salary but works everyday. It also has an attribute `m_lifespan` and will die after working for more days

`m_lifespan`.

`m_lifespan`: private member of `Cow` with the type **const int**.

`int Cow::getLifespan() const`: It returns the `m_lifespan`.

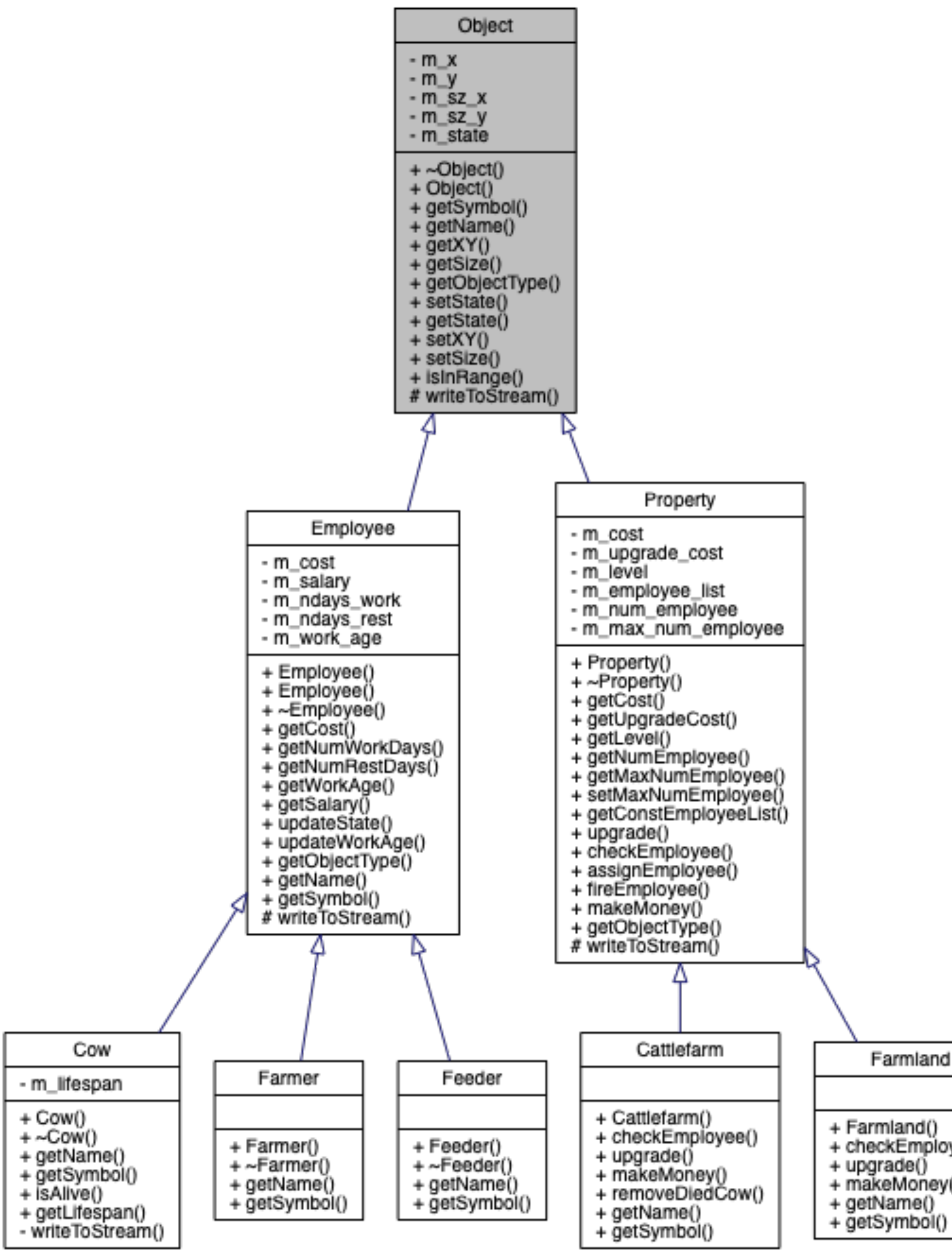
`bool Cow::isAlive() const`: It returns True if its workage < lifespan.

Each employee occupies a 1x1 space on the map. The cost, salary, number of work/rest days are written inside the cpp f each Todo-Employee. Once constructed, the states of all types of employees are set as `NORMAL`. After being assigned to property, its state will be changed between `WORK` (first) and `REST`. All kinds of employees get salaries in both `WORK` and `RE` states. Its position is set once he is assigned.

You may find the members and their details in `Employee.h` before implementing your employee-related classes.

Class inheritance graph

The detailed class inheritance graph is here for your reference.

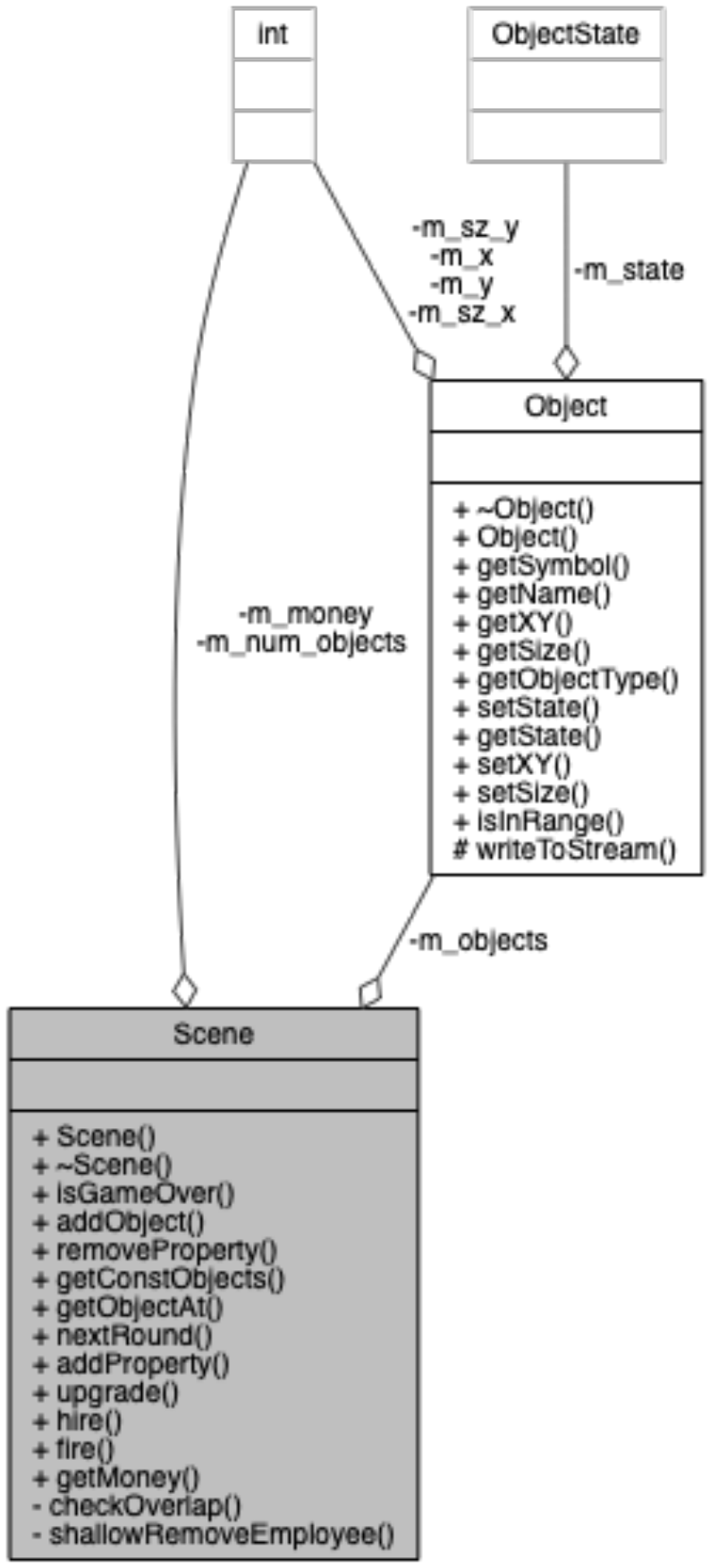


Scene

`Scene` is the object managing the basic game operations. It logs all the `Object` (both `Property` and `Employee`) on the scr
You may find the members and their details in `Scene.h` before implementing the member functions of `Scene`.

Collaboration diagram between `Scene` and `Object`

The detailed class inheritance graph is here for your reference.



Some hints to help you finish your PA2

- Please read the comments carefully. We give details and some examples in the comments of the header files, especia

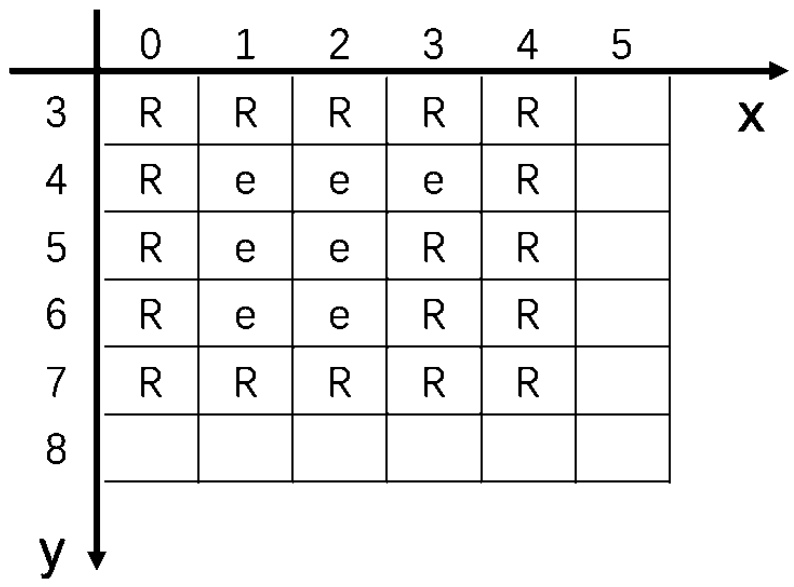
Property.h, **Employee.h**, **Scene.h**. The comments help you understand the input and output of each function.

- Take care of the memory leak problem. It is noted that the pointer of each `Employee` might be owned by both its assig

`Property` and the `Scene`. When you delete it, please consider whether you should only delete the pointer or the object.

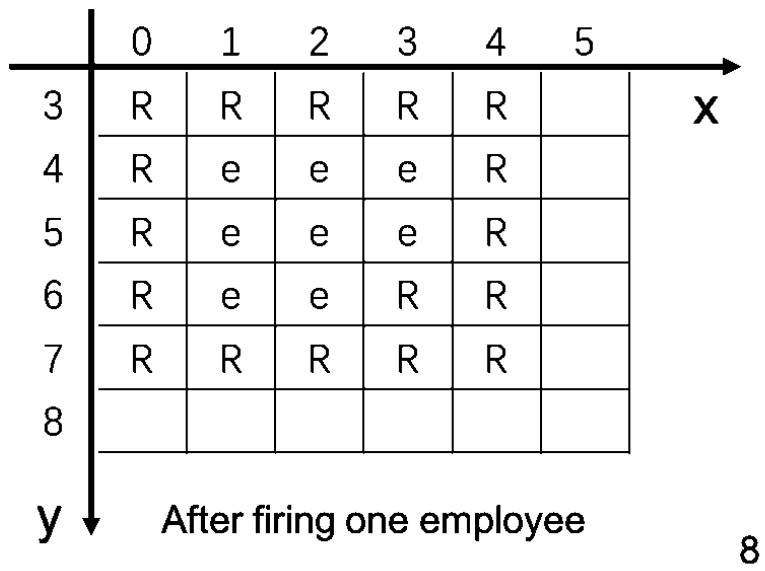
- You can use the `getRandInt()` function to generate random numbers when you implement your `TodoFarmland.cpp`
- After assigning an employee to a property, it will set the (X, Y) position of the employee in terms of the following order

example, if we build a `Farmland` (Size 5 x 5) at (0, 3), the first assigned employee should be set at (1, 4), and the second assigned employee should be set as (1, 5)... In the following figure, the 'R's represent the `Farmland`, and the 'e's represe `Employee`.



- After an employee is fired (completely deleted from its property). The property object may need to rearrange the (X, Y position of all its remaining employees. You have to ensure all the assigned employees stay inside their property. For exa

if we build a `Farmland` (Size 5 x 5) at (0, 3) and have assigned 9 `Employee`, now we want to fire an `Employee` and hire a ne `Farmer`. You have to make sure all the assigned `Employee` and the `Farmer` are inside the `Farmland`. The following animatio gives you two correct placements (inside) and one wrong placement (outside) of such case. Of course, there are more c and incorrect cases. The animation is to help you understand what we mean by "inside" and "outside" of a property.



8

- You can use the `test.cpp` to test your code. It provides some basic test cases. Passing the `pa2_test.exe` does not guarantee the correctness of your code. But to make your game work well, your code should at least pass such a basic t You can feel free to write your own test cases to test your code in a similar way as `test.cpp`.

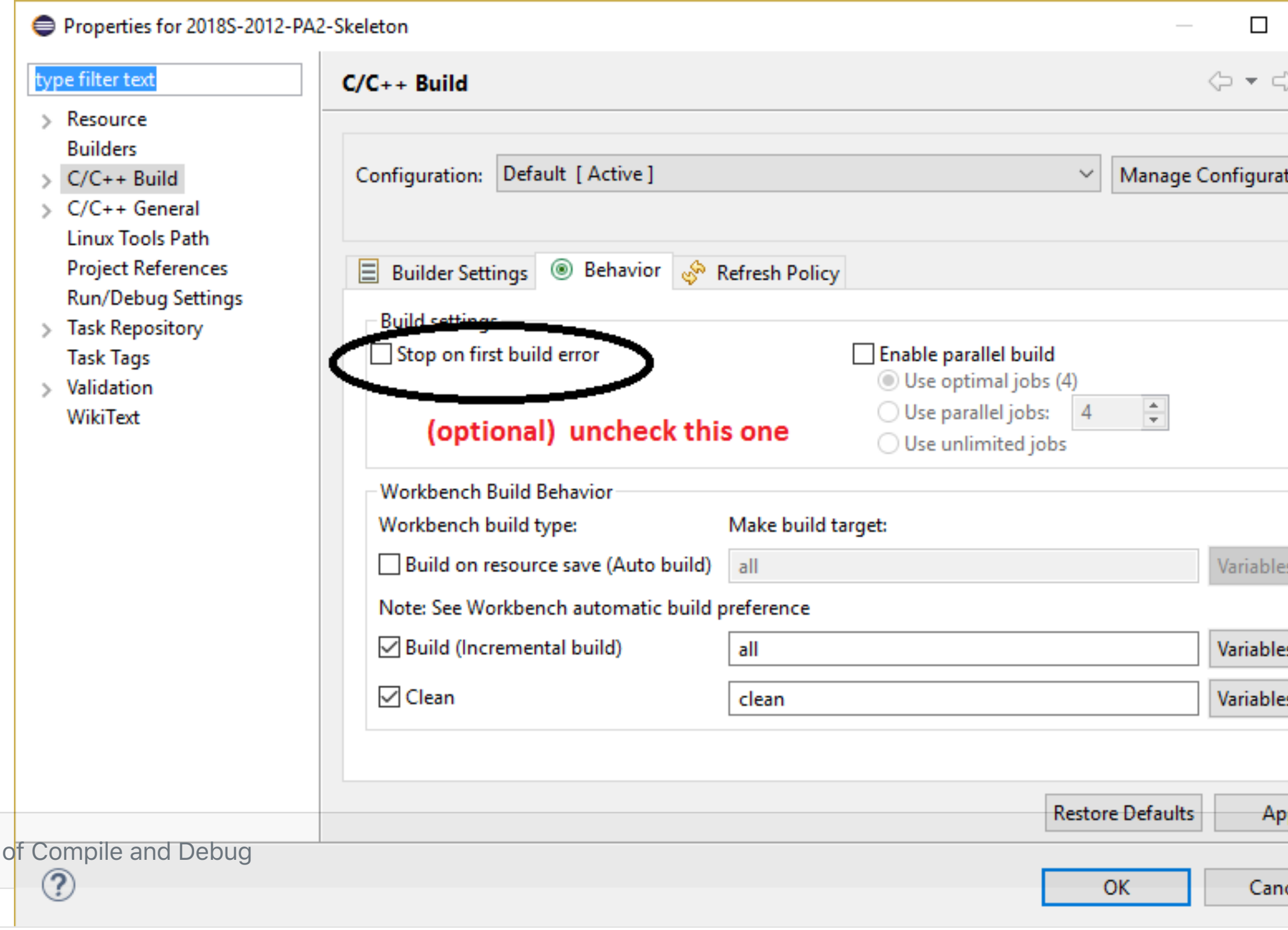
End of Description

Compilation and Debug

- Please read the honor code and strictly follow it.
- You need to implement with a Makefile project. The Makefile is given in the skeleton.
- You cannot use the Run button in Eclipse to run the program. Instead, you need to open the folder that contains th executable file and double click on it. For Mac users, you are suggested to build and run on a terminal by

```
> make
> ./pa2_test.exe
> ./pa2.exe
```

- You should start with the given skeleton code.
- Read the "Specification Clarification" and "Reported Bugs" section for some common clarifications. You should ch that a day before the deadline to make sure you don't miss any clarification, even if you have already submitted yo then.
- We will be grading your code on a Linux machine. Filenames are case sensitive in Linux. Therefore, when you inclu `Property.h` for example, you need to type `#include "Property.h"` but not `"property.h"`, although it would also c on Windows.
- You are not allowed to use STL, global variables, static variables, or any additional libraries. You can include **typein** you wish to use some RTTI.
- You need to clean up your memory properly.
- You are not required to do any cin or cout inside your required function. It is likely you have committed some mista you do so.
- Optionally you may want to build the project partially before you have completed the assignment. In Eclipse, you n turn off the "Stop on first build error" from the project properties. On Linux/Mac command prompt, you can type "m k" for "keep going".



End of Compile and Debug

Download

[Download Skeleton Code]

[Download Demo (Windows)], [Download TestDemo (Windows)]

[Download Demo (Mac)], [Download TestDemo (Mac)]

[Download Demo (Linux)], [Download TestDemo (Linux)]

End of Download

Memory Leak

Same to PA1, you can use [Dr.Memory] or [Valgrind] to check memory leak.
Since some memory check softwares like `Valgrind` may provide false positive results when dealing with ncurses library. can check memory leak with `pa2_test.exe` instead, which does not dependent on ncurses library.

End of Memory Leak

Grading

1. If your program can be compiled, you will receive 10%.
2. Each of the Property class carries 5%, total = 15%
3. Each of the Employee class carries 5%, total = 20%
4. Correct implementation of the class Scene = 25%
5. No crash bonus. If your program does not crash during the grading, you receive 15%. Each crash deducts 5% from 15%.
6. No leak bonus. If no memory leak is detected in your code, you receive 15%.

End of Grading

Submission and Deadline

Deadline: 23:59:00 on 25 April 2020 (Saturday)

Canvas Submission

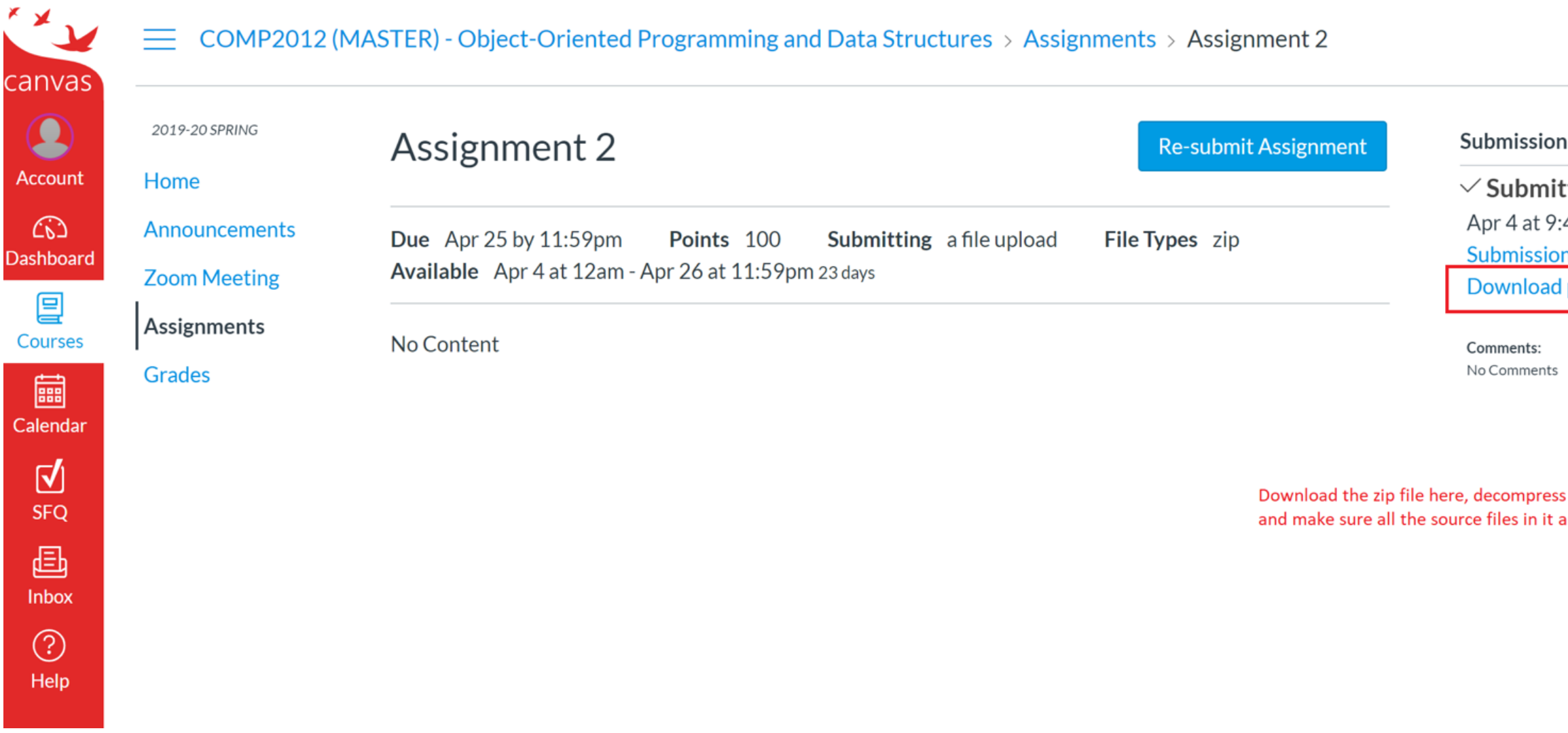
Create a single zip file that contains only the **Todo*.h and Todo*.cpp files** and name it `pa2.zip` . The makefile would in f create a zip file called pa2.zip for you when you build the project. If you create your pa2.zip with makefile, please double its content. You should not submit any file that does not begin with Todo and you can't modify it in working out your solu we will use our original files to compile with your submitted cpp file to generate the executable for your submission for g Finally, you should submit only the **pa2.zip** through the **Canvas Submission Page**:

- Submission Page for L1
- Submission Page for L2
- Submission Page for L3
- Submission Page for L4

The filenames have to be exactly the same. It must be a zip file, not rar, not 7z, not tar, not gz, etc. Inside the zip file, t files need to have correct names. If your submissions cannot be uncompressed, they will not be graded.

Make sure your source files can be successfully compiled.If we cannot even compile your source file, your work will not graded, and you will get zero mark for your PA2. Therefore, you should at least put in dummy implementations to the pa you cannot finish so that there will be no compilation error.

Make sure you actually upload the correct version of your source files - we only grade what you upload. Some stu in the past submitted an empty file or a wrong file or an exe file which is worth zero mark. So **you must download and c check the file you have submitted**. You may refer to the illustration below.



You may submit your file multiple times, but only the last version will be graded.

Submit early to avoid any last-minute problem. Only canvas submissions will be accepted.

Note 1: If you have no idea how to create a zip file, you may see How to create a zip file in Windows 10 or How to create file in Mac OS X.

Note 2: Canvas may append a number to the filename of the file you have submitted. e.g. pa2-1.zip. It is OK as long as y have named your file as pa2.zip when you submit it.

Using Virtual Barn

It is **required** that your submissions can be compiled and run successfully in our official Windows Eclipse which can be downloaded from the "Using Eclipse at home (Windows)" section here. You need to unzip the zip file first, and then run Eclipse program extracted from it. Do not just double-click the zip file. If you have used other IDE/compiler/OS (including macOS Eclipse) to work out your solution, you should test your program in the aforementioned official environment befo submission. This version of Eclipse is also installed on our lab machines.

If you have no access to a standard Windows machine, you may remote control a Windows machine in HKUST virtual ba Choose the "Programming Software" server, and you will find Eclipse shortcut on the desktop. This is a newer version of Eclipse with a slightly different interface. However, its compiler has the same version as ours and can be used to verify i program can be compiled by us for grading. In particular, to create a new C++ project, you can do so from "File" menu - "New" -> "Project..." -> "C++ project" -> Type a project name -> Choose MinGW compiler -> "Finish". Also, to build the project, save your files, then use the Hammer and Run buttons, circled in the following screenshot (NOT the ones on the

End of Submission and Deadline

Late submission policy

There will be a penalty of -1 point (out of a maximum 100 points) for every minute you are late. For instance, since the d of assignment 2 is 23:59:00 on 25 April 2020, if you submit your solution at 1:00:00 on 26 April 2020, there will be a pe -61 points for your assignment. However, the lowest grade you may get from an assignment is zero: any negative score the deduction due to late penalty (and any other penalties other than plagiarism penalty) will be reset to zero.

FAQ

Frequently Asked Questions

Specification Clarification

- void Scene::removeProperty(Property*): It completely removes the property in the scene. All the employees owned this property should also be completely removed. Please also remember to update the `m_objects` and `m_num_objects` accordingly.
- You should implement your code after the banner "// TODO: Start to implement your code". Otherwise, the code lines you implemented before the banner will be neglected by the auto grader.
- For the following functions, you should return False if the argument is nullptr.

```
bool checkEmployee(Employee*) const;

bool Property::assignEmployee(Employee*);

bool Property::fireEmployee(Employee*);

bool Scene::upgrade(Property*);

bool Scene::fire(Employee*);
```

- You are allowed to use `const_cast` in this assignment.

Reported Bugs

Not Yet.

End of FAQ

Menu

Page maintained by

Peng YUN

Email: pyun@cse.ust.hk
Last Modified: 04/11/2020 19:03:2304/07/2020 16:56:38

Homepage

Course Homepage