Object-Oriented Software Engineering hw5

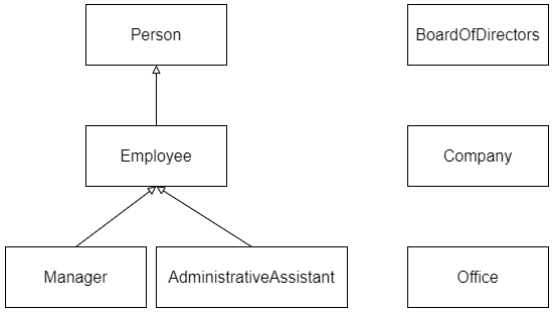
- Author: 黃柏瑄 (P78081528)

Environment

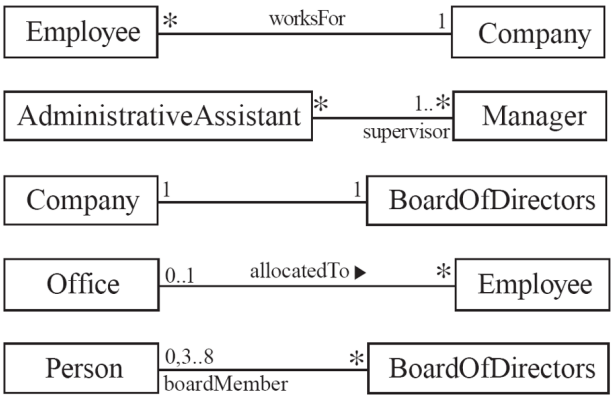
- OS: Ubuntu18.04.5 (WSL2)
- C++ compiler: g++ (Ubuntu 8.4.0-1ubuntu1~18.04) 8.4.0

Source code

Simple UML



5 relation



File architecture

```
$ tree . -I 'bin|*.md|Makefile|img'
.
├── administrative_assistant.h
├── board_of_directors.h
├── common.h
├── company.h
├── employee.h
├── manager.h
├── office.h
├── person.h
├── relation1.cc
├── relation2.cc
├── relation3.cc
├── relation4.cc
└── relation5.cc
```

- File `common.h`

```
#ifndef COMMON_H
#define COMMON_H
```

```
3
4  enum class Position {
5    Chairman,
6    Director,
7    Manager,
8    ProductManager,
9    SeniorEngineer,
10   JuniorEngineer,
11   AdministrativeAssistant,
12 };
13
14 enum class Gender : unsigned char {
15   M,  // male
16   F,  // female
17   O,  // other
18 };
19
20 #endif /* COMMON_H */
```

- File `person.h`

```
1  #ifndef PERSON_H
2  #define PERSON_H
3
4  #include <string>
5
6  #include "common.h"
7
8  class Person {
9   public:
10   Person(std::string name, Gender gender) : name_{name}, gender_{gender} {}
11   std::string get_name() const { return name_; }
12   Gender get_gender() const { return gender_; }
13   virtual ~Person() = default;
14
15  protected:
16   std::string name_;
17   Gender gender_;
18 };
19
20 #endif /* PERSON_H */
```

- File `employee.h`

```
1  #ifndef EMPLOYEE_H
2  #define EMPLOYEE_H
3
4  #include <string>
5
6  #include "common.h"
7  #include "person.h"
8
9  class Company;
10 class Office;
11
12 class Employee : public Person {
13  public:
14   Employee(std::string name, Gender gender, int id, Position position)
15       : Person{name, gender}, id_{id}, position_{position} {}
16   int get_id() const { return id_; }
17   Position get_position() const { return position_; }
18   void set_workfor(Company *c) { workfor_ = c; }
19   const Company *get_workfor() const { return workfor_; }
20   void set_office(Office *o) { office = o; }
21   const Office *get_office() const { return office; }
22   virtual ~Employee() = default;
23
```

```
 protected:
  int id_;
  Position position_;
  const Company *workfor_{nullptr};
  const Office *office{nullptr};
};

#endif /* EMPLOYEE_H */
```

- File `manager.h`

```
#ifndef MANAGER_H
#define MANAGER_H

#include <string>
#include <vector>

#include "common.h"
#include "employee.h"

class AdministrativeAssistant;

class Manager : public Employee {
 public:
  Manager(std::string name, Gender gender, int id)
      : Employee{name, gender, id, Position::Manager} {}
  void AddSubordinate(AdministrativeAssistant* a) {
    subordinates_.emplace_back(a);
  }
  std::vector<const AdministrativeAssistant*> get_subordinates() const {
    return subordinates_;
  }

 private:
  std::vector<const AdministrativeAssistant*> subordinates_{};
};

#endif /* MANAGER_H */
```

- File `administrative_assistant.h`

```
#ifndef ADMINISTRATIVE_ASSISTANT_H
#define ADMINISTRATIVE_ASSISTANT_H

#include <string>
#include <vector>

#include "common.h"
#include "employee.h"
#include "manager.h"

class Manager;

class AdministrativeAssistant : public Employee {
 public:
  AdministrativeAssistant(std::string name, Gender gender, int id,
                          Manager* supervisor)
      : Employee{name, gender, id, Position::AdministrativeAssistant},
        supervisors_{supervisor} {
    supervisor->AddSubordinate(this);
  }
  void AddSupervisor(Manager* supervisor) {
    supervisors_.emplace_back(supervisor);
    supervisor->AddSubordinate(this);
  }
  std::vector<const Manager*> get_supervisors() const { return supervisors_; }

```

```
 private:
  std::vector<const Manager*> supervisors_;
};

#endif /* ADMINISTRATIVE_ASSISTANT_H */
```

- File `board_of_directors.h`

```
#ifndef BOARD_OF_DIRECTORS_H
#define BOARD_OF_DIRECTORS_H

#include <iostream>
#include <vector>
#include "common.h"
#include "person.h"

class Company;

class BoardOfDirectors {
 public:
  BoardOfDirectors() {}
  BoardOfDirectors(std::vector<const Person*>& p) {
    if (p.size() > 8 || p.size() < 3) {
      std::cerr << "[ERROR] The number of member should be in range 3 to 8.\n";
      return;
    }
    board_member_ = p;
  }
  void set_company(const Company* c) { company_ = c; }
  const Company* get_company() { return company_; }
  std::size_t get_num_of_board() const { return board_member_.size(); }

 private:
  std::vector<const Person*> board_member_{};
  const Company* company_{nullptr};
};

#endif /* BOARD_OF_DIRECTORS_H */

```

- File `company.h`

```
#ifndef COMPANY_H
#define COMPANY_H

#include <string>
#include <vector>

#include "board_of_directors.h"
#include "common.h"
#include "employee.h"

class BoardOfDirectors;

class Company {
 public:
  Company(std::string name, std::string unified_business_number)
      : name_{name}, unified_business_number_{unified_business_number} {}
  std::string get_name() const { return name_; }
  std::string get_unified_business_number() const {
    return unified_business_number_;
  }
  void AddEmployee(Employee *e) {
    employees_.emplace_back(e);
    e->set_workfor(this);
  }
  std::size_t get_num_of_employees() const { return employees_.size(); }
```

```
  void set_board_of_directors(BoardOfDirectors *b) {
    b->set_company(this);
    board_of_directors_ = b;
  }
  const BoardOfDirectors *get_board_of_directors() {
    return board_of_directors_;
  }

  std::vector<const Employee *> Filter(std::string name) {
    std::vector<const Employee *> ret;
    for (const Employee *e : employees_)
      if (e->get_name() == name) ret.emplace_back(e);
    return ret;
  }

  std::vector<const Employee *> Filter(int id) {
    std::vector<const Employee *> ret;
    for (const Employee *e : employees_)
      if (e->get_id() == id) ret.emplace_back(e);
    return ret;
  }

  std::vector<const Employee *> Filter(Gender gender) {
    std::vector<const Employee *> ret;
    for (const Employee *e : employees_)
      if (e->get_gender() == gender) ret.emplace_back(e);
    return ret;
  }

  std::vector<const Employee *> Filter(Position position) {
    std::vector<const Employee *> ret;
    for (const Employee *e : employees_)
      if (e->get_position() == position) ret.emplace_back(e);
    return ret;
  }

 private:
  std::string name_;
  std::string unified_business_number_;
  std::vector<const Employee *> employees_{};
  const BoardOfDirectors *board_of_directors_{nullptr};
};

#endif /* COMPANY_H */
```

- File `office.h`

```
#ifndef OFFICE_H
#define OFFICE_H

#include <vector>

#include "common.h"
#include "employee.h"

class Office {
 public:
  Office(std::string name) : name_{name} {}
  std::string get_name() const { return name_; }
  void AddEmployee(Employee* e) {
    allocated_to_.emplace_back(e);
    e->set_office(this);
  }

 private:
  std::string name_;
  std::vector<const Employee*> allocated_to_{};
};
```

```
22
23  #endif
```

- File `relation1.cc`

```
1   #include <iostream>
2   #include <vector>
3
4   #include "common.h"
5   #include "company.h"
6   #include "employee.h"
7
8   int main() {
9     /* New Company */
10    auto apple = new Company("Apple", "0123456789");
11
12    /* New employees */
13    auto alice = new Employee("Alice", Gender::F, 0, Position::Chairman);
14    auto bob = new Employee("Bob", Gender::M, 1, Position::SeniorEngineer);
15    auto carol = new Employee("Carol", Gender::M, 2, Position::ProductManager);
16    auto dave = new Employee("Dave", Gender::M, 3, Position::JuniorEngineer);
17    auto eve = new Employee("Eve", Gender::F, 4, Position::JuniorEngineer);
18
19    /* Add relation */
20    apple->AddEmployee(alice);
21    apple->AddEmployee(bob);
22    apple->AddEmployee(carol);
23    apple->AddEmployee(dave);
24    apple->AddEmployee(eve);
25
26    /* Validation */
27    std::cout << "Number of employees in the company, " << apple->get_name()
28              << ", is " << apple->get_num_of_employees() << ".\n";
29    std::cout << alice->get_name() << " work for "
30              << alice->get_workfor()->get_name() << ".\n";
31
32    /* Filter function */
33    std::cout << "Filter id where position is \"Junior Engineer\":\n";
34    auto filter = apple->Filter(Position::JuniorEngineer);
35    for (const auto e : filter) {
36      std::cout << e->get_id() << " ";
37    }
38    std::cout << "\n";
39
40    std::cout << "Filter name where gender is \"M\":\n";
41    auto filter_gender = apple->Filter(Gender::M);
42    for (const auto e : filter_gender) {
43      std::cout << e->get_name() << " ";
44    }
45    std::cout << "\n";
46
47    /* Release allocated resources */
48    for (auto &p : {alice, bob, carol, dave, eve}) delete p;
49    delete apple;
50    return 0;
51  }
```

- File `relation2.cc`

```
1   #include <iostream>
2   #include <vector>
3
4   #include "administrative_assistant.h"
5   #include "common.h"
6   #include "manager.h"
7
8   int main() {
```

```
  Employee* alice = new Manager("Alice", Gender::F, 0);
  Employee* bob = new Manager("Bob", Gender::M, 0);
  Employee* carol = new Manager("Carol", Gender::M, 0);
  Employee* dave = new Manager("Dave", Gender::M, 0);

  /* Each administrative assistant should have at least one manager as
   * the supervisor */
  Employee* eve = new AdministrativeAssistant("Eve", Gender::F, 0,
                                              dynamic_cast<Manager*>(alice));
  Employee* isaac = new AdministrativeAssistant("Isaac", Gender::M, 0,
                                               dynamic_cast<Manager*>(carol));
  Employee* justin = new AdministrativeAssistant("Justin", Gender::M, 0,
                                                dynamic_cast<Manager*>(alice));

  /* Assign more manager as the supervisor */
  dynamic_cast<AdministrativeAssistant*>(eve)->AddSupervisor(
      dynamic_cast<Manager*>(bob));
  dynamic_cast<AdministrativeAssistant*>(eve)->AddSupervisor(
      dynamic_cast<Manager*>(dave));

  /* Validation */
  std::cout << "The supervisors of " << eve->get_name() << ":\n";
  for (const auto i :
       dynamic_cast<AdministrativeAssistant*>(eve)->get_supervisors()) {
    std::cout << i->get_name() << " ";
  }
  std::cout << "\n";

  std::cout << "The subordinates of " << alice->get_name() << ":\n";
  for (const auto i : dynamic_cast<Manager*>(alice)->get_subordinates()) {
    std::cout << i->get_name() << " ";
  }
  std::cout << "\n";

  /* Release allocated resources */
  for (auto& p : {alice, bob, carol, dave, eve, isaac, justin}) delete p;
  return 0;
}
```

- File `relation3.cc`

```
#include <iostream>
#include <vector>

#include "board_of_directors.h"
#include "common.h"
#include "company.h"

int main() {
  auto apple = new Company("Apple", "0123456789");
  auto board_of_apple = new BoardOfDirectors();
  apple->set_board_of_directors(board_of_apple);

  std::cout << "The company of the board: ";
  std::cout << board_of_apple->get_company()->get_name() << "\n";
  std::cout << "The number of board members of Apple company: ";
  std::cout << apple->get_board_of_directors()->get_num_of_board() << "\n";

  /* Release allocated resources */
  delete apple;
  delete board_of_apple;
  return 0;
}
```

- File `relation4.cc`

```
#include <iostream>
```

```
#include "employee.h"
#include "office.h"

int main() {
  /* New employees */
  auto alice = new Employee("Alice", Gender::F, 0, Position::Chairman);
  auto bob = new Employee("Bob", Gender::M, 1, Position::SeniorEngineer);
  auto carol = new Employee("Carol", Gender::M, 2, Position::ProductManager);
  auto dave = new Employee("Dave", Gender::M, 3, Position::JuniorEngineer);
  auto eve = new Employee("Eve", Gender::F, 4, Position::JuniorEngineer);

  auto office = new Office("room1");
  office->AddEmployee(bob);
  office->AddEmployee(carol);
  office->AddEmployee(dave);

  for (auto &e : {alice, bob, carol, dave, eve}) {
    if (e->get_office() != nullptr)
      std::cout << e->get_name() << " is in " << e->get_office()->get_name()
                << ".\n";
    else
      std::cout << e->get_name() << " not in any office.\n";
    delete e;
  }
  delete office;
  return 0;
}
```

- File `relation5.cc`

```
#include <vector>

#include "board_of_directors.h"
#include "common.h"
#include "person.h"

int main() {
  std::vector<const Person*> v1;
  v1.push_back(new Person("p0", Gender::F));
  v1.push_back(new Person("p1", Gender::F));
  v1.push_back(new Person("p2", Gender::M));
  v1.push_back(new Person("p3", Gender::M));
  v1.push_back(new Person("p4", Gender::F));

  auto board1 = new BoardOfDirectors(v1);
  std::cout << "The number of member: " << board1->get_num_of_board() << "\n";

  std::vector<const Person*> v2;
  v2.push_back(new Person("r0", Gender::F));
  v2.push_back(new Person("r1", Gender::F));

  auto board2 = new BoardOfDirectors(v2);
  std::cout << "The number of member: " << board2->get_num_of_board() << "\n";

  /* Release allocated resources */
  for (auto& p : v1) delete p;
  for (auto& p : v2) delete p;
  for (auto& p : {board1, board2}) delete p;
}
```

Executive results

- relation1:

```
$ ./bin/relation1
Number of employees in the company, Apple, is 5.
Alice work for Apple.
Filter id where position is "Junior Engineer":
3 4
Filter name where gender is "M":
Bob Carol Dave
```

- relation2:

```
$ ./bin/relation2
The supervisors of Eve:
Alice Bob Dave
The subordinates of Alice:
Eve Justin
```

- relation3:

```
$ ./bin/relation3
The company of the board: Apple
The number of board members of Apple company: 0
```

- relation4:

```
$ ./bin/relation4
Alice not in any office.
Bob is in room1.
Carol is in room1.
Dave is in room1.
Eve not in any office.
```

- relation5:

```
$ ./bin/relation5
The number of member: 5
[ERROR] The number of member should be in range 3 to 8.
The number of member: 0
```