Object-Oriented Software Engineering hw6

- 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

File architecture

```
$  tree . -I 'bin|*.md|Makefile|img'
.
├── airline.h
├── booking.h
├── common.cc
├── common.h
├── employee_role.h
├── main.cc
├── passenger_role.h
├── person.h
├── person_role.h
├── regular_flight.h
└── specific_flight.h
```

- File `common.h`

```cpp
#ifndef COMMON_H
#define COMMON_H

#include <cstdint>
#include <iostream>

struct Time {
  int8_t h;
  int8_t m;
};
/**
 * @brief Define the output format for Time struct.
 *
 * @param out The output stream.
 * @param time The time that needs to be printed to output stream.
 * @return std::ostream&
 */
std::ostream &operator<<(std::ostream &out, const Time time);

struct Date {
  int16_t year;
  int8_t month;
  int8_t day;
};
/**
 * @brief Define the output format for Date struct.
 *
 * @param out The output stream.
 * @param date The date that needs to be printed to output stream.
 * @return std::ostream&
 */
std::ostream &operator<<(std::ostream &out, const Date date);

#endif /* COMMON_H */
```

- File `common.cc`

```
#include "common.h"

std::ostream &operator<<(std::ostream &out, const Time time) {
  out << static_cast<int>(time.h) << ":" << static_cast<int>(time.m);
  return out;
}

std::ostream &operator<<(std::ostream &out, const Date date) {
  out << static_cast<int>(date.year) << "/" << static_cast<int>(date.month)
      << "/" << static_cast<int>(date.day);
  return out;
}
```

- File `airline.h`

```
#ifndef AIRLINE_H
#define AIRLINE_H

#include <string>
#include <vector>

class Person;
class RegularFlight;

class Airline {
 public:
  Airline(std::string name) : name{name} {}
  void addPerson(Person* person) { people.emplace_back(person); }
  void addRegularFlight(RegularFlight* reg_flight) {
    flights.emplace_back(reg_flight);
  }
  std::vector<RegularFlight*>& getRegularFlights() { return flights; }
  std::vector<Person*>& getPeople() { return people; }

 private:
  std::string name;
  std::vector<Person*> people{};
  std::vector<RegularFlight*> flights{};
};

#endif /* AIRLINE_H */
```

- File `booking.h`

```
#ifndef BOOKING_H
#define BOOKING_H

class PassengerRole;
class SpecificFlight;

class Booking {
 public:
  Booking(int seatNumber) : seatNumber{seatNumber} {}
  void linkPassengerRole(PassengerRole* passenger) {
    this->passenger = passenger;
  }
  void linkSpecificFlight(SpecificFlight* specific_flight) {
    this->specific_flight = specific_flight;
  }
  SpecificFlight* getSpecificFlight() const { return specific_flight; }
  int getSeatNumber() const { return seatNumber; }

 private:
  int seatNumber;
  PassengerRole* passenger{nullptr};
  SpecificFlight* specific_flight{nullptr};
};

#endif /* BOOKING_H */
```

- File `person.h`

```
#ifndef PERSON_H
#define PERSON_H

#include <initializer_list>
#include <iostream>
#include <string>
#include <vector>

#include "airline.h"
#include "employee_role.h"
#include "passenger_role.h"
#include "person_role.h"
#include "specific_flight.h"

class Person {
 public:
  Person(std::string name, std::string idNumber, Airline* airline,
         std::initializer_list<PersonRole*> roles)
      : name{name}, idNumber{idNumber} {
    if (airline != nullptr) {
      linkAirline(airline);
    }
    for (auto& p : roles) {
      this->addPersonRole(p);
    }
  }
  Person(std::string name, std::string idNumber)
      : Person{name, idNumber, nullptr, {}} {}
  Person(std::string name, std::string idNumber,
         std::initializer_list<PersonRole*> roles)
      : Person{name, idNumber, nullptr, roles} {}
  ~Person() {
    for (auto& p : roles) {
      delete p;
    }
  }
  std::string getName() const { return name; }
  std::string getIdNumber() const { return idNumber; }
  EmployeeRole* get_employee_role() {
    for (auto role : roles) {
      EmployeeRole* emp = dynamic_cast<EmployeeRole*>(role);
      if (emp != nullptr) {
        return emp;
      }
    }
    std::cerr << name << " does not have an EmployeeRole\n";
    return nullptr;
  }
  PassengerRole* get_passenger_role() {
    for (auto role : roles) {
      PassengerRole* pas = dynamic_cast<PassengerRole*>(role);
      if (pas != nullptr) {
        return pas;
      }
    }
    std::cerr << name << " does not have an EmployeeRole\n";
    return nullptr;
  }
  void addPersonRole(PersonRole* person_role) {
```

```cpp
    if (roles.size() > 2) {
      std::cerr << "PersonRole should not more than 2\n";
      return;
    }
    roles.push_back(person_role);
    person_role->linkPerson(this);
  }
  void linkAirline(Airline* airline) {
    this->airline = airline;
    airline->addPerson(this);
  }

 private:
  std::string name;
  std::string idNumber;
  std::vector<PersonRole*> roles{};
  Airline* airline{nullptr};
};

#endif /* PERSON_H */
```

- File `person_role.h`

```cpp
#ifndef PERSON_ROLE_H
#define PERSON_ROLE_H

class Person;

class PersonRole {
 public:
  void linkPerson(Person* person) { this->person = person; }
  Person* getPerson() const { return person; }
  virtual ~PersonRole() = default;

 protected:
  Person* person{nullptr};
};

#endif /* PERSON_ROLE_H */
```

- File `employee_role.h`

```cpp
#ifndef EMPLOYEE_ROLE_H
#define EMPLOYEE_ROLE_H

#include <iostream>
#include <vector>

#include "person_role.h"
#include "specific_flight.h"

class EmployeeRole : public PersonRole {
 public:
  EmployeeRole(std::string jobFunction) : jobFunction{jobFunction} {}
  EmployeeRole(std::string jobFunction, EmployeeRole* supervisor)
      : EmployeeRole{jobFunction} {
    addSupervisor(supervisor);
  }
  void perfomJobFunction() { std::cout << jobFunction << "\n"; }
  void addSubordinate(EmployeeRole* emp) {
    subordinates.emplace_back(emp);
    emp->linkSupervisor(emp);
  }
  void addSupervisor(EmployeeRole* emp) { emp->addSubordinate(this); }
  void linkSupervisor(EmployeeRole* emp) { this->supervisor = emp; }
  void addSpecificFlight(SpecificFlight* specific_flight) {
    this->specific_flights.push_back(specific_flight);
    specific_flight->addEmployeeRole(this);
  }
  std::vector<EmployeeRole*>& getSubordinates() { return subordinates; }

 private:
  std::string jobFunction{""};
  EmployeeRole* supervisor{nullptr};
  std::vector<EmployeeRole*> subordinates{};
  std::vector<SpecificFlight*> specific_flights{};
};

#endif /* EMPLOYEE_ROLE_H */
```

- File `passenger_role.h`

```cpp
#ifndef PASSENGER_ROLE_H
#define PASSENGER_ROLE_H

#include <vector>

#include "booking.h"
#include "person_role.h"
#include "specific_flight.h"

class PassengerRole : public PersonRole {
 public:
  PassengerRole() {}
  ~PassengerRole() {
    for (auto& b : bookings) {
      delete b;
    }
  }
  void addBooking(Booking* booking) {
    bookings.emplace_back(booking);
    booking->linkPassengerRole(this);
  }

  Booking* bookSpecificFlight(SpecificFlight* specific_flight, int seatNumber) {
    Booking* b = new Booking(seatNumber);
    this->addBooking(b);
    specific_flight->addBooking(b);
    return b;
  }
  void printBookings() {
    for (const auto& b : bookings) {
      std::cout << b->getSeatNumber() << " : "
                << b->getSpecificFlight()->getDate() << " : "
                << b->getSpecificFlight()->getRegularFlight()->getTime()
                << "\n";
    }
  }
  void cancelBooking(Booking* booking) {
    for (auto it = bookings.begin(); it != bookings.end(); ++it) {
      if (*it == booking) {
        (*it)->getSpecificFlight()->cancelBooking(*it);
        delete *it;
        bookings.erase(it);
        break;
      }
    }
  }

 private:
  std::vector<Booking*> bookings{};
};

#endif /* PASSENGER_ROLE_H */
```

- File `regular_flight.h`

```cpp
#ifndef REGULAR_FLIGHT_H
#define REGULAR_FLIGHT_H

#include <vector>

#include "airline.h"
#include "common.h"

class SpecificFlight;

class RegularFlight {
 public:
  RegularFlight(Time time, int flightNumber, Airline* airline)
      : time{time}, flightNumber{flightNumber} {
    linkAirline(airline);
  }
  std::vector<SpecificFlight*>& getSpecificFlights() {
    return specific_flights;
  }
  Time getTime() const { return time; }
  int getFlightNumber() const { return flightNumber; }
  void addSpecificFlight(SpecificFlight* specific_flight) {
    specific_flights.emplace_back(specific_flight);
  }
  void linkAirline(Airline* airline) {
    this->airline = airline;
    airline->addRegularFlight(this);
  }

 private:
  Time time;
  int flightNumber;
  std::vector<SpecificFlight*> specific_flights{};
  Airline* airline{nullptr};
};

#endif /* REGULAR_FLIGHT_H */
```

- File `specific_flight.h`

```cpp
#ifndef SPECIFIC_FLIGHT_H
#define SPECIFIC_FLIGHT_H

#include <vector>

#include "booking.h"
#include "common.h"
#include "regular_flight.h"

class EmployeeRole;

class SpecificFlight {
 public:
  SpecificFlight(Date date, RegularFlight* regular_flight) : date{date} {
    linkRegularFlight(regular_flight);
  }
  Date getDate() const { return date; }
  RegularFlight* getRegularFlight() const { return regular_flight; }
  void addBooking(Booking* booking) {
    bookings.emplace_back(booking);
    booking->linkSpecificFlight(this);
  }
  void linkRegularFlight(RegularFlight* regular_flight) {
    this->regular_flight = regular_flight;
    regular_flight->addSpecificFlight(this);
  }
  void addEmployeeRole(EmployeeRole* emp) { this->employees.push_back(emp); }
  void cancelBooking(Booking* booking) {
    for (auto it = bookings.begin(); it != bookings.end(); ++it) {
      if (*it == booking) {
        bookings.erase(it);
        break;
      }
    }
  }

 private:
  Date date;
  std::vector<Booking*> bookings{};
  std::vector<EmployeeRole*> employees{};
  RegularFlight* regular_flight{nullptr};
};

#endif /* SPECIFIC_FLIGHT_H */
```

- File `main.cc`

```cpp
#include "airline.h"
#include "booking.h"
#include "employee_role.h"
#include "passenger_role.h"
#include "person.h"
#include "person_role.h"
#include "regular_flight.h"
#include "specific_flight.h"

int main() {
  Airline* happy_air = new Airline("Happy Air");
  RegularFlight* flight1 = new RegularFlight(Time{8, 17}, 1, happy_air);
  SpecificFlight* flight1_0516 = new SpecificFlight(Date{2021, 5, 16}, flight1);
  SpecificFlight* flight1_0517 = new SpecificFlight(Date{2021, 5, 17}, flight1);
  SpecificFlight* flight1_0518 = new SpecificFlight(Date{2021, 5, 18}, flight1);

  RegularFlight* flight2 = new RegularFlight(Time{18, 47}, 2, happy_air);
  SpecificFlight* flight2_0516 = new SpecificFlight(Date{2021, 5, 16}, flight2);

  Person* alice =
      new Person("Alice", "0123", happy_air, {new EmployeeRole("captain")});
  Person* bob =
      new Person("Bob", "0124", happy_air, {new EmployeeRole("crew")});
  Person* carol =
      new Person("Carol", "0125", happy_air,
                 {new EmployeeRole("crew", bob->get_employee_role())});
  Person* dave = new Person("Dave", "0126", happy_air,
                            {new EmployeeRole("crew", bob->get_employee_role()),
                             new PassengerRole()});

  Person* eve = new Person("Eve", "0127", {new PassengerRole()});
  Person* isaac = new Person("Isaac", "0128", {new PassengerRole()});
  Person* justin = new Person("Justin", "0129", {new PassengerRole()});

  std::cout << "\nDate for flight1:\n";
  for (const auto& f : flight1->getSpecificFlights()) {
    std::cout << f->getDate() << "\n";
  }
  std::cout << "\nDate for flight2:\n";
  for (const auto& f : flight2->getSpecificFlights()) {
    std::cout << f->getDate() << "\n";
  }
  std::cout << "\nTime for flights in Happy Air:\n";
  for (const auto& f : happy_air->getRegularFlights()) {
    std::cout << "No." << f->getFlightNumber() << ": " << f->getTime() << "\n";
  }
  std::cout << "\nEmployees in Happy Air:\n";
  for (const auto& p : happy_air->getPeople()) {
    std::cout << p->getName() << "\n";
  }
  std::cout << "\nThe employees whose supervisor is Bob:\n";
  for (const auto& s : bob->get_employee_role()->getSubordinates()) {
    std::cout << s->getPerson()->getName() << "\n";
  }

  /* Test booking and canceling */
  Booking* b1 = dave->get_passenger_role()->bookSpecificFlight(flight1_0516, 1);
  dave->get_passenger_role()->bookSpecificFlight(flight1_0517, 3);
  std::cout << "\n" << dave->getName() << " has booked: \n";
  dave->get_passenger_role()->printBookings();
  dave->get_passenger_role()->cancelBooking(b1);
  std::cout << "\n" << dave->getName() << " has booked: \n";
  dave->get_passenger_role()->printBookings();

  eve->get_passenger_role()->bookSpecificFlight(flight1_0516, 2);
  isaac->get_passenger_role()->bookSpecificFlight(flight1_0517, 2);
  justin->get_passenger_role()->bookSpecificFlight(flight2_0516, 1);

  for (auto& p : {alice, bob, carol, dave, eve, isaac, justin}) delete p;
  for (auto& p : {flight1_0516, flight1_0517, flight1_0518, flight2_0516})
    delete p;
  for (auto& p : {flight1, flight2}) delete p;
  delete happy_air;
  return 0;
}
```

Executive results

```
$ ./bin/main

Date for flight1:
2021/5/16
2021/5/17
2021/5/18

Date for flight2:
2021/5/16

Time for flights in Happy Air:
No.1: 8:17
No.2: 18:47

Employees in Happy Air:
Alice
Bob
Carol
Dave

The employees whose supervisor is Bob:
Carol
Dave

Dave has booked:
1 : 2021/5/16 : 8:17
3 : 2021/5/17 : 8:17

Dave has booked:
3 : 2021/5/17 : 8:17
```