Object-Oriented Software Engineering hw1

- Author: 黃柏瑄 (P78081528)

Environment

- OS: Ubuntu18.04.5 (WSL2)
- C++ compiler: g++ (Ubuntu 7.5.0-3ubuntu1~18.04) 7.5.0

Source code

- Source code (`hw1.cc`):

```cpp
// g++ --std=c++1z -O2 -Wall -o hw1 hw1.cc
// ./hw1
#include <iostream>
#include <memory>
#include <string>
#include <unordered_map>

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

template <typename T, typename... Args>
std::shared_ptr<T> make_aggregate_shared(Args&&... args) {
  return std::make_shared<T>(T{std::forward<Args>(args)...});
}
struct Ticket {
  std::string buyer_name;
  int num_of_people;
  std::string bus_name;
  Date bus_departure_date;
};

class TicketTransactor {
 public:
  std::string get_name() const { return name_; }

 protected:
  explicit TicketTransactor(const std::string name) noexcept : name_{name} {}
  void NewTransaction(const int ticket_index,
                      const std::shared_ptr<Ticket> ticket) {
    tickets_.insert(std::make_pair(ticket_index, std::move(ticket)));
  }
  void EraseTransaction(const int ticket_index) {
    auto ticket = tickets_.find(ticket_index);
    if (ticket != tickets_.end()) {
      tickets_.erase(ticket);
    } else {
      std::cout << name_ << " did not book ticket with index: " << ticket_index
                << ".\n";
    }
  }
  std::string name_;
  std::unordered_map<int, std::shared_ptr<Ticket>> tickets_;
};

class Person : public TicketTransactor {
 public:
  explicit Person(const std::string name) noexcept : TicketTransactor{name} {}
  void BuyTicket(const int ticket_index, const std::shared_ptr<Ticket> ticket) {
    NewTransaction(ticket_index, std::move(ticket));
```

```
  }
  void PrintBookedBuses() const {
    if (tickets_.empty()) {
      std::cout << name_ << " does not book any ticket for bus.\n";
      return;
    }
    std::cout << name_ << " has booked: ";
    for (const auto& t : tickets_) {
      std::cout << "(" << t.second->bus_name << ", "
                << t.second->bus_departure_date << ") ";
    }
    std::cout << "\n";
  }
};

class BusForBooking : public TicketTransactor {
 public:
  explicit BusForBooking(const std::string name, const Date date) noexcept
      : TicketTransactor{name}, departure_date_{date} {}
  Date get_departure_date() const { return departure_date_; }
  void SellTicket(const int ticket_index,
                  const std::shared_ptr<Ticket> ticket) {
    NewTransaction(ticket_index, std::move(ticket));
  }
  void PrintPassengers() const {
    if (tickets_.empty()) {
      std::cout << name_ << " does not have any passenger.\n";
      return;
    }
    std::cout << "The passengers of " << name_ << ": ";
    for (const auto& t : tickets_) {
      std::cout << "(" << t.second->buyer_name << ", "
                << t.second->num_of_people << ") ";
    }
    std::cout << "\n";
  }

 private:
  Date departure_date_;
};

class TicketMachine {
 public:
  static TicketMachine& GetTicketMachine() noexcept {
    static TicketMachine instance;
    return instance;
  }
  TicketMachine(const TicketMachine&) = delete;
  void operator=(const TicketMachine&) = delete;
  void Book(Person* buyer, BusForBooking* bus, const int num_of_people) const {
    auto ticket = make_aggregate_shared<Ticket>(buyer->get_name(),
                                                num_of_people, bus->get_name(),
                                                bus->get_departure_date());
    bus->SellTicket(ticket_index_, ticket);
    buyer->BuyTicket(ticket_index_, ticket);
    ticket_index_++;
  }

 private:
  TicketMachine() {}
  static int ticket_index_;
};
int TicketMachine::ticket_index_ = 0;

int main() {
  /* People */
  auto alice = std::make_unique<Person>("Alice");
  auto bob = std::make_unique<Person>("Bob");
  auto carol = std::make_unique<Person>("Carol");
  auto dave = std::make_unique<Person>("Dave");
  auto eve = std::make_unique<Person>("Eve");

  /* Bus */
  auto bus100 = std::make_unique<BusForBooking>("Bus100", Date{2021, 2, 25});
```

```
    auto bus101 = std::make_unique<BusForBooking>("Bus101", Date{2021, 2, 26});
    auto bus102 = std::make_unique<BusForBooking>("Bus102", Date{2021, 2, 27});
    auto bus103 = std::make_unique<BusForBooking>("Bus103", Date{2022, 2, 28});

    /* Book tickets */
    auto& tmachine = TicketMachine::GetTicketMachine();
    tmachine.Book(alice.get(), bus100.get(), 4);
    tmachine.Book(alice.get(), bus102.get(), 2);
    tmachine.Book(bob.get(), bus100.get(), 6);
    tmachine.Book(carol.get(), bus101.get(), 3);
    tmachine.Book(dave.get(), bus100.get(), 5);

    /* Validation */
    bus100->PrintPassengers();
    alice->PrintBookedBuses();
    alice->PrintBookedBuses();
    bus101->PrintPassengers();
    bob->PrintBookedBuses();
    bus103->PrintPassengers();
    eve->PrintBookedBuses();
    return 0;
}
```

- Executive result:

```
$ g++ --std=c++1z -O2 -Wall -o hw1 hw1.cc
$ ./hw1
The passengers of Bus100: (Dave, 5) (Alice, 4) (Bob, 6)
Alice has booked: (Bus102, 2021/2/27) (Bus100, 2021/2/25)
Alice has booked: (Bus102, 2021/2/27) (Bus100, 2021/2/25)
The passengers of Bus101: (Carol, 3)
Bob has booked: (Bus100, 2021/2/25)
Bus103 does not have any passenger.
Eve does not book any ticket for bus.
```