Object-Oriented Software Engineering hw2

- 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 `hw2.cc`:

```cpp
/**
 * @file hw2.cc
 * @author Huang Po-Hsuan (aben20807@gmail.com)
 * @brief OOSE hw2
 *
 * Coding style: Google C++ Style Guide
 * (https://google.github.io/styleguide/cppguide.html)
 *
 * Compile: g++-8 --std=c++1z -O2 -Wall -o hw2 hw2.cc
 * Run: ./hw2
 */
#include <iostream>
#include <memory>
#include <string>
#include <unordered_map>

struct Date {
  uint16_t year;
  uint8_t month;
  uint8_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) {
  out << date.year << "/" << static_cast<int>(date.month) << "/"
      << static_cast<int>(date.day);
  return out;
}
/**
 * @brief Adapter to make aggregate struct be shared.
 *
 * @tparam T The type of aggregate struct.
 * @tparam Args The variadic type of args.
 * @param args The in-order elements of aggregate struct.
 * @return std::shared_ptr<T>
 */
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 Booking {
  std::string buyer_name;
  std::string bus_name;
  int num_of_people;
  Date bus_departure_date;
};
```

```cpp

/**
 * @brief An abstract class to record the transaction of booking.
 *
 * All classes derived from this class have an unordered map, bookings_.
 * Each element of bookings_ contains an integer index and corresponding shared
 * pointer to a Booking object. Unordered map's average time to search, insert,
 * and delete are O(1).
 *
 * Note that one Booking object will have two shared pointer to point it as
 * below:
 *  +-----------------+           +------------------+
 *  | passenger       |           | bus for booking  |
 *  | +-------------+|           | +--------------+ |
 *  | | +-+ +-+ +-+  ||           | | +-+ +-+ +-+  | |
 *  | | ||| ||| | |  ||    +------------| ||| | |  | |
 *  | | +|+ +|+ +-+  ||    |      | | +-+ +|+ +-+  | |
 *  | +--|---|-------+|    |      | +------|-------+ |
 *  +----|---|--------+    |      +--------|---------+
 *       |   |             |               |
 *       |   |             |               |
 *       |   |             |    +-------+  |
 *       |   +------------------>|booking|<-+
 *       |                 |    +-------+
 *       |                 v
 *       |           +-------+
 *       +---------->|booking|
 *                   +-------+
 */
class BookingTransactor {
 public:
  std::string get_name() const { return name_; }
  void AddBooking(const int booking_index,
                  const std::shared_ptr<Booking> booking) {
    bookings_.emplace(booking_index, std::move(booking));
  }
  void DeleteBookingByIndex(const int booking_index) {
    auto erase_count = bookings_.erase(booking_index);
    if (erase_count == 0) {
      std::cout << name_
                << " did have the booking with index: " << booking_index
                << ".\n";
    }
  }
  /**
   * @brief Print the information in different aspects.
   *
   * For example, passenger wants to see the bus info; bus wants to see
   * passenger info.
   */
  virtual void PrintBookings() const = 0;

 protected:
  explicit BookingTransactor(const std::string name) noexcept : name_{name} {}
  std::string name_;
  std::unordered_map<int, std::shared_ptr<Booking>> bookings_;
};

class Passenger : public BookingTransactor {
 public:
  explicit Passenger(const std::string name) noexcept
      : BookingTransactor{name} {}
  /**
   * @brief Overridden function to print bus info from passenger's booking list.
   */
  void PrintBookings() const override {
    if (bookings_.empty()) {
      std::cout << name_ << " does not book any booking for bus.\n";
      return;
```

```
    }
    std::cout << name_ << " has booked:";
    for ([[maybe_unused]] const auto &[_, booking_ptr] : bookings_) {
      std::cout << " (" << booking_ptr->bus_name << ", "
                << booking_ptr->bus_departure_date << ")";
    }
    std::cout << ".\n";
  }
};

class BusForBooking : public BookingTransactor {
 public:
  explicit BusForBooking(const std::string name, const Date date) noexcept
      : BookingTransactor{name}, departure_date_{date} {}
  Date get_departure_date() const { return departure_date_; }
  /**
   * @brief Overridden function to print passenger info from bus's booking list.
   */
  void PrintBookings() const override {
    if (bookings_.empty()) {
      std::cout << name_ << " does not have any passenger.\n";
      return;
    }
    std::cout << "The passengers of " << name_ << ":";
    for ([[maybe_unused]] const auto &[_, booking_ptr] : bookings_) {
      std::cout << " (" << booking_ptr->buyer_name << ", "
                << booking_ptr->num_of_people << ")";
    }
    std::cout << ".\n";
  }

 private:
  /**
   * @brief In this homework, I use the departure date rather than 班次.
   */
  Date departure_date_;
};

/**
 * @brief A singleton booking machine.
 *
 * Used to connect two booking transactors.
 */
class BookingMachine {
 public:
  /**
   * @brief Get the Booking Machine object.
   *
   * Because the constructor is private, the way to get booking machine is to
   * use this function.
   * @return BookingMachine&
   */
  static BookingMachine &GetBookingMachine() {
    static BookingMachine instance;
    return instance;
  }
  /**
   * @brief Copy constructor and copy assignment are deleted so that the object
   * cannot be copied.
   */
  BookingMachine(const BookingMachine &) = delete;
  void operator=(const BookingMachine &) = delete;
  /**
   * @brief Add one booking to connect two transactors.
   *
   * Every bookings increase the booking_index_ to make it unique.
   * Shared pointer (shared_ptr) is used to share the booking object to two
   * transactors, and the booking object will be freed automatically if the
   * pointer counter becomes 0.
```

```
   * @param passenger The pointer to the passenger.
   * @param bus The pointer to the bus.
   * @param num_of_people how many seats (number of people) are booked in this
   * action.
   */
  void MakeBooking(Passenger *const passenger, BusForBooking *const bus,
                   const int num_of_people) {
    auto booking = make_aggregate_shared<Booking>(
        passenger->get_name(), bus->get_name(), num_of_people,
        dynamic_cast<BusForBooking *>(bus)->get_departure_date());
    passenger->AddBooking(booking_index_, booking);
    bus->AddBooking(booking_index_, booking);
    booking_index_++;
  }

 private:
  BookingMachine() {}
  inline static int booking_index_{0};
};

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

  /* New buses */
  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 bus bookings */
  auto &tmachine = BookingMachine::GetBookingMachine();
  tmachine.MakeBooking(alice.get(), bus100.get(), 4);
  tmachine.MakeBooking(alice.get(), bus102.get(), 2);
  tmachine.MakeBooking(bob.get(), bus100.get(), 6);
  tmachine.MakeBooking(carol.get(), bus101.get(), 3);
  tmachine.MakeBooking(dave.get(), bus100.get(), 5);

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

Compilation and Executive result

```
$ g++-8 --std=c++1z -O2 -Wall -o hw2 hw2.cc
$ ./hw2
The passengers of Bus100: (Dave, 5) (Alice, 4) (Bob, 6).
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 booking for bus.
```