Lab 5.3.3 Flight booking system: part 3

Objectives

Familiarize the student with:

- modelling real-world entities with classes and objects;
- limiting acceptable input range;
- managing multiple objects.

Scenario

Let's finish working on our booking system.

The last part for our booking system will be to manage multiple flights.

To do this, we'll modify our program's command list.

- The command "create [id] [cap]" will try to create a new empty flight with ID [id] and capacity [cap].
- The command "delete [id]" will try to remove the flight with ID [id].
- The command "add [id] [n]" will try to add n reservations to the flight with ID [id].
- The command "cancel [id] [n]" will try to cancel n reservations from the flight with ID [id].
- If an operation fails for any reason, the program will issue the message "Cannot perform this operation". You can add a more helpful message to identify why the operation failed.
- The command "quit" will stop the execution of the program.

For the sake of simplicty, let's limit the maximum number of handled flights to ten.

Use the code from your previous exercise as a starting point.

Note that we need to add an access method to the flight **id** field.

As a convention, we can assume that 0 is not a valid flight ID, so a flight with an ID of 0 may be considered as nonexistent.

```
#include <iostream>

class FlightBooking {
public:
  FlightBooking(int id, int capacity, int reserved);
  FlightBooking();
  void printStatus();
  bool reserveSeats(int number_ob_seats);
  bool cancelReservations(int number_ob_seats);
  int getId() { retutn id };
private:
  int id;
  int capacity;
  int reserved;
};

FlightBooking::FlightBooking()
{
  id = 0; capacity = 0; reserved = 0;
}

// ...

int main() {
  FlightBooking booking[10];

  // Use this to have some starting value
  // booking[0] = FlightBooking(1, 400, 0);
  // booking[0].printStatus();

  std::string command = "";
  while (command != "quit")
  {
    std::cout << "What would you like to do?: "
    std::cin.getline(command);

    // handle the command
  }

  return 0;
}
```

Example input

```
create 101 400
create 307 180
add 101 404
add 307 9
cancel 101 200
delete 101
quit
```

User prompts were omitted in the output

Example output

```
No flights in the system

Flight 101 : 0/400 (0%) seats reserved

Flight 101 : 0/400 (0%) seats reserved
Flight 307 : 0/180 (0%) seats reserved

Flight 101 : 404/400 (101%) seats reserved
Flight 307 : 0/180 (0%) seats reserved

Flight 101 : 404/400 (101%) seats reserved
Flight 307 : 9/180 (5%) seats reserved

Flight 101 : 204/400 (51%) seats reserved
Flight 307 : 9/180 (5%) seats reserved

Flight 307 : 9/180 (5%) seats reserved
```

Example input

```
add 101 404
create 101 400
add 101 500
delete 101
cancel 101 200
delete 101
quit
```

User prompts were omitted in the output

Example output

```
No flights in the system

Cannot perform this operation: flight 101 not found
No flights in the system

Flight 101 : 0/400 (0%) seats reserved

Cannot perform this operation: capacity reached
Flight 101 : 0/400 (0%) seats reserved

No flights in the system

Cannot perform this operation: flight 101 not found
No flights in the system

Cannot perform this operation: flight 101 not found
No flights in the system
```