XIAMEN UNIVERSITY MALAYSIA

Course Code	:	CST209
Course Name	:	Object-Oriented Programming-C++
Lecturer	:	Geetha Kanaparan
Academic Session	:	2024/09
Assessment Title	:	Group Assignment
Submission Due Date	:	29 November 2024 (Friday), before 11.59 pm.

Prepared by :

Student ID	Student Name
CYS2309202	Sun Zhengwu
PHY2209508	Wang Yibo
CYS2309193	Chen Junfeng

Date Received :

Feedback from Lecturer:

Mark:

Own Work Declaration

I/We acknowledge that my/our work will be examined for plagiarism or any other form of academic misconduct, and that the digital copy of the work may be retained for future comparisons.

I/We confirm that all sources cited in this work have been correctly acknowledged and that I/we fully understand the serious consequences of any intentional or unintentional academic misconduct.

In addition, I/we affirm that this submission does not contain any materials generated by AI tools, including direct copying and pasting of text or paraphrasing. This work is my/our original creation, and it has not been based on any work of other students (past or present), nor has it been previously submitted to any other course or institution.

Signature:  Wang YiBo

Date:27/11/2024

1. Content

2. C++ CODE

```
#include <iostream>
#include <string>
#include <vector>
#include <unordered_map>
#include <memory>  // For smart pointers
#include <algorithm>
#include <limits>
#include <iomanip>
#include <sstream>

using namespace std;

// ================== Data Structures ==================
```

// Map of countries to their cities and corresponding postal codes

```
unordered_map<string, unordered_map<string, string>>
postalCodeMap = {
    {"Singapore", {{"Singapore", "159053"}}},
    {"Japan", {
        {"Tokyo", "100-0001"}, {"Osaka", "530-0001"},
{"Nagoya", "450-0001"},
        {"Yokohama", "220-0001"}, {"Sapporo", "060-0001"},
{"Fukuoka", "810-0001"},
        {"Kawasaki", "210-0001"}, {"Kyoto", "600-8001"},
{"Saitama", "330-0001"}, {"Kobe", "650-0001"}
    }},
    {"China", {
        {"Beijing", "100000"}, {"Shanghai", "200000"}, {"Hong
Kong", "999077"},
```

```
        {"Shenzhen", "518000"}, {"Chengdu", "610000"},
{"Luoyang", "471000"},
        {"Chongqing", "400000"}, {"Yantai", "264000"},
{"Fuzhou", "350000"}, {"Harbin", "150000"}
    }},
    {"Indonesia", {
        {"Jakarta", "10110"}, {"Bandung", "40111"},
{"Yogyakarta", "55111"},
        {"Medan", "20111"}, {"Malang", "65111"}, {"Makassar",
"90111"},
        {"Palembang", "30111"}, {"Semarang", "50111"},
{"Tangerang", "15111"}, {"Surabaya", "60111"}
    }},
    {"Malaysia", {
        {"Klang", "41000"}, {"Putrajaya", "62000"}, {"Sepang",
"43900"}, {"Subang Jaya", "47500"},
        {"George Town", "10000"}, {"Kuala Lumpur", "50000"},
{"Kota Kinabalu", "88000"},
        {"Malacca City", "75000"}, {"Johor Bahru", "80000"},
{"Petaling Jaya", "46000"}
    }},
    {"South Korea", {
        {"Seoul", "04524"}, {"Busan", "46000"}, {"Incheon",
"22310"},
        {"Daegu", "41150"}, {"Daejeon", "34100"}, {"Gwangju",
"61937"},
        {"Ulsan", "44787"}, {"Bucheon", "14519"}, {"Suwon",
"16490"}, {"Cheongju", "28501"}
    }}
};

// Map of city distances (in km) from each city to others
unordered_map<string, unordered_map<string, int>>
cityDistances = {
    {"Singapore", {
```

```
        {"Beijing", 5000}, {"Shanghai", 4000}, {"Hong Kong",
2600},
        {"Shenzhen", 2600}, {"Chengdu", 3300}, {"Luoyang",
3800},
        {"Chongqing", 3200}, {"Yantai", 4300}, {"Fuzhou",
3100},
        {"Harbin", 5400}, {"Jakarta", 900}, {"Bandung", 1000},
        {"Yogyakarta", 1300}, {"Medan", 600}, {"Malang",
1300},
        {"Makassar", 1500}, {"Palembang", 600}, {"Semarang",
1200},
        {"Surabaya", 1400}, {"Tangerang", 880}, {"Klang",
300},
        {"Putrajaya", 300}, {"Sepang", 300}, {"Subang Jaya",
350},
        {"George Town", 700}, {"Kuala Lumpur", 350}, {"Kota
Kinabalu", 1400},
        {"Malacca City", 250}, {"Johor Bahru", 20}, {"Petaling
Jaya", 350},
        {"Seoul", 4600}, {"Busan", 4500}, {"Incheon", 4600},
        {"Daegu", 4500}, {"Daejeon", 4500}, {"Gwangju", 4500},
        {"Ulsan", 4500}, {"Bucheon", 4600}, {"Suwon", 4600},
        {"Cheongju", 4500}
    }},
    {"Hong Kong", {
        {"Singapore", 2600}, {"Tokyo", 2900}, {"Osaka", 2600},
{"Nagoya", 2600},
        {"Yokohama", 2900}, {"Sapporo", 3000}, {"Fukuoka",
2000},
        {"Kawasaki", 2900}, {"Kyoto", 2600}, {"Saitama",
2900},
        {"Kobe", 2600}, {"Beijing", 2000}, {"Shanghai", 1200},
        {"Shenzhen", 40}, {"Chengdu", 1500}, {"Luoyang",
1500},
        {"Chongqing", 1400}, {"Yantai", 1800}, {"Fuzhou",
700},
```

```
        {"Harbin", 2800}, {"Jakarta", 3200}, {"Bandung",
3100},
        {"Yogyakarta", 2900}, {"Medan", 2300}, {"Malang",
2800},
        {"Makassar", 3000}, {"Palembang", 2500}, {"Semarang",
2800},
        {"Surabaya", 2700}, {"Tangerang", 3000}, {"Klang",
2500},
        {"Putrajaya", 2500}, {"Sepang", 2500}, {"Subang Jaya",
2500},
        {"George Town", 2200}, {"Kuala Lumpur", 2500},
        {"Kota Kinabalu", 1800}, {"Malacca City", 2500},
        {"Johor Bahru", 2500}, {"Petaling Jaya", 2500},
        {"Seoul", 2100}, {"Busan", 2100}, {"Incheon", 2100},
        {"Daegu", 2100}, {"Daejeon", 2000}, {"Gwangju", 2100},
        {"Ulsan", 2100}, {"Bucheon", 2100}, {"Suwon", 2100},
        {"Cheongju", 2100}
    }},
    {"Shenzhen", {
        {"Singapore", 2600}, {"Tokyo", 3000}, {"Osaka", 2700},
{"Nagoya", 2700},
        {"Yokohama", 3000}, {"Sapporo", 3100}, {"Fukuoka",
2100},
        {"Kawasaki", 3000}, {"Kyoto", 2700}, {"Saitama",
3000},
        {"Kobe", 2700}, {"Beijing", 2100}, {"Shanghai", 1200},
        {"Hong Kong", 40}, {"Chengdu", 1300}, {"Luoyang",
1500},
        {"Chongqing", 1100}, {"Yantai", 1800}, {"Fuzhou",
700},
        {"Harbin", 2800}, {"Jakarta", 3200}, {"Bandung",
3100},
        {"Yogyakarta", 2900}, {"Medan", 2300}, {"Malang",
2800},
        {"Makassar", 3000}, {"Palembang", 2500}, {"Semarang",
2800},
```

```
        {"Surabaya", 2700}, {"Tangerang", 3000}, {"Klang",
2500},
        {"Putrajaya", 2500}, {"Sepang", 2500}, {"Subang Jaya",
2500},
        {"George Town", 2200}, {"Kuala Lumpur", 2500},
        {"Kota Kinabalu", 1800}, {"Malacca City", 2500},
        {"Johor Bahru", 2500}, {"Petaling Jaya", 2500},
        {"Seoul", 2100}, {"Busan", 2100}, {"Incheon", 2100},
        {"Daegu", 2100}, {"Daejeon", 2000}, {"Gwangju", 2100},
        {"Ulsan", 2100}, {"Bucheon", 2100}, {"Suwon", 2100},
        {"Cheongju", 2100}
    }},
    {"Chengdu", {
        {"Singapore", 3300}, {"Tokyo", 3300}, {"Osaka", 3000},
{"Nagoya", 3100},
        {"Yokohama", 3300}, {"Sapporo", 3500}, {"Fukuoka",
2300},
        {"Kawasaki", 3300}, {"Kyoto", 3000}, {"Saitama",
3300},
        {"Kobe", 3000}, {"Beijing", 1500}, {"Shanghai", 1700},
        {"Hong Kong", 1500}, {"Shenzhen", 1300}, {"Luoyang",
1000},
        {"Chongqing", 270}, {"Yantai", 1800}, {"Fuzhou",
1700},
        {"Harbin", 2800}, {"Jakarta", 4000}, {"Bandung",
3900},
        {"Yogyakarta", 3700}, {"Medan", 3100}, {"Malang",
3600},
        {"Makassar", 3800}, {"Palembang", 3300}, {"Semarang",
3600},
        {"Surabaya", 3500}, {"Tangerang", 3800}, {"Klang",
3200},
        {"Putrajaya", 3200}, {"Sepang", 3200}, {"Subang Jaya",
3200},
        {"George Town", 2900}, {"Kuala Lumpur", 3200},
        {"Kota Kinabalu", 2500}, {"Malacca City", 3200},
```

```
        {"Johor Bahru", 3200}, {"Petaling Jaya", 3200},
        {"Seoul", 2600}, {"Busan", 2700}, {"Incheon", 2600},
        {"Daegu", 2700}, {"Daejeon", 2600}, {"Gwangju", 2700},
        {"Ulsan", 2700}, {"Bucheon", 2600}, {"Suwon", 2600},
        {"Cheongju", 2600}
    }},
    {"Luoyang", {
        {"Singapore", 3800}, {"Tokyo", 2700}, {"Osaka", 2400},
{"Nagoya", 2400},
        {"Yokohama", 2700}, {"Sapporo", 2700}, {"Fukuoka",
1800},
        {"Kawasaki", 2700}, {"Kyoto", 2400}, {"Saitama",
2700},
        {"Kobe", 2400}, {"Beijing", 700}, {"Shanghai", 900},
        {"Hong Kong", 1500}, {"Shenzhen", 1500}, {"Chengdu",
1000},
        {"Chongqing", 800}, {"Yantai", 900}, {"Fuzhou", 1200},
        {"Harbin", 1900}, {"Jakarta", 4600}, {"Bandung",
4500},
        {"Yogyakarta", 4300}, {"Medan", 3700}, {"Malang",
4100},
        {"Makassar", 4300}, {"Palembang", 3800}, {"Semarang",
4100},
        {"Surabaya", 4000}, {"Tangerang", 4300}, {"Klang",
3800},
        {"Putrajaya", 3800}, {"Sepang", 3800}, {"Subang Jaya",
3800},
        {"George Town", 3500}, {"Kuala Lumpur", 3800},
        {"Kota Kinabalu", 3200}, {"Malacca City", 3800},
        {"Johor Bahru", 3800}, {"Petaling Jaya", 3800},
        {"Seoul", 1400}, {"Busan", 1500}, {"Incheon", 1400},
        {"Daegu", 1500}, {"Daejeon", 1400}, {"Gwangju", 1500},
        {"Ulsan", 1500}, {"Bucheon", 1400}, {"Suwon", 1400},
        {"Cheongju", 1400}
    }},
    {"Chongqing", {
```

```
        {"Singapore", 3200}, {"Tokyo", 3200}, {"Osaka", 3100},
{"Nagoya", 3200},
        {"Yokohama", 3200}, {"Sapporo", 3400}, {"Fukuoka",
2200},
        {"Kawasaki", 3200}, {"Kyoto", 3100}, {"Saitama",
3200},
        {"Kobe", 3100}, {"Beijing", 1400}, {"Shanghai", 1500},
        {"Hong Kong", 1400}, {"Shenzhen", 1100}, {"Chengdu",
270},
        {"Luoyang", 800}, {"Yantai", 1800}, {"Fuzhou", 1600},
        {"Harbin", 2800}, {"Jakarta", 4200}, {"Bandung",
4100},
        {"Yogyakarta", 3900}, {"Medan", 3300}, {"Malang",
3800},
        {"Makassar", 4000}, {"Palembang", 3500}, {"Semarang",
3800},
        {"Surabaya", 3700}, {"Tangerang", 4000}, {"Klang",
3700},
        {"Putrajaya", 3700}, {"Sepang", 3700}, {"Subang Jaya",
3700},
        {"George Town", 3400}, {"Kuala Lumpur", 3700},
        {"Kota Kinabalu", 3000}, {"Malacca City", 3700},
        {"Johor Bahru", 3700}, {"Petaling Jaya", 3700},
        {"Seoul", 2500}, {"Busan", 2600}, {"Incheon", 2500},
        {"Daegu", 2600}, {"Daejeon", 2500}, {"Gwangju", 2600},
        {"Ulsan", 2600}, {"Bucheon", 2500}, {"Suwon", 2500},
        {"Cheongju", 2500}
    }},
    {"Yantai", {
        {"Singapore", 4300}, {"Tokyo", 1900}, {"Osaka", 1700},
{"Nagoya", 1800},
        {"Yokohama", 1900}, {"Sapporo", 2000}, {"Fukuoka",
1200},
        {"Kawasaki", 1900}, {"Kyoto", 1700}, {"Saitama",
1900},
        {"Kobe", 1700}, {"Beijing", 700}, {"Shanghai", 700},
```

```
        {"Hong Kong", 1800}, {"Shenzhen", 1800}, {"Chengdu",
1800},
        {"Luoyang", 900}, {"Chongqing", 1800}, {"Fuzhou",
1000},
        {"Harbin", 1100}, {"Jakarta", 4500}, {"Bandung",
4400},
        {"Yogyakarta", 4200}, {"Medan", 3600}, {"Malang",
4100},
        {"Makassar", 4300}, {"Palembang", 3700}, {"Semarang",
4000},
        {"Surabaya", 3900}, {"Tangerang", 4200}, {"Klang",
3800},
        {"Putrajaya", 3800}, {"Sepang", 3800}, {"Subang Jaya",
3800},
        {"George Town", 3500}, {"Kuala Lumpur", 3800},
        {"Kota Kinabalu", 3200}, {"Malacca City", 3800},
        {"Johor Bahru", 3800}, {"Petaling Jaya", 3800},
        {"Seoul", 1000}, {"Busan", 1200}, {"Incheon", 1000},
        {"Daegu", 1200}, {"Daejeon", 1100}, {"Gwangju", 1300},
        {"Ulsan", 1200}, {"Bucheon", 1000}, {"Suwon", 1000},
        {"Cheongju", 1100}
    }},
    {"Fuzhou", {
        {"Singapore", 3100}, {"Tokyo", 2400}, {"Osaka", 2100},
{"Nagoya", 2200},
        {"Yokohama", 2400}, {"Sapporo", 2600}, {"Fukuoka",
1400},
        {"Kawasaki", 2400}, {"Kyoto", 2100}, {"Saitama",
2400},
        {"Kobe", 2100}, {"Beijing", 1600}, {"Shanghai", 600},
        {"Hong Kong", 700}, {"Shenzhen", 700}, {"Chengdu",
1700},
        {"Luoyang", 1200}, {"Chongqing", 1600}, {"Yantai",
1000},
        {"Harbin", 2200}, {"Jakarta", 3700}, {"Bandung",
3600},
```

```
        {"Yogyakarta", 3400}, {"Medan", 2800}, {"Malang",
3300},
        {"Makassar", 3500}, {"Palembang", 3000}, {"Semarang",
3300},
        {"Surabaya", 3200}, {"Tangerang", 3500}, {"Klang",
3100},
        {"Putrajaya", 3100}, {"Sepang", 3100}, {"Subang Jaya",
3100},
        {"George Town", 2800}, {"Kuala Lumpur", 3100},
        {"Kota Kinabalu", 2400}, {"Malacca City", 3100},
        {"Johor Bahru", 3100}, {"Petaling Jaya", 3100},
        {"Seoul", 1700}, {"Busan", 1800}, {"Incheon", 1700},
        {"Daegu", 1800}, {"Daejeon", 1700}, {"Gwangju", 1800},
        {"Ulsan", 1800}, {"Bucheon", 1700}, {"Suwon", 1700},
        {"Cheongju", 1700}
    }},
{"Harbin", {
        {"Singapore", 5400}, {"Tokyo", 1300}, {"Osaka", 1900},
{"Nagoya", 2000},
        {"Yokohama", 1300}, {"Sapporo", 1000}, {"Fukuoka",
1900},
        {"Kawasaki", 1300}, {"Kyoto", 1900}, {"Saitama",
1300},
        {"Kobe", 1900}, {"Beijing", 1000}, {"Shanghai", 2200},
        {"Hong Kong", 2800}, {"Shenzhen", 2800}, {"Chengdu",
2800},
        {"Luoyang", 1900}, {"Chongqing", 2800}, {"Yantai",
1100},
        {"Fuzhou", 2200}, {"Jakarta", 5300}, {"Bandung",
5200},
        {"Yogyakarta", 5000}, {"Medan", 4300}, {"Malang",
4800},
        {"Makassar", 5000}, {"Palembang", 4500}, {"Semarang",
4800},
        {"Surabaya", 4700}, {"Tangerang", 5000}, {"Klang",
4600},
```

```
        {"Putrajaya", 4600}, {"Sepang", 4600}, {"Subang Jaya",
4600},
        {"George Town", 4300}, {"Kuala Lumpur", 4600},
        {"Kota Kinabalu", 3900}, {"Malacca City", 4600},
        {"Johor Bahru", 4600}, {"Petaling Jaya", 4600},
        {"Seoul", 900}, {"Busan", 1100}, {"Incheon", 900},
        {"Daegu", 1100}, {"Daejeon", 1000}, {"Gwangju", 1200},
        {"Ulsan", 1100}, {"Bucheon", 900}, {"Suwon", 900},
        {"Cheongju", 1000}
    }},
    {"Jakarta", {
        {"Singapore", 900}, {"Tokyo", 5800}, {"Osaka", 5400},
{"Nagoya", 5500},
        {"Yokohama", 5800}, {"Sapporo", 5800}, {"Fukuoka",
4700},
        {"Kawasaki", 5800}, {"Kyoto", 5400}, {"Saitama",
5800},
        {"Kobe", 5400}, {"Beijing", 5500}, {"Shanghai", 4500},
        {"Hong Kong", 3200}, {"Shenzhen", 3200}, {"Chengdu",
4000},
        {"Luoyang", 4600}, {"Chongqing", 4200}, {"Yantai",
4500},
        {"Fuzhou", 3700}, {"Harbin", 5300}, {"Bandung", 150},
        {"Yogyakarta", 500}, {"Medan", 1400}, {"Malang", 700},
        {"Makassar", 1400}, {"Palembang", 600}, {"Semarang",
500},
        {"Surabaya", 700}, {"Tangerang", 30}, {"Klang", 1200},
        {"Putrajaya", 1200}, {"Sepang", 1200}, {"Subang Jaya",
1200},
        {"George Town", 1300}, {"Kuala Lumpur", 1200},
        {"Kota Kinabalu", 1800}, {"Malacca City", 1200},
        {"Johor Bahru", 1200}, {"Petaling Jaya", 1200},
        {"Seoul", 5300}, {"Busan", 5200}, {"Incheon", 5300},
        {"Daegu", 5200}, {"Daejeon", 5200}, {"Gwangju", 5200},
        {"Ulsan", 5200}, {"Bucheon", 5300}, {"Suwon", 5300},
        {"Cheongju", 5200}
```

```
    }},
    {"Bandung", {
        {"Singapore", 1000}, {"Tokyo", 5700}, {"Osaka", 5300},
{"Nagoya", 5400},
        {"Yokohama", 5700}, {"Sapporo", 5900}, {"Fukuoka",
4600},
        {"Kawasaki", 5700}, {"Kyoto", 5300}, {"Saitama",
5700},
        {"Kobe", 5300}, {"Beijing", 5400}, {"Shanghai", 4400},
        {"Hong Kong", 3100}, {"Shenzhen", 3100}, {"Chengdu",
3900},
        {"Luoyang", 4500}, {"Chongqing", 4100}, {"Yantai",
4400},
        {"Fuzhou", 3600}, {"Harbin", 5200}, {"Jakarta", 150},
        {"Yogyakarta", 500}, {"Medan", 1400}, {"Malang", 700},
        {"Makassar", 1400}, {"Palembang", 600}, {"Semarang",
500},
        {"Surabaya", 700}, {"Tangerang", 120}, {"Klang",
1200},
        {"Putrajaya", 1200}, {"Sepang", 1200}, {"Subang Jaya",
1200},
        {"George Town", 1300}, {"Kuala Lumpur", 1200},
        {"Kota Kinabalu", 1800}, {"Malacca City", 1200},
        {"Johor Bahru", 1200}, {"Petaling Jaya", 1200},
        {"Seoul", 5200}, {"Busan", 5100}, {"Incheon", 5200},
        {"Daegu", 5100}, {"Daejeon", 5100}, {"Gwangju", 5100},
        {"Ulsan", 5100}, {"Bucheon", 5200}, {"Suwon", 5200},
        {"Cheongju", 5100}
    }},
    {"Yogyakarta", {
        {"Singapore", 1300}, {"Tokyo", 5500}, {"Osaka", 5100},
{"Nagoya", 5200},
        {"Yokohama", 5500}, {"Sapporo", 5700}, {"Fukuoka",
4400},
        {"Kawasaki", 5500}, {"Kyoto", 5100}, {"Saitama",
5500},
```

```
        {"Kobe", 5100}, {"Beijing", 5200}, {"Shanghai", 4200},
        {"Hong Kong", 2900}, {"Shenzhen", 2900}, {"Chengdu",
3700},
        {"Luoyang", 4300}, {"Chongqing", 3900}, {"Yantai",
4200},
        {"Fuzhou", 3400}, {"Harbin", 5000}, {"Jakarta", 500},
        {"Bandung", 500}, {"Medan", 1700}, {"Malang", 300},
        {"Makassar", 1200}, {"Palembang", 1000}, {"Semarang",
200},
        {"Surabaya", 300}, {"Tangerang", 600}, {"Klang",
1500},
        {"Putrajaya", 1500}, {"Sepang", 1500}, {"Subang Jaya",
1500},
        {"George Town", 1600}, {"Kuala Lumpur", 1500},
        {"Kota Kinabalu", 2100}, {"Malacca City", 1500},
        {"Johor Bahru", 1500}, {"Petaling Jaya", 1500},
        {"Seoul", 5000}, {"Busan", 4900}, {"Incheon", 5000},
        {"Daegu", 4900}, {"Daejeon", 4900}, {"Gwangju", 4900},
        {"Ulsan", 4900}, {"Bucheon", 5000}, {"Suwon", 5000},
        {"Cheongju", 4900}
    }},
    {"Medan", {
        {"Singapore", 600}, {"Tokyo", 5000}, {"Osaka", 4700},
{"Nagoya", 4800},
        {"Yokohama", 5000}, {"Sapporo", 5200}, {"Fukuoka",
4000},
        {"Kawasaki", 5000}, {"Kyoto", 4700}, {"Saitama",
5000},
        {"Kobe", 4700}, {"Beijing", 4700}, {"Shanghai", 3600},
        {"Hong Kong", 2300}, {"Shenzhen", 2300}, {"Chengdu",
3100},
        {"Luoyang", 3700}, {"Chongqing", 3300}, {"Yantai",
3600},
        {"Fuzhou", 2800}, {"Harbin", 4300}, {"Jakarta", 1400},
        {"Bandung", 1400}, {"Yogyakarta", 1700}, {"Malang",
1800},
```

```
        {"Makassar", 2100}, {"Palembang", 1000}, {"Semarang",
1600},
        {"Surabaya", 1700}, {"Tangerang", 1400}, {"Klang",
600},
        {"Putrajaya", 600}, {"Sepang", 600}, {"Subang Jaya",
600},
        {"George Town", 400}, {"Kuala Lumpur", 600},
        {"Kota Kinabalu", 1100}, {"Malacca City", 600},
        {"Johor Bahru", 600}, {"Petaling Jaya", 600},
        {"Seoul", 4900}, {"Busan", 4800}, {"Incheon", 4900},
        {"Daegu", 4800}, {"Daejeon", 4800}, {"Gwangju", 4800},
        {"Ulsan", 4800}, {"Bucheon", 4900}, {"Suwon", 4900},
        {"Cheongju", 4800}
    }},
    {"Malang", {
        {"Singapore", 1400}, {"Tokyo", 5400}, {"Osaka", 5000},
{"Nagoya", 5100},
        {"Yokohama", 5400}, {"Sapporo", 5600}, {"Fukuoka",
4300},
        {"Kawasaki", 5400}, {"Kyoto", 5000}, {"Saitama",
5400},
        {"Kobe", 5000}, {"Beijing", 5100}, {"Shanghai", 4100},
        {"Hong Kong", 2800}, {"Shenzhen", 2800}, {"Chengdu",
3600},
        {"Luoyang", 4100}, {"Chongqing", 3800}, {"Yantai",
4100},
        {"Fuzhou", 3300}, {"Harbin", 4800}, {"Jakarta", 700},
        {"Bandung", 700}, {"Yogyakarta", 300}, {"Medan",
1800},
        {"Makassar", 1300}, {"Palembang", 900}, {"Semarang",
300},
        {"Surabaya", 100}, {"Tangerang", 800}, {"Klang",
1300},
        {"Putrajaya", 1300}, {"Sepang", 1300}, {"Subang Jaya",
1300},
        {"George Town", 1500}, {"Kuala Lumpur", 1300},
```

```
        {"Kota Kinabalu", 1900}, {"Malacca City", 1300},
        {"Johor Bahru", 1300}, {"Petaling Jaya", 1300},
        {"Seoul", 4900}, {"Busan", 4800}, {"Incheon", 4900},
        {"Daegu", 4800}, {"Daejeon", 4800}, {"Gwangju", 4800},
        {"Ulsan", 4800}, {"Bucheon", 4900}, {"Suwon", 4900},
        {"Cheongju", 4800}
    }},
    {"Makassar", {
        {"Singapore", 1500}, {"Tokyo", 5600}, {"Osaka", 5200},
{"Nagoya", 5300},
        {"Yokohama", 5600}, {"Sapporo", 5800}, {"Fukuoka",
4600},
        {"Kawasaki", 5600}, {"Kyoto", 5200}, {"Saitama",
5600},
        {"Kobe", 5200}, {"Beijing", 5300}, {"Shanghai", 4300},
        {"Hong Kong", 3000}, {"Shenzhen", 3000}, {"Chengdu",
3800},
        {"Luoyang", 4300}, {"Chongqing", 4000}, {"Yantai",
4300},
        {"Fuzhou", 3500}, {"Harbin", 5000}, {"Jakarta", 1400},
        {"Bandung", 1400}, {"Yogyakarta", 1200}, {"Medan",
2100},
        {"Malang", 1300}, {"Palembang", 1700}, {"Semarang",
1400},
        {"Surabaya", 1300}, {"Tangerang", 1600}, {"Klang",
2000},
        {"Putrajaya", 2000}, {"Sepang", 2000}, {"Subang Jaya",
2000},
        {"George Town", 2100}, {"Kuala Lumpur", 2000},
        {"Kota Kinabalu", 1400}, {"Malacca City", 2000},
        {"Johor Bahru", 2000}, {"Petaling Jaya", 2000},
        {"Seoul", 5200}, {"Busan", 5100}, {"Incheon", 5200},
        {"Daegu", 5100}, {"Daejeon", 5100}, {"Gwangju", 5100},
        {"Ulsan", 5100}, {"Bucheon", 5200}, {"Suwon", 5200},
        {"Cheongju", 5100}
    }},
```

```
    {"Palembang", {
        {"Singapore", 600}, {"Tokyo", 5100}, {"Osaka", 4800},
{"Nagoya", 4900},
        {"Yokohama", 5100}, {"Sapporo", 5300}, {"Fukuoka",
4100},
        {"Kawasaki", 5100}, {"Kyoto", 4800}, {"Saitama",
5100},
        {"Kobe", 4800}, {"Beijing", 4800}, {"Shanghai", 3700},
        {"Hong Kong", 2500}, {"Shenzhen", 2500}, {"Chengdu",
3300},
        {"Luoyang", 3800}, {"Chongqing", 3500}, {"Yantai",
3700},
        {"Fuzhou", 3000}, {"Harbin", 4500}, {"Jakarta", 600},
        {"Bandung", 600}, {"Yogyakarta", 1000}, {"Medan",
1000},
        {"Malang", 900}, {"Makassar", 1700}, {"Semarang",
1000},
        {"Surabaya", 1100}, {"Tangerang", 500}, {"Klang",
500},
        {"Putrajaya", 500}, {"Sepang", 500}, {"Subang Jaya",
500},
        {"George Town", 800}, {"Kuala Lumpur", 500},
        {"Kota Kinabalu", 1600}, {"Malacca City", 500},
        {"Johor Bahru", 500}, {"Petaling Jaya", 500},
        {"Seoul", 4900}, {"Busan", 4800}, {"Incheon", 4900},
        {"Daegu", 4800}, {"Daejeon", 4800}, {"Gwangju", 4800},
        {"Ulsan", 4800}, {"Bucheon", 4900}, {"Suwon", 4900},
        {"Cheongju", 4800}
    }},
    {"Semarang", {
        {"Singapore", 1200}, {"Tokyo", 5400}, {"Osaka", 5100},
{"Nagoya", 5200},
        {"Yokohama", 5400}, {"Sapporo", 5700}, {"Fukuoka",
4300},
        {"Kawasaki", 5400}, {"Kyoto", 5100}, {"Saitama",
5400},
```

```
        {"Kobe", 5100}, {"Beijing", 5100}, {"Shanghai", 4000},
        {"Hong Kong", 2800}, {"Shenzhen", 2800}, {"Chengdu",
3600},
        {"Luoyang", 4100}, {"Chongqing", 3800}, {"Yantai",
4000},
        {"Fuzhou", 3300}, {"Harbin", 4800}, {"Jakarta", 500},
        {"Bandung", 500}, {"Yogyakarta", 200}, {"Medan",
1600},
        {"Malang", 300}, {"Makassar", 1400}, {"Palembang",
1000},
        {"Surabaya", 400}, {"Tangerang", 600}, {"Klang",
1300},
        {"Putrajaya", 1300}, {"Sepang", 1300}, {"Subang Jaya",
1300},
        {"George Town", 1500}, {"Kuala Lumpur", 1300},
        {"Kota Kinabalu", 2100}, {"Malacca City", 1300},
        {"Johor Bahru", 1300}, {"Petaling Jaya", 1300},
        {"Seoul", 5100}, {"Busan", 5000}, {"Incheon", 5100},
        {"Daegu", 5000}, {"Daejeon", 5000}, {"Gwangju", 5000},
        {"Ulsan", 5000}, {"Bucheon", 5100}, {"Suwon", 5100},
        {"Cheongju", 5000}
    }},
    {"Surabaya", {
        {"Singapore", 1400}, {"Tokyo", 5300}, {"Osaka", 5000},
{"Nagoya", 5100},
        {"Yokohama", 5300}, {"Sapporo", 5600}, {"Fukuoka",
4200},
        {"Kawasaki", 5300}, {"Kyoto", 5000}, {"Saitama",
5300},
        {"Kobe", 5000}, {"Beijing", 5000}, {"Shanghai", 3900},
        {"Hong Kong", 2700}, {"Shenzhen", 2700}, {"Chengdu",
3500},
        {"Luoyang", 4000}, {"Chongqing", 3700}, {"Yantai",
3900},
        {"Fuzhou", 3200}, {"Harbin", 4700}, {"Jakarta", 700},
```

```
        {"Bandung", 700}, {"Yogyakarta", 300}, {"Medan",
1700},
        {"Malang", 100}, {"Makassar", 1300}, {"Palembang",
1100},
        {"Semarang", 400}, {"Tangerang", 800}, {"Klang",
1300},
        {"Putrajaya", 1300}, {"Sepang", 1300}, {"Subang Jaya",
1300},
        {"George Town", 1500}, {"Kuala Lumpur", 1300},
        {"Kota Kinabalu", 1900}, {"Malacca City", 1300},
        {"Johor Bahru", 1300}, {"Petaling Jaya", 1300},
        {"Seoul", 5000}, {"Busan", 4900}, {"Incheon", 5000},
        {"Daegu", 4900}, {"Daejeon", 4900}, {"Gwangju", 4900},
        {"Ulsan", 4900}, {"Bucheon", 5000}, {"Suwon", 5000},
        {"Cheongju", 4900}
    }},
    {"Tangerang", {
        {"Singapore", 880}, {"Tokyo", 5700}, {"Osaka", 5300},
{"Nagoya", 5400},
        {"Yokohama", 5700}, {"Sapporo", 5900}, {"Fukuoka",
4600},
        {"Kawasaki", 5700}, {"Kyoto", 5300}, {"Saitama",
5700},
        {"Kobe", 5300}, {"Beijing", 5400}, {"Shanghai", 4400},
        {"Hong Kong", 3000}, {"Shenzhen", 3000}, {"Chengdu",
3800},
        {"Luoyang", 4300}, {"Chongqing", 4000}, {"Yantai",
4400},
        {"Fuzhou", 3500}, {"Harbin", 5200}, {"Jakarta", 30},
        {"Bandung", 120}, {"Yogyakarta", 600}, {"Medan",
1400},
        {"Malang", 800}, {"Makassar", 1600}, {"Palembang",
500},
        {"Semarang", 600}, {"Surabaya", 800}, {"Klang", 1200},
        {"Putrajaya", 1200}, {"Sepang", 1200}, {"Subang Jaya",
1200},
```

```
        {"George Town", 1400}, {"Kuala Lumpur", 1200},
        {"Kota Kinabalu", 1800}, {"Malacca City", 1200},
        {"Johor Bahru", 1200}, {"Petaling Jaya", 1200},
        {"Seoul", 5200}, {"Busan", 5100}, {"Incheon", 5200},
        {"Daegu", 5100}, {"Daejeon", 5100}, {"Gwangju", 5100},
        {"Ulsan", 5100}, {"Bucheon", 5200}, {"Suwon", 5200},
        {"Cheongju", 5100}
    }},
    {"Klang", {
        {"Singapore", 300}, {"Tokyo", 5400}, {"Osaka", 5000},
{"Nagoya", 5100},
        {"Yokohama", 5400}, {"Sapporo", 5800}, {"Fukuoka",
4300},
        {"Kawasaki", 5400}, {"Kyoto", 5000}, {"Saitama",
5400},
        {"Kobe", 5000}, {"Beijing", 4600}, {"Shanghai", 3800},
        {"Hong Kong", 2500}, {"Shenzhen", 2500}, {"Chengdu",
3200},
        {"Luoyang", 3800}, {"Chongqing", 3700}, {"Yantai",
3800},
        {"Fuzhou", 3100}, {"Harbin", 4600}, {"Jakarta", 1200},
        {"Bandung", 1200}, {"Yogyakarta", 1500}, {"Medan",
600},
        {"Malang", 1300}, {"Makassar", 2000}, {"Palembang",
500},
        {"Semarang", 1300}, {"Surabaya", 1300}, {"Tangerang",
1200},
        {"Putrajaya", 30}, {"Sepang", 40}, {"Subang Jaya",
30},
        {"George Town", 300}, {"Kuala Lumpur", 40},
        {"Kota Kinabalu", 1600}, {"Malacca City", 150},
        {"Johor Bahru", 300}, {"Petaling Jaya", 30},
        {"Seoul", 5000}, {"Busan", 4900}, {"Incheon", 5000},
        {"Daegu", 4900}, {"Daejeon", 4900}, {"Gwangju", 4900},
        {"Ulsan", 4900}, {"Bucheon", 5000}, {"Suwon", 5000},
        {"Cheongju", 4900}
```

```
    }},
    {"Putrajaya", {
        {"Singapore", 300}, {"Tokyo", 5400}, {"Osaka", 5000},
{"Nagoya", 5100},
        {"Yokohama", 5400}, {"Sapporo", 5800}, {"Fukuoka",
4300},
        {"Kawasaki", 5400}, {"Kyoto", 5000}, {"Saitama",
5400},
        {"Kobe", 5000}, {"Beijing", 4600}, {"Shanghai", 3800},
        {"Hong Kong", 2500}, {"Shenzhen", 2500}, {"Chengdu",
3200},
        {"Luoyang", 3800}, {"Chongqing", 3700}, {"Yantai",
3800},
        {"Fuzhou", 3100}, {"Harbin", 4600}, {"Jakarta", 1200},
        {"Bandung", 1200}, {"Yogyakarta", 1500}, {"Medan",
600},
        {"Malang", 1300}, {"Makassar", 2000}, {"Palembang",
500},
        {"Semarang", 1300}, {"Surabaya", 1300}, {"Tangerang",
1200},
        {"Klang", 30}, {"Sepang", 30}, {"Subang Jaya", 40},
        {"George Town", 300}, {"Kuala Lumpur", 40},
        {"Kota Kinabalu", 1600}, {"Malacca City", 150},
        {"Johor Bahru", 300}, {"Petaling Jaya", 40},
        {"Seoul", 5000}, {"Busan", 4900}, {"Incheon", 5000},
        {"Daegu", 4900}, {"Daejeon", 4900}, {"Gwangju", 4900},
        {"Ulsan", 4900}, {"Bucheon", 5000}, {"Suwon", 5000},
        {"Cheongju", 4900}
    }},
    {"Sepang", {
        {"Singapore", 300}, {"Tokyo", 5400}, {"Osaka", 5000},
{"Nagoya", 5100},
        {"Yokohama", 5400}, {"Sapporo", 5800}, {"Fukuoka",
4300},
        {"Kawasaki", 5400}, {"Kyoto", 5000}, {"Saitama",
5400},
```

```
        {"Kobe", 5000}, {"Beijing", 4600}, {"Shanghai", 3800},
        {"Hong Kong", 2500}, {"Shenzhen", 2500}, {"Chengdu",
3200},
        {"Luoyang", 3800}, {"Chongqing", 3700}, {"Yantai",
3800},
        {"Fuzhou", 3100}, {"Harbin", 4600}, {"Jakarta", 1200},
        {"Bandung", 1200}, {"Yogyakarta", 1500}, {"Medan",
600},
        {"Malang", 1300}, {"Makassar", 2000}, {"Palembang",
500},
        {"Semarang", 1300}, {"Surabaya", 1300}, {"Tangerang",
1200},
        {"Klang", 40}, {"Putrajaya", 30}, {"Subang Jaya", 40},
        {"George Town", 300}, {"Kuala Lumpur", 50},
        {"Kota Kinabalu", 1600}, {"Malacca City", 150},
        {"Johor Bahru", 300}, {"Petaling Jaya", 50},
        {"Seoul", 5000}, {"Busan", 4900}, {"Incheon", 5000},
        {"Daegu", 4900}, {"Daejeon", 4900}, {"Gwangju", 4900},
        {"Ulsan", 4900}, {"Bucheon", 5000}, {"Suwon", 5000},
        {"Cheongju", 4900}
    }},
    {"Subang Jaya", {
        {"Singapore", 300}, {"Tokyo", 5400}, {"Osaka", 5000},
{"Nagoya", 5100},
        {"Yokohama", 5400}, {"Sapporo", 5800}, {"Fukuoka",
4300},
        {"Kawasaki", 5400}, {"Kyoto", 5000}, {"Saitama",
5400},
        {"Kobe", 5000}, {"Beijing", 4600}, {"Shanghai", 3800},
        {"Hong Kong", 2500}, {"Shenzhen", 2500}, {"Chengdu",
3200},
        {"Luoyang", 3800}, {"Chongqing", 3700}, {"Yantai",
3800},
        {"Fuzhou", 3100}, {"Harbin", 4600}, {"Jakarta", 1200},
        {"Bandung", 1200}, {"Yogyakarta", 1500}, {"Medan",
600},
```

```
        {"Malang", 1300}, {"Makassar", 2000}, {"Palembang",
500},
        {"Semarang", 1300}, {"Surabaya", 1300}, {"Tangerang",
1200},
        {"Klang", 30}, {"Putrajaya", 40}, {"Sepang", 40},
        {"George Town", 300}, {"Kuala Lumpur", 20},
        {"Kota Kinabalu", 1600}, {"Malacca City", 150},
        {"Johor Bahru", 300}, {"Petaling Jaya", 20},
        {"Seoul", 5000}, {"Busan", 4900}, {"Incheon", 5000},
        {"Daegu", 4900}, {"Daejeon", 4900}, {"Gwangju", 4900},
        {"Ulsan", 4900}, {"Bucheon", 5000}, {"Suwon", 5000},
        {"Cheongju", 4900}
    }},
    {"George Town", {
        {"Singapore", 700}, {"Tokyo", 5200}, {"Osaka", 4800},
{"Nagoya", 4900},
        {"Yokohama", 5200}, {"Sapporo", 5500}, {"Fukuoka",
4000},
        {"Kawasaki", 5200}, {"Kyoto", 4800}, {"Saitama",
5200},
        {"Kobe", 4800}, {"Beijing", 4400}, {"Shanghai", 3500},
        {"Hong Kong", 2200}, {"Shenzhen", 2200}, {"Chengdu",
2900},
        {"Luoyang", 3500}, {"Chongqing", 3400}, {"Yantai",
3500},
        {"Fuzhou", 2800}, {"Harbin", 4300}, {"Jakarta", 1300},
        {"Bandung", 1300}, {"Yogyakarta", 1600}, {"Medan",
400},
        {"Malang", 1500}, {"Makassar", 2100}, {"Palembang",
800},
        {"Semarang", 1500}, {"Surabaya", 1500}, {"Tangerang",
1400},
        {"Klang", 300}, {"Putrajaya", 300}, {"Sepang", 300},
        {"Subang Jaya", 300}, {"Kuala Lumpur", 300},
        {"Kota Kinabalu", 1000}, {"Malacca City", 500},
        {"Johor Bahru", 700}, {"Petaling Jaya", 300},
```

```
        {"Seoul", 4700}, {"Busan", 4600}, {"Incheon", 4700},
        {"Daegu", 4600}, {"Daejeon", 4600}, {"Gwangju", 4600},
        {"Ulsan", 4600}, {"Bucheon", 4700}, {"Suwon", 4700},
        {"Cheongju", 4600}
    }},
    {"Kuala Lumpur", {
        {"Singapore", 300}, {"Tokyo", 5400}, {"Osaka", 5000},
{"Nagoya", 5100},
        {"Yokohama", 5400}, {"Sapporo", 5800}, {"Fukuoka",
4300},
        {"Kawasaki", 5400}, {"Kyoto", 5000}, {"Saitama",
5400},
        {"Kobe", 5000}, {"Beijing", 4600}, {"Shanghai", 3800},
        {"Hong Kong", 2500}, {"Shenzhen", 2500}, {"Chengdu",
3200},
        {"Luoyang", 3800}, {"Chongqing", 3700}, {"Yantai",
3800},
        {"Fuzhou", 3100}, {"Harbin", 4600}, {"Jakarta", 1200},
        {"Bandung", 1200}, {"Yogyakarta", 1500}, {"Medan",
600},
        {"Malang", 1300}, {"Makassar", 2000}, {"Palembang",
500},
        {"Semarang", 1300}, {"Surabaya", 1300}, {"Tangerang",
1200},
        {"Klang", 40}, {"Putrajaya", 40}, {"Sepang", 40},
        {"Subang Jaya", 20}, {"George Town", 300},
        {"Kota Kinabalu", 1600}, {"Malacca City", 150},
        {"Johor Bahru", 300}, {"Petaling Jaya", 20},
        {"Seoul", 5000}, {"Busan", 4900}, {"Incheon", 5000},
        {"Daegu", 4900}, {"Daejeon", 4900}, {"Gwangju", 4900},
        {"Ulsan", 4900}, {"Bucheon", 5000}, {"Suwon", 5000},
        {"Cheongju", 4900}
    }},
    {"Kota Kinabalu", {
        {"Singapore", 1600}, {"Tokyo", 4600}, {"Osaka", 4200},
{"Nagoya", 4300},
```

```
        {"Yokohama", 4600}, {"Sapporo", 5100}, {"Fukuoka",
3600},
        {"Kawasaki", 4600}, {"Kyoto", 4200}, {"Saitama",
4600},
        {"Kobe", 4200}, {"Beijing", 3800}, {"Shanghai", 3100},
        {"Hong Kong", 1800}, {"Shenzhen", 1800}, {"Chengdu",
2500},
        {"Luoyang", 3200}, {"Chongqing", 3000}, {"Yantai",
3200},
        {"Fuzhou", 2400}, {"Harbin", 3900}, {"Jakarta", 1800},
        {"Bandung", 1800}, {"Yogyakarta", 2100}, {"Medan",
1100},
        {"Malang", 1900}, {"Makassar", 1400}, {"Palembang",
1600},
        {"Semarang", 2100}, {"Surabaya", 2000}, {"Tangerang",
1800},
        {"Klang", 1600}, {"Putrajaya", 1600}, {"Sepang",
1600},
        {"Subang Jaya", 1600}, {"George Town", 1000}, {"Kuala
Lumpur", 1600},
        {"Malacca City", 1600}, {"Johor Bahru", 1600},
{"Petaling Jaya", 1600},
        {"Seoul", 4700}, {"Busan", 4600}, {"Incheon", 4700},
        {"Daegu", 4600}, {"Daejeon", 4600}, {"Gwangju", 4600},
        {"Ulsan", 4600}, {"Bucheon", 4700}, {"Suwon", 4700},
        {"Cheongju", 4600}
    }},
    {"Malacca City", {
        {"Singapore", 200}, {"Tokyo", 5400}, {"Osaka", 5000},
{"Nagoya", 5100},
        {"Yokohama", 5400}, {"Sapporo", 5800}, {"Fukuoka",
4300},
        {"Kawasaki", 5400}, {"Kyoto", 5000}, {"Saitama",
5400},
        {"Kobe", 5000}, {"Beijing", 4600}, {"Shanghai", 3800},
```

```
        {"Hong Kong", 2500}, {"Shenzhen", 2500}, {"Chengdu",
3200},
        {"Luoyang", 3800}, {"Chongqing", 3700}, {"Yantai",
3800},
        {"Fuzhou", 3100}, {"Harbin", 4600}, {"Jakarta", 1200},
        {"Bandung", 1200}, {"Yogyakarta", 1500}, {"Medan",
600},
        {"Malang", 1300}, {"Makassar", 2000}, {"Palembang",
500},
        {"Semarang", 1300}, {"Surabaya", 1300}, {"Tangerang",
1200},
        {"Klang", 150}, {"Putrajaya", 150}, {"Sepang", 150},
        {"Subang Jaya", 150}, {"George Town", 500}, {"Kuala
Lumpur", 1500},
        {"Kota Kinabalu", 1600}, {"Johor Bahru", 200},
        {"Petaling Jaya", 150}, {"Seoul", 5000}, {"Busan",
4900},
        {"Incheon", 5000}, {"Daegu", 4900}, {"Daejeon", 4900},
        {"Gwangju", 4900}, {"Ulsan", 4900}, {"Bucheon", 5000},
        {"Suwon", 5000}, {"Cheongju", 4900}
    }},
    {"Johor Bahru", {
        {"Singapore", 30}, {"Tokyo", 5400}, {"Osaka", 5000},
{"Nagoya", 5100},
        {"Yokohama", 5400}, {"Sapporo", 5800}, {"Fukuoka",
4300},
        {"Kawasaki", 5400}, {"Kyoto", 5000}, {"Saitama",
5400},
        {"Kobe", 5000}, {"Beijing", 4600}, {"Shanghai", 3800},
        {"Hong Kong", 2500}, {"Shenzhen", 2500}, {"Chengdu",
3200},
        {"Luoyang", 3800}, {"Chongqing", 3700}, {"Yantai",
3800},
        {"Fuzhou", 3100}, {"Harbin", 4600}, {"Jakarta", 1200},
        {"Bandung", 1200}, {"Yogyakarta", 1500}, {"Medan",
600},
```

```
        {"Malang", 1300}, {"Makassar", 2000}, {"Palembang",
500},
        {"Semarang", 1300}, {"Surabaya", 1300}, {"Tangerang",
1200},
        {"Klang", 30}, {"Putrajaya", 40}, {"Sepang", 40},
        {"Subang Jaya", 40}, {"George Town", 300}, {"Kuala
Lumpur", 40},
        {"Kota Kinabalu", 1600}, {"Malacca City", 150},
        {"Petaling Jaya", 40}, {"Seoul", 5000}, {"Busan",
4900},
        {"Incheon", 5000}, {"Daegu", 4900}, {"Daejeon", 4900},
        {"Gwangju", 4900}, {"Ulsan", 4900}, {"Bucheon", 5000},
        {"Suwon", 5000}, {"Cheongju", 4900}
    }},
    {"Petaling Jaya", {
        {"Singapore", 350}, {"Tokyo", 5400}, {"Osaka", 5000},
{"Nagoya", 5100},
        {"Yokohama", 5400}, {"Sapporo", 5800}, {"Fukuoka",
4300},
        {"Kawasaki", 5400}, {"Kyoto", 5000}, {"Saitama",
5400},
        {"Kobe", 5000}, {"Beijing", 4600}, {"Shanghai", 3800},
        {"Hong Kong", 2500}, {"Shenzhen", 2500}, {"Chengdu",
3200},
        {"Luoyang", 3800}, {"Chongqing", 3700}, {"Yantai",
3800},
        {"Fuzhou", 3100}, {"Harbin", 4600}, {"Jakarta", 1200},
        {"Bandung", 1200}, {"Yogyakarta", 1500}, {"Medan",
600},
        {"Malang", 1300}, {"Makassar", 2000}, {"Palembang",
500},
        {"Semarang", 1300}, {"Surabaya", 1300}, {"Tangerang",
1200},
        {"Klang", 30}, {"Putrajaya", 40}, {"Sepang", 40},
        {"Subang Jaya", 20}, {"George Town", 300}, {"Kuala
Lumpur", 20},
```

```
        {"Kota Kinabalu", 1600}, {"Malacca City", 150},
        {"Johor Bahru", 300}, {"Seoul", 5000}, {"Busan",
4900},
        {"Incheon", 5000}, {"Daegu", 4900}, {"Daejeon", 4900},
        {"Gwangju", 4900}, {"Ulsan", 4900}, {"Bucheon", 5000},
        {"Suwon", 5000}, {"Cheongju", 4900}
    }},
{"Seoul", {
        {"Singapore", 4600}, {"Tokyo", 1160}, {"Osaka", 980},
{"Nagoya", 1000},
        {"Yokohama", 1160}, {"Sapporo", 1900}, {"Fukuoka",
700},
        {"Kawasaki", 1140}, {"Kyoto", 1000}, {"Saitama",
1160},
        {"Kobe", 1000}, {"Beijing", 950}, {"Shanghai", 1300},
        {"Hong Kong", 2100}, {"Shenzhen", 2100}, {"Chengdu",
2600},
        {"Luoyang", 1400}, {"Chongqing", 2500}, {"Yantai",
1000},
        {"Fuzhou", 1700}, {"Harbin", 900}, {"Jakarta", 5300},
        {"Bandung", 5200}, {"Yogyakarta", 5000}, {"Medan",
4700},
        {"Malang", 5100}, {"Makassar", 5200}, {"Palembang",
4800},
        {"Semarang", 5100}, {"Surabaya", 5000}, {"Tangerang",
5200},
        {"Klang", 5000}, {"Putrajaya", 5000}, {"Sepang",
5000},
        {"Subang Jaya", 5000}, {"George Town", 4700}, {"Kuala
Lumpur", 5000},
        {"Kota Kinabalu", 4700}, {"Malacca City", 5000},
{"Petaling Jaya",5000},
        {"Johor Bahru", 5000}, {"Busan", 325}, {"Incheon",
30},
        {"Daegu", 240}, {"Daejeon", 140}, {"Gwangju", 270},
        {"Ulsan", 330}, {"Bucheon", 20}, {"Suwon", 30},
```

```
        {"Cheongju", 120}
    }},
    {"Busan", {
        {"Singapore", 4900}, {"Tokyo", 980}, {"Osaka", 700},
{"Nagoya", 800},
        {"Yokohama", 980}, {"Sapporo", 1700}, {"Fukuoka",
200},
        {"Kawasaki", 980}, {"Kyoto", 720}, {"Saitama", 980},
        {"Kobe", 700}, {"Beijing", 1100}, {"Shanghai", 850},
        {"Hong Kong", 2100}, {"Shenzhen", 2100}, {"Chengdu",
2500},
        {"Luoyang", 1600}, {"Chongqing", 2400}, {"Yantai",
1000},
        {"Fuzhou", 1500}, {"Harbin", 1200}, {"Jakarta", 5200},
        {"Bandung", 5100}, {"Yogyakarta", 4900}, {"Medan",
4500},
        {"Malang", 5000}, {"Makassar", 5100}, {"Palembang",
4700},
        {"Semarang", 5000}, {"Surabaya", 4900}, {"Tangerang",
5100},
        {"Klang", 4900}, {"Putrajaya", 4900}, {"Sepang",
4900},
        {"Subang Jaya", 4900}, {"George Town", 4600}, {"Kuala
Lumpur", 4900},
        {"Kota Kinabalu", 4500}, {"Malacca City",
4900},{"Petaling Jaya",4900},
        {"Johor Bahru", 4900}, {"Seoul", 325}, {"Incheon",
350},
        {"Daegu", 100}, {"Daejeon", 200}, {"Gwangju", 220},
        {"Ulsan", 70}, {"Bucheon", 340}, {"Suwon", 300},
        {"Cheongju", 260}
    }},
    {"Daegu", {
        {"Singapore", 4600}, {"Tokyo", 1000}, {"Osaka", 800},
{"Nagoya", 900},
```

```
        {"Yokohama", 1000}, {"Sapporo", 1800}, {"Fukuoka",
300},
        {"Kawasaki", 1000}, {"Kyoto", 800}, {"Saitama", 1000},
        {"Kobe", 800}, {"Beijing", 1600}, {"Shanghai", 1000},
        {"Hong Kong", 2100}, {"Shenzhen", 2100}, {"Chengdu",
2500},
        {"Luoyang", 1700}, {"Chongqing", 2400}, {"Yantai",
1200},
        {"Fuzhou", 1600}, {"Harbin", 1400}, {"Jakarta", 5200},
        {"Bandung", 5100}, {"Yogyakarta", 4900}, {"Medan",
4500},
        {"Malang", 5000}, {"Makassar", 5100}, {"Palembang",
4700},
        {"Semarang", 5000}, {"Surabaya", 4900}, {"Tangerang",
5100},
        {"Klang", 4900}, {"Putrajaya", 4900}, {"Sepang",
4900},
        {"Subang Jaya", 4900}, {"George Town", 4600}, {"Kuala
Lumpur", 4900},
        {"Kota Kinabalu", 4500}, {"Malacca City",
4900},{"Petaling Jaya",4900},
        {"Johor Bahru", 4900}, {"Seoul", 240}, {"Busan", 100},
        {"Incheon", 240}, {"Daejeon", 130}, {"Gwangju", 170},
        {"Ulsan", 150}, {"Bucheon", 260}, {"Suwon", 230},
        {"Cheongju", 160}
    }},
    {"Daejeon", {
        {"Singapore", 4800}, {"Tokyo", 1080}, {"Osaka", 900},
{"Nagoya", 920},
        {"Yokohama", 1080}, {"Sapporo", 1800}, {"Fukuoka",
580},
        {"Kawasaki", 1080}, {"Kyoto", 900}, {"Saitama", 1080},
        {"Kobe", 900}, {"Beijing", 1400}, {"Shanghai", 1100},
        {"Hong Kong", 2000}, {"Shenzhen", 2000}, {"Chengdu",
2400},
```

```
        {"Luoyang", 1500}, {"Chongqing", 2300}, {"Yantai",
1100},
        {"Fuzhou", 1600}, {"Harbin", 1100}, {"Jakarta", 5100},
        {"Bandung", 5000}, {"Yogyakarta", 4800}, {"Medan",
4400},
        {"Malang", 4900}, {"Makassar", 5000}, {"Palembang",
4600},
        {"Semarang", 4900}, {"Surabaya", 4800}, {"Tangerang",
5000},
        {"Klang", 4800}, {"Putrajaya", 4800}, {"Sepang",
4800},
        {"Subang Jaya", 4800}, {"George Town", 4500}, {"Kuala
Lumpur", 4800},
        {"Kota Kinabalu", 4400}, {"Malacca City",
4800},{"Petaling Jaya",4900},
        {"Johor Bahru", 4800}, {"Seoul", 140}, {"Busan", 200},
        {"Incheon", 140}, {"Daegu", 130}, {"Gwangju", 150},
        {"Ulsan", 190}, {"Bucheon", 150}, {"Suwon", 100},
        {"Cheongju", 80}
    }},
    {"Gwangju", {
        {"Singapore", 4800}, {"Tokyo", 1250}, {"Osaka", 1020},
{"Nagoya", 1050},
        {"Yokohama", 1250}, {"Sapporo", 1900}, {"Fukuoka",
600},
        {"Kawasaki", 1250}, {"Kyoto", 1020}, {"Saitama",
1250},
        {"Kobe", 1020}, {"Beijing", 1300}, {"Shanghai", 1100},
        {"Hong Kong", 2000}, {"Shenzhen", 2000}, {"Chengdu",
2500},
        {"Luoyang", 1500}, {"Chongqing", 2400}, {"Yantai",
1100},
        {"Fuzhou", 1500}, {"Harbin", 1300}, {"Jakarta", 5100},
        {"Bandung", 5000}, {"Yogyakarta", 4800}, {"Medan",
4500},
```

```
        {"Malang", 4900}, {"Makassar", 5000}, {"Palembang",
4600},
        {"Semarang", 4900}, {"Surabaya", 4800}, {"Tangerang",
5000},
        {"Klang", 4800}, {"Putrajaya", 4800}, {"Sepang",
4800},
        {"Subang Jaya", 4800}, {"George Town", 4500}, {"Kuala
Lumpur", 4800},
        {"Kota Kinabalu", 4400}, {"Malacca City",
4800},{"Petaling Jaya",4900},
        {"Johor Bahru", 4800}, {"Seoul", 270}, {"Busan", 220},
        {"Incheon", 270}, {"Daegu", 170}, {"Daejeon", 150},
        {"Ulsan", 250}, {"Bucheon", 250}, {"Suwon", 240},
        {"Cheongju", 200}
    }},
    {"Ulsan", {
        {"Singapore", 4900}, {"Tokyo", 1000}, {"Osaka", 740},
{"Nagoya", 820},
        {"Yokohama", 1000}, {"Sapporo", 1800}, {"Fukuoka",
260},
        {"Kawasaki", 1000}, {"Kyoto", 760}, {"Saitama", 1000},
        {"Kobe", 740}, {"Beijing", 1100}, {"Shanghai", 870},
        {"Hong Kong", 2100}, {"Shenzhen", 2100}, {"Chengdu",
2500},
        {"Luoyang", 1700}, {"Chongqing", 2400}, {"Yantai",
1100},
        {"Fuzhou", 1600}, {"Harbin", 1200}, {"Jakarta", 5200},
        {"Bandung", 5100}, {"Yogyakarta", 4900}, {"Medan",
4500},
        {"Malang", 5000}, {"Makassar", 5100}, {"Palembang",
4700},
        {"Semarang", 5000}, {"Surabaya", 4900}, {"Tangerang",
5100},
        {"Klang", 4900}, {"Putrajaya", 4900}, {"Sepang",
4900},
```

```
        {"Subang Jaya", 4900}, {"George Town", 4600}, {"Kuala
Lumpur", 4900},
        {"Kota Kinabalu", 4500}, {"Malacca City",
4900},{"Petaling Jaya",4900},
        {"Johor Bahru", 4900}, {"Seoul", 330}, {"Busan", 70},
        {"Incheon", 330}, {"Daegu", 150}, {"Daejeon", 190},
        {"Gwangju", 250}, {"Bucheon", 340}, {"Suwon", 300},
        {"Cheongju", 260}
    }},
    {"Bucheon", {
        {"Singapore", 5000}, {"Tokyo", 1160}, {"Osaka", 980},
{"Nagoya", 1000},
        {"Yokohama", 1160}, {"Sapporo", 1900}, {"Fukuoka",
700},
        {"Kawasaki", 1160}, {"Kyoto", 980}, {"Saitama", 1160},
        {"Kobe", 980}, {"Beijing", 950}, {"Shanghai", 1300},
        {"Hong Kong", 2100}, {"Shenzhen", 2100}, {"Chengdu",
2600},
        {"Luoyang", 1400}, {"Chongqing", 2500}, {"Yantai",
1000},
        {"Fuzhou", 1700}, {"Harbin", 900}, {"Jakarta", 5300},
        {"Bandung", 5200}, {"Yogyakarta", 5000}, {"Medan",
4700},
        {"Malang", 5100}, {"Makassar", 5200}, {"Palembang",
4800},
        {"Semarang", 5100}, {"Surabaya", 5000}, {"Tangerang",
5200},
        {"Klang", 5000}, {"Putrajaya", 5000}, {"Sepang",
5000},
        {"Subang Jaya", 5000}, {"George Town", 4700}, {"Kuala
Lumpur", 5000},
        {"Kota Kinabalu", 4700}, {"Malacca City",
5000},{"Petaling Jaya",5000},
        {"Johor Bahru", 5000}, {"Seoul", 20}, {"Busan", 340},
        {"Incheon", 20}, {"Daegu", 260}, {"Daejeon", 150},
        {"Gwangju", 250}, {"Ulsan", 340}, {"Suwon", 20},
```

```
        {"Cheongju", 120}
    }},
    {"Suwon", {
        {"Singapore", 4990}, {"Tokyo", 1150}, {"Osaka", 970},
{"Nagoya", 990},
        {"Yokohama", 1150}, {"Sapporo", 1900}, {"Fukuoka",
690},
        {"Kawasaki", 1150}, {"Kyoto", 970}, {"Saitama", 1150},
        {"Kobe", 970}, {"Beijing", 950}, {"Shanghai", 1290},
        {"Hong Kong", 2090}, {"Shenzhen", 2090}, {"Chengdu",
2590},
        {"Luoyang", 1390}, {"Chongqing", 2490}, {"Yantai",
990},
        {"Fuzhou", 1690}, {"Harbin", 890}, {"Jakarta", 5290},
        {"Bandung", 5190}, {"Yogyakarta", 4990}, {"Medan",
4690},
        {"Malang", 5090}, {"Makassar", 5190}, {"Palembang",
4790},
        {"Semarang", 5090}, {"Surabaya", 4990}, {"Tangerang",
5190},
        {"Klang", 4990}, {"Putrajaya", 4990}, {"Sepang",
4990},
        {"Subang Jaya", 4990}, {"George Town", 4690}, {"Kuala
Lumpur", 4990},
        {"Kota Kinabalu", 4690}, {"Malacca City",
4990},{"Petaling Jaya",5000},
        {"Johor Bahru", 4990}, {"Seoul", 30}, {"Busan", 300},
        {"Incheon", 30}, {"Daegu", 230}, {"Daejeon", 100},
        {"Gwangju", 240}, {"Ulsan", 300}, {"Bucheon", 20},
        {"Cheongju", 110}
    }},
    {"Cheongju", {
        {"Singapore", 4990}, {"Tokyo", 1150}, {"Osaka", 970},
{"Nagoya", 990},
        {"Yokohama", 1150}, {"Sapporo", 1900}, {"Fukuoka",
690},
```

```
        {"Kawasaki", 1150}, {"Kyoto", 970}, {"Saitama", 1150},
        {"Kobe", 970}, {"Beijing", 950}, {"Shanghai", 1290},
        {"Hong Kong", 2090}, {"Shenzhen", 2090}, {"Chengdu",
2590},
        {"Luoyang", 1390}, {"Chongqing", 2490}, {"Yantai",
990},
        {"Fuzhou", 1690}, {"Harbin", 890}, {"Jakarta", 5290},
        {"Bandung", 5190}, {"Yogyakarta", 4990}, {"Medan",
4690},
        {"Malang", 5090}, {"Makassar", 5190}, {"Palembang",
4790},
        {"Semarang", 5090}, {"Surabaya", 4990}, {"Tangerang",
5190},
        {"Klang", 4990}, {"Putrajaya", 4990}, {"Sepang",
4990},
        {"Subang Jaya", 4990}, {"George Town", 4690}, {"Kuala
Lumpur", 4990},
        {"Kota Kinabalu", 4690}, {"Malacca City",
4990},{"Petaling Jaya",4900},
        {"Johor Bahru", 4990}, {"Seoul", 110}, {"Busan", 260},
        {"Incheon", 120}, {"Daegu", 160}, {"Daejeon", 80},
        {"Gwangju", 200}, {"Ulsan", 260}, {"Bucheon", 110},
        {"Suwon", 110}
    }}
};
```

//ClassDefinitions

```

```

// Address Class

```
class Address {
    string country;
    string city;
    string zipCode;
```

```
public:
    // Constructor to initialize Address
    Address(string country, string city, string zipCode)
        : country(country), city(city), zipCode(zipCode) {}

    // Display the full address
    void displayAddress() const {
        cout << city << ", " << country << " - " << zipCode <<
endl;
    }

    // Getters for address components
    string getCountry() const { return country; }
    string getCity() const { return city; }
    string getZipCode() const { return zipCode; }
};
```

// Client Hierarchy

```
// Base Class for Client
class Client {
protected:
    string name;
    string clientType;
public:
    // Constructor to initialize Client
    Client(string name, string type) : name(name),
clientType(type) {}

    // Virtual destructor for proper cleanup of derived
classes
    virtual ~Client() {}

    // Virtual method to display client details
```

```
    virtual void displayClientDetails() const {
        cout << "Client Name: " << name << "\n";
        cout << "Client Type: " << clientType << "\n";
    }

    // Getters for client information
    string getName() const { return name; }
    string getClientType() const { return clientType; }
};

// Derived class for Private Clients
class PrivateClient : public Client {
public:
    // Constructor initializing Client as Private
    PrivateClient(string name) : Client(name, "Private") {}
};

// Derived class for Business Clients
class BusinessClient : public Client {
public:
    // Constructor initializing Client as Business
    BusinessClient(string name) : Client(name, "Business") {}
};
```

// Cargo Hierarchy

```
// Base Class for Cargo
class Cargo {
public:
    string type;
    double weight;
    double length, width, height;

    // Constructor to initialize Cargo
```

```
    Cargo(string type, double weight, double length, double
width, double height)
        : type(type), weight(weight), length(length),
width(width), height(height) {}

    // Virtual destructor
    virtual ~Cargo() {}

    // Calculate the volume of the cargo
    virtual double getVolume() const {
        return length * width * height;
    }

    // Pure virtual method for cost calculation factor
    virtual double getCostFactor() const = 0;
};

// Derived class for Box Cargo
class Box : public Cargo {
public:
    // Constructor initializing Cargo as Box
    Box(double weight, double length, double width, double
height)
        : Cargo("Box", weight, length, width, height) {}

    // Override to provide cost factor for Box
    double getCostFactor() const override {
        return 1.0; // Cost factor for box type
    }
};

// Derived class for Document Cargo
class Document : public Cargo {
public:
    // Constructor initializing Cargo as Document
```

```
    Document(double weight, double length, double width,
double height)
        : Cargo("Document", weight, length, width, height) {}

    // Override to provide cost factor for Document
    double getCostFactor() const override {
        return 1.2; // Cost factor for document type
    }
};

// Derived class for Pallet Cargo
class Pallet : public Cargo {
public:
    // Constructor initializing Cargo as Pallet
    Pallet(double weight, double length, double width, double
height)
        : Cargo("Pallet", weight, length, width, height) {}

    // Override to provide cost factor for Pallet
    double getCostFactor() const override {
        return 1.3; // Cost factor for pallet type
    }
};

// Derived class for Fragile Cargo
class FragileCargo : public Cargo {
public:
    // Constructor initializing Cargo as Fragile
    FragileCargo(double weight, double length, double width,
double height)
        : Cargo("Fragile", weight, length, width, height) {}

    // Override to provide cost factor for Fragile Cargo
    double getCostFactor() const override {
        return 1.5; // Higher cost factor for fragile cargo
    }
```

```
};
```

// Shipment Hierarchy

```
// Base Class for Shipment
class Shipment {
protected:
    Address origin;
    Address destination;
    vector<shared_ptr<Cargo>> cargoList;
    int routeDistance;

public:
    // Constructor to initialize Shipment
    Shipment(Address origin, Address destination, int
distance)
        : origin(origin), destination(destination),
routeDistance(distance) {}

    // Virtual destructor
    virtual ~Shipment() {}

    // Add cargo to the shipment
    void addCargo(shared_ptr<Cargo> cargo) {
        cargoList.push_back(cargo);
    }

    // Pure virtual function to calculate total cost
    virtual double calculateTotalCost() const = 0;

    // Pure virtual function to get shipment type
    virtual string getShipmentType() const = 0;

    // Display shipment details
```

```
    virtual void displayShipmentDetails() const {
        cout << "\n================== Shipment Details
==================\n";
        cout << "Shipment Type: " << getShipmentType() <<
"\n";
        cout << "Shipment from: "; origin.displayAddress();
        cout << "Shipment to: "; destination.displayAddress();
        cout << "Total Distance: " << routeDistance << "
km\n";
        cout << "Number of Packages: " << cargoList.size() <<
"\n";
        for (const auto& cargo : cargoList) {
            cout << "- " << cargo->type << " | Weight: " <<
cargo->weight << " kg | Volume: " << cargo->getVolume() << "
cm^3\n";
        }
        cout << "Total Cost: RM" << fixed << setprecision(2)
<< calculateTotalCost() << "\n";
        cout <<
"=====================================================\n";
    }
};

// Derived class for Road Shipment
class RoadShipment : public Shipment {
public:
    // Constructor initializing Shipment as RoadShipment
    RoadShipment(Address origin, Address destination, int
distance)
        : Shipment(origin, destination, distance) {}

    // Calculate total cost for road shipment
    double calculateTotalCost() const override {
        double cost = 0.0;
        double baseCostFactor = 0.5; // Road shipment base
cost factor
```

```
        int distance = 0;
        try {
            distance =
cityDistances.at(origin.getCity()).at(destination.getCity());
        }
        catch (const out_of_range&) {
            cout << "Error: Distance not found between " <<
origin.getCity() << " and " << destination.getCity() << ".\n";
            return 0.0;
        }

        for (const auto& cargo : cargoList) {
            double volume = cargo->getVolume();
            double weight = cargo->weight;
            double cargoCostFactor = cargo->getCostFactor();
// Get cost factor based on cargo type

            // Calculate cost based on distance and cargo
volume
            if (distance <= 2000) {
                if (volume <= 0.05) {
                    cost += (0.4 * distance + 15 * weight +
500) * baseCostFactor * cargoCostFactor;
                }
                else {
                    cost += (0.5 * distance + 18 * weight +
600) * baseCostFactor * cargoCostFactor;
                }
            }
            else if (distance <= 4000) {
                if (volume <= 0.1) {
                    cost += (0.45 * distance + 16 * weight +
800) * baseCostFactor * cargoCostFactor;
                }
                else {
```

```
                    cost += (0.55 * distance + 20 * weight +
1000) * baseCostFactor * cargoCostFactor;
                }
            }
            else {
                cost += (0.55 * distance + 20 * weight + 1000)
* baseCostFactor * cargoCostFactor;
            }
        }
        return cost;
    }

    // Return the shipment type
    string getShipmentType() const override {
        return "Road";
    }
};

// Derived class for Air Shipment
class AirShipment : public Shipment {
public:
    // Constructor initializing Shipment as AirShipment
    AirShipment(Address origin, Address destination, int
distance)
        : Shipment(origin, destination, distance) {}

    // Calculate total cost for air shipment
    double calculateTotalCost() const override {
        double cost = 0.0;
        double baseCostFactor = 1.0; // Air shipment base cost
factor
        int distance = 0;
        try {
            distance =
cityDistances.at(origin.getCity()).at(destination.getCity());
        }
```

```
        catch (const out_of_range&) {
            cout << "Error: Distance not found between " <<
origin.getCity() << " and " << destination.getCity() << ".\n";
            return 0.0;
        }

        for (const auto& cargo : cargoList) {
            double volume = cargo->getVolume();
            double weight = cargo->weight;
            double cargoCostFactor = cargo->getCostFactor();
// Get cost factor based on cargo type

            // Calculate cost based on distance and cargo
volume
            if (distance <= 2000) {
                if (volume <= 0.05) {
                    cost += (0.4 * distance + 15 * weight +
500) * baseCostFactor * cargoCostFactor;
                }
                else {
                    cost += (0.5 * distance + 18 * weight +
600) * baseCostFactor * cargoCostFactor;
                }
            }
            else if (distance <= 4000) {
                if (volume <= 0.1) {
                    cost += (0.45 * distance + 16 * weight +
800) * baseCostFactor * cargoCostFactor;
                }
                else {
                    cost += (0.55 * distance + 20 * weight +
1000) * baseCostFactor * cargoCostFactor;
                }
            }
            else {
```

```
                cost += (0.55 * distance + 20 * weight + 1000)
* baseCostFactor * cargoCostFactor;
            }
        }
        return cost;
    }

    // Return the shipment type
    string getShipmentType() const override {
        return "Air";
    }
};

// Derived class for Sea Shipment
class SeaShipment : public Shipment {
public:
    // Constructor initializing Shipment as SeaShipment
    SeaShipment(Address origin, Address destination, int
distance)
        : Shipment(origin, destination, distance) {}

    // Calculate total cost for sea shipment
    double calculateTotalCost() const override {
        double cost = 0.0;
        double baseCostFactor = 0.7; // Sea shipment base cost
factor
        int distance = 0;
        try {
            distance =
cityDistances.at(origin.getCity()).at(destination.getCity());
        }
        catch (const out_of_range&) {
            cout << "Error: Distance not found between " <<
origin.getCity() << " and " << destination.getCity() << ".\n";
            return 0.0;
        }
```

```
        for (const auto& cargo : cargoList) {
            double volume = cargo->getVolume();
            double weight = cargo->weight;
            double cargoCostFactor = cargo->getCostFactor();
// Get cost factor based on cargo type

            // Calculate cost based on distance and cargo
volume
            if (distance <= 2000) {
                if (volume <= 0.05) {
                    cost += (0.4 * distance + 15 * weight +
500) * baseCostFactor * cargoCostFactor;
                }
                else {
                    cost += (0.5 * distance + 18 * weight +
600) * baseCostFactor * cargoCostFactor;
                }
            }
            else if (distance <= 4000) {
                if (volume <= 0.1) {
                    cost += (0.45 * distance + 16 * weight +
800) * baseCostFactor * cargoCostFactor;
                }
                else {
                    cost += (0.55 * distance + 20 * weight +
1000) * baseCostFactor * cargoCostFactor;
                }
            }
            else {
                cost += (0.55 * distance + 20 * weight + 1000)
* baseCostFactor * cargoCostFactor;
            }
        }
        return cost;
    }
```

```
    // Return the shipment type
    string getShipmentType() const override {
        return "Sea";
    }
};
```

// Payment Interface and Implementations

```
// Payment Interface
class IPayment {
public:
    // Pure virtual method to process payment
    virtual void processPayment() const = 0;
    virtual ~IPayment() {}
};

// Derived class for Credit Card Payment
class CreditCardPayment : public IPayment {
private:
    double amount;
    string cardNumber;
public:
    // Constructor to initialize CreditCardPayment
    CreditCardPayment(double amt, string cardNum) :
amount(amt), cardNumber(cardNum) {}

    // Process credit card payment
    void processPayment() const override {
        cout << "Processing credit card payment of RM" <<
amount << " using card number " << cardNumber << ".\n";
    }
};

// Derived class for E-Wallet Payment
class EWalletPayment : public IPayment {
```

```
private:
    double amount;
    string walletID;
public:
    // Constructor to initialize EWalletPayment
    EWalletPayment(double amt, string walletID) : amount(amt),
walletID(walletID) {}

    // Process e-wallet payment
    void processPayment() const override {
        cout << "Processing e-wallet payment of RM" << amount
<< " using wallet ID " << walletID << ".\n";
    }
};

// Derived class for Bank Transfer Payment
class BankTransferPayment : public IPayment {
private:
    double amount;
    string accountNumber;
public:
    // Constructor to initialize BankTransferPayment
    BankTransferPayment(double amt, string accNum) :
amount(amt), accountNumber(accNum) {}

    // Process bank transfer payment
    void processPayment() const override {
        cout << "Processing bank transfer payment of RM" <<
amount << " to account " << accountNumber << ".\n";
    }
};
```

// Function Prototypes

```
void displayMainMenu();
void displayUserTypeMenu();
```

```
string selectCountry(const string& context);
string selectCity(const string& country, const string&
context);

void displayPackageDetailsMenu();
void displayTransportModeMenu();
void displayPaymentMenu();
int getValidIntegerInput();
```

// Postal Code Validation

```
// Verify if the provided postal code is valid for the given
country and city
bool isPostalCodeValid(const string& country, const string&
city, const string& postalCode) {
    auto countryIt = postalCodeMap.find(country);
    if (countryIt != postalCodeMap.end()) {
        auto cityIt = countryIt->second.find(city);
        if (cityIt != countryIt->second.end()) {
            return cityIt->second == postalCode; // Check if
postal code matches
        }
    }
    return false; // Country or city not found
}

// Prompt user to enter a valid postal code
string getValidPostalCode(const string& country, const string&
city) {
    string postalCode;
    while (true) {
        cout << "Enter the postal code for " << city << ", "
<< country << " (or enter 0 to return): ";
        cin >> postalCode;
```

```
        if (postalCode == "0") {
            return ""; // Return empty string to indicate
going back
        }

        if (isPostalCodeValid(country, city, postalCode)) {
            cout << "Postal code is valid.\n";
            break;
        }
        else {
            // Provide the correct postal code for assistance
            string correctPostalCode =
postalCodeMap[country][city];
            cout << "Invalid postal code. The correct postal
code for " << city << ", " << country << " is "
                << correctPostalCode << ". Please try
again.\n";
        }
    }
    return postalCode;
}
```

// Main Function

```
int main() {
    bool exitProgram = false;
    string clientName;
    string originCountry, destinationCountry;
    string originCity, destinationCity;
    string originZipCode, destinationZipCode;
    int routeDistance = 0;
    vector<shared_ptr<Cargo>> cargoList;
    shared_ptr<Shipment> shipment = nullptr;

    while (!exitProgram) {
        displayMainMenu();
```

```
        int mainChoice = getValidIntegerInput();

        if (mainChoice == 1) {
            // ================== Get Quote Process
==================
            cout << "Enter your name: ";
            cin.ignore();
            getline(cin, clientName);

            // Choose user type (Private or Business)
            displayUserTypeMenu();
            int userTypeChoice = getValidIntegerInput();

            if (userTypeChoice == 3) continue; // Return to
main menu

            unique_ptr<Client> client = nullptr;
            double maxWeight = 0.0, maxLength = 0.0, maxWidth
= 0.0, maxHeight = 0.0;

            // Initialize client based on user type and set
cargo limits
            if (userTypeChoice == 1) {
                client =
make_unique<PrivateClient>(clientName);
                maxWeight = 70.0;
                maxLength = 120.0;
                maxWidth = 80.0;
                maxHeight = 80.0;
            }
            else if (userTypeChoice == 2) {
                client =
make_unique<BusinessClient>(clientName);
                maxWeight = 2500.0;
                maxLength = 302.0;
                maxWidth = 223.0;
```

```
                maxHeight = 220.0;
            }
            else {
                cout << "Invalid choice. Returning to main
menu.\n";
                continue;
            }

//===================CollectShipmentDetails====================
            bool shipmentDetailsCompleted = false;
            while (!shipmentDetailsCompleted) {
                // Origin selection
                bool originCompleted = false;
                while (!originCompleted) {
                    cout << "\n=== Origin Selection ===\n";
                    originCountry = selectCountry("origin");
                    if (originCountry.empty()) {
                        // Return to main menu
                        break; // Breaks out of origin
selection loop
                    }

                    originCity = selectCity(originCountry,
"origin");
                    if (originCity.empty()) {
                        // Return to origin country selection
                        continue; // Repeats origin country
selection
                    }

                    originZipCode =
getValidPostalCode(originCountry, originCity);
                    if (originZipCode.empty()) {
                        // Return to origin city selection
                        continue; // Repeats origin city
selection
```

```
                    }

                    originCompleted = true;
                }

                if (!originCompleted) {
                    // User chose to return to main menu
                    break; // Exits shipment details loop,
returns to main menu
                }

                // Destination selection
                bool destinationCompleted = false;
                while (!destinationCompleted) {
                    cout << "\n=== Destination Selection
===\n";
                    destinationCountry =
selectCountry("destination");
                    if (destinationCountry.empty()) {
                        // Return to origin selection
                        originCompleted = false;
                        break; // Breaks out to re-select
origin
                    }

                    destinationCity =
selectCity(destinationCountry, "destination");
                    if (destinationCity.empty()) {
                        // Return to destination country
selection
                        continue; // Repeats destination
country selection
                    }

                    destinationZipCode =
getValidPostalCode(destinationCountry, destinationCity);
```

```
                    if (destinationZipCode.empty()) {
                        // Return to destination city
selection
                        continue; // Repeats destination city
selection
                    }

                    destinationCompleted = true;
                }

                if (!originCompleted) {
                    // User went back to origin selection
                    continue; // Repeats the shipment details
loop
                }

                if (!destinationCompleted) {
// User chose to return to origin selection
                continue; // Repeats the shipment details loop
                }

                // Check if origin and destination are the
same
                if (originCountry == destinationCountry &&
originCity == destinationCity) {
                    cout << "Error: Destination cannot be the
same as origin. Please select a different destination.\n";
                    // Go back to destination selection
                    continue; // Repeats the shipment details
loop
                }

                // Distance lookup
                try {
                    routeDistance =
cityDistances.at(originCity).at(destinationCity);
```

```
                }
                catch (const out_of_range&) {
                    cout << "Error: Distance not found between
" << originCity << " and " << destinationCity << ".\n";
                    // Go back to destination selection
                    continue; // Repeats the shipment details
loop
                }

                // All details collected successfully
                shipmentDetailsCompleted = true;
            }

            if (!shipmentDetailsCompleted) {
                // User returned to main menu
                continue;
            }

//   ====================   Adding   Cargo   ====================
//Reference: [1]
            bool addingCargo = true;
            while (addingCargo) {
                cout << "\n=== Cargo Limits ===\n";
                cout << "Maximum Weight: " << maxWeight << "
kg\n";
                cout << "Maximum Length: " << maxLength << "
cm\n";
                cout << "Maximum Width: " << maxWidth << "
cm\n";
                cout << "Maximum Height: " << maxHeight << "
cm\n";
                displayPackageDetailsMenu();

                int cargoTypeChoice;
                cout << "Select Cargo Type (1 - Document, 2 -
Box, 3 - Pallet, 4 - Fragile, 5 - Exit): ";
```

```
                cin >> cargoTypeChoice;

                // Validate cargo type choice
                if (cin.fail() || cargoTypeChoice < 1 ||
cargoTypeChoice > 5) {
                    cin.clear();

cin.ignore(numeric_limits<streamsize>::max(), '\n');
                    cout << "Invalid cargo type. Please select
a number between 1 and 5.\n";
                    continue;
                }

                // Check if the user wants to exit the cargo
addition
                if (cargoTypeChoice == 5) {
                    if (cargoList.empty()) {
                        cout << "You must add at least one
cargo before exiting. Please select a cargo type.\n";
                        continue; // Re-prompt the user to
select a cargo type
                    }
                    else {
                        cout << "Exiting cargo addition...\n";
                        break; // Exit the cargo addition loop
                    }
                }

                double weight, length, width, height;

                // Validate weight input
                do {
                    cout << "Enter Weight (kg): ";
                    cin >> weight;
                    if (cin.fail() || weight <= 0) {
                        cin.clear();
```

```
cin.ignore(numeric_limits<streamsize>::max(), '\n');
                        cout << "Invalid weight. Please enter
a positive number.\n";
                        continue;
                    }
                    if (weight > maxWeight) {
                        cout << "Weight exceeds limit (" <<
maxWeight << " kg). Please try again.\n";
                        continue;
                    }
                    break;
                } while (true);

                // Validate dimensions input
                do {
                    cout << "Enter Length, Width, Height (in
cm, separated by commas): ";
                    string dimensions;
                    cin >> ws; // Clear leading whitespace
                    getline(cin, dimensions); // Read the
entire line

                    // Parse the input dimensions
                    stringstream ss(dimensions);
                    string token;
                    vector<double> values;

                    while (getline(ss, token, ',')) {
                        try {
                            double value = stod(token);
                            values.push_back(value);
                        }
                        catch (...) {
                            cout << "Invalid dimensions
format. Please enter valid numbers.\n";
```

```
                            values.clear(); // Clear invalid
data
                            break;
                        }
                    }

                    // Ensure exactly three dimensions are
provided
                    if (values.size() != 3) {
                        cout << "Invalid dimensions. Please
enter exactly three values separated by commas.\n";
                        continue;
                    }

                    length = values[0];
                    width = values[1];
                    height = values[2];

                    // Check if dimensions are positive and
within limits
                    if (length <= 0 || width <= 0 || height <=
0) {
                        cout << "Invalid dimensions. Please
enter positive numbers.\n";
                        continue;
                    }
                    if (length > maxLength || width > maxWidth
|| height > maxHeight) {
                        cout << "Dimensions exceed limits ("
                            << "Length: " << maxLength << "
cm, "
                            << "Width: " << maxWidth << " cm,
"
                            << "Height: " << maxHeight << "
cm). Please try again.\n";
                        continue;
```

```
                    }

                    break; // Valid input received
                } while (true);

                // Create and store the cargo object based on
selected type
                if (cargoTypeChoice == 1) {

cargoList.push_back(make_shared<Document>(weight, length,
width, height));
                }
                else if (cargoTypeChoice == 2) {

cargoList.push_back(make_shared<Box>(weight, length, width,
height));
                }
                else if (cargoTypeChoice == 3) {

cargoList.push_back(make_shared<Pallet>(weight, length, width,
height));
                }
                else if (cargoTypeChoice == 4) {

cargoList.push_back(make_shared<FragileCargo>(weight, length,
width, height));
                }

                cout << "Cargo added successfully!\n";
                cout << "Do you want to add more cargo?
(yes/no): ";
                string addMore;
                cin >> addMore;
                transform(addMore.begin(), addMore.end(),
addMore.begin(), ::tolower);
                if (addMore == "no") {
```

```
                    addingCargo = false;
                }
            }

//====================TransportModeSelection====================
            displayTransportModeMenu();
            int transportChoice = getValidIntegerInput();
            if (transportChoice == 4) continue; // Return to
main menu

            // Create shipment object based on selected
transport mode
            if (transportChoice == 1) {
                shipment =
make_shared<RoadShipment>(Address(originCountry, originCity,
originZipCode), Address(destinationCountry, destinationCity,
destinationZipCode), routeDistance);
            }
            else if (transportChoice == 2) {
                shipment =
make_shared<SeaShipment>(Address(originCountry, originCity,
originZipCode), Address(destinationCountry, destinationCity,
destinationZipCode), routeDistance);
            }
            else if (transportChoice == 3) {
                shipment =
make_shared<AirShipment>(Address(originCountry, originCity,
originZipCode), Address(destinationCountry, destinationCity,
destinationZipCode), routeDistance);
            }
            else {
                cout << "Invalid transport type. Returning to
main menu.\n";
                continue;
            }
```

```
            // Add cargo to shipment
            for (const auto& cargo : cargoList) {
                shipment->addCargo(cargo);
            }

            // Display shipment details
            shipment->displayShipmentDetails();

// ================== Payment Confirmation==================
//Reference: [2]
            cout << "Do you want to proceed with payment?
(yes/no): ";
            string confirm;
            cin >> confirm;
            transform(confirm.begin(), confirm.end(),
confirm.begin(), ::tolower);
            if (confirm == "yes") {
                double quote = shipment->calculateTotalCost();
                displayPaymentMenu();
                shared_ptr<IPayment> payment = nullptr;
                int paymentChoice = getValidIntegerInput();
                if (paymentChoice == 1) {
                    string cardNumber;
                    cout << "Enter Credit Card Number: ";
                    cin >> cardNumber;
                    payment =
make_shared<CreditCardPayment>(quote, cardNumber);
                }
                else if (paymentChoice == 2) {
                    string walletID;
                    cout << "Enter E-Wallet ID: ";
                    cin >> walletID;
                    payment =
make_shared<EWalletPayment>(quote, walletID);
                }
                else if (paymentChoice == 3) {
```

```
                    string accountNumber;
                    cout << "Enter Bank Account Number: ";
                    cin >> accountNumber;
                    payment =
make_shared<BankTransferPayment>(quote, accountNumber);
                }
                if (payment) {
                    payment->processPayment();
                    // Display client information after
successful payment
                    cout << "\n===== Payment Successful!
=====\n";
                    cout << "Client Information\n";
                    client->displayClientDetails();
                    cout <<
"================================\n";
                }
                else {
                    cout << "Invalid payment method. Payment
not processed.\n";
                }
            }

            // Clear cargo list for next use
            cargoList.clear();
        }
        else if (mainChoice == 2) {
            // ================== Exit Program
==================
            exitProgram = true;
            cout << "Thank you for using Aseana Parcels.
Goodbye!\n";
        }
        else {
            // Handle invalid main menu choice
            cout << "Invalid choice. Please try again.\n";
```

```
        }
    }

    return 0;
}
```

// Utility Functions

```
// Display the main menu options
void displayMainMenu() {
    cout << "\n================== Main Menu
==================\n";
    cout << "1. Get a Quote\n";
    cout << "2. Exit\n";
    cout << "Please select an option: ";
}

// Display the user type selection menu
void displayUserTypeMenu() {
    cout << "\n================ User Type Menu
================\n";
    cout << "1. Private User\n";
    cout << "2. Business User\n";
    cout << "3. Return to Main Menu\n";
    cout << "Please select user type: ";
}

// Allow user to select a country for origin or destination
string selectCountry(const string& context) {
    vector<string> countries = { "China", "Malaysia",
"Indonesia", "Japan", "South Korea", "Singapore" };
    while (true) {
```

```
        cout << "\n========== " << (context == "origin" ?
"Origin" : "Destination") << " Country Selection
==========\n";
        for (int i = 0; i < countries.size(); ++i) {
            cout << i + 1 << ". " << countries[i] << "\n";
        }
        cout << "0. Return to Previous Menu\n";
        cout << "Please select a country: ";

        int choice = getValidIntegerInput();

        if (choice >= 1 && choice <= countries.size()) {
            return countries[choice - 1];
        }
        else if (choice == 0) {
            // Return to previous menu
            return ""; // Indicate going back
        }
        else {
            cout << "Invalid choice. Please try again.\n";
        }
    }
}

// Allow user to select a city within the chosen country
//Reference: [3]
string selectCity(const string& country, const string&
context) {
    auto countryIt = postalCodeMap.find(country);
    if (countryIt == postalCodeMap.end()) {
        cout << "Error: Invalid country selected.\n";
        return "";
    }

    while (true) {
```

```
        cout << "\n=== " << (context == "origin" ? "Origin" :
"Destination") << " City Selection in " << country << "
===\n";

        const auto& cityList = countryIt->second;
        int index = 1;
        for (const auto& city : cityList) {
            cout << index++ << ". " << city.first << "\n";
        }

        cout << "0. Return to Previous Menu\n";
        cout << "Please select a city: ";

        int choice = getValidIntegerInput();

        if (choice >= 1 && choice <= cityList.size()) {
            auto it = cityList.begin();
            advance(it, choice - 1);
            cout << "\nYou selected city: " << it->first <<
"\n";
            return it->first;
        }
        else if (choice == 0) {
            // Return to previous menu
            return ""; // Indicate going back
        }
        else {
            cout << "Invalid choice. Please try again.\n";
        }
    }
}

// Display package details menu (currently just a header)
void displayPackageDetailsMenu() {
    cout << "\n=============== Package Details Menu
===============\n";
```

```
}

// Display transport mode selection menu
void displayTransportModeMenu() {
    vector<string> transportModes = { "Road", "Sea", "Air" };
    cout << "\n================ Transport Mode Menu
================\n";
    for (int i = 0; i < transportModes.size(); ++i) {
        cout << i + 1 << ". " << transportModes[i] << "\n";
    }
    cout << "4. Return to Main Menu\n";
    cout << "Please select mode of transport (1-3, or 4 to
return): ";
}

// Display payment method selection menu
void displayPaymentMenu() {
    cout << "\n================= Payment Menu
=================\n";
    cout << "1. Credit Card\n";
    cout << "2. E-Wallet\n";
    cout << "3. Bank Transfer\n";
    cout << "Please select payment method (1-3): ";
}

// Get a valid integer input from the user with error handling
//Reference: [4]
int getValidIntegerInput() {
    int input;
    while (true) {
        cin >> input;

        if (cin.fail()) {
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(),
'\n');
```

```
            cout << "Invalid choice. Please enter a number: ";
        }
        else {
            return input;
        }
    }
}
```

3. UML diagram and explanation

3.1 Explanation for design

In this project, our mission is to design a system for Aseana Parcels company for helping users to get a quote for the shipment they choose. Based on the background, we summarized the key point of this scenario.

There are diverse customer groups acting as users so that it will cause multiple cargo types and ways of shipment. After they get quotes, users can choose different payment method. Scalability and modularity of the system also need to be considered for OO programming.

In our group project, we primarily conducted analysis and implementation from the following aspects:

Aseana Parcels needs to provide customized services for different customer groups. Private customers focus more on the convenient transportation of small parcels, while corporate clients prefer efficient handling of bulk shipments. To achieve this, we adopted the concepts of inheritance and encapsulation in the design of the customer module. Through the abstract base class '`Client`', we unified the definition of common attributes of customers (such as name and customer type) and delegated the implementation of differentiated functionalities to the derived classes '`PrivateClient`' and '`BusinessClient`'.

At the same time, through encapsulation, the core information of the customer is hidden within the class, and it exposes only necessary interfaces (such as `displayClientDetails`), which ensures data security. The background mentions that different types of goods have varying impacts on transportation costs. To address this requirement, we designed an abstract class `Cargo` and implemented the specific processing logic for the various types.

Another important design is the application of polymorphism. In the `Cargo` class, we defined a pure virtual function `getCostFactor`, requiring derived classes to implement their own methods for calculating cost factors. For example, the cost factor

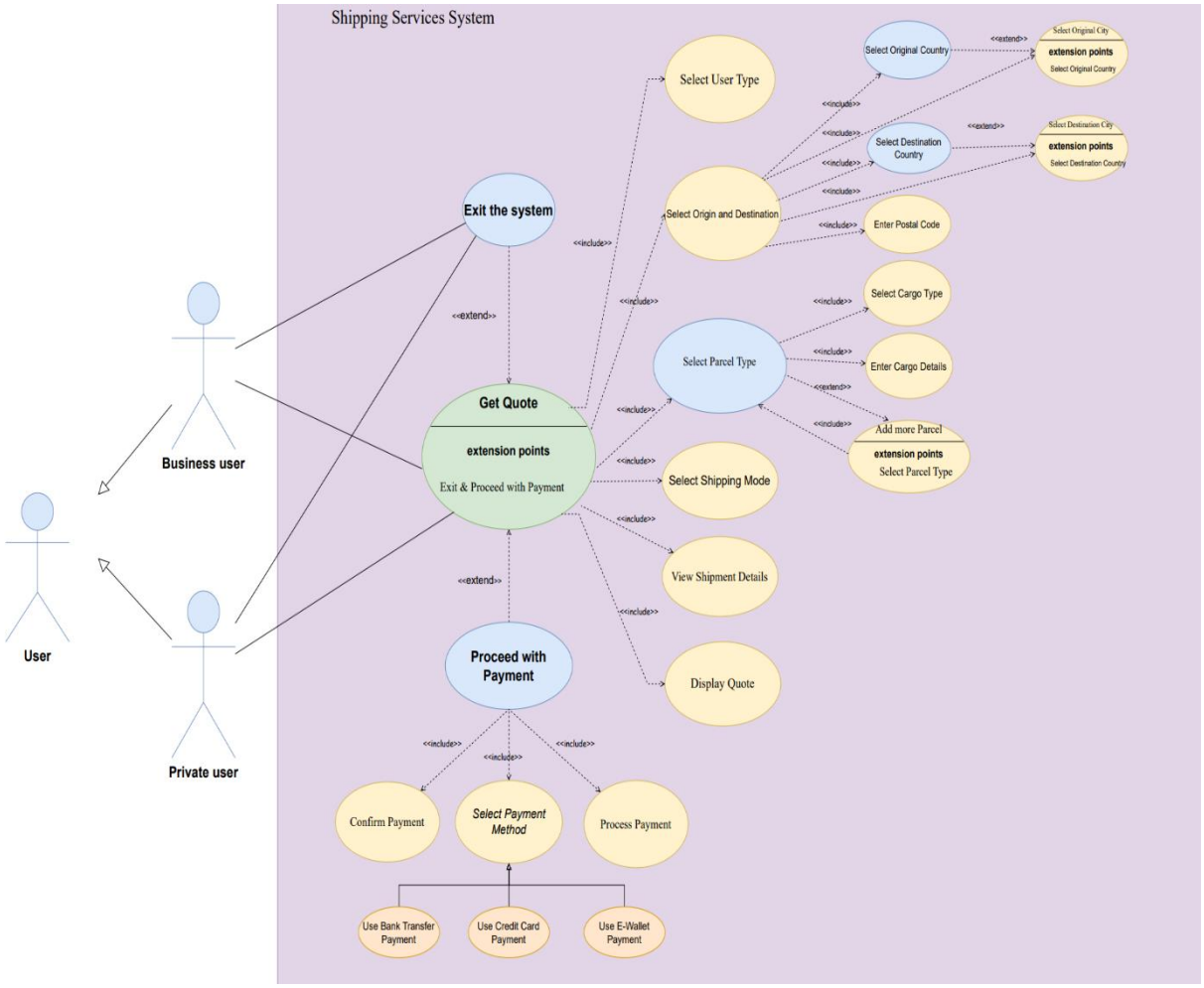
for the `Box` class is set to 1.00, whereas the `FragileCargo` class, due to its high-risk during Transportation, has a cost factor as high as 1.50.

The design of shipment methods is one of the core components of the system, which addresses the different cost calculation requirements mentioned in the background for land, sea, and air transportation. To achieve that, we designed an abstract class `Shipment`, encapsulating the common attributes of all transportation methods, such as the starting address, destination address, cargo list, and transportation distance. By defining pure virtual functions `calculateTotalCost` and `getShipmentType`, we create interfaces for subclasses to customize their transportation logic. Then, the logic for adding cargo is encapsulated within methods, allowing operations on the `cargoList` only through public interfaces, thus preventing errors caused by directly manipulating underlying data.

The design of the payment module also emphasizes flexibility and scalability, which directly caters to the different payment requirements mentioned in the background. We designed the `IPayment` interface to abstruct the payment behavior into the `processPayment` method. Specific payment methods are supported through implementation classes such as `CreditCardPayment`, `EWalletPayment`, and `BankTransferPayment`.

In the design process, we effectively applied object-oriented principles of encapsulation, inheritance, and polymorphism. Each module achieves high cohesion and low coupling while accurately reflecting Aseana Parcels' business requirements to ensure both current functionality and future scalability.

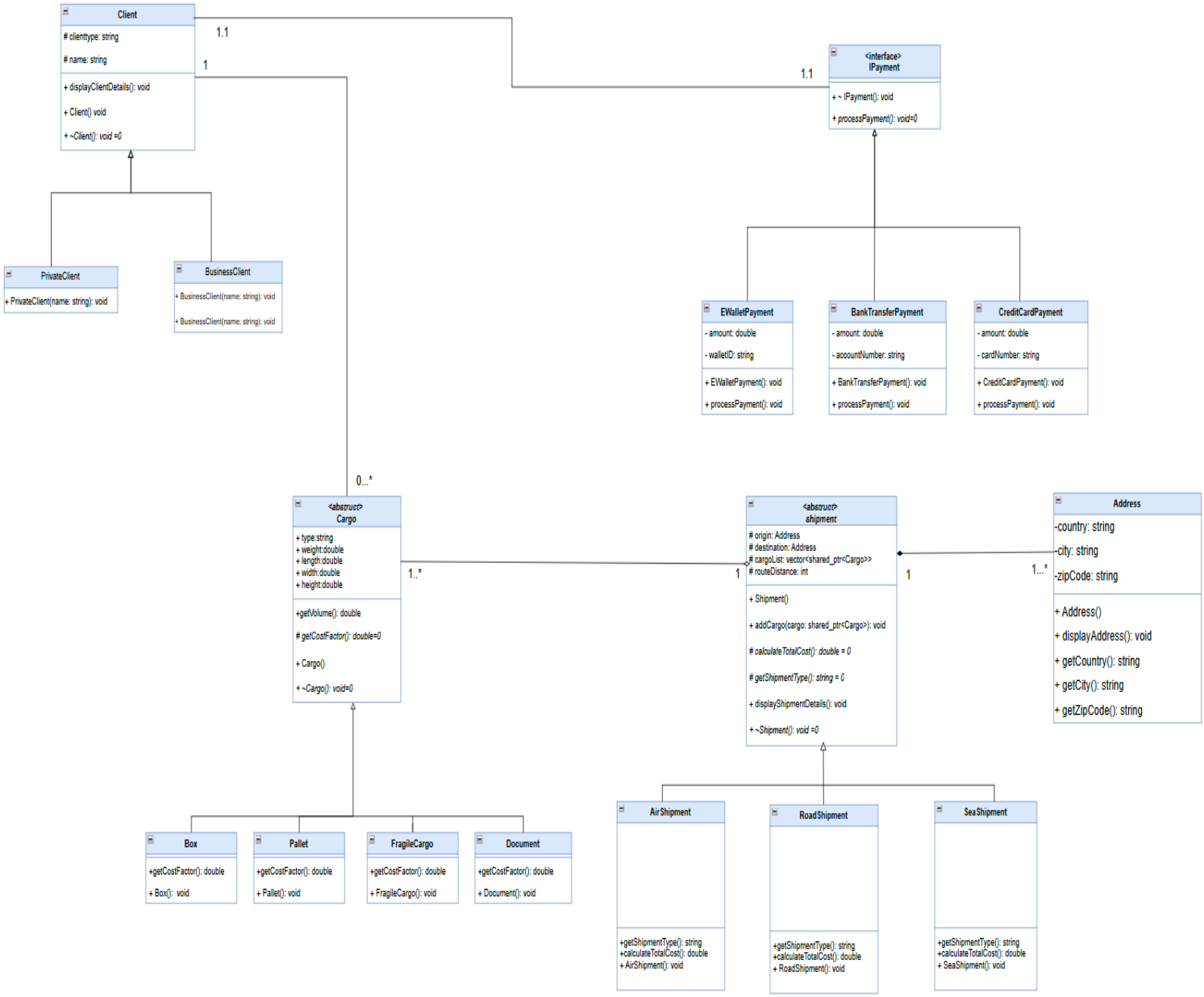
3.2 Use-case diagram



Group6_use-case_diagram.pdf (命令行)

(Double-click the mouse to open the source file)

3.3 Class diagram



Group6_Class_diagram.pdf (命令行)

(Double-click the mouse to open the source file)

4. Details Explanation

```
// Define the Address class to represent address information
class Address {
    string country;
    string city;
    string zipCode;
public:
    Address(string country, string city, string zipCode)
        : country(country), city(city), zipCode(zipCode) {}
    // Method to display the address in the format "City, Country - ZipCode"
    void displayAddress() const {
        cout << city << ", " << country << " - " << zipCode << endl;
    }
    //Use the function to get the country, city, and zip code
    string getCountry() const { return country; }
    string getCity() const { return city; }
    string getZipCode() const { return zipCode; }
};
```

Define the Address class that contains three private variables: country, city, and zip code. Initialize with a constructor. And output in order, while providing three functions to access the three private variables.

```
// Define the Client class as the base class for different types of clients
class Client {
protected:
    string name;
    string clientType;
public:
    Client(string name, string type) : name(name), clientType(type) {}
    virtual ~Client() {} // Virtual destructor for proper cleanup of derived classes
    // Virtual methods, used to display customer details
    virtual void displayClientDetails() const {
        cout << "Client Name: " << name << "\n";
        cout << "Client Type: " << clientType << "\n";
    }
};
//Define the PrivateClient class, which inherits from the Client class
class PrivateClient : public Client {
public:
    PrivateClient(string name) : Client(name, "Private") {}
};
//Define the BusinessClient class, which inherits from the Client class
class BusinessClient : public Client {
public:
    BusinessClient(string name) : Client(name, "Business") {}
};
```

This code defines a base class, Client, to store and display customer information, and creates two derived classes, PrivateClient and BusinessClient, through inheritance. The Client class contains the protected member variables name and clientType and provides a constructor to initialize these variables. displayClientDetails() is a virtual function that displays the client's details, ensuring that they can be overridden in a derived class. Virtual destructor for proper cleanup of derived classes Virtual destructor for proper cleanup of derived classes ~Client() ensures that memory is properly cleanup of derived class.

```
// Define the Cargo class as a base class to represent different types of goods
class Cargo {
public:
    string type;
    double weight;
    double length, width, height;

    Cargo(string type, double weight, double length, double width, double height)
        : type(type), weight(weight), length(length), width(width), height(height) {}
    virtual ~Cargo() {}// Virtual destructor to ensure that derived class objects are properly destroyed
   // Method of calculating the volume of goods, return length * width * height
    virtual double getVolume() const {
        return length * width * height;
    }
    virtual double getCostFactor() const = 0; // Abstract method for cost calculation
};
// Define the Box class, which inherits from the Cargo class
class Box : public Cargo {
public:
    Box(double weight, double length, double width, double height)
        : Cargo("Box", weight, length, width, height) {}
// Implement a pure virtual function that returns the cost factor of the box cargo, which is 0.05
    double getCostFactor() const override {
        return 0.05;
    }
};
// Define the Document class, which inherits from the Cargo class
class Document : public Cargo {
public:
    Document(double weight, double length, double width, double height)
        : Cargo("Document", weight, length, width, height) {}
    double getCostFactor() const override {
        return 0.01;
};
// Define the Pallet class, which inherits from the Cargo class
class Pallet : public Cargo {
public:
    Pallet(double weight, double length, double width, double height)
        : Cargo("Pallet", weight, length, width, height) {}
    double getCostFactor() const override {
        return 0.07; // Cost factor for pallet type
    }
};
// Define the FragileCargo class, which inherits from the Cargo class
class FragileCargo : public Cargo {
public:
    FragileCargo(double weight, double length, double width, double height)
        : Cargo("Fragile", weight, length, width, height) {}
    double getCostFactor() const override {
        return 0.10; // Higher cost factor for fragile cargo
    }
};
```

This code defines a Carg0 base class and four derived classes (Box, Document, Pallet and FraglieCargo) for simulating different types of cargo and their costing, such as type, weight and size. It contains a pure virtual function getCostFactor, which makes Cargo an abstract class. Each derived class inherits from the Cargo class and implements the getCostFactor() method to define its own cost factor.

```
// Define a class that represents shipping information
class Shipment {
protected:
    Address origin;
    Address destination;
    vector<shared_ptr<Cargo>> cargoList;
    int routeDistance;
public:
    Shipment(Address origin, Address destination, int distance)
        : origin(origin), destination(destination), routeDistance(distance) {}
    virtual ~Shipment() {}

    void addCargo(shared_ptr<Cargo> cargo) {// Add the cargos to the cargos list
        cargoList.push_back(cargo);
    }
    virtual double calculateTotalCost() const = 0; // Pure virtual function
    virtual string getShipmentType() const = 0; // Pure virtual function

    virtual void displayShipmentDetails() const { // Display shipping details
        cout << "\n================== Shipment Details ==================\n";
        cout << "Shipment Type: " << getShipmentType() << "\n";
        cout << "Shipment from: "; origin.displayAddress();
        cout << "Shipment to: "; destination.displayAddress();
        cout << "Total Distance: " << routeDistance << " km\n";
        cout << "Number of Packages: " << cargoList.size() << "\n";
        for (const auto& cargo : cargoList) { //Walk through the list of cargos
            cout << "- " << cargo->type << " | Weight: " << cargo->weight << " kg | Volume: " << cargo->getVolume
        }
        cout << "Total Cost: RM" << fixed << setprecision(2) << calculateTotalCost() << "\n";
        cout << "=======================================================\n";
```

This code defines an abstract base class named Shipment that represents the basic information and operations of shipping. Classes include data such as the starting address, destination, cargo list, and shipping distance. The constructor initializes these member variables and provides methods to add goods to the list.

calculateTotalCost and getShipmentType are pure virtual functions, meaning Shipment is an abstract class that cannot be instantiated directly and must be inherited and implemented by subclasses.

The displayShipmentDetails method is responsible for printing detailed shipping information, including shipping type, start and destination, cargo list, shipping distance, and total cost.

```
// Derived classes for specific shipment types
class RoadShipment : public Shipment {
public:
    RoadShipment(Address origin, Address destination, int distance)
        : Shipment(origin, destination, distance) {}

    double calculateTotalCost() const override {
        double cost = 0.0;
        double baseCostFactor = 0.5; // Road shipment base cost factor
        for (const auto& cargo : cargoList) {
            double volume = cargo->getVolume();
            double weight = cargo->weight;
            double cargoCostFactor = cargo->getCostFactor(); // Get factors for specific cargo types

            double userFactor = 0.0;
            if (cargo->type == "Private") {
                userFactor = 1.0;
            } else if (cargo->type == "Business") {
                userFactor = 0.85; // 15% discount for business clients
            }
```

```
    if (routeDistance <= 2000) {
        if (volume <= 0.05) {
            cost += (0.4 * routeDistance + 15 * weight + 500) * baseCostFactor * cargoCostFactor *userFactor;
        } else {
            cost += (0.5 * routeDistance + 18 * weight + 600) * baseCostFactor * cargoCostFactor *userFactor;
        }
    } else if (routeDistance <= 4000) {
        if (volume <= 0.1) {
            cost += (0.45 * routeDistance + 16 * weight + 800) * baseCostFactor * cargoCostFactor *userFactor;
        } else {
            cost += (0.55 * routeDistance + 20 * weight + 1000) * baseCostFactor * cargoCostFactor *userFactor;
        }
    } else {
        cost += (0.55 * routeDistance + 20 * weight + 1000) * baseCostFactor * cargoCostFactor *userFactor;
    }
 }
 return cost;
```

This section of code defines a RoadShipment class, which inherits from the Shipment base class and implements the cost calculation method for road transportation. The calculateTotalCost method overrides the pure virtual function in the base class to calculate the total shipping cost based on the shipping distance, volume, weight, and other factors.

```
class IPayment {// Define an abstract base class IPayment to represent the payment interface
public:
    virtual void processPayment() const = 0;
    virtual ~IPayment() {}
};
// Define a derived class, CreditCardPayment, that inherits from IPayment
class CreditCardPayment : public IPayment {
private:
    double amount;
    string cardNumber;
public:
    CreditCardPayment(double amt, string cardNum) : amount(amt), cardNumber(cardNum) {}
    void processPayment() const override {
        cout << "Processing credit card payment of RM" << amount << " using card number " << cardNumber << ".\n";
    }
};
```

This code defines an abstract structure for payment processing. IPayment is an abstract base class that contains a pure virtual function, processPayment, that represents the

interface for payment processing. By using pure virtual functions, you ensure that all derived classes that inherit IPayment must implement their own processPayment methods.

CreditCardPayment is a concrete derived class that inherits from IPayment and implements the processPayment function. It contains two private member variables: amount (payment amount) and cardNumber (credit card number), and is initialized by a constructor. The processPayment method outputs information about processing credit card payments.

```
// Function Prototypes
void displayMainMenu();
void displayUserTypeMenu();
string selectCountry();
string selectCity(const string& country);
void displayPackageDetailsMenu();
void displayTransportModeMenu();
void displayPaymentMenu();
int getValidIntegerInput();
// verify postcode
bool isPostalCodeValid(const string& country, const string& city, const string& postalCode) {
    auto countryIt = postalCodeMap.find(country);
    if (countryIt != postalCodeMap.end()) {
        auto cityIt = countryIt->second.find(city);
        if (cityIt != countryIt->second.end()) {
            return cityIt->second == postalCode; // Verify that the zip code matches the stored code
        }
    }
    return false; // If the country or city is not found or the zip code does not match, return false
```

This code defines a set of function prototypes and a function to verify the zip code. The function prototype declares each menu display function and user input function to ensure the modularity and clear structure of the program. The isPostalCodeValid function is used to check whether the zip code for a given country or city is valid. It verifies this by looking up countries and cities in the postalCodeMap data structure and then comparing the given zip code. Returns true if matched; Otherwise, return false

```
string getValidPostalCode(const string& country, const string& city) {
    string postalCode;
    while (true) {// Loop until the user enters a valid zip code
        cout << "Enter the postal code for " << city << ", " << country << " (or enter 0 to return): ";
        cin >> postalCode;

        if (postalCode == "0") {
            return "";
        }

        if (isPostalCodeValid(country, city, postalCode)) {// Verify that the code entered by the user is valid
            cout << "Postal code is valid.\n";
            break;
        }
        else {

            string correctPostalCode = postalCodeMap[country][city];
            cout << "Invalid postal code. The correct postal code for " << city << ", " << country << " is "
                << correctPostalCode << ". Please try again.\n";
                // Prompt the user for the correct ZIP code and ask to re-enter it
        }
    }
    return postalCode;
```

This function is used to get the valid zip code entered by the user. It takes two parameters, country and city, and prompts the user for a zip code in a loop. If the zip code entered by the user is valid, the function will indicate that the verification was successful and return the zip code; If not, the user is shown the correct zip code and asked to re-enter it. The user can exit the loop by entering 0, in which case the function returns an empty string

```
while (!exitProgram) {// Loop the main program until exitProgram is true
        displayMainMenu();
        int mainChoice = getValidIntegerInput();

        if (mainChoice == 1) {
            // Get Quote
            cout << "Enter your name: ";
            cin.ignore();
            getline(cin, clientName);

            // Choose user type
            displayUserTypeMenu();
            int userTypeChoice = getValidIntegerInput();

            if (userTypeChoice == 3) continue; // Return to main menu

            unique_ptr<Client> client = nullptr;// Define a pointer to store the Client object
            double maxWeight = 0.0, maxLength = 0.0, maxWidth = 0.0, maxHeight = 0.0;

            if (userTypeChoice == 1) {// Check whether the user selects "Private Client"
                client = make_unique<PrivateClient>(clientName);
                maxWeight = 70.0;
                maxLength = 120.0;
                maxWidth = 80.0;
                maxHeight = 80.0;
            }
            else if (userTypeChoice == 2) {// Check whether the user selects "Business Client"
                client = make_unique<BusinessClient>(clientName);
                maxWeight = 2500.0;
                maxLength = 302.0;
                maxWidth = 223.0;
                maxHeight = 220.0;
            }

            else {
                cout << "Invalid choice. Returning to main menu.\n";
                continue;
            }
```

This code is part of the main program loop and is used to deal with different types of customers and get quotes for them. The program passes while (!) The exitProgram loop

continues until the user opts out. The user first sees the main menu and then selects different functions based on the input

```
originCountry = selectCountry();
            if (originCountry.empty()) {
                // Return to main menu
                continue;
            }
            originCity = selectCity(originCountry);
            if (originCity.empty()) {
                // Return to origin country selection
                continue;
            }
            originZipCode = getValidPostalCode(originCountry, originCity);
            if (originZipCode.empty()) {
                // Return to origin city selection
                continue;
            }

            // Destination selection
            destinationCountry = selectCountry();
            if (destinationCountry.empty()) {
                // Return to main menu
                continue;
            }
            destinationCity = selectCity(destinationCountry);
            if (destinationCity.empty()) {
                // Return to destination country selection
                continue;
            }
            destinationZipCode = getValidPostalCode(destinationCountry, destinationCity);
            if (destinationZipCode.empty()) {
                // Return to destination city selection
                continue;
            }

            if (originCountry == destinationCountry && originCity == destinationCity) {
                cout << "Error: Destination cannot be the same as origin. Please select a different destination.
                continue;
            }
```

This code is used to process the origin and destination information entered by the user, including country, city, and zip code. Each step has input validation and error handling mechanisms to ensure that the user provides valid data and the user selects the starting country, city, and zip code in turn.

If the user selects Return at any step, the program skips the current loop iteration by continuing to return to the previous menu. Again, the user selects the destination country, city, and zip code in turn, and verifies it. Any invalid input or return operation will cause the loop to go back to the previous level. The program finally checks whether the origin is the same as the destination.

If so, an error is displayed and the user is asked to re-select the destination. This ensures that users do not choose the same origin and destination, thereby preventing invalid shipping requests.

```
bool addingCargo = true;// Initializes the flag for adding goods
            while (addingCargo) {
                cout << "\n=== Cargo Limits ===\n";
                cout << "Maximum Weight: " << maxWeight << " kg\n";
                cout << "Maximum Length: " << maxLength << " cm\n";
                cout << "Maximum Width: " << maxWidth << " cm\n";
                cout << "Maximum Height: " << maxHeight << " cm\n";
                displayPackageDetailsMenu();

                int cargoTypeChoice;
                cout << "Select Cargo Type (1 - Document, 2 - Box, 3 - Pallet, 4 - Fragile, 5 - Exit): ";
                cin >> cargoTypeChoice;

                if (cargoTypeChoice == 5) {
                    cout << "Exiting cargo addition...\n";
                    break;
                }

                double weight, length, width, height;

                // Validate weight input
                do {
                    cout << "Enter Weight (kg): ";
                    cin >> weight;
                    if (cin.fail() || weight <= 0) {
                        cin.clear();
                        cin.ignore(numeric_limits<streamsize>::max(), '\n');
                        cout << "Invalid weight. Please enter a positive number.\n";
                        continue;
                    }
                    if (weight > maxWeight) {// Check whether the input weight exceeds the limit
                        cout << "Weight exceeds limit (" << maxWeight << " kg). Please try again.\n";
                        continue;
                    }
                    break;
```

```
                do {
                    cout << "Enter Length, Width, Height (in cm, separated by commas): ";
                    string dimensions;
                    cin >> ws; // Clear whitespace characters
                    getline(cin, dimensions);// Read the entire line of input


                    stringstream ss(dimensions);
                    string token;
                    vector<double> values;

                    while (getline(ss, token, ',')) {
                        try {
                            double value = stod(token); // Converts a string to a double-precision floating-po
                            values.push_back(value);
                        }
                        catch (...) {
                            cout << "Invalid dimensions format. Please enter valid numbers.\n";
                            values.clear();
                        }
                    }

                   // Ensure that three dimensions are entered
                    if (values.size() != 3) {
                        cout << "Invalid dimensions. Please enter exactly three values separated by commas.\n"
                        continue;
                    }

                    length = values[0];
                    width = values[1];
                    height = values[2];


                    if (length <= 0 || width <= 0 || height <= 0) {
                        cout << "Invalid dimensions. Please enter positive numbers.\n";
                        continue;
                    }
```

This code is used to add and verify cargo information to ensure that the weight and dimensions entered by the user meet the specified limits. First, the program loops through the maximum weight, length, width and height limits of the goods, then

prompts the user to select the type of goods and provides an opt-out option. If the user opts out, the program terminates the cargo addition process. When the weight is entered, a do-while loop is used to verify that the value entered by the user is positive and does not exceed the maximum weight limit. If the input is invalid or exceeds the limit, the program prompts an error and asks for re-input. For dimension input, the user enters the length, width, and height in comma-separated form. The program parses the input using a stream of strings and ensures that three valid positive dimensions are entered. If the input format is incorrect, the number of dimensions is incorrect, or the size exceeds the limit, the program will also prompt the user and ask for reinput

```
// Transport mode selection
            displayTransportModeMenu();
            int transportChoice = getValidIntegerInput();
            if (transportChoice == 4) continue; // Return to main menu

            if (transportChoice == 1) {
                shipment = make_shared<RoadShipment>(Address(originCountry, originCity, originZipCode), Address(des
            }
            else if (transportChoice == 2) {
                shipment = make_shared<SeaShipment>(Address(originCountry, originCity, originZipCode), Address(dest
            }
            else if (transportChoice == 3) {
                shipment = make_shared<AirShipment>(Address(originCountry, originCity, originZipCode), Address(dest
            }
            else {
                cout << "Invalid transport type. Returning to main menu.\n";
                continue;
            }

            // Add cargo to shipment
            for (const auto& cargo : cargoList) {
                shipment->addCargo(cargo);
            }

            shipment->displayShipmentDetails();
```

This code deals with the selection of the mode of transport and the addition of goods.

First, use displayTransportModeMenu() to display transport options and obtain the transport mode entered by the user. If the user chooses to return to the main menu (Option 4), the program uses continue to skip the current loop and return to the main menu. Depending on the user's choice, the program creates different types of transport objects:

1 for RoadShipment, 2 for SeaShipment, and 3 for AirShipment. Each transport object is initialized with the start address and destination address. If the user enters an invalid shipping type, the program prompts an error and returns to the main menu. In the Add cargo phase, the program adds all the goods to the shipping object by traversing the cargoList and calling the addCargo method. Finally, display the complete shipping details via displayShipmentDetails().

```
// Payment confirmation
            cout << "Do you want to proceed with payment? (yes/no): ";
            // Prompt the user to confirm whether to proceed with the payment.
            string confirm;
            cin >> confirm;
            transform(confirm.begin(), confirm.end(), confirm.begin(), ::tolower);
            if (confirm == "yes") {// Check whether the user confirms to continue the payment
                double quote = shipment->calculateTotalCost();
                displayPaymentMenu();
                shared_ptr<IPayment> payment = nullptr;
                int paymentChoice = getValidIntegerInput();
                if (paymentChoice == 1) {
                    string cardNumber;
                    cout << "Enter Credit Card Number: ";
                    cin >> cardNumber;
                    payment = make_shared<CreditCardPayment>(quote, cardNumber);
                }
                else if (paymentChoice == 2) {
                    string walletID;
                    cout << "Enter E-Wallet ID: ";
                    cin >> walletID;
                    payment = make_shared<EWalletPayment>(quote, walletID);
                }
                else if (paymentChoice == 3) {
                    string accountNumber;
                    cout << "Enter Bank Account Number: ";
                    cin >> accountNumber;
                    payment = make_shared<BankTransferPayment>(quote, accountNumber);
                }
                if (payment) {
                    payment->processPayment();
                    // Call the payment object's sprocessPayment() method to process the payment.
                }
                else {
                    cout << "Invalid payment method. Payment not processed.\n";
                }
            }
```

This code handles payment confirmation and payment method selection. First, the program prompts the user to confirm whether to proceed with the payment and converts the user input to lower case to ensure that it is handled correctly regardless of whether "yes" or "YES" is entered.

If the user confirms payment, the program calculates the total cost of shipping and displays a menu of available payment methods. Users can choose to pay by credit card, e-wallet or bank transfer. Depending on the payment method chosen by the user, the program will be prompted to enter the corresponding payment information (such as credit card number, e-wallet ID or bank account number), and use the smart pointer to create the corresponding payment object. Finally, the processPayment() method of the payment object is called to process the payment.

```
string selectCountry() {// Define a function that lets the user select a country
    vector<string> countries = { "China", "Malaysia", "Indonesia", "Japan", "South Korea", "Singapore" };
    while (true) {
        cout << "\n========== Origin and Destination Menu ==========\n";
        for (int i = 0; i < countries.size(); ++i) {
            cout << i + 1 << ". " << countries[i] << "\n";
        }
        cout << countries.size() + 1 << ". Return to Main Menu\n";
        cout << "Please select a country: ";

        int choice = getValidIntegerInput();

        if (choice >= 1 && choice <= countries.size()) {
          // Output the selected country and limit duplications here
            return countries[choice - 1];// Return the selected country, subtracting 1 to get the correct index
        }
        else if (choice == countries.size() + 1) {
            cout << "Returning to the Main Menu...\n";
            return ""; // Return an empty string to return to the main menu
        }
        else {
            cout << "Invalid choice. Please try again.\n";
        }
    }
}
```

This code implements a country selection menu, allowing the user to select a country from a predefined list of countries, or choose to return to the main menu. First, we define a string vector containing the names of countries, including China, Malaysia, Indonesia, Japan, South Korea, and Singapore.

A while(true) loop continues to prompt the user to select a country. All countries and their corresponding numbers are printed through the for loop, with an option at the end to return the user to the main menu. When the user enters, the program calls the getValidIntegerInput() function to ensure that the user enters a valid integer

```
string selectCity(const string& country) {// Define a function that allows the user to select a city
    auto countryIt = postalCodeMap.find(country);
    if (countryIt == postalCodeMap.end()) {
        cout << "Error: Invalid country selected.\n";
        return "";
    }

    while (true) {
        cout << "\nHere are the available cities in " << country << ":\n";

        const auto& cityList = countryIt->second; // Get all cities in the current country
        int index = 1;
        for (const auto& city : cityList) {
            cout << index++ << ". " << city.first << "\n";
        }

        cout << "0. Return to the previous menu\n";
        cout << "Please select a city: ";

        int choice = getValidIntegerInput();

        if (choice >= 1 && choice <= cityList.size()) {
            auto it = cityList.begin();
            advance(it, choice - 1);
            cout << "\nYou selected city: " << it->first << "\n";
            return it->first; // Return the city name
        }
        else if (choice == 0) {
            cout << "Returning to the previous menu...\n";
            return "";
        }
        else {
            cout << "Invalid choice. Please try again.\n";
        }
    }
}
```

This code implements a city selection function, first the user selects a country by typing, and the program finds a list of relevant cities in that country in the postalCodeMap. If the country is not found, the program prompts an error and returns an empty string.

If a country is found, the program displays a list of all available cities in that country and lets the user select a city by entering a number. The user can also choose to return to the previous menu (enter 0).

If the user enters a valid city number, the program returns the name of the selected city and outputs information about the selected city. If the input is invalid, the program prompts the user to re-enter until a valid city is selected or the previous level menu is selected.

```
int getValidIntegerInput() {// Define a function that takes valid integer input
    int input;
    while (true) {// Loops until the user enters a valid integer
        cin >> input;

        if (cin.fail()) {// Check if the input failed (for example, the user entered a non-numeric character)
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            cout << "Invalid choice. Please enter a number: ";
        }
        else {
            return input;
        }
    }
}
```

This code defines a function called getValidIntegerInput() to get valid integer input from the user. First, the program tries to read the user input and store it in the input variable.

If the input fails (for example, if the user entered a non-numeric character), cin.fail() returns true, in which case the program clears the error state of the input stream (via cin.clear()) and ignores the remaining characters in the input stream up to the newline (via cin.ignore()) to ensure that subsequent input is not contaminated. The program then prompts the user to re-enter a number. Only when the user enters a valid integer does the function return that integer and end the loop.

5. Test case

Case 1：Private user Lily sent an item from Sepang, Malaysia to Beijing, China. The package was selected as box, the weight was 15kg, the length was 40cm, the width was 30cm, the height was 30cm, the shipping method was air transport, and the payment method was selected as e-wallet

```
================== Main Menu ==================
1. Get a Quote
2. Exit
Please select an option: 1
Enter your name: Lily

================ User Type Menu ================
1. Private User
2. Business User
3. Return to Main Menu
Please select user type: 1

=== Origin Selection ===

=========== Origin Country Selection ===========
1. China
2. Malaysia
3. Indonesia
4. Japan
5. South Korea
6. Singapore
0. Return to Previous Menu
Please select a country: 2

=== Origin City Selection in Malaysia ===
1. Kota Kinabalu
2. Malacca City
3. Kuala Lumpur
4. Petaling Jaya
5. Johor Bahru
6. Sepang
7. George Town
8. Subang Jaya
9. Putrajaya
10. Klang
0. Return to Previous Menu
Please select a city: 6

You selected city: Sepang
Enter the postal code for Sepang, Malaysia (or enter 0 to return): 43900
Postal code is valid.

=== Destination Selection ===

=========== Destination Country Selection ===========
1. China
2. Malaysia
3. Indonesia
4. Japan
5. South Korea
6. Singapore
0. Return to Previous Menu
Please select a country: 1

=== Destination City Selection in China ===
1. Harbin
2. Fuzhou
3. Luoyang
4. Chengdu
5. Shenzhen
6. Yantai
7. Chongqing
8. Hong Kong
9. Shanghai
10. Beijing
0. Return to Previous Menu
Please select a city: 10

You selected city: Beijing
Enter the postal code for Beijing, China (or enter 0 to return): 100000
Postal code is valid.
```

```
=== Cargo Limits ===
Maximum Weight: 70 kg
Maximum Length: 120 cm
Maximum Width: 80 cm
Maximum Height: 80 cm

=============== Package Details Menu ===============
Select Cargo Type (1 - Document, 2 - Box, 3 - Pallet, 4 - Fragile, 5 - Exit): 2
Enter Weight (kg): 15
Enter Length, Width, Height (in cm, separated by commas): 40,30,30
Cargo added successfully!
Do you want to add more cargo? (yes/no): no

================ Transport Mode Menu ================
1. Road
2. Sea
3. Air
4. Return to Main Menu
Please select mode of transport (1-3, or 4 to return): 3

================== Shipment Details ==================
Shipment Type: Air
Shipment from: Sepang, Malaysia - 43900
Shipment to: Beijing, China - 100000
Total Distance: 4600 km
Number of Packages: 1
- Box | Weight: 15 kg | Volume: 36000 cm^3
Total Cost: RM3830.00
======================================================
Do you want to proceed with payment? (yes/no): yes

================= Payment Menu =================
1. Credit Card
2. E-Wallet
3. Bank Transfer
Please select payment method (1-3): 2
Enter E-Wallet ID: 185011514
Processing e-wallet payment of RM3830.00 using wallet ID 185011514.

===== Payment Successful! =====
Client Information
Client Name: Lily
Client Type: Private
==============================
```

Case 2： Different types of users may have different cargo restrictions, which will be indicated in the user interface.

```
=== Cargo Limits ===
Maximum Weight: 70 kg
Maximum Length: 120 cm
Maximum Width: 80 cm
Maximum Height: 80 cm
```

```
=== Cargo Limits ===
Maximum Weight: 2500 kg
Maximum Length: 302 cm
Maximum Width: 223 cm
Maximum Height: 220 cm
```

Case 3：When users make a choice, they have the option to return

```
================= User Type Menu =================
1. Private User
2. Business User
3. Return to Main Menu
```

```
=========== Origin Country Selection ===========
1. China
2. Malaysia
3. Indonesia
4. Japan
5. South Korea
6. Singapore
0. Return to Previous Menu
```

```
Here are the available cities in Malaysia:
1. Kota Kinabalu
2. Malacca City
3. Kuala Lumpur
4. Petaling Jaya
5. Johor Bahru
6. Sepang
7. George Town
8. Subang Jaya
9. Putrajaya
10. Klang
0. Return to the previous menu
```

```
================ Transport Mode Menu ================
1. Road
2. Sea
3. Air
4. Return to Main Menu
Please select mode of transport (1-3, or 4 to return):
```

Case 4: The user selected the city Kota Kinabalu and entered the zip code incorrectly. The user interface will prompt the correct zip code.

```
You selected city: Kota Kinabalu
Enter the postal code for Kota Kinabalu, Malaysia (or enter 0 to return): 15000
Invalid postal code. The correct postal code for Kota Kinabalu, Malaysia is 88000. Please try again.
Enter the postal code for Kota Kinabalu, Malaysia (or enter 0 to return):
```

Case 5： If the user inputs a weight, length, width, or height that exceeds the limit, it will be recognized and asked to re-enter

```
Enter Weight (kg): 800
Weight exceeds limit (70 kg). Please try again.
Enter Weight (kg):
```

```
Select Cargo Type (1 - Document, 2 - Box, 3 - Pallet, 4 - Fragile, 5 - Exit): 4
Enter Weight (kg): 11
Enter Length, Width, Height (in cm, separated by commas): 12,15,99
Dimensions exceed limits (Length: 120 cm, Width: 80 cm, Height: 80 cm). Please try again.
```

6. Reference

[1] DeRoss, N. (n.d.). ShipmentCostCalculator: ShipCost.cpp [Source code]. GitHub. Retrieved November 28, 2024, from https://github.com/ndeross/ShipmentCostCalculator/blob/main/ShipCost.cpp

[2] Peralta, V. (n.d.). DHLGuias: dhl.cpp [Source code]. GitHub. Retrieved November28,2024from https://github.com/VicPeralta/DHLGuias/blob/main/DhlGuias/src/dhl.cpp

[3] GeeksforGeeks. (n.d.). **Searching in a map using std::map functions inC++**.Retrieved November 28, 2024, from https://www.geeksforgeeks.org/searching-in-a-map-using-stdmap-functions-in-cpp/

[4] cppreference.com. (n.d.). **std::numeric_limits**. Retrieved November 28, 2024, from https://en.cppreference.com/w/cpp/types/numeric_limits

7. Marking Rubric

Component Title	Group Assignment – Object-Oriented Programming-C++				**Percentage (%)**	**25**	
Criteria	**Score and Descriptors**					**Weight (%)**	**Marks**
	Excellent (5)	**Good (4)**	**Average (3)**	**Need Improvement (2)**	**Poor (1)**		
C++ code for classes (CLO1)	Well implemented classes that fulfils the design requirements. Additional classes (if needed) are available. All classes are well documented.	Classes are implemented according to the design requirements. All classes are reasonably documented.	Basic classes implemented according to the design requirements. Partially documented classes.	Classes are poorly implemented and not according to the design requirements. Very poorly documented classes.	Non-functional classes. Classes have no bearing to design. No documentation at all.	15	
C++ code for input, output and process control (CLO1)	Program provides excellent I/O. Clear documentation within the code (comments are clear and in relevant places) Names of classes and variables are meaningful. There is a consistent coding standard followed throughout the program.	I/O is adequate. There are comments in the code. Names of classes and variables are somewhat meaningful. There is some coding standard but not consistent.	Basic I/O but functioning program. Minimal comments in the code. Names of classes and variables are random and generic. Little to no coding standard.	Poor I/O. Program is functioning. No comments. Names of classes and variables are random and generic. No coding standard.	Program does not function. No comments. Names of classes and variables are random and generic. No coding standard.	15	

Object-oriented programming (OOP) design (CLO3)	Excellent design. Diagrams are well-organised with explanation for every aspect of the design and is well documented. Shows creativity in design while still providing a proper solution to the problem. Design is consistent with program.	Design is complete. Diagrams with clear explanation for every aspect of the design is given. Adequately solves the problem. Design is mostly consistent with the program.	Design is complete with diagrams. Explanation for every aspect of the design and application is given. Solution to the problem is not clear in the design. Design is somewhat inconsistent with the program.	Partially completed design. There are some diagrams and explanation on the design and concepts of the application. Partial solution is present. Design mostly not consistent with the program.	Incomplete design. No diagrams. No explanation of the design or concepts of the application. Design not consistent with the program.	15	
Inheritance and polymorphism (CLO3)	Excellent and correct abstraction of base class. Derived classes show clear inheritance with overloaded functions. Virtual member functions are properly selected and defined.	Correct abstraction of base class. Derived classes show clear inheritance with overloaded functions. There are some virtual member functions.	Correct abstraction of base class. There are some derived classes. There are some virtual member functions.	Abstraction of base class is not adequate. Derived classes are not properly chosen. No member function over-riding or any type of polymorphism.	No base class abstraction. No inheritance or polymorphic concepts shown.	15	
Report (CLO3)	Excellent document formatting with topics, sub-topics that are well-organised and presented. The flow of the report is pleasing. All diagrams are relevant to the report and well captioned. Table of Contents and Table of Diagrams are well formatted and presented in the document. Appendix with the source code is clearly presented with proper fonts and formatting.	Well-formatted document with topics, sub-topics that are well presented. The flow of the report is good. All diagrams are relevant to the report. Table of Contents and Table of Diagrams are available. Appendix with the source code is available.	Adequate document formatting. The flow of the report is adequate. Diagrams are available. Table of Contents is available. Appendix with the source code is available.	Document formatting is not proper. The report presentation flow is not proper. Diagrams are too few. Table of Contents is available. Appendix with the source code is available.	No document formatting. The report presentation flow is not proper. No diagrams in the report. Table of Contents is available. Appendix with the source code is available.	10	
					SUB-TOTAL	**70**	
Test (CLO4)	Detailed understanding of the group assignment. Thorough explanation of the skills applied and OO concepts used. Contribution is significant.	Good level of understanding of the group assignment. Able to explain skills applied and OO concepts used with some detail. Contribution is satisfactory.	Reasonable understanding of the group assignment. Some explanation of the skills applied and OO concepts used. Contribution is adequate.	Weak understanding of the group assignment. Barely any explanation of the skills applied and OO concepts used. Contribution is poor.	None/hardly any understanding of the group assignment. No explanation of the skills applied and OO concepts used. Contribution is questionable.	30	
					TOTAL	**100**	