Assignment 2

Aims:

- Gain deeper understanding of search algorithms and heuristics
- Apply design patterns in the construction of an object-oriented program
- Learn more about the Java class libraries

Due Date: Week 9, Friday, May 4, 11:59 p.m.

Value: 10%

Shipment Planner

This assignment is inspired by the problem of scheduling a shipping plan for one ship to deliver cargo. Your aim is to schedule an optimal plan for the ship to complete a set of shipments, where only one shipment is carried at a time. While all of the required shipments need to be part of the final plan, the plan may contain trips where the ship does not carry any cargo, but needs to travel somewhere else to pick up a new shipment. The ship requires time for refuelling before undertaking each trip (whether with or without cargo).

More precisely, you will be given a number of distinct specified shipments (e.g. Shanghai to Sydney) that have to be made. The shipments can be scheduled in any order, but the ship always starts in Sydney (and can finish anywhere). The aim is to minimize the total time for the plan, taking into account travel time between ports and a refuelling time before each trip. For simplicity, the name of each port will be just one word. Assume also that the travel time between any two ports is the same in either direction (so need only be specified in one direction) and that the travel time for all possible trips between ports is specified. Furthermore, assume the following "triangle inequality" on travel times: for any ports A, B and C, the travel time from A to C is always less than or equal to the travel time from A to B plus the travel time from B to C.

In this assignment, you will implement an A* search procedure for the shipping plan problem. In your design, make use of the `Strategy` pattern to supply a heuristic to the search procedure, and don't forget to ensure that your heuristic is admissible. **Implementing A* is the main requirement for this assignment, so that your program is guaranteed to produce an optimal solution. If your program does not always produce an optimal solution, then it is wrong.**

Assessment will be based on the design of your program in addition to

correctness and efficiency. You should submit at least a UML class diagram used for the design of your program, i.e. not generated from code afterwards. You should also include a runtime complexity analysis of your heuristic in the comments of your ShipmentPlanner class.

All input will be a sequence of lines of the following form, and all ports and travel times will be declared before any shipment requirements:

```
Refuelling <time> <name>
          # Refuelling time is <time> days in port
<name>
Time <time> <name1> <name2>
          # Travel time is <time> days from port
<name1> to port <name2>
Shipment <name1> <name2>
          # Shipment is required from <name1> to
<name2>
```


Create all your Java source files in the **default package**. Call your main Java file `ShipmentPlanner.java`. Read input from a file whose name is passed as an argument to the main method in the call to `java ShipmentPlanner` and print output to `System.out`. For machine marking, the output will be redirected to a text file that will be compared to the expected output (so do not print out extra spaces, etc.) **and remember to close the input file**. **For the purposes of machine marking, problems will be used for which there is only one optimal solution, though in the case of multiple optimal solutions, your program should produce one of them.**

To read input from a text file (whose name should be passed as a **command line** argument to java, e.g. `java ShipmentPlanner input.txt`), use code such as:

```
  Scanner sc = null;
  try
  {
    sc = new Scanner(new File(args[0]));    // args[0] is the first command
line argument
    // Read input from the scanner here
  }
  catch (FileNotFoundException e)
  {
    System.out.println(e.getMessage());
  }
  finally
  {
    if (sc != null) sc.close();
```

}

Sample Input

Below is an example of the input form and meaning. Note that you will have to submit at least three input test files with your assignment. These test files should include one or more comments to specify what scenario is being tested. However, the following sample input includes many comments added only to explain the input format; your input test files do not need to contain this many comments.

```
Refuelling 6 Sydney              # Refuelling time is
6 days in Sydney
Refuelling 4 Shanghai            # Refuelling time is
4 days in Shanghai
Refuelling 4 Singapore           # Refuelling time is
4 days in Singapore
Refuelling 6 Vancouver           # Refuelling time is
6 days in Vancouver
Refuelling 8 Manila              # Refuelling time is
8 days in Manila
Time 18 Sydney Shanghai          # Travel time is 18
days from Sydney to Shanghai
Time 24 Sydney Singapore         # Travel time is 24
days from Sydney to Singapore
Time 18 Sydney Vancouver         # Travel time is 18
days from Sydney to Vancouver
Time 10 Sydney Manila            # Travel time is 10
days from Sydney to Manila
Time 10 Shanghai Singapore       # Travel time is 10
days from Shanghai to Singapore
Time 24 Shanghai Vancouver       # Travel time is 24
days from Shanghai to Vancouver
Time 12 Shanghai Manila          # Travel time is 12
days from Shanghai to Manila
Time 24 Singapore Vancouver      # Travel time is 24
days from Singapore to Vancouver
Time 22 Singapore Manila         # Travel time is 22
days from Singapore to Manila
Time 20 Vancouver Manila         # Travel time is 20
days from Vancouver to Manila
Shipment Singapore Vancouver     # Shipment is
required from Singapore to Vancouver
Shipment Shanghai Singapore      # Shipment is
```

```
required from Shanghai to Singapore
Shipment Sydney Vancouver       # Shipment is
required from Sydney to Vancouver
Shipment Sydney Manila          # Shipment is
required from Sydney to Manila
```

Sample Output

The above example does not have a unique optimal solution. One valid output corresponding to the above input is as follows. The first line in the output should give the number of nodes *n* expanded in your search, the number of nodes taken **off** the queue, which will vary according to the heuristic used. The second line of the output should give the cost of the solution found as an integer, which is the total time taken, and should be the same regardless of the heuristic and the solution. The remainder of the output should give a sequence of trips that make up an optimal solution. If your program produces a different optimal solution from the one shown (with the same cost) then it is correct.


```
n nodes expanded
cost = 126
Ship Sydney to Manila
Ship Manila to Shanghai
Ship Shanghai to Singapore
Ship Singapore to Vancouver
Ship Vancouver to Sydney
Ship Sydney to Vancouver
```


In particular, for this problem, the following output is also correct.

```
n nodes expanded
cost = 126
Ship Sydney to Vancouver
Ship Vancouver to Sydney
Ship Sydney to Manila
Ship Manila to Shanghai
Ship Shanghai to Singapore
Ship Singapore to Vancouver
```

Submission

- Submit all your files using the following command:

```
give cs2511 ass2 *.java *.pdf *.txt
```

- Your submission should include:
    - All your `.java` source files (there is no need to include `.class` files: your Java files will be recompiled on the CSE machine)
    - A `.pdf` file containing your design documents (a UML class diagram and, optionally, other diagrams necessary to understand your design)
    - A series of `.txt` files (at least three) that you have used as input files to test your system (each including comments to indicate the scenarios tested), and the corresponding `.txt` output files (call these `input1.txt, output1.txt, input2.txt, output2.txt,` etc.)
- When your files are submitted, a test will be done to ensure that your Java files compile on the CSE machine (**take note of any error messages printed out)**
- Check that your submission has been received using the command:


```
2511 classrun -check ass2
```


Assessment

Marks for this assignment are allocated as follows:

- Correctness and efficiency (automarked): 60%
- Design (30%), programming style and analysis of heuristic (10%): 40%

Late penalty: 2 marks per day or part-day late off the mark obtainable for up to 3 (calendar) days after the due date

Assessment Criteria

- Correctness and efficiency: Assessed on standard tests read from an input file, using calls of the form:

```
java ShipmentPlanner input1.txt > output1.txt   # Output to
```

```
System.out is redirected to
output1.txt
```

To test the efficiency of your implementation and the quality of your heuristic, the automarked tests will be run on the CSE remote login servers with a time limit of 10 seconds.

- Design: Adherence to object-oriented design principles, clarity of UML diagrams and conformance of UML diagrams to code, **and appropriate use of design patterns**
- Programming style: Adherence to standard Java programming style, understandable class and variable names, adequate Javadoc and comments
- Analysis of heuristic: Runtime complexity analysis of the heuristic function supplied in the comments of `ShipmentPlanner.java`

Plagiarism

Remember that ALL work submitted for this assignment must be your own work and no code sharing or copying is allowed. You may use code from textbooks or the Internet only with suitable attribution of the source in your program. You should **carefully** read the UNSW policy on academic integrity and plagiarism, noting, in particular, that *collusion* (working together on an assignment, or sharing parts of assignment solutions) is a form of plagiarism.