CS2030 (2310) Exercise #3: Cruise Loaders

Tags & Categories

Tags:

Categories:

Related Tutorials

Task Content

Cruise Loaders

Topic Coverage

- Inheritance
- Polymorphism
- Method Overriding
- CS2030 Java Style Guide

Problem Description

The Kent Ridge Cruise Centre has just opened for business!

As soon as a cruise ship docks at the centre, it is serviced immediately by a number of loaders based on its requirement. If there are insufficient loaders, new ones will be deployed. A loader that is deployed to serve a cruise does not get redeployed until the cruise is served completely. Thereafter, it can be deployed again to serve another cruise if necessary. All loaders take turn to serve cruises.

For example, an incoming cruise arrives at 12PM, requires two loaders, and 60 minutes for it to be fully served. Suppose the cruise centre has two available loaders at 12PM. These two loaders will serve the cruise from 12PM - 1PM. They can only serve other cruises from 1PM onwards.

As another example, if an incoming cruise arrives at 12PM, requires three loaders, and 60 minutes for it to be served. On top of the two loaders available, an additional one will be deployed with all three loaders serving the cruise from 12PM-1PM. These three loaders will be available to serve other cruises from 1PM onwards.

Task

Your task is to determine the service schedule for the loaders. With each arriving cruise, check through the inventory of existing loaders; the first (or first few) loaders available will be used to serve the cruise. If there are not enough loaders, deploy new one(s) to serve the cruise.

Take note of the following assumptions:

- Input cruises are presented chronologically by arrival time.
- There are no duplicate cruises.
- All cruises will arrive and be completely served within a single day.

Level 1

Design a `Cruise` class to represent a cruise having a unique identifier string, the time of arrival as an integer, the number of loaders required to load the cruise as an integer, and the service time in minutes as an integer.

Each cruise has four attributes:

- a unique string identifier, e.g. S1234
- a time of arrival in HHMM format, e.g. 2359 denoting the cruise arriving at 11:59PM on that day
- the time needed to serve the cruise (in minutes), and
- the number of loaders needed to serve the cruise.

Note that the time of arrival is in HHMM format. Specifically, 0 or 0000 refers to 00:00 (12AM), 30 or 0030 refers to 00:30 (12:30AM), and 130 or 0130 refers to 01:30 (1:30AM).

Implement a getServiceTime method, which returns the time taken to complete the service (in number of minutes), and a getArrivalTime method, which returns the arrival time (in number of minutes) since midnight. For example, if the cruise arrives at 12PM (noon time), the arrival time is (12 * 60) = 720.

In addition, implement a getNumOfLoadersRequired method, which returns the number of loaders required to load the cruise.

Although getServiceTime and getNumOfLoadersRequired methods seem to violate *Tell-Don't-Ask*, these will be useful for other types of cruises that do not such parameters specified explicitly.

Tip: the %0Xd format specifier might be of use to you, where the integer will be represented by an X-digit zero-padded number. For instance,

```
String.format("%04d", 20);
```

would return the string 0020.

```
jshell> new Cruise("A1234", 0, 2, 30)
$.. ==> A1234@0000

jshell> new Cruise("A2345", 30, 2, 30)
$.. ==> A2345@0030

jshell> new Cruise("A3456", 130, 2, 30)
$.. ==> A3456@0130

jshell> new Cruise("A3456", 130, 2, 30).getArrivalTime()
$.. ==> 90

jshell> new Cruise("A3456", 130, 2, 30).getNumOfLoadersRequired()
$.. ==> 2

jshell> new Cruise("A3456", 130, 5, 30).getNumOfLoadersRequired()
$.. ==> 5

jshell> new Cruise("A1234", 0, 2, 30).getServiceTime()
$.. ==> 30

jshell> new Cruise("A1234", 0, 2, 45).getServiceTime()
$.. ==> 45

jshell> new Cruise("CS2030", 1200, 2, 100).getServiceTime()
$.. ==> 100
```

Level 2

Start off by designing a Loader class which simply encapsulates the identifier of the loader.

```
jshell> new Loader(1)
$.. ==> Loader #1
```

Rather than setting up a direct dependency between Cruise and Loader, we set up an *association* between the two classes by creating another class Service. This would also allow us to define the start and end times of the service.

```
jshell> new Service(new Loader(1), new Cruise("A1234", 0, 2, 30))
$.. ==> Loader #1 serving A1234@0000

jshell> new Service(new Loader(1), new Cruise("A1234", 0, 2, 30)).getServiceStartTime()
$.. ==> 0

jshell> new Service(new Loader(1), new Cruise("A1234", 0, 2, 30)).getServiceEndTime()
$.. ==> 30

jshell> new Service(new Loader(2), new Cruise("A3456", 130, 2, 30))
$.. ==> Loader #2 serving A3456@0130
```

```
jshell> new Service(new Loader(2), new Cruise("A3456", 130, 2, 30)).getServiceStartTime()
$.. ==> 90

jshell> new Service(new Loader(2), new Cruise("A3456", 130, 2, 30)).getServiceEndTime()
$.. ==> 120
```

Level 3

Let's now work on the loader service schedule. In the file `level3.jsh`, write a method

```
ImList<Service> serveCruises(ImList<Cruise> cruises) {
   ...
}
```

that takes in a list of cruises, and returns the service schedule of the loaders in the form a list.

Use the following loader deployment strategy:

- Initialize two service queues as empty: one for `ACTIVE` services and one for `EXPIRED` services:
- For each cruise that arrives:
  - Go through the services in `ACTIVE` queue, and look for those that have completed service. Transfer these services into the `EXPIRED` queue.
  - Pick expired services from the `EXPIRED` queue, make them active again by serving the new cruise and place them into the `ACTIVE` queue; if the cruise still requires additional loaders, create new ones and deploy them.

Use the `ImList` class and methods provided to manage the queues.

The given `ImList` class includes javadoc comments. To automatically generate HTML documentation from the comments, issue the command:

```
$ javadoc -d doc ImList.java
```

You may then navigate through the documentation from `allclasses-index.html` found in the `doc` directory.

As the number of java files get increasing larger, rather than opening the files one at a time within JShell, you can enumerate the files while invoking jshell, but in a bottom up dependency order, starting with the simplest standalone class. Remember to compile your classes first.

```
$ javac your_java_files
$ jshell your_java_files_in_bottom-up_dependency_order
jshell> /open level3.jsh

jshell> ImList<Cruise> cruises = new ImList<Cruise>().
   ...> add(new Cruise("A1234", 0, 2, 30)).
   ...> add(new Cruise("A2345", 30, 2, 30)).
   ...> add(new Cruise("A3456", 130, 2, 30))
cruises ==> [A1234@0000, A2345@0030, A3456@0130]

jshell> for (Service s : serveCruises(cruises)) {
   ...>     System.out.println(s);
   ...> }
Loader #1 serving A1234@0000
Loader #2 serving A1234@0000
Loader #1 serving A2345@0030
Loader #2 serving A2345@0030
Loader #1 serving A3456@0130
Loader #2 serving A3456@0130

jshell> cruises = new ImList<Cruise>().
   ...> add(new Cruise("A1234", 0, 2, 30)).
   ...> add(new Cruise("A2345", 0, 2, 10)).
   ...> add(new Cruise("A3456", 10, 2, 20))
cruises ==> [A1234@0000, A2345@0000, A3456@0010]

jshell> for (Service s : serveCruises(cruises)) {
   ...>     System.out.println(s);
   ...> }
Loader #1 serving A1234@0000
Loader #2 serving A1234@0000
Loader #3 serving A2345@0000
Loader #4 serving A2345@0000
Loader #3 serving A3456@0010
Loader #4 serving A3456@0010
```

Level 4

The objective of this level is to determine whether your current implementation can be easily extended with minimal modification to the client.

There are two types of cruises:

- `SmallCruise`:
  - has an identifier that starts with an uppercase character `S`;
  - takes 30 minutes for a loader to load;
  - requires only one loader
- `BigCruise`:
  - has an identifier that starts with an uppercase character `B`;
  - takes one minute to serve every 50 passengers;
  - requires one loader per 40 meters in length of the cruise (or part thereof) to fully load

Design the `SmallCruise` and `BigCruise` classes.

- For `SmallCruise`, the arguments of the constructor are its identifier, and time of arrival;
- For `BigCruise`, the arguments of the constructor are its identifier, time of arrival, the length of the cruise, and number of passengers.

If your program has been designed correctly, you should not need to modify your existing classes or `level3.jsh`.

```
$ javac your_java_files
$ jshell your_java_files_in_bottom-up_dependency_order
jshell> new SmallCruise("S0001", 0).getArrivalTime()
$.. ==> 0

jshell> new SmallCruise("S0001", 0).getServiceTime()
$.. ==> 30

jshell> new SmallCruise("S0001", 0).getNumOfLoadersRequired()
$.. ==> 1

jshell> Cruise cruise = new SmallCruise("S0123", 1220)
cruise ==> S0123@1220

jshell> cruise = new BigCruise("B0001", 0, 70, 3000)
cruise ==> B0001@0000

jshell> cruise.getArrivalTime()
$.. ==> 0

jshell> cruise.getServiceTime()
$.. ==> 60

jshell> cruise.getNumOfLoadersRequired()
$.. ==> 2

jshell> /open level3.jsh

jshell> ImList<Cruise> cruises = new ImList<Cruise>().add(new SmallCruise("S1111", 1300))
cruises ==> [S1111@1300]

jshell> serveCruises(cruises)
$.. ==> [Loader #1 serving S1111@1300]

jshell> cruises = new ImList<Cruise>().
   ...> add(new BigCruise("B1111", 1300, 80, 3000)).
   ...> add(new SmallCruise("S1111", 1359)).
   ...> add(new SmallCruise("S1112", 1500)).
   ...> add(new SmallCruise("S1113", 1529))
cruises ==> [B1111@1300, S1111@1359, S1112@1500, S1113@1529]

jshell> for (Service s : serveCruises(cruises)) {
   ...>     System.out.println(s);
   ...> }
Loader #1 serving B1111@1300
Loader #2 serving B1111@1300
Loader #3 serving S1111@1359
Loader #1 serving S1112@1500
Loader #2 serving S1113@1529

jshell> cruises = new ImList<Cruise>().
   ...> add(new SmallCruise("S1111", 900)).
   ...> add(new BigCruise("B1112", 901, 100, 1)).
   ...> add(new BigCruise("B1113", 902, 20, 4500)).
   ...> add(new SmallCruise("S2030", 1031)).
   ...> add(new BigCruise("B0001", 1100, 30, 1500)).
   ...> add(new SmallCruise("S0001", 1130))
cruises ==> [S1111@0900, B1112@0901, B1113@0902, S2030@1031, B0001@1100, S0001@1130]

jshell> for (Service s : serveCruises(cruises)) {
   ...>     System.out.println(s);
   ...> }
Loader #1 serving S1111@0900
Loader #2 serving B1112@0901
Loader #3 serving B1112@0901
Loader #4 serving B1112@0901
Loader #2 serving B1113@0902
Loader #3 serving S2030@1031
Loader #4 serving B0001@1100
Loader #1 serving S0001@1130
```

Level 5

The Cruise Centre has just introduced a new eco-friendly policy in an effort to go green. Their policy states that every third loader that is deployed must be made of recycled materials (referred to as recycled loaders). These recycled loaders will go through a 60-minute long maintenance after every service. It is unable to serve any cruise during this period.

For example, if a recycled loader serves a `SmallCruise` that arrives at 12:30PM, then the next time the loader can serve another cruise is 2PM (30min + 60min after 12:30PM).

By modifying `level3.jsh`, incorporate the above with minimal modifications to the file `level5.jsh`.

```
$ javac your_java_files
$ jshell your_java_files_in_bottom-up_dependency_order
jshell> /open level5.jsh

jshell> ImList<Cruise> cruises = new ImList<Cruise>().
   ...> add(new BigCruise("B1111", 0, 60, 1500)).
   ...> add(new SmallCruise("S1112", 0)).
   ...> add(new BigCruise("B1113", 30, 100, 1500)).
   ...> add(new BigCruise("B1114", 100, 100, 1500)).
   ...> add(new BigCruise("B1115", 130, 100, 1500)).
   ...> add(new BigCruise("B1116", 200, 100, 1500))
cruises ==> [B1111@0000, S1112@0000, B1113@0030, B1114@0100, B1115@0130, B1116@0200]

jshell> for (Service s : serveCruises(cruises)) {
   ...>     System.out.println(s);
   ...> }
Loader #1 serving B1111@0000
Loader #2 serving B1111@0000
Recycled Loader #3 serving S1112@0000
Loader #1 serving B1113@0030
Loader #2 serving B1113@0030
Loader #4 serving B1113@0030
Loader #1 serving B1114@0100
Loader #2 serving B1114@0100
Loader #4 serving B1114@0100
Recycled Loader #3 serving B1115@0130
Loader #1 serving B1115@0130
Loader #2 serving B1115@0130
Loader #4 serving B1116@0200
Loader #1 serving B1116@0200
Loader #2 serving B1116@0200
```

Submission (Course)

Select course: CS2030 (2023/2024 Sem 1) - Programming Methodology II

Your Files:

BROWSE

SUBMIT (only .java, .c, .cpp, .h, .jsh, and .py extensions allowed)

To submit multiple files, click on the Browse button, then select one or more files. The selected file(s) will be added to the upload queue. You can repeat this step to add more files. Check that you have all the files needed for your submission. Then click on the Submit button to upload your submission.

© Copyright 2009-2023 National University of Singapore. All Rights Reserved.
Terms of Use | Privacy | Non-discrimination

MySoC | Computing Facilities | Search | Campus Map
School of Computing, National University of Singapore