Cruise Loaders

Topic Coverage

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

Problem Description

The Kent Ridge Cruise Centre has just opened and you are required to design a program to decide how many loaders to buy based on a single-day cruise schedule.

Cruises

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.

Every cruise must be served by loaders immediately upon arrival.

There are two types of cruises:

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

Loaders

Loaders have to be purchased to serve cruises. Each loader comprises two attributes:

- a unique integer identifier
- the cruise that it is currently serving

A loader will serve a cruise as soon as it arrives, and continues to do so until the service time has elapsed (i.e. it cannot serve a cruise while in the midst of serving another one).

For example, if an incoming cruise arrives at 12PM, requires two loaders, and 60 min for it to be fully served, then, at 12PM, there must be two vacant loaders. These two loaders will serve the cruise from 12PM - 1PM. They can only serve another cruise from 1PM onwards.

Task

Your task is to determine the loader allocation schedule using the following steps:

- For each cruise, check through the inventory of existing loaders, starting from the loader first purchased, and so on;
- The first (or first few) loaders available will be used to serve the cruise;
- If there are not enough loaders, purchase new one(s) to serve the cruise.

Take note of the following assumptions:

- Input cruises are presented chronologically by arrival time.
- There can be up to 30 cruises in one day.
- The number of loaders servicing a cruise will not exceed 9.
- There are no duplicate cruises.
- All cruises will arrive and be completely served within a single day..

Although this problem can be implemented procedurally, you are to model your solution using an object-oriented approach instead.

This task is divided into several levels. You need to complete ALL levels.

Level 1

Represent a Cruise

Design an *immutable* `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.

```
class Cruise {
    private final String identifier;
    private final int arrivalTime;
    private final int numOfLoader;
    private final int serviceTime;

    ...
}
```

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 `getServiceCompletionTime` method, which returns the time the service completes (in number of minutes) since midnight, 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; the service completion time is 12:30PM, which is 750 minutes since midnight, i.e. (12 * 60) + 30 = 750.

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

A string representation of a cruise is in the form:

```
cruiseID@HHMM
```

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> /open Cruise.java
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).getServiceCompletionTime()
$.. ==> 30
jshell> new Cruise("A1234", 0, 2, 45).getServiceCompletionTime()
$.. ==> 45
jshell> new Cruise("CS2030", 1200, 2, 100).getServiceCompletionTime()
$.. ==> 820
jshell> new Cruise("D1010", 2329, 2, 30).getServiceCompletionTime()
$.. ==> 1439
jshell> /exit
```

Check the format correctness of the output by running the following on the command line:

```
$ javac *.java
$ jshell -q < test1.jsh
```

Check your styling by issuing the following

```
$ checkstyle *.java
```

Level 2

Use Loaders to serve Cruises

Design an *immutable* `Loader` class to serve a cruise.

```
class Loader {
    private final int identifier;
    private final Cruise cruise;
```

```
    ...
}
```

Include the following in the `Loader` class:

- a constructor that takes in an integer denoting its unique identifier, as well as the first cruise that it serves.
- a `canServe(Cruise)` method that returns `true` if the loader is available to serve the given cruise, or `false` otherwise.
- a `serve(Cruise)` method to serve a given cruise. If the loader is available, the method returns the loader serving this cruise; otherwise the existing loader is returned
- the methods `getIdentifier()` and `getNextAvailableTime()` to return the loader's identifier, and the next available time for service.

The string representation of each Loader is in the form:

```
Loader ID serving cruiseID@cruisearrivaltime
```

```
jshell> /open Cruise.java
jshell> /open Loader.java
jshell> new Loader(1, new Cruise("A1234", 0, 1, 30))
$.. ==> Loader 1 serving A1234@0000
jshell> new Loader(1, new Cruise("A1234", 0, 1, 30)).getIdentifier()
$.. ==> 1
jshell> new Loader(1, new Cruise("A1234", 0, 1, 30)).getNextAvailableTime()
$.. ==> 30
jshell> new Loader(1, new Cruise("A1234", 0, 1, 30)).canServe(new Cruise("A2345", 30, 1, 30))
$.. ==> true
jshell> new Loader(1, new Cruise("A1234", 0, 1, 30)).serve(new Cruise("A2345", 30, 1, 30))
$.. ==> Loader 1 serving A2345@0030
jshell> new Loader(1, new Cruise("A1234", 0, 1, 30)).serve(new Cruise("A2345", 30, 1, 30)).getNextAvailableTime()
$.. ==> 60
jshell> new Loader(1, new Cruise("A1234", 0, 1, 30)).canServe(new Cruise("A2345", 10, 1, 30))
$.. ==> false
jshell> new Loader(1, new Cruise("A1234", 0, 1, 30)).serve(new Cruise("A2345", 10, 1, 30))
$.. ==> Loader 1 serving A1234@0000
jshell> new Loader(1, new Cruise("A1234", 0, 1, 30)).serve(new Cruise("A2345", 10, 1, 30)).getNextAvailableTime()
$.. ==> 30
jshell> new Loader(1, new Cruise("A1234", 0, 1, 30)).serve(new Cruise("A2345", 10, 1, 30)).getIdentifier()
$.. ==> 1
jshell> /exit
```

Check the format correctness of the output by running the following on the command line:

```
$ javac *.java
$ jshell -q < test2.jsh
```

Check your styling by issuing the following

```
$ checkstyle *.java
```

Level 3

Represent Small Cruises

Design the `SmallCruise` class. The arguments of the constructor are its identifier, and time of arrival. Note that you should not need to change your `Loader` class if you have implemented it properly.

```
jshell> /open Loader.java
jshell> /open Cruise.java
jshell> /open SmallCruise.java
jshell> new SmallCruise("S0001", 0).getArrivalTime()
$.. ==> 0
jshell> new SmallCruise("S0001", 0).getServiceCompletionTime()
$.. ==> 30
jshell> new SmallCruise("S0001", 0).getNumOfLoadersRequired()
$.. ==> 1
jshell> (Cruise) new SmallCruise("S0123", 1220)
$.. ==> S0123@1220
jshell> new Loader(1, new SmallCruise("S1245", 2330))
$.. ==> Loader 1 serving S1245@2330
jshell> new Loader(1, new SmallCruise("S1245", 2330)).canServe(new SmallCruise("S2345", 2359))
$.. ==> false
jshell> new Loader(1, new SmallCruise("S1245", 2330)).serve(new SmallCruise("S2345", 2359))
$.. ==> Loader 1 serving S1245@2330
jshell> new Loader(1, new SmallCruise("S2030", 0))
$.. ==> Loader 1 serving S2030@0000
jshell> /exit
```

Check the format correctness of the output by running the following on the command line:

```
$ javac *.java
$ jshell -q < test3.jsh
```

Check your styling by issuing the following

```
$ checkstyle *.java
```

Level 4

Represent Big Cruises

Design the `BigCruise` class. The arguments of the constructor are its identifier, time of arrival, the length of the cruise, and number of passengers, in that order.

```
jshell> /open Loader.java
jshell> /open Cruise.java
jshell> /open SmallCruise.java
jshell> /open BigCruise.java
jshell> Cruise b = new BigCruise("B0001", 0, 70, 3000)
jshell> b.getArrivalTime()
$.. ==> 0
jshell> b.getServiceCompletionTime()
$.. ==> 60
jshell> b.getNumOfLoadersRequired()
$.. ==> 2
jshell> new Loader(1, b).serve(b)
$.. ==> Loader 1 serving B0001@0000
jshell> new Loader(1, b).serve(b).getNextAvailableTime()
$.. ==> 60
jshell> new Loader(2, b)
$.. ==> Loader 2 serving B0001@0000
jshell> new Loader(3, b)
$.. ==> Loader 3 serving B0001@0000
jshell> new Loader(4, new BigCruise("B2345", 0, 30, 1450)).serve(new SmallCruise("S0000", 29))
$.. ==> Loader 4 serving S0000@0029
jshell> new Loader(5, new BigCruise("B3456", 0, 75, 1510)).serve(new SmallCruise("S0001", 30))
$.. ==> Loader 5 serving B3456@0000
jshell> /exit
```

Check the format correctness of the output by running the following on the command line:

```
$ javac *.java
$ jshell -q < test4.jsh
```

Check your styling by issuing the following

```
$ checkstyle *.java
```

Level 5

Output the loader allocation schedule

Write a method `serveCruises(Cruise[])` that takes in an array of cruises, and outputs the allocation schedule of the loaders required to service all the cruises. Save the method in the file `level5.jsh`.

You may assume that there are at most 30 cruises in one day, and the number of loaders servicing a cruise will not exceed 9.

```
jshell> /open Cruise.java
jshell> /open SmallCruise.java
jshell> /open BigCruise.java
jshell> /open Loader.java
jshell> /open level5.jsh
jshell> Cruise[] cruises = {
   ...>     new SmallCruise("S1111", 1300)}
jshell> serveCruises(cruises);
Loader 1 serving S1111@1300
jshell> Cruise[] cruises = {
   ...>     new BigCruise("B1111", 1300, 80, 3000),
   ...>     new SmallCruise("S1111", 1359),
   ...>     new SmallCruise("S1112", 1400),
   ...>     new SmallCruise("S1113", 1429)}
jshell> serveCruises(cruises);
Loader 1 serving B1111@1300
Loader 2 serving B1111@1300
Loader 3 serving S1111@1359
Loader 1 serving S1112@1400
Loader 2 serving S1113@1429
jshell> Cruise[] cruises = {
   ...>     new SmallCruise("S1111", 900),
   ...>     new BigCruise("B1112", 901, 100, 1),
   ...>     new BigCruise("B1113", 902, 20, 4500),
   ...>     new SmallCruise("S2030", 1031),
```

```
   ...>     new BigCruise("B0001", 1100, 30, 1500),
   ...>     new SmallCruise("S0001", 1130)}
jshell> serveCruises(cruises);
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 1 serving S2030@1031
Loader 2 serving B0001@1100
Loader 1 serving S0001@1130
jshell> /exit
```

Check the format correctness of the output by running the following on the command line:

```
$ javac *.java
$ jshell -q < test5.jsh
```

Check your styling by issuing the following

```
$ checkstyle *.java
```