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.

Loaders

Loaders have to be purchased to serve cruises. Each loader will serve a cruise as soon as it arrives, and continues to do so until the service time has elapsed..

While the loaders are in the midst of serving a cruise, they cannot serve another cruise.

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.

Cruises

All cruises have four attributes: a unique identifier, and a time of arrival, the time needed to serve the cruise (in minutes) and the number of loaders needed to serve the cruise.

The unique identifier is a String, for example, S1234. The time of arrival comes in HHMM format, such as 2359, denoting that the cruise has arrived at 11:59PM on that day.

Every cruise must be served by loaders immediately upon arrival. There are two types of cruises:

- `SmallCruise`:
  - `SmallCruises` have an identifier that starts with a character â€˜Sâ€™ (case sensitive).
  - Takes a fixed 30min for a loader to fully load.
  - Requires only one loader for it to be fully served.
  - A schedule entry is read as such:

    ```
    S1234 0000
    ```

    denoting that cruise S1234 arrives at 12AM, requires 1 loader and 30 minutes to be fully served.
- `BigCruise`:

    - `BigCruises` have an identifier that starts with the character 'B' (case-sensitive). Furthermore, they have a variable number of loaders and time required to serve it.
    - 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,
    - A schedule entry is read as such:

      ```
      B1234 0000 200 3720
      denoting that cruise B1234 arrives at 12AM, has a length of 200 meters, with 372
      This cruise requires ceil(200/40) = 5 loaders and takes ceil(3720/50) = 75 minut
      ```

The program determines the number of loaders and the allocation schedule using the following steps:

- For each cruise, check through the inventory of 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 a new one(s), and that loader(s) will be used to serve the cruise.

Task

Write a program that reads in the number of cruises in the schedule as an integer, and a list of cruises that will arrive for that day.

The program will output the loader allocation schedule. 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 duplicates in the input cruises.
- All cruises will arrive and be completely served within a single day.
- Input validation is not required and all inputs are assumed to be correct.

Although this problem can be implemented procedurally, you are to model your solution using an object-oriented approach instead. This task is divided into five levels. You need to complete ALL levels.

Level 1

Represent a Cruise

Design an *immutable* class `Cruise` that represents a `Cruise` instance, with a unique identifier string, th an integer, the number of loaders required to load the cruise as an integer, and the service time in min

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

Implement a `getServiceCompletionTime` method, which returns the time (in minutes) it takes for the s completed and a `getArrivalTime` method, which returns the arrival time in minutes since the start of t

For example, if the `Cruise` arrives at 12PM, the arrival time is (12 * 60) = 720; the service completior which is 750 minutes from 00:00. (12 * 60) + 30 = 750).

In addition, implement a `getNumOfLoadersRequired` method, which returns the number of loaders requ 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 number.

For instance,

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

would return `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 < level1.jsh
```

Check your styling by issuing the following

```
$ checkstyle *.java
```

Level 2

Use Loaders to serve Cruises

Design a *immutable* class `Loader` that is able to serve a `Cruise` using a `serve(Cruise)` method. Add a `
method that returns a boolean value to check if a given loader is available to serve the given cruise. N

- The constructor takes in an integer denoting its ID.
- `Loader` instances are immutable.
- If the `loader` cannot serve a cruise, a `null` value is returned.

The string representation of each Loader is in the form:

```
Loader id serving cruiseid@cruisearrivaltime
```

if the loader is serving a cruise, or simply

```
Loader id
```

if the loader is not serving any cruise.

```
jshell> /open Cruise.java
jshell> /open Loader.java
jshell> new Loader(1)
$.. ==> Loader 1
jshell> new Loader(1).canServe(new Cruise("A1234", 0, 1, 30))
$.. ==> true
jshell> new Loader(1).serve(new Cruise("A1234", 0, 1, 30))
$.. ==> Loader 1 serving A1234@0000
jshell> new Loader(1).serve(new Cruise("A1234", 0, 1, 30)).canServe(new Cruise("A2345", 3
$.. ==> true
jshell> new Loader(1).serve(new Cruise("A1234", 0, 1, 30)).serve(new Cruise("A2345", 30,
$.. ==> Loader 1 serving A2345@0030
jshell> new Loader(1).serve(new Cruise("A1234", 0, 1, 30)).canServe(new Cruise("A2345", 1
$.. ==> false
jshell> new Loader(1).serve(new Cruise("A1234", 0, 1, 30)).serve(new Cruise("A2345", 10,
$.. ==> null
jshell> Loader loader = new Loader(2)
jshell> loader.serve(new Cruise("U8888", 0, 1, 20))
$.. ==> Loader 2 serving U8888@0000
jshell> loader.serve(new Cruise("U8888", 1, 1, 20))
$.. ==> Loader 2 serving U8888@0001
jshell> loader = new Loader(3)
jshell> loader.canServe(null)
$.. ==> true
jshell> loader.serve(null)
$.. ==> Loader 3
jshell> /exit
```

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

```
$ javac *.java
$ jshell -q < level2.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 ID, and time of arrival. Note th 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> new SmallCruise("S0123", 1220)
$.. ==> S0123@1220
jshell> new Loader(1).serve(new SmallCruise("S1245", 2330))
$.. ==> Loader 1 serving S1245@2330
jshell> new Loader(1).serve(new SmallCruise("S1245", 2330)).canServe(new SmallCruise("S23
$.. ==> false
jshell> new Loader(1).serve(new SmallCruise("S1245", 2330)).serve(new SmallCruise("S2345"
$.. ==> null
jshell> new Loader(1).serve(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 < level3.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 ID, time of arrival, the length of 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).serve(b).serve(b)
$.. ==> null
jshell> new Loader(2).serve(b)
$.. ==> Loader 2 serving B0001@0000
jshell> new Loader(3).serve(b)
$.. ==> Loader 3 serving B0001@0000
jshell> new Loader(4).serve(new BigCruise("B2345", 0, 30, 1450)).serve(new SmallCruise("S
$.. ==> Loader 4 serving S0000@0029
jshell> new Loader(5).serve(new BigCruise("B3456", 0, 75, 1510)).serve(new SmallCruise("S
$.. ==> null
jshell> /exit
```

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

```
$ javac *.java
$ jshell -q < level4.jsh
```

Check your styling by issuing the following

```
$ checkstyle *.java
```

Level 5

Completing the program

Write a program that reads in the number of cruises that arrive daily, and the daily cruise schedule. Th
allocation schedule for the loaders. Bear in mind that there can be up to 30 cruises in one day, and the
servicing a cruise will not exceed 9.

By including the functionality of input and output, compile and run the program as follows:

```
$ javac Main.java
```

where the Main class, together with all its dependency classes will be compiled. Then

```
$ java Main
```

to run the program. The following are sample runs of the program. User input is underlined.

```
$ java Main
1
```

```
S1111 1300
Loader 1 serving S1111@1300


$ java Main
4
B1111 1300 80 3000
S1111 1359
S1112 1400
S1113 1429
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


$ java Main
6
S1111 0900
B1112 0901 100 1
B1113 0902 20 4500
S2030 1031
B0001 1100 30 1500
S0001 1130
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


$ java Main
0
```

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

```
$ java Main < test.in | diff - test.out
```

Check your styling by issuing the following

```
$ checkstyle *.java
```