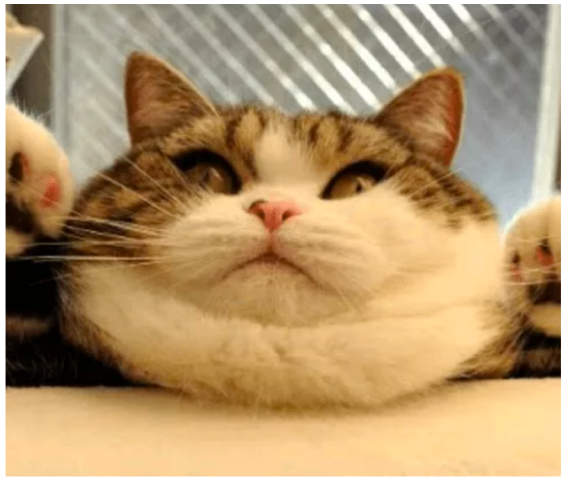
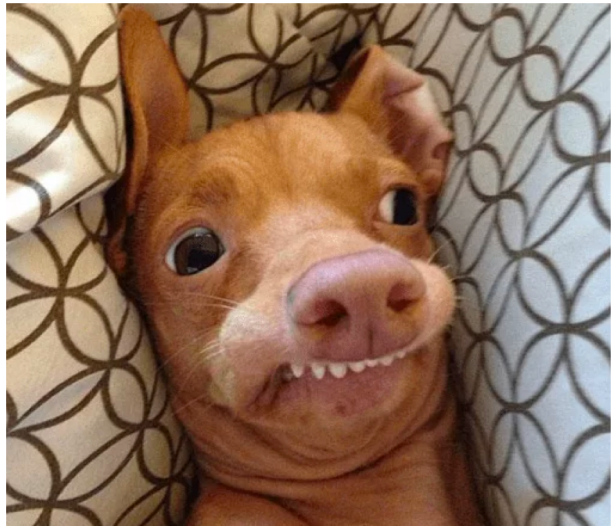


CS2030 LAB #2

“Cruise Loaders”



Cruise Loaders

Output: allocation schedule for the loaders

- Design a program to decide how many loaders to buy based on a single-day cruise schedule

Input: The number of cruises that arrive daily, and the daily cruise schedule

- Loaders:
  - Used to serve cruises
  - While the loaders are in the midst of serving a cruise, they cannot serve another cruise up until the current service has been completed.
- Cruises (2 TYPES):
  - Cruise
  - Big Cruise

Inheritance needed → so that they can share the same methods albeit with some differences in the method (ie OVERRIDING)

Cruise VS BigCruise

Cruise

- Takes a fixed 30min for a loader to fully load.
- Requires only one loader for it to be fully served.

Input format: `A1234 0000`

(cruise A1234 arrives at 12AM, requires 1 loader and 30 minutes to be fully served.)

BigCruise

- starts with the character 'B' (case-sensitive).
- Takes T minutes to serve, where T is between 0 to 99.
- Requires X loaders to fully load, where X is between 0 to 9.

Input format: `B1234 0000 5 75`

(cruise B1234 arrives at 12AM, requires 5 loaders and 75 minutes to be fully served.)

Assumptions you can make

- Input cruises are presented chronologically by arrival time.
- There can be up to 30 cruises in one day.
- 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 can be assumed to be correct.

Level 1

Creating a Cruise class:

- Mutable (Internal state of a `Cruise` object can be changed after being created. But think about which variables can be changed and which should be `final`)
- Variables:
  - id of the cruise (example: A1234) [String]
  - Arrival time [int]
  - What else do you need? Hint:
    - Is the time require to serve a `Cruise` the same as the rest of `Cruise` objects? Should we have a `static final` variable for it?
    - Same for the number of loaders needed
    - Should we also have a variable for the number of loaders that served the `Cruise`? (see level 2)

Level 1

Method needed in a Cruise class:

- **getServiceCompletionTime():** Returns the time in minutes of when the service will be completed.
- Example: If the Cruise arrives at 12PM, the service completion time is 12:30PM, which is 750 minutes from 00:00. (12 * 60) + 30 = 750)
- **Formula: ((time / 100) * 60) + (time % 100)** [where time is in the format MMHH (e.g. 1200)]

```
jshell> /open Cruise.java
jshell> Cruise c = new Cruise("A1234", 0);
jshell> c
c ==> A1234@0000
jshell> new Cruise("A2345", 30);
$.. ==> A2345@0030
jshell> new Cruise("A3456", 130);
$.. ==> A3456@0130
jshell> c.getServiceCompletionTime();
$.. ==> 30
jshell> new Cruise("CS2030", 1200).getServiceCompletionTime();
$.. ==> 750
jshell> new Cruise("D1010", 2329).getServiceCompletionTime();
$.. ==> 1439
jshell> /exit
```

Level 2

Use Loaders to serve Cruises:

- Design a Loader class that can serve a Cruise using a serve(Cruise) method
- Immutable class
- Variables needed:
  - ID [to differentiate between other loaders every time you “purchase” a new Loader when there are insufficient ones]
  - Should it keep an instance of the `Cruise` its serving currently?
  - Also, do we need to know when is the next time this `Loader` will be ready after serving its current `Cruise`?
  - Lastly, since the class must be immutable (ie. the internal state of the object should not be changed once it is created) what should we do with the variables?

Level 2

Method needed in a Loader class:

- **serve(Cruise):**
  - Returns a string representation like this: `Loader id serving cruiseid@cruisearrivaltime`
  - If the loader cannot serve a cruise, or that cruise has already been served, that loader will not serve the cruise, and returns null.

Think:

How do we know if the loader can serve a cruise? (Hint: when its ready time is more than the cruise arrival time)

How do we know when the cruise has already been served? (Hint: use a counter in the Cruise class? See previous hint in level 1 variables)

Level 2

```
jshell> /open Cruise.java
jshell> /open Loader.java
jshell> new Loader(1);
$.. ==> Loader 1
jshell> new Loader(1).serve(new Cruise("A1234", 0));
$.. ==> Loader 1 serving A1234@0000
jshell> new Loader(1).serve(new Cruise("A1234", 0)).serve(new Cruise("A2345", 30));
$.. ==> Loader 1 serving A2345@0030
jshell> new Loader(2).serve(new Cruise("A1245", 2330));
$.. ==> Loader 2 serving A1245@2330
jshell> new Loader(2).serve(new Cruise("A1245", 2330)).serve(new Cruise("A2345", 2359));
$.. ==> null
jshell> new Loader(2).serve(new Cruise("A1245", 2330)).serve(new Cruise("A2345", 2359));
$.. ==> null
jshell> Cruise c = new Cruise("CS2030", 0);
jshell> new Loader(3).serve(c);
$.. ==> Loader 3 serving CS2030@0000
jshell> new Loader(3).serve(c);
$.. ==> null
jshell> /exit
```

Level 3

Creating a BigCruise class:

- Are the behaviors of the `BigCruise` class similar to that of the `Cruise` class? If yes, should it extend the `Cruise` class?
- If we use inheritance here, what are the major differences between the `Cruise` and `BigCruise` class? → varying number of loaders required and time required [should they be `static` just like in `Cruise`?]
- The arguments of the `BigCruise` constructor are its ID, time of arrival, number of loaders required, and time required, in that order.

Level 3

```
jshell> /open Loader.java
jshell> /open Cruise.java
jshell> /open BigCruise.java
jshell> Cruise b = new BigCruise("B0001", 0, 2, 60);
jshell> b.getServiceCompletionTime();
$.. ==> 60
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);
$.. ==> null
jshell> new Loader(4).serve(new BigCruise("B2345", 0, 1, 29)).serve(new Cruise("A0000", 29))
$.. ==> Loader 4 serving A0000@0029
jshell> new Loader(5).serve(new BigCruise("B3456", 0, 2, 31)).serve(new Cruise("A2345", 30))
$.. ==> null
jshell> /exit
```

Level 4 (The Big Picture)

Input: The number of cruises that arrive daily, and the daily cruise schedule

```
4
B1111 1300 2 60
A1111 1359
A1112 1400
A1113 1429
```

Output: allocation schedule for the loaders

```
Loader 1 serving B1111@1300
Loader 2 serving B1111@1300
Loader 3 serving A1111@1359
Loader 1 serving A1112@1400
Loader 2 serving A1113@1429
```

Level 4 (The Big Picture)

- For each cruise, check through the inventory of loaders, starting from the loader first purchased and so on. [should you use a data structure (what type) to keep track of the loaders?]
- The first (or first few) loaders available will be used to serve the cruise. [when do you know the cruise is fully served? Hint: create a method in the cruise class to check if there are already enough loaders serving it → ie you need a counter in your cruise class]
- If there are not enough loaders, purchase a new one(s), and that loader(s) will be used to serve the cruise. [Once you reach the end of your data structure storing the loaders in the loop while serving, you should create a new loader]

Level 4 (The Big Picture)

- In your main class:
  - Read the number of cruise and cruise schedule (input) and serve the cruise accordingly
  - You can create two methods in the main class:
    - readCruise()
    - serveCruise(Cruise)

```
/**
 * Just the main method, no big deal.
 * @param args Just the arguments, no big deal either.
 */
public static void main(String[] args) {
    numberOfScheduleEntries = SCANNER.nextInt();
    for (int i = 0; i < numberOfScheduleEntries; ++i) {
        serveCruise(readCruise());
    }
}
```

Comments

- Think of what other methods you need in your classes that will be helpful (One of such methods is to check if a cruise is fully served and also to check if a loader can serve the new cruise [see previous hints])
- You can have other helper methods as well to make your code less verbose and neater
- Also take note:
  - Compile before using jshell
  - `checkstyle *.java` (check for checkstyle in your scripts)

All the best!

Wai Lun - wailun@u.nus.edu

Si Mun - simun@u.nus.edu