SWEN20003
Object Oriented Software Development
Workshop 6

Rohyl Joshi

Semester 1, 2020

1 Discussion

Interfaces

1. What is an interface?

2. How is implementing an interface different to inheriting from a class?

3. In what situations would we use one (or more) interfaces over inheritance?

4. Define an interface called `Transferable` that allows objects to be represented in a network-friendly format (e.g. for sending to other computers). What are some classes that might use this interface? Why are we using an interface instead of inheritance?

5. How do polymorphism and interfaces relate?

6. What is the `Comparable` interface? What implementations of `compareTo` might we use to compare classes:

   - `Student`
   - `Fruit`
   - `MusicalArtist`

2 Design

You are tasked with improving the design of a software called +Etacolla. This software allocates students class times for their enrolled subjects. Here is the current core of +Etacolla.

```java
public class Etacolla {
    private final MonsterUniService mUniService;
    public Etacolla(){
        this.mUniService = new MonsterUniService();
    }
    public void generateTimetable(Student student){
        List<String> subjectNames = mUniService.getEnrolledSubjectCodes(student);
        List<Subject> subjects = new ArrayList<>();
        for (String subjectName : subjectNames){
            Subject subject = mUniService.getSubject(subjectName);
            subjects.add(subject);
        }
        // allocate activities to student ...
        List<Activity> allocated = allocatePreferences(student.getPreferences(), subjects);
        for (Activity activity : allocated){
            mUniService.registerStudentInActivity(student, activity);
        }
    }
}
```

+Etacolla's first client was Monster University, but more universities are hopping on board. How can we make the above design more adaptable and resilient to change?

3 Implementation

Work on Project 1.

Assessable Question

We are designing a statistics collection software for a transit system. You are provided with a log of passenger information. Our very first task is to ensure that the log items are 'sorted'. Each log item is in the format: `USER_ID:DEPARTURE_CITY:DESTINATION_CITY:TIME_ARRIVAL`. There is no guaranteed order to the items in log file. Every field is of type `String` **except** `TIME_ARRIVAL` which will always be a positive integer. You can assume nothing of any field apart from its data type and that it is non-empty.

Your Task

Fill in the provided method `public String[] processLogLines()` in the file `TransitProcessor.java`. The method should return an array of type `String`. The array should contain the log lines in sorted order for a given input. Each line is an element in the array. You do **not** need to perform I/O. This is automatically done for you in the `LogProcessor` parent class, and also in the provided testing class `TransitProcessorTest`. **You should not modify the `LogProcessor` abstract class**. As always, you are free to create your own classes to support your solution within the same directory.

The log lines must be sorted by the following criteria (in order) in accordance with their data type's natural ordering:

- Arrival Time
- Departure City
- Destination City

Example Input Log Lines

```
5:Melbourne:Sydney:2
2:Cairns:Townsville:0
5:Sydney:Melbourne:8
25:Port Hedland:Weipa:6
36:Annerly:Gold Coast:2
922:Port Hedland:Perth:6
```

Example Output Log Lines

```
2:Cairns:Townsville:0
36:Annerly:Gold Coast:2
5:Melbourne:Sydney:2
922:Port Hedland:Perth:6
25:Port Hedland:Weipa:6
5:Sydney:Melbourne:8
```

Package: The package consists of an abstract class `LogProcessor`, the skeleton for your solution `TransitProcessor`, and a testing class `TransitProcessorTest`. The `LogProcessor` constructor **already** handles the reading in of lines from a log file, and stores the lines in an attribute: `private final String[] lines`. To acccess these lines in the child class (i.e. `TransitProcessor`) you can call the inherited `public String[] getLines()` method. The testing class has been provided for your convenience.

You must only use the Java standard library to develop your solution[1]. At the very least, you should maintain the following structure:

```
username-workshops
└─ workshop-6
   └─ src
      └─ TransitProcessor.java
```

Deadline: Friday the 15th May, 11:59pm
Remember, git is supposed to be integrated into your workflow, so you should be managing your IntelliJ project directly within your local repository, not somewhere else and copying it in or manually uploading to GitLab. As always, you can see an example repository here.

[1]**The javafx package is not part of JDK 11**