SYSC 2004 Object-0riented Software Development

Lab 5

Lab 5:

Background Reading

- *Objects First with Java*, Chapters 1-4, 5.4, and 6.

Objective

The objective of this lab is to continue developing the application you started in last week's lab, and to practice using Java's `ArrayList` class to maintain a collection of objects.

Note that this lab builds on your solution to lab #4. If you didn't complete lab #4, you should do that **before** the start of your lab #5.

Getting Started

1. Download file `train-v2.zip` from cuLearn. Save the file on the desktop.
2. Locate your train project from Lab #4 and save that folder on the desktop.
3. Right-click on the `train-v2.zip` folder and select Extract All... to extract all the files into a folder called `train-v2`.
4. Launch BlueJ.
5. Open the *train-v2* project in BlueJ.
6. The cars in the train are modelled by class `Car`, which you implemented in last week's lab. Here is how to add a copy of this class to the *train-v2* project.
    1. From the BlueJ main menu, select Edit > Add Class from File... A dialogue box will open.
    2. Browse to the `train` folder that contains your work from Lab 4. Select class `Car`, and click the Add button. A copy of the class will be placed in the *train-v2* project. (This will not delete the `Car` class from your *train* project, and any changes that you make to `Car` in *train-v2* will not affect the original class in *train*.)

Developing Class `Train`

A passenger train is made up of one or more business-class cars and economy-class cars. Each car contains several seats. When you purchase a ticket for a train trip, a specific seat is booked for you. Purchasing a ticket to ride in a business-class car costs more than a ticket for a seat in an economy-class car.

The *train-v2* project contains an incomplete implementation of a class that models a train. `Train` has a private field, `cars`, the list of cars in the train. The list is implemented as an `ArrayList` of `Car` objects. (A summary of some of the `ArrayList` methods is provided at the end of this lab.)

Your job is to complete the implementation of `Train`. You cannot define additional constructors or methods in `Seat`, `Car`, or `Train`, or change the specification of any of the constructors and methods. All fields must be private; i.e., it is not permitted for objects of one class to directly access the fields of objects of another class.

You can test your `Train` class interactively, using BlueJ's object bench. You can use the BlueJ debugger (which you learned how to use in Lab 3) to locate any bugs in your code. In addition, you have been provided with a test class called `TrainTest`. To use this class, you must enable BlueJ's testing tools (see Lab 2 if you've forgotten how to do this). After you write each method (and it compiles without errors), click the Run Tests button. The test results will tell you if your methods are working. Write the methods in the order listed below, and make sure you finish each step before moving on to the next step. At this point in the course, you are not expected to understand all of the code in `TrainTest` (in particular, the use of methods `assertEquals()`, `assertSame()`, `assertTrue()` and `assertFalse()`), but you are welcome to try to figure out what each of the test methods do.

Part 1 - Developing Class `Train` Section A

1. Add Javadoc comments at the top of the class. See class `Seat` for an example.
2. Class `Train` has one constructor, which has no parameters. Write this constructor so that it initializes the `cars` field with the reference to a new `ArrayList` object that can store references to `Car` objects. This constructor does **not** initialize the `ArrayList` with `Car` objects. If your constructor works, `testCreateEmptyTrain()` should pass.
3. Generate the Javadoc and check that it looks good for the portions of class `Train` completed.
4. Get Part 1 checked by a TA.

Part 2 - Developing Class `Train` Section B

1. Method `addCar()` has a single parameter of type `Car` and returns nothing. This method adds the specified `Car` object to the list of cars in this train. Write this method. If your method works, `testAddCar()` should pass.
2. Did you remember to add Javadoc comments? Generate the Javadoc and check that it looks good for the portions of class `Train` completed.
3. Get Part 2 checked by a TA.

Part 3 - Developing Class `Train` Section C

1. Method `issueTicket()` has a single `boolean` parameter and returns a `boolean`. If the parameter is `true`, the method attempts to issue a ticket for a seat in a business-class car; if `false`, it attempts to issue a ticket for a seat in an economy-class car. This method will attempt to book a seat in the first car of the appropriate type, but if a seat is not available it will attempt to book a seat in the second car of the appropriate type, and so on. This method must ensure that a request to issue a ticket in a business-class car will never result in a seat being booked in an economy-class car, and vice-versa. If a ticket is successfully issued, the method returns `true`; otherwise, it returns `false`. This method must NOT call the `seats()` method in class `Car` (that method is reserved for use by test cases). Hint: recall that class Car has a method called `bookNextSeat()`. Write this method. Before testing this method, from the BlueJ menu select View > Show Terminal. If your method works, `testIssueTicket()` should pass, and the tickets will be "printed" in the Terminal Window.
2. Did you remember to add Javadoc comments? Generate the Javadoc and check that it looks good for the portions of class `Train` completed.
3. Get Part 3 checked by a TA.

Part 4 - Developing Class `Train` Section D

1. Method `cancelTicket()` has two parameters, an integer car id and an integer seat number, and returns a `boolean`. If the car id and seat number are valid, and if the specified seat in the specified car has been booked, this method cancels the booking for that seat and returns `true`. If the car id or seat number are not valid, or if the specified seat has not been booked, the method returns `false`. This method must NOT call the `seats()` method in class `Car` (that method is reserved for use by test cases). Hint: recall that class Car has a method called `cancelSeat()`. Write this method. If your method works, `testCancelTicket()` should pass.
2. Did you remember to add Javadoc comments? Generate the Javadoc and check that it looks good the entire `Train` class.
3. Get Part 4 checked by a TA.
4. You will build on Lab 5 in Lab 8, so be sure to save your work by e-mailing it to yourself or saving to a USB.

API Summary - Class ArrayList<E>

```
// Appends object o (of type E) to the end of this list. Returns true if
// successful, otherwise returns false.
boolean add(E o);

// Returns the number of elements in the list.
int size();

// Returns the object (of type E) at the specified position (index)
// in the list. (index must be >= 0 and < size()).
// The object is not removed from the list.
E get(int index);
```
