SYSC 2004 Object-0riented Software Development

Lab 4

Lab 4:

Background Reading

- *Objects First with Java*, Chapters 1-4, 6, and 7.

Objective

The objective of this lab is to build a simple application consisting of multiple classes, and practice using Java arrays to maintain a collection of objects.

Getting Started

1. Download file `train.zip` from cuLearn. Save the file somewhere on your M: drive or in C:/Temp. (The M: drive is recommended, because C:/Temp is erased when you log out.)
2. Right-click on the `train.zip` folder and select Extract All... to extract all the files into a folder called `train`.
3. Launch BlueJ.
4. Open the *train* project in BlueJ.
5. Note that 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. This lab involves understanding class `Seat` and completing class `Car`. The `Train` class will remain incomplete in this lab.

Part 1 - Exploring class `Seat`

1. The *train* project contains a complete implementation of a class that models seats in a passenger car in a train. The seats in the train's cars are modelled by class `Seat`. Double-click on the class icon for `Seat` and read the source code. Interactively create a `Seat` object that represents Seat #5, with a ticket price of $25.00. Save the object on the object bench.
2. Open the inspector on your object. Interactively invoke `number()`, `price()` and `isBooked()`. What values do these methods return?

3.

   a. Invoke `book()` to reserve the seat. What is the meaning of the value returned by this method? Invoke `isBooked()`. Explain the value returned by this method.

   b. Now that the seat has been booked, invoke `book()` again. What is the meaning of the value returned by this method? What does `isBooked()` return now? Why?

4.

   a. Now that the seat is booked, invoke `cancelBooking()` to cancel the seat's booking. What is the meaning of the value returned by this method? Invoke `isBooked()`. Explain the value returned by this method.

   b. Now that the booking has been cancelled, invoke `cancelBooking()` a second time. What is the meaning of the value returned by this method? What does `isBooked()` return now? Why?

5.

   a. Confirm that all the unit tests pass for class `Seat`. (Revisit lab #2 if you've forgotten how to do this.)

   b. Generate the Javadoc for the project (under the "Tools" menu), and confirm that it is complete and correct for class `Seat`.

6. Call over a TA and demonstrate that you understand how the `Seat` class works.

Introduction to Class `Car`

The *train* project contains an incomplete implementation of a class that models passenger cars in a train. `Car` defines three `private` fields:

1. `id`, which stores an integer identifier for this car (e.g., 1375, 2166);
2. `businessClass`, which stores a `boolean` value (`true` if the car is a business-class car, `false` if the car is an economy-class car); and
3. `seats`, the list of the seats in this car.

There are also four constants that define the prices of tickets and the number of seats in economy-class cars and business-class cars

This list is implemented as an array of `Seat` objects. Arrays in Java are objects. Although Java arrays are in many ways identical to arrays in C and C++, there are some important differences, which are described below.

Because arrays are objects, we use the `new` operator to create them. The following statements show how to declare a variable called `list`, of type `Seat`-array, and initialize it with a reference to an array object that has room for 10 `Seat` objects:

```
Seat[] list; // list will be an array of Seat objects
list = new Seat[10]; // list is an array of 10 Seat objects
```

We could do all of the above in one line:

```
Seat[] list = new Seat[10]; // list is an array of 10 Seat
objects
```

Individual elements of Java arrays are accessed by position, with the same syntax as C/C++. For example, `list[0]` is the first element, `list[1]` is the second element, and `list[i]` is element *i* + 1. In this example, each element in the array has type `Seat`. This means that each element in the array stores a reference to a `Seat` object. Initially, this can be a bit confusing: variable `list` stores a reference to an array object, and each element in the array stores a reference to a `Seat` object.

Creating an array object does not automatically initialize the elements of the array. In this example, the expression `new Seat[10]` does not automatically create 10 `Seat` objects and save the references to those objects in the array elements. We have to initialize the array ourselves. For example, the following statements initialize `list[0]` with a reference to an object that represents Seat #5:

```
Seat seat1;
seat1 = new Seat(5, 25.0);
list[0] = seat1;
```

We don't need variable `seat1`, so these statements can be simplified to:

```
list[0] = new Seat(5, 25.0);
```

Also, the price of a ticket for this seat is $25.00 - notice that 25.0 is the second value in the constructor's argument list, and is used to initialize parameter `cost` in the constructor's formal parameter list.

Java arrays differ from C/C++ arrays another way. Each array has an identifier called `length`, which is the maximum capacity of the array object. For example, the expression `list.length` always returns 10, because we created an array with capacity 10. Just as with C/C++, there is no way of determining the number of items that are currently stored in the array - that's something you have to keep track of, using a separate variable.

Note that `length` is an identifier **not** a method. Thus, we say `list.length` not `list.length()`.

Your job is to complete the implementation of `Car`. You cannot define additional constructors or methods in either `Seat` or `Car`, 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 `Car` class interactively, using BlueJ's object bench. You can also 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 `CarTest`. 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 `CarTest` (in particular, the use of methods `assertEquals()`, `assertTrue()` and `assertFalse()`), but you are welcome to try to figure out what each of the test methods does — it is fairly intuitive.

Part 2 - Developing Class `Car` Section A

1. Add Javadoc comments at the top of the class. See class `Seat` for an example.
2. Class `Car` has one constructor, which has two parameters. The first parameter is the integer id of the car (for example, 1385). The second parameter is a `boolean` value: if `true`, the car is a business-class car; if `false`, the car is an economy-class car. Write this constructor so that it initializes the `id` and `businessClass` fields with the values of the corresponding parameters, and initializes the `seats` field with a new list (array) of seats.

   Notes:

   - There are 30 seats in a business-class car, and 40 seats in an economy-class car. (Be sure to use the constants provided.)
   - Each element in the list must be initialized with a new `Seat` object.
   - The seats in each car are numbered consecutively, starting from 1.
   - The cost of a ticket for a seat in a business-class car is $125, while the cost of a ticket for a seat in an economy-class car is $50. (Be sure to use the constants provided.)
   - If your constructor works, `testCreateBusinessCar()` and `testCreateEconomyCar()` should pass.
   - Complete the Javadoc comments for the constructor. See the methods in class `Seat` for some examples.
3. Generate the Javadoc and check that it looks good for the portions of class `Car` completed.
4. Get Part 2 checked by a TA.

Part 3 - Developing Class `Car` Section B

1. Method `isBusinessClass()` has no parameters and returns a `boolean` value. It returns `true` if the car is a business-class car and `false` if it is an economy-class car. Write this method. If your method works, `testIsBusinessClass()` should pass.
2. Method `id()` has no parameters and returns an `int`. This method returns the id of the car. Write this method. If your method works, `testID()` should pass.
3. Did you remember to add Javadoc comments? Generate the Javadoc and check that it looks good for the portions of class `Car` completed.
4. Get Part 3 checked by a TA.

Part 4 - Developing Class `Car` Section C

1. Method `bookNextSeat()` has no parameters and returns a `boolean`. This method searches the list of seats in the car and reserves the first available seat that it can find. After booking a seat, this method should print a ticket by invoking `printTicket()` and then return `true`. If no seats in the car are available, this method returns `false`. Write this method. Before testing this method, from the BlueJ menu select View > Show Terminal. If your method works, `testBookNextSeat()` should pass, and the tickets will be "printed" in the Terminal Window.
2. Method `cancelSeat()` has a single parameter (an integer seat number) and returns a `boolean`. If the seat number is valid and if that seat has been reserved, this method cancels the booking for the seat and returns `true`. If the seat number is not valid, the method returns `false`. If the seat number is valid, but the seat has not been reserved, the method returns `false`. Write this method. If your method works, `testCancelSeat()` should pass.
3. Did you remember to add Javadoc comments? Generate the Javadoc and check that it looks good the entire `Car` class.
4. Get Part 4 checked by a TA.
5. You will build on your Lab 4 in Lab 5, so be sure to save your code by e-mailing it to yourself or putting it on a USB so that you can use it next week!
