Introduction to OO Programming

Workshop Instructions

4 – Java Library Classes

JAVA LIBRARY CLASSES

Objectives

The objective of this workshop is to practice some important Java library classes (ArrayList, HashMap, Date, DateFormat) and handle exceptions.

Exercise

Setting up

1) Go to the course IVLE site, and download the solution for the second workshop: **Wshp3Solution-src.zip**.

2) Unpack the zip file in to a new Eclipse project, and then Refresh. Use the files as a starting point for this exercise.

UML Diagram

Read the following description of the classes and draw a class diagram.

Use ArrayList in the Club class (Use TDD approach)

Arrays are not a very flexible way of storing a dynamic list such as our members' list. We will modify class **`Club`** so that it uses an **`ArrayList`** to keep its list of members, instead of its current **`Member[]`** array object.

3) Modify the methods **`getMembers()`**, **`addMember()`**, **`removeMember()`** and **`showMembers()`** to use an **`ArrayList`** instead of a **`Member[]`** array.

4) Make sure that the same **`ClubApplication`** code as at the end of the previous workshop works.

Use HashMap in the Club class (Use TDD approach)

If we need to retrieve objects by key, a **`HashMap`** object is a suitable choice. We will modify class **`Club`** so that it uses a **`HashMap`** to keep its list of facilities. The *key* for this table should be the facility's name.

5) Provide methods for handling facilities, equivalent to those provided for members. Implement the following methods in class **`Club`** using **`HashMap`**: **`getFacility(String name), getFacilities(), addFacility(), removeFacility()`** and **`showFacilities()`**

6) Write a **`show()`** method which invokes **`showFacilities()`** and **`showMembers()`** to list the content of both lists, one after the other.

7) Modify **`ClubApplication`** so it invokes the **`addFacility()`** method rather than instantiating the **`Facility`** object directly, and test the new code.

Add bookings (Use TDD approach)

Club facilities may be booked (reserved) by members for a given period of time. We will use a **`Booking`** class to represent these bookings.

8) Create a **`Booking`** class, which references the **`Member`** and **`Facility`** objects, and contains two **`Date`** objects (the start date and the end date for the booking). Add a constructor that will accept initialisation values for each of these members. Ensure all members are private, and add an accessor method for each (e.g. **`getMember()`**, and so on).

9) Add methods **`overlaps()`** to the **`Booking`** class. This will accept another **`Booking`** object as the parameter, and will return true if the two bookings overlap in time (clearly, only for the same facility).

10) Add method **`toString()`** (and optionally **`show()`**) to display the member's name, the facility's name, and the beginning/end dates (you can simply use the **`toString()`** methods of the various objects, we will format the dates later).

Test the Booking class

11) In the **`ClubApplication`** class, instantiate a **`SimpleDateFormat`** object that will create **`Date`** objects by parsing a text string in a particular date format.
    Refer to the documentation- a format string such as **`"d-MMM-yyyy H:mm"`** will allow you to write a date such as: "**`1-MAR-2007 15:00`**"

12) Using the newly created methods and object, create a **`Booking`** object in the **`ClubApplication`** class, and call its **`toString()`** method to verify your code.

13) Since the **`toString()`** method of **`Booking`** prints the start and end dates in a verbose format, you can use a **`SimpleDateFormat`** object within this method to format the output string (you can use the same format as above). You should explore the possibility of all **`Booking`** objects sharing the same formatting object.

Create a "bad booking" exception

Clearly, you cannot create a **`Booking`** object with arbitrary attribute values- in many cases, the object would not make sense, and the constructor for **`Booking`** should throw an exception.

14) Create an exception class **`BadBookingException`**, ensuring it can support a **`String`** message.

15) Modify the constructor of the **`Booking`** object so it throws **`BadBookingException`** when attempting to make a booking:
    - without a **`Member`** reference, or
    - without a **`Facility`** reference, or
    - without either a start date or an end date, or
    - with a start date which is later then the end date

16) Modify the **`ClubApplication`** class, to handle the new exception appropriately. Try creating some objects with bad parameters, and verify that everything works.

Create a container for the bookings (Use TDD approach)

Your club should have a container for bookings made by members. This will be a new class called **`BookingRegister`**, which will keep lists of **`Booking`** objects, indexed by **`Facility`**. In other words, **`BookingRegister`** will contain a **`HashMap`** in which the *key* is a **`Facility`** object, and the *value* is an **`ArrayList`** containing all **`Booking`** objects for that **`Facility`**.

17) Create a class **`BookingRegister`** and give it a private **`HashMap`** attribute. Make sure the table is instantiated when we create the **`BookingRegister`**.

18) Add a method **`addBooking()`** to class **`BookingRegister`**. This method will accept reference to the **`Member`** and **`Facility`** objects, and to the start and end **`Date`** objects. The **`addBooking()`** method should
    - instantiate a **`Booking`** object
    - retrieve the **`ArrayList`** corresponding to the given **`Facility`** from the **`HashMap`**, using the **`Facility`** object as the key
    - if no **`ArrayList`** object is retrieved (i.e. this is the first booking for the **`Facility`**), a new empty **`ArrayList`** object must be created, and put into the table, using the **`Facility`** object as the key
    - go through all the existing **`Booking`** objects in the **`ArrayList`**, and make sure they do not overlap with the new booking; if there is an overlap, the **`addBooking()`** method must throw a **`BadBookingException`**
    - if there are no overlaps, the new **`Booking`** object is added to the **`ArrayList`**.

19) Add a method **`getBookings()`** to class **`BookingRegister`**. This method will accept as parameters a **`Facility`** object, and two **`Date`** objects (which specify a date range). It must return a **`ArrayList<Booking>`** object, containing all **`Booking`** objects for the given **`Facility`** that fall within the time range specified.

20) Add a method **`removeBooking()`** to class **`BookingRegister`**. This method will accept a reference to a **`Booking`** object as a parameter, and will remove that booking from the list of bookings for the relevant **`Facility`**.

21) Include an instance of **`BookingRegister`** in the **`Club`** class. Also in the **`Club`** class, add a method **`addBooking()`** which will accept the membership number of a member, the name of a facility, and a pair of **`Date`** objects. This method should obtain references to the appropriate **`Member`** and **`Facility`** objects, then use the **`BookingRegister`** object to store the booking.

22) Add method **`getBookings()`** to the **`Club`** class. As parameters, it will accept the name of a facility, and two **`Date`** object (which specify a date range). This method will simply use the **`BookingRegister`** to get all **`Booking`** objects within the time interval specified. You can also add a method **`showBookings()`** to the **`Club`** class, which accepts the same parameters, and uses the **`Booking.show()`** method to print each retrieved booking to the screen.

Make sure that you use the TDD method to develop the above classes.

