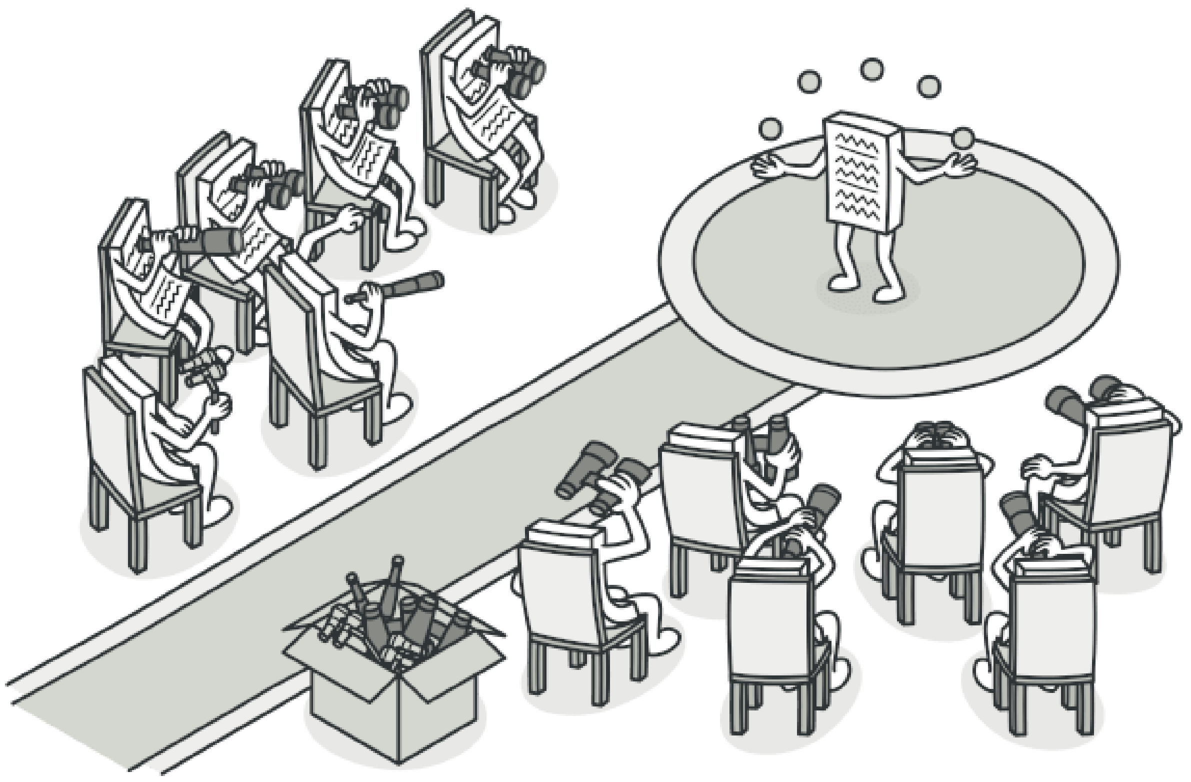


Observer Assignment

29.12.2022

Ohad Wolfman, ID 316552496

Zev Kehat, ID 203283908

Overview

This project was built as an assignment in an Object-Oriented Programming course, taken as part of our BSc studies at Ariel University.

The objective of the assignment was to understand and implement the Observer design pattern in Java. This was done by creating a Java project which includes Observer objects and an Observable object, while implementing the class and methods to define the relationship between the two objects.

The project is composed of the following classes:

1. UndoableStringBuilder.java – This class defines the type "UndoableStringBuilder" (USB). A USB is a mutable object that contains two subtypes – a StringBuilder (stb) and a Stack (stk). The stb contains a mutable String object, thus allowing to append strings, delete, insert, and replace substrings, as well as to reverse the string. The stk is a mutable Stack object which keeps track of all modifications to the stb by saving all previous versions in LIFO order, thus allowing an undo() method to undo the changes made to the USB's stb.
2. Member.java – This class is an interface of the Observer type "Member". The interface contains a single method to update a given USB, called "update()".
3. ConcreteMember.java – This class is an implementation of the Member.java interface, to define the Observer type "ConcreteMember". A ConcreteMember is a type of Member, with a name, a GroupAdmin (see "GroupAdmin.java" class), and a USB object which shallow copies the GroupAdmin's USB. A ConcreteMember is able to register and unregister from a group of ConcreteMembers, or get an update from the group's GroupAdmin when it has changed its USB, in order to update its own USB.
4. Sender.java – This class is an interface, defining the methods of the Observable type "Sender". This interface includes the methods register, unregister, insert, append, delete, and undo.
5. GroupAdmin.java - This class is an implementation of the Sender.java interface, to define the Observable type "GroupAdmin". A GroupAdmin is a type of Sender, with a list of members and a USB. When a ConcreteMember registers to a GroupAdmin, it is added to the members list, and will receive updates of changes being done in the GroupAdmin's USB , thus changing its own USB. When the ConcreteMember unregisters from the GroupAdmin, it is removed from the list and will stop getting updates. Besides those two methods, the GroupAdmin implements the rest of the Sender interface's methods (append, insert, delete and undo) to modify the current object's USB, using the USB class's corresponding methods.

6. JvmUtilities.java – A utility class for tracking the Java Virtual Machine (JVM) resources allocated by the Operating System, in order to check that no resources are wasted in our implementation of the program.
7. Tests.java – A class allocated for running tests to verify our program works as intended. The class is available for review as part of this project's GitHub repo.

Code Overview and Methods

1. **General Comments:**
   a. **Programming Decisions** – The assignment allowed us to make a few implementation decisions, which we decided as follows:
      1) Our understanding of the assignment is that a ConcreteMember cannot be registered to two different GroupAdmins at the same time, which is why when a ConcreteMember that is registered to a GroupAdmin registers to a different GroupAdmin, it will automatically unregister from the first GroupAdmin. The problem with implementing this is that the new GroupAdmin trying to register a ConcreteMember doesn't "know" who that ConcreteMembers current GroupAdmin is and therefore the "conflict" can be settled by the ConcreteMember (see example[1]).
      2) A ConcreteMember's existence should be directly linked to the GroupAdmin with which it is associated, therefore the ConcreteMember constructor takes a GroupAdmin. Any other ways to create a ConcreteMember are not recommended.
      3) There is a time gap between a ConcreteMember registering to a GroupAdmin, and that GroupAdmin sending out a group update of the USB status. In order to avoid a ConcreteMember that is registered to a GroupAdmin not containing the updated USB, we have added a "personalized" update for that specific ConcreteMember, as part of the register() method.
      4) Once a ConcreteMember is unregistered from a GroupAdmin, its GroupAdmin field and USB field will be null. After registering the ConcreteMember to a new GroupAdmin, those fields will change accordingly.

   b. **Features:**
      1) ConcreteMembers are given names upon initialization, starting from "m1" and up to "mn[2]". We added this feature as a type of identifier, in order to differentiate between ConcreteMember objects.
      2) We added a toString() method for the ConcreteMember, so that any user can get a simple representation of its fields.

[1] To explain this issue, we can look at the following analogy: A student is supposed to attend a professor's lecture, but then another professor calls the student over to enter his own lecture. The second professor isn’t able to notify the first one that the student won't be attending, because he doesn’t know who's lecture the student was supposed to attend, unless the student tells him.

[2] "n" represents the number of ConcreteMembers initialized.

3) We added a printGroup() method for the GroupAdmin, which prints out the GroupAdmin USB's current string and the ConcreteMembers calling their toString() method. To enable the iteration of the GroupAdmin's ConcreteMembers we defined the members list to be of type ConcreteMember, but the inherited register() and unregister() methods call for a Member object[3]. Therefore, within those two methods, we added a cast of the Member object to be a ConcreteMember object. If changing the use/configuration of the program, it is recommended to verify this doesn't create an exception or error.

c. **Running the program:**

1) There is no main method added to the classes, but one can be added to either the GroupAdmin class or the ConcreteMember class.

2) Once a main method is added, create a GroupAdmin object and by calling the GroupAdmin methods, modify the GroupAdmins USB string as you'd like:

```
GroupAdmin myGroup1 = new GroupAdmin();
myGroup1.append("Hello");
myGroup1.append(" World!");
```

3) After creating the GroupAdmin, initialize new ConcreteMembers as you please, assigning them to the GroupAdmin created:

```
ConcreteMember m1 = new ConcreteMember(myGroup1);
ConcreteMember m2 = new ConcreteMember(myGroup1);
```

4) The user can continue creating and modifying GroupAdmins, as well as create, register, and unregister ConcreteMembers. Registering and unregistering can be done with calls from each object (identical results):

```
myGroup1.register(m1);
m1.register(myGroup1);
myGroup2.unregister(m2);
m2.unregister(myGroup2);
```

5) In order to see the status of a GroupAdmin, call the printGroup method:

```
myGroup1.printGroup();
```

2. **UndoableStringBuilder Methods:**

a. All of the following methods add the modified version of the USB's string to the USB's stack.

b. ***append(String str)*** – Adds given *str* to the end of the USB's string.

c. ***delete(int start, int end)*** – Deletes a substring from the *start* index until the *end*-1 index.

1) If end index is greater than the USB's string's length, the entire string will be deleted.

[3] The assignment guidelines did not allow changing the given interfaces.

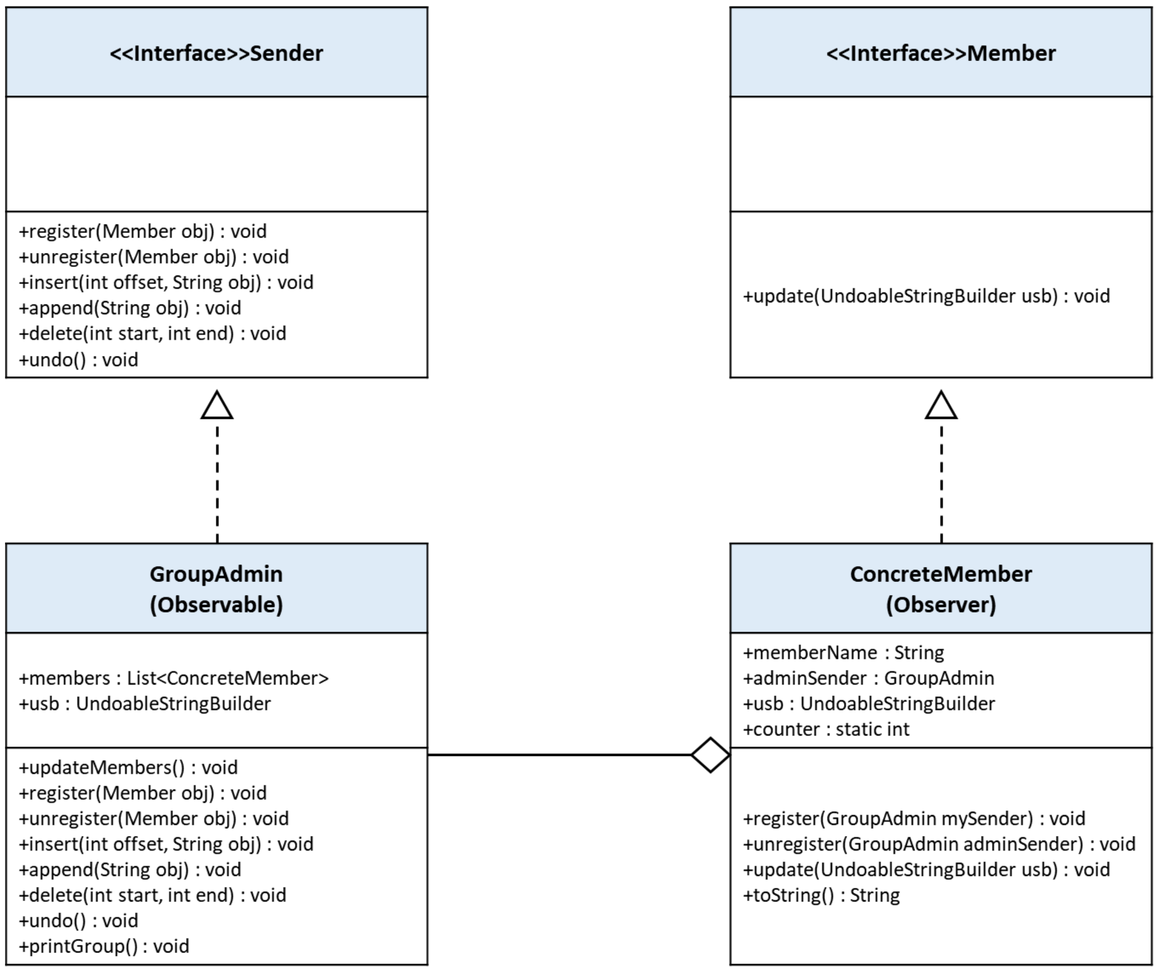
        2) If start=end, no substring will be deleted.

   d. ***insert(int offset, String str)*** – Inserts specified *str* into the USB's string, starting at index *offset*, and pushes forward the rest of the string (all chars appearing after *offset* will be of original index + *offset*).
   e. ***replace(int start, int end, String str)*** – Replaces specified substring from *start* index to *end*-1 index from USB's string with specified *str*.
   f. ***reverse()*** – Reverses the order of the USB's string (last is first, first is last).
   g. ***undo()*** – Undoes the last modification of the USB's string. Running it several times will undo the operations from latest to earliest.
   h. ***toString()*** – Prints the USB's current string.

3. **<u>Member Methods:</u>**
   a. ***update(UndoableStringBuilder usb)*** – Abstract interface method.
4. **<u>ConcreteMember Methods:</u>**
   a. ***ConcreteMember(GroupAdmin adminSender)*** – ConcreteMember constructor, that initializes the ConcreteMember and registers it to the specified GroupAdmin.
   b. ***register(GroupAdmin mySender)*** – Adds the ConcreteMember to the specified GroupAdmin's members list, and shallow copies its USB.
      1) The ConcreteMember's adminSender field will now point to specified GroupAdmin.
      2) If the ConcreteMember is already registered to the GroupAdmin, nothing will happen.
      3) If the ConcreteMember is already registered to a different GroupAdmin, it will first unregister from it and only then register the new one.
   c. ***unregister(GroupAdmin adminSender)*** – Removes the ConcreteMember from the specified GroupAdmin's members list.
      1) The ConcreteMember's USB and adminSender will become null.
      2) If the ConcreteMember attempts to unregister from a GroupAdmin it doesn't belong to, nothing will happen.
   d. ***update(UndoableStringBuilder usb)*** – Copies the specified USB to the ConcreteMember's USB (shallow copy).
   e. ***toString()*** – Returns a string representation of the ConcreteMember.
   f. ***getAdminUser()*** – Returns this instance's GroupAdmin. Used for Tests class.
   g. ***getUsb()*** – Returns this instance's USB. Used for Tests class.
   h. ***getMemberName()*** – Returns this instance's name. Used for Tests class.

5. **<u>Sender Methods:</u>**
   a. All methods are abstract interface methods.
   b. ***void register(Member obj)*** – Register the observer object to observers collection.
   c. ***void unregister(Member obj)*** – Unregister the observer object from observers collection.
   d. ***void insert(int offset, String obj)*** – Inserts the string into this character sequence.
   e. ***void append(String obj)*** – Appends the specified string to this character sequence.
   f. ***void delete(int start, int end)*** – Removes the characters in a substring of this sequence.
   g. ***void undo()*** – Erases the last change done to the document, reverting it to an older state.
6. **<u>GroupAdmin Methods:</u>**
   a. ***updateMembers()*** – Updates ConcreteMembers registered to the GroupAdmin with its current USB state.
   b. ***register(Member obj)*** – Casts the Member object to a ConcreteMember and calls its register() method, with the current GroupAdmin instance as the argument.
   c. ***unregister(Member obj)*** – Casts the Member object to a ConcreteMember and calls its unregister() method, with the current GroupAdmin instance as the argument.
   d. ***insert(int offset, String obj)*** – Calls the insert() method of the GroupAdmin's USB, and updates members calling the updateMembers() method.
   e. ***append(String obj)*** – Calls the append() method of the GroupAdmin's USB, and updates members calling the updateMembers() method.
   f. ***delete(int start, int end)*** – Calls the delete() method of the GroupAdmin's USB, and updates members calling the updateMembers() method.
   g. ***undo()*** – Calls the undo() method of the GroupAdmin's USB, and updates members calling the updateMembers() method.
   h. ***printGroup()*** – Prints a string representation of the GroupAdmin's status:
      1) The current USB string.
      2) The registered members of the GroupAdmin, using the ConcreteMember's toString() method.
   i. ***getMembers()*** – Returns this instance's list of members. Used for Tests class.
   j. ***getUsb()*** – Returns this instance's USB. Used for Tests class.

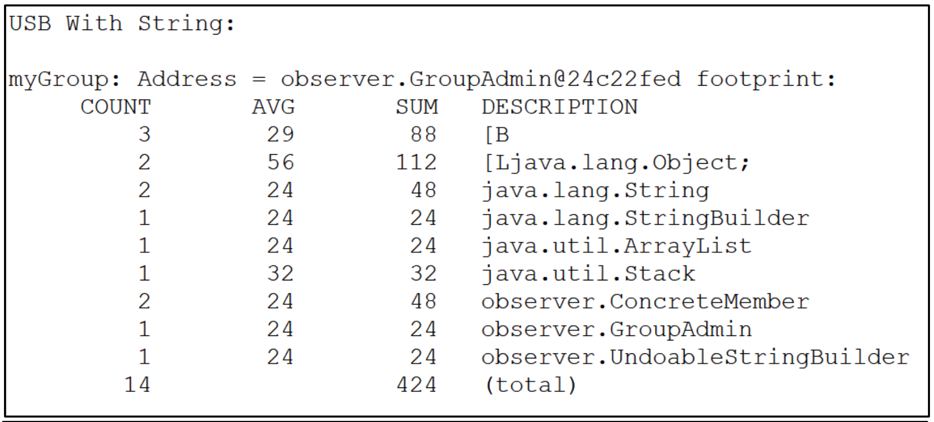
Project Diagram


<<Interface>>Sender
+register(Member obj) : void
+unregister(Member obj) : void
+insert(int offset, String obj) : void
+append(String obj) : void
+delete(int start, int end) : void
+undo() : void
<<Interface>>Member
+update(UndoableStringBuilder usb) : void
GroupAdmin (Observable)
+members : List<ConcreteMember>
+usb : UndoableStringBuilder
+updateMembers() : void
+register(Member obj) : void
+unregister(Member obj) : void
+insert(int offset, String obj) : void
+append(String obj) : void
+delete(int start, int end) : void
+undo() : void
+printGroup() : void
ConcreteMember (Observer)
+memberName : String
+adminSender : GroupAdmin
+usb : UndoableStringBuilder
+counter : static int
+register(GroupAdmin mySender) : void
+unregister(GroupAdmin adminSender) : void
+update(UndoableStringBuilder usb) : void
+toString() : String

Resources

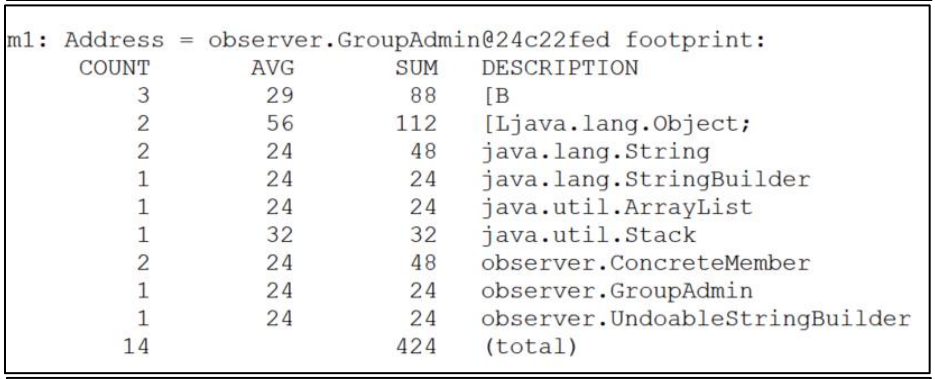
When testing the resources used by the program, we can see that because of our implementation's requirement of having each type - GroupAdmin and ConcreteMember - hold pointers to each other, that they end up containing each other as objects. This is because we were required to shallow-copy the GroupAdmin's USB, and not deep copy the USB's string or the entire USB itself, and also because we decided to implement the project objective in a way that would verify a ConcreteMember was removed from its GroupAdmin when trying to register to a different one.

Had we decided to allow (ignore) the "double registration", or created multiple USB objects in each ConcreteMember, or had the ConcreteMember deep-copy the USB's string only, we would be able to disconnect between the two types, and they would hold different memory sizes.

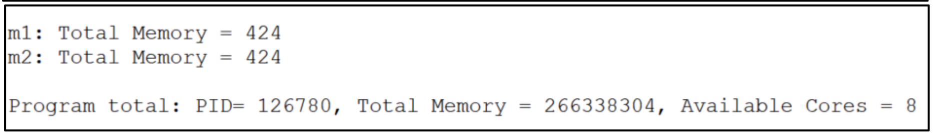


```
USB With String:

myGroup: Address = observer.GroupAdmin@24c22fed footprint:
     COUNT       AVG       SUM   DESCRIPTION
         3        29        88   [B
         2        56       112   [Ljava.lang.Object;
         2        24        48   java.lang.String
         1        24        24   java.lang.StringBuilder
         1        24        24   java.util.ArrayList
         1        32        32   java.util.Stack
         2        24        48   observer.ConcreteMember
         1        24        24   observer.GroupAdmin
         1        24        24   observer.UndoableStringBuilder
        14                 424   (total)
```



```
m1: Address = observer.GroupAdmin@24c22fed footprint:
     COUNT       AVG       SUM   DESCRIPTION
         3        29        88   [B
         2        56       112   [Ljava.lang.Object;
         2        24        48   java.lang.String
         1        24        24   java.lang.StringBuilder
         1        24        24   java.util.ArrayList
         1        32        32   java.util.Stack
         2        24        48   observer.ConcreteMember
         1        24        24   observer.GroupAdmin
         1        24        24   observer.UndoableStringBuilder
        14                 424   (total)
```



```
m1: Total Memory = 424
m2: Total Memory = 424

Program total: PID= 126780, Total Memory = 266338304, Available Cores = 8
```