Assignment: Design Patterns

**Objectives:**

* Reinforce the method used in class to learn design patterns
* Reinforce understanding of class/package diagrams
* Practice using design patterns to solve design problems

**Part I: The Learning Method** (40 pts)

In class, we have used the same method to learn five design patterns. Please summarize what you have learned by filling out the table below.

|  |  |  |
| --- | --- | --- |
| Design Patterns | Problem to be solved | Solution |
| Iterator | - The elements of an aggregate object need to be accessed and traversed without exposing its representation or data structures.  - New traversal operations need to be defined for an aggregate object without changing its interface. | - We define a separate iterator object that encapsulates accessing and traversing an aggregate object.  - Clients can use an iterator to access and traverse an aggregate without knowing its representation or data structures. |
| Composite | - A part-whole hierarchy must be represented so that clients can treat both part and whole objects uniformly.  - A part-whole hierarchy needs to be represented as a tree structure. | -We could define a unified Component interface for both part leaf objects and whole composite objects.  -For a tree structure, individual leaf objects implement the component interface directly, and composite objects forward requests to their child components. |
| Singleton | - We must ensure that a class has only one instance.  - The sole instance of a class must be accessed easily.  - A class should control its instantiation.  - The number of instances of a class must be restricted. | -We can hide the constructor of the class.  -We can define a public static operation like “getInstance()” that returns the sole instance of the class. |
| Observer | - We need to define a one-to-many dependency between objects so that when one object changes state, all of its dependents are notified | - We can encapsulate the core (\*or common or engine) components in a Subject abstraction, and the variable (\*or optional or user interface) components in an Observer hierarchy. |
| Strategy | - We need a family of interchangeable algorithms for accomplishing the same objective. | - We can capture the abstraction in an interface, and the bury the implementation details in derived classes. |

**Part II: Class/Package Diagram** (10 pts)

Please read the following diagram carefully, and answer Question 2.1 – 2.3.



2.1 How many packages are in the diagram? What are they?

Packages appear as rectangles with small tabs on top. There are two packages in the diagram, which are “java.util” and “java.lang”.

2.2 What do  and  represent respectively? What is the difference between them?

The arrow with a solid line represents inheritance. The arrow with a dashed line represents realization. The inheritance line refers to a type of relationship where one class is a child of another. To show inheritance, a solid line from the child class to the parent class is drawn using an unfilled arrowhead. The realization line is a relationship between two model elements, in which element (the client) realizes the behavior that the other model element specifies.

The realization line denotes the implementation of the functionality defined in one class by another class.

2.3 Why methods in an interface are *italicized*?

Methods in an interface are italicized to show that the methods are abstract.

**Part III: Application** (50 pts)

3.1 (10 points) Try to understand the following program.

|  |
| --- |
| import java.util.Iterator;  import java.util.LinkedList;  public class IteratorDemo {  public static void main(String[] args) {  LinkedList<String> cities = new LinkedList<String>();  cities.add("Chicago");  cities.add("Denver");  cities.add("Miami");  cities.add("Los Angeles");  cities.add("Seattle");    Iterator<String> iterator1 = cities.iterator();  Iterator<String> iterator2 = cities.iterator();  System.out.println("Iterator1 type for the datastructure is: " + iterator1.toString());  System.out.println("Iterator2 type for the datastructure is: " + iterator2.toString());  while (iterator1.hasNext()){  String city1 = iterator1.next();  String city2 = iterator2.next();  System.out.println(city1+", "+city2);  }  }  } |

3.1.1 What is the expected output (based on your understanding of the code)?

It would print out the list of cities twice, then print the 1st city name twice, then the 2nd name twice underneath, and so on until there are no more items in the linkedlist.

3.1.2 Run the program in Eclipse, and compare the actual output with your answer in 3.1.1.

It actually printed the addresses of the linked lists first, contrary to my beliefs. Then, it would print each city name twice in sequence, with the first name going on top and the last one going on bottom.

3.1.3 What did you learn from this example? You may visit the following link for more insights:

<https://sourcemaking.com/design_patterns/iterator/java/1>

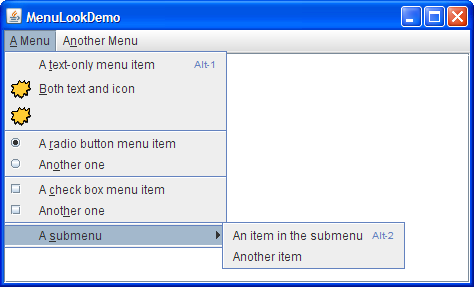
I learned that the toString() method is used to return a string representation of the elements of the collection and that the iterator simplifies the collection, allowing for many traversals to be active simultaneously.

3.1.4 Modify the program so that a Hashset is used instead of a Linkedlist. Which line(s) should be modified?

At the top “import java.util.LinkedList;” should be replaced with “import java.util.HashSet;”.

And in the line under “public static void main(String[] args) {“, “LinkedList<String> cities = new LinkedList<String>();” should be replaced with “HashSet<String> cities = new HashSet<String>();”.

3.2 (10 points) In Java Swing, menus are constructed using the composite pattern.

Briefly explain how the composite pattern is used to construct the above menu.

In this case, the component is “JMenuItem” and the composite is “JMenu”. The composite pattern used to construct the above menu in such a way that Composite objects forward requests to their child components (\*like a submenu). Individual Leaf objects could implement the Component interface directly (\*like the text-only menu item).

3.3 (10 points) In Windows 10, only one instance of the “Task Manager” object is needed. Design and implement a class called “TaskManager” using the singleton pattern. (Do not need to consider thread safety.)

public class TaskManager {

private static Boolean exists = false;

private static TaskManager instance;

private TaskManager(){

exists = true;

}

}

public static TaskManager createInstance(){

if(!exists) instance = new TaskManager();

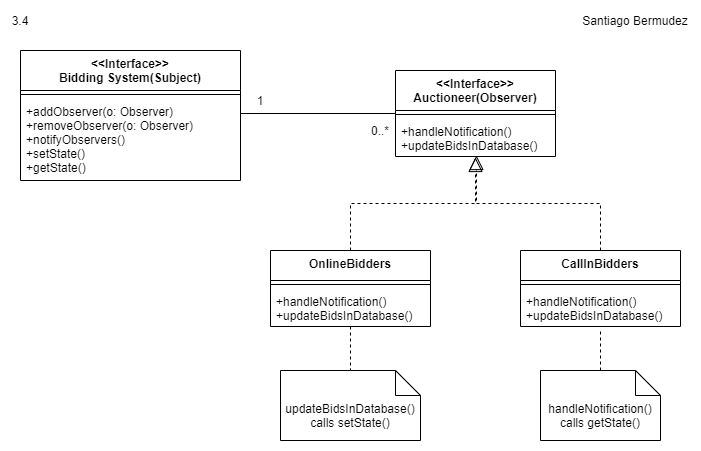
return instance;

}

3.4 (10 points) Consider a simple bidding system which has the following functionalities:

* Display the latest bid to online bidders
* Announce the latest bid to call-in bidders (who are on the phone)
* Save all bids to a database.

Use the observer pattern to design this system, and present your design using a class diagram.



3.5 (10 points) Design a program that can sort an array using different sorting algorithms, such as quick sort, merge sort, bubble sort, insertion sort. Present your design using a class diagram.

