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

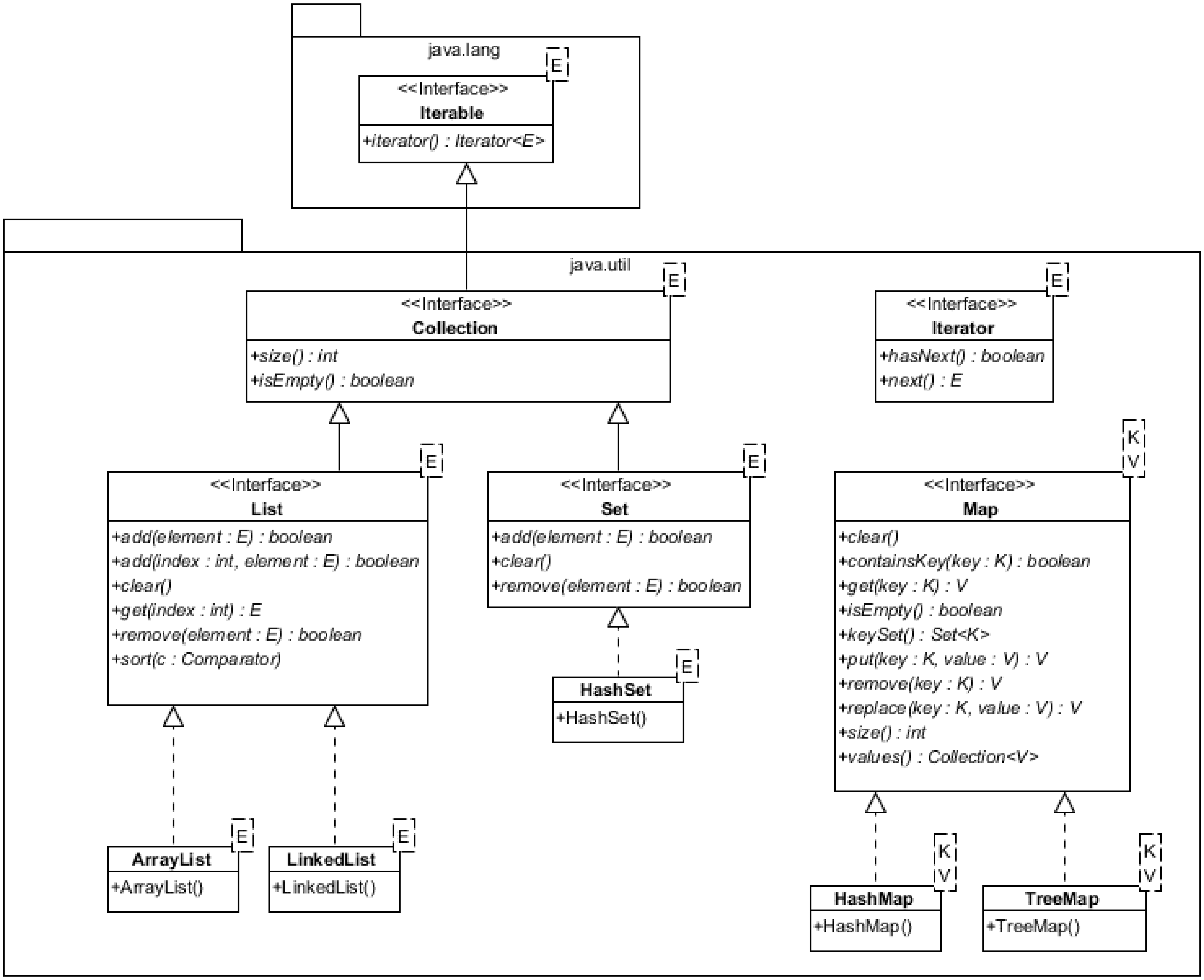
Part I: The Learning Method (40 pts)

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 should be **accessed** and **traversed** without **exposing its representation** (data structures). • New traversal operations should be defined for an aggregate object without changing its interface.	• **Define a separate (iterator) object** that encapsulates accessing and traversing an aggregate object. • Clients use an iterator to access and traverse an aggregate without knowing its representation (data structures).
Composite	• A part-whole hierarchy should be represented so that clients can **treat part and whole objects uniformly**. • A part-whole hierarchy should be represented as **tree** structure.	• Define a **unified Component interface** for both part (Leaf) objects and whole (Composite) objects. • Individual Leaf objects implement the Component interface directly, and Composite objects forward requests to their child components
Singleton	• How can it be ensured that a class has only one instance? • How can the sole instance of a class be accessed easily? • How can a class control its instantiation? • How can the number of instances of a class be restricted?	• Hide the constructor of the class. • Define a public static operation (getInstance()) that returns the sole instance of the class.
Observer	• You need to decorate a class that does not implement an explicit interface	• Objects often need to communicate with each other • Traditional message passing techniques tend to couple objects too tightly • Specialize the class to be decorated but still use delegation
Strategy	• Have a family of *interchangeable* algorithms for accomplishing the same objective	• Different text formatting algorithms (e.g., for paragraph formatting in a word processor) • Different metrics for finding the distance between, for example, colors

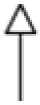
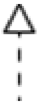
Part II: Class/Package Diagram (10 pts)

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?

- 2 – java.lang and java.util

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

- The solid line arrow represents an inheritance and the dotted arrow represents an interface

2.3 Why methods in an interface are *italicized*?

- They are implemented methods in an abstract class.

Part III: Application (50 pts)

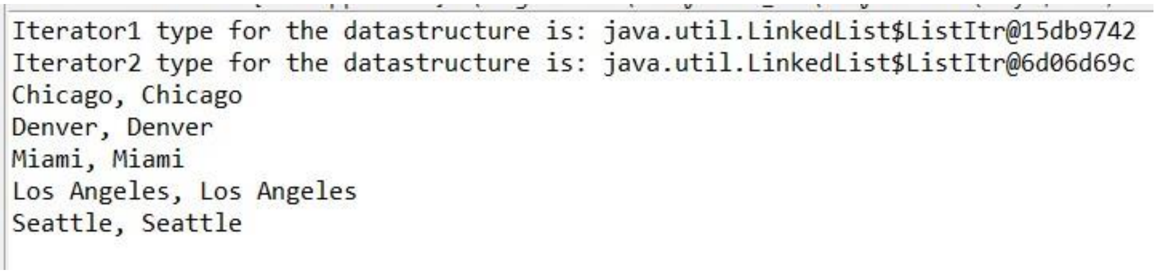
3.1 (10 points) Try to understand the following program.

```
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 iterators used and the cities

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



```
Iterator1 type for the datastructure is: java.util.LinkedList$ListItr@15db9742
Iterator2 type for the datastructure is: java.util.LinkedList$ListItr@6d06d69c
Chicago, Chicago
Denver, Denver
Miami, Miami
Los Angeles, Los Angeles
Seattle, Seattle
```

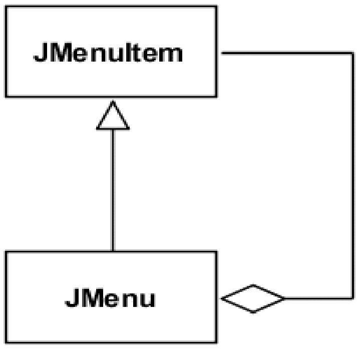
3.1.3 What did you learn from this example? You may visit the following link for more insights: https://sourcemaking.com/design_patterns/iterator/java/1

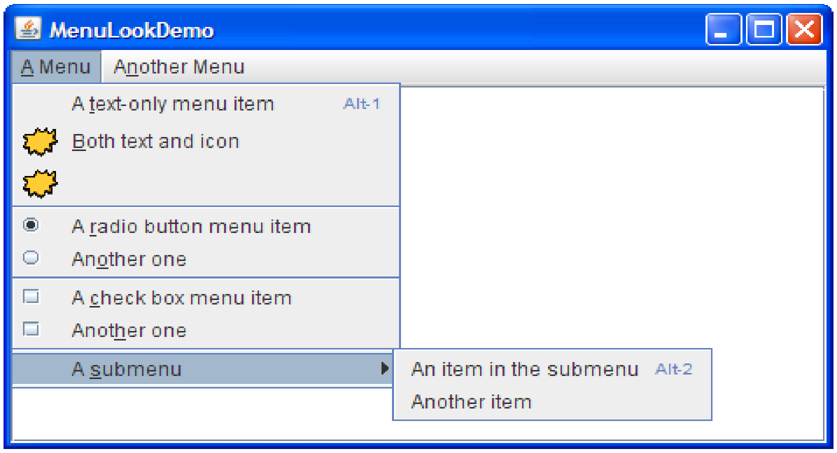
- The program provides access to its internal data structure. The user/client can accidentally or maliciously trash that data structure, which was what happened.

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

- After importing java.util.*, you would replace **LinkedList<String> cities = new LinkedList<String>();** with **HashSet<String> cities = new HashSet<String>();**
- The iterator strings would stay the same: **Iterator<String> iterator1 = cities.iterator();**

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

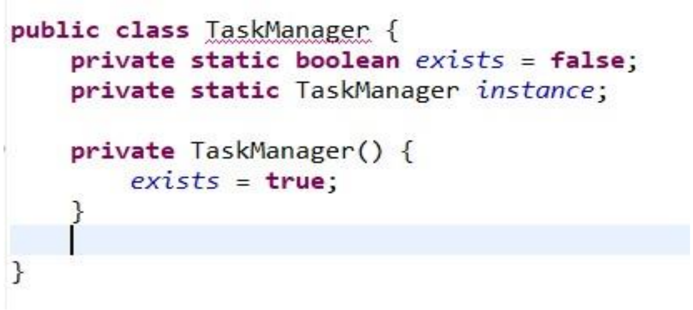




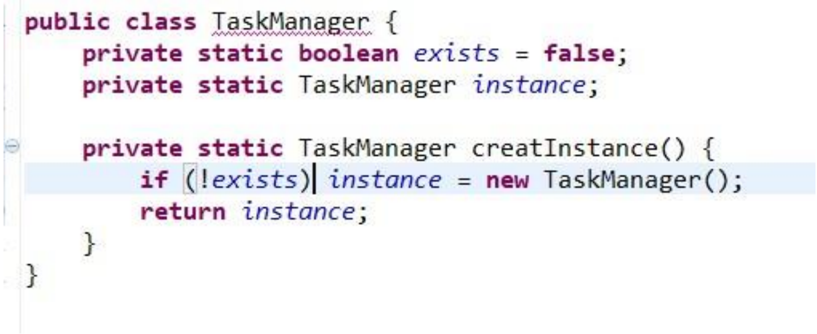
Briefly explain how the composite pattern is used to construct the above menu.

- The composite pattern ignores differences between **individual** objects and **compositions** of objects by grouping the different shapes into one application

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.)



```java
public class TaskManager {
    private static boolean exists = false;
    private static TaskManager instance;

    private TaskManager() {
        exists = true;
    }

}
```

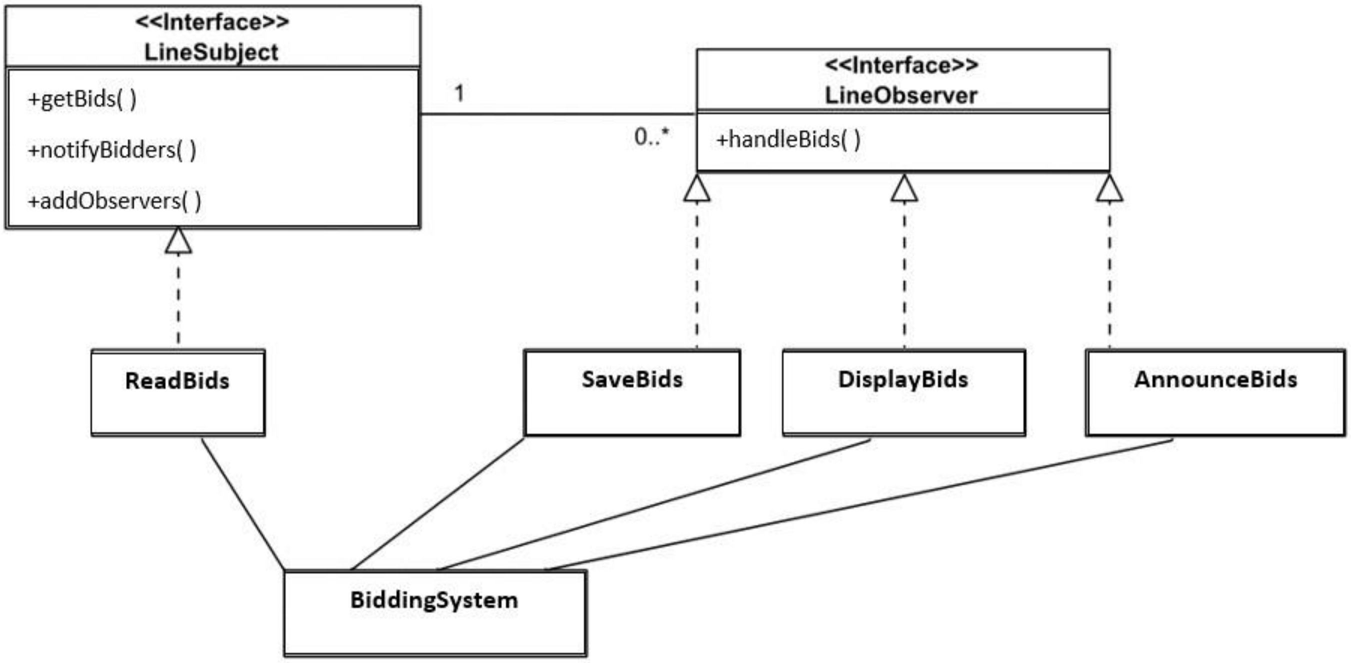


```java
public class TaskManager {
    private static boolean exists = false;
    private static TaskManager instance;

    private static TaskManager creatInstance() {
        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.


<<Interface>>
LineSubject
+getBids( )
+notifyBidders( )
+addObservers( )
1
0..*
<<Interface>>
LineObserver
+handleBids( )
ReadBids
SaveBids
DisplayBids
AnnounceBids
BiddingSystem