

# 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

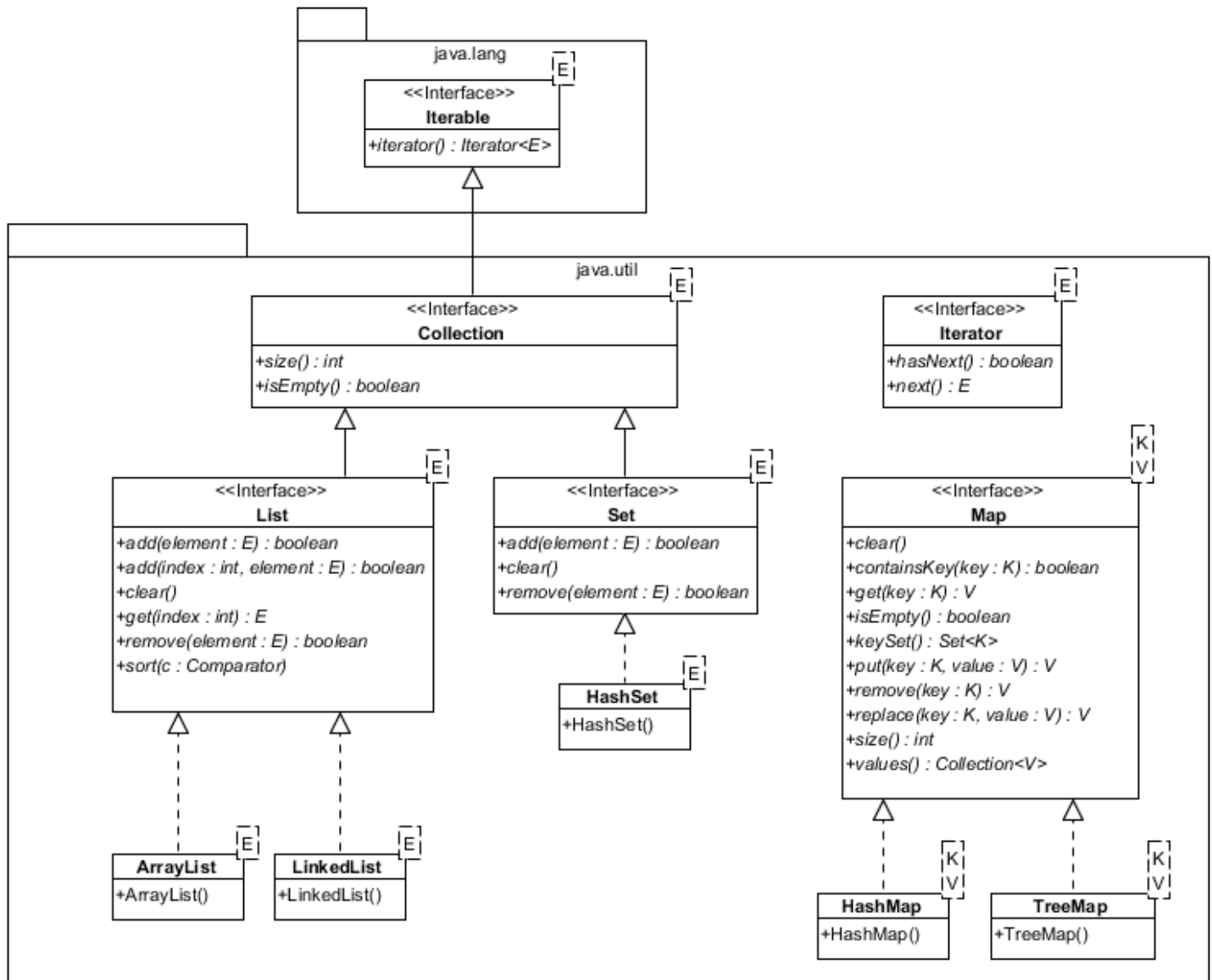
## Part I: The Learning Method (40 pts)

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	<ul style="list-style-type: none"> <li>• The elements of an aggregate object should be accessed and traversed without exposing its representation (data structures)</li> <li>• New traversal operations should be defined for an aggregate object without changing its interface</li> </ul>	<ul style="list-style-type: none"> <li>• Define a separate (iterator) object that encapsulates accessing and traversing an aggregate object</li> <li>• Clients use an iterator to access and traverse an aggregate without knowing its representation (data structures)</li> </ul>
Composite	<ul style="list-style-type: none"> <li>• A part-whole hierarchy should be represented so that clients can treat part and whole objects uniformly</li> <li>• A part-whole hierarchy should be represented as tree structure</li> </ul>	<ul style="list-style-type: none"> <li>• Define a unified Component interface for both part (Leaf) objects and whole (Composite) objects</li> <li>• Individual Leaf objects implement the Component interface directly, and Composite objects forward requests to their child components</li> </ul>
Singleton	<ul style="list-style-type: none"> <li>• How can it be ensured that a class has only one instance?</li> <li>• How can the sole instance of a class have accessed easily?</li> <li>• How can a class control its instantiation?</li> <li>• How can the number of instances of a class be restricted?</li> </ul>	<ul style="list-style-type: none"> <li>• Hide the constructor of the class</li> <li>• Define a public static operation (getInstance()) that returns the sole instance of the class</li> </ul>
Observer	<ul style="list-style-type: none"> <li>• Objects often need to communicate with each other</li> <li>• Traditional message passing techniques should not couple objects too tightly</li> </ul>	<ul style="list-style-type: none"> <li>• Define a one-to-many dependency between objects so that when one object changes state, all of its dependents are notified</li> </ul>
Strategy	<ul style="list-style-type: none"> <li>• Have a family of interchangeable algorithms for accomplishing the same objective (e.g., different text formatting algorithms, different metrics for finding the distance between, colors...)</li> </ul>	<ul style="list-style-type: none"> <li>• Encapsulates an algorithm inside a class</li> </ul>



## 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?

There are two packages: `java.lang` and `java.util`

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

 represents Inheritance

 represents Interface Realization/ Implementation

2.3 Why methods in an interface are *italicized*?

The methods in an interface are italicized is because we want to quickly tell whether an identifier names a class or an interface (since there are only two possibilities).

### 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)?

The output will prints out a list of cities : Chicago, Denver, Miami, Los Angeles, Seattle.

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

My prediction is slightly different from the actual output run in Eclipse. The output prints out two times the name of each cities.

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](https://sourcemaking.com/design_patterns/iterator/java/1)

I learn that iterator design pattern can allow many traversals to be active simultaneously, and separate collection algorithms from collection data structures.

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

The lines should be modified :

```
import java.util.LinkedList;

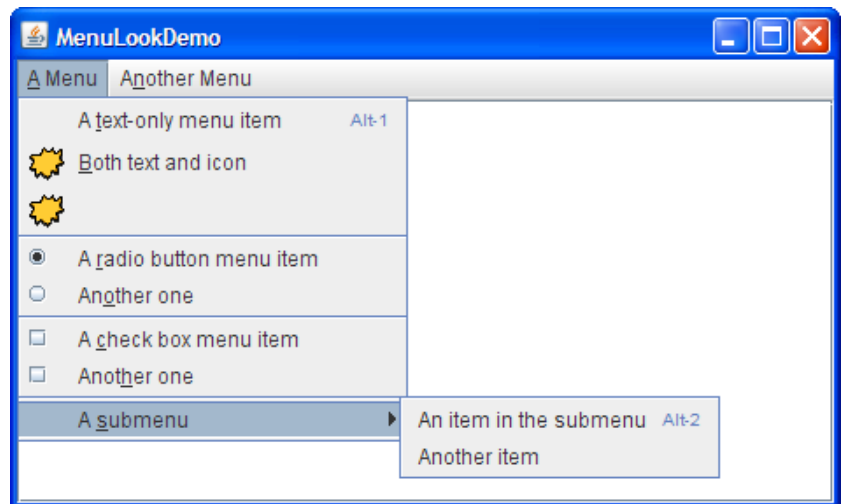
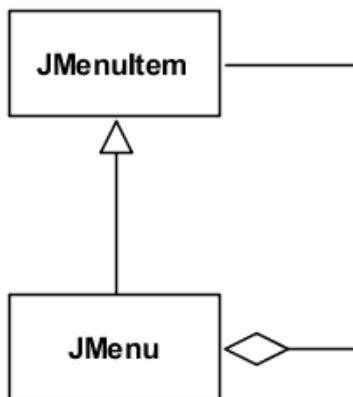
LinkedList<String> cities = new LinkedList<String>();
```

HashSet modified:

```
import java.util.HashSet;

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.

There will be a unified Component interface which is JMenuItem for other classes (JMenu...) that represented as tree structure. In this case, JMenu as a Composite object forward requests to their child components.

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.)

*// Make the constructor of class TaskManager as private*

```
public class TaskManager
{
    private static boolean    exists = false;
    private static TaskManager instance;
    private TaskManager()
    {
        exists = true;
    }
}
```

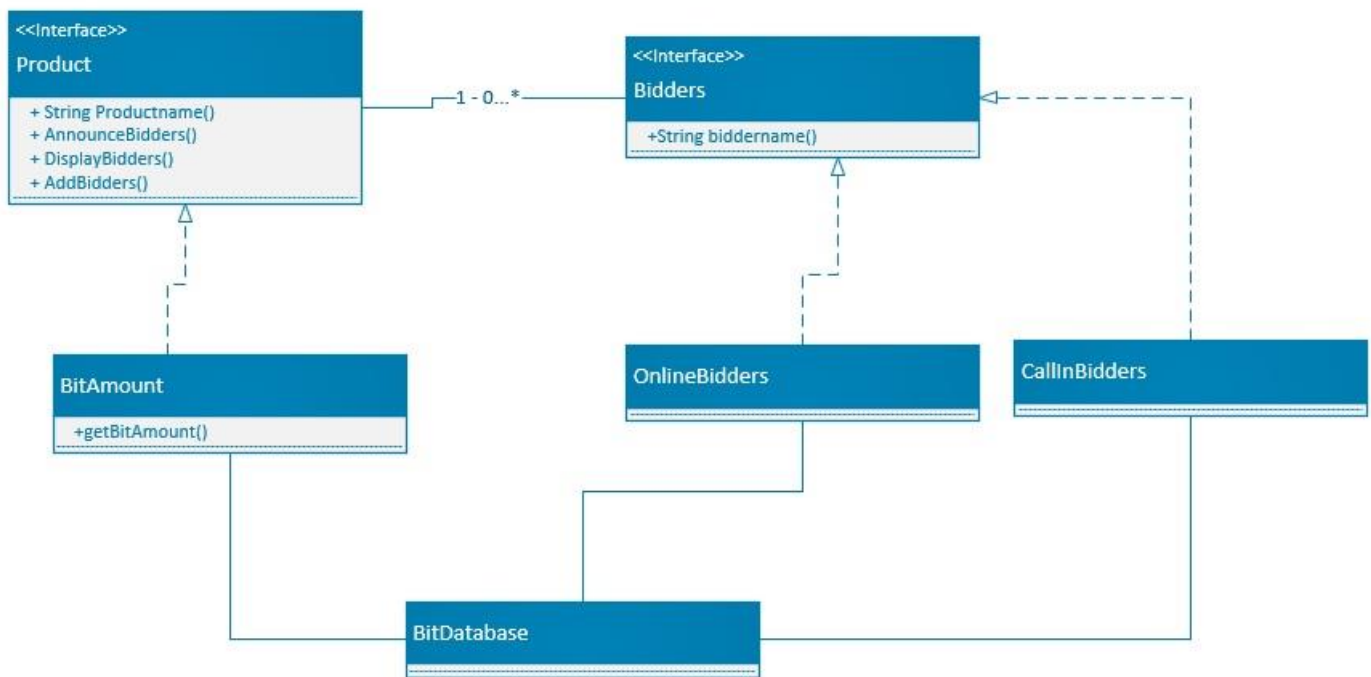
// Define a public static operation (getInstance()) that returns the sole instance of the class TaskManager

```
public static TaskManager getInstance()
{
    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.

