

## Design Patterns Iterator, Composite, Decorator

B.Tech. (IT), Sem-6,  
Applied Design Patterns and Application Frameworks (ADPAF)

Dharmsinh Desai University  
Prof. H B Prajapati

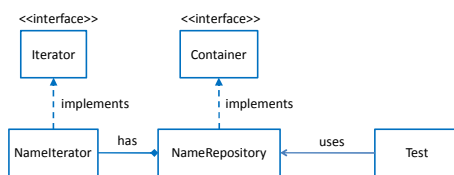
1

## The Iterator design pattern

- This pattern is of type behavioral pattern.
- Different collection types can have different representations and different ways to access elements.
- Iterator pattern allows a common way to access the elements of a collection object in sequential manner without any need to know its underlying representation.

2

## The Iterator design pattern



3

## Example: The Iterator design pattern

- Iterator interface  

```
package designpattern.iterator;
public interface Iterator {
    public boolean hasNext();
    public Object next();
}
```
- Container interface  

```
package designpattern.iterator;
public interface Container {
    public Iterator getIterator();
}
```

4

## Example: The Iterator design pattern

```

package designpattern.iterator;
public class NameRepository implements Container {
    public String names[] = {"Kisan", "Radha", "Ganga", "Narmada"};
    @Override
    public Iterator getIterator() {
        return new NameIterator();
    }
    private class NameIterator implements Iterator {
        int index;
        @Override
        public boolean hasNext() {
            if(index < names.length){
                return true;
            }
            return false;
        }
    }
}
  
```

5

## Example: The Iterator design pattern

```

@Override
public Object next() {
    if(this.hasNext()){
        return names[index++];
    }
    return null;
}
}
  
```

6

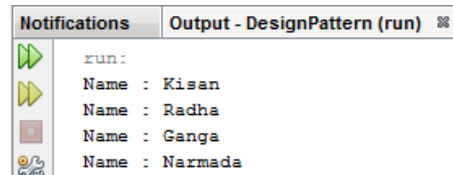
### Example: The Iterator design pattern

```
package designpattern.iterator;

public class Test {
    public static void main(String[] args) {
        NameRepository namesRepository = new NameRepository();
        for(Iterator itrVar = namesRepository.getIterator();
            itrVar.hasNext();){
            String name = (String)itrVar.next();
            System.out.println("Name : " + name);
        }
    }
}
```

7

### Running the Example: The Iterator design pattern



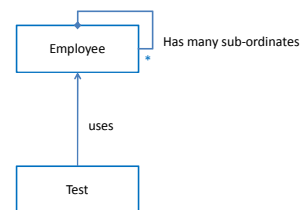
8

### The Composite design pattern

- It is of type structural design pattern.
- The composite design pattern is used where we need to treat a group of objects in a similar way as a single object.
- Composite pattern composes objects in term of a **tree structure** to represent part as well as whole hierarchy.
- This pattern creates a class that contains **a group of its own** objects.
- This class provides ways to modify its group of same objects.

9

### The Composite design pattern



10

### Example: The Composite design pattern

```
package designpattern.composite;
import java.util.ArrayList;
import java.util.List;
public class Employee {
    private String name;
    private String department;
    private List<Employee> subordinates;
    // constructor
    public Employee(String name,String dept) {
        this.name = name;
        this.department = dept;
        subordinates = new ArrayList<Employee>();
    }
}
```

11

### Example: The Composite design pattern

```
public void add(Employee e) {
    subordinates.add(e);
}
public void remove(Employee e) {
    subordinates.remove(e);
}
public List<Employee> getSubordinates(){
    return subordinates;
}
public String toString(){
    return ("Employee :{ Name : "+ name
        +", dept : "+ department + " }");
}
}
```

12

## Example: The Composite design pattern

```
package designpattern.composite;
public class Test {
    public static void main(String[] args) {
        Employee dean = new Employee("Prof. Panchal", "Dean, FoT");
        Employee hodIT = new Employee("Prof. Chhajed", "Head, IT");
        Employee hodEC = new Employee("Prof. Kothari", "Head, EC");
        Employee fac1IT = new Employee("Prof. Prajapati", "AP, IT");
        Employee fac2IT = new Employee("Prof. Dabhi", "AP, IT");
        Employee fac1EC = new Employee("Prof. Thumar", "P, EC");
        Employee fac2EC = new Employee("Prof. Dalal", "AP, EC");

        dean.add(hodIT);
        dean.add(hodEC);
    }
}
```

13

## Example: The Composite design pattern

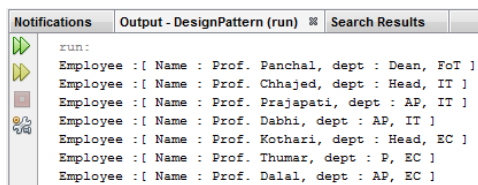
```

        hodIT.add(fac1IT);
        hodIT.add(fac2IT);
        hodEC.add(fac1EC);
        hodEC.add(fac2EC);
        //print all employees of the organization
        System.out.println(dean);
        for (Employee headEmployee : dean.getSubordinates()) {
            System.out.println(headEmployee);
            for (Employee employee : headEmployee.getSubordinates()) {
                System.out.println(employee);
            }
        }
    }
}
}

```

14

## Running Example: The Composite design pattern



15

## The Decorator design pattern

- It is of type structural design pattern.
- Decorator pattern allows a user to **add new functionality** to an existing object without altering its structure.
- This pattern acts as a **wrapper** to existing class.
- This pattern creates a decorator class which **wraps the original class** and provides additional functionality keeping class methods **signature intact**.

16

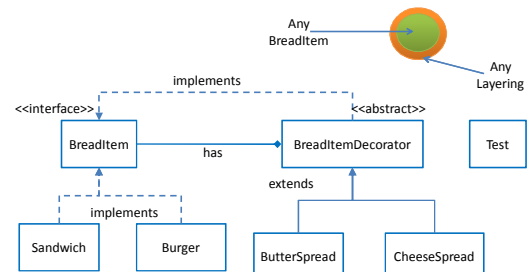
### Example

- Assume a Food shop has the following food items with the indicated cost

Food Item	Price (Rs.)
Sandwich	30
Burger	40
Sandwich with butter spread	35
Burger with butter spread	45
Sandwich with cheese spread	40
Burger with cheese spread	50
Additional layer of butter spread	05
Additional layer of cheese spread	10

17

### Example: The Decorator design pattern



18

### Example: The Decorator design pattern

- We create following:
  - A BreadItem interface and concrete classes implementing the BreadItem interface.
  - An abstract decorator class BreadItemDecorator
    - Implements the BreadItem interface and
    - Has BreadItem object as its instance variable.
  - ButterSpread and CheeseSpread as concrete classes implementing BreadItemDecorator.

19

### Example: The Decorator design pattern

```

• BreadItem
package designpattern.decorator.breadfood;
public interface BreadItem {
    public float cost();
}
• Sandwich
package designpattern.decorator.breadfood;
public class Sandwich implements BreadItem{
    @Override
    public float cost() {
        return 30;
    }
}

```

20

### Example: The Decorator design pattern

```

package designpattern.decorator.breadfood;
public class Burger implements BreadItem{
    @Override
    public float cost() {
        return 40;
    }
}

```

21

### Example: The Decorator design pattern

```

package designpattern.decorator.breadfood;
public class BreadItemDecorator implements BreadItem{
    protected BreadItem breadItem;

    public BreadItemDecorator(BreadItem breadItem) {
        this.breadItem = breadItem;
    }
    @Override
    public float cost() {
        return breadItem.cost();
    }
}

```

22

### Example: The Decorator design pattern

```

package designpattern.decorator.breadfood;

public class ButterSpread extends BreadItemDecorator{
    public ButterSpread(BreadItem breadItem) {
        super(breadItem);
    }
    @Override
    public float cost() {
        return breadItem.cost()+5;
    }
}

```

23

### Example: The Decorator design pattern

```

package designpattern.decorator.breadfood;

public class CheeseSpread extends BreadItemDecorator{
    public CheeseSpread(BreadItem breadItem) {
        super(breadItem);
    }
    @Override
    public float cost() {
        return breadItem.cost()+10;
    }
}

```

24

## Example: The Decorator design pattern

```
package designpattern.decorator.breadfood;
```

```
public class FoodTest {
    public static void main(String[] args) {
        System.out.println("Price of Sandwich = "+new Sandwich().cost());
        System.out.println("Price of Burger = "+new Burger().cost());
        System.out.println("Price of Sandwich with butter spread = "+new
        ButterSpread(new Sandwich()).cost());
    }
}
```






25

## Example: The Decorator design pattern

```
BreadItem burger=new Burger();
BreadItem burgerWithCheeseSpread=new CheeseSpread(burger);
BreadItem burgerWithCheeseAndButterSpread=new
ButterSpread(burgerWithCheeseSpread);
System.out.println("Price of Burger with Cheese and butter spread =
"+burgerWithCheeseAndButterSpread.cost());
}
```

26

## Running the Example: The Decorator design pattern

Notifications	Output - DesignPattern (run)	Search Results	Notif
	run:		
	Price of Sandwich = 30.0		
	Price of Burger = 40.0		
	Price of Sandwich with butter spread = 35.0		
	Price of Burger with Cheese and butter spread = 55.0		
	BUILD SUCCESSFUL (total time: 0 seconds)		

27

## References

- Oracle Certified Professional Java SE 7 Programmer Exams 1Z0-804 and 1Z0-804 (A Comprehensive OCPJP 7 Certification Guide), by S G Ganesh and Tushar Sharma, publisher Apress
- Java Design Patterns, problem solving approaches, tutorials point, [www.tutorialspoint.com](http://www.tutorialspoint.com)

28