OOSE 12/29/2018

Final Exam

Name: 陳佳妏 ID: B10523014

1. How to prevent cloning of a singleton object?

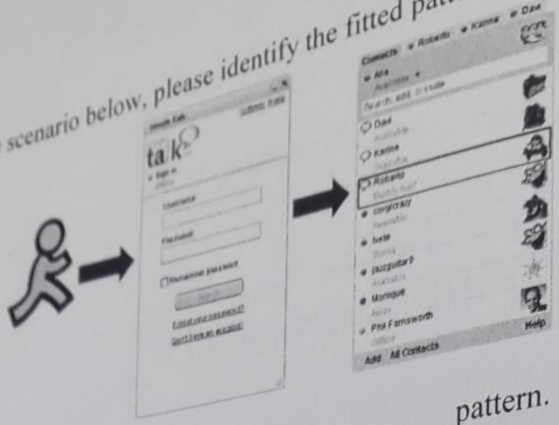
2. We obtained a result as follows. What kind of design pattern is applied to this situation? Describe your answer.

```
IMAGE1[proxy.virtual.ProxyImage@15db9742] calling display Image first time :
  Loading HiResolution_100MB_Dog Photo
  Displaying HiResolution_100MB_Dog Photo

IMAGE1[proxy.virtual.ProxyImage@15db9742] calling display Image second time :
  Displaying HiResolution_100MB_Dog Photo

IMAGE1[proxy.virtual.ProxyImage@15db9742] calling display Image third time :
  Displaying HiResolution_100MB_Dog Photo
```

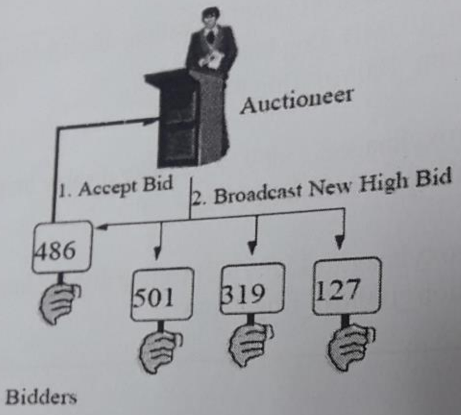
3. Here are some common design pattern applications. Please fill in the appropriate answer.

1) ______________ pattern is used by Runtime, Calendar classes.
2) ______________ pattern is used by based on the current time in the given time zone with the given locale, e.g. public static Calendar getInstance(TimeZone zone, Locale locale).
3) ______________ pattern is used extensively in Java IO, like BufferedReader and BufferedWriter which enhances Reader and Writer objects to perform Buffer level reading and writing for improved performance.
4) ______________ pattern is used to get a way to access the elements of a collection object in sequential manner without any need to know its underlying representation.
5) ______________ pattern is used to rovide an abstract class to be subclassed to create an HTTP servlet suitable for a Web site. A subclass of HttpServlet must override at least one method, e.g. public abstract class HttpServlet extends GenericServlet.

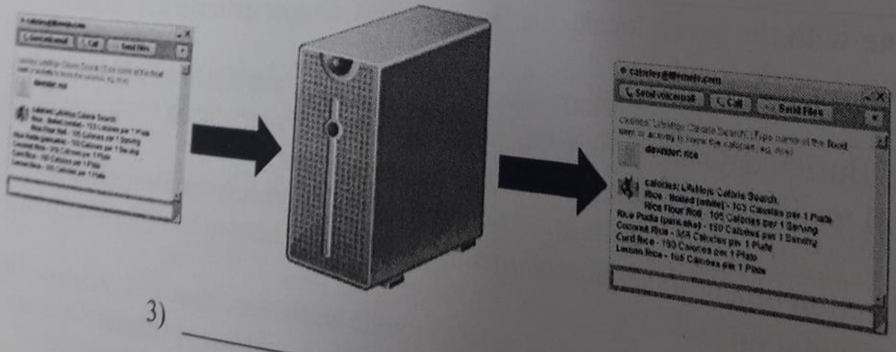
4. Given the scenario below, please identify the fitted pattern.



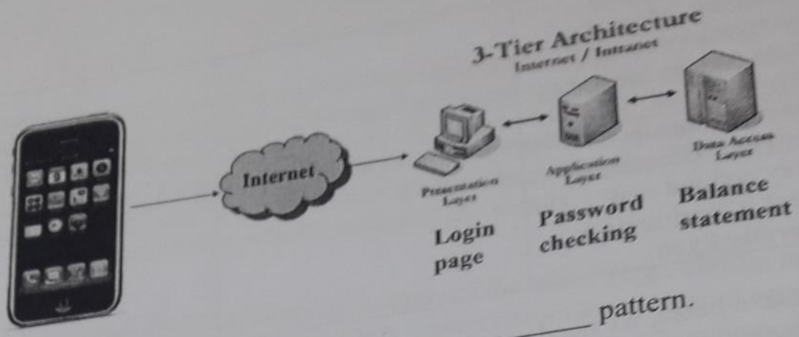
1) ______________________________ pattern.



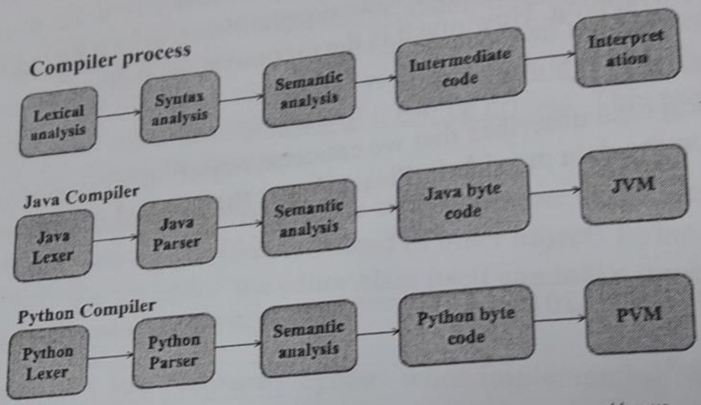
2) ______________________________ pattern.


Send voicemail
Call
Send Files
Calories: Lifehigh Calorie Search
Rice - Boiled (white) - 103 Calories per 1 Plate
Rice Flour Rice - 105 Calories per 1 Serving

3) ______________________________ pattern.



4) ______________________________ pattern.



5) ______________________________ pattern.

5. Please compare Decorator and Proxy patterns based on the following questions and draw their **structure diagrams**.

	Decorator	Proxy
1) What is the type of the pattern?		
2) Define a <u>common or different</u> interface for the decorator / proxy and the target object?		
3) The functionality added by a decorator / proxy to a real object is permanent — it cannot be removed once added.		

int
Caretaker mc = new

4) This is application dependent. More than one type of decorator / proxy can be applied to a given object.		
5) The relationship between a decorator / proxy and the real object it represents is generally fixed.		

6. To make a pizza, we need a dough and other chosen topping. The cost of dough, cheese and ketchup is ~~100~~, 10, 5, respectively. A customer wants to add double layer cheese and a ketchup to the basic dough. Please write **a line of code** to describe the wrapping. How much is the pizza with the topping?

100
10

2↑ cheese
1↑ ketchup

7. The following case describes that we create an object of any one of sub-classes depending on the data provided. Please detect **the applied pattern** and **write its result.**

```
package oose.finalexam2018;

import java.lang.*;

public class Patterns {

    public static void main(String[] args) {

            String obj = new String("a");

            System.out.println("Now you are taking the " + obj.valueOf("OOSE
final exam."));

            Integer wish = 100;
            System.out.println("Integer Value = " + obj.valueOf(wish));
    }
}
```

element

$\boxed{b1}\boxed{b2}$

8. Imagine that you're working on a new text-editor app. Your current task is to create a toolbar with a bunch of buttons for various operations of the editor. While all of these buttons look similar, they're all supposed to do different things. You are going to apply **Command pattern** to solve the problem. Command objects serve as links between various GUI and business logic objects. The GUI object just triggers the command, which handles all the details.

Commands which result in changing the state of the editor (e.g., cutting and pasting) make a backup copy of the editor's state before executing an operation associated with the command. After a command is executed, it's supposed to place into the command history (a stack of command objects) along with the backup copy of the editor's state at that point. Later, if the user needs to revert an operation, the app can take the most recent command from the history, read the associated backup of the editor's state, and restore it. While it isn't that easy to save an application's state because some of it can be private. This problem can be mitigated with the **Memento pattern**.

The Memento pattern delegates creating the state snapshots to the actual owner of that state, the originator object. Hence, instead of other objects trying to copy the editor's state from the "outside," the editor class itself can make the snapshot since it has full access to its own state.

Please use **Command and Memento** together when implementing "**undo**". In this case, commands are responsible for performing various operations over a target object, while mementos save the state of that object just before a command gets executed.