Patterns Post Test

1.Portable Application

If an application is to be portable, it needs to encapsulate platform dependencies. These "platforms" might include: windowing system, operating system, database, etc. Too often, this encapsulation is not engineered in advance, and lots of #ifdef case statements with options for all currently supported platforms begin to procreate like rabbits throughout the code.To Design a application which is portal, which design pattern will you use ?

A) Bridge pattern
B) Builder Pattern
C) Abstract Factory Pattern

2.Ray-tracing

Ray-tracing is a technology for generating photo-realistic images from scene description data. Very advanced types of this technology have recently been used for creating quite a few, and very impressive, animated pictures (such as “Shrek", or “Nemo").

Imagine you have to realize a ray-tracer application. At the core of such an application is a data structure, which maintains and represents the scene graph or the “world". Objects can be very simple objects, which have a geometrically defined shape and a typically algorithmically defined material, but they can also be arbitrarily complex compositions of other objects. The scene graph is then repeatedly traversed to check for intersections with various optical rays. To Design a class structure, which can represent a scene graph which design pattern is applicable?

A) Composite pattern
B) Iterator Pattern
C) Facade Pattern

3.Flight booking

Suppose you should design an application, which enables clients to book the cheapest flight to a destination of their choice from a number of providers. Every provider implements an interface IFlightProvider, which provides operations for querying for a connection, and for booking a flight. In this application you will need a way to enable clients to interface to these providers and book the cheapest flight on offer for the destination and date they are interested in. Flight providers should require (and receive) no knowledge on other flight providers known to the system. Which design pattern could you use?

A) Observer pattern
B) Strategy Pattern
C) Facade Pattern

4. Advertisement

You design an application for an advertisement company, which produces different sort of publications such as books, articles, leaflets, etc. The company publishes those products in many media formats such as printed material, CD, DVD, online websites, etc. How you would model the company products hierarchy (Publications, Media)?

Please choose only one answer:

A. Use composite design pattern to enable the publication and media hierarchies to be treated in the same way as a single instance of an object.

B. Use the adapter design pattern to translates publication interface into a compatible media interface.
C. Use bridge design pattern to build separate hierarchies for the publications and the media, to decouple the abstraction from its implementation so that the two can vary independently

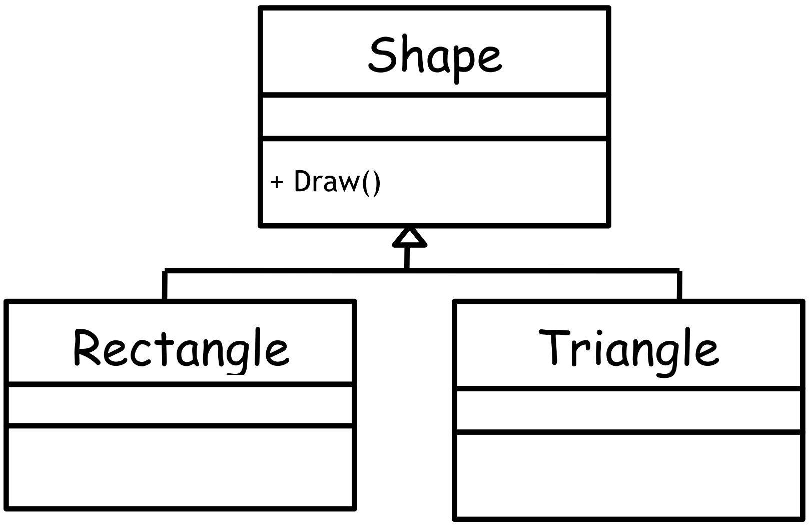
5.**Shell command**

Which design pattern is used by your shell to realize a command like:
`ls -l * | cut -f1,2 | sort`
Observation:
It is totally irrelevant what the three commands do, and you do not even need to know what they are doing.

A) Command pattern
B) Decorator Pattern
C) Intepetor Pattern

6.Shapes

We have fixed number of drawing objects supported in the system. We should be able to add operations like scaling, rotation, etc later to the drawing objects.



A) Proxy pattern
B) Visitor Pattern
C) Decorator Pattern

7.File Transfer

Consider a class with methods that clients can invoke to transfer files to and from some remote server.

```
class FileTransfer
{
    public virtual void Download(string url, byte[] data, int size)
    {
        // download the requested resource
    }

    public virtual void Upload(string url, byte[] data, int size)
    {
        // upload the requested resource
    }
}
```

In addition to being able to upload and download files, the client application would also like to optionally log all file transfer requests, perform access checks for each invocation.

A) Visitor pattern
B) Proxy Pattern
C) Adapter Pattern

8. Library Version

Aggregate types in the first version of the Java programming language such as Vector returned an iterator of type Enumeration.

```
class Vector {
  public Enumeration elements();
}

interface Enumeration {
  public boolean hasMoreElements();
  public Object nextElement()
}
```

Client code wanting to process the elements of any aggregate accepted a parameter of type Enumeration.

```
public void clientCode(Enumeration e) {
  . . .
}
```

Version 1.2 of the language added new aggregate types such as LinkedList and a new type of iteration called Iterator.

```
class LinkedList {
  public Iterator iterator();
}

interface Iterator {
  public boolean hasNext();
  public Object next();
  public void remove();
}
```

Assume you are working with a LinkedList and would like to pass its Iterator to the client code expecting an Enumeration. What design pattern can help in this situation?

A) Abstract Factory pattern
B) Adapter Pattern
C) Proxy Pattern

9. **Dependancy**

You are working on software that interacts with a new hardware device through a device driver and are anxious to test against the actual hardware. The hardware manufacture has a beta version of the device driver they plan to release but is warning the interface of the driver could change between now and the release date. The device driver is used throughout your code and you are concerned about writing code to an interface that is subject to change. What design pattern can be used to mitigate the risks involved?

A) Builder pattern
B) Adapter Pattern
C) Proxy Pattern

10. **Generic event-driven simulator**

Consider a generic event-driven simulator, i.e. a simulator skeleton that may simulate a wide range of systems. The simulator contains a queue of events and a variable indicating the simulated time. Each event has a time stamp that indicates the time at which the event will occur. The queue contains events, which are sorted in the ascending order of the time at which the events occur. The generic event-driven simulator executes the following infinite loop provided by the `simulate` method of class `GenericSimulator`:

```
abstract class GenericSimulator
{
protected EventQueue eventQueue;
protected double simulatedTime;
protected State newState, oldState;
protected Event currentEvent;
public abstract State consequences(State oldState, Event event);
public abstract void actions(State newState, State oldState);

public void simulate()
      {
            while (true)
            {
                  simulatedTime = eventQueue.top().time;
                  do
                  {
                        currentEvent = eventQueue.pop().event;
                        newState = consequences(oldState, currentEvent);
                        actions(newState, oldState);
                        oldState = newState;
                  } while (simulatedTime <= eventQueue.top().time);
            }
      }
}
```

The design of the generic event-driven simulator is based (partly) on a design pattern. Which design pattern?

A) State pattern
B) Strategy Pattern
C) Template Pattern