Name _____

The following might be names of patterns: Abstract Class, Abstract Factory, Adapter, Bridge, Builder, Chain of Responsibility, Collaborator, Command, Command Processor, Decorator, Façade, Factory Method, Flyweight, Interpreter, Mediator, Memento, Observer, Prototype, Proxy, Singleton, Specification, State, Strategy

The purpose of many of the design patterns is to make it easy to change some property of the system. What design pattern would you use to make it easy to change:

1. (3 points) The way a set of objects interact with each other

2. (3 points) The algorithm that an object uses

3. (3 points) The implementation of an abstraction

4. (3 points) The class of the object that a method returns.

5. (3 points) The kind and number of objects that react to changes in the object you are designing.

6. (3 points) The internal state of an object, but you want to be able to remember this state and then later tell the object to go back to that state.

7. (3 points) What design pattern should you think of first when you want to implement undo?

8. (3 points) What design pattern should you think of first when you want to implement one and only one instance of a class?

9. (3 points) What design pattern should you think of when you want to hide how you construct a complex object?

10. (3 points) What design pattern should you think of when when you have a class that you can not modify conveniently?

Suppose you were designing the file system of an operating system.

11. (3 points) You want to reuse the classes that access I/O devices from another project. However, those classes don't have the interface you want. One solution is to refactor them. Another solution is to reuse them without changing them. Which design pattern would help you reuse the classes for the I/O devices in your operating system?

12. (3 points) Your file system has many classes in it, but you want to hide them from the application programmers and present a simpler interface to them. What design pattern lets you hide the complexity of a subsystem like a file system?

13. (3 points) Suppose your system should be able to implement many kinds of file systems. A Linux file system has Linux directories and Linux files, while a BSD file system and an NT file system have their own ways of representing directories and files. You want to be able to write reusable algorithms for the file system, but when these algorithms are creating directories and files, they will create them for Linux, BSD, or NT as appropriate. Which design pattern would help you create objects of the right class?

14. (4 points) List two patterns that reduce tight coupling.

15. (10 points) Decorator and Chain of Responsibility are similar because they both contain multiple nested classes. In other respects, they are quite different. Explain the differences between Decorator and Chain of Responsibility.

16. (5 points) Observer/Strategy. Suppose you are building a simulation of a chemical factory. Your simulation contains a class called Container that represents a container that holds chemicals. Chemicals can flow into it and out of it. It has a method:

```
addVolume(float amount) {
        volume = volume + amount;
        if (volume > maximum) overflow();
        Iterator a = channels.iterator();
        while (a.hasNext())          ((Channel) a.next()).calculateInput();
        Iterator b = displays.iterator();
        while (b.hasNext())          ((ContainerDisplay) b.next()).redisplay(); }
```

There are a variety of "Channel" classes, each of which implements "calculateInput()", and a variety of "ContainerDisplay" classes, each of which implements "redisplay()". You decide that it would be better to use the Observer pattern and make Channels and ContainerDisplays be observers of the Container. Show the new version of addVolume()

17. (5 points) How would the Channel and ContainerDisplay classes have to change?

18. (10 points) Suppose there is a class Foo with the following method

```
Object doSomething(Object x) {
        if (flag == "first")       return doFirst(x);
        if (flag == "second" ) return doSecond(x);
        if (flag == "third" ) return doThird(x);
        throw new Exception("Illegal option");
}
```

This looks like a case statement. This might be OK if flag came from user input, but you discover that it is assigned in several methods and is always one of the string literals you see above. You notice that the doFirst() method calls a private method removeAll() of the foo class and is the only place where removeAll() is called. You notice that all three call a private method setUp. You decide to get rid of the case statement by using the Strategy pattern. Describe in detail what you would do. Show the new implementation of doSomething() for class Foo.

19. (10 points) What is the difference between intrinsic and extrinsic state? Give an example of each.

20. (10 points) The memento pattern states that it does not violate encapsulation but yet accesses an object's internal state. Explain how the memento does not violate encapsulation and discuss your language's support for this feature of the memento.