SE2811 Quiz 8 Name:

This is a twenty-point quiz. Points are worth half as much as points on ten-point quizzes, but the quiz will take longer to complete than a 10 point quiz.

1. (5 pts.) Select the pattern/idiom that best meets the design needs. Please ask clarifying questions as required.
	1. You have a wide variety of vendor-supplied classes that provide similar functionality but have different method names for each class.
		1. Strategy
		2. Factory
		3. Singleton
		4. Observer
		5. Decorator
		6. Composite
		7. Command
	2. You have a really sweet GUI that gives the user all kinds of capabilities to modify different components of the GUI and objects. Select a pattern that will allow you to treat all of the capabilities the same way.
		1. Strategy
		2. Factory
		3. Singleton
		4. Observer
		5. Decorator
		6. Composite
		7. Command
	3. In this really sweet GUI, you’ve already got a way to treat all these capabilities consistently, and now you’d like the user to be able to build their own capabilities from existing capabilities, but treat the user’s capabilities just like the original ones.
		1. Strategy
		2. Factory
		3. Singleton
		4. Observer
		5. Decorator
		6. Composite
		7. Command
	4. You would like to implement a logging system that all parts of your program can access.
		1. Strategy
		2. Factory
		3. Singleton
		4. Observer
		5. Decorator
		6. Composite
		7. Command
	5. The project has “undo” on the requirements list.
		1. Strategy
		2. Factory
		3. Singleton
		4. Observer
		5. Decorator
		6. Composite
		7. Command
2. (2 pts.) Select the design principle best meets the description
	1. Keeping a reference to a client object instead of extending it
		1. Reducing coupling
		2. Increasing cohesion
		3. Favor composition over inheritance
		4. Program to interfaces, not implementations
		5. Closed for extension, open for modification
		6. Increasing transparency
	2. Including methods used by some derived classes in an interface or super-class.
		1. Reducing coupling
		2. Increasing cohesion
		3. Favor composition over inheritance
		4. Program to interfaces, not implementations
		5. Closed for extension, open for modification
		6. Increasing transparency
3. (7 pts.) Suppose you have a class **Cup** with two methods, void fill() and void empty(). Write a **FillCupCommand** class implementing an **Observer** interface that could be used in a larger program. Although you do not need to get the method names perfect, you need to implement all of them.
4. (6 pts.) In pseudo-code, write the implementation of the **Invoker**’s undo method. You should use Java syntax, but you do not need to remember the exact names of Java API methods, just what functionality is required for the problem at hand. You do not need to support redo capability.

(If you need another sheet of paper, please ask. I have some)