Name _____

Answer 9 of the following questions. Indicate which of the questions you wish to be graded. Answer essay questions as briefly as possible.

1. What design pattern should you think of first when you:

   A. Want to implement undo.

   B. Have a design that requires a tree of objects.

   C. Want to reuse an object but it has the wrong interface.

2. The text claims that three design patterns help overcome the inability to alter classes conveniently. List two of the design patterns that help overcome the inability to alter classes conveniently.

3. List one benefit and one liability of the Decorator pattern.

4. Some authors consider the Proxy and Decorator patterns to be similar. In what ways are the two patterns similar?

5. Explain the benefits and disadvantages of using the Null object pattern in implementing a Binary Search tree.

6. Explain one of the following types of coupling: Data Coupling, Control Coupling, Inside Internal Object Coupling.

7. Explain one of the following types of cohesion: Logical, Temporal, Procedural, Communication, Sequential. Give an example.

8. The design patterns text uses two main design principles. Select one pattern that uses both principles. Show how it uses both.

9. You now have implemented three different ways to find all elements of a tree with a given digital sum. Select two of the methods and and contrast the coupling and cohesion of the two methods.

10. Select one of the patterns Visitor or Composite and explain how the pattern works.

11. Implement an External iterator on an array.

12. (Java/C++ Version) You are building a simulation of a steam plant. Your simulation contains a class called Boiler that represents a boiler that is heated and contains water. The water boils creating steam and increases the pressure in the boiler. There are pipes connected to the boiler. The steam flows out into the pipes. The Boiler class has the method below. There are a number of Pipe classes and Views classes.

```
increasePressure( float morePressure) {
        pressure =+ morePressure;
        if (pressure > maximumSafePressure)
                releaseSafetyValve();
        for (int k = 0; k < pipes.length(); k++)
                pipes[k].calculatePressure();
        for (int k = 0; k < views.length(); k++)
                views[k].updateDisplay();
```

A. Modify the increasePressure: method to use the Observer pattern.

B. How would the Pipe and View classes have to change?

12. (Smalltalk Version) You are building a simulation of a steam plant. Your simulation contains a class called Boiler that represents a boiler that is heated and contains water. The water boils creating steam and increases the pressure in the boiler. There are pipes connected to the boiler. The steam flows out into the pipes. The Boiler class has the method below. There are a number of Pipe classes and Views classes.

```
increasePressure: aNumber
        pressure := pressure + aNumber.
        pressure > maximumSafePressure ifTrue: [self releaseSafetyValve].
        pipes do: [:each | each calculatePressure].
        views do: [:each | each updateDisplay].
```

A. Modify the increasePressure: method to use the Observer pattern.

B. How would the Pipe and View classes have to change?

13. (Java/C++ Version)The following methods are from classes B & C that have a common superclass A. Rewrite the methods using the Template method. Show all the methods you would create and which classes they would be in.

```
(Class B) processUsing(DataSource data, ConnectionHandler connection) {
        try {
                byte[ ] piece = data.getPiece( fileId, pieceIndex);
                Message response = new Piece(fileId, pieceIndex, piece);
                connection.sendMessage( response);
        catch (Exception e) { connection.sendMessage( new ErrorMessage);}
}
```

```
(Class C) processUsing(DataSource data, ConnectionHandler connection) {
        try {
                FileData file = data.getFile( filename);
                Message response = new SearchRequest(file);
                connection.sendMessage( response);
        catch (Exception e) { connection.sendMessage( new ErrorMessage);}
}
```

13. (Smalltalk Version)The following methods are from classes B & C that have a common superclass A. Rewrite the methods using the Template method. Show all the methods you would create and which classes they would be in.

```
Class B>>processUsing: aDataSource connection: aConnectionHandler
        [piece := aDataSource atFile: fileId piece: pieceIndex.
        response := Piece file: fileId index: pieceIndex piece: piece.
        aConnectionHandler sendMessage: response]
                on: Error
                do: [:exception | aConnectionHandler sendMessage: ErrorMessage new]

Class C>>processUsing: aDataSource connection: aConnectionHandler
        [files := aDataSource filesNamed: filename.
        response := SearchResult files: files.
        aConnectionHandler sendMessage: response]
                on: Error
                do: [:exception | aConnectionHandler sendMessage: ErrorMessage new]
```