1. Describe the two main important points of SCI?

- Curving back to myself – I create again and again: composite pattern defines the composite to have a type of itself in it's class declaration. During execution each composite calls itself to unfold the details and get the required result from the primitive classes (leafs) which contain the required values and does not have anymore composite object in their definition. Likewise, Transcendental Meditation (TM) is the technique which allows us to go deep in to ourselves and then to the pure field of creative intelligence. It is the finer state and the source of all thoughts. This pure field is unfolded and then gives rise to the different creations in the universe.
- Do less and accomplish more: design patterns provide solutions to recurring problems. When we design our system we use design patterns to easily design the system in such a way that the system will meet object oriented design principles such as extensibility, maintainability, encapsulated, decoupling, cohesive, etc. likewise, by calling the mantra during TM, the mind will transcend easily to the pure field of creative intelligence where there is unbounded source of thoughts and we can get the benefit of it.

2. Define:

a. What is Encapsulation? Give one example of encapsulation of behavior.

- It is a technique used to hide implementations (Details) of an object such as properties and behaviors and allowing access to the object as appropriate.
- Example a Department class can have a behavior to calculate the department salary (CalculateSalary()) the user of the class doesn't know the details of the implementation of the CalculateSalary() behavior. It is encapsulated in the class and the user is only given the total salary of the department. The user can not see the implementation of the behavior.

technique used for hiding the properties and behaviors of an object and allowing outside access only as appropriate

b. What is design pattern? How they defer from frameworks?
Design pattern is a reusable solution to a problem. A recurring solution to a standard problem.
Framework is a complete or semi complete implementation to a domain problem and focuses on a particular domain. Groups of patterns create frameworks.

c. What is inversion of control in OO? What is Variation Oriented Design (VOD)?
Inversion of control in OO: message flow is outward from the framework. i.e. the framework sends messages to our code to invoke as necessary whatever functionality they provide.
Variation Oriented Design: locate where the changing and non-changing parts of the problem domain (system) and design the system in such a way that the non-changing part (Fixed part) is closed for modification and the changing (varying) part is open for modification. (Open closed principle)

d. What are the four main sections of event driven program structure?
   a. Component
   b. Setup / wiring

c. Run / start
d. Event loop

e. Template relies on inheritance. Would it be possible to get the same functionality of Template method using object composition? What would be some of the tradeoffs be?
- Yes it is possible to get the same functionality using object composition. We can get an advantage of being highly decoupled. But the implementation could be more complicated since we are expected to correctly implement the parent – child relationship found in the template pattern.

f. Compare strategy and template pattern. Where is polymorphism used in either pattern?
- They are the same that both are used to vary algorithms. The algorithm is what they encapsulate change. They are different that they encapsulate only the specializations (special operations) of the algorithms (in the case of template pattern) verses encapsulating the whole algorithms each time in the case of strategy pattern.
- Polymorphism is used in template pattern when the user is calling directly the derived class object via the base class object. The user call is fixed and when he calls the special object, the correct concrete class is called polymorphically.
- In Strategy the user is not directly calling the concrete classes. Instead he/she is calling context and then the appropriate strategy is called through the context object. The different strategies are encapsulated as strategy and the appropriate strategy is called polymorphically through the context

g. What is meant by a polymorphic iterator? What is factory method (for iteration)

- Polymorphic iterator: Unified or common interface with multiple concrete implementations. Client can use one interface and can switch polymorphically accessing many different iterators
- Factory method for iterator: method in the collection that returns the appropriate iterator.

h. What is the intent of the composite pattern? Where does self-referral occur in the composite pattern?
- Composite pattern: provides a tree structure which represent part-whole hierarchy. It allows us to access the individual and composite parts uniformly.
- Self referral occurs when the composite class refers back to itself and becomes recursive.

i. How does the command pattern use polymorphism? What was the benefit of using command in a GUI action listener?
- by defining the common interface (P2I) to the command and abstracting to the concrete commands. The user then gets access to the correct concrete command through the command interface polymorphically.
- Using command in GUI allows us to create command manager so that we can control the commands being dispatched from the event dispatcher and perform any operation we need that includes redos and undos on the commands. This is because normally the event dispatcher doesn’t controls the commands being performed.

j. Internal iterator uses a functor to operate on each element. Is this a command pattern?

- No. functors in the internal iterator is used to perform iteration and perform operation in itself (using the doAll method). It does not execute or call methods or operation on another class (Receiver). Whereas in command pattern the command object is expected to execute (call) an operation on another class (in the Receiver class).

3. Design an application to search for an item from a list of items. The application should decide the search algorithm to be used and configure a search manager object (context) with this algorithm. For example, if the list is already sorted, the application should use the binary search algorithm as opposed to the linear search algorithm. Design each algorithm as a different strategy class. Draw the UML diagram and provide java skeleton code to elaborate your design.

```
//Context Class
Public Class SearchManager {
    Private List<Item> items = new List<Item>();
    Private ISearchStrategy strategy;
    Public SearchManager(){
        }

    Public void setSearchStragegy(ISearchStrategy strategy){
            This.strategy = strategy;
    }

    Public void add(Item itm){
            Items.add(itm);
    }

    Public void search(Item itm){
            strategy.search(itm, items);
    }
}

//Strategy interface
Interface ISearchStrategy{
    Public void Search(Item item, List<Item> items);
}

//Concrete Strategy
Public Class BinarySearch implements ISearchStrategy{
    Public BinarySearch(){
        }

        Public void Search(Item item, List<Item> items){
            //binary search code goes here
        }
}

//Concrete Strategy
Public Class LinearSearch implements ISearchStrategy{
    Public LinearSearch(){
        }

        Public void Search(Item item, List<Item> items){
            // linear Search code goes here
        }
}

Public Class Main {
    Public static void main(String[] args){
            SearchManager manager = new SearchManager();
            Manager.add(itm1);
            Manager.add(itm2);
            Manager.add(itm3);

            Manager.setSearchStrategy(new BinarySearch());
            Manager.Search(itm1);

            Manager.setSearchStrategy(new LinearSearch());
            Manager.search(itm2);
        }
}
```

4. Extend the standard java vector class to create a new myvector class which also has an internal iterator. Name the internal iterator method doAll(), and it should take a single argument which is a functor that encapsulates the processing for each element in the loop. Use a standard functor interface, which includes two methods:

a. Functor<T> compute(T element)
b. Functor<T> getValue(T result)

Implement a sum functor which computes the sum of a collection of integers.

```
Public Class MyVector<T> extends Vector<T>{
    Public void doAll(IFunctor f){
            For(T t : this)
                    f.compute(t);
        }
}

Interface IFunctor<T> {
    Public void compute(T element);
    Public T getValue();
}
```

```
Public Class Sum implements IFunctor<Integer>{

    Private Integer total = new integer(0);

    Public void compute (Integer element){
            total+=element;
        }

        Public Integer getValue(){
            Return total;
        }
}

Public Class Main {
    Public static void main(String[] args){
            Private MyVector<Integer> collection = new MyVector<Integer>();
            collection.add(1);
            collection.add(2);

            IFunctor<Integer> sum = new Sum();

            collection.doAll(sum);
            system.out.println(sum.getValue());
        }
}
```