OOSE Final Exam 01/04/2018

1. Please indicate the design pattern of the following description. **10%**

- Attach additional responsibilities to an object dynamically. The pattern provides a flexible alternative to subclassing for extending functionality.
- Define a family of algorithms, encapsulate each one, and make them interchangeable. The pattern lets the algorithm vary independently from clients that use it.
- Provide a centralized communication medium between different objects in a system.
- Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.
- Provide an interface for creating families of related or dependent objects without specifying their concrete classes.
- Represent an operation to be performed on the elements of an object structure. This pattern lets you define a new operation without changing the classes of the elements on which it operates.
- Separate the construction of a complex object from its representation so that the same construction process can create different representations
- Use sharing to support large numbers of fine-grained objects efficiently.
- Without violating encapsulation, capture and externalize an object's internal state so that the object can be restored to this state later.
- You want to decouple event handlers from an event sources so handlers can be added or removed without impacting the event source.

2. Discuss the similarities and differences of Adapter and Proxy patterns and draw their structure diagrams. **20%**

3. Due to the heavy load of Edit Controller, we would like to separate the database access responsibilities from the controller and assign them to DBMgr. In this design, the controller is responsible for handling actor requests while the DBMgr handles database access. In this way, the DBMgr hides the database access from the business objects. Please write the code to explain the way DBMgr will use to change the DB access dynamically. **20%** The diagram is shown in Fig.1. Skeleton code is provided for your reference as Fig. 2.

4. The template method pattern can be applied to improve the database access commands in the persistence framework. Fig.2 shows how to accomplish this. Based on the given diagram, please specify which shall be concrete operation, primitive operation, factory method and hook operation? Why do you think so? **10%** Please implement the template method in the example in Java code. **20%**

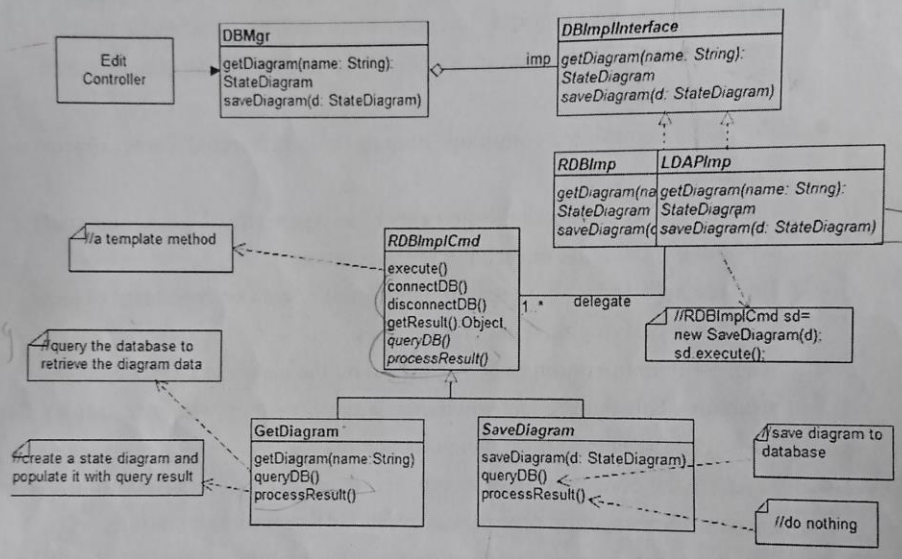

Fig. 1

```
public interface ImplementationInterface {
  public void operation1();
  public void operation2();
}

public class ConcreteImplementationK
    implements ImplementationInterface {    // K=1, 2, ...
  public ConcreteImplementationK() { ... }
  public void operation1() {
    /* implement Concrete Implementation K
       specific behavior for operation 1*/
  }
  public void operation2() {
    /* implement Concrete Implementation K
       specific behavior for operation 2*/
  }
}
```

```
public class ClientInterface {
  ImplementationInterface imp
    =new ConcreteImplementationK();
  public void operation1() {
      imp.operation1();      // behavior is imp dependent
  }
  public void operation2() {
      imp.operation2();      // behavior is imp dependent
  }
  /* other operations including operations
     to change the imp to refer to a different
     concrete implementation */
}

public class Client {
  public void use() {
    ClientInterface cif=new ClientInterface();
    cif.operation1();
    cif.operation2();
  }
}
```

Fig. 2

5. Based on the following figure, please implement the Factory Method pattern to create the appropriate Iterator requested by the caller in Java code. The concrete Iterator will traverse the Composite-based expression tree and access each of its elements one at a time. 20%

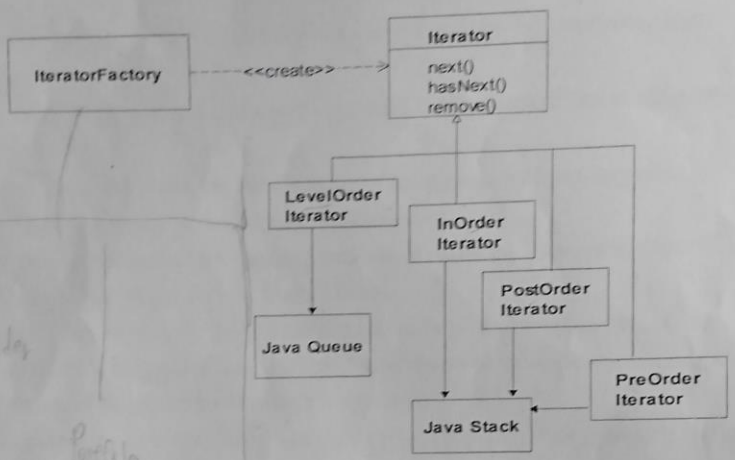

Fig. 3