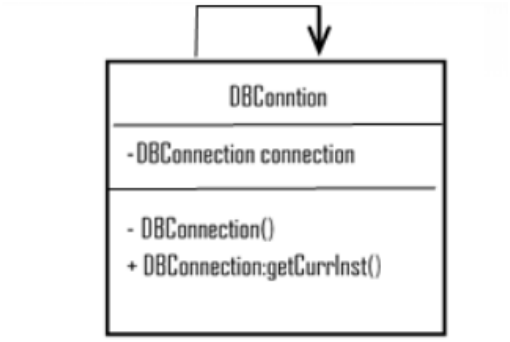
Questions **2**
Example for Singleton 2
Example for factory method 2
Example for the decorator pattern 4
Example of the proxy pattern 5
Example for State pattern 6
Example for the Strategy design pattern. 7

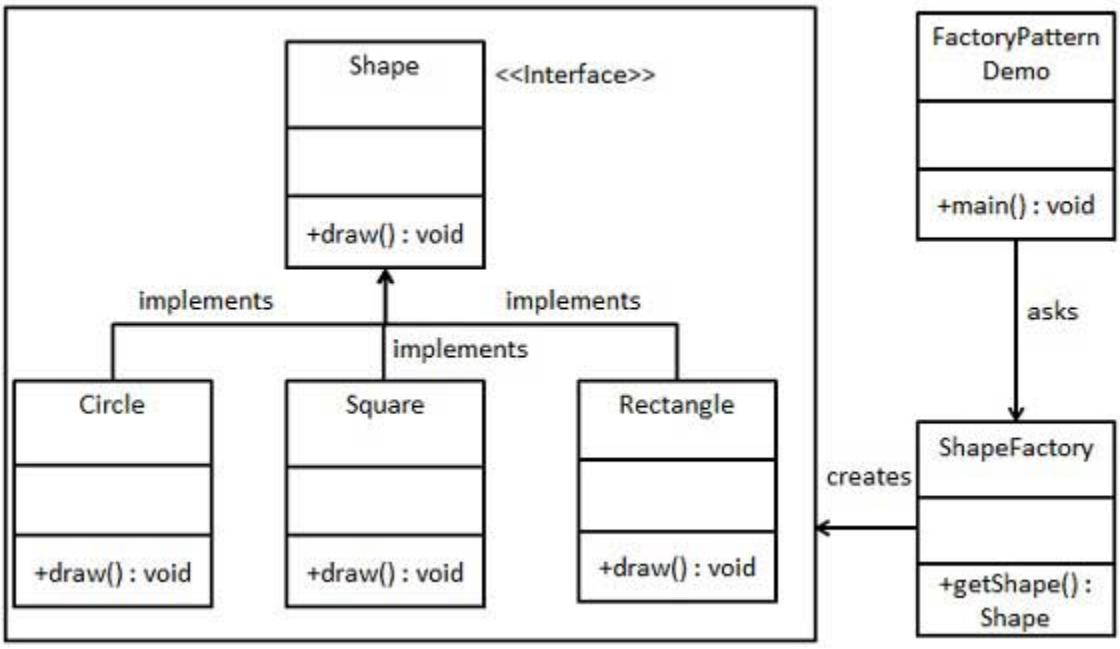
Questions

Q1. Example for Singleton

To Establish a Database connection to your application, use a **singleton** design pattern. Define a singleton class, create two instances using the Singleton class, and write a simple condition to determine whether or not that object is the same singleton instance.



Q2. a. Example of the factory method



You have to apply a factory method to this question.
You have to write the body section of each child's class.
Once the user requests a circle, display the “drawing a circle” message in the console.
If it is square, display the message as “drawing square” or “drawing a rectangle”; otherwise, a rectangle.

Example for Factory Method

You are given a class called Person. The person has two fields: ID and Name.

Implement a *non-static* Person_Factory with a Create_Person method that takes a person's name.

The person's Id should be set as a 0-based index of the object created. So, the first person the factory makes should have Id=0, the second Id=1, and so on.

```
public class Person
 {
   public int Id { get; set; }
   public string Name { get; set; }

 }
```

Example for the decorator pattern

You must use the decorator pattern to add additional functionality to the application that draws various shapes.
You will need to create a component interface, decorator interface, concrete components, and concrete decorators.
create a **Shape** interface(component) and add a simple **draw method**.
implement concrete classes and the Shape interface (concrete components)

- **Rectangle** and **Circle**
- Implement draw() by just displaying some default string.

You are required to create an abstract decorator class named **ShapeDecorator**

- Implementing the Shape interface
- It has a Shape object as its interface variable

You are required to create some concrete decorator classes

- **RedShapeDecorator**, **GreenShapeDecorator**
- Add additional functionalities such as setting a red or green border or changing styling.

Finally, create a class to include the primary method that uses your decorator to decorate shape objects.

Draw a relevant class diagram and then start the implementation.

Example of the proxy pattern

Internet access restriction application

The proxy will first check the host you are connecting to.
It connects to the real internet if it's not part of the restricted site list.

Step 1: Create a subject class that contains a method to connect to the internet
Takes a string representing the hostname
It throws an exception if the hostname is restricted

Step 2: Create the real subject class which actually connects to the internet.
Can just hard code some output that says “connected to the internet…”

Step 3: Create the proxy class
Has reference to the actual subject class
Contains a list of restricted sites (to search)

If the host you are trying to connect to is a banned site, throw an exception.
Otherwise, connect to the internet.

Step 4: Create the client
Uses the proxy to try to connect to legitimate sites

Example for State pattern

We want to simulate the alert states of a mobile phone; the phone can be in either a vibration state or a silent state.
Based on this alert state, the behavior of the mobile service changes when an alert happens.

Step 1: Create a **state interface** containing an alert method
Step 2: create the **context class**

Should have a reference to the state interface
It should have a setState method and an alert method. (alert will use the state reference to cal the instance method)

Step 3: Create two concrete state classes

One for vibration and one for silent(each alert method can simply display a default msg)

Step4: Create a client class that creates a context object and sets its various states while issuing alerts.

Example for the Strategy design pattern.

A text editor can have different textformatters to format text.

We can create different text formatters and then pass the required one to the text editor so the editor can format the text as needed.

The text editor will reference a common interface called **TextFormatter**, which has one abstract method, “txtFormat (String txt),” and the **TextEeditor’s** job will be to pass the text to the formatter to format the text.

Formatting options are **Uppercase** and **Lowercase**.

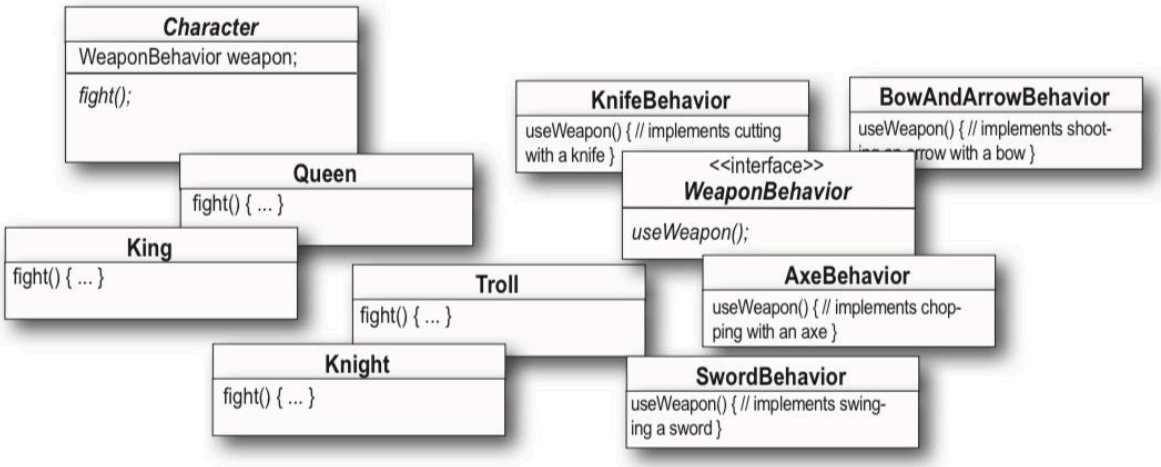
So you have to identify the concreteStrategy classes, Context, and Strategy interface and define the method inside them.

Draw a suitable class diagram and follow the implementation accordingly.

When the user inputs a “Testing text in caps formatter” the Output should be as follows.

```
[CapTextFormatter]: TESTING TEXT IN CAPS FORMATTER
[LowerTextFormatter]: testing text in lower formatter
```

Example for stratergy pattern




```
setWeapon(WeaponBehavior w) {
    this.weapon = w;
}
```