Question 1
You have an Animal class that provides Feed() as an abstact:

```
publicabstractclass Animal
{
  publicabstractvoid Feed();

}



publicclass Dog : Animal
{

  publicoverridevoid Feed()
  {
  }

}

publicclass Rattlesnake : Animal
{

  publicoverridevoid Feed()
  {
  }

}
```

But then you realise that you have a need for some of the animals to be treated as pets and have them groomed. You may be tempted to do:

```
publicabstractclass Animal
   {

     publicabstractvoid Feed();

     publicabstractvoid Groom();

   }
```

which would be fine for the Dog, but it may not be fine for the Rattlesnake So, according to DIP what is the best implementation ?

Don't put the Groom (); inside the abstract class Animal. If there are only one kind of animal needs groom, the method Groom(); can be just put inside the class of Dog. If there are a plenty of animals need groom, an interface that contains Groom(); can be created. Make the animal classes that need groom inherit the interface.

Question 2
Though the method we talked about in activity sounds great and observes SRP but by using that, we should not only discompose the Class Animal into two new classes but also modify the Class Client. That’s a huge modification. Better idea? Hint: We can conform to the SRP in method level instead of being tangled in details

I think the answers of the last question can also answer this question.

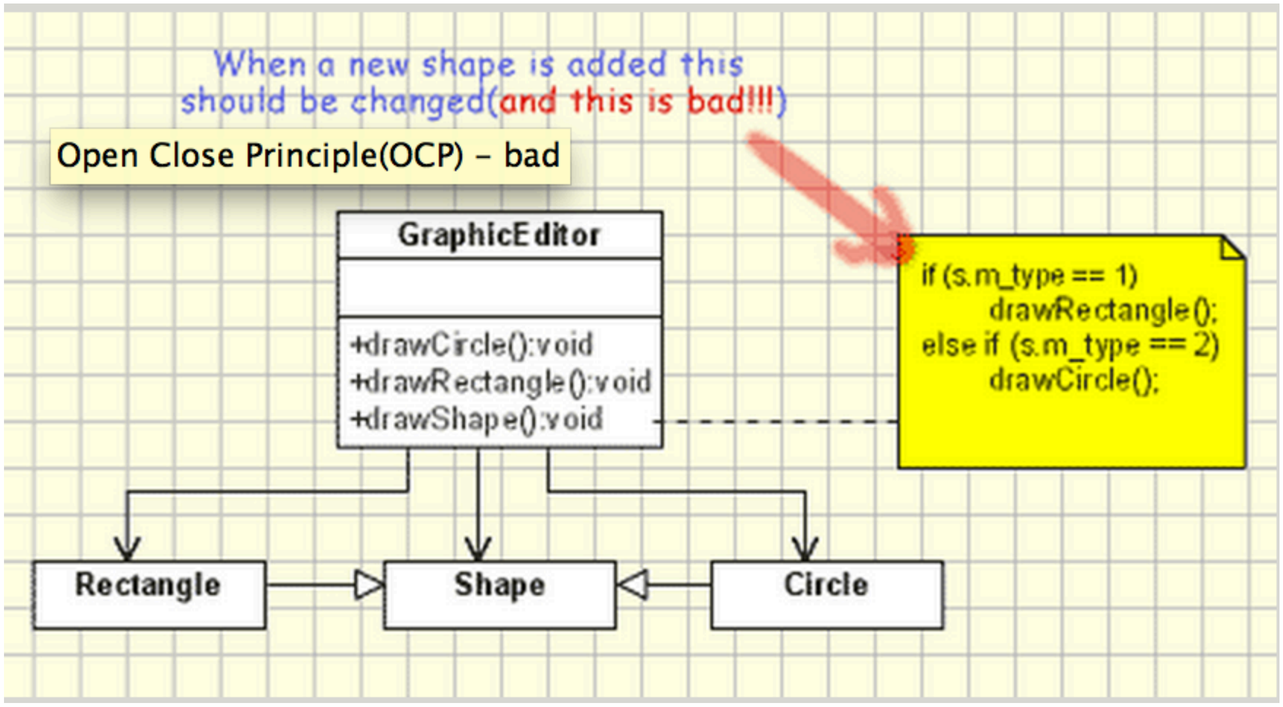
Question 3
Describe an example that violates OCP. This can either be a real world example or one you thought of yourself.

Bellow is an example which violates the Open Close Principle. It implements a graphic editor which handles the drawing of different shapes. It's obviously that it does not follow the Open Close Principle since the GraphicEditor class has to be modified for every new shape class that has to be added. There are several disadvantages:

- for each new shape added the unit testing of the GraphicEditor should be redone.

- when a new type of shape is added the time for adding it will be high since the developer who add it should understand the logic of the GraphicEditor.

- adding a new shape might affect the existing functionality in an undesired way, even if the new shape works perfectly


When a new shape is added this should be changed(and this is bad!!!)
Open Close Principle(OCP) - bad
GraphicEditor
+drawCircle():void
+drawRectangle():void
+drawShape():void
Rectangle
Shape
Circle

```
if (s.m_type == 1)
     drawRectangle();
else if (s.m_type == 2)
     drawCircle();
```

Question 4
Which of the following are true of the Liskov Substitution Principle? (Choose all that apply)

- **Subtype must be substitutable for their base types.**
- **No matter where base types appear, subtype can appear. That's to say, base types can appear wherever subtype appears.**
- **The DBC (design by contract) is specified by declaring preconditions and post conditions for each method.**
- **Subclass substitute for super class without errors or exceptions. Clients have no need to know which class they are using.**

True: The DBC (design by contract) is specified by declaring preconditions and post conditions for each method.