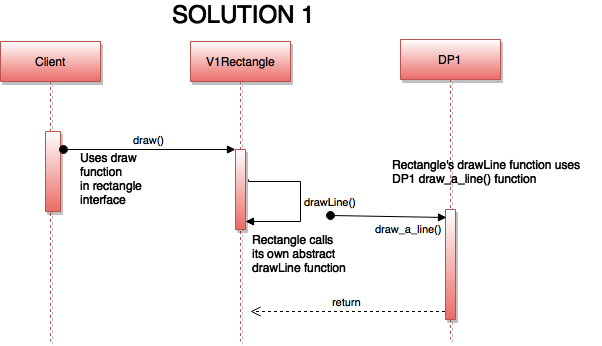
**ALPEREN İNAL o524116028**

**HW 5 Bridge Pattern**

**Solution 1 Sequence Diagram**

****

For each kind of Shape, I have to implement the shape with each drawing program, deriving a version of DP1 and a version of DP2 for Rectangle and deriving a version of DP1 and a version of DP2 one for Circle.

Unfortunately, this approach introduces new problems. If I get another drawing program, I will have six different kinds of **Shape**s

If I then get another type of **Shape** , another variation in concept, I will have nine different types of **Shape**s

The class explosion problem arises because in this solution the abstraction (the kinds of **Shape** s) and the implementation (the drawing programs) **are tightly coupled**. *Each type of shape must know what type of drawing program it is using*. I need a way to separate the variations in abstraction from the variations of implementation so that the number of classes only grows linearly.

This is exactly the intent of the Bridge pattern: “to decouple an abstraction from its implementation so that the two can vary independently.”

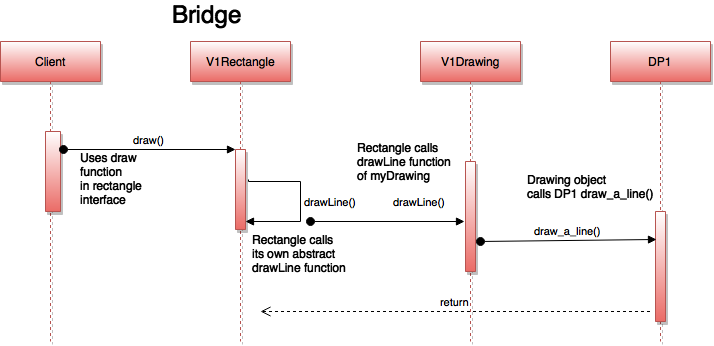
**Solution 2**

I still have the same four classes representing all of my possible combinations. However, by first deriving versions for the different drawing programs, I eliminated the redundancy between the DP1 and DP2 classes.

Unfortunately, I am unable to eliminate the redundancy between the two types of Rectangles and the two types of Circles, each pair of which has the same draw method.

In any event, the class explosion that was present before is still present here.

**Bridge Pattern Solution Sequence Diagram**



The Bridge pattern let me see that viewing the implementation as something outside of my objects, something that is used by the objects, gives me much greater freedom by hiding the variations in implementation from my calling program. By designing my objects this way, I also noticed how I was containing variations in separate class hierarchies. The hierarchy on the left side contains the variations in my abstractions. The hierarchy on the right side contains the variations in how I will implement those abstractions.

