**Bridge Pattern vs Strategy Pattern**

1. Specify the deciding factors that determine which of these two patterns should be applied in a situation, emphasizing the reasons why the Bridge pattern is considered to be structural while the Strategy pattern is considered to be behavioral.

One factor is depending on the structure of the business logic and the goal of the designer is wanted. For example, in strategy pattern, we can have a clear variation between different algorithms implemented to be used inside one unit of business logic(Context).

But, if we have different algorithms implemented but also we need to use all of them over different contexts or multiple units of business logic, so Bridge pattern is the best option

* + **Example for Strategy Pattern:**

Suppose we must implement a game of fighters, each fighter can do different movements and different kind of punch and kicks.

So, we can create different behaviors such as “regular punch”, “special punch”, “regular kick”, “special kick”, each of those behaviors should implement a common interface like “punchBehavior” and “kickBehavior” respectively.

Now, when we create a Fighter object, we can set a behavior but at any moment in runtime, we can change his behavior.

* + **Example for Bridge Pattern:**

We need to create a process to manufacture vehicles. We start creating an interface “Vehicle”, who receive as parameters different manufacture process.

So, we create an interface “ManufactureProcess” that define each one of the process necessary to build a vehicle or anything else. Thus, we create concrete process implementing the “ManufactureProcess” interface. In example, We can create “ProduceCar” and “AssembleCar”.

Finally, we can create different type of cars implementing the “Vehicle” class. In example, Car or Bike and in the constructor, we can pass instance of different process of manufacture.

I.E.:

Vehicle vehicle1 = new Car(new Produce(), new Assemble());  
vehicle1.manufacture();  
Vehicle vehicle2 = new Bike(new Produce(), new Assemble());  
vehicle2.manufacture();