**Scenario 2.1 (Builder Design Pattern)**

**Definition**

The construction of a complex object should be separated from its representation in order to achieve multiple representations within the same process.

**Where We Require Builder Pattern?**

Let’s discuss the scenario in our application. In our application the primary entity is Customer, let’s say. Ideally and practically as well, once a customer object is fully created, we will not want to change its state.

We need to make immutable Customer Class as we have multiple instances of customers to be created.

Customer Class have ample attributes to get created but its not mandatory that all of them should be passed while creating the customer some of the fields are optional. We need to create multiple customers with different scenarios with variant attributes to satisfy variant customer creation scenarios with non-mandatory fields.

In normal practice, if we want to make an immutable Customer class, then we must pass all information as parameters to the constructor. But later if one or more attributes are added we don’t want to change whole current implementation we need more constructors

The builder pattern will help you to consume additional attributes while retaining the immutability of the Customer class.

Text

Description automatically generated

**Implementing Builder Pattern**

Below solution delineates an additional class CustomerBuilder which helps us in building desired Customer instance with all mandatory attributes and a combination of optional attributes, without losing the immutability.

Text

Description automatically generated

above-created Customer object **does not have any setter method**, so its state cannot be changed once it has been built. This provides the desired immutability.