Scenario 2.2 (Factory Design Pattern)

Definition

The **factory method pattern** is a creational pattern that uses factory methods to deal with the problem of creating objects without having to specify the exact class of the object that will be created.

This is done by creating objects by calling a factory method—either specified in an interface and implemented by child classes, or implemented in a base class and optionally overridden by derived classes—rather than by calling a constructor.

Where We Require Factory Pattern?

The factory design pattern is used **when we have a superclass with multiple sub-classes and based on input, we need to return one of the sub-class**.

This pattern takes out the responsibility of the instantiation of a class from the client program to the factory class.

In our case, we have a superclass named Event and sub-classes named BookingTheEvent and HostingTheEvent.

Step 1: Create an interface event

```
public interface Event {
    void organise();
}
```

Step 2: Create concrete classes implementing the same interface.

```
public class BookingTheEvent implements Event {

    @Override
    public void organise() {

    }
}
```

```
public class HostingTheEvent implements Event {

    @Override
    public void organise() {

    }
}
```

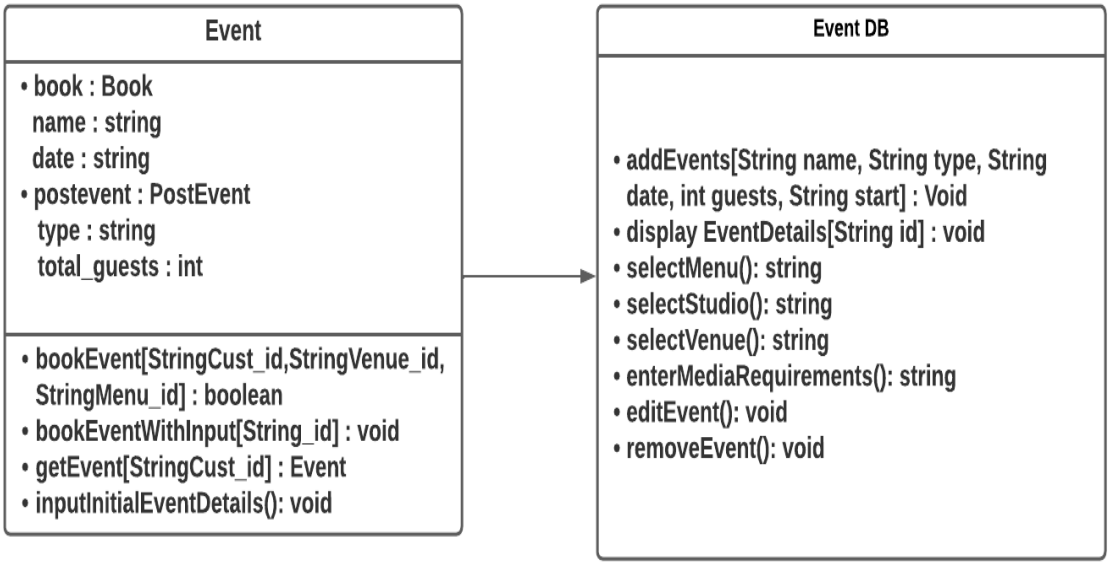
Step 3: Create a Factory to generate object of concrete class based on given information.

```
public class EventFactory {

    //use getEvent method to get object of type event
    public Event getEvent(String eventType){
        if(eventType == null){
            return null;
        }
        if(eventType.equalsIgnoreCase("BookingTheEvent")){
            return new BookingTheEvent();

        } else if(eventType.equalsIgnoreCase("HostingTheEvent")){
            return new HostingTheEvent();

        }

        return null;
    }
}
```

Step 4: Use the Factory to get object of concrete class by passing an information such as type.

```
public class FactoryPatternDemo {

   public static void main(String[] args) {
      EventFactory eventFactory = new EventFactory();

      //get an object of BookingTheEvent and call its organise method.
      Event event1 = eventFactory.getEvent("BookingTheEvent");

      //call organise method of BookingTheEvent
      event1.organise();

      //get an object of HostingTheEvent and call its organise method.
      Event event2 = eventFactory.getEvent("HostingTheEvent");

      //call organise method of HostingTheEvent
      event2.organise();
   }
}
```

Class diagram:



Sequence diagram: