Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

DEPARTMENT OF INFORMATION TECHNOLOGY

COURSE CODE: DJ19ITC802 **DATE:** 15-02-2024

COURSE NAME: Design Patterns Laboratory **CLASS:** BE - IT

EXPERIMENT NO. 2

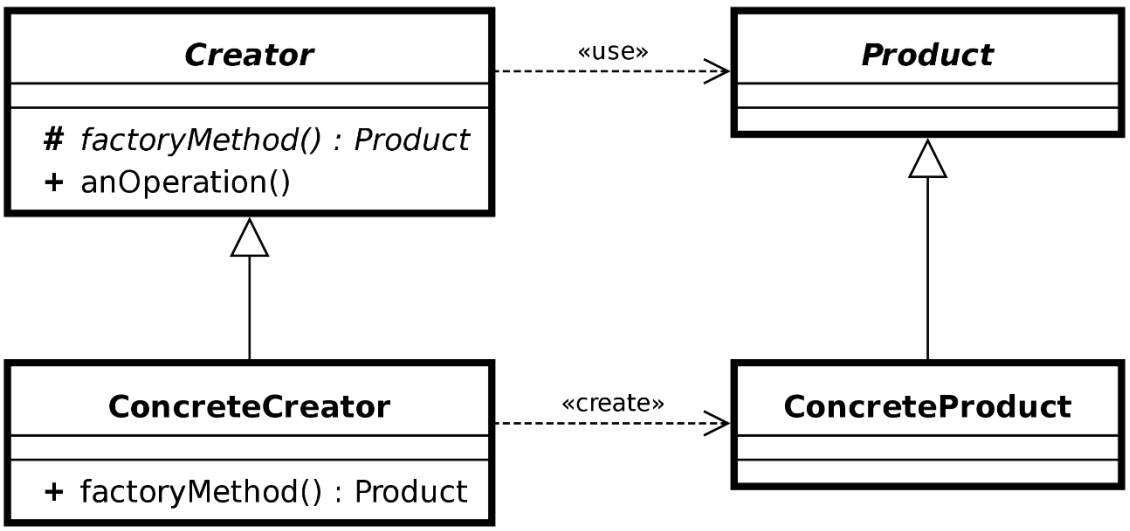
CO/LO: Identify and apply the most suitable design pattern to address a given application design problem.

AIM: Design a UML class diagram for Factory Pattern and implement the same for any real life scenario.

DESCRIPTION:

The factory method is a creational design pattern, i.e., related to object creation. In the Factory pattern, we create objects without exposing the creation logic to the client and the client uses the same common interface to create a new type of object. The idea is to use a static member-function (static factory method) that creates & returns instances, hiding the details of class modules from the user. A factory pattern is one of the core design principles to create an object, allowing clients to create objects of a library in a way such that it doesn't have a tight coupling with the class hierarchy of the library.

SOURCE CODE:



UML Diagram for Factory Pattern

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Implementation of the Factory Pattern for a real-life scenario. Let's say we have a car dealership that sells different types of cars. We can use the Factory Pattern to create instances of different car models.

```
from abc import ABC, abstractmethod
class Car(ABC):
    @abstractmethod
    def drive(self):
        pass
class Sedan(Car):
    def drive(self):
        print("Driving a sedan")
class SUV(Car):
    def drive(self):
        print("Driving an SUV")
class CarFactory:
    @staticmethod
    def create_car(car_type):
        if car_type == "sedan":
            return Sedan()
        elif car_type == "suv":
            return SUV()
# Client code
sedan = CarFactory.create_car("sedan")
suv = CarFactory.create_car("suv")
sedan.drive()
suv.drive()
```

OUTPUT

```
Driving a sedan
Driving an SUV
```

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

CONCLUSION:

We have a Car abstract class that defines the drive() method. We also have two concrete implementations of Car: Sedan and SUV. We then have a CarFactory class that creates instances of Sedan and SUV based on a car type string. Finally, in the client code, we use the CarFactory to create instances of Sedan and SUV and call their drive() methods.

REFERENCES:

[1] https://www.geeksforgeeks.org/factory-method-for-designing-pattern/