Factory Pattern

Factory pattern is a creational design pattern it provides an interface for creating objects in a super class but allows sub classes to alter the type of objects that will be created.

Goal is to hide the creation information from the client code.

```
#include <iostream>

enum CarType{
    FORD,
    SWIFT,
    RENAULT,
    TATA
};
class Car {
 public:
   virtual void drive()=0;
  static std::unique_ptr<Car>CreateCar(CarType);
};

class Ford : public Car {
  public:
    void drive() {
     cout<<”Ford car”<<endl;
};

class Swift : public Car {
   public:
     void drive() {
      cout<< “Swift car”<<endl;
};

class Renault : public Car {
   public:
       void drive() {
        cout<< “Renault car”<<endl;
};

class Tata : public Car {
  public:
    void drive() {
    cout<<”Tata Car”<<endl;
};
```

```
unique_ptr<Car>Car::CreateCar(CarType type) {

  std::unique_ptr<Car>m_car;
     if(type == FORD) {
         m_car = std::make_unique<Ford>();
   }
    else if(type == TATA) {
       m_car = std::make_unique<Tata>();
   }
    else if(type == RENAULT) {
     m_car = std::make_unique<Renault>();
   }
   else {
     m_car = std::make_unique<Swift>();
   }
   return m_car;
}

class CarOrder {
private:
  std::unique_ptr<Car> m_car;
 public:
    explicit CarOrder(CarType type){
       m_car = Car::createCar(type);
   }
    std::unique_ptr<Car> getCar() {
      return std::move(m_car);
   }
};

int main () {
 CarOrder m_carOrder{TATA};
  std::unique_ptr<Car>m_car = m_carOrder.getCar();
m_car→drive();
return 0;
}
```

source links: https://cengizhanvarli.medium.com/factory-and-abstract-factor-design-pattern-for-c-c3552a3fa828