Abstract Factory Pattern

Abstract pattern is a creational design pattern it provides interface for creating families of related or dependent object without specifying their concrete classes.

Difference between Abstract factory and factory pattern…?

The difference between abstract factory and factory pattern is usage of interfaces.

In the factory pattern we are creating an interface for all similar classes. But in abstract pattern creating an interface for same type of classes.

For example we create interface for all types of vehicles But in abstract factory pattern create seperate interface for cars, aircrafts, trucks...etc.

It is more flexible than factory type. Do not need write so many if/else blocks as factory as much in this type.

```
 #include <iostream>
#include <memory>

using namespace std;

class Vehicle
{
    public:
           virtual void startEngine() = 0;
           virtual void getReady() =0;
};
 class AirCraft : public Vehicle
{
   public:
        void startEngine()
         {
            cout<<"airCraft startEngine"<<endl;
         }
         void getReady()
         {
            cout<<"airCraft getReady"<<endl;
          }
};
```

```
class Drone : public Vehicle
{
    public:
        void startEngine()
         {
            cout<<”drone startEngine”<<endl;
         }
         void getReady()
         {
            cout<<”drone getReady”<<endl;
          }
};
class Car : public Vehicle
{
    public:
        void startEngine()
         {
            cout<<”Car startEngine”<<endl;
         }
         void getReady()
         {
            cout<<”car getReady”<<endl;
          }
};
class Truck : public Vehicle
{
    public:
        void startEngine()
         {
            cout<<”Car startEngine”<<endl;
         }
         void getReady()
         {
            cout<<”car getReady”<<endl;
          }
};

class Factory
{
        public:
            virtual unique_ptr<Vehicle>createSmallVehicle()=0;
             virtual unique_ptr<Vehicle>createBigVehicle()=0;
};
```

```
class FlyingTypeVehicle : public Factory
{
       public:
            unique_ptr<Vehicle>createSmallVehicle()
            {
                  return std::make_unique< Drone>();
            }

            unique_ptr<Vehicle>createBigVehicle()
            {
                 return std::make_unique< AirCraft>();
            }
};
class GroundTypeVehicle : public Factory
{
        public:
             unique_ptr<Vehicle>createSmallVehicle()
            {
                  return std::make_unique< Car>();
            }
             unique_ptr<Vehicle>createBigVehicle()
            {
                 return std::make_unique< Truck>();
            }
};

int main() {
   unique_ptr<Factory> m_factory = make_unique< FlyingTypeVehicle>();
      m_factory-> createSmallVehicle()-> startEngine();
      m_factory->  createBigVehicle()-> getReady();

unique_ptr<Factory> g_factory = make_unique< GroundTypeVehicle>();
       g_factory-> createSmallVehicle()-> startEngine();
      g_factory->  createBigVehicle()-> getReady();

return 0;

}
```

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