Abstract factory

There is a pattern called 'abstract factory' which is a fairly complex way to create sets of objects.

It's based on using more than one implementation of the different creational patterns.

What is abstract factory?

Abstract factory is a creational design pattern that produces sets of related objects.

In its simplest form, it could be a set of factories that allows you to produce parts of object set.

This is a solution for situations when you need to create a group of related objects which may be modified to some extent.

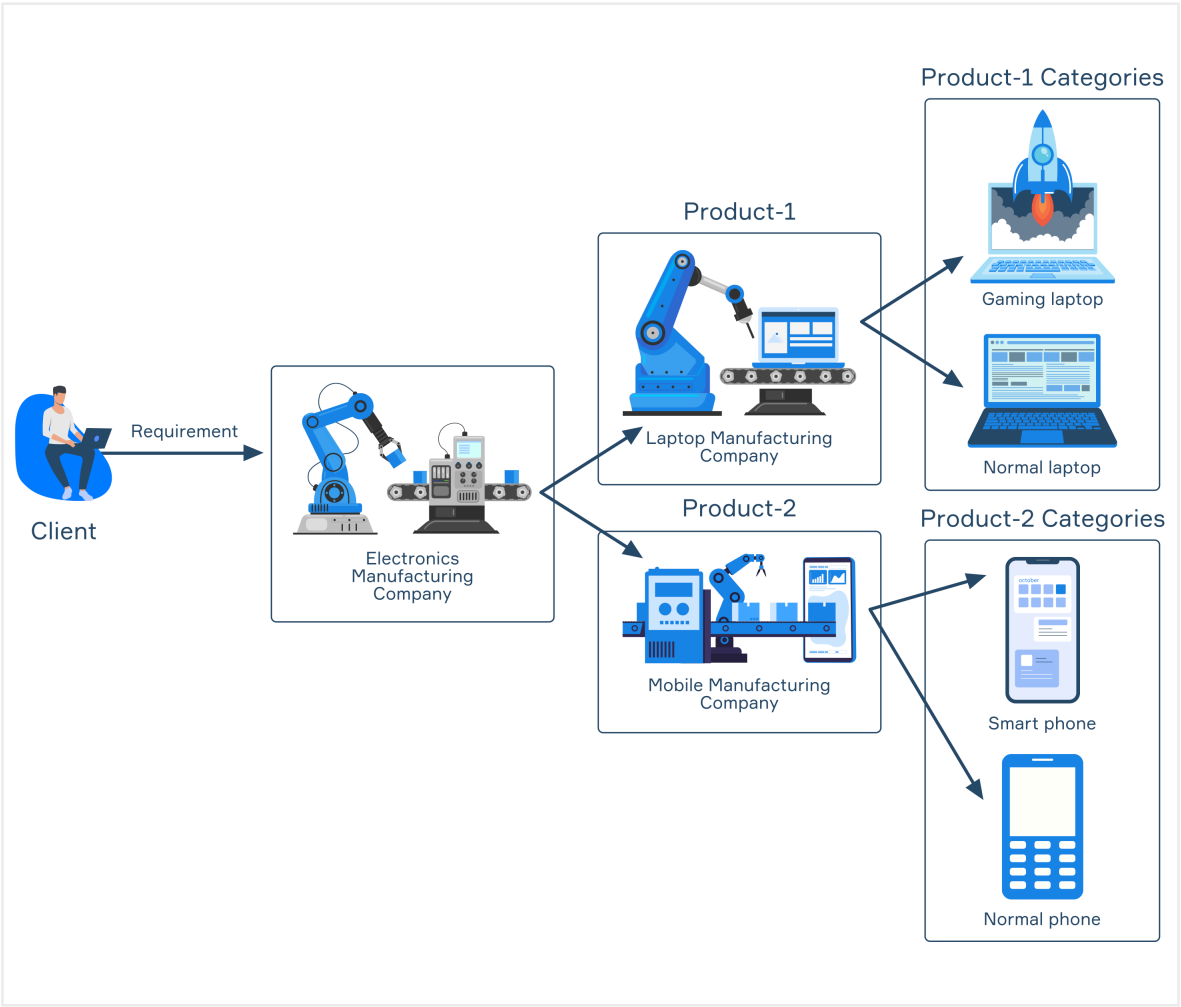
It's a fairly complicated process that could increase the complexity of your code. Also, this pattern does not violate SRP and dedicates a single problem to a single class.

The abstract factory works through distinct interfaces for each object. These interfaces allow you to make different types of objects to form a needed variation of object set.

It's useful to implement an abstract factory when you have a set of factory methods in your code. This pattern is often based on ***factory methods***, but you can also use ***a prototype*** pattern for object creation.

Abstract factory example

Let's imagine for example, that there is some electronics manufacturing company, that produces two types of products: laptops and phones. These products can be produced by their separate factories, but they will still be connected to the main factory that produces them all.



Our abstract factory in this case is made of two factories. Product factory for laptops and product factory for phones.
As it should be when the factory method is used, we can individually modify objects that are made by our factory.
So we have a few variations of laptops and phones which can create different types of objects. They have separate object creation methods, but still make up a part of our company.

Abstract factory in form of pseudocode

Now let's try to recreate our electronics company example in the form of pseudocode. First, we will define our main interface. It will look something like this:

```
interface ElectronicsCompany is
  method createDevice(): Device
  method createMobDevice(): MobDevice
```

This interface declares methods that will return different products. Here we have methods that define creation for two sets of products: Laptops and Phones. Now we will define our factories:

```
class LaptopFactory implements ElectronicsCompany is
  method createDevice(): Device is
```

```
    return new Laptop()

  method createMobDevice(): MobDevice is
    return new GamingLaptop()

class PhoneFactory implements ElectronicsCompany is
  method createDevice(): Device is
    return new Phone()

  method createMobDevice(): MobDevice is
    return new SmartPhone()
```

We have two variants of each product, so we have two different methods for their creation in both of our factories. Methods that are responsible for object creation will be defined separately:

```
interface Device is
  method create()

class Laptop implements Device is...

class Phone implements Device is...

interface MobDevice is
  method create()

class GamingLaptop implements MobDevice is...

class SmartPhone implements MobDevice is...
```

Calling abstract factory methods

As an example of abstract factory implementation into our code, we will create a Company class that will initiate our factories:

```
class Company is
  constructor Company(factory: ElectronicsCompany) is
    this.factory = factory

  method makeDevice() is
    return factory.createDevice()

  method makeMobDevice() is
    return factory.createMobDevice()
```

In here we will initiate our factories for needed variants of our products. Now we will just define conditions for our code to run in the form of CompanyManager:

```
class CompanyManager is
```

```
method main() is
  if (config == "Phone") then
    factory = new PhoneFactory()
  else if (config == "Laptop") then
    factory = new LaptopFactory()

  Company com = new Company(factory)
```

Through some form of config file, we will get a type of factory we need to run and construct our Company. This is a basic way to use abstract factories. As you can see, it can be really flexible and it allows you to easily increase the number of objects and objects variations.