**Абстрактна фабрика** е създаващ шаблон за дизайн, който се използва в обектно-ориентираното програмиране.

**Фабриката** е средство за създаване на обекти. Целта на този шаблон за дизайн е да изолира създаването на обектите от тяхното използване.

Абстрактната фабрика енкапсулира група от методи Фабрика имащи близко предназначение. Клиентският код създава конкретна имплементация на абстрактната фабрика, след това използва основния интерфейс за да създава конкренти обекти. Клиентът не е задължен да знае коя от тези фабрики е създала конкретния обект, защото той използва само основния интерфейс към създадените обекти.

Този шаблон позволява замяната на конкретни класове, дори по време на изпълнение, без да е нужна промяна на кода, който ги използва.

Като цяло това е много полезен шаблон, защото позволява бърза замяна на класове от едно място.

Пример на С# за този патерн може да се види: <http://www.codeproject.com/Articles/328373/Understanding-and-Implementing-Abstract-Factory-Pa>

**Creating the Abstract Products**

In our case, we need two abstract products ISmart and IDumb.

http://www.codeproject.com/images/minus.gifCollapse | [Copy Code](http://www.codeproject.com/Articles/328373/Understanding-and-Implementing-Abstract-Factory-Pa)

interface IDumb

{

string Name();

}

interface ISmart

{

string Name();

}

**Creating the Concrete Products**

Now let us go ahead and create some concrete products for IDumb:

http://www.codeproject.com/images/minus.gifCollapse | [Copy Code](http://www.codeproject.com/Articles/328373/Understanding-and-Implementing-Abstract-Factory-Pa)

class Asha : IDumb

{

public string Name()

{

return "Asha";

}

}

class Primo : IDumb

{

public string Name()

{

return "Guru";

}

}

class Genie : IDumb

{

public string Name()

{

return "Genie";

}

}

Let's do the same for ISmart:

http://www.codeproject.com/images/minus.gifCollapse | [Copy Code](http://www.codeproject.com/Articles/328373/Understanding-and-Implementing-Abstract-Factory-Pa)

class Lumia : ISmart

{

public string Name()

{

return "Lumia";

}

}

class GalaxyS2 : ISmart

{

public string Name()

{

return "GalaxyS2";

}

}

class Titan : ISmart

{

public string Name()

{

return "Titan";

}

}

So we have all the concrete classes ready for all the Dumb Phones and smart phones irrespective of their manufacturers.

**Creating the Abstract Factory**

Now the way we associate these Concrete products with their manufacturers is using the Concrete factories. But before having the concrete factories, we need to have an Abstract Factory.

http://www.codeproject.com/images/minus.gifCollapse | [Copy Code](http://www.codeproject.com/Articles/328373/Understanding-and-Implementing-Abstract-Factory-Pa)

interface IPhoneFactory //'I' stands for interface no relation with Iphone

{

ISmart GetSmart();

IDumb GetDumb();

}

**Creating the Concrete Factories**

Now we can create our Concrete Factories for each manufacturer:

http://www.codeproject.com/images/minus.gifCollapse | [Copy Code](http://www.codeproject.com/Articles/328373/Understanding-and-Implementing-Abstract-Factory-Pa)

class SamsungFactory : IPhoneFactory

{

public ISmart GetSmart()

{

return new GalaxyS2();

}

public IDumb GetDumb()

{

return new Primo();

}

}

class HTCFactory : IPhoneFactory

{

public ISmart GetSmart()

{

return new Titan();

}

public IDumb GetDumb()

{

return new Genie();

}

}

class NokiaFactory : IPhoneFactory

{

public ISmart GetSmart()

{

return new Lumia();

}

public IDumb GetDumb()

{

return new Asha();

}

}

**Creating the Client**

Now we have all the Abstract product classes ready, all the Concrete Product classes ready. Our Abstract Factory is ready and all the Concrete Factories are ready. Now we can write client that will use this hierarchy of related products to create the products.

http://www.codeproject.com/images/minus.gifCollapse | [Copy Code](http://www.codeproject.com/Articles/328373/Understanding-and-Implementing-Abstract-Factory-Pa)

enum MANUFACTURERS

{

SAMSUNG,

HTC,

NOKIA

}

class PhoneTypeChecker

{

ISmart sam;

IDumb htc;

IPhoneFactory factory;

MANUFACTURERS manu;

public PhoneTypeChecker(MANUFACTURERS m)

{

manu = m;

}

public void CheckProducts()

{

switch (manu)

{

case MANUFACTURERS.SAMSUNG:

factory = new SamsungFactory();

break;

case MANUFACTURERS.HTC:

factory = new HTCFactory();

break;

case MANUFACTURERS.NOKIA:

factory = new NokiaFactory();

break;

}

Console.WriteLine(manu.ToString() + ":\nSmart Phone: " +

factory.GetSmart().Name() + "\nDumb Phone: " + factory.GetDumb().Name());

}

}

static void Main(string[] args)

{

PhoneTypeChecker checker = new PhoneTypeChecker(MANUFACTURERS.SAMSUNG);

checker.CheckProducts();

Console.ReadLine();

checker = new PhoneTypeChecker(MANUFACTURERS.HTC);

checker.CheckProducts();

Console.ReadLine();

checker = new PhoneTypeChecker(MANUFACTURERS.NOKIA);

checker.CheckProducts();

Console.Read();

}