From No Factory to Factory Method

Marla **Sukesh**

23 Nov 2012CPOL

This article will tell about the very known design pattern "Factory Pattern" used in the programming.

Factory Method pattern - a relief from New Keyword

Introduction

When we hear the word Design Pattern, the first question that comes to our mind is, “What are Design Patterns?”
Design Patterns are reusable and documented solutions for recurring problems in software development.
Gang of four (GOF) have categorized Design Patterns into 3 categories

- Creational
- Structural
- Behavioral

When I started to learn Design Pattern I found Factory method as a very intriguing pattern. It is one of the most controversial and confused Creational Pattern. There is a alot of confusion in People to understanding this pattern .
First of all let me make all of you clear there is nothing like Factory Pattern (as per GOF design pattern list), GOF has something called Factory Method Pattern. (We will come across with both the terminology later in this stage)

In this article for discussion and explanation purpose I am introducing to you, two fictional characters Alexander (a .NET developer) and Thomas (CEO of an IT institute

called Programmer24X7).

Story begins on the day when Thomas contacts Alexander and asks him to take up role of architect in his in-house project called “Course Management System”. We will see how Alexander progresses his knowledge about Factory Method Pattern day by day and how finally he come up with a great solution.

Before you starts reading this article, a small advice from my side, keep some snacks ready besides you, because it’s going to be a weary 6 day story. Basically I am trying to play with your patience. don’t worry just kidding. It’s gonna be fun, you will enjoy reading this article and towards the end of this article you can proudly say “I know Factory Method Pattern”.

So relax and let’s start our journey to learn Factory Method Pattern

Index

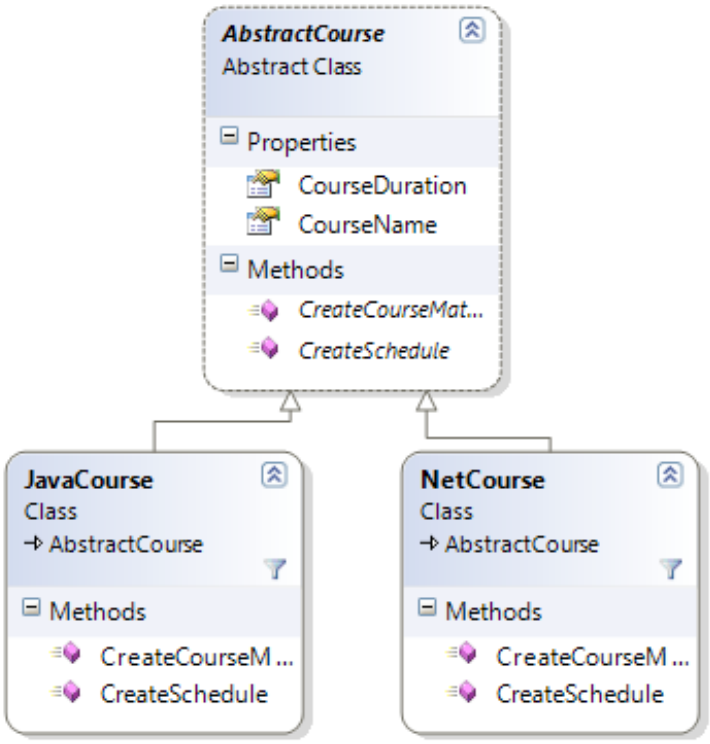
Day 1

Alexander (our .NET developer) collects requirements from Thomas (our Programmer24X7 CEO). One of his primary requirements was to build a module which will manage online courses offered in the Programmer24X7 (technical institute).

How Alexander started?

I) The system Alexander is going to develop is Course Management System, so the first logical step was to start creating Course Library.

II) Next step will be creating instance of courses and building the UI – User Interface. Following is the most obvious code every programmer thinks about and even Alexander did the same.



Approach 1 – Without Factory Approach

```
AbstractCourse objCourse = null;
switch(StrUserInput)
{
   case "N":
   objCourse = new NetCourse();
   break;
   case "J":
   objCourse = new JavaCourse();
   break;
}
objCourse.CreateCourseMaterial();
objCourse.CreateSchedule();
//Display objCourse.CourseName and objCourse.CourseDuration
```

Final step – demonstration

Since module was ready, Alexander demonstrated the same to Thomas, and Thomas felt great about the architecture. Alexander's efforts were well appreciated but, it all didn't end there. He then took the requirement to next logical level where he shared his further plan with regards to the application. He said,

- He is planning to add new online courses like C++, VC++ etc. and would like to stop few courses in future.
- In future In Programmer24X7 offline courses are going to be conducted, so make sure to make everything reusable.

Thomas requested Alexander to make changes as per this advance scope. Well it came to Alexander as a Big Surprise

It was like out of the frying pan but into the fire situation for Alexander.

"Adding a new course means", open the existing UI Code existing logic (add some more switch cases and in future remove some existing case.)

Consider a situation of builder, who is asked to add one more floor between 1st and 2nd floor after the building construction is over taking care that building doesn't collapse.

Problem with the Approach 1

- **SOLID principle OCP** (Open Closed Principle) will be violated when Course library modifies. (OCP says, software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification)
- Reusing the same (Course creation) logic in another UI is not possible.

What Alexander learnt?

Alexander realized an important thing about software development and learnt a great lesson "Changes are integral part of development and system should developed in such a way that it can be easily adapted to such changes without modifying existing **tested** sections."

Isn't it funny, change the system but don't change the code. Well let's see how we can do that.

Move to Index

Day 2

How Alexander Proceeded?

Alexander woke up early morning next day. Coffee was in his hand but mind was still thinking about the problem in hand. It was very difficult for him to find the right star from dark sky but finally he managed to find the Moon and there came Approach 2.

- Sky – My Brain
- Stars – Thoughts
- Moon – Solution for Course Library problem

Approach 2 – Simple Factory Approach

In the first approach UI was creating Course Objects.

Now how it will be if the power of object creation is taken away from UI and give it to someone else. Here someone else is called Simple Factory class.

What is Simple Factory?

Simple Factory is a class with a public static method which will actually do the object creation task according to the input it gets.

```
public class CourseFactory
{
   public static AbstractCourse CreateCourse(string ScheduleType)
   {
      AbstractCourse objCourse = null;
      switch(ScheduleType)
      {
         case "N":
         objCourse = new NetCourse();
         break;
         case "J":
         objCourse = new JavaCourse();
         break;
         //Add more case conditions here for VC++ and C++
      }
      objCourse.CreateCourseMaterial();
      objCourse.CreateSchedule();
      return objCourse;
   }
}
```

UI

```
AbstractCourse objCourse = CourseFactory.CreateCourse(StrUserInput);
//Display objCourse.CourseName and objCourse.CourseDuration
```

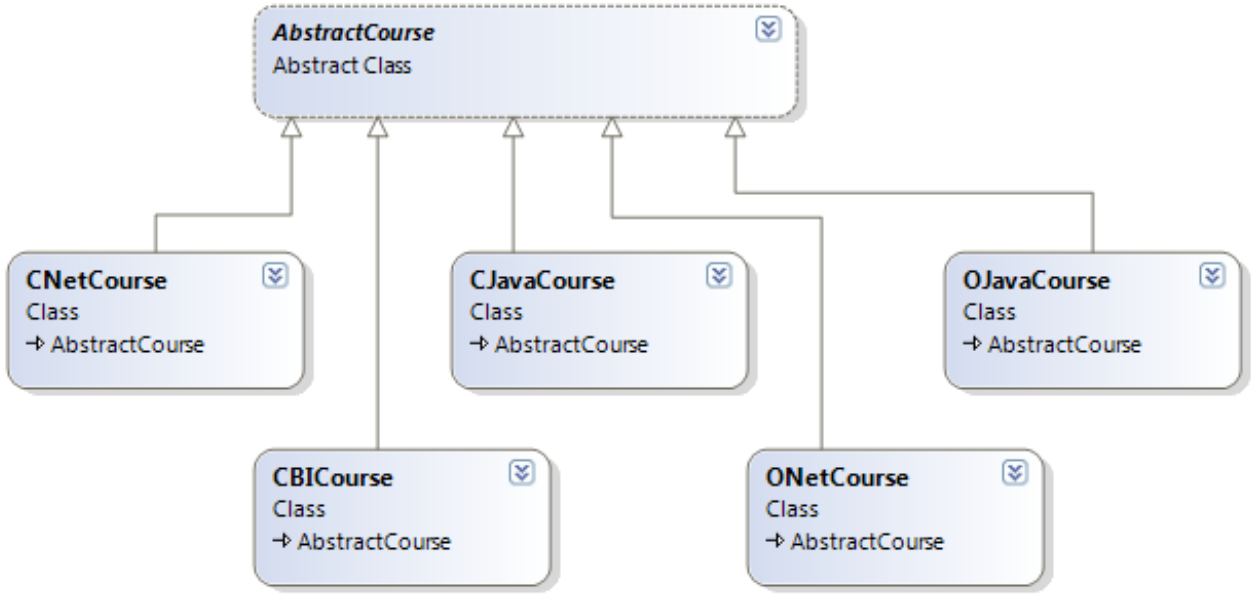
Advantages of the Approach 2

- Whenever new course will be introduced, the one which gets changed is factory not client code.
- As Factory is class any one can use it who have access. In short Course logic now can be reused.

Final step – demonstration

Later in the evening Alexander reached Programmer24X7 office to demonstrate this revised architecture. But before Alexander started to express his anxiety, there came another surprising request. Thomas said, “I think, it’s better if we add one more UI for corporate schedule as well. Reason is that, Now-a-days my corporate training count has increased a lot, so it’s becoming impossible for me to manage.”

I) Now the very first thing we should do is, changing course library (because we need to add corporate courses as well).



II) Add some more case conditions to our Simple Factory class.

```
public class CourseFactory
{
   public static AbstractCourse CreateCourse(string ScheduleType)
   {
      AbstractCourse objCourse = null;
      switch(ScheduleType)
      {
         case "CN":
         objCourse = new CNetCourse();
         break;
         case "CJ":
         objCourse = new CJavaCourse();
         break;
         case "CB":
         objCourse = new CBICourse();
         break;
         case "OJ":
         objCourse = new OJavaCourse();
         break;
         case "ON":
         objCourse = new ONetCourse();
         break;
      }
      objCourse.CreateCourseMaterial();
      objCourse.CreateSchedule();
      return objCourse;
   }
}
```

Problem with the Approach 2

New requirements are easily covered using Simple Factory Approach, then what's the matter?

All though all classes are derived from Abstract Course, adding five case conditions in a single Simple Factory violates **SOLID Principle SRP** – Single Responsibility Principle. (SRP says a class should have only one reason to change.)
And here Factory class will be changed

- Whenever new corporate course introduces or modified.
- Whenever new online course introduces or modified.

So this Simple Factory solution will not workout in current problem context.

Move to Index

Day 3

Let's list down all our requirements.

- We need Factory for online Courses
- We need Factory for corporate Courses
- We don't want to create single Factory for both modules.

How if use Multiple Simple Factories.

1. OnlineCourseFactory
2. CorporateCourseFactory

Approach 3 – Multiple Simple Factory Approach

In this approach we will have multiple simple factories each having a static method which will create an instance based on input it gets.

```
public class CorporateCourseFactory
{
   public static AbstractCourse CreateCourse(string ScheduleType)
   {
      AbstractCourse objCourse = null;
      switch(ScheduleType)
      {
         case "N":
         objCourse = new CNetCourse();
         break;
         case "J":
         objCourse = new CJavaCourse();
         break;
         case "B":
         objCourse = new CBICourse();
         break;
      }
      objCourse.CreateCourseMaterial();
      objCourse.CreateSchedule();
      return objCourse;
   }
}
public class OnlineCourseFactory
{
   public static AbstractCourse CreateCourse(string ScheduleType)
   {
      AbstractCourse objCourse = null;
      switch(ScheduleType)
      {
         case "N":
         objCourse = new ONetCourse();
         break;
         case "J":
         objCourse = new OJavaCourse();
         break;
      }
      objCourse.CreateCourseMaterial();
      objCourse.CreateSchedule();
      return objCourse;
   }
}
```

Online Course UI

```
AbstractCourse objCourse = OnlineCourseFactory.CreateCourse(StrUserInput);
//Display objCourse.CourseName and objCourse.CourseDuration
```

Corporate Course UI

```
AbstractCourse objCourse = CorporateCourseFactory.CreateCourse(StrUserInput);
//Display objCourse.CourseName and objCourse.CourseDuration
```

Everything settled problem solved.

Move to Index

Day 4

Next day Alexander reached Programmer24X7 and explained the architecture to Thomas What is Thomas's reply? Good or Bad?

Small transcript of the big discussion we had after looking into the architecture.

Thomas: Sorry Alexander but I don't think this solution will work.

Alexander: Why Sir?

Thomas: See, it's great that there exist a separate Factory for every Course group. (OnlineCourseFactory, CoporateCourseFactory).
But how will you keep a control over each and every factory. I meant how you will make sure that every factory is creating objects properly, for instance how will you be sure that each factory is following a company standard and creates course materials and schedule prior to returning the object.
It may possible that in future one more kind of factory say OfflineCourseFactory is added for managing offline courses and unknowingly it just violates the company standard and creates course materials in his own way.

Alexander: You are right Sir. Let me think about a full proof and final solution, Give me a day.

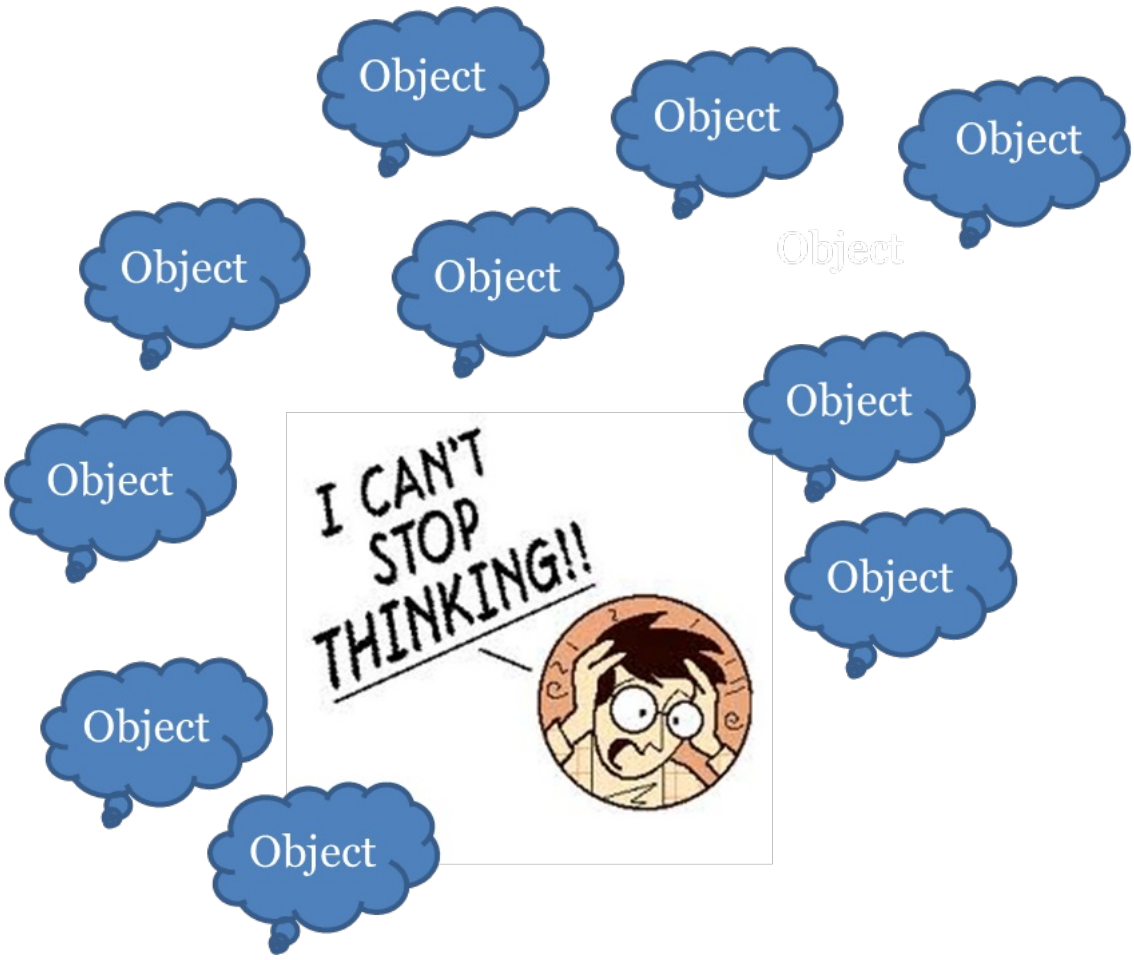
Thomas: No problem Alexander. Have a nice day.

Problem with the Approach 3

> Every factory is independent here. There is no strict rule for defining factories. In this approach each factory can have their own structure and standards.

Move to Index

Day 4 – Night



Let's list down all our requirements.

- We need Factory for online Courses
- We need Factory for corporate Courses
- We don't want to create single Factory for both modules.
- There should be some rule how to define factories, which everybody should follow.

How it would be if we use combination of approach 3 (that is whenever required create new factory)
plus Inheritance or Composition.

Time to sleep.

Move to Index

Day 5

What Alexander was talking about last night?

- Put all common functionalities in a single place (class) and reuse them in all factories.
  We call it “ReusbaleClass” for discussion purpose. We need reusability - we can do it by 2 ways.
  - With inheritance – Derive all Factories from ReusbaleClass --> Call it Solution 1
  - With composition – All factories will have a member of type ReusbaleClass --> Call it Solution 2
- Let factories override some functionality (which is different for every factory) defined in ReusbaleClass.
  We need to override means Solution 1 will fit perfect.

Approach 4 – Factory Method Approach

In short we need a class here which will do all the common tasks and expose a virtual or abstract function.

So let’s create AbstractCourseFactory which will encapsulate common functionalities with an additional overridable (virtual or abstract) method and then recreate our OnlineCourseFactory and CorporateCourseFactory.

```
public abstract class AbstractCourseFactory
{
   public AbstractCourse CreateCourse(string ScheduleType)
   {
      AbstractCourse objCourse = this.GetCourse(ScheduleType);
      objCourse.CreateCourseMaterial();
      objCourse.CreateSchedule();
      return objCourse;
   }
   public abstract AbstractCourse GetCourse(string ScheduleType);
}
```

*Please note GetCourse method is **abstract***

Now everyone (every factory) will easily override this GetCourse method.

```
public class CorporateCourseFactory:AbstractCourseFactory
{
   public override AbstractCourse GetCourse(string ScheduleType)
   {
      switch(ScheduleType)
      {
         case "N":
         return new CNetCourse();
         case "J":
         return new CJavaCourse();
         default:
         return null;
      }
   }
}
public class OnlineCourseFactory : AbstractCourseFactory
{
   public override AbstractCourse GetCourse(string ScheduleType)
   {
      switch(ScheduleType)
      {
         case "N":
         return new ONetCourse();
         case "J":
         return new OJavaCourse();
         default:
         return null;
      }
   }
}
```

How Gang of Four defined Factory Method Pattern

“This pattern defines an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses". Definition is self-explanatory now.

Move to Index

Day 6

Time to show the final demo to Thomas.

Now final UI code will look like

Online Course UI

```
AbstractCourseFactory objFactory = new OnlineCourseFactory();
AbstractCourse objCourse = objFactory.CreateCourse(StrUserInput);
//Display objCourse.CourseName and objCourse.CourseDuration
```

Corporate Course UI

```
AbstractCourseFactory objFactory = new CorporateCourseFactory();
AbstractCourse objCourse = objFactory.CreateCourse(StrUserInput);
//Display objCourse.CourseName and objCourse.CourseDuration
```

Advantages with the Approach 4

> If in the future a new course group (say for instance offline course) introduced in Programmer24X7, a new factory will be derived from AbstractCourse, encapsulating creation of all concrete courses (say for instance Offline Java Courses, Offline .NET Course) related to that group. It's that simple.

That's all about the Factory Method patterns. Finally Thomas seems happy because all his requirements seem satisfied and Alexander is also happy because now he has become champ in Factory Method Pattern.

Move to Index

Additional approaches and thoughts

- **Is parent factory has to be abstract always?**
  Answer for this question is No. We can make it non abstract if required and add default implementation to GetCourse method making it virtual.
- **Is switch is must in factories?**
  No. We can replace switch loop with the Dictionary in .NET.
- **Is inheritance from second level factory is possible?**
  Yes absolutely and that's how factory method works. Initially we had AbstractCourseFactory with its default implementation for GetCourse method, which will be later extended by OnlineCourseFactory adding its own implementation for GetCourse method, which in turn can be a virtual method. So that later say when a new OnlineAdvanceCourse is introduced in Programmer24X7 existing, OnlineCourseFactory can easily be extended.

- **What will be the advantage of using MultiLevel inheritance in case of Factory Method Pattern?**
  Well, consider following scenario.
  - We have our AbstractCourseFactory and OnlineCourseFactory.
  - OnlineCourseFactory override GetCourse method for defining online courses that is Java and .net.
  - Now let's assume new courses are added after some year to Online Course and we are not even interested in opening OnlineCourseFactory.
  - OnlineCourseFactoryVersion2 will be introduced now.
  - Now if we derive OnlineAdvanceCourseFactory from AbstractCourseFactory we are supposed to define all previous courses like Java and .NET along with new one resulting redundant code.
  - Other than that what we can do is we will take help of Dictionaries for storing all available courses, and goes adding courses to it on each level.

Let's look at the code snippet for same.

```
public abstract class AbstractCourseFactory
{
   protected Dictionary<string, AbstractCourse> ObjCourses;
   public AbstractCourseFactory()
   {
      ObjCourses = new Dictionary<string, AbstractCourse>();
   }
}
public class OnlineCourseFactoryVersion2 : OnlineCourseFactory
{
   public OnlineCourseFactoryVersion2()
   {
      ObjCourses.Add("V", new OVCourse());
   }
}
```

Note: So using Dictionaries in Factory Method Pattern is always considered as a good approach.

So this completes my first article on Factory Pattern... Hope you liked it... Please don't forget to leave your comments below.... Your positive and negative comments will encourage me to write better... Thank you!!!

Move to Index

Source Code Downloads

Click Following links for complete source code

For any training related to other design patterns and .NET you can contact me at SukeshMarla@Gmail.com or at www.sukesh-marla.com

Click and go here to get more on .NET and C# learning stuffs

License

Share

About the Author