Software Architecture & Design SEC3071

Lecture No. 33

Muhammad Shahid
Department of Computer Science
National Textile University

shahid.abdullah@hotmail.com

Last Lecture Review

- Creational Design Patterns
- Singleton Design Pattern
- Possible Implementations
  - Using Global Variables
  - Problem in the Approach
  - Solution
- Singleton Design Pattern
  - Definition
  - Class Diagram
  - Code Implementation
- Singleton in Multithreaded Environment
- Singleton Pattern – Applicability

Agenda – What will you Learn Today?

Builder Design Pattern

Design Patterns Space

		Purpose		
		Creational	**Structural**	**Behavioral**
Scope	**Class**	Factory Method	Adapter	Interpreter
	Object	▪ Abstract Factory ▪ Builder ▪ Prototype ▪ Singleton	▪ Adapter ▪ Bridge ▪ Composite ▪ Decorator ▪ Facade ▪ Flyweight ▪ Proxy	▪ Chain of Responsibility ▪ Command ▪ Iterator ▪ Mediator ▪ Memento ▪ Observer ▪ State ▪ Strategy ▪ Visitor

Builder Design Pattern

Builder Design Pattern - Intent

- Separate the construction of a complex object from its representation so that the same construction process can create different representations

Builder Design Pattern

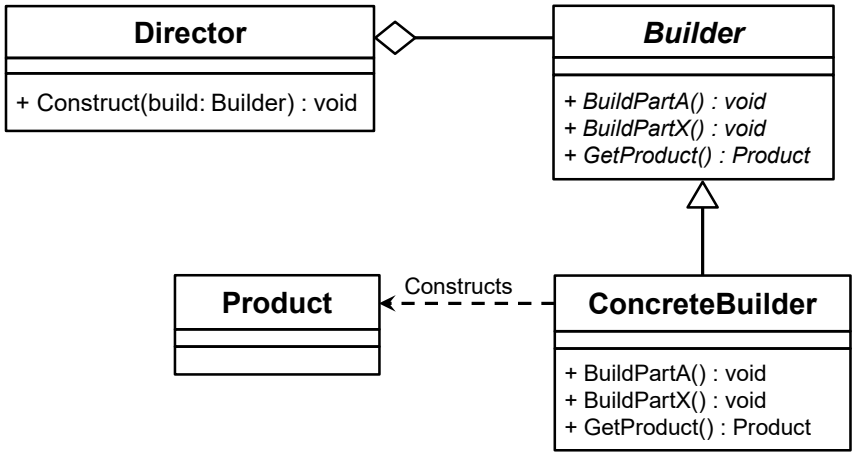
- Instead of having client objects invoke different builder methods directly, the Builder pattern suggests using a dedicated object referred to as a **Director**, which is responsible for invoking different builder methods required for the construction of the final object

Builder Design Pattern

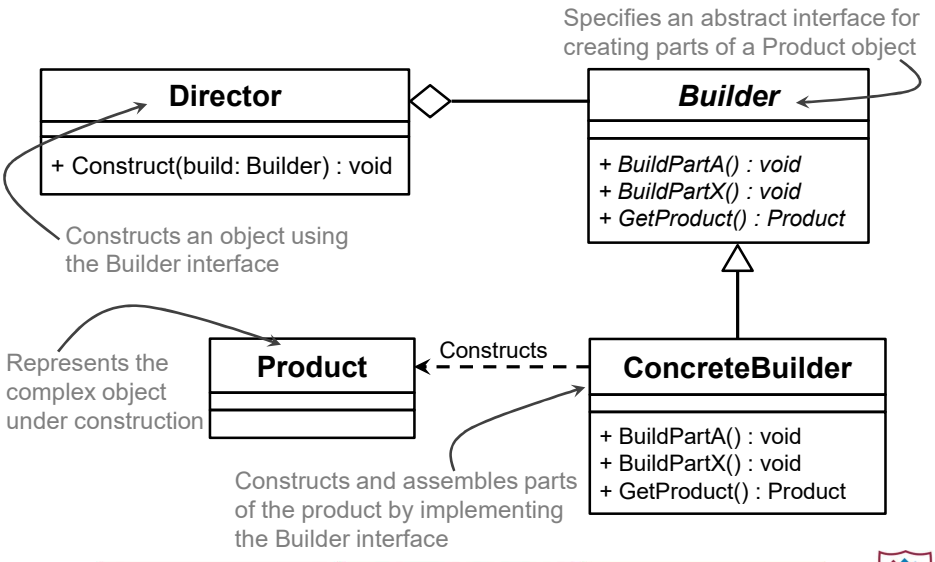
- Different client objects can make use of the **Director** object to create the required object and once the object is constructed, the client object can directly request from the **Builder** the fully constructed object

- A new method `GetProduct` can be declared in the common **Builder** interface/abstract class to be implemented by different concrete builder

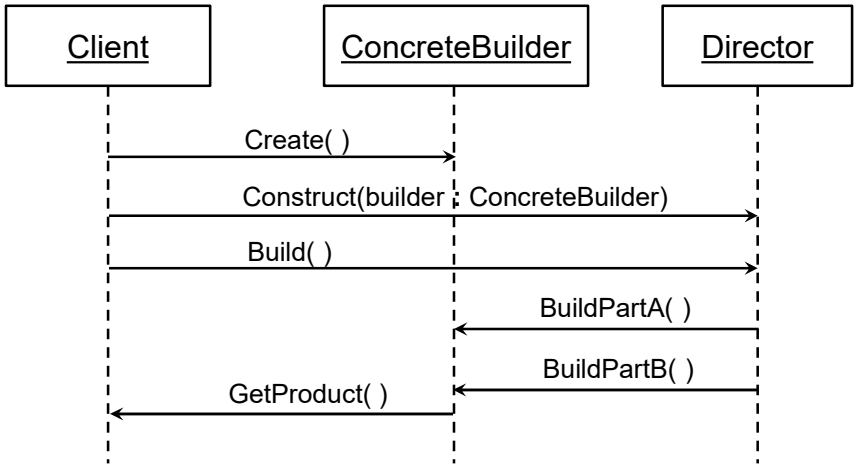
Class Diagram



Class Diagram



Builder DP - Sequence Diagram

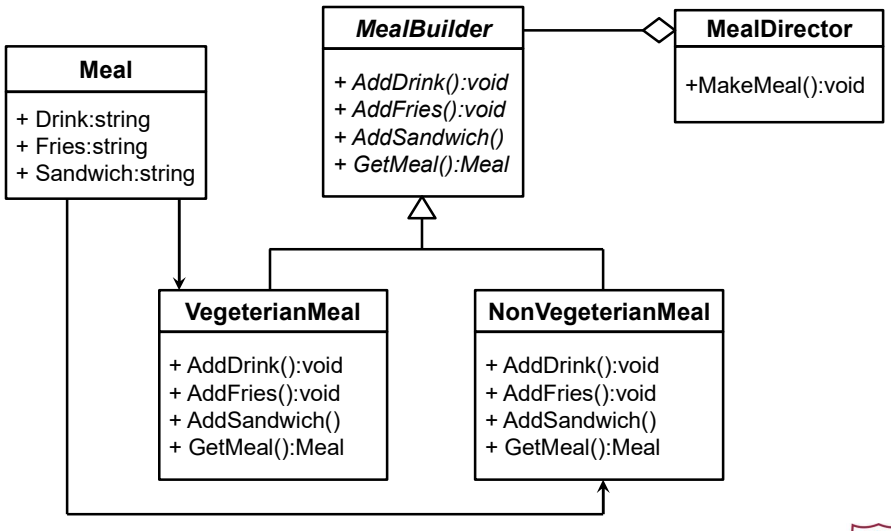


Application – Get Your Meal

Get Your Meal

- Consider the choice of eating meal. The meal is offered for vegetarians and non-vegetarians people. So depending upon the persons meal can be prepared. The preparation of meal may comprise of Sandwich, Fries and Drink. So meal is prepared (constructed) by constructing these products.

Get Your Meal – Class Diagram



Meal & MealBuilder Classes

```
public class Meal
{
      public string Sandwich { get; set; }
      public string Fries { get; set; }
      public string Drink { get; set; }

      public override string ToString()
      {
            return string.Format("Sandwitch:{0}\n
            Fries:{1}\n Drink:{2}", Sandwich, fries,
            drink);
      }
} // End of Meal class
```


Meal & MealBuilder Classes

```
public abstract class MealBuilder
{
      public abstract void AddSandwich();
      public abstract void AddFries();
      public abstract void AddDrink();
      public abstract Meal GetMeal();
} // End of MealBuilder class
```


Meal & MealBuilder Classes

```
public class VegeterianMeal : MealBuilder
{
      private Meal meal = new Meal();
      public override void AddSandwich() {
            meal.Sandwich = "Mixed Vegetable";
      }
      public override void AddFries() {
            meal.Fries = "Potato";
      }
      public override void AddDrink() {
            meal.Drink = "Sprite";
      }
      public override Meal GetMeal(){ return meal; }
} // End of VegeterianMeal class
```


Meal & MealBuilder Classes

```
public class NonVegeterianMeal : MealBuilder
{
      private Meal meal = new Meal();
      public override void AddSandwich() {
            meal.Sandwich = "Beef";
      }
      public override void AddFries() {
            meal.Fries = "Fish";
      }
      public override void AddDrink() {
            meal.Drink = "Pepsi";
      }
      public override Meal GetMeal(){ return meal; }
} // End of NonVegeterianMeal class
```


Meal & MealBuilder Classes

```
public class MealDirector
{
      public void MakeMeal(MealBuilder mealBuilder)
      {
            mealBuilder.AddSandwich();
            mealBuilder.AddFries();
            mealBuilder.AddDrink();
      }
} // End of MealDirector class
```


Meal & MealBuilder Classes

```
static void Main(string[] args)
{
      MealDirector nvDir = new MealDirector();
      MealBuilder nvBuilder = new NonVegeterianMeal();
      nvDir.MakeMeal(nvBuilder);
      Meal nvMeal = nvBuilder.GetMeal();
      Console.WriteLine("---- Non Vegeterian Meal ----");
      Console.WriteLine(nvMeal);

      MealDirector vDir = new MealDirector();
      MealBuilder vBuilder = new VegeterianMeal();
      vDir.MakeMeal(vBuilder);
      Meal vegMeal = vBuilder.GetMeal();
      Console.WriteLine("---- Vegeterian Meal ----");
      Console.WriteLine(vegMeal);
} // End of Main method
```


Tester Class

```
---------- VEGETERIAN ----------
     Sandwich:      Mixed Vegetable
     Fries:         Potato
     Drink:         Sprite

-------- NON-VEGETERIAN --------
     Sandwich:      Beef
     Fries:         Fish
     Drink:         Pepsi
```


Builder Pattern – Applicability

- Use the builder pattern when:
  - The algorithm for creating a complex object should be independent of the parts that make up the object and how they are assembled
  - The construction process must allow different representations for the object that's constructed
  - Builder patterns fits when different variations of an object may need to be created and the inheritance into those objects is well-defined

Builder Pattern – Applicability

- Builder Design Pattern
  - Motivation
  - Intent
  - Class Diagram
  - Sequence Diagram
  - Implementation
- Application – Get Your Meal
- Applicability

Questions