|  |  |
| --- | --- |
|  | LAB | Gang of Four Design Patterns |
|  | WORKSHOP | Design Patterns |
|  | Crystal Tenn  crtenn@microsoft.com |

# SETUP LAB ENVRIOMENT

## Prerequisites

To perform the tasks in this lab you need following:

* Visual Studio 2015 or 2017, any version

## Objectives

In this lab you will complete following tasks:

* Work through samples of code and read through why you would use each design pattern.
* Pay close attention to the highly used design patterns!!! They will be marked as 4/5 or 5/5 for frequency of use.
* At the end you will have a working solution with 23 projects with one working sample per design pattern.

### Estimated Completion Time: 60 minutes.

## Reference

**Creational Patterns (5)**

* Abstract Factory: Creates an instance of several families of classes
* Builder: Separates object construction from its representation
* Factory Method: Creates an instance of several derived classes
* Prototype: A fully initialized instance to be copied or cloned
* Singleton: A class of which only a single instance can exist

**Structural Patterns (7)**

* Adapter: Match interfaces of different classes
* Bridge: Separates an object’s interface from its implementation
* Composite: A tree structure of simple and composite objects
* Decorator: Add responsibilities to objects dynamically
* Façade: A single class that represents an entire subsystem
* Flyweight: A fine-grained instance used for efficient sharing
* Proxy: An object representing another object

**Behavioral Patterns (11)**

* Chain of Resp.: A way of passing a request between a chain of objects
* Command: Encapsulate a command request as an object
* Interpreter: A way to include language elements in a program
* Iterator: Sequentially access the elements of a collection
* Mediator: Defines simplified communication between classes
* Memento: Capture and restore an object's internal state
* Observer: A way of notifying change to a number of classes
* State: Alter an object's behavior when its state changes
* Strategy: Encapsulates an algorithm inside a class
* Template Method: Defer the exact steps of an algorithm to a subclass
* Visitor: Defines a new operation to a class without change

## Task: Creational Patterns

Creational Patterns deal with **controlling the creation objects**. These design patterns all relate to rules to do with creation of instances(s) of classes.

Try to remember this phrase: **“**Both types of **Factory** (Abstract Factory and Factory) patterns help you **Build** (Builder) a **Single** (Singleton) **Prototype.”**

* Abstract Factory: Creates an instance of several families of classes
* Builder: Separates object construction from its representation
* Factory Method: Creates an instance of several derived classes
* Prototype: A fully initialized instance to be copied or cloned
* Singleton: A class of which only a single instance can exist

**Abstract Factory:** “The super factory that creates other factories.”

Frequency of Use: 5/5 High

Use case: You want to use this pattern if you see

1. Open the CoreDesignPatterns solution file in Visual Studio.
2. The Abstract project has already been created for you. View the files included. Note that there are two main interfaces, IColor and ICar. ICar has implementations of Toyota and Mustang. IColor has implementations of Silver and Red.
3. Create a Car Factory inheriting from AbstractFactory to generate concrete class objects. Add the code below:
4. Create a Color Factory inheriting from AbstractFactory to generate concrete class objects. Add the code below:
5. Create a Factory Producer class to get factories
6. Verify your output.

**Builder**:

Frequency of Use: 2/5 Low

Use Case:

1. The Builder project has already been created for you. View the VehicleBuilder class.
2. View the Motorcycle Builder, ScooterBuilder, and CarBuilder classes.
3. Create a new class called Shop.cs and add the following code:

/// <summary>

/// The 'Director' class

/// </summary>

public class Shop

{

// Builder uses a complex series of steps

public void Construct(VehicleBuilder vehicleBuilder)

{

vehicleBuilder.BuildFrame();

vehicleBuilder.BuildEngine();

vehicleBuilder.BuildWheels();

vehicleBuilder.BuildDoors();

}

}

1. Set the Builder project as your startup project, run it. You should get the following output:

**Factory Method:**

Frequency of Use: 5/5 High

Use case:

1. The FactoryClient and FactoryMethod projects have already been created for you.
2. Please view the files in the Factory Method project in the Concrete and Contracts file directories.
3. In the empty Factory Directory, add a class called Strawberry Factory.cs and add the following code:
4. In the FactoryClient project’s GroceryStoreClient.cs file, add the following code:
5. Set this project as startup and run it, you should get an output like this:

Prototype:

Frequency of Use: 3/5 Medium

Use Case:

1. Go to the project called Prototype.

Singleton:

Frequency of Use: 4/5 Medium High

Use Case:

1. Go to the project called Singleton.
2. Create a LoggingService.cs file:
3. Go to the SingletonLauncher class, uncomment the code there.

## Task: Structural Patterns

**The Adapter Pattern**

Frequency of Use: 4/5 Medium High

Use Case:

1. Go to the Adapter pattern project under the StructuralPatterns folder.
2. View the code
3. Add a new class CompoundAdapter.cs where you put the following code:

**Bridge Pattern**

Frequency of Use: 3/5 Medium

Use Case:

1. Go to the Adapter pattern project under the StructuralPatterns folder.

View the code

**Composite Pattern**

Frequency of Use: 4/5 Medium High

Use Case:

1. Go to the Adapter pattern project under the StructuralPatterns folder.

View the code

**Decorator Pattern**

Frequency of Use: 3/5 Medium

Use Case:

1. Go to the Adapter pattern project under the StructuralPatterns folder.

View the code

**Facade Pattern**

Frequency of Use: 5/5 High

Use Case:

1. Go to the Adapter pattern project under the StructuralPatterns folder.

View the code

**Flyweight Pattern**

Frequency of Use: 1/5 Very Low

Use Case:

1. Go to the Adapter pattern project under the StructuralPatterns folder.

View the code

**Proxy Pattern**

Frequency of Use: 4/5 Medium High

Use Case:

1. Go to the Adapter pattern project under the StructuralPatterns folder.

View the code

## Task: Behavioral Patterns

Chain of Responsibility Pattern

Frequency of Use: 2/5 Low

Use Case:

1. Go to the Adapter pattern project under the StructuralPatterns folder.

View the code

Command Pattern

Frequency of Use: 4/5 Medium High

Use Case:

1. Go to the Adapter pattern project under the StructuralPatterns folder.

View the code

Interpreter Pattern

Frequency of Use: 1/5 Very Low

Use Case:

1. Go to the Adapter pattern project under the StructuralPatterns folder.

View the code

Iterator Pattern

Frequency of Use: 5/5 High

Use Case:

1. Go to the Adapter pattern project under the StructuralPatterns folder.

View the code

Mediator Pattern

Frequency of Use: 2/5 Low

Use Case:

1. Go to the Adapter pattern project under the StructuralPatterns folder.

View the code

Memento Pattern

Frequency of Use: 1/5 Very Low

Use Case:

1. Go to the Adapter pattern project under the StructuralPatterns folder.

View the code

Observer Pattern

Frequency of Use: 5/5 High

Use Case:

1. Go to the Adapter pattern project under the StructuralPatterns folder.

View the code

State Pattern

Frequency of Use: 3/5 Medium

Use Case:

1. Go to the Adapter pattern project under the StructuralPatterns folder.

View the code

Strategy Pattern

Frequency of Use: 4/5 Medium High

Use Case:

1. Go to the Adapter pattern project under the StructuralPatterns folder.

View the code

Template Method Pattern

Frequency of Use: 3/5 Medium

Use Case:

1. Go to the Adapter pattern project under the StructuralPatterns folder.

View the code

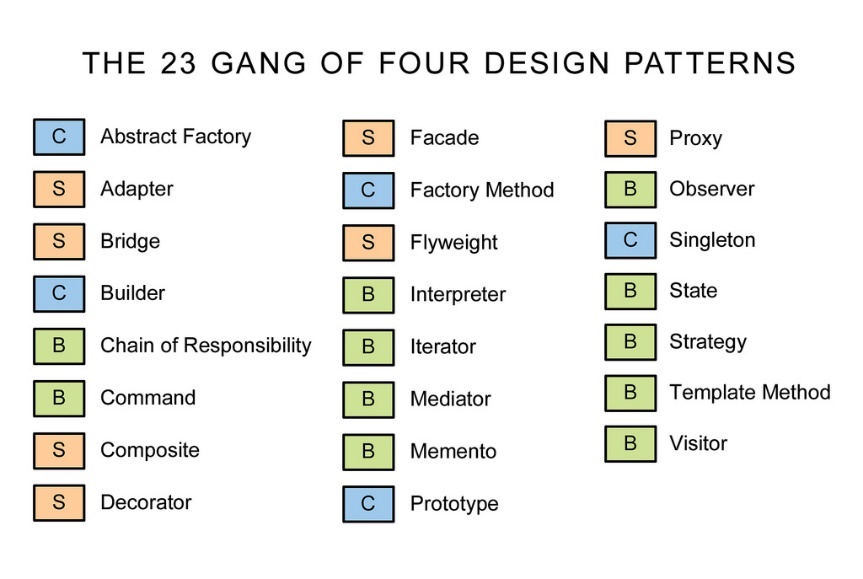
Visitor Pattern

Frequency of Use: 1/5 Very Low

Use Case:

1. Go to the Adapter pattern project under the StructuralPatterns folder.

View the code



Quick Quiz!

1. Which patterns are the most commonly used?
2. Which patterns are the least commonly used?