DESIGN PATTERNS

WHAT ARE DESIGN PATTERNS?

Design patterns are predefined solutions to general software development problems.

Design pattern solutions represent best practices.

Design patterns are tried and tested solutions developed by experienced object-oriented developers.

WHY USE DESIGN PATTERNS?

STANDARDIZE SOLUTIONS

A standard solution for a particular scenario allows developers to build an application around a commonly know pattern

WHY USE DESIGN PATTERNS?

BEST PRACTICES

Design patterns have been developed and tested by experienced developers over a long period of time and increases the speed and ease of software development of less experienced developers

MAIN DESIGN PATTERN TYPES

CREATIONAL PATTERNS

Provides a way to create objects without directly using the new operator.

STRUCTURAL PATTERNS

Defines new ways to compose objects to obtain new functionality.

MAIN DESIGN PATTERN TYPES

BEHAVIORAL PATTERNS

Define ways to communicate between objects.

COMMON DESIGN PATTERNS

- Factory
- Singleton
- MVC

FACTORY PATTERN

This pattern can use an instance to create other objects.

In other words, we can design a class that will produce other objects of other classes.

FACTORY PATTERN IMPLEMENTATION

- Create an interface.
- Create concrete classes implementing the interface.
- Create a factory to generate an object based on a given parameter.
- Use a factory to get an object of the class by passing in the requested parameter type.

FACTORY PATTERN EXAMPLE

```
public interface Shape {
void draw();
}

public class Rectangle:Shape {

    public void draw() {
    Console.WriteLine("Inside Rectangle draw()
    method.");
    }
}
```

FACTORY PATTERN EXAMPLE

```
public class ShapeFactory {
//use getShape method to get object of type shape
public Shape getShape(string ShapeType){
//Code to create different object types
}
}
```

FACTORY PATTERN EXAMPLE

```
Shape shape1 = shapeFactory.getShape("CIRCLE");
//call draw method of Circle
shape1.draw();
//get an object of Rectangle and call its draw method.
Shape shape2 = shapeFactory.getShape("RECTANGLE");
//call draw method of Rectangle
shape2.draw();
```

SINGLETON PATTERN

The Singleton pattern enables programmers to control how many instances (mostly one) that are allowed to be made.

When those instances are created, we can get global access to them.

SINGLETON PATTERN IMPLEMENTATION

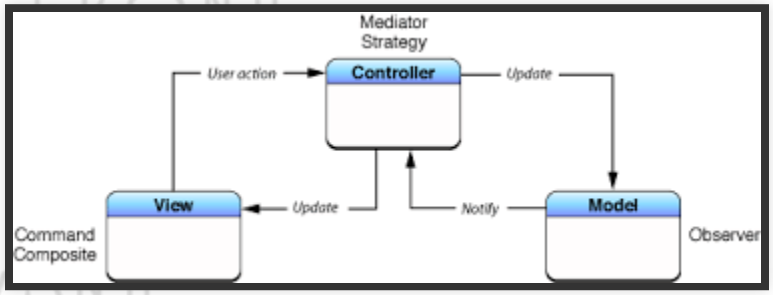
- Create an Singleton class
- Create Singleton static object declaration
- Make a private constructor
- Create a static get instance method

SINGLETON PATTERN EXAMPLE

```
public class SingleObject
{
private static SingleObject instance = new SingleObject();
private SingleObject(){}
public static SingleObject getInstance(){return instance;}
}
```

MVC PATTERN

The Model-View-Controller pattern is used to separate the layers of an interactive application



MVC PATTERN

MODEL

Model represents an object or JAVA POJO carrying data. It can also have logic to update controller if its data changes.

VIEW

View represents the visualization of the data that model contains

MVC PATTERN

CONTROLLER

Controller acts on both model and view. It controls the data flow into model object and updates the view whenever data changes. It keeps view and model separate.

RECAP

WHAT YOU SHOULD KNOW AT THIS POINT:

- What is a design pattern.
- Know when to use a design pattern.
- How three main design patterns.
- How how to implement the three main design patterns.