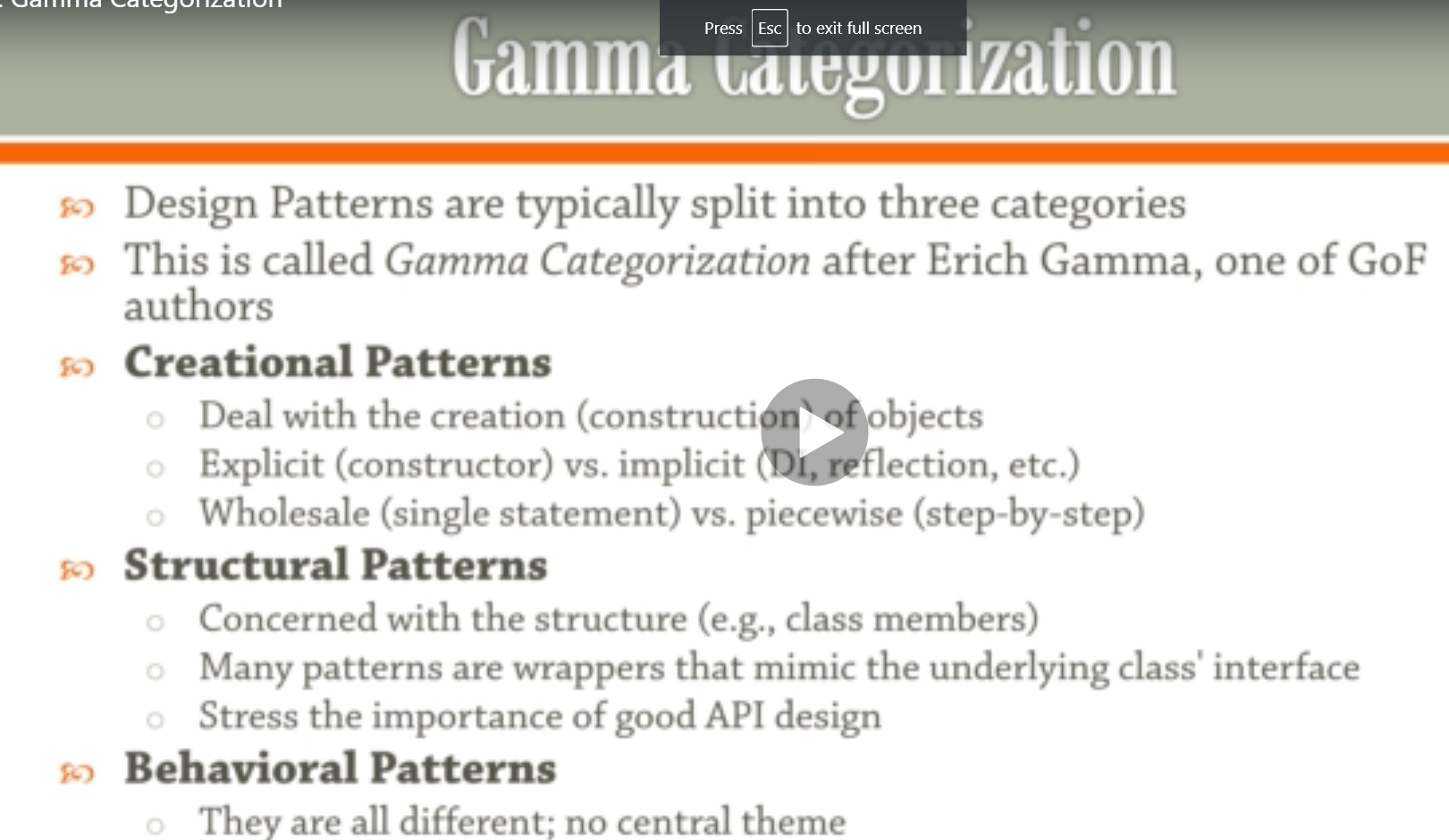
**Lesson01 Gamma Categorization**

**Notes:-**



**1-there are different types of pattern types as below**

**A-Creational Pattern**

**B-Structural Pattern**

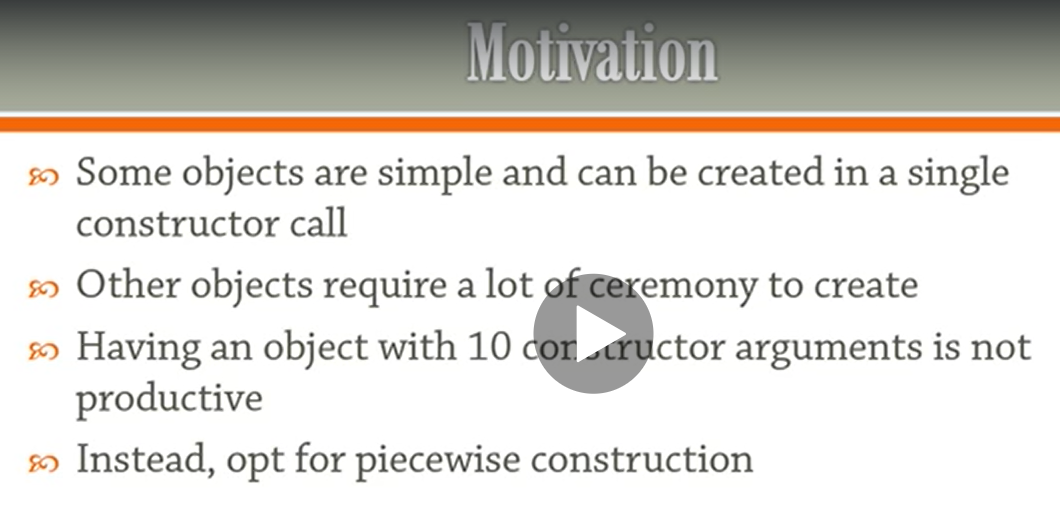
**C-Behavioral Pattern**

**Lesson02 Builder Design Pattern**

**Notes:-**

**1-we using builder design pattern when the initialize constructor begin too complicated**

**(When piecewise object construction is complicated, provide API for doing it successfully)**



**Lesson03 Life without Builder Design Pattern**

**Notes:-**

**1-in order to write html builder we have to write hard code in each set html element as below**

**static void Main(string[] args){**

**// life without builder design pattern**

**//its not practical to set hard code to set the html element and text as below**

**var hello = "hello";**

**var sb = new StringBuilder();**

**sb.Append("<p>");**

**sb.Append(hello);**

**sb.Append("</p>");**

**Console.WriteLine(sb);**

**var words = new[] { "hello", "world" };**

**sb.Clear();**

**sb.Append("<ul>");**

**foreach (var word in words){sb.AppendFormat("<li>{0}</li>", word);}**

**sb.Append("</ul>");**

**Console.WriteLine(sb);}**

**Lesson04 Life with Builder Design Pattern**

**Notes:-**

**1-we create Models >HtmlElement.cs**

**(The basic idea is to make class that contains element of the same type Html Element and the Builder contains the instance of the Html Element which is the parent element)**

**using System;**

**using System.Collections.Generic;**

**using System.Text;**

**namespace BuilderDesignPatternPro.Models{**

**//class HtmlELement contain the injection of create StringBuilder**

**public class HtmlElement{**

**//we define the element name and text**

**public string Name, Text;**

**//we define list of the same class**

**public List<HtmlElement> Elements = new List<HtmlElement>();**

**//used for the provide space**

**private const int indentSize = 2;**

**public HtmlElement() { }**

**public HtmlElement(string name, string text){**

**Name = name ?? throw new ArgumentNullException(paramName: nameof(name));**

**Text = text ?? throw new ArgumentNullException(paramName: nameof(text));}**

**//we using custom string template that create string builder**

**public override string ToString(){return ToStringTemplate(0);}**

**private string ToStringTemplate(int ident){**

**var sb = new StringBuilder();**

**var i = new string(' ', indentSize \* ident);**

**sb.AppendLine($"{i}<{Name}>");**

**if (!string.IsNullOrWhiteSpace(Text)){**

**sb.Append(new string(' ', indentSize \* (ident + 1)));**

**sb.AppendLine(Text);}**

**foreach (var e in Elements){sb.Append(e.ToStringTemplate(ident + 1));}**

**sb.AppendLine($"{i}</{Name}>");**

**return sb.ToString();}}}**

**2-create Builder > HtmlBuilder.cs**

**using BuilderDesignPatternPro.Models;**

**namespace BuilderDesignPatternPro.Builder{**

**//we define HtmlBuilder that using HtmlELement class that provide creational pattern that create String builder internally**

**//this HtmlBuilder provide add child and tostring custom method**

**public class HtmlBuilder{**

**private readonly string rootName;**

**//we create instance of Html Element statically as below**

**HtmlElement root = new HtmlElement();**

**public HtmlBuilder(string rootName){**

**this.rootName = rootName;**

**root.Name = rootName;}**

**public void AddChild(string childName, string childText){**

**var e = new HtmlElement(childName, childText);**

**root.Elements.Add(e);}**

**public override string ToString(){return root.ToString();}**

**public void Clear(){root = new HtmlElement { Name = rootName };}}}**

**3-on Program.cs we set the following code**

**static void Main(string[] args){**

**//we pass the rootName that used as parent for the child elements**

**var builder = new HtmlBuilder("ul");**

**builder.AddChild("li", "hello");**

**builder.AddChild("li", "world");**

**//this method call the custom ToString() that pass the Name ul in the end of the html**

**Console.WriteLine(builder.ToString());**

**Console.ReadLine();}**