Table of Contents

[SOLID Principles 3](#_Toc108367152)

[Single Responsibility Principle (SRP) 3](#_Toc108367153)

[Open Closed Principle (OCP) 3](#_Toc108367154)

[Liskov Substitutional Principle (LSP) 3](#_Toc108367155)

[Interface Segregation Principle (ISP) 3](#_Toc108367156)

[Dependency Inversion Principle 3](#_Toc108367157)

[Design Patterns 3](#_Toc108367158)

[Creational Design Pattern 3](#_Toc108367159)

[Abstract Factory 3](#_Toc108367160)

[Builder 3](#_Toc108367161)

[Factory Pattern 3](#_Toc108367162)

[Case Study 3](#_Toc108367163)

[Reference 5](#_Toc108367164)

[Prototype 6](#_Toc108367165)

[Singleton 6](#_Toc108367166)

[Structural Design Pattern 6](#_Toc108367167)

[Adaptor 6](#_Toc108367168)

[Bridge 6](#_Toc108367169)

[Composite 6](#_Toc108367170)

[Decorator 6](#_Toc108367171)

[Façade 6](#_Toc108367172)

[Flyweight 6](#_Toc108367173)

[Proxy 6](#_Toc108367174)

[Behavioral Design Pattern 6](#_Toc108367175)

[Chain of Resp 6](#_Toc108367176)

[Commanda 6](#_Toc108367177)

[Interpreter 6](#_Toc108367178)

[Iterator 6](#_Toc108367179)

[Mediator 6](#_Toc108367180)

[Memento 6](#_Toc108367181)

[Observer 6](#_Toc108367182)

[State 6](#_Toc108367183)

[Strategy 6](#_Toc108367184)

[Template Method 6](#_Toc108367185)

[Visitor 6](#_Toc108367186)

[Reference 7](#_Toc108367187)

# SOLID Principles

## Single Responsibility Principle (SRP)

## Open Closed Principle (OCP)

## Liskov Substitutional Principle (LSP)

## Interface Segregation Principle (ISP)

## Dependency Inversion Principle

# Design Patterns

# Creational Design Pattern

## Abstract Factory

## Builder

## Factory Pattern

It is from the family of creational patterns, which means to construct or create something.

## Case Study

|  |
| --- |
| public static void Main(string[] args)  {  GetInvoice\_bad(1);  }  public static void GetInvoice\_bad(int invoiceTypeId)  {  if (invoiceTypeId== 1)  {  Models.InvoiceWithHeader invoiceWithHeader = new();  invoiceWithHeader.Print();  }  else if (invoiceTypeId== 2)  {  Models.InvoiceWithoutHeader invoiceWithoutHeader = new();  invoiceWithoutHeader.Print();  }  } |

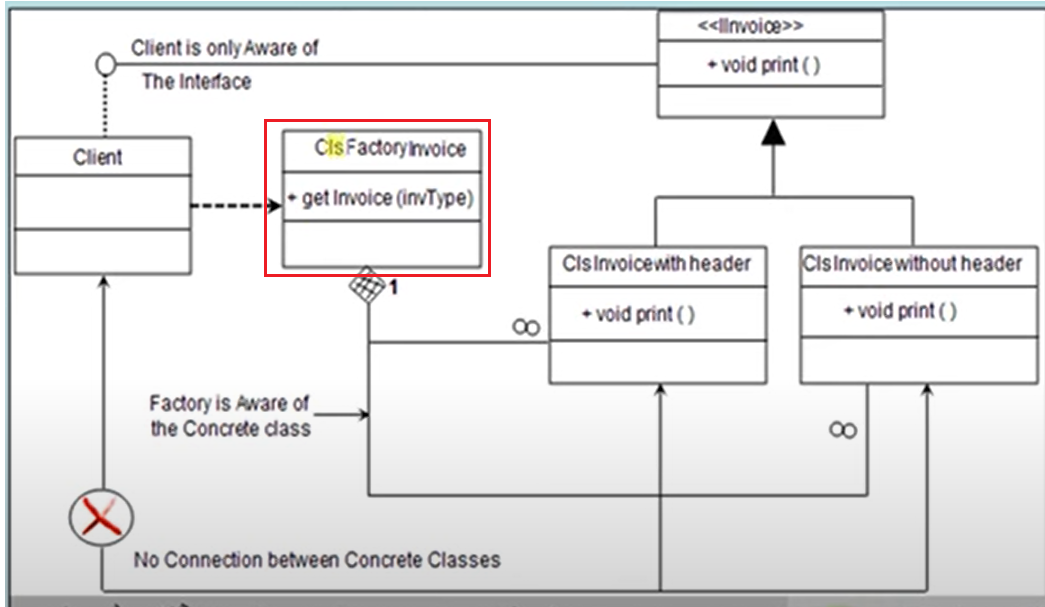
1. From the above code it is clear that we are creating and object based on address type id.

Problems:

* “new” keyword: use new keyword on each type of address creation. Which may results lot of instance of object. (Lot of scattered new keywords)
* Condition: creating objects based on conditions may result to enhance controller logic every time whenever a new address type added which result in creating a “new” object and code duplication. (Client is aware of all invoice types)

Resolution:

* We can overcome the above problem, which the help of Factory pattern.
* “Factory Class” – Creating a Factory class, and it is responsible for creating a object based on address type (which will help to resolve the dependency with new key word)
* Creating an Interface, and concrete class implements that interface, will resolve a problem of adding a new client.



|  |
| --- |
| public interface IInvoice  {  void Print();  }  public class InvoiceWithHeader : IInvoice  {  public void Print()  {  Console.WriteLine("Hello from Invoice with header");  }  }  public class InvoiceWithoutHeader : IInvoice  {  public void Print()  {  Console.WriteLine("Hello from Invoice without header");  }  }  public class InvoiceFactory  {  public static IInvoice GetInvoice(int invoiceType)  {  IInvoice invoiceObj = null;  if (invoiceType == 1)  invoiceObj = new InvoiceWithHeader();  else if (invoiceType == 2)  invoiceObj = new InvoiceWithoutHeader();  return invoiceObj;  }  } |

|  |
| --- |
| public static void GetInvoice\_factory()  {  IInvoice invoice = null;  int invoiceTypeId = 0;  Console.WriteLine("Enter Invoice Type, Supported (1 & 2)");  invoiceTypeId = Convert.ToInt32(Console.ReadLine());  invoice = InvoiceFactory.GetInvoice(invoiceTypeId);  invoice.Print();  }  public static void Main(string[] args)  {  GetInvoice\_factory();  } |

## Reference

<https://github.com/saihari-lgr/Patterns>

## Prototype

## Singleton

# Structural Design Pattern

## Adaptor

## Bridge

## Composite

## Decorator

## Façade

## Flyweight

## Proxy

# Behavioral Design Pattern

## Chain of Resp

## Commanda

## Interpreter

## Iterator

## Mediator

## Memento

## Observer

## State

## Strategy

## Template Method

## Visitor

# Reference

|  |
| --- |
| <https://www.youtube.com/watch?v=agkWYPUcLpg&list=PLqF02iWwxMP5sp42wwR_UZ8aB7MTEqGPh> |
|  |
|  |