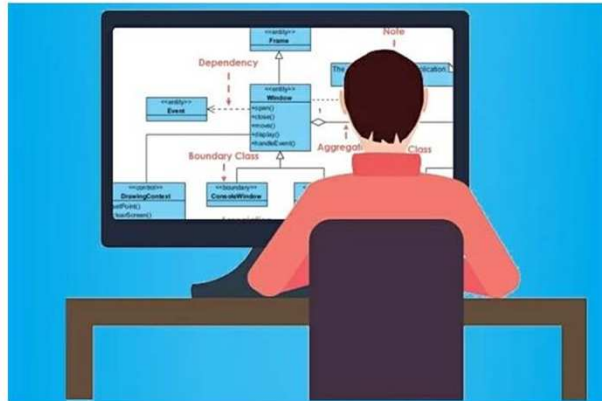


# Software Design & Architecture

## Spring 2022 - Week-06



مدرس: مهندس ماجد کلیم  
جامعہ بحریہ، واقعہ گاہ کراچی  
*Engr. Majid Kaleem*

### WEEKLY AGENDA

TENTATIVE WEEKLY DATES	TENTATIVE TOPICS
1 Mar 7 <sup>th</sup> – Mar 11 <sup>th</sup>	INTRODUCTION TO THE COURSE; DEFINING SOFTWARE ARCHITECTURE & DESIGN CONCEPTS
2 Mar 14 <sup>th</sup> – Mar 18 <sup>th</sup>	DESIGN PRINCIPLES; OBJECT-ORIENTED DESIGN WITH UML
3 Mar 21 <sup>st</sup> – Mar 25 <sup>th</sup>	SYSTEM DESIGN & SOFTWARE ARCHITECTURE; OBJECT DESIGN; MAPPING DESIGN TO CODE
4 Mar 28 <sup>th</sup> – Apr 1 <sup>st</sup>	FUNCTIONAL DESIGN; UI DESIGN; WEB APPLICATIONS DESIGN <b>ASSIGNMENT &amp; QUIZ #1</b>
5 Apr 4 <sup>th</sup> – Apr 8 <sup>th</sup>	MOBILE APPLICATION DESIGN; PERSISTENCE LAYER DESIGN
6 Apr 11 <sup>th</sup> – Apr 15 <sup>th</sup>	CREATIONAL DESIGN PATTERNS
7 Apr 18 <sup>th</sup> – Apr 22 <sup>nd</sup>	STRUCTURAL DESIGN PATTERNS <b>ASSIGNMENT &amp; QUIZ #2</b>
8 Apr 25 <sup>th</sup> – Apr 29 <sup>th</sup>	BEHAVIORAL DESIGN PATTERNS
<b>← MID TERM EXAMINATIONS →</b>	
9 May 9 <sup>th</sup> – May 13 <sup>th</sup>	INTERACTIVE SYSTEMS WITH MVC ARCHITECTURE; SOFTWARE REUSE
10 May 16 <sup>th</sup> – May 20 <sup>th</sup>	ARCHITECTURAL DESIGN ISSUES; ARCHITECTURE DESCRIPTION LANGUAGES (ADLS)
11 May 23 <sup>rd</sup> – May 27 <sup>th</sup>	ARCHITECTURAL STYLES/PATTERNS & DESIGN QUALITIES
12 May 30 <sup>th</sup> – Jun 3 <sup>rd</sup>	ARCHITECTURAL STYLES/PATTERNS & DESIGN QUALITIES <b>ASSIGNMENT &amp; QUIZ #3</b>
13 Jun 6 <sup>th</sup> – Jun 10 <sup>th</sup>	QUALITY TACTICS; ARCHITECTURE DOCUMENTATION
14 Jun 13 <sup>th</sup> – Jun 17 <sup>th</sup>	ARCHITECTURAL EVALUATION TECHNIQUES
15 Jun 20 <sup>th</sup> – Jun 24 <sup>th</sup>	MODEL DRIVEN DEVELOPMENT <b>ASSIGNMENT (PRESENTATIONS) &amp; QUIZ #4</b>
16 Jun 27 <sup>th</sup> – Jul 1 <sup>st</sup>	REVISION WEEK
<b>← FINAL TERM EXAMINATIONS →</b>	

## WHAT ARE SOFTWARE DESIGN PATTERNS?

- Design patterns are documented *tried* and *tested solutions* (*obtained by trial and error by numerous software developers over quite a substantial period of time*) for recurring problems in a given context.
- A *design pattern* is a *generic* or *template solution* to a given problem type or genre, in a given context.
- A *design pattern* is a general *repeatable solution* to a *commonly occurring problem* in software design.

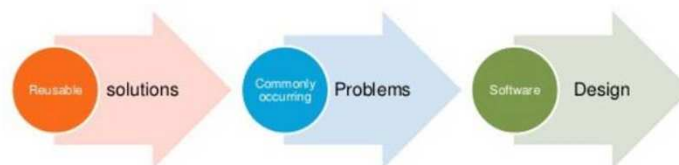
14-Apr-2022

Engr. Majid Kaleem

3

## WHAT ARE SOFTWARE DESIGN PATTERNS?

- A design pattern is a general *reusable solution* to a *commonly occurring problem* in *software design*.
- A design pattern is not a finished design that can be transformed directly into code.
- It is a description or template for how to solve a problem that can be used in many different situations.
- Learning these patterns helps *unexperienced developers* to learn software design in an easy and faster way.



14-Apr-2022

Engr. Majid Kaleem

4

## WHAT ARE SOFTWARE DESIGN PATTERNS?

- Let's say if you want to implement a sorting algorithm the first thing comes to mind is bubble sort.
- So the *problem is sorting* and *solution is bubble sort*.
- Same holds true for design patterns.

14-Apr-2022

Engr. Majid Kaleem

5

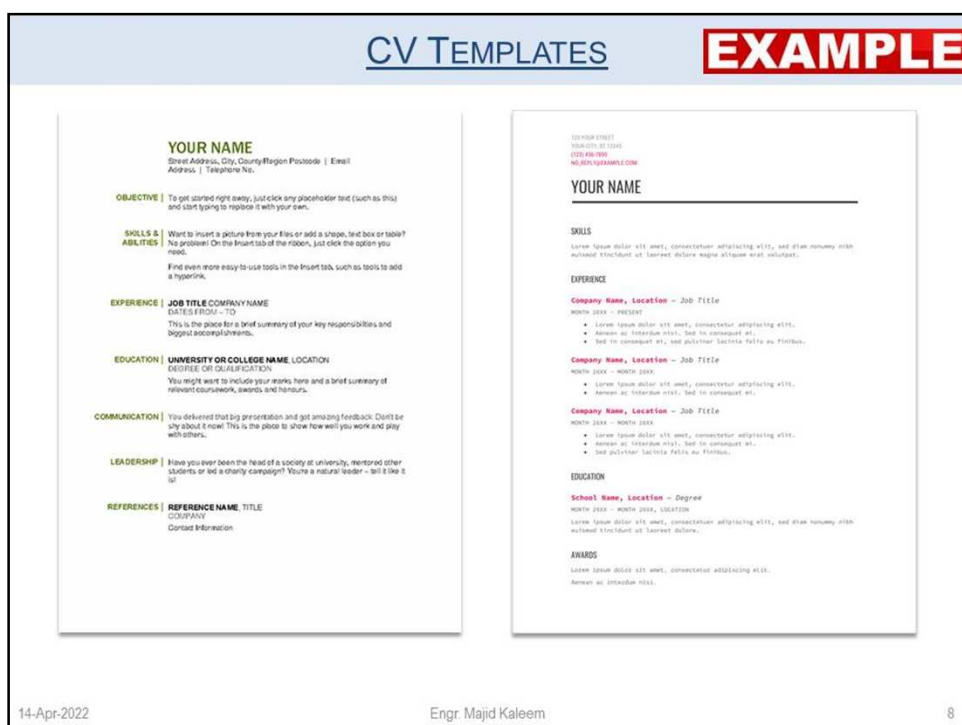
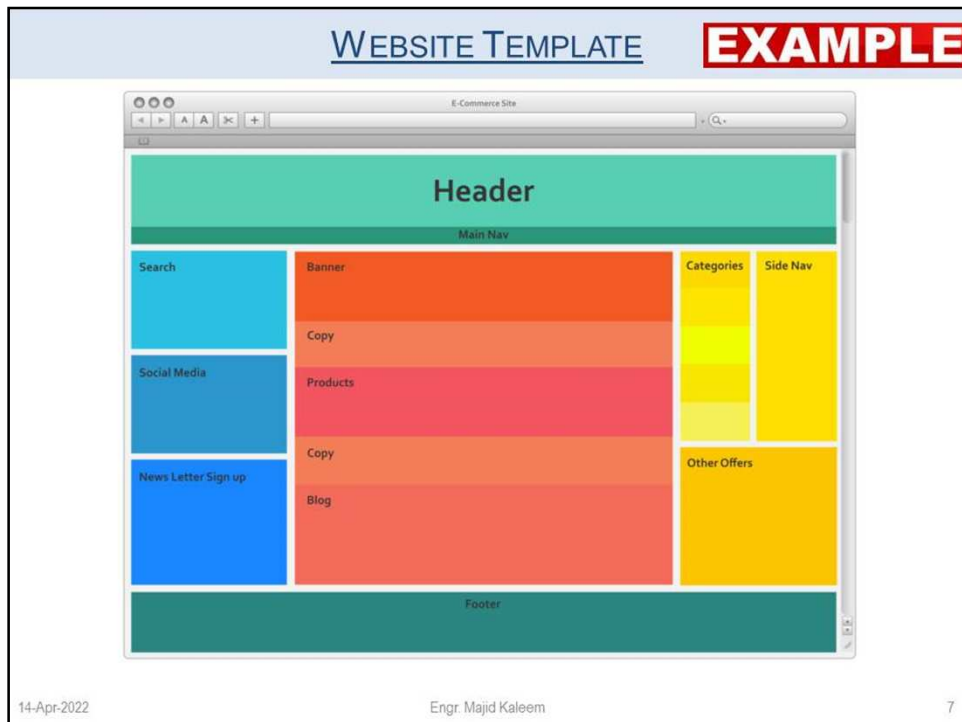
## TEMPLATE?

- Kind of sample/design!


14-Apr-2022

Engr. Majid Kaleem

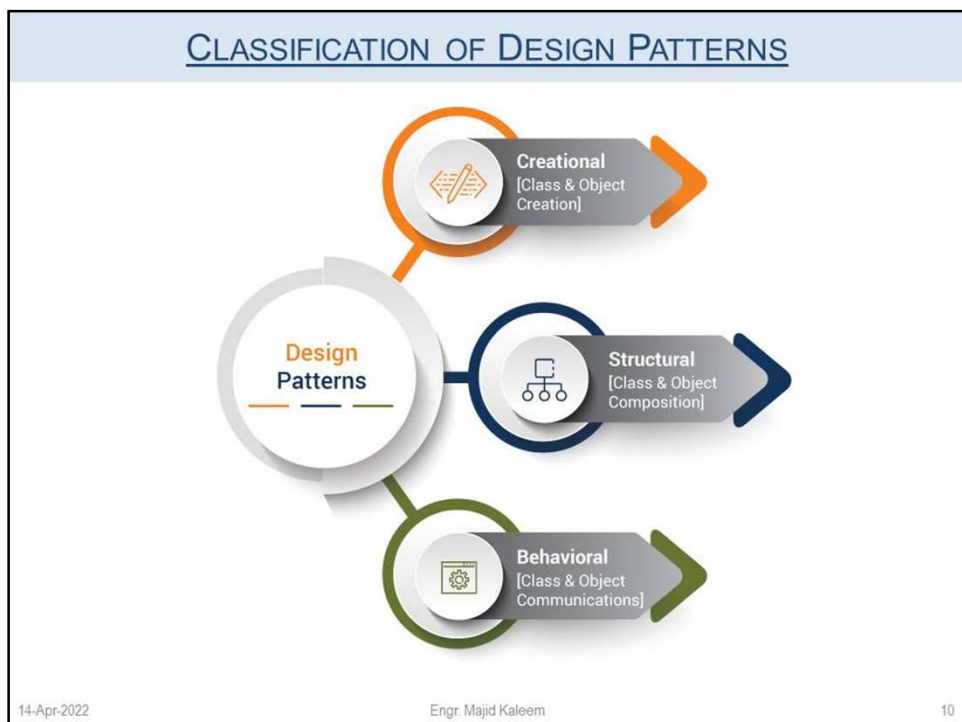
6



UNSTITCHED DRESS**EXAMPLE**

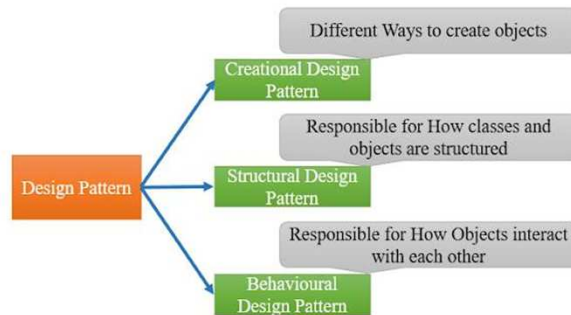


14-Apr-2022Engr. Majid Kaleem9



## CLASSIFICATION OF DESIGN PATTERNS

- They are categorized in three groups: Creational, Structural, and Behavioral as listed below:



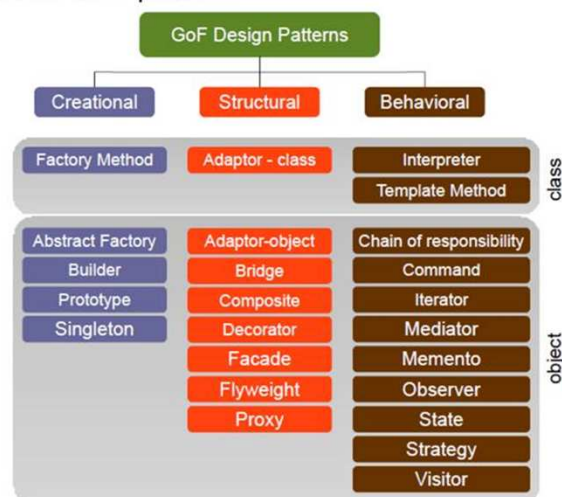
14-Apr-2022

Engr. Majid Kaleem

11

## CLASSIFICATION OF DESIGN PATTERNS

- The 23 Gang of Four (GoF) patterns are generally considered the foundation for all other patterns:



14-Apr-2022

Engr. Majid Kaleem

12



## CLASSIFICATION OF DESIGN PATTERNS

- We classify design patterns by two criteria:
- The first criterion, called *PURPOSE*, reflects what a pattern does.
- Patterns can have either creational, structural, or behavioral purpose.
  - Creational patterns concern the *process of object creation*.
  - Structural patterns deal with the *composition of classes or objects*.
  - Behavioral patterns characterize the ways in which *classes or objects interact and distribute responsibility*.

14-Apr-2022

Engr. Majid Kaleem

13

## CLASSIFICATION OF DESIGN PATTERNS

- The second criterion, called *SCOPE*, specifies whether the pattern applies primarily to classes or to objects.
- *Class patterns* deal with *relationships* between classes and their subclasses.
- These relationships are established through *inheritance*, so they are static—fixed at compile-time.
- *Object patterns* deal with object *relationships*, which can be changed at run-time and are more dynamic.
- Almost all patterns use *inheritance* to some extent.
- So the only patterns labeled "class patterns" are those that focus on class relationships.
- Note that most patterns are in the *Object scope*.

14-Apr-2022

Engr. Majid Kaleem

14

## CLASSIFICATION OF DESIGN PATTERNS

PURPOSE	SCOPE	
	Class	Object
Creational	Creational class patterns defer some part of object creation to subclasses	Creational object patterns defer it to another object
Structural	Structural class patterns use inheritance to compose classes	Structural object patterns describe ways to assemble objects
Behavioral	Behavioral class patterns use inheritance to describe algorithms and flow of control	Behavioral object patterns describe how a group of objects cooperate to perform a task that no single object can carry out alone

14-Apr-2022

Engr. Majid Kaleem

15

## DESCRIBING DESIGN PATTERNS

- We describe design patterns using a consistent format.
- Each pattern is divided into sections according to the following template.
- The template lends a uniform structure to the information, making design patterns easier to learn, compare, and use:
  1. Pattern Name and Classification
  2. Intent
  3. Also Known As
  4. Motivation
  5. Applicability
  6. Structure
  7. Participants
  8. Collaborations
  9. Consequences
  10. Implementation
  11. Sample Code
  12. Known Uses
  13. Related Patterns

14-Apr-2022

Engr. Majid Kaleem

16



## DESCRIBING DESIGN PATTERNS

1. **Pattern Name and Classification** - The pattern's name conveys the essence of the pattern succinctly. A good name is vital, because it will become part of your design vocabulary. The pattern's classification reflects whether they belong to creational, structural or behavioral category..
2. **Intent** - A short statement that answers the following questions: What does the design pattern do? What is its rationale and intent? What particular design issue or problem does it address?
3. **Also Known As** - Other well-known names for the pattern, if any. Motivation - A scenario that illustrates a design problem and how the class and object structures in the pattern solve the problem. The scenario will help you understand the more abstract description of the pattern that follows.
4. **Motivation** - A scenario that illustrates a design problem and how the class and object structures in the pattern solve the problem. The scenario will help you understand the more abstract description of the pattern that follows.

14-Apr-2022

Engr. Majid Kaleem

17

## DESCRIBING DESIGN PATTERNS

5. **Applicability** - What are the situations in which the design pattern can be applied? What are examples of poor designs that the pattern can address? How can you recognize these situations?
6. **Structure** - A graphical representation of the classes in the pattern using a notation based on the Object Modelling Technique (OMT) . We also use interaction diagrams to illustrate sequences of requests and collaborations between objects.
7. **Participants** - The classes and/or objects participating in the design pattern and their responsibilities.
8. **Collaborations** - How the participants collaborate to carry out their responsibilities.
9. **Consequences** - How does the pattern support its objectives? What are the trade-offs and results of using the pattern? What aspect of system structure does it let you vary independently?

14-Apr-2022

Engr. Majid Kaleem

18

## DESCRIBING DESIGN PATTERNS

10. *Implementation* - What pitfalls, hints, or techniques should you be aware of when implementing the pattern? Are there language-specific issues?
11. *Sample Code* - Code fragments that illustrate how you might implement the pattern in C++ or Java (or any other language).
12. *Known Uses* - Examples of the pattern found in real systems. We include at least two examples from different domains.
13. *Related Patterns* - What design patterns are closely related to this one? What are the important differences? With which other patterns should this one be used?

14-Apr-2022

Engr. Majid Kaleem

19

## CREATIONAL DESIGN PATTERNS

- In software engineering, creational design patterns are design patterns that deal with *object creation mechanisms*, trying to create objects in a manner suitable to the situation.
- *The basic form of object creation could result in design problems or added complexity to the design.*
- Creational design patterns solve this problem by somehow controlling this object creation.

14-Apr-2022

Engr. Majid Kaleem

20

## CREATIONAL DESIGN PATTERNS

- These design patterns are all about class *instantiation*.
- This pattern can be further divided into class-creation patterns and object-creational patterns.
- While class-creation patterns use *inheritance* effectively in the instantiation process, object-creation patterns use *delegation* effectively to get the job done.
- *In everyday life, examples of a creational pattern include a bank check that can be instantiated only once, or a web application that dynamically creates a set of controls compatible with a given web browser.*

14-Apr-2022

Engr. Majid Kaleem

21

## CREATIONAL DESIGN PATTERNS

Product	
-code: String -description: String -price: double	Fields
+setCode(String) +getCode(): String +setDescription(String) +getDescription(): String +setPrice(double) +getPrice(): double +getFormattedPrice(): String	

Fields

Methods

```
using System;

public class Product
{
    public Product()
    {
    }
}

Product prdt = new Product();
```



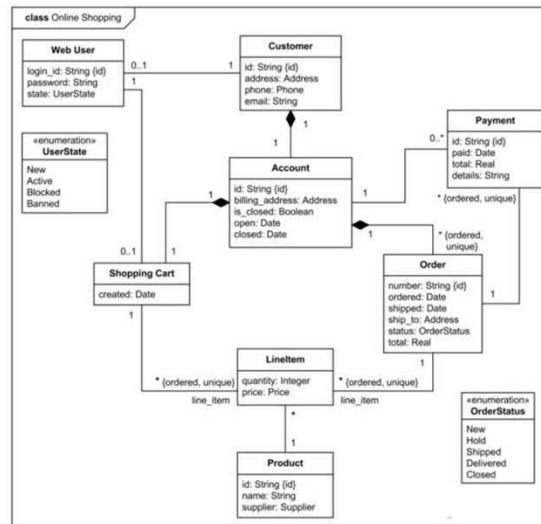
*When object creation is simple, constructor will be sufficient*

14-Apr-2022

Engr. Majid Kaleem

22

## CREATIONAL DESIGN PATTERNS



```

using System;

public class Account
{
    public Account()
    {
    }
}

Account Acct = new Account();

```



*When object creation is complex, constructor alone will not be sufficient*

14-Apr-2022

Engr. Majid Kaleem

23

## USEFUL WEB RESOURCES

1. [https://sourcemaking.com/design\\_patterns](https://sourcemaking.com/design_patterns)
2. <https://refactoring.guru/design-patterns/catalog>

14-Apr-2022

Engr. Majid Kaleem

24

## 1. SINGLETON PATTERN

- **Purpose:**

- There are situations in a project where we want only **one instance** of the object to be created and **shared** between the clients.
- This pattern ensures a class has only one instance and provides a **global (app-level)** point of access to it.
- In other words, a class must ensure that only single instance should be created and single object can be used by all other classes.
- Ensures that at most only one instance of an object exists throughout application and no client can create an instance of the object from outside.
- Moreover, the object should not be created until it is actually needed.

Singleton
-instance
-Singleton() +GetInstance()

**Code & Examples:** <https://www.dofactory.com/net/singleton-design-pattern>

14-Apr-2022

Engr. Majid Kaleem

25

## 1. SINGLETON PATTERN

### Real-Life Example

Suppose you are a member of a sports team and your team is participating in a tournament. When your team plays against another team, as per the rules of the game, the captains of the two sides must have a coin toss. If your team does not have a captain, you need to elect someone to be the captain first. Your team must have one and only one captain.

### Computer World Example

In some software systems, you may decide to maintain only one file system (e.g. NTFS) so that you can use it for the centralized management of resources.

14-Apr-2022

Engr. Majid Kaleem

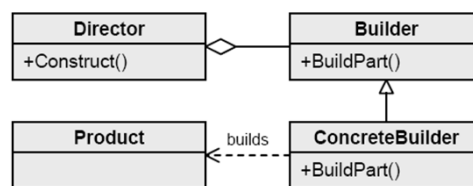
26



## 2. BUILDER PATTERN

- **Purpose:**

- Builder pattern helps us to **separate** the construction of a complex object from its **representation** so that the same construction process can create different representations.
- Builder pattern is useful when the construction of the object is very **complex**.
- The main objective is to separate the construction of objects and their representations.
- If we are able to separate the construction and representation, we can then get **many representations** from the same construction.



**Code & Examples:** <https://www.dofactory.com/net/builder-design-pattern>

14-Apr-2022

Engr. Majid Kaleem

27

## 2. BUILDER PATTERN

### Real-Life Example

This pattern is used by fast food restaurants to construct children's meals. Children's meals typically consist of a main item, a side item, a drink, and a toy (e.g., a burger, fries, coke, and toy car).

Note that there can be variation in the contents of the children's meal, but the construction process is the same.

Whether a customer orders a burger, cheeseburger, or chicken, the process is the same.

The employee at the counter directs the crew to assemble a main item, side item, and toy.

These items are then placed in a bag.

The drink is placed in a cup and remains outside of the bag.

14-Apr-2022

Engr. Majid Kaleem

28



## 2. BUILDER PATTERN

### Real-Life Example

To complete an order for a computer, different hardware parts are assembled based on customer preferences. For example, a customer can opt for a 500GB hard disk with an Intel processor, and another customer can choose a 250GB hard disk with an AMD processor.

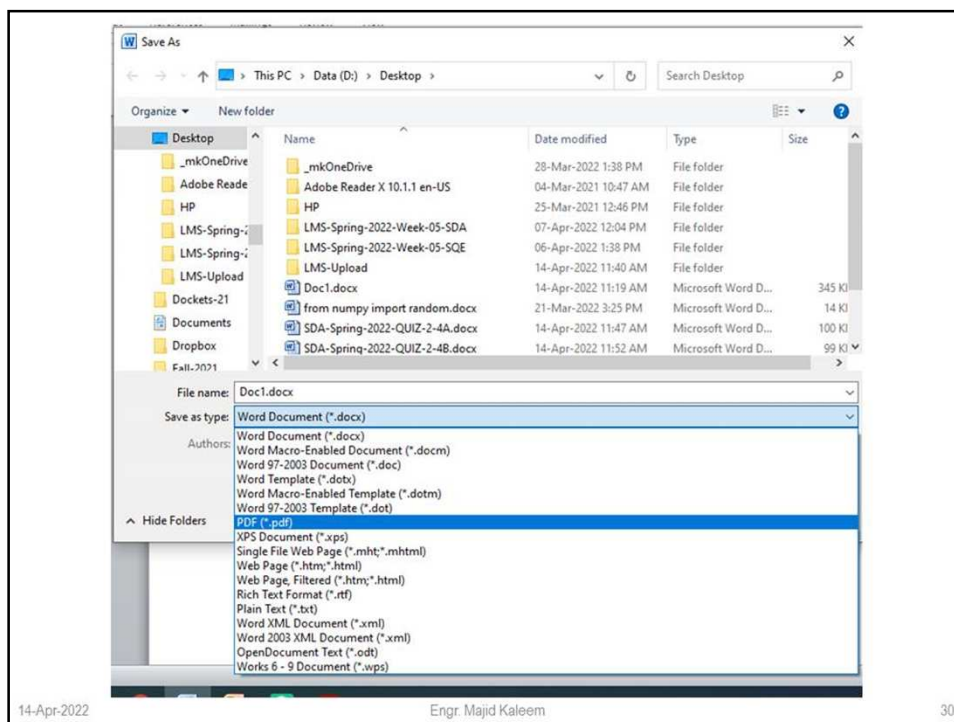
### Computer World Example

You can use this pattern when you want to convert one text format to another text format, such as converting from DOCX to PDF.

14-Apr-2022

Engr. Majid Kaleem

29



14-Apr-2022

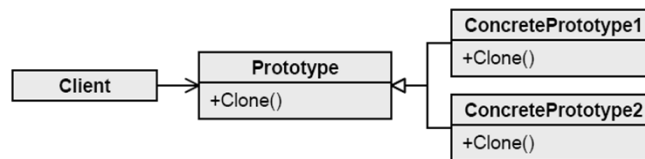
Engr. Majid Kaleem

30

### 3. PROTOTYPE PATTERN

- **Purpose:**

- A fully initialized instance to be copied or cloned instead of **creating new one** and can also be customized as per the requirement.
- It gives us a way to create **new objects** from the **existing instance** of the object.
- In one sentence we **clone** the existing object with its data.
- By cloning any changes to the cloned object does not affect the original object value.



**Code & Examples:** <https://www.dofactory.com/net/prototype-design-pattern>

14-Apr-2022

Engr. Majid Kaleem

31

### 3. PROTOTYPE PATTERN

#### **Real-Life Example**

Suppose you have a master copy of a valuable document. You need to incorporate some change into it to analyze the effect of the change. In this case, you can make a photocopy of the original document and edit the changes in the photocopied document.

#### **Computer World Example**

Let's assume that you already have an application that is stable. In the future, you may want to modify the application with some small changes.

You must start with a copy of your original application, make the changes, and then analyze further.

Surely you do not want to start from scratch to merely make a change; this would cost you time and money.

14-Apr-2022

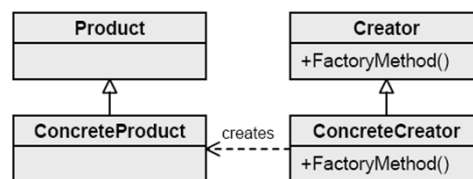
Engr. Majid Kaleem

32

## 4. FACTORY METHOD

- **Purpose:**

- Defines an interface or abstract class for creating an object but let the subclasses decide which class to instantiate.
- In other words, subclasses are responsible to create the instance of the class.
- Factory Method could be used to **create objects** related to a **single family**.
- Creates objects of several related classes without specifying the exact object to be created.
- Factory pattern is meant to **centralize** the creation of objects.



**Code & Examples:** <https://www.dofactory.com/net/factory-method-design-pattern>

14-Apr-2022

Engr. Majid Kaleem

33

## 4. FACTORY METHOD

### Real-Life Example

*In a restaurant, based on customer inputs, a chef varies the taste of dishes to make the final products.*

### Computer World Example

*In an application, you may have different database users. For example, one user may use Oracle, and the other may use SQL Server. Whenever you need to insert data into your database, you need to create either a SqlConnection or an OracleConnection and only then can you proceed. If you put the code into if-else (or switch) statements, you need to repeat a lot of code, which isn't easily maintainable. This is because whenever you need to support a new type of connection, you need to reopen your code and make those modifications. This type of problem can be resolved using the Factory Method pattern.*

14-Apr-2022

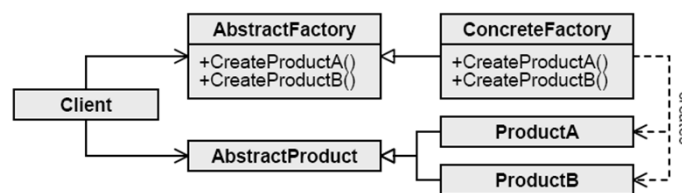
Engr. Majid Kaleem

34

## 5. ABSTRACT FACTORY

- **Purpose:**

- The purpose of the Abstract Factory is to provide an *interface* or *abstract class* for creating families or *sets* of related (or dependent) objects, without specifying concrete sub-classes.
- The Abstract Factory is used when you must coordinate the creation of families or *sets* of objects.
- It's also sometimes called a factory of factories.



**Code & Examples:** <https://www.dofactory.com/net/abstract-factory-design-pattern>

14-Apr-2022

Engr. Majid Kaleem

35

## 5. ABSTRACT FACTORY

### Real-Life Example

This pattern is found in the sheet metal stamping equipment used in the manufacture of automobiles. The stamping equipment is an Abstract Factory which creates automobile body parts.

The same machinery is used to stamp right hand doors, left hand doors, right front fenders, left front fenders, hoods etc. for different models of cars.

Through the use of rollers to change the stamping dies, the concrete classes produced by the machinery can be changed within three minutes.

14-Apr-2022

Engr. Majid Kaleem

36

## 5. ABSTRACT FACTORY

### Real-Life Example

Suppose you are decorating your room with two different types of tables; one is made of wood and one of steel. For the wooden type, you need to visit to a carpenter, and for the other type, you may need to go to a metal shop. All of these are table factories. So, based on demand, you decide what kind of factory you need.

### Computer World Example

When dealing with user interfaces, the system might need to use one set of objects to work on one operating system/browser and another set of objects to work on a different operating system/browser. The Abstract Factory pattern ensures that the system always gets the correct objects for the situation.

14-Apr-2022

Engr. Majid Kaleem

37

```

If(anyQuestions)
{
    askNow();
}
else
{
    thankYou();
    submitAttendance();
    endClass();
}

```

14-Apr-2022

Engr. Majid Kaleem

38



## REFERENCES

1. **Software Architecture**, *Perspectives on an Emerging Discipline* By Mary Shaw & David Garlan
2. **The Art of Software Architecture**, *Design Methods & Techniques* By Stephen T. Albin
3. **Essential Software Architecture**, By Ian Gorton
4. **Microsoft Application Architecture Guide**, By Microsoft
5. **Design Patterns**, *Elements of Reusable Object-Oriented Software* By Erich Gamma, Richard Helm, Ralph Johnson & John Vlissides
6. **Refactoring, Improving the Design of Existing Code**, By Martin Fowler & Kent Beck