

# Unit 4

What is Component Diagram-

Component diagrams are used to visualize the organization of system components and the dependency relationships between them. They provide a high-level view of the components within a system.

The components can be a software component such as a database or user interface; or a hardware component such as a circuit, microchip or device; or a business unit such as supplier, payroll or shipping.

# Component diagram

- Are used in Component-Based-Development to describe systems with Service-Oriented-Architecture
- Show the structure of the code itself
- Can be used to focus on the relationship between components while hiding specification detail
- Help communicate and explain the functions of the system being built to stakeholders

# Component Diagram Symbols

- We have explained below the common component diagram notations that are used to draw a component diagram.

## Component

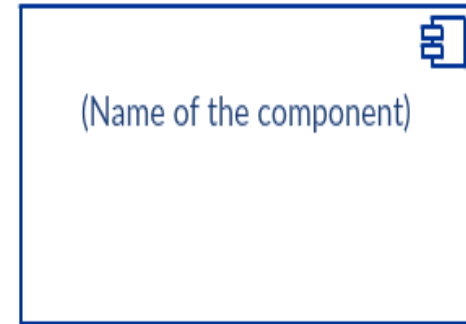
There are three ways the component symbol can be used.

- 1) Rectangle with the component stereotype (the text <<component>>). The component stereotype is usually used above the

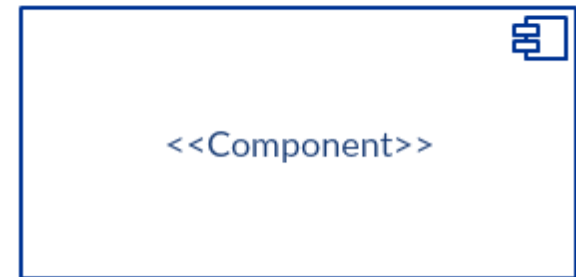


# Component Diagram Symbols

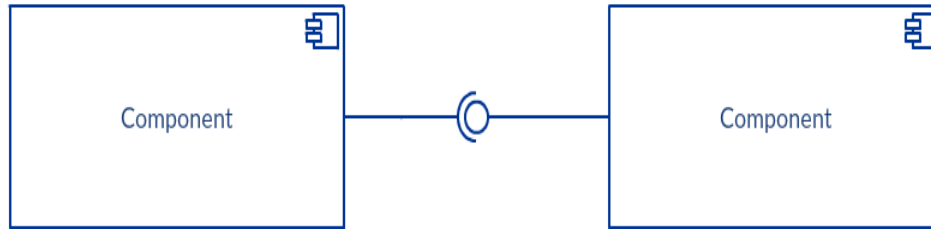
2) Rectangle with the component icon in the top right corner and the name of the component.



3) Rectangle with the component icon and the component stereotype.

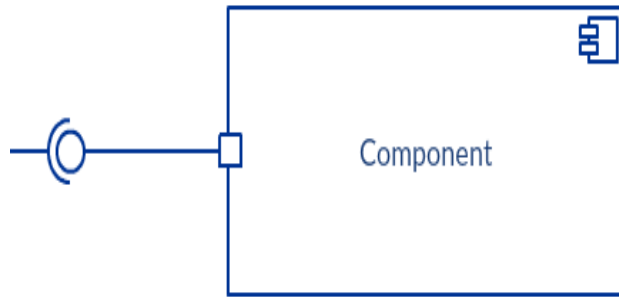


# Provided Interface and the Required Interface



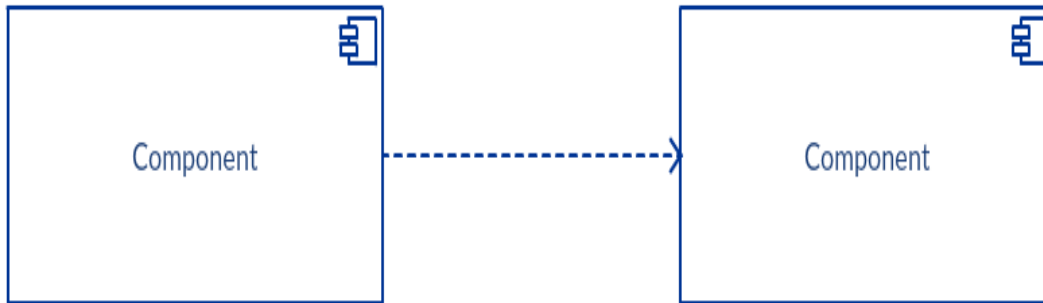
Interfaces in component diagrams show how components are wired together and interact with each other. The assembly connector allows linking the component's required interface (represented with a semi-circle and a solid line) with the provided interface (represented with a circle and solid line) of another component. This shows that one component is providing the service that the other is requiring.

# Port



Port (represented by the small square at the end of a required interface or provided interface) is used when the component delegates the interfaces to an internal class.

# Dependencies



Although you can show more detail about the relationship between two components using the ball-and-socket notation (provided interface and required interface), you can just as well use a dependency arrow to show the relationship between two components.

# How to Draw a Component Diagram

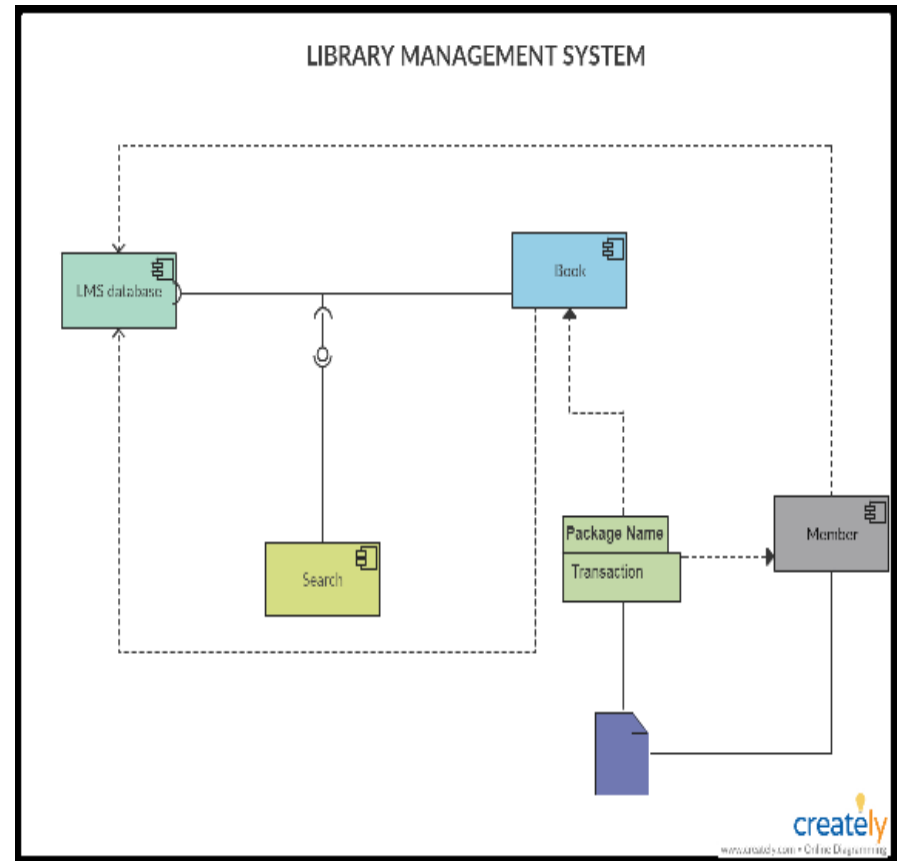
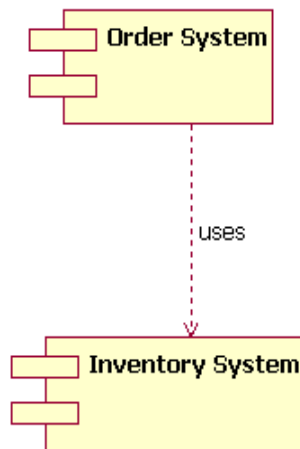
You can use a component diagram when you want to represent your system as components and want to show their interrelationships through interfaces. It helps you get an idea of the implementation of the system. Following are the steps you can follow when drawing a component diagram.

- Step 1: figure out the purpose of the diagram and identify the artifacts such as the files, documents etc. in your system or application that you need to represent in your diagram.
- Step 2: As you figure out the relationships between the elements you identified earlier, create a mental layout of your component diagram
- Step 3: As you draw the diagram, add components first, grouping them within other components as you see fit
- Step 4: Next step is to add other elements such as interfaces, classes, objects, dependencies etc. to your component diagram and complete it.
- Step 5: You can attach notes on different parts of your component diagram to clarify certain details to others.



# Component Diagram Examples

- Component Diagram for Library Management System.



# Deployment Diagram

- Deployment diagrams are used to visualize the topology of the physical components of a system, where the software components are deployed.
- Deployment diagrams are used to describe the static deployment view of a system. Deployment diagrams consist of nodes and their relationships.

# Purpose of Deployment Diagrams

- The term Deployment itself describes the purpose of the diagram. Deployment diagrams are used for describing the hardware components, where software components are deployed. Component diagrams and deployment diagrams are closely related.
- Component diagrams are used to describe the components and deployment diagrams shows how they are deployed in hardware.
- UML is mainly designed to focus on the software artifacts of a system. However, these two diagrams are special diagrams used to focus on software and hardware components.
- Most of the UML diagrams are used to handle logical components but deployment diagrams are made to focus on the hardware topology of a system. Deployment diagrams are used by the system engineers.

The purpose of deployment diagrams can be described as –

- Visualize the hardware topology of a system.
- Describe the hardware components used to deploy software components.

# How to Draw a Deployment Diagram?

Deployment diagram represents the deployment view of a system. It is related to the component diagram because the components are deployed using the deployment diagrams. A deployment diagram consists of nodes. Nodes are nothing but physical hardware used to deploy the application.

Deployment diagrams are useful for system engineers. An efficient deployment diagram is very important as it controls the following parameters –

- Performance
- Scalability

Before drawing a deployment diagram, the following artifacts should be identified –

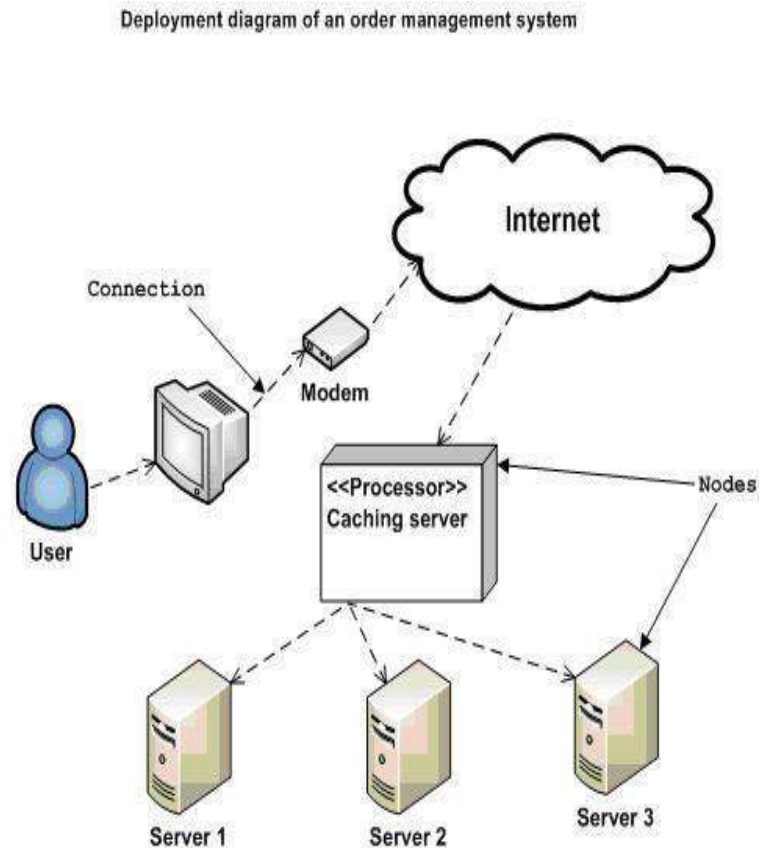
- Nodes
- Relationships among nodes

Following is a sample deployment diagram to provide an idea of the deployment view of order management system. Here, we have shown nodes as –

- Monitor
- Modem
- Caching server
- Server

# Deployment Diagrams

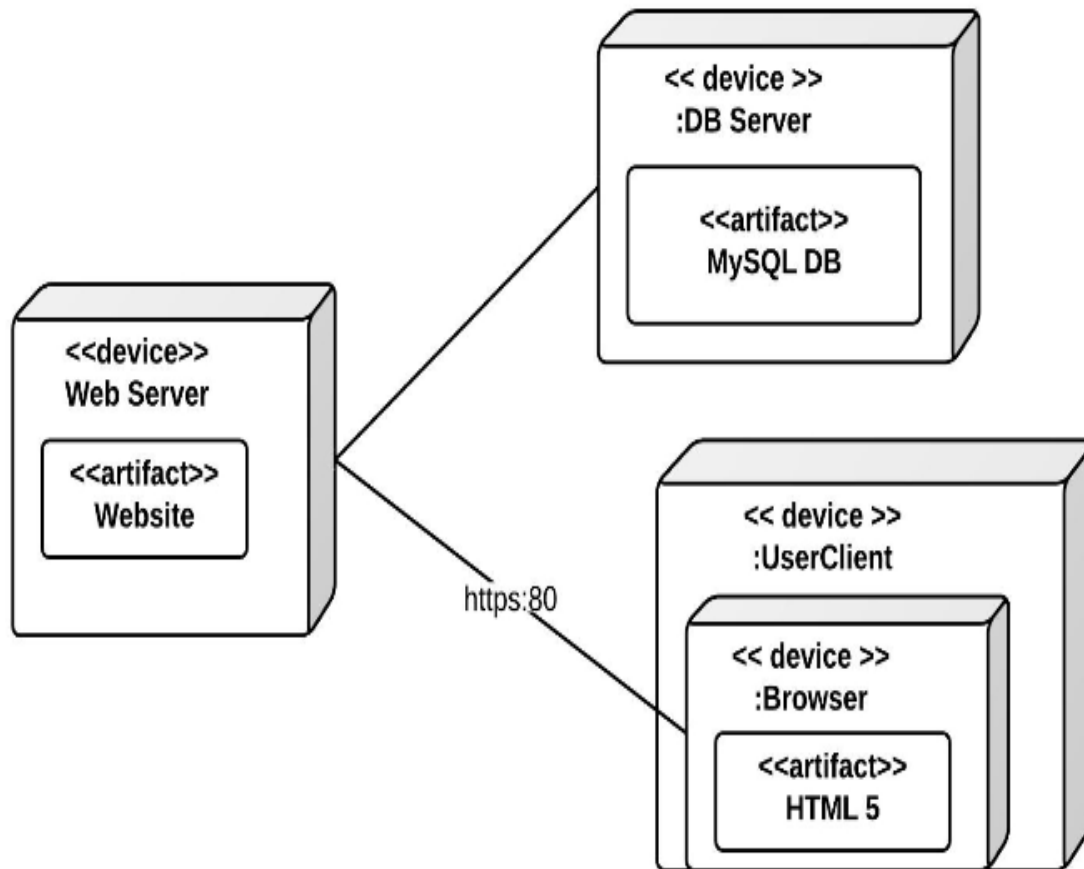
- The application is assumed to be a web-based application, which is deployed in a clustered environment using server 1, server 2, and server 3. The user connects to the application using the Internet. The control flows from the caching server to the clustered environment.
- The following deployment diagram has been drawn considering all the points



# Where to Use Deployment Diagrams?

- Deployment diagrams are mainly used by **system engineers**. These diagrams are used to describe the physical components (hardware), their distribution, and association.
- Deployment diagrams can be visualized as the hardware components/nodes on which the software components reside.
- Software applications are developed to model complex business processes. Efficient software applications are not sufficient to meet the business requirements. Business requirements can be described as the need to support the increasing number of users, quick response time, etc.
- To meet these types of requirements, hardware components should be designed efficiently and in a cost-effective way.
- Now a days software applications are very complex in

# Deployment diagram





# Deployment diagrams can be used

–

- To model the hardware topology of a system.
- To model the embedded system.
- To model the hardware details for a client/server system.
- To model the hardware details of a distributed application.

# What is a package diagram?

- Package diagrams are structural diagrams used to show the organization and arrangement of various model elements in the form of packages. A package is a grouping of related UML elements, such as diagrams, documents, classes, or even other packages. Each element is nested within the package, which is depicted as a file folder within the diagram, then arranged hierarchically within the diagram. Package diagrams are most commonly used to provide a visual

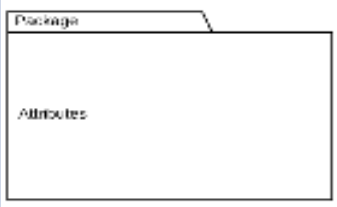

# Benefits of a package diagram

A well-designed package diagram provides numerous benefits to those looking to create a visualization of their UML system or project.

- They provide a clear view of the hierarchical structure of the various UML elements within a given system.
- These diagrams can simplify complex class diagrams into well-ordered visuals.
- They offer valuable high-level visibility into large-scale projects and systems.

# Basic components of a package diagram

- The makeup of a package diagram is relatively simple. Each diagram includes only two symbols:

Symbol Image	Symbol Name	Description
	Package	Groups common elements based on data, behavior, or user interaction
	Dependency	Depicts the relationship between one element (package, named element, etc) and another

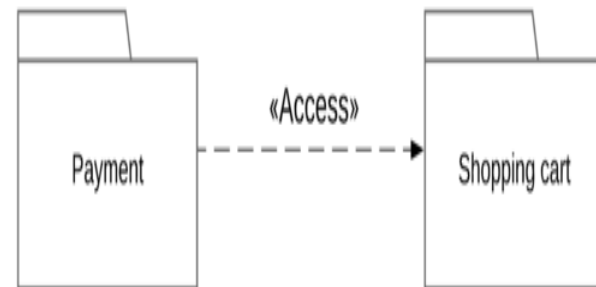
These symbols can be used in a variety of ways to represent different iterations of packages, dependencies, and other elements within a system. Here are the basic components you'll find within a package diagram:

- **Package:** A namespace used to group together logically related elements within a system. Each element contained within the package should be a packageable element and have a unique name.
- **Packageable element:** A named element, possibly owned directly by a package. These can include events, components, use cases, and packages themselves. Packageable elements can also be rendered as a rectangle within a package, labeled with the

# Dependency notations in a package diagram

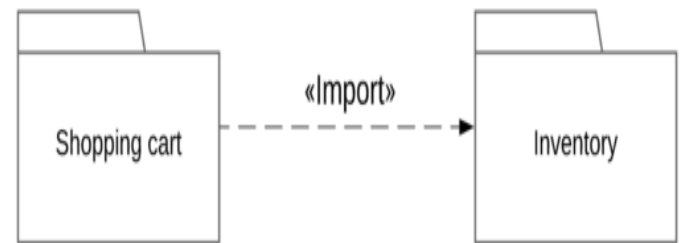
Package diagrams are used, in part, to depict import and access dependencies between packages, classes, components, and other named elements within your system. Each dependency is rendered as a connecting line with an arrow representing the type of relationship between the two or more elements.

There are two main types of dependencies:



# Dependency notations in a package diagram

- Import: Indicates that functionality has been imported from one package to another.
- Example:  
`X<<import>>Y<<import>>Z`. Import is transitive.  
So in the above example, X import from Y and Y imports from Z.  
When we use `<<import>>` it means that packages 'X' can use elements in



# Dependency notations in a package diagram

- Conversely with the example:

$X \ll \text{access} \gg Y \ll \text{access} \gg Z$

Access is non-transitive. So in the above example, X will not be able to access any of the elements in Z.

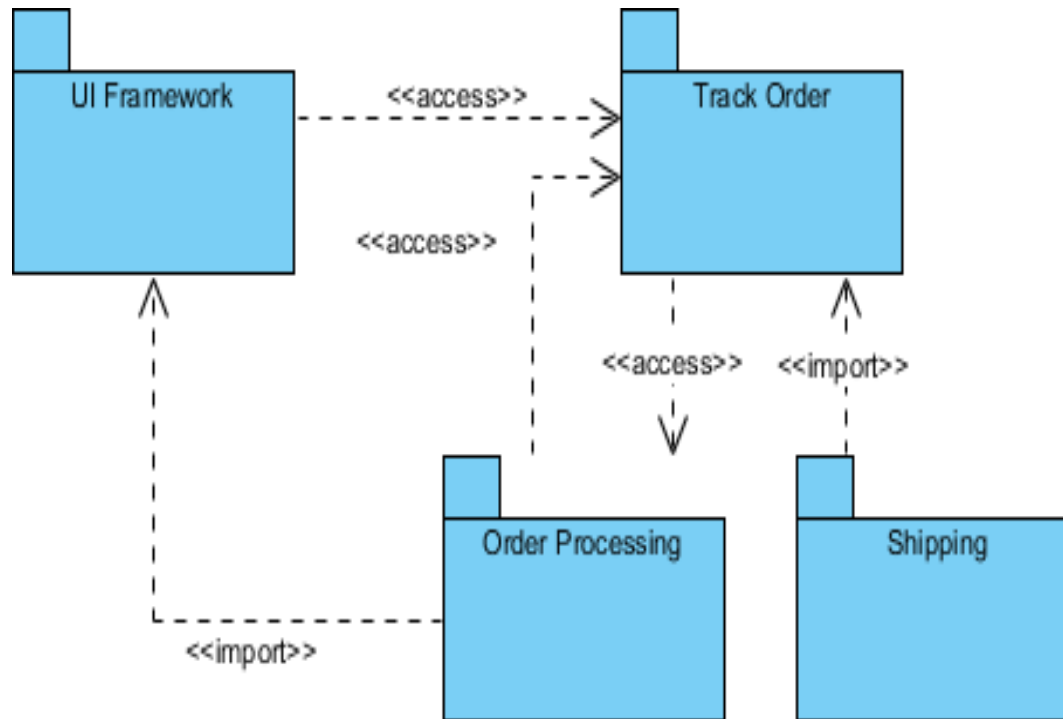


# Using packages with other UML diagrams

- Packages are UML constructs that can be used to organize the elements within any UML classifier in a variety of UML diagrams. Package diagrams are most commonly found used in:
  - **Use-case diagrams:** Each use-case is depicted as an individual package
  - **Class diagrams:** Classes are organized into packages

Packages can also be used within other UML

# Package diagram example



# OMT(object modeling techniques)

- The object modeling techniques is an methodology of object oriented analysis, design and implementation that focuses on creating a model of objects from the real world and then to use this model to develop object-oriented software. object modeling technique, OMT was developed by James Rumbaugh. Now-a-days, OMT is one of the most popular object oriented development techniques. It is primarily used by system and software developers to support full life cycle development while targeting object oriented implementations.
- OMT has proven itself easy to understand, to draw and to use. It is very successful in many application domains: telecommunication, transportation, compilers etc. The popular object modeling

# Phase of OMT

The OMT methodology covers the full software development life cycle. The methodology has the following phase.

- **Analysis** - Analysis is the first phase of OMT methodology. The aim of analysis phase is to build a model of the real world situation to show its important properties and domain. This phase is concerned with preparation of precise and correct modelling of the real world. The analysis phase starts with defining a problem statement which includes a set of goals. This problem statement is then expanded into three models; an object model, a dynamic model and a functional model. The object model shows the static data structure or skeleton of the real world system and divides the whole application into objects. In others words, this model represents the artifacts of the system. The dynamic model represents the interaction between artifacts above designed represented as events, states and transitions. The functional

# Phase of OMT

**System design** - The system design phase comes after the analysis phase. System design phase determines the overall system architecture using subsystems, concurrent tasks and data storage. During system design, the high level structure of the system is designed. The decisions made during system design are:

- The system is organized in to sub-systems which are then allocated to processes and tasks, taking into account concurrency and collaboration.
- Persistent data storage is established along with a strategy to manage shared or global information.
- Boundary situations are checked to help guide trade off priorities.

# Phase of OMT

**Object design** - The object design phase comes after the system design phase is over. Here the implementation plan is developed. Object design is concerned with fully classifying the existing and remaining classes, associations, attributes and operations necessary for implementing a solution to the problem. In object design:

- Operations and data structures are fully defined along with any internal objects needed for implementation.
- Class level associations are determined.

# Phase of OMT

**Implementation** - Implementation phase of the OMT is a matter of translating the design into programming language constructs. It is important to have good software engineering practice so that the design phase is smoothly translated into the implementation phase. Thus while selecting programming language all constructs should be kept in mind for following noteworthy points.

- To increase flexibility.
- To make amendments easily.
- For the design traceability.
- To increase efficiency.

# Design Patterns

Design patterns represent the best practices used by experienced object-oriented software developers. Design patterns are solutions to general problems that software developers faced during software development. These solutions were obtained by trial and error by numerous software developers over quite a substantial period of time.

## **What is Gang of Four (GOF)?**

In 1994, four authors Erich Gamma, Richard Helm, Ralph Johnson und John Vlissides

published a book titled Design Patterns - Elements of Reusable Object-Oriented

Software which initiated the concept of Design Pattern in Software development.



Program to an interface not an implementation

Favor object composition over inheritance

Usage of Design Pattern

Design Patterns have two main usages in software development.

Common platform for developers

Design patterns provide a standard terminology and are specific to particular scenario. For example, a singleton design pattern signifies use of single object so all developers familiar with single design pattern will make use of single object and they can tell each other that program is following a singleton pattern.

### **Best Practices**

Design patterns have been evolved over a long period of time and they provide best solutions to certain problems faced during software development. Learning these patterns helps unexperienced developers to learn software design in an easy and faster way

## Types of Design Pattern

- As per the design pattern reference book Design Patterns - Elements of Reusable ObjectOriented Software , there are 23 design patterns. These patterns can be classified in three
- categories: Creational, Structural and behavioral patterns. We'll also discuss another category
- of design patterns: J2EE design patterns.

S. N.	Pattern & Description
1	<b>Creational Patterns</b> -These design patterns provides way to create objects while hiding the creation logic, rather than instantiating objects directly using new operator. This gives program more flexibility in deciding which objects need to be created for a given use case.
2	<b>Structural Patterns</b> -These design patterns concern class and object composition. Concept of inheritance is used to compose interfaces and define ways to compose objects to obtain new functionalities.
3	<b>Behavioral Patterns</b> -These design patterns are specifically concerned with communication between objects.
4	<b>J2EE Patterns</b> -These design patterns are specifically concerned with the presentation tier. These patterns are identified by Sun Java Center.

# Factory Pattern

Factory pattern is one of the most used design patterns in Java. This type of design pattern

comes under creational pattern as this pattern provides one of the best ways to create an object.

In Factory pattern, we create object without exposing the creation logic to the client and refer

to newly created object using a common interface.

## **Implementation**

We're going to create a Shape interface and concrete classes implementing

the Shape interface. A factory class ShapeFactory is defined as a next step.

FactoryPatternDemo, our demo class will use ShapeFactory to get a Shape object. It will pass

