

# Summary of Creational Patterns

## What is a design Pattern?

A pattern is a named abstraction from a concrete form that represents a recurring solution to a particular problem.

## Why Design pattern?

Design patterns Solve design problems in many different ways.

- Find appropriate objects
- Determine object granularity
- Specify Object Interfaces , etc.

## Elements of Design patterns:

- Name : a handle we can use to describe the problem
- Problem : describes when to apply the pattern
- Solution: describes the elements, relationships, responsibilities, and collaborations.
- Consequences: results and trade-offs of applying the pattern.

## Classification of design Patterns:

### Creational patterns

They abstracts instantiation process

### Structural Patterns:

They are concerned with how classes and objects are composed to form larger structures.

### Behavioral patterns:

They are concerned with algorithms and the assignment of responsibilities between objects

# Creational Patterns

- This Design pattern is all about class instantiation.
- Make system independent of how objects are created composed and represented
- Class-creation patterns:
  - use inheritance effectively in the instantiation process
- Object-creation patterns:
  - use delegation effectively to get the job done

## Abstract Factory Pattern

Provide an interface for creating families of related or dependent objects without specifying their concrete classes.

## Builder Pattern

Separate the construction of a complex object from its representation so that the same construction process can create different representations

## Factory Method Pattern

Define an interface for creating an object, but let subclasses decide which class to instantiate.

## Prototype Pattern

Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype.

## Abstract factory

### Intent:

provide an interface for creating families of related or dependent objects without specifying their concrete classes.

### Applicability:

- A system should be independent of how its products are created, composed, and represented.
- If you want to provide a class library products, and you want to reveal just their interfaces , not their implementations.

## Builder

### Definition:

Builder is an object creational design pattern that codifies the construction process outside of the actual steps that carries out the construction - thus allowing the construction process itself to be reused.”

### Applicability

- Builder patterns is useful in a large system.
- Builder patterns fits when different variations of an object may need to be created and the inheritance into those objects is well-defined.

## Advantages of of Builder

- **Encapsulation:** The builder pattern encapsulates the construction process of a complex object and thereby increases the modularity of an application.
- A builder lets you vary an object's representation without changing the way the object is constructed.
- Give control over the build process.

## Factory Method

Helps to model an interface for creating an object which at creation time can let its subclasses decide which class to instantiate

### **Applicability:**

- A class can't anticipate the class of objects it must create
- A class wants its subclasses to specify the objects it creates.

## **Advantages of Factory Method:**

- The use of factories gives the programmer the opportunity to abstract the specific attributes of an Object into specific subclasses which create them.
- The Factory Pattern promotes loose coupling by eliminating the need to bind application-specific classes into the code.

## **Prototype**

- Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype

### **Applicability:**

- When the class to instantiate are specified at run-time, for example, by dynamic loading; or
- To avoid building a class hierarchy of factories that parallels the class hierarchy of products; or
- When instances of a class can have one of only a few different combinations of state

# Singleton

## Intent:

Ensure a class only has one instance, and provide global point of access to it.

## Applicability:

- There must be exactly one instance of a class, and it must be accessible to clients from a well-known access point.
- When the sole instance should be extensible by subclassing and clients should be able to use an extended instance without modifying their code.

## Comparison of Creational patterns

|                                 | Abstract Factory   | Factory Method   | Builder   | Prototype  | Singleton   |
|---------------------------------|--|--|---|--|---|
| <b>When to use</b>              | Need complete flexibility - new kinds of derived classes could be created later which would be usable by the client  | Need limited flexibility - choose from among a limited set of configurations   | Don't want the details, want a configuration which will enable me to start working on it right away | Don't want the details, don't even want to pick what package I will get - it will probably be passed to me | Want to create only one of these - could be known to me, or could be passed to me   |
| <b>Client knowledge</b>         | Less knowledge of type - client knows base type, not specific class. Knowledge of class flexible, but knowledge of elements of the configuration, it has to create them explicitly | More knowledge - client knows specific type, and knows about the internal configuration - it has to create them explicitly | Less knowledge of details, no knowledge of configuration  | More knowledge of details, some knowledge of configuration - enough to make the right choice               | Orthogonal to how much client has to know, but knowledge limited by the fact that most of the time, it uses an instance that already exists |
| <b>Advantages/Disadvantages</b> | Great flexibility, Low client knowledge  | Reduced flexibility, High client knowledge   | Reduced flexibility, minimal client knowledge   | Good flexibility, minimal client knowledge   | Variable flexibility, minimal client knowledge  |

Thank you.