Factory pattern

Factory: A class sole job is to easily create and return instances of other classes

Motivation Factory:
1 The factory design pattern the most used in modern programming languages like Java and C#
2 creates objects without exposing the instantiation logic to the client.
3 Refers to the newly created object through a common interface

The advantage is obvious:
New shapes can be added without changing a single line of code in the framework there are certain factory implementations that allow adding new products without even modifying the factory class.

Java API Usage:
1 borders (BorderFactory)
2 Key strokes (KeyStroke)
3 network connections(SocketFactory)

Factory Method pattern

Intent: Define an interface for creating an object but let subclasses decide which class to instantiate . Factory Method lets a class defer instantiation to subclasses.

Applicability
Use the Factory Method pattern in any of the following situations:
1 A class can't anticipate the class of object it must create.
2 A class wants its subclasses to specify the objects it creates.

Singleton Pattern

The singleton pattern: is one of the simplest design patterns: it involves only one class which is responsible to instantiate itself, to make sure it creates not more than one instance.

Motivation:
1 Sometimes it's important to have only one instance for a class. For example, in a system there should be only one window manager (or only a file system or only a print spooler).
2 Usually singletons are used for centralized management of internal or external resources and they provide a global point of access to themselves.

Intent:
1 Ensure that only one instance of a class is create.
2 Provide a global point of access to the object.

. Here are some real situations where the singleton is used:
1 Logger Classes
2 Configuration Classes
3 Factories implemented as Singletons

The Builder Design Pattern

When to use:
1 Too Many arguments to pass from client program via the constructor that can be error prone because most of the time, the type of arguments are same and from client side its hard to maintain the order of the argument.
2 Some of the parameters might be optional but using a constructor, we are forced to send all the parameters and optional parameters need to send as NULL.

The Observer Pattern

Intent

Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically

Motivation

The need to maintain consistency between related objects without making classes tightly coupled.

Applicability:

Use the Observer pattern in any of the following situations:

1 When an abstraction has two aspects, one dependent on the other. Encapsulating these aspects in separate objects lets you vary and reuse them independently.
2 When a change to one object requires changing others.
3 When an object should be able to notify other objects without making assumptions about those objects.

Benefits:

1 Minimal coupling between the Subject and the Observer.
2 Can reuse subjects without reusing their observers and vice versa
3 Observers can be added without modifying the subject
4 All subject knows its list of observers
5 Subject does not need to know the concrete class of an observer, just that each observer implements the update interface

Data Bus pattern

Intent

1 Allows send of messages/events between components of an application without them needing to know about each other. They only need to know about the type of the message/event being sent.
2 Observer pattern but supporting many-to-many communication

Applicability

Use Data Bus pattern when

1 you want your components to decide themselves which messages/events they want to receive
2 you want to have many-to-many communication
3 you want your components to know nothing about each other

The Template Pattern

Motivation

1 If we take a look at the dictionary definition of a template we can see that a template is a present format, used as a starting point for a particular application so that the format does not have to be recreated each time it is used.
2 On the same idea is the template method is based. A template method defines an algorithm in a base class using abstract operations that subclasses override to provide concrete behavior.

Intent

1 Define the skeleton of an algorithm in an operation, deferring some steps to subclasses.
2 Template Method lets subclasses redefine certain steps of an algorithm without letting them to change the algorithm's structure.

Applicability & Examples

The Template Method pattern should be used:

1 to implement the invariant parts of an algorithm once and leave it up to subclasses to implement the behavior that can vary.
2 when refactoring is performed and common behavior is identified among classes. A abstract base class containing all the common code (in the template method) should be created to avoid code duplication.

The Strategy Pattern

Motivation

There are common situations when classes differ only in their behavior. For this cases is a good idea to isolate the algorithms in separate classes in order to have the ability to select different algorithms at runtime.

Intent

Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.