Qt 5 Design Patterns

# Software design pattern

* In software engineering, a software design pattern is a general, reusable solution to a commonly occurring problem within a given context in software design.
* It is not a finished design that can be transformed directly into source or machine code. Rather, it is a description or template for how to solve a problem that can be used in many different situations.
* Design patterns are formalized best practices that the programmer can use to solve common problems when designing an application or system.

## Why use Design Patterns?

The Code can be complex, but the pattern should be:

* Reusable
* Well Defined
* Solve a specific issue
* Easy to use
* Proven Code
* Simple

# SOLID

S.O.L.I.D is an acronym for the first five object-oriented design(OOD)\*\* principles\*\* by Robert C. Martin, popularly known as Uncle Bob.

1. Single-Responsibility Principle

* A class should have one and only one reason to change, i.e. it should have a **single responsibility**.

1. Open-Closed Principle

* OCP states that software entities should be open for extension, but closed for modification
* This simply means that a class should be easily extendable without modifying the class itself.

1. Liskov substitution principle

* Objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program.
* The principle defines that objects of a superclass shall be replaceable with objects of its subclasses **without breaking** the application.

1. Interface segregation principle

* Many client-specific interfaces are better than one general-purpose interface.
* A client should never be forced to implement an interface that it doesn’t use, or clients shouldn’t be forced to depend on methods they do not use.
* it simply means that larger interfaces should be split into smaller ones. By doing so, we can ensure that implementing classes only need to be concerned about the methods that are of interest to them.

1. Dependency inversion principle

* High-level modules/classes should not depend on low-level modules/classes. Both should depend upon abstractions.
* Abstractions should not depend on details. Details should depend on abstractions.

<https://www.digitalocean.com/community/conceptual_articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design>