Principles

Don't Repeat Yourself
Separate what varies from what stays the same
Favor composition over inheritance
Program to an Interface, not an Implementation
(Strive for loosely-coupled designs between objects that interact.)
Open-Closed Principle
RAII: Resource Acquisition Is Initialization
DIP: Dependency Inversion Principle

Patterns

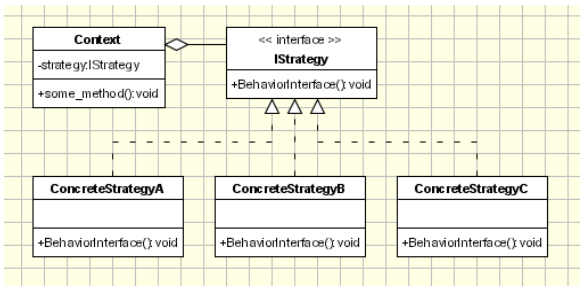
Strategy
Observer
Decorator
Simple Factory
Factory Method
Abstract Factory
(Builder)
Singleton
Double-Checked Locking
Null Object

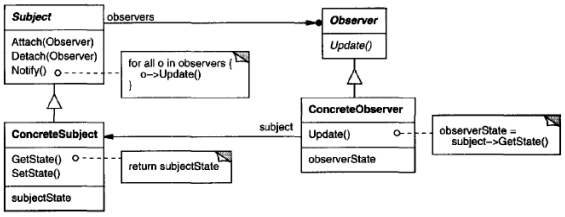
SOLID

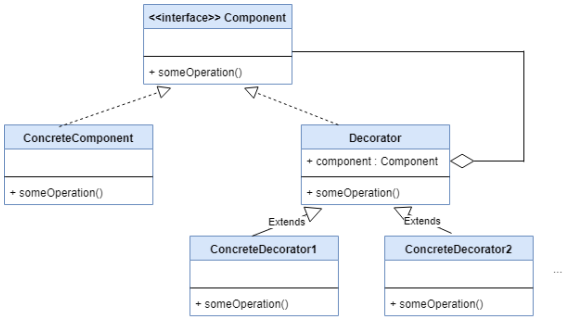
**Single Responsibility Principle (SRP)**: A class should have one, and only one, reason to change, meaning it should only have one responsibility.
**Open/Closed Principle (OCP)**: Software entities should be open for extension but closed for modification.
**Liskov Substitution Principle (LSP)**: Subtypes should be substitutable for their base types without altering the correctness of the program.
**Interface Segregation Principle (ISP)**: Clients should not be forced to depend on interfaces they do not use, favoring smaller, more specific interfaces.
**Dependency Inversion Principle (DIP)**: High-level modules should depend on abstractions, not concrete implementations.

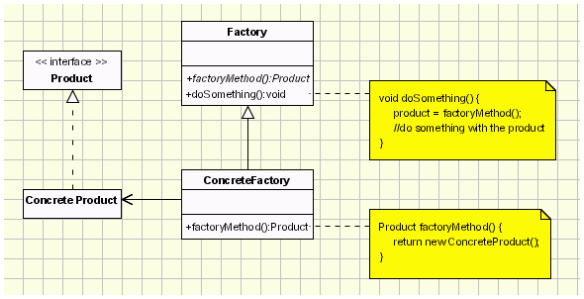
UML

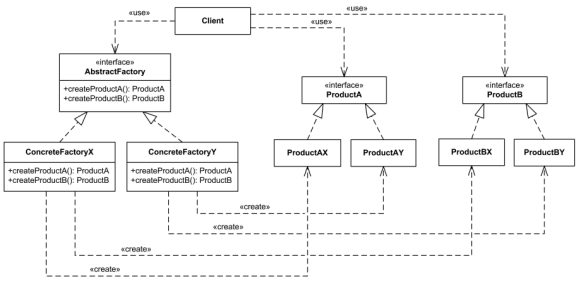
Triangle means inheritance
Triangle should be open
Technically, use dotted lines for "Implements an Interface", but I'm not hung up about it.
Diamond means "contains"
Diamond next to the container box
Preferred: filled in
Arrow means "uses"
Best if "uses" lines include text about the nature of the relationship
Inherited classes BELOW base class
Class boxes have:
Name
Data members
Method names

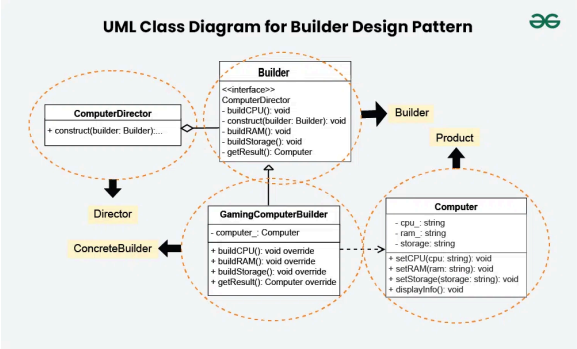


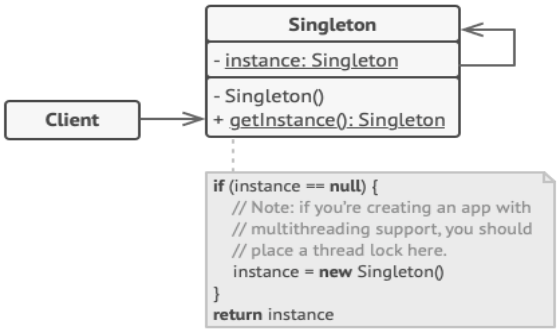



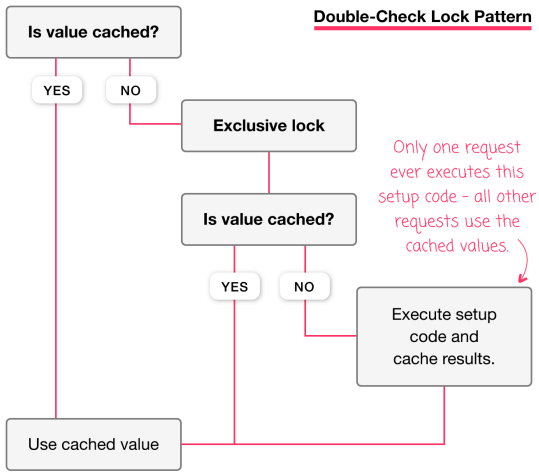



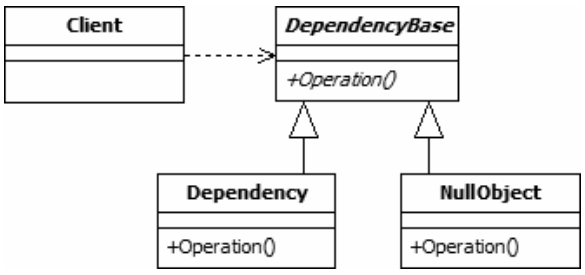






Examples

our application has some data that needs to be refreshed at
irregular intervals. You are already set up as an Observer, but when
notified, you don't want to refresh right then, but wait until someone
uses the data. So you just set a "needs update" flag.
Every time you use the data, you check this flag.
The program is multithreaded, and you don't want two refreshes to
happen at once.
Performance is important
(Hint: it's already a global flag, so Singleton isn't the answer...)
20
Double-checked locking
Note: the outer check is for performance:
to avoid locking if you don't have to.
The inner check is the one for integrity
Use where the outer check is done frequently, but locking is necessary infrequently.

Notes about Strategy:
Basically never has any data in the Strategy object itself
Generally only one method: in some cases, may have two or
more closely related methods.
The Decorator Program used predicates:
Predicates may be
Strategies.

What are the two key properties of Singleton? - Global Access, Single Instance

To implement the Null Object pattern, create a class that implements the same interface or inherits from the same base class as the object it substitutes, but with methods that perform no meaningful actions (e.g., no-ops). This avoids the need for null checks, as the null object can be used in place of real instances while adhering to the same behavior contract without side effects.