Design Patterns

1. Pattern Basics

Patterns are reusable solutions to common design problems. Help with communication, reuse, and design thinking. Shared design vocabulary. Reuse of proven solutions. Aid learning, communication, and refactoring. Improve design thinking without replacing it.

---

2. Types of Patterns

- **Architectural patterns**: fundamental structural schema for software systems. Provides a set of predefined subsystems, specifies their responsibilities, and includes rules and guidelines for organizing the relationships between them.
- **Design patterns (GOF)**: Scheme for refining the subsystems or components of a software system, or the relationships between them. Describes commonly recurring structure of communicating components that solves a general design problem within a particular context
    - **Creational**: object creation strategies (e.g., Factory, Singleton).
    - **Structural**: class composition (e.g., Composite, Bridge, Façade).
    - **Behavioral**: interaction/behavior logic (e.g., Observer, Strategy, Template Method).
- **Idioms**: language-specific patterns (e.g., Singleton in Java).
- **GRASP**: responsibility assignment in OOP (from Craig Larman).
    - **Expert**: assign responsibility to class with necessary data.
    - **Creator**: assign creation to class that aggregates or uses the object.
    - **High Cohesion**: keep classes focused.
    - **Low Coupling**: reduce dependency between classes.
    - **Controller**: use a controller class to handle system events.

---

3. Key Patterns

Singleton

One global instance, access via static method. Used for config, logging, shared resources.

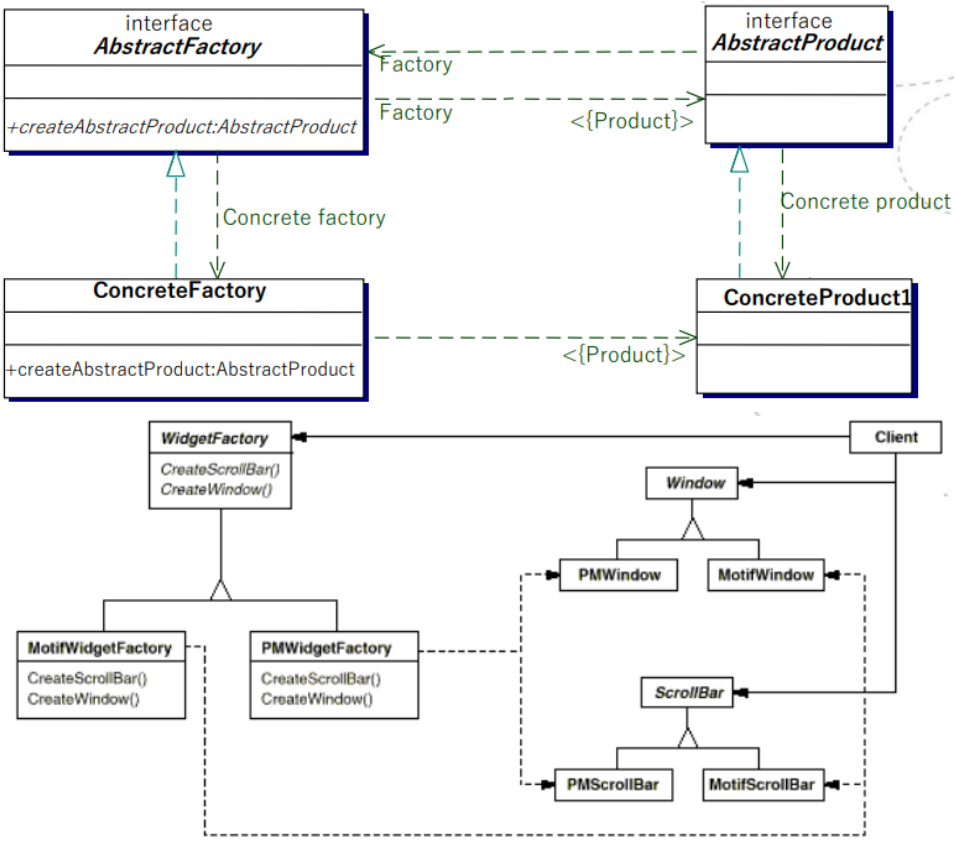
- Risk: can become a hidden global variable, hard to test.
- Usability: The Abstract Factory, Builder, and Prototype patterns can use Singletons in their implementation. Facade objects are often Singletons because only one Facade object is required. State objects are often Singletons.

- Examples: logging service, printer spooler, license key manager…

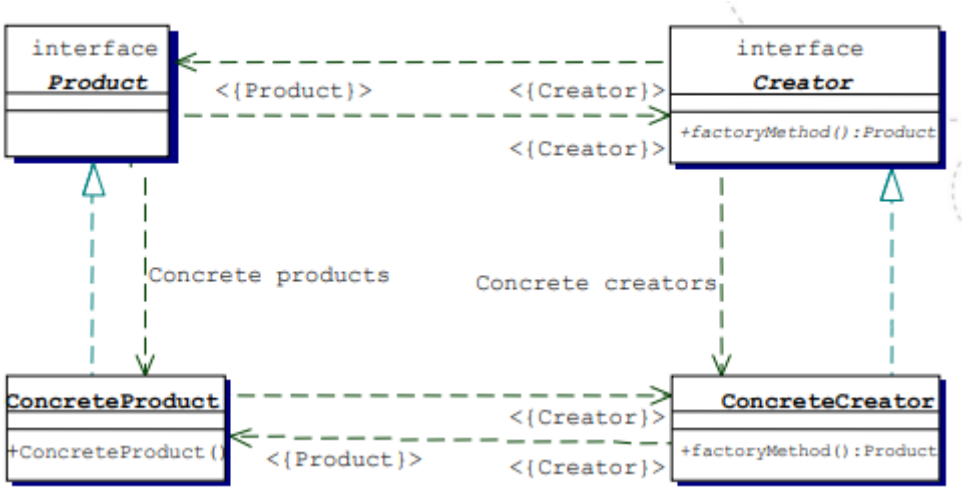
Factory Method / Abstract Factory

Factory Pattern: creational design pattern that provides an interface for creating objects without specifying their exact clases. Delegate instantiation logic to subclasses.

- Abstract Factory: a family of related classes can have different implementation details.
    - Problem: the client should not know anything about which variant they are using
    - Examples: GUI themes, database connectors, cross-platform application components…



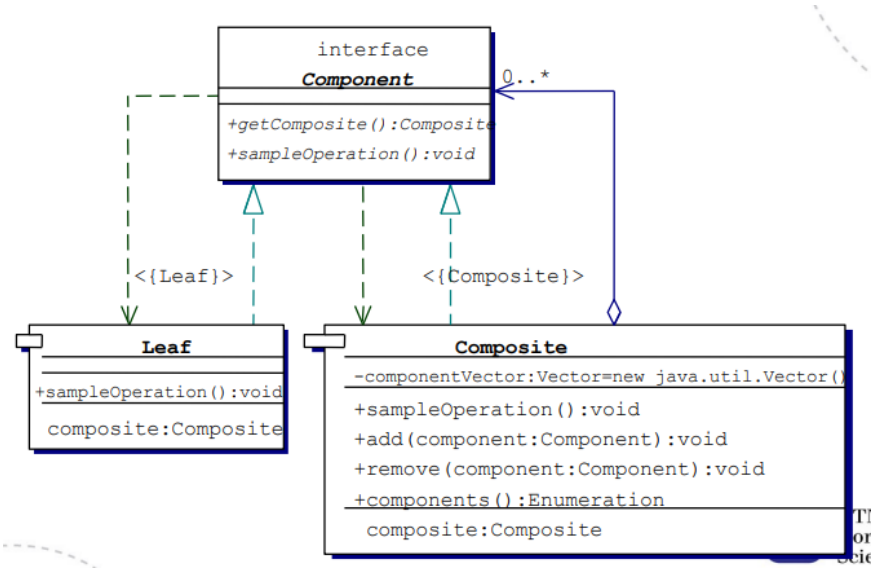
- Factory Method: provide a method for creation at the interface level. Defer the actual creation responsibility to subclasses. Common in toolkits and frameworks.
    - Examples: GUI toolkit, document editors for different types of documents, game engine creating enemy types based on level…

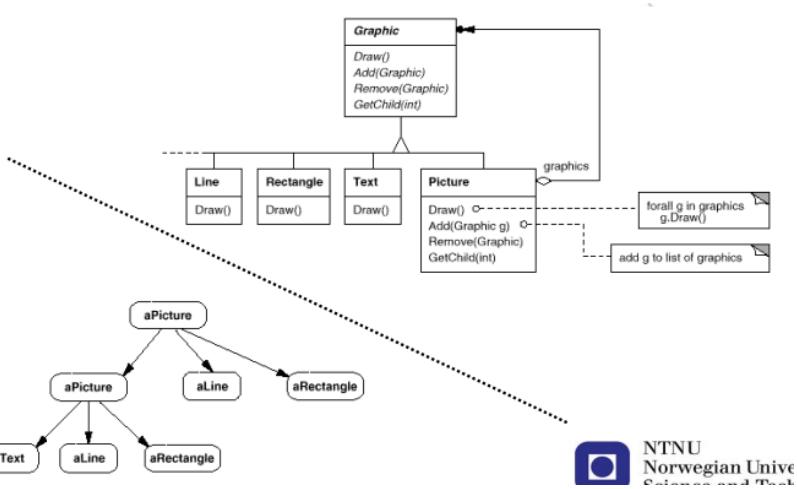


Composite

Clients often need to interact with both individual and groups of elements in the same way, so it defines a common interface for individual (leafs) and groups (composites). Each composite holds a collection of components (which can be other composites or leaves) and forwards method calls to its children.

- Examples: Used for trees, UI hierarchies, file systems, graphic editors (shapes)...

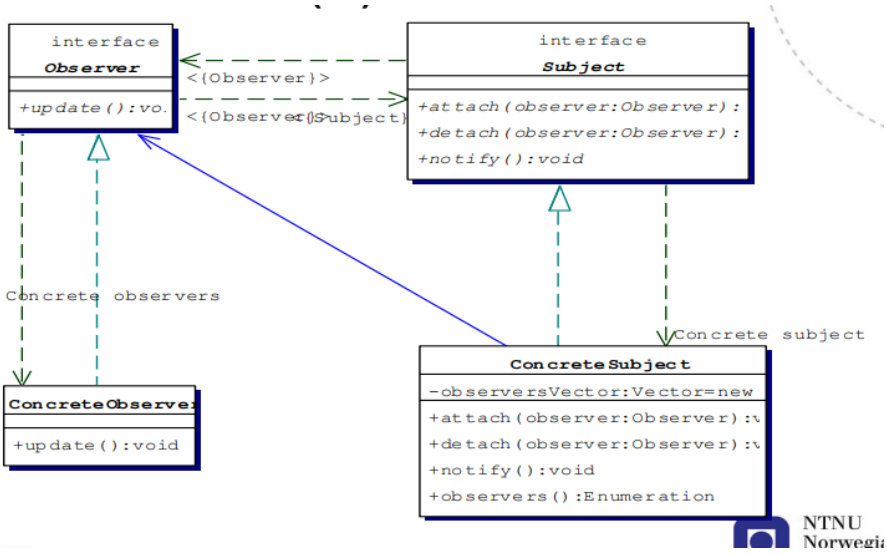




Observer

One-to-many notification: subject informs observers of state changes. This leads to high coupling and sometimes you don't know in advance how many elements you might need to notify.

- Examples: GUI frameworks, real-time updates, event systems, chat applications…



Strategy

Swap algorithms at runtime. Encapsulate behavior into different classes, making them interchangeable, good for testing and flexibility. Define common interface for all strategies and then separate classes into different versions of the behavior. Changes entire algorithm via composition at runtime.

- Examples: sorting algorithms, game AI, compression formats, payment processing…

Template Method

Algorithm skeleton in a superclass. Subclasses override specific steps. Changes specific steps via inheritance at compile time.

- Examples: exporting documents, online order process, game levels, report generation…

---

4. Pattern Languages

Some patterns are applied in sequence. A **pattern language** = a collection of related patterns used to build a system architecture. Helps link low-level and high-level design.

---

5. Design Context

Programming language impacts how patterns are applied. Use language features (e.g., inheritance, interfaces) to implement patterns naturally. A design can’t be performed independently of the technology.