1. Strategy pattern + Template Method pattern

Strategy pattern + Template method

Behavioural patterns.
Seperating algorithm from detail.
Extends behaviour of a system.
Runtime vs. compile time.

Strategy Pattern

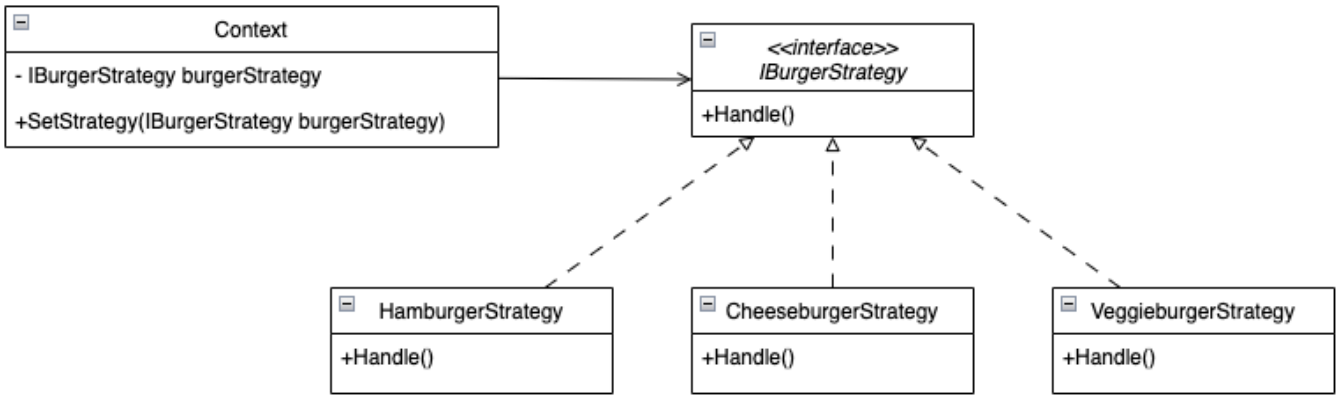
Family of algorithms.
Composition.
Run time → Dynamic Polymorphism.

Definition

> **Strategy pattern defines a family of algorithms and make them interchangeable. Since each algorithm is encapsulated, the client can use different algorithms easily.**

Behavior of the context defined at runtime by delegating it to another object.

Example - Burger



Chose burger at runtime via composition / delegation.

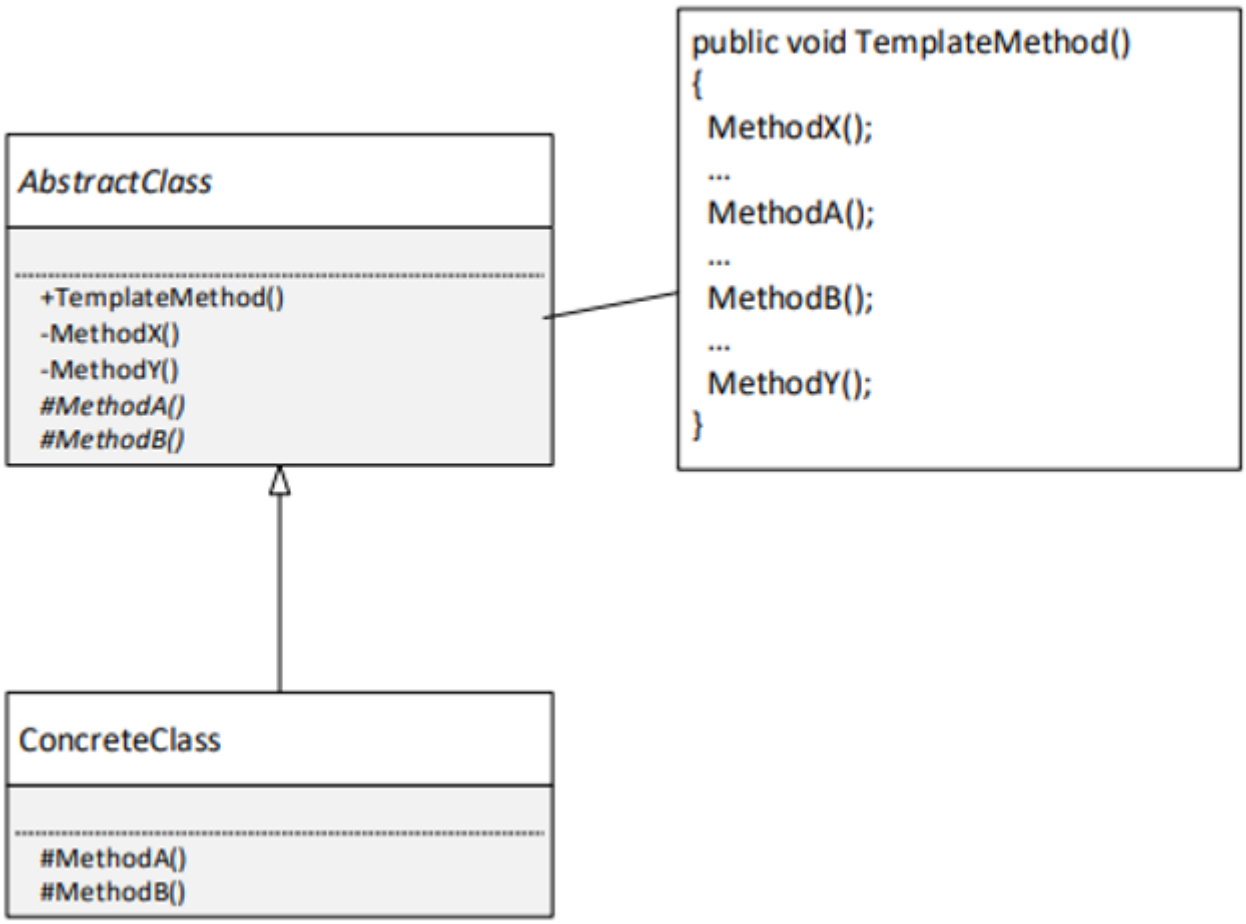
Template method

Skeleton which can be overridden.
Compile time → Static polymorphism
Inheritance.

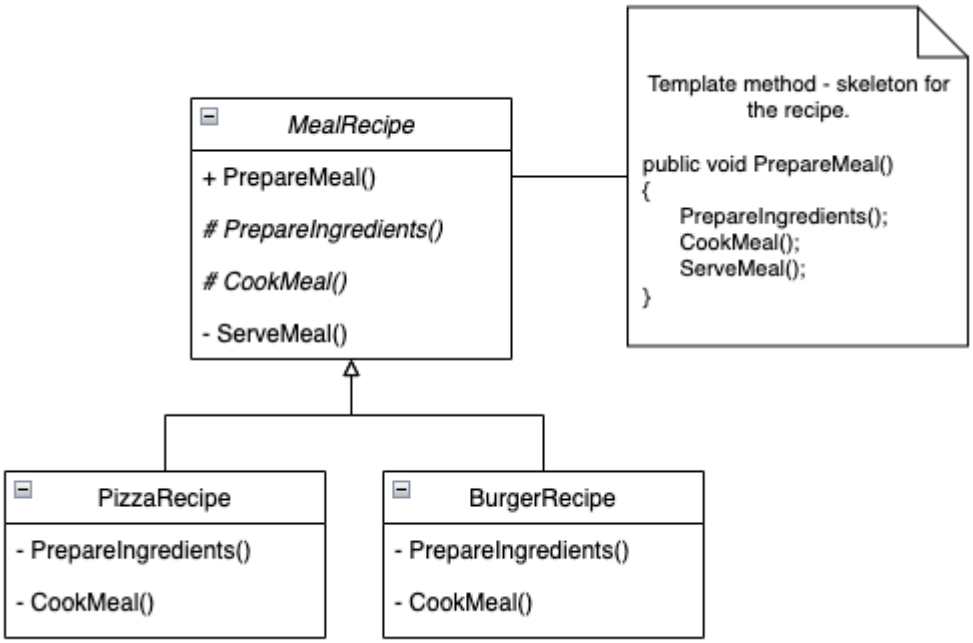
Definition

> **The Template Method Pattern defines the skeleton of an algorithm in a method, deferring some steps to subclasses. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure.**

Diagrams



Example - MealRecipe



Strategy vs. Template

Template → inheritance: A subclass inherits and modifies methods from an abstract.
Strategy → delegation: An object delegates an action to another object.

Template → consistent by using a common structure in the base class
Strategy → easy to switch strategies at runtime because the pattern is based on composition.

SOLID

S Both patterns support **SRP**, as they have one single responsiblity per class / interface
O Both patterns support **OCP**, as it is possible to add more classes that implement the interfaces, without altering the exisiting code.
L Bot patterns adhere to **LSP**, as they ensure that implemend or subclasses can be used interchangeably without affecting the correctness of the program.
I Template method provides a form of segegration with abstract classes. Each concretestrategy defines a seperate interface.
D The classes depend on abstract entities instead of concrete ones.

Comparison

Decorator pattern
State pattern