

Design Patterns

I can stand brute force, but brute
reason is quite unbearable
– Oscar Wilde

4-Feb-20 10:55 PM

1

Contents

Introduction

Strategy DP, Observer DP, Decorator DP

Factory and Abstract Factory DP

Singleton DP, Command DP, Adapter DP

Facade DP, Template DP, Iterator DP, Composite DP

State DP, Proxy DP, Bridge DP, Builder DP,

Chain of Responsibility DP, Flyweight DP

Interpreter DP, Mediator DP, Memento DP, Prototype DP

Visitor DP, Miscellaneous Items

Introduction

Experience is a hard teacher, because
she give the test first, the lesson
afterwards. – Chinese Proverb

Introduction

- Coding is easy if a clear design is given.
- Good OO design is difficult
- Program Designing is an art to a large extent.

4-Feb-20 10:55 PM

3

Reusable, flexible, maintainable OO design is difficult.
It is almost impossible to get it right in the first time.

Design is more than using interfaces, inheritance, polymorphism, composition.

By just knowing OO basics, we cannot become good OO designers.

What is a Design Pattern?

- A design pattern (DP) is an especially clever and insightful way of solving a particular class of problems
 - A pattern is primarily a way to chunk up advice about a topic.
- Definition of a Design Pattern
 - A description of communicating objects and classes that are customized to solve a general design problem in a particular context.

4-Feb-20 10:55 PM

4

DPs creates a design that is more reusable, flexible and maintainable.

We start thinking in a higher level of abstraction.

DPs can document the architecture of a system and enhance its understanding.

We can communicate better with fewer words.

Improvement in productivity.

Someone has already solved a similar problem.

Design patterns help us learn from others' successes instead of our own failures

Typical IT budget

This is the problem



New Projects
<20%



Maintenance and Operations
>80%

Most organizations don't scrutinize M&O expenditures
No other major expenditure has so little oversight

Source Burton group. Infoq presentation on SOA.

Rule of Three

- Once is an event, twice is an incident, thrice is a pattern.
 - Design patterns are discovered not invented.

4-Feb-20 10:55 PM

6

Design patterns are recurring solutions to design problems we see over and over again.

Design patterns emerge from practice, not from theory.

What is a Design Pattern?

- DPs are proven OO designs.
 - They are flexible, elegant and reusable.
 - They are language independent.
 - They do not give us the code, they give us general solutions to a design problem.

4-Feb-20 10:55 PM

7

OOP gives us the tools to cope with change:

Encapsulation

Inheritance

Polymorphism

Having tools is good. Using them well is better. Welcome to Design Patterns.

Frameworks and Libraries

- Design Patterns are different.

4-Feb-20 10:55 PM

8

Framework is a set of co-operating classes, which together make up reusable design, for a specific kind of software. The Framework provides an architectural guidance by structuring the system in abstract classes, as well their responsibilities and interactions. The framework user adapts the framework to his specific needs by deriving concrete classes or by assembling classes provided by the framework.

How are design patterns different from frameworks or libraries?

Design patterns are more abstract than frameworks or libraries.

Design patterns are smaller architectural elements than frameworks / libraries.

Frameworks and libraries usually use design patterns.

We generally call the library code. A framework generally calls our code.

Design patterns focus more on reuse of recurring architectural design themes, while frameworks focus on detailed design and implementation.

Examples of Design Patterns

- Encapsulation
- Inheritance
- Exceptions

4-Feb-20 10:55 PM

9

Encapsulation: Problem description:

Exposed fields can be directly manipulated from outside, leading to violations of the representation invariant or undesirable dependencies that prevent changing the implementation.

Solution: Hide some components, permitting only stylized access to object.

Disadvantages: Indirection may reduce performance.

Sub-Classing: Problem: Similar abstractions have similar members (fields and methods). Repeating these is tedious, error-prone and a maintenance headache.

Solution: Inherit default members from a super class; select the correct implementation via run-time dispatching.

Disadvantages: Code for a class is spread out, potentially reducing understandability. Virtual functions introduce runtime overheads.

Exceptions: Problem: Errors occurring in one part of code should be handled elsewhere. Code should not be cluttered up with error-handling code nor return values preempted by error-codes.

Solution: Introduce language structures for throwing and catching exceptions.

Disadvantages:

Code may still be cluttered (for checked exceptions).

It is hard to know where an exception will be handled.

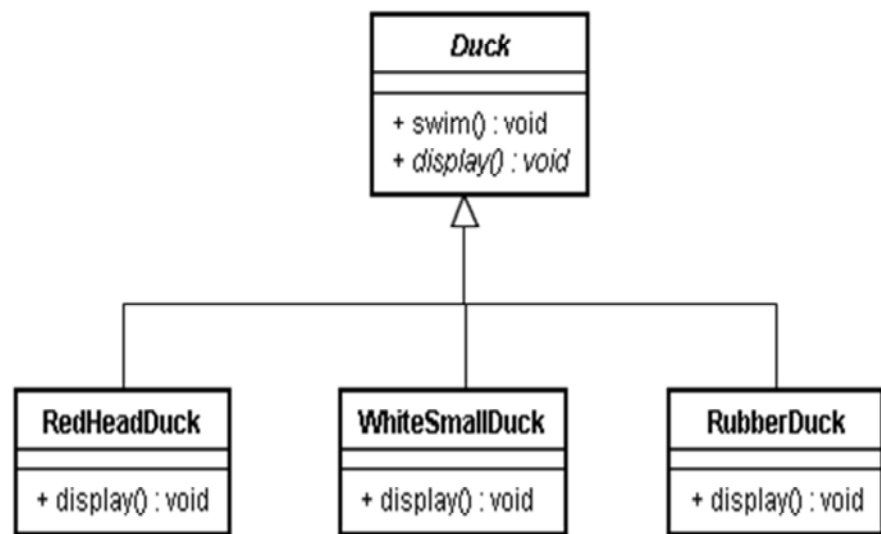
Programmers may misuse the feature for jumping from location to another. This will be confusing and inefficient.

Strategy DP

The most powerful designs are always the
result of a continuous process of
simplification and refinement

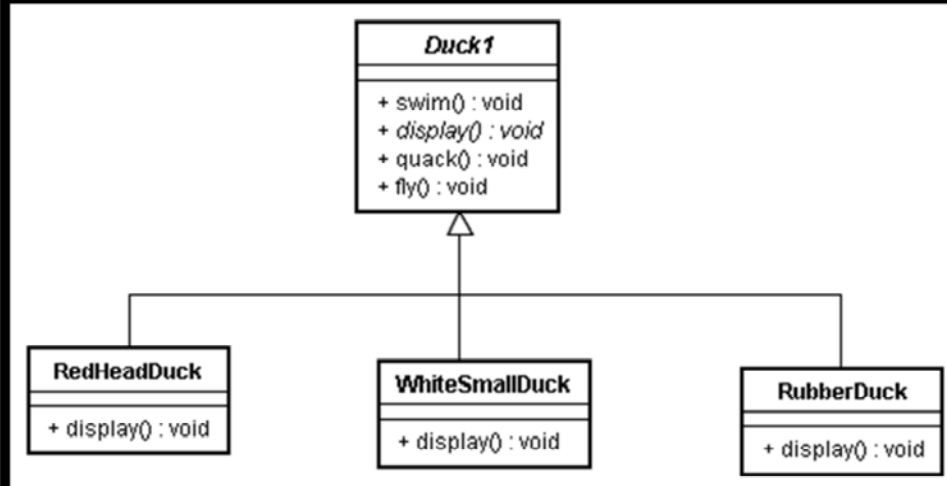
- Kevin Mullet

Example – Initial coding

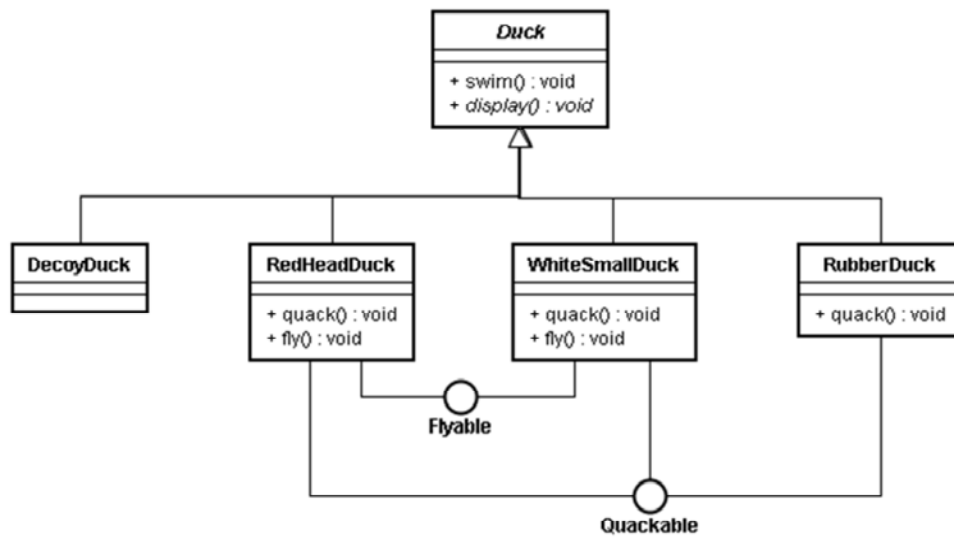


Example – After modifications

- Quack and fly operations get added to all the ducks.



Example – After Improvement



Design Principle

- Principle
 - An aspect of the application can have two parts
 - A part that varies
 - A part that stays constant.
 - Both these parts must be separate

4-Feb-20 10:55 PM

14

Result of the principle

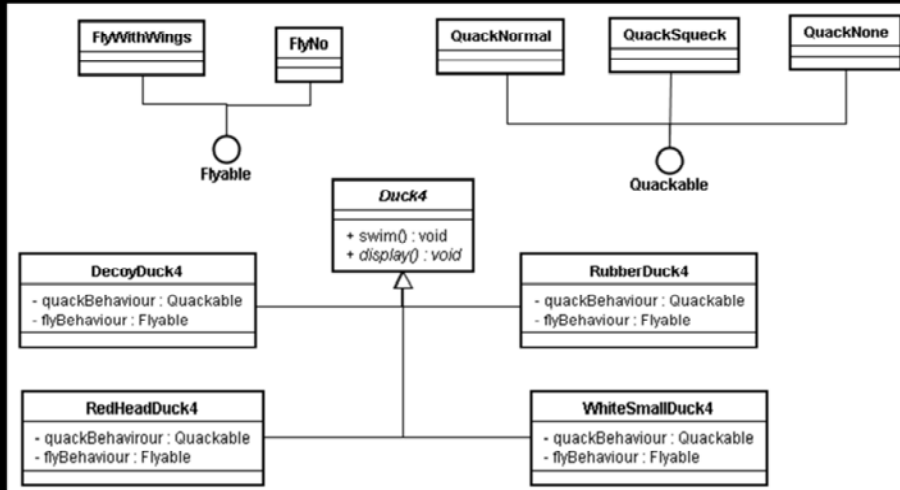
Cheaper to maintain.

Fewer unintended consequences

More flexibility.

Easier to understand

Example – After Improvement



4-Feb-20 10:55 PM

15

The strategy pattern

defines a family of algorithms, encapsulates each one and makes them interchangeable.

lets the algorithm vary independently from the clients that use it.

Code example in Java, C# and cpp available.

Design Principle

- Principle: Program to an Interface, not an implementation

4-Feb-20 10:55 PM

16

Meaning of principle

Interface here could also be an abstract class.

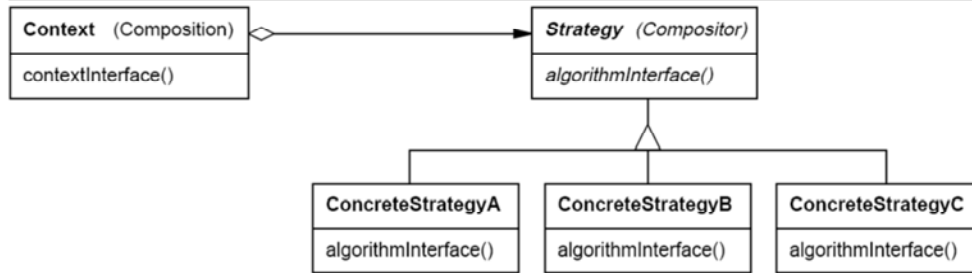
Instead of

```
dog d1 = new dog();  
d1.bark();  
use  
animal a1 = new dog();  
a1.makeSound();
```

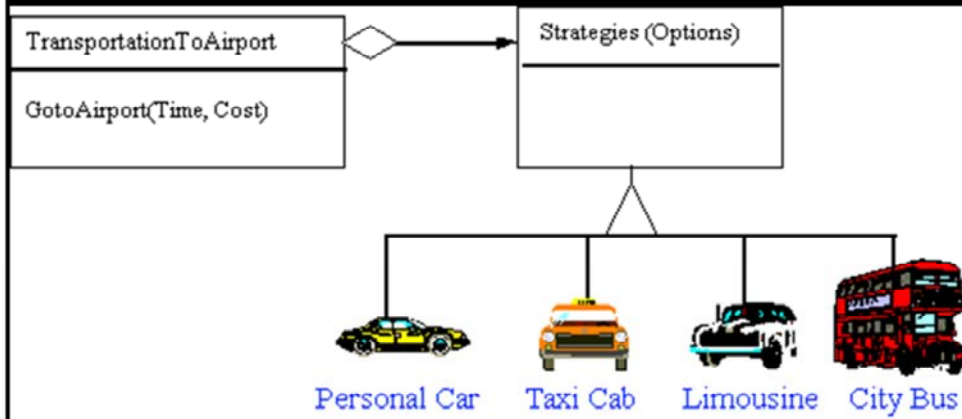

Design Principle results

- Result of this principle
 - Fewer unintended consequences
 - More flexibility

Strategy DP



Real Life Analog



Assignment: Use Strategy

- In an HR system, we have an Employee class. Now we have different policies for Permanent employees and Contract employees.



4-Feb-20 10:55 PM

20

If we subclass, we lose flexibility

We have an EmployeeType interface and assign object for every employee.

Design Principle

- Principle: Prefer composition over inheritance.

4-Feb-20 10:55 PM

21

Use Inheritance

LSP

Base class is abstract

There is a “kind of” relationship

Unfortunately,

Newcomers usually tend to overuse Inheritance.

Most OO books tend to emphasize Inheritance.

=====

Inheritance is defined at compile time, while composition can be determined at runtime.

Inheritance breaks encapsulation principle.

Inheritance is difficult to maintain. More time is spent in maintenance as compared to building an application.

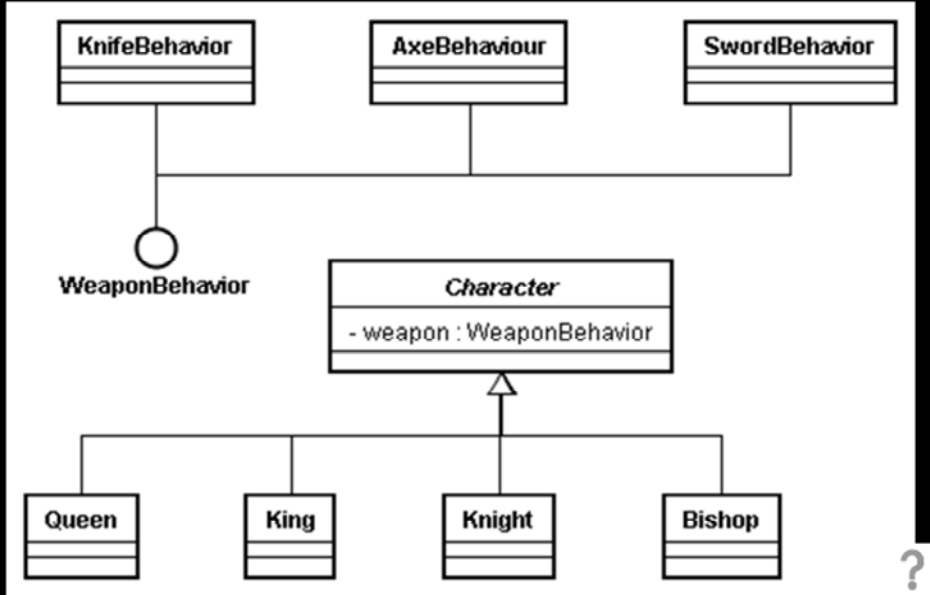
In composition, an object can be replaced by another as long as the interface is the same.

Question

- The following classes exist: Character, Queen, King, Bishop, Knight, KnifeBehaviour, BowAndArrowBehaviour, WeaponBehaviour, AxeBehaviour, SwordBehaviour.
 - The first 5 classes can use one weapon at a time.
 - They can change their weapons at runtime.
 - Arrange the classes appropriately.



Solution



4-Feb-20 10:55 PM

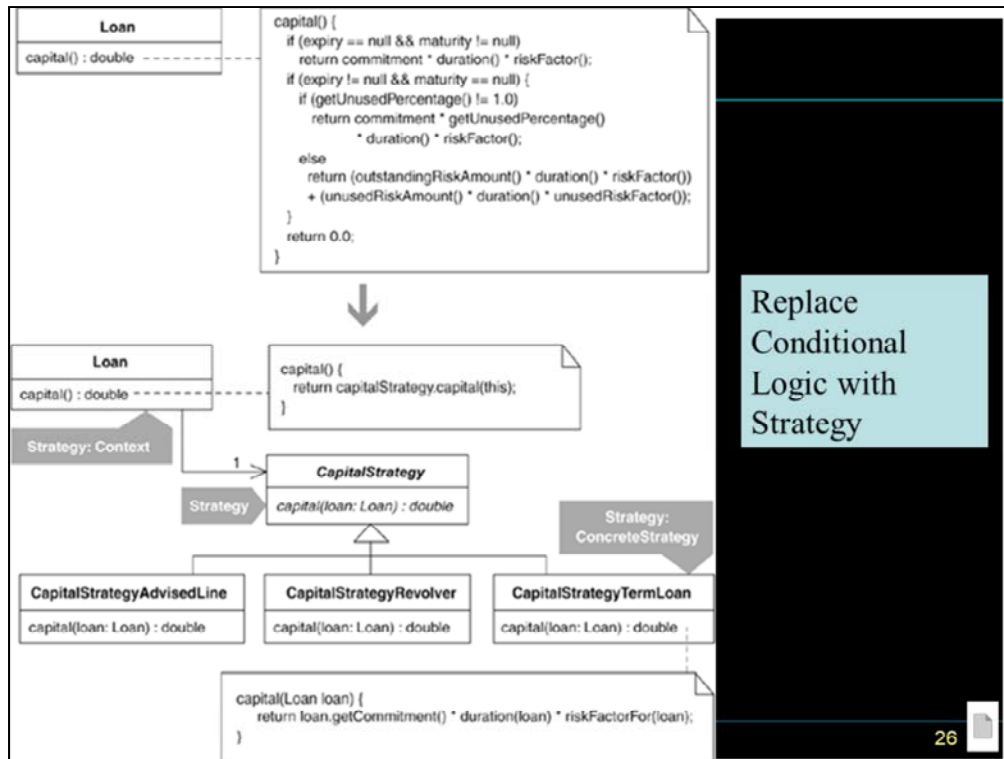
23

Examples of Strategy DP

- Comparator objects in JDK.
- Delegates in C#
- Save files in different formats. Similarly
 - Compress files with different algorithms
 - Capture video data with different compression schemes.
 - Plot the same data in different formats: line graph, bar chart or pie chart.
 - Different encryption algorithms are being used in an application.

Examples of Strategy DP

- A rich GUI editor needs different algorithms for formatting the text.
- Swing uses different algorithms for laying out GUI components.
- Different optimizations during compilation
- We need to calculate tax. Different tax calculation algorithms are used in different countries.
- Allocating objects to a multi-threaded application from
 - an object pool
 - creating & destroying them on a need basis
 - Using a singleton that is thread-safe



Benefits and Liabilities

- +Clarifies algorithms by decreasing or removing conditional logic.
- +Simplifies a class by moving variations on an algorithm to a hierarchy.
- +Enables one algorithm to be swapped for another at runtime.
- Complicates a design when an inheritance-based solution or a refactoring from "Simplifying Conditional Expressions" is simpler.
- Complicates how algorithms obtain or receive data from their context class.