Design Patterns

Introduction

COMP3607
Object Oriented Programming II

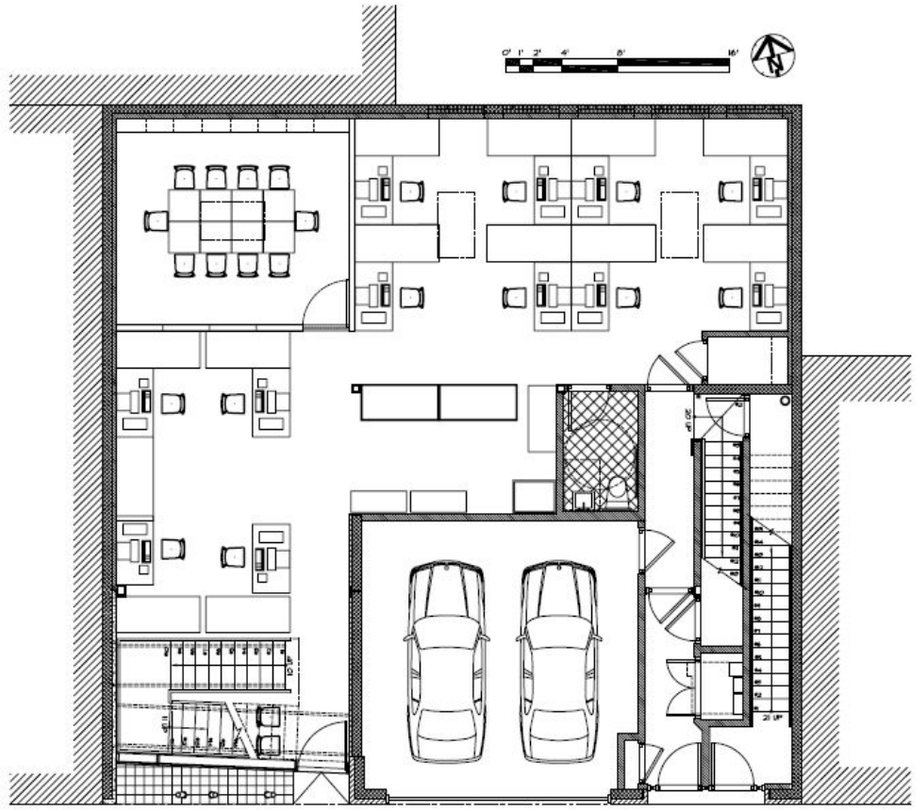
Week 5

Outline

- What is a design pattern
- Essential elements of a design pattern
- Types of design patterns
- Solving problems using design patterns
- Why use design patterns

Solving Problems

When a problem occurs over and over again in our environment, we can come up with a solution to deal with it.



1ST FLOOR PLAN

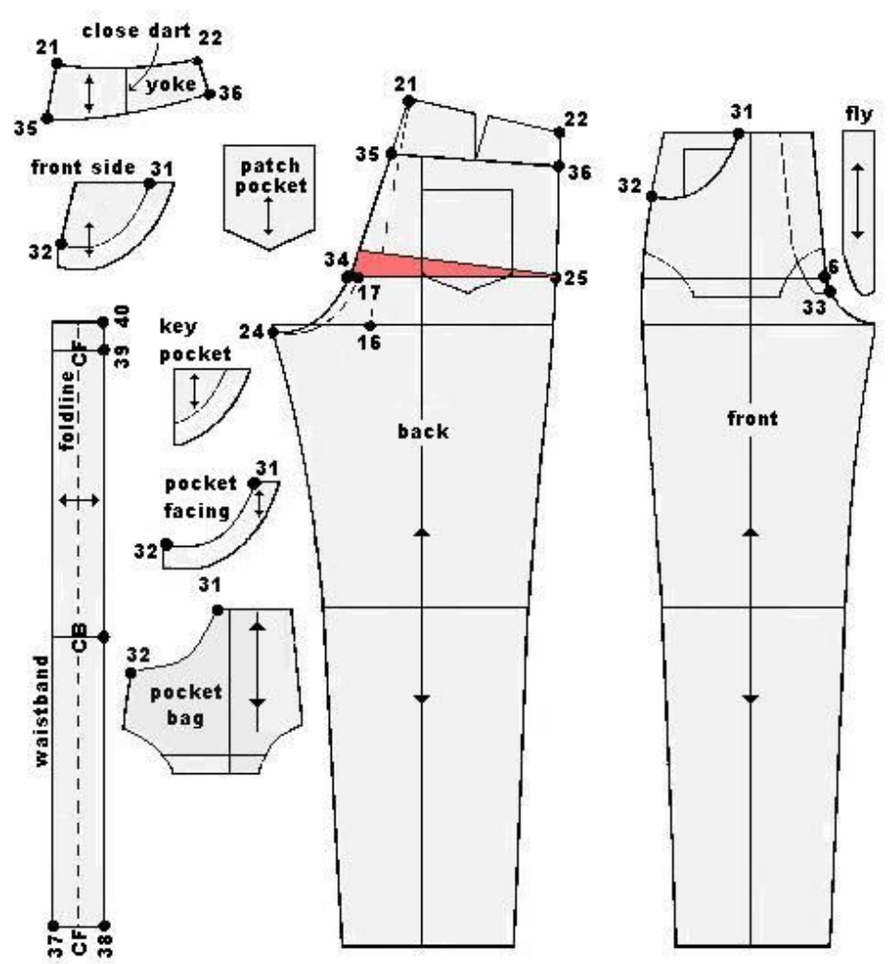
Extending Solutions

When that solution is designed well, we can use it many, many times over. Sometimes, the core of the solution can be used such that the outcomes are different.

Basic Pant

Add a zipper (style)

Add zippers (shorts)



What is a Design Pattern?

A design pattern therefore allows software developers to solve specific design problems using successful solutions that have worked in the past.

It names, abstracts, and identifies key aspects of a common design structure. For example, participating classes and instances, their roles and collaborations, and the distribution of responsibilities.

Reading Resource: Design Patterns, Chapter 1

What is a Design Pattern?

The end result of using a design pattern is a solution with a more flexible, elegant, reusable object-oriented design.

The choice of programming language is important. Some design pattern features can and cannot be implemented easily in certain languages.

Reading Resource: Design Patterns, Chapter 1

Solving Problems with Design Patterns

- Finding appropriate objects
- Determining object granularity
- Specifying object interfaces
- Specifying object implementations
- Putting reuse mechanisms to work
- Relating run-time and compile-time structures
- Designing for change

Reading Resource: Design Patterns, Chapter 1, pp 11-28

Finding Appropriate Objects

The hard part about o-o design is decomposing a system into objects. Many factors (encapsulation, granularity, dependency, flexibility, performance, reuse etc.) influence the decomposition and can conflict.

Real-world modelling can the limit the abstractions necessary for flexible solutions. Some don't have real-world counterparts. These are also seldom found during analysis or even in early design stages.

Design patterns help with identification of less-obvious abstractions and the objects that can capture them.

Determining Object Granularity

Objects can represent everything down to the hardware or all the way up to entire applications. Great variation in size and number of objects in a design.

Design patterns help us decide what should be an object.

E.g. Decomposing an object into smaller ones, supporting large numbers of objects at different levels of detail, ways of creating objects that yield other objects etc.

Specifying Object Interfaces

Interface to an object: set of all signatures defined by an object's operations (complete set of requests that can be sent to an object).

Objects are only known through their interfaces which say nothing about their implementations. Different objects are free to implement requests differently. Polymorphism allows these requests to be accepted with different behaviour at run-time.

Design patterns help define interfaces by identifying key elements and the kinds of data that get sent across an interface. They also help define relationships between interfaces.

Specifying Object Implementations

Program to an interface, not an implementation

Design patterns give different ways of associating an interface with its implementation such that objects are manipulated solely in terms of their interfaces (defined by abstract classes).

Clients remain unaware of the specific types of objects they use, as long as the objects adhere to the interface that clients expect.

Clients remain unaware of the classes that implement these objects. They only know about the abstract classes(es) defining the interface.

Putting Reuse Mechanisms To Work

Favour object composition over class inheritance

- Inheritance vs Composition:
  - Object composition is defined dynamically at run-time through objects.
  - Object composition keeps class hierarchies small, focused and encapsulated.

Putting Reuse Mechanisms To Work

Favour object composition over class inheritance

- Delegation: Two objects are involved in handling a request - a receiving object delegates operations to its delegate. Behaviours are composed at run-time and can easily change.

Essential Elements of a Design Pattern

A pattern has four essential elements:

1. Pattern Name
2. Problem
3. Solution
4. Consequence

Pattern Element: Name

The pattern name is a handle used to describe a design problem, its solutions and consequences in a word or two.

Pattern Name	Design Problem
Singleton	Ensures that a class has only one instance
Observer	Defines a 1:M dependency between objects for consistent update of state
Adapter	Converts the interface of a class into another interface clients expect

Examples

Pattern Element: Problem

The problem describes when to apply the pattern and explains the context of use e.g. how to represent algorithms as objects, or how to describe class or object structures, or a list of conditions required for applying a pattern.

Pattern Name	Design Problem
Singleton	Ensures that a class has only one instance

Example

Pattern Element: Solution

The problem describes when to apply the pattern and explains the context of use e.g. how to represent algorithms as objects, or how to describe class or object structures, or a list of conditions required for applying a pattern.

Pattern Name	Solution
Singleton	Define an Instance( ) class method that lets clients access its unique, private instance

Example

Pattern Element: Consequences

Consequences are the results and trade-offs of applying a pattern. These are critical for evaluating design alternatives and for understanding the costs/benefits of applying a pattern. E.g. time, space, implementation issues, reusability, extensibility, portability etc.

Pattern Name	Consequences
Singleton	• Controlled access to sole instance • Permits a variable number of instances • Permits refinement of methods

Example

Design Pattern Space

		Purpose		
		Creational	Structural	Behavioural
Scope	Class	Factory Method	Adapter (class)	Template
	Object	Abstract Factory Singleton	Adapter (object) Composite Façade Flyweight Proxy	Command Iterator Mediator Observer State Strategy

Design Pattern Scope

The scope of a design pattern specifies whether the pattern applies primarily to **classes** or to **objects**.

Class patterns deal with relationships between classes and their subclasses. These relationships are established through inheritance and hence are static (fixed at compile-time).

Object patterns deal with object relationships which can be changed at run-time and are more dynamic. Most patterns are in the object scope.

Design Pattern Purpose

The purpose reflects what a pattern does. Patterns can have either **creational**, **structural** or **behavioural** purpose.

Creational patterns concern the process of object creation.

Structural patterns deal with the composition of classes or objects.

Behavioural patterns characterise the ways in which classes or objects interact and distribute responsibility.

Creational Patterns

Creational patterns abstract the instantiation process. They help make a system independent of how its objects are created, composed and represented.

A class creational pattern uses inheritance to vary the class that is instantiated.

An object creational pattern will delegate instantiation to another object.

Reading Resource: Design Patterns, Chapter 3, pp 81-85

Creational Patterns

Recurring themes:

1. Encapsulate knowledge about concrete classes used
2. Hide how instances are created and put together.

Creational Patterns
-Examples-

		Purpose
		Creational
Scope	Class	1. Factory Method
	Object	2. Abstract Factory 3. Singleton

Structural Patterns

Structural patterns are concerned with how classes and objects are composed to form larger structures.

Structural class patterns use inheritance to compose interfaces or implementations.

Structural object patterns describe ways to compose objects to realise new functionality.

Reading Resource: Design Patterns, Chapter 4, pp 137-138

Structural Patterns

Recurring themes:

1. Classes combine the properties of parent classes using single and multiple inheritance

2. Object composition changes at run-time (for structural object patterns only)

Structural Patterns
-Examples-

		Purpose
		Structural
Scope	Class	1. Adapter (class)
	Object	2. Adapter (object) 3. Composite 4. Façade 5. Flyweight 6. Proxy

Behavioural Patterns

Behavioural patterns are concerned with algorithms and the assignment of responsibilities between objects.

These patterns describe patterns of objects and classes, and the patterns of communication between them.

Behavioural class patterns use inheritance to distribute behaviour between classes.

Behavioural object patterns use object composition rather than inheritance.

Reading Resource: Design Patterns, Chapter 5, pp 221-222

Behavioural Patterns

Recurring themes:

1. Encapsulating variation (Describing aspects of a program that are likely to change)

2. Objects as arguments

3. Decoupling senders and receivers

Behavioural Patterns
-Examples-

		Purpose
		Behavioural
Scope	Class	1. Template
	Object	2. Command 3. Iterator 4. Mediator 5. Observer 6. State 7. Strategy

References and Reading Selected Pages/Chapters

- Gamma, Helm, Johnson, and Vlissides: Design Patterns:
  - Chapter 1
  - Chapters 3, 4, 5 (selected pages)