COMP 2511
Object Oriented Design & Programming

Week 12

Revision

Design Toolbox

OO Basics

- *Abstraction*
- *Encapsulation*
- *Inheritance*
- *Polymorphism*

OO Design Principles

- *Principle of least knowledge – talk only to your friends*
- *Encapsulate what varies*
- *Favour composition over inheritance*
- *Program to an interface, not an implementation*
- *Classes should be open for extension and closed for modification*
- *Don't call us, we'll call you*
- *A class should have only one reason to change*
- *Strive for loosely coupled designs between objects that interact*
- *Depend on abstractions, do not depend on abstract classes*

Design Patterns

Structural Patterns

- *Composite*
- *Decorator*

Behavioural Patterns

- *Strategy*
- *State*
- *Template Method*
- *Iterator*
- *Observer*
- *Visitor*
- *Command*

Creational Patterns

- *Factory Method*
- *Abstract Factory*
- *Builder*
- *Singleton*

Design Principles & Design Patterns

- **Design Principles** are basic techniques that:
  - can be applied to designing or writing code to make software more maintainable, flexible and extensible
  - Helps to build software, where modules strive towards loose coupling and high cohesion

- **Design Patterns** are:
  - Repeatable solutions to commonly occurring problems in software design
  - Provide an architecture to adhere to design principles

- **Refactoring Techniques** are:
  - Series of code transformation that help to improve structure of code without modifying the external observable behaviour and help to conform to design principles
  - Some techniques are based on implementing design patterns

Design Patterns

A repeatable solution to a commonly occurring problem in software design

Creational Patterns

- *Factory Method*
- *Abstract Factory*
- *Builder*
- *Singleton*

Structural Patterns

- *Composite*
- *Decorator*

Behavioural Patterns

- *Strategy*
- *State*
- *Template Method*
- *Iterator*
- *Observer*
- *Visitor*
- *Command*

Design Principles (1)

Principle of Least Knowledge (Law of Demeter)

A method **M** in an object **O** can call on:

- Any other method within **O** itself
- Any methods of parameters passed to M
- A method N of another object, if that object is instantiated within M
- Any methods of any type of object that is a direct component of **O**

Design Principles (2)

Liskov's Substitution Principle

- Subtypes should be substitutable for their base types i.e. the contract of the base class must be honoured by their derived class

- When you inherit from a base class, you must be able to substitute your subclass for that base class without things going terribly wrong. Otherwise, you've used inheritance incorrectly!"

Design Principles (3)

Encapsulate what varies

- Identify aspects of your code that varies and “encapsulate” and separate it from code that stays the same, so that it won’t affect your real code
- Encapsulate behaviour (State, Strategy)
- Encapsulate object creation (Factory, Abstract Factory)
- Encapsulate method invocation (Command)

Design Principles (4)

Favour composition over inheritance

- Use of inheritance for “re-use” only is incorrect
- A more flexible way to re-use behaviour is use classes with composition and delegate the implementation of the behaviour to the composed class
- Enables you to change behaviour at run-time

Strive for loosely-coupled designs between objects that interact

- Loosely coupled designs minimise interdependency between objects and allow us to build flexible OO systems

Design Principles (5)

Hollywood Principle: Don’t calls us, we will call you

- Enable low-level components to hook themselves into a system, but the high-level components decide “when and how” they are needed

- Template Method Pattern makes use of this principle, which defines the skeleton of an algorithm and defers some steps to sub classes

Design Principles (6)

Classes should be open for extension but closed for modification (OCP)

- New changes must be implemented by new code instead of altering existing code
- The strategy, state pattern, decorator and template method patterns are examples of patterns that help to conform to OCP

A class should have only one reason to change (SRP)

- Classes that adhere to SRP tend to have high cohesion

Design Principles (7)

Program to an interface (super-type) and not to an implementation

- The declared type of the variable should be a super-type (abstract class or interface)
- Helps to exploit polymorphism

10. Depend upon abstractions. Do not depend on create classes

- A high-level module should not depend on “low-level” modules, but both should depend on abstractions

Grouping Design Patterns: Creational Patterns (1)

Involve object instantiation and all provide a way to decouple a client from the objects it needs to instantiate

- Factory Method: Defines an interface for creating an object, but lets sub-classes decide which concrete class to instantiate
- Abstract Factory: Provides an interface to create familes of related objects without specifying their concrete classes
- Builder: Separate the construction of a complex object from its representation so that the same construction process can create different representations
- Singleton: Ensures a class has only one instance

Grouping Design Patterns: Structural Patterns(2)

Enables composition of objects into larger structures, providing a way to build simple and efficient class hierarchies

- Composite: Compose objects into tree structures to represent part-whole hierarchies, enabling clients to treat individual objects and compositions of objects in a uniform way
- Decorator: Attach additional responsibilities to an object dynamically. Provide a flexible alternative to sub-classing for extending functionality

Grouping Design Patterns: Behavioural Patterns(3)

Are concerned with how objects interact and identifies common communication patterns between these objects

- Command: Encapsulates a command request as an object
- Iterator: Provides a way to traverse the elements of a collection without exposing its implementation
- Observer: Allows objects to be notified when state changes
- State: Encapsulates state-based behaviours and uses delegation to alter an object's behaviour when its internal state changes
- Strategy: Encapsulates a family of interchangeable algorithms or behaviours and uses delegation to decide which one to use
- Template Method: Defines skeleton of an algorithm, deferring some steps of the algorithm to sub-classes
- Visitor: Defines a new operation to a class without changing it