

# COMP2511

## Course Review Exam Structure

Prepared by  
Dr. Ashesh Mahidadia

# Course Review

# Object Oriented Programming in Java: Introduction

- Abstraction
  - Encapsulation
  - Inheritance (single vs multiple)
  - Polymorphism
  - Objects, Classes, Interfaces
  - Method Forwarding
  - Method Overriding
  - Generics
  - Exceptions
- 
- Domain Modeling

# Object Oriented Design : Principles

- Encapsulate what varies
- Favour composition over inheritance
- Program to an interface, not an implementation
- Principle of least knowledge (Law of Demeter)
- Liskov's Substitution Principle
- Classes should be (OCP) open for extension and closed for modification
- Avoid multiple/diverse responsibilities for a class
- Strive for loosely coupled designs between objects that interact

# Code Smells and Refactoring

- ❖ **Smells**: design aspects that violate fundamental design principles and impact software quality
- ❖ **Design Smells vs Code Smells**
- ❖ **Code smells** are usually not bugs; they are not technically incorrect and do not prevent the program from functioning.
- ❖ They indicate **weaknesses** in design that may slow down development or increase the risk of bugs or failures in the future.
- ❖ Regardless of the granularity, smells in general indicate violation of software design principles, and eventually lead to code that is rigid, fragile and require “*refactoring*”
- ❖ **Code refactoring** is the process of **restructuring** existing computer code **without changing** its external **behavior**.

# Design Patterns

## ❖ Creational Patterns

- ❖ Abstract Factory
- ❖ Factory Method
- ❖ Builder
- ❖ Singleton

## ❖ Structural Patterns

- ❖ Adapter
- ❖ Composite
- ❖ Decorator

## ❖ Behavioral Patterns

- ❖ Iterator
- ❖ Observer
- ❖ State
- ❖ Strategy
- ❖ Template
- ❖ Visitor

# Design Patterns: Creational Patterns

**Creational design** patterns deal with object creation mechanisms, trying to create objects in a manner suitable to the situation.

Four well-known creational design patterns:

- ❖ **Factory method pattern:** allows a class to defer instantiation to subclasses.
- ❖ **Abstract factory pattern:** provides an interface for creating related or dependent objects without specifying the objects' concrete classes.
- ❖ **Builder pattern:** separates the construction of a complex object from its representation so that the same construction process can create different representations.
- ❖ **Singleton pattern:** ensures that a class only has one instance, and provides a global point of access to it.

from the corresponding wikipedia page.

# Design Patterns: Structural Patterns

**Structural design patterns** are design patterns that ease the design by identifying a simple way to realize relationships among entities.

Three well-known structural design patterns:

- ❖ **Adapter pattern:** *'adapts'* one interface for a class into one that a client expects
- ❖ **Composite pattern:** a tree structure of objects where every object has the **same interface** (leaf and composite nodes)
- ❖ **Decorator pattern:** add additional functionality to a class at **runtime** where subclassing would result in an exponential rise of new classes.

from the corresponding wikipedia page.



# Design Patterns: Behavioral Patterns

**Behavioral design patterns** are design patterns that identify common communication patterns among objects and realize these patterns. By doing so, these patterns increase flexibility in carrying out this communication.

Six well-known structural design patterns:

- ❖ **Iterator pattern:** Iterators are used to access the elements of an aggregate object sequentially without exposing its underlying representation.
- ❖ **Observer pattern:** Objects register to observe an event that may be raised by another object. Also known as Publish/Subscribe or Event Listener.
- ❖ **Strategy pattern:** Algorithms can be selected at runtime, using composition.
- ❖ **State pattern:** A clean way for an object to partially change its type at runtime.
- ❖ **Template method pattern:** Describes the program skeleton of a program; algorithms can be selected at runtime, using inheritance.
- ❖ **Visitor pattern:** A way to separate an algorithm from an object.

from the corresponding wikipedia page.

# Exam Structure

# Final Exam : Structure

- ❖ **Online** exam (like practical exams in the previous courses)
- ❖ **Three** parts
- ❖ Part-1: **Multiple Choice** Questions (20 marks),
  - **make sure** to save and importantly to **submit** (by clicking on the button)
- ❖ Part-2: **Short Answer** Questions (30 marks) – answer in a text box provided, brief and to the point, only few lines!
  - **make sure** to save and importantly to **submit** (by clicking on the button)
- ❖ Part-3: **Programming** (50 marks) – four questions (see the sample exam),
  - follow the instructions in a question to **submit**.

# Sample Final Exam

- Go to :

<https://www.cse.unsw.edu.au/~cs2511/19T2/lectures/SampleExam/SampleExam.html>

# Evaluation

- ❖ CATEI Evaluation available via myUNSW.
- ❖ Tell us what you like/dislike about the course, we do take your input seriously.
- ❖ Thanks ...

And Finally ... ..

That's All Folks

Good Luck with the Exams

and with your future computing studies



End