**Week 1 - Software Quality and OOP design**

Development comprises 3 broad activities:

* Analysis
* Design
* Programming

Classes are “concrete entity” (student, course, etc.) or abstract concept or category, to discover classes, look for nouns or significant concepts or artifacts.

1. Association: one class uses another class through attribute level reference (does not imply ownership) (A uses B)

Example: a book can be borrowed by many readers and a reader can borrow many books

1. Aggregation: similar to association, but only refers to attribute level access (not parameter or local variables). It implies part/whole semantics but objects on both sides have independent lifetimes (weak part – whole association). (B is part of A)

Example: a staff can work in a department, a department can have many staffs, but when a department is removed, all staffs will not be removed but can move to other departments

1. Composition: same as aggregation, but the part/whole semantics is enforced, if a whole is deleted, then so are the parts. The “part” object can belong to only one “whole” object as value attribute (strong part – whole association)

Example: a customer can have many electronic wallets and when a customer is removed, the electronic wallets of that customer are removed as well.

1. Dependency: a class uses another class as local variable or an input parameter but not in the attribute. (A uses B)

Example: a shopping cart will depend on the product to perform add operation. When there are changes in the product, the shopping cart may change as well but not vice versa

1. Inheritance: a class inherits all attributes of another class, and also can have its own methods or attributes (A is-a B)

OOP is based on a number of objects working together to perform a function, which can give benefits of easier “code maintenance”, and “reusability”. But a poorly designed OOP program can also be difficult to use and maintain.

Agile development emphasizes an iterative, incremental and evolutionary approach.

Encapsulation indicates that both attributes and methods are contained within a single code module.

Information hiding indicates that private information of a class can only be accessed by its own methods (can have private methods as well). Therefore, outside access to class’s internals are on a “need to know basis” (only some private information can be accessed from outside and no more than that).

The dependence of a class on others is called “coupling”. Many coupling between classes can cause the problems of modifying a class will cause significant changes to many other classes.

Cohesion means that each class has elements with well – defined and logical roles to fulfill its purpose. The object of the class will define one concept or perform one task well.

Cohesion and coupling are trade – off, improve one may have negative effects to others.

**Week 2 - Inheritance and Poly-morphism**

Method overloading is the process of creating many methods with the same name but vary in the type of number of arguments. The actual method invoked is based on the “declared type” at “compile time”

* In inheritance, child class inherits all non – private attributes and methods of parent class even static methods
* Inherited attributes can be modified, and methods are called as if they were part of parent class without having to rewrite or copy them
* This enhances code reusability and extensibility.

The only way to change private attributes and methods in superclass is through the mutators or methods of that superclass.

“Final” keyword indicates that a class or a method cannot be extended or overridden.

Polymorphism enhances code reusability and extensibility by calling the method in a generic way. It can simplify code usage and understandability related classes can be treated in the same way without any conditional codes.

NOTE: a subclass reference is automatically a superclass reference, but not vice versa

Example: class Dog is subclass of class Animal, can convert class Animal to class Dog if class Animal references Dog object. Otherwise, the Animal object cannot be converted to Dog object.

Classes do not inherit constructors

**Week 3 - Abstract Classes and Interfaces**

“Abstract class”:

* cannot be instantiated
* is always inherited and usually enhanced by subclasses
* can have abstract methods
* cannot be instantiated using “new” keyword but can define constructors that can be invoked from subclass, and methods to invoked, implemented, or overridden.

An abstract method:

* is a method signature without any implementation
* can only exist in an abstract class
* a subclass of an abstract superclass must implement all abstract methods of that superclass. Otherwise, it must become abstract class
* can exist in interfaces.

A subclass can be abstract even though the superclass is concrete.

“Interface” is a class – like constructor ­­with only constant and abstract methods. It is similar to abstract class but cannot contain variables and concrete methods.