Object-Oriented Design

UNIVERSITY OF ALBERTA

Course Glossary

We have attempted to identify the most important terms in the course here. Look here if you encounter a term without a definition in context. If you need further explanation, you may have to return to the lesson where we first introduced the term, or do some independent study.

---

abstract - in object-oriented design, a class that cannot be instantiated directly, but must be subclassed. It can also apply to a method or attribute of a class.

abstract data types - data types that are defined by the their behaviour as opposed to their structure. Defined by the developer rather than the programming language

access modifiers - keywords that control which other classes can access a variable or method in a class. These include **public, protected, private,** and no keyword, sometimes called **default**.

abstraction - the act of simplifying a concept in context. In object-oriented design, it is the simplification of the real world entity into its most important attributes and behaviours for the purpose of the software.

attribute - a property that object of a class must have, even though their values may be different. For example, two student objects each have a grade attribute, even though one has an 'A' and the other has a 'B.'

behaviours - the actions that an object can take

boundary object - an object whose role is to interface with an external component, such as a user or an adjacent system

class diagram - a UML diagram for showing the behaviours, attributes, inheritance, and connections of classes

code review - systematic reviews of written code done by the development team. Not only to find mistakes, but to get developers using the same conventions, constructs, design principles, etc.

cohesion - describing the complexity within a module, e.g. a class or a method. **high cohesion** describes a module that has a clear purpose and is no more complex than it needs to be. **low cohesion** describes a module which has an unclear purpose or which is overly complex.

component - a discrete part that has a particular role or function, called a **responsibility**. From a design perspective, a component will eventually be turned into an object, function, or group of subcomponents

conceptual integrity - the consistency of software. Software with conceptual integrity will seem like it is programmed by one developer, even if it was programmed by a team

concern - a general term, referring to some action or role that is part of the solution to the problem.

control object - an object whose role is to manage other objects or control their interactions

counterexamples - during model checking, counterexamples are instances wherein the system did not behave as expected

coupling - describing of the complexity of connections between modules. **tightly coupled** modules are highly dependent on each other and difficult to reuse in other contexts. **loosely coupled** modules are less dependent and easier to reuse.

CRC - stands for class responsibility collaborator: a technique for summarizing and mapping objects in an object-oriented design.

deadlock - a situation in which the system can never continue because subprocesses need other subprocesses to act before they can continue

decomposition - breaking an entity into parts that can be implemented separately

degree - when talking about coupling, degree is the number of connections between the two modules of interest. This is one dimension of how coupled these modules are.

design - the process of planning a software solution, taking the requirements and constraints into account. Divided into higher-level conceptual design and more specific technical design.

design patterns - established solutions to common coding problems. Characterized by their general form and function rather than by specific code

ease - when talking about coupling, ease is how obvious connections are between modules. This is one dimension of how coupled these modules are.

encapsulation - bundling attributes and behaviours into an object, exposing features of that object to other objects as necessary, and restricting the remaining features

entity - the role or behaviour that is being represented by a software object or process

flexibility - when talking about coupling, flexibility is how easily a module can be swapped for a different module. This is one dimension of coupling.

getter - a method for getting the value of a class variable which is not directly accessible

generalization - factoring out common features of classes or functions that can be reused in other places. Allows for more code reuse

global variable - a variable that is accessible by any subroutine or subcomponent

flexibility - the ability of a design to adapt to changes or be adapted to different purposes

implementation - the process of creating a working program from the design

information hiding - designing classes so that the information that other classes do not need is hidden away from them.

inheritance - attributes or behaviours that subclasses inherit from a superclass or implement through an interface

instantiate - to create an object of a class

interface inheritance - a method of inheritance wherein if a class implements an interface, it must define all the methods specified in the interface

Liskov substitution principle - a principle stating that a superclass should be able to be replaced by its subclass without changing behaviour significantly

local variable - a variable accessible only to one class or subroutine

maintenance - modifying software after delivery to fix, improve, or change features

maintainable - the ability of code to be changed

model - the abstract representation of the key concepts and relationships that make up a software solution

model checking - a systematic check of all of the system's states. Consists of several steps: **modeling phase**, **running phase,** and **analysis phase**

module - general term to refer to a programming unit, like a class or a method.

namespace - an abstract container for a group of related modules, given a unique identifier.

object-oriented modelling - modelling a software solution using concepts from object-oriented languages such as Java. Characterized by representing key concepts with software objects.

override - a subclass may have a method that is already in the superclass, in which case the subclass' method will be used instead.

package - a means of organizing related classes into the same namespace.

polymorphism - the ability to interact with objects of different types in the same way. Usually achieved through inheritance or through interfaces in Java

programming paradigm - the style or way in which programming achieves its objectives, which can vary by language and toolset

quality attributes - properties of a software system that indicate its quality

requirements - the requirements that the software will be designed to fulfill. These could be functional requirements such as providing some result, or business requirements such as being user-friendly, or meeting budgetary restrictions

responsibility - the purpose or function of a component of the software

reusable - the ability of code to be reused in different contexts

rule of least astonishment - a design principle dictating that a component should behave as one would expect it to behave

separation of concerns - a principle dictating that different concerns should be in different modules

service-oriented architecture - a type of architecture characterized by providing services to external clients. The clients do not know how these services are provided.

sequence diagram - a UML diagram that shows the sequence of actions that form one process

setter - a method for setting the value of a class variable which is not directly accessible. Allows for gatekeeping e.g. restricting the values to which the variable can be set

software architecture - the higher-level structure of a software system; how various components are arranged into a coherent and functional whole

state diagram - a UML diagram that shows the different states of a system

tradeoff - a decision between alternatives that each provide benefits and downsides

Unified Modelling Language - a visual design language encompassing many different diagrams that depict software in different ways

verification - confirming that the software solution meets the requirements