“The Object-Oriented Thought Process,” 5th

Notes open for creative commons use @ developer blog: https://unfoldkyle.com, github: SmilingStallman*, email: kmiskell@protonmail.com*

**Reading this book was to serve as a refresher to the core concepts of OOP and to get back into an OO mindset after a long time in functional and procedural programming. The notes can provide the same, shortened.

Intro to OO Concepts

-Book more focuses on teaching basic OOP concept than code. How to think OO.

-Object wrappers – OO code that wraps other code (ex. procedural), to make it work like an object. Useful for wrapping functionality from legacy code.

-Basis of OOP – People think in terms of objects, so why not base architecture around them too?

-Object – an entity that has both attributes and behaviors
-Attributes – the data of the object. Should all be *private/protected*, accessed via abstractions.
-Methods – the behavior of the object. Functions inside a class. Access/use attributes.

-In procedural, data is often global, and separated from functions. Functions still access this data, though, and hence access to data is uncontrolled and unpredictable. With objects, data and behavior are grouped together and encapsulated from other objects.
-In proper OO design, there is no such thing as global data

-Objects allow data hiding of details from other objects, resulting in strong control over data and how logic is related among entities. By hiding implementation, use is simplified to a limited interface.

-Encapsulation – the segregation of the behaviors and attributes of an object from other objects. Allows hiding of implementation and only access through interface and select methods/attributes.

-Interfaces – the communication methods between objects. *public* methods. Implementation is *private* or *protected* and accessed through *public* interfaces. Through this, implementation can change without affecting objects that use interface.

-Objects should not manipulate the attributes of other objects. Many small objects that perform specific tasks > huge objects that take on too many responsibilities.

-Getter and setter methods allow controlled access to data and support info hiding. Getter method, aka accessor method. Setter method, aka mutator method.

-Instantiation – creation of an instance of an object. Each instance of an object contains its

own state for attributes and reference to methods (if non-static).

-Class – blueprint for an object that defines attributes, behaviors, and messages of objects built from it. Classes instantiated into objects, which are instances of classes.

-Message – behavior implemented in an object and called by another object (ex. *public* method). “Object A calls Object B which sends it a message.” Opposite to *private* behaviors.

-Primary visual tool in OOD are UML diagrams, with class diagram being most common type. In class diagrams + means *public* and – means *private*.

-Instantiation method uses to create object is considered part of the interface

-Inheritance – allows a class to share the attributes and methods of another class. Child classes can be created as abstractions of parent classes that share like behavior/attr. Inheriting classes can have their own behaviors/attr separate from parent classes. Common for data to be inherited but behavior not to be. Supports DRY.

-Superclass – class from which another class inherits from. Parent class.
-Subclass – extends (inherits from) superclass. Child class. Derived class.
-Superclass can be subclass also and vice versa.

-Abstraction – class design as a mixture of hidden implementation and public interfaces so as only to provide an abstraction of a class, not a full definition, on a, “need to know,” basis. Also seen through inheritance, where child is abstraction of parent class, and composition, where class contains instances of other classes.

-As inheritance tree grows, so does complexity of abstraction, hence leaving many devs wary of using inheritance.

-Single inheritance – classes inheriting from one class only
-Multiple inheritance – classes inheriting from multi classes. Not allowed in some langs (ex. Java). Can result in nameclashes.

-Is-a relationships – child class is-a parent class. Dog is-a Mammal. Dog *extends* Mammal.
-Allows child classes to be referenced as parent classes. Ex. Circle and Square both extend Shape. An array of shapes can also hold Circles and Squares.

-Polymorphism - “many shapes” in Greek. The ability of a class to take on multiple forms through inheriting subclasses, where different forms can have different implementation, but share the same common interface. Abstract methods, etc.

-Overriding – when a child class redefines the implementation for an existing method in a parent class of same name, return type, etc.

-Abstract method – method given definition (name, return type, scope, etc.) but no implementation. Implemented by child classes or classes implementing interface, etc. All child classes inheriting from class with *abstract* method must implement details for. Allows forced uniformity and shared interface amongst child classes.

-Abstract methods allow flexibility as can create subtype objects as instances of parent classes, then call same method on all, where each subtype has own implementation, but all share same interface.

-Constructor – method called to build object from class. Holds initialization details for attributes, start-up tasks, etc..

-Composition – when an object contains instances of other objects. *has-a* relationship. Car *has-a* Engine.

Thinking in Objects

-Analysis of business problems and possible solutions should come before choosing a language framework. The requirements determine the tools, not vice versa. Do conceptual analysis and design first.

-Separation of interface and implementation is essential and all not “need to be known” should be hidden. The driver only needs to know how to use what is in the cab, not under the hood. The interface should be minimal.

-Interchangeable objects must have the same interface (else not interchangeable)

-Implementation should be able to be changed without requiring change in interface definition (method name, return type, etc.)

-API is one example of an interface

-Middleware – connect high and low level. Software layer between apps and OS, API allowing access to DB data, etc. Can be used to connect new and legacy code by acting as in-between for data model/type conversion, etc.

-Object persistence – the concept of saving the state of an object to be used at a later time, even when falls out of scope, by saving state in DB, etc.

-Standalone application – can exist entirely from code defined in app, without needing to pull data from DB, etc.

-