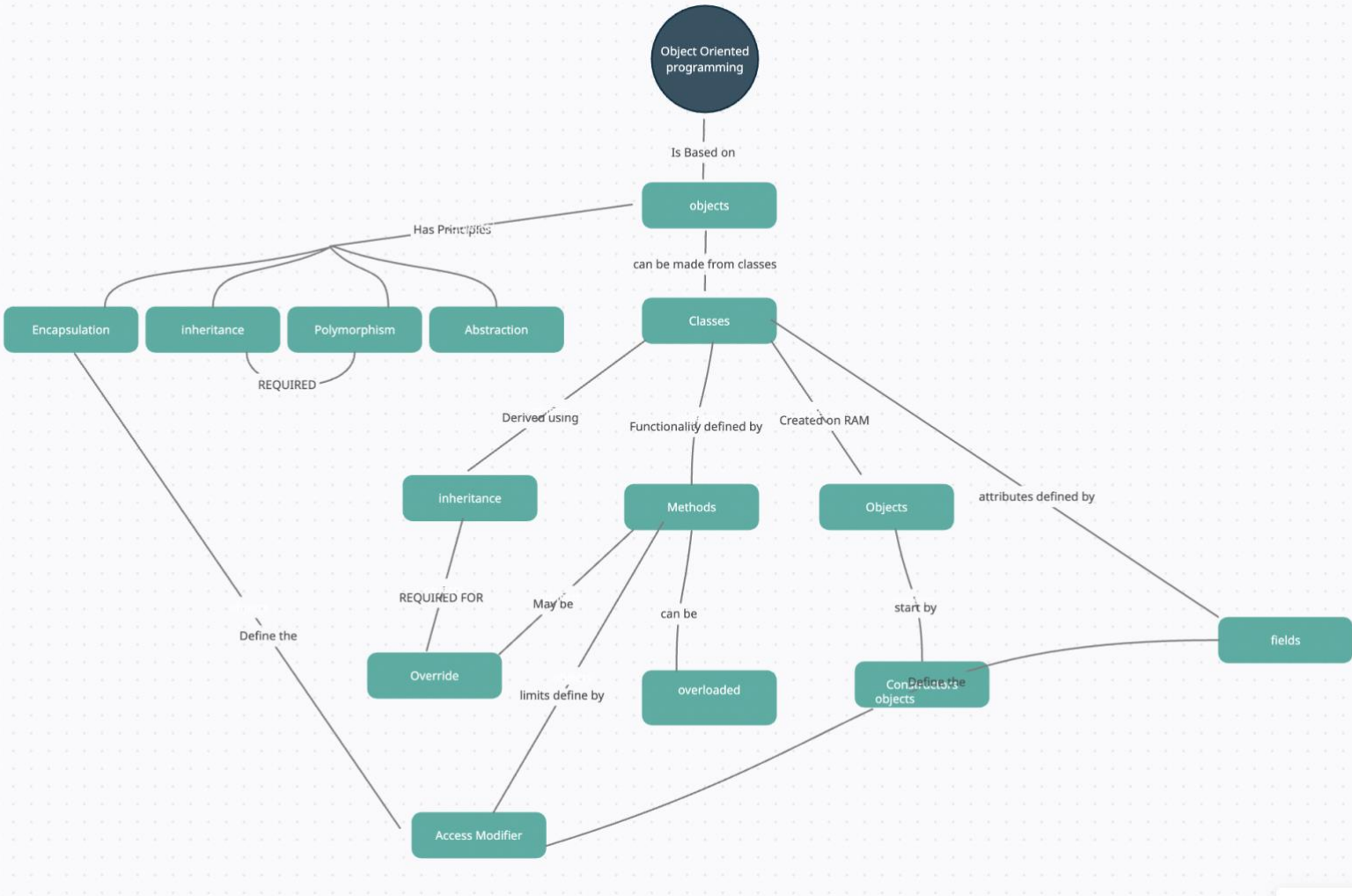
Swinburne University of Technology

COS20007 Object Oriented Programming

Key Object Oriented Concepts

PDF generated at 16:36 on Thursday 25th May, 2023

Produced by OOP Submission Formatter


Object Oriented programming
Is Based on
objects
Has Principles
can be made from classes
Encapsulation
inheritance
Polymorphism
Abstraction
Classes
REQUIRED
Derived using
Functionality defined by
Created on RAM
attributes defined by
inheritance
Methods
Objects
REQUIRED FOR
May be
can be
start by
Define the
Override
overloaded
fields
limits define by
Define the objects
Access Modifier

Object Oriented Programming 6.1P: Object Oriented Concepts

Inheritance

To manage complexity, inheritance in the OOP paradigm allows you to create a class that's derived from another base class. The naming convention used to describe this is child and parent classes, where a child class is the inherited sub class from the parent class. The parent class acts as a base class and thus allowing the child class to inherit all its fields and methods. Ultimately, this encourages code reuse and allows for an efficient design. An efficient design may provide a family of classes that have inherited members/functionality from each other so that the same code isn't repeated. Likewise, an extension of an existing software may use inheritance from its current family of classes to add new features.

The multiple shapes task uses inheritance very effectively. The objective for this task is print various shapes. We can easily create separate classes for each shape and define everything within those classes. However, a much more appropriate way to think about this is that all these different shapes have common characteristics such as, the shapes coordinates, whether it's been selected or its colour. These characteristics are defined in a more general class called shape as this applies to all types of shapes. The inheritance part comes into play when we define child classes off shape that are more specific such as a myCircle or myRectangle. These sub-classes inherit all the data members of Shape and can add extra data members that are necessary to the child class and child class only.

Abstraction

The principle of Abstraction in OOP relates to how much you can generalise an object without adding any specific details that define its characteristics related to that specific object. Let's look at the clock example from this unit, abstraction in the clock's domain is the idea that the clock is made up of hours, seconds and minutes, the clock only needs to tick and reset for it be a clock. At a later stage the clock's detail that are specific to that clock can be added. That maybe be, what colour it is, the shape of the clock, whether its digital or quartz (maybe an atomic clock) but fundamentally all these clocks have the same characteristics and that is they need to tick and provide the time in seconds, minutes and hours and due to the nature of earth's spin, it needs to reset. Ultimately abstraction defines the nature of objects to be that they have their own roles and responsibilities.

Encapsulation

Encapsulation restricts the ease of access to attributes within a class. The core idea of encapsulation is to disallow the interference with current data as this can have detrimental effects to software. This hides the internal workings of class attributes and the only form of access to these is through methods and access modifiers (getters and setters) and in essence "encapsulates" the attributes and methods into a single unit. The methods and properties essentially allow us to use the objects without needing to understand any internal implementation. They take in any data/parameters when the function/property is called and implement to wherever it is assigned to depending on the design. This prevents any tampering from the outside, be it a developer or someone else and thus encapsulates

the class and the only knowledge needed is to know what function/method to use and the parameters it takes in.

Using the same example from multiple shapes, the attributes (_color, _x, _y, _selected) are set private, and the allowed access to these attributes is done through public getters and setters. Each attribute has its own public property that allows to retrieve the variable and set its value, this way the internal state of the shape object is protected, and any access or modification is done through controlled methods.

Polymorphism

Polymorphism is a feature of Object-oriented-programming where objects of different types can have many forms, which allows objects of different classes to be treated uniformly based on their common base class, which promotes code ruse and extensibility.

Using the Shapes task as an example, there exists a base class “shapes” for all shapes, of which, you can derive other specific shape classes, be It rectangle or circle and inherit all the properties of the base shape class. Each of the derived class has its own unique characteristics and behaviours that are specific to the shape, be it rectangle or line. Despite being able to derive any shape, polymorphism allows us to refer to each specific shape object as instances of the Shape class and when we invoke a method on the specific shape object, the specific implementation of the method associated with the shape is used. For example, if the **IsAt()** method is called, then the specific implementation to circle using the radius is used. This is the essence of polymorphism and its how objects can take on many forms.

This process promotes code ruse by allowing us to design generic code through a common base class or interface that defines the set of methods to be used and can be invoked by the specific derived classes just like the **Draw()**, **DrawOutline()**, and **IsAt()** inside shape. As such, further down the line, if we were to add more objects that adhere to the common subclass or interface into the mix, they can make use of the existing code defined from the sub class which will reduce any code duplication and provide an overall more efficient design.

Roles

You can set any method and attribute to an object, in fact there are infinite number of things and object and know and do. When you start defining objects, there a specific thing it can do and know within a domain. A Shape object can have length, width, location, and be able to draw, but it cannot honk, run or swim. Thus, it is essential that the object must adhere to its role and what it meant to do and know.

Responsibilities

Responsibilities of objects are the set of methods and attributes that the object must know, and the object can do. Each object is designed to do something within a program, thus being responsible for that certain task. An object has that duty to perform that task.

Collaborations

When multiple objects are created, sometimes we need them to interact with each other. This process means processes are shared. The interaction between objects relates to roles and responsibilities communicating with each other and sharing data.

Coupling

Coupling is a measure of how classes are related to each other. It can be broken up into low and high coupling

Low Coupling

When two classes are linked with low coupling, it means it's much easier to change any code without affecting the other class, thus providing a more efficient design as the two classes are less dependent on each other.

High Coupling

Opposite to low coupling, when two classes are linked with high coupling, it means it's much more difficult to change any code without affecting the other class, thus providing a less efficient design and making it more difficult to work with as the two classes are dependent on each other.

Cohesion

At the top we talked about roles and responsibilities and how each object has its own roles and responsibilities. Cohesion, like coupling can be measured with low and high cohesion. Essentially cohesion refers to how well a class is sticking to its roles and responsibilities and not moving away from them. Low cohesion means the class has a wide variety of method/actions and it's not focused enough to its roles and responsibilities, high cohesion means the class is focused on what it should be doing.