OOP (Object Oriented Programming)

OOP is a basically programming paradigm (i.e. a way of writing code) in which we create program that utilizes an interactions between different “objects” (entities that contain both data and functionality) to perform a certain task (i.e. build and run a game).

This paradigm follows 4 core principles: Abstraction, Encapsulation, Inheritance, and Polymorphism

Abstraction is nothing but a fancy alternative of the word “mapping”. It mainly means modeling real world elements into objects that are in a code. This is done by providing roles of the object (i.e. its purpose), its responsibilities (i.e. what it will know and do) and its collaboration (i.e. determining how it will interact with other objects in the program). Although we use similar degree of mapping while coding in general, OOP goes one step further by fully defining it into “small building blocks” and using them to make the program itself; ensuring high levels of cohesion (i.e. classes having clear purpose) and low levels of coupling(i.e. less interdependency between classes).

Encapsulation is the way the previously mentioned building blocks or “objects” are made. Each object is an instance of a class, containing fields (i.e. the type of data that it can store) and methods (i.e. the functions it can perform) stored inside it. These can be set as public (accessible to all from outside the class) or private (only accessible to things inside the class itself). Usually the fields in a class are kept as private while methods are kept as public.

Inheritance is a type of collaboration (is-a-kind relationship) used by the class. It is used when a class can be said as a child of another class. An analogy can be said as the relation between “phone” and “iPhone”, “Android” and “Nokia” phones. Here, “phone” is the parent class and the “iPhone”, “Android” and “Nokia” are the child classes as they are all a type of “phone” class with common data and functionality, albeit with slight difference in features. Thus, it’s good to keep in mind that a child class can be used to not only inherit the features of the parent class, but also override them or add new features, if need be.

This also introduces a new protection level beside public and private which is protected. This means the child or any other derived class from parent can see the methods, but no one else.

Polymorphism literally means many forms. It describes the concept that objects of different types can be accessed through the same interface (the features specified by class; the implementing classes must provide them), but each type can provide its own, independent implementation of this interface! We can check this by the “instanceof” test.

Here, it must be kept in mind that the interface and inheritance are not the same. A class can inherit form only a single class but can implement interface from many interfaces!

(Please see concept map in 7.2 for clarification if needed)

(Please turn over for references used here)

References used here:

- OOP Lectures notes of week 1 to 5
- https://searchapparchitecture.techtarget.com/definition/object-oriented-programming-OOP
- https://en.wikipedia.org/wiki/Object-oriented_programming#Encapsulation
- https://www.w3schools.in/java-questions-answers/difference-between-classes-objects/
- https://stackify.com/oop-concept-polymorphism/
- https://eng.libretexts.org/Courses/Delta_College/C_-_Data_Structures/05%3A_Polymorphism/5.4%3A_Difference_between_Inheritance_and_Polymorphism
- https://ducmanhphan.github.io/2019-03-23-Coupling-and-Cohension-in-OOP/