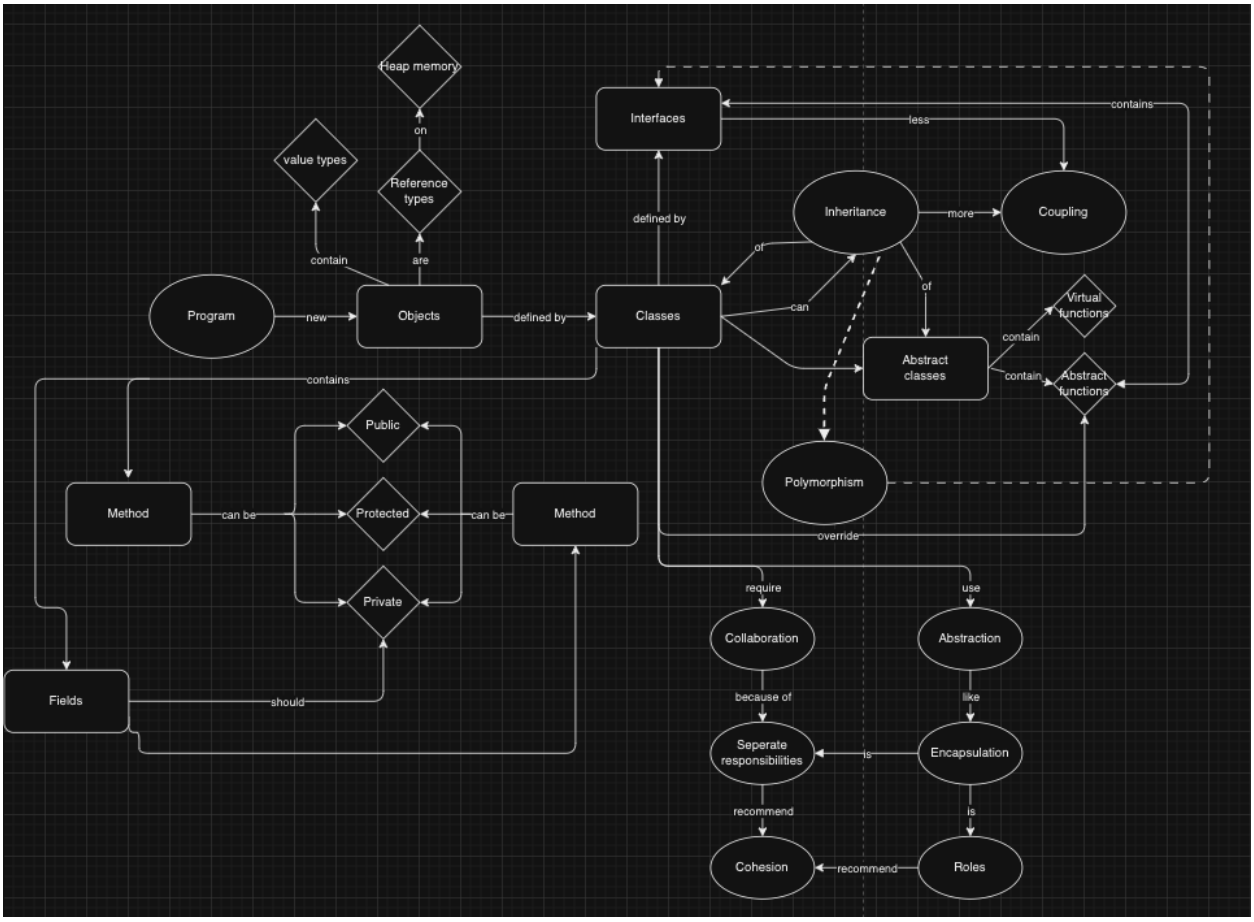
# Key Object Oriented Concepts Report

Beside from functional programming, object-oriented programming is a programming paradigm that are widely used in programming. It focuses on putting objects into the creation of software rather than using the logic and functions. This paradigm of programming is suitable for complex, up-to-date and large-scale applications as it allows programmers to alter and arrange objects. The image below briefly illustrates the crux of OOP.




As told by the name, the fundamental of object-oriented programming is objects, which is like real life objects that have capabilities and knowledges. When taking this concept to programming, programmers use class to build objects. A class owns a field of property, which is use for data representation and methods for object functionality. In simple term, a class just like a person that has their own personality and behaviors. Nowadays, many programming languages support object-oriented, some languages, including C# or Java, utilize the keyword “new” to create an object from a class and this object is treated as an instance.

OOP introduces four major principles: Abstraction, Encapsulation, Inheritance and Polymorphism

- **Abstraction**: The core principle of abstraction states that objects solely know certain things or do certain actions, allowing the specifics of the way objects learn and operate. To be exact, classifications, roles, responsibilities, and collaborations are the four essential yet foundation properties of object that helps establishing abstraction. Classifications stand for defining the properties and methods. Each object must have a function, a purpose, and the right duty to carry out its mission. The relationship between items is also identified by abstraction and this is referred to as collaborations.
- **Encapsulation**: The concept of encapsulation is the way programmers manage the information accessibilities by encapsulating properties and methods inside the object. Since users can use the functions inside an object without knowing internal implementations and changing the implementation unintentionally could leads to a collapse, the internal working of functions must be kept secret while allowing programmers to use it. This could be done by using access modifiers.
- **Inheritance**: Generally, inheritance helps creating a “child” class from a base (or parent) class, which makes child class able to inherit all properties and methods of the parent base class. To make the code reusable, programmers will follow inheritance to initialize a “family” of classes. To be able to do this, abstract class and interface are introduced to support inheritance, which support creating members without implementing any particular logic. Although abstract class is designed to be inherited, interface is born in order to make classes do not have the same additional functionality of the class family. An interface can be used in multiple base classes, a class can only have one base class.
- **Polymorphism**: Polymorphism relates to a term of a single object with many forms, which could be done via using inheritance. Specifically, a child class could carries both parent’s type and its type but the developers could change its type throughout the program. This programming paradigm has been introduced in SwinAdventure Case Study 4, which uses the Locate function to dynamically locates a range of distinct items, such as a player, a location, an inventory and a game object.
- **Roles**: Just like its name, the roles represent the objectives of an object’s methods, which includes what has been given to those methods, what to expect or what to require from those objects.
- **Responsibility**: The term responsibility represents the obligation of the objects. The missions that those objects must conduct must be equal to their responsibilities, such as handling any actions or owning some knowledges.

- **Coupling**: Coupling illustrates the dependent and the connection between two objects. Updating the code in one class would not affect the other when there is a small coupling between those classes. Oppositely, strong coupling would conflict changes between classes if there were some updated codes on one class, thus making the program hard to maintain and modify the code. Therefore, a low coupling is recommended when developing OOP programs.
- **Cohesion**: Cohesion defines the functionality of a class. A strong cohesiveness indicates that the class is strongly focused on its missions and knew what it is doing. On the other side, low cohesiveness shows that the class contains a wide range of properties and functions without any concentrates on its missions. Therefore, a high cohesion and low could should be a major criteria for a good OOP design structure.