# Task 1 – Understanding Object-Oriented Programming

A) Explain what the Object-Oriented Programming paradigm is and outline at least three benefits which make it advantageous for software development.

Object-oriented programming is a programming paradigm based on the operation around the objects. These objects can store data in the parameters with closely related code or functionality in the functions (methods). OOP is based on the idea of classes and objects, where they organise the program into basic reusable ‘blueprints’. The use of this paradigm is based on threatening the program as real-world objects, such as the user. It helps to interact with the object and build relationships.

Especially useful for big and complex projects, where collaborative development is a priority, and the project is divided into smaller groups, according to the OOP paradigm, programmers work with these smaller objects. To implement a project with the OOP paradigm, the first is needed to plan the objects and design their relationships, the process known as Data Modelling. Then the Data and Functions are combined to create the Object. Connections between several objects also could be established at this point.

Other benefits of the OOP are reusable code, scalability and efficiency, which we will consider more closely after taking closer at OOP concepts.

B) Explain the concepts of abstraction and encapsulation in Object-Oriented Programming, and outline how they contribute to creating well-structured and maintainable software systems.

**1. Encapsulation:**

Encapsulation is one of the core principles of OOP and it is a process of bundling data (parameters) and functions (methods or procedures) into a single object. Encapsulation intends to hide the internal state of an object from the outside. Access to the ‘private’ data parameters is implemented through the ‘public’ methods, which do not allow for changing the parameters directly, but only through these public functions with proper control and validation. Encapsulation highly enhances security by preventing accidental or unauthorized modifications to the data.

**Real-world example:** Imagine we're creating a software system for a training centre that allows us to manage a list of students in various courses. This system should be able to add new students, update student details, and remove students from the course. We will use encapsulation to achieve these functionalities securely and efficiently.

* Student Class: Represents each student with private attributes such as student\_id, first\_name, last\_name, age, fees\_due and course. It also includes methods to get and set these attributes, ensuring that any changes to the student's details are controlled and validated.
* StudentManager Class: This class manages all students within a course. It includes a private list that holds the students. The StudentManager provides public methods to interact with this list.

**2. Abstraction**

Abstraction is the second fundamental principle of OOP that focuses on exposing only the key features of an object while hiding unnecessary details and background functionality. It allows us to handle complexity by presenting at a more simple level rather than overloading with the details. In simple terms, abstraction is about identifying what an object does rather than how it achieves what it does. The key benefits of abstraction are simplicity, focus on what matters and flexibility.

**Real-world example:** Let's continue with the scenario of creating a software system for a training centre, focusing on the management of a list of students in various courses.

* **Simplicity:** The StudentManager class simplifies the complexity of managing students by providing a high-level interface. Users of the StudentManager do not need to know how students are stored or how the operations are implemented.
* **Focusing on the Key Features:** By abstracting the details of Student Management, the system allows users to focus on what actions can be performed (add, edit, remove) rather than how these actions are executed. This separation of concerns makes the system easier to understand and use.
* **Flexibility:** The abstract of the StudentManager allows the underlying implementation of how students are managed to change (e.g., moving from a list to a database) without affecting the rest of the system. This flexibility is a direct benefit of abstraction.

**3. Polymorphism**

Polymorphism, a third core concept in OOP, enables objects of different classes to be treated as objects of a common superclass. It allows methods to do different things based on the object it is acting upon, even though they share the same name. This is achieved through two main types: compile-time (or static) polymorphism and runtime (or dynamic) polymorphism. Compile-time polymorphism is achieved through method overloading, while runtime polymorphism is achieved through method overriding.

**4. Inheritance**

Inheritance is the final main concept in OOP that allows a class to inherit properties and methods from another class. The class that inherits is known as the subclass (or derived class), and the class from which it inherits is known as the superclass (or base class). Inheritance facilitates code reusability, enabling new objects to take on existing properties and behaviours of other objects while introducing their unique features.

# Task 2 – Designing an Object-Oriented Programming solution

Create a UML diagram for the design of a basic bank account. The design should account for two distinct types of accounts, savings accounts and current accounts. Current accounts will have set transaction fees which will apply to all withdrawals. Savings accounts will have a set interest rate which will be used to calculate interest for the account (balance \* interest rate). Below is a breakdown of the attributes and functionality which should be represented by your class design.

asd