Abstraction simplifies complex concepts by focusing only on essential details, creating modular and reusable code that is easier to understand and use. Encapsulation compartmentalizes data into classes, carefully controlling access to methods and properties to prevent unnecessary exposure of internal details. Inheritance allows classes to inherit attributes and behaviours from parent classes, promoting code reuse and simplification by avoiding redundant implementations. Polymorphism enables the same code to behave differently in various contexts, enhancing code flexibility and reusability. Together, these principles empower developers to create diverse programs efficiently, with modular, reusable, and maintainable codebases that can adapt to evolving requirements without extensive modifications.

These principles were applied on my final project by the following way:

***Abstraction*** - The classes “Video” and “Comment” encapsulate the details of videos and comments by providing the essential information to the outside world through properties. The way abstraction is achieved is by exposing the necessary attributes such as “Title”, “Author”, etc. By hiding the unnecessary details, the code becomes easier to understand and to be reused.

***Encapsulation*** – The classes “Product”, “Customer”, “Address” and “Order” encapsulate related behaviours into single unit classes. Encapsulations restricts the direct access from the outside promoting data integrity.

***Inheritance*** – The classes “Lecture”, “Reception” and “Outdoor” inherit common attributes from the base class “Event”. It promotes code reusage by allowing child classes to inherit and extend the functionality of the parents’ classes. Child classes override methods from parent classes, acquiring specialized behaviours, enhancing code flexibility.

***Polymorphism*** – The classes “Running”, “Cycling” and “Swimming” inherit from the base class “Activity” and provide their implementations of methods such as “GetDistance()”, “GetSpeed()” and “GetPace()”. It enables dynamic method invocation based on the actual type of object at runtime. The “Main” method calls overridden method, demonstrating polymorphic behaviour.

Using the four principles of programming with classes helped the final project to become more flexible for future changes, in several different ways.

***Abstraction:*** In the Video and Comment system, abstraction is used by exposing only necessary properties and methods such as “Title”, “Author”, etc. Developers can use these classes without worrying with the internal details.

***Encapsulation:*** It restricts direct access to some of the object’s components, which can prevent the accidental modification of data. For instance, the “Product”, “Customer”, “Address”, and “Order” classes have private fields and public properties/methods for interaction.

***Inheritance:*** Inheritance allows a class (child) to inherit attributes and methods from another class (parent). This promotes code reuse and a hierarchical classification. In the Event Planning system, the “Lecture”, “Reception”, and “Outdoor” classes inherit from the Event base class.

***Polymorphism:*** It enables methods to be defined in a base class and overridden in derived classes. In the Activity tracking system, the Running, “Cycling”, and Swimming classes inherit from the “Activity” base class and provide their own implementations of “GetDistance”, “GetSpeed”, and “GetPace” methods. This gives the developer great flexibility and dynamic behaviour.

***Conclusion***

Overall, using these principles made the project scalable, maintainable, reusable, and extensible. Adding new event types, products, or activities would require minimal changes. By adhering to these principles, the project remains flexible and adaptable to future requirements.