**Abstraction**

Abstraction in programming is representing complex ideas in a simplified way by identifying and separating the essential characteristics of an object or system, leaving aside unnecessary details. This ability allows programmers to focus on relevant aspects of the problem, developing more efficient and scalable solutions. In addition, hiding complexity behind simpler interfaces results in code that is more maintainable and easier to understand. The additional advantage of abstraction is that it facilitates the creation of reusable software components in different contexts, making code modification and understanding easier.

This principle was used in my project because the idea of keeping track of the comments or list of comments for each video was becoming complex. Therefore, through Abstraction I thought of simplified solutions that could help me to make the program achieve its goal. What I did was to focus on what each video should have and how it should be shown, so I made a class called Video, Comments, etc. This helped me separate the necessary details from the unnecessary ones.

**Encapsulation**

Encapsulation in programming is based on the principle of Abstraction and consists of hiding the internal details of a class, allowing access to data and behaviors only through public methods. This provides a safe and standardized way of interacting with the data and ensures that any problem or change in the implementation affects only the code that has access to the class, avoiding external modifications through private methods.

This other principle was used in Foundation 4. This program has some methods that should not be visible with the rest of the program, as it is with the ADDSHIPCOST, CREATEPACKINGLABEL, etc. methods. This allowed me to have better control of the class components while working outside the classes.

**Inheritance**

Inheritance in programming is the ability of a more specific class, called a "Derived Class", to inherit attributes and methods from another more general class, known as a "Base Class". This is based on an "Is a" relationship, where a Derived Class is a type of the Base Class. For example, a "Dolphin" is a "Mammal". Inheritance allows us to create new classes from a base class, which facilitates code reuse and generates a hierarchy in the application. This allows us to inherit components from a base class to other classes, and the base class can add, use, or redefine the inherited methods, which provides an additional benefit in code reuse and flexibility.

This principle helped me to avoid redundancy when creating objects. In Foundation 3 there were 3 types of events with similar characteristics, and what I did was to inherit attributes from a Parent class to Daughter classes, this helped me to write less code and to keep a better order of the code already written.

**Polymorphism**

Polymorphism, a key concept in object-oriented programming, derives from the Greek words "poly" (many) and "morph" (forms), signifying the quality of having multiple forms. It enables an object or line of code to exhibit different behaviors based on the context in which it is utilized. For effective application, polymorphism relies on the principles of abstraction, encapsulation, and inheritance.

Derived classes can inherit attributes and methods from their superclass, and polymorphism permits them to override or modify the behavior of inherited methods, a process known as method overriding.

This principle helped me to avoid redundancies when creating objects. In Foundation 3 there were 3 types of events with similar characteristics, and what I did was to inherit attributes from a Parent class to the child classes, this helped me to write less code and to keep a better order of the code already written.

**Explain how these principles make your final project more flexible for changes.**

The usefulness of using programming principles such as abstraction, encapsulation, inheritance, and polymorphism, allows me and will allow me to make changes because each program is divided by tasks, each method does a specific task, each object has its specific methods and each class has constructors, attributes, etc. The fact that the program is divided into different parts helps me to find scalable solutions faster, also in case an object does not work, we can be sure that the problem lies within that class and that is thanks to the encapsulation. If an object were to change its method, we would go directly to check how the class is or if any method has been overwritten, this can also be done by applying polymorphism principles. These principles help my program to be much more flexible for changes.