Programming with Classes – Articulate

Name: Roberto Vergilio Soares Junior

Abstraction:

Meaning: Abstraction involves identifying and separating the essential aspects of an object from its specific context, focusing only on the relevant details for the problem at hand.

Application in the project: In the final project, abstraction can be applied when creating classes that represent specific entities and functionalities, such as the Despesa and DespesaManager classes. These classes encapsulate the internal details of their respective functionalities, allowing the user to interact with them without needing to understand how they work internally.

Encapsulation:

Meaning: Encapsulation involves hiding the internal details of a class and providing a public interface to interact with it. This allows the internal state of an object to be protected and controlled through public methods.

Application in the project: In the final project, encapsulation is applied in the Despesa and DespesaManager classes, where attributes are kept private and accessed only through public methods like GetValor(), GetData(), and GetCategoria().

Inheritance:

Meaning: Inheritance allows a class to inherit attributes and methods from another class, promoting code reuse and the creation of class hierarchies.

Application in the project: In the final project, inheritance can be applied if you extend the functionality to handle different types of expenses. For example, you can have DespesaFixa and DespesaVariavel classes that inherit from the Despesa class, sharing common attributes and methods.

Polymorphism:

Meaning: Polymorphism allows objects of different classes to be treated uniformly, so that methods with the same name can have different behaviors depending on the type of the object.

Application in the project: In the final project, polymorphism can be applied if you create common methods that operate on a list of expenses, where each type of expense (e.g., DespesaFixa and DespesaVariavel) has specific behaviors for calculations or reports.

How these principles make your final project more flexible for future changes:

Abstraction allows you to adjust internal implementation details without affecting the public interface, facilitating code maintenance and evolution.

Encapsulation protects the internal state of classes, allowing you to change the internal implementation without affecting the rest of the code that depends on it.

Inheritance facilitates code extension, allowing you to add new functionalities or types of expenses without modifying existing code.

Polymorphism allows you to write code that works with objects of different types, making your code more generic and adaptable to changes in project requirements.

In summary, the application of these principles makes your final project more modular, cohesive, and flexible for future changes, promoting better organization and maintenance of the code over time.