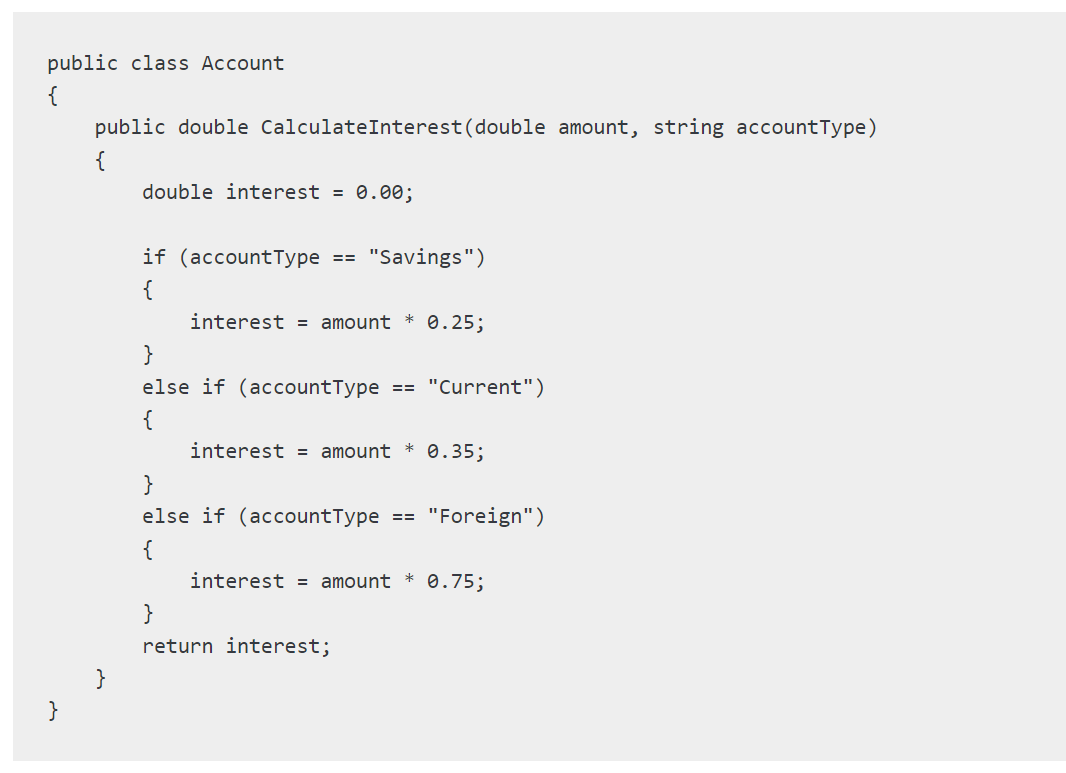
Open / Closed principle **open for extension**” and “**closed for modification**.”

---------------------------------

Bad code:



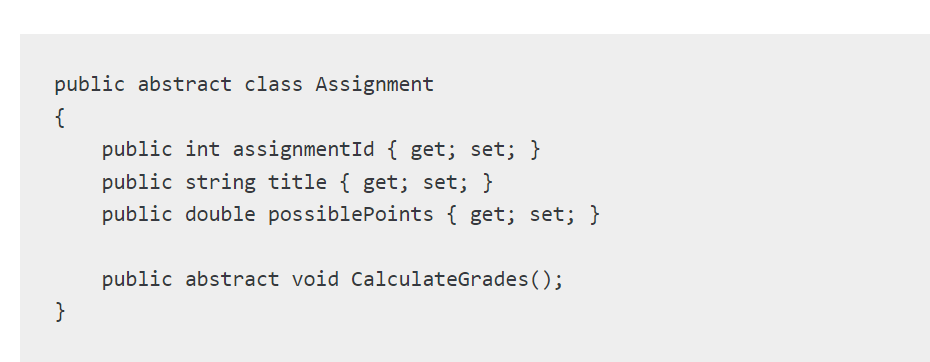
* If we need to modify

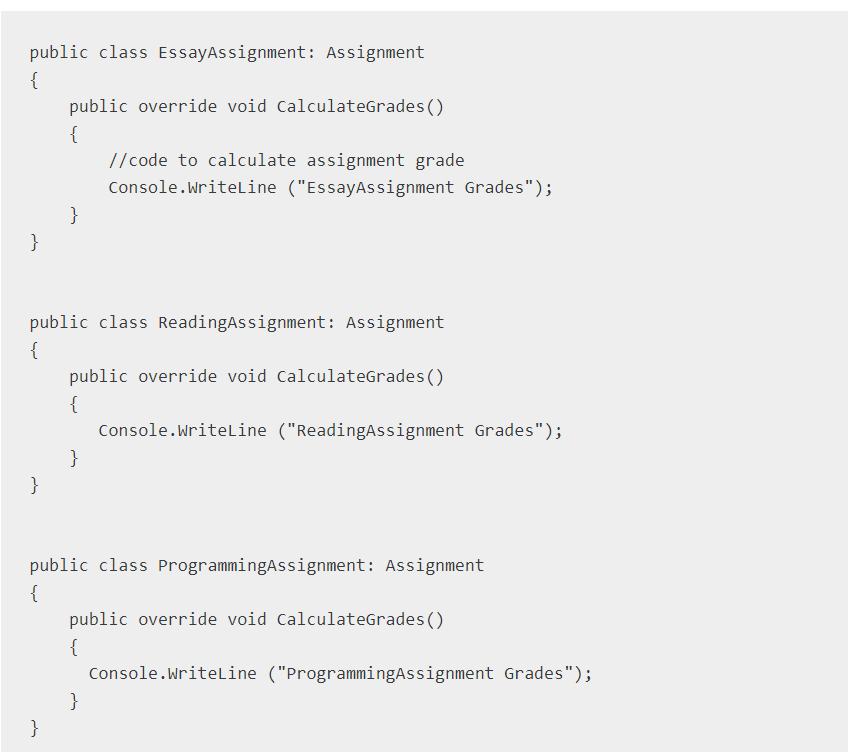
Elseif { } --- Elseif { }

Its to close for any modify . Only open to extension

So we solve it -->

**Good code**





Or use interface .

## **Benefits of using the Open/Closed principle in C#**

The following are benefits of the Open Closed Principle:

1. **Improved code quality and maintainability** – The code that follows OCP is less error-prone when introducing new behaviors. Also, since such codes are modularized, they are easy to unit tests. These characteristics largely contribute to higher code quality and maintainability.
2. **Increased code reuse** – Using OCP removes duplicated codes packing the common codes into reusable classes, modules, functions, etc., that can be extensible and customized to have different behaviors.
3. **Better separation of concerns** – It promotes the Single Responsibility Principle so that each class, function, or module has exactly one well-defined responsibility. Also, it encourages you to use abstractions and interfaces to separate the concerns into separate components.
4. **Enhanced extensibility** – OCP allows code to be open for extension without the need to modify, rewrite, or refactor the existing code. This ability lets developers easily extend the software application to have enhanced features.

Some hints:

When it comes to applying the Open-Closed Principle (OCP) to models in code, you can follow a few best practices to ensure that your software entities are open for extension but closed for modification. Here's how you can apply the OCP to models:

1. Use an abstract base class or interface: Create an abstract base class or interface that defines the common behavior and properties shared by all models. This base class should provide a clear contract for how models should be implemented.
2. Implement specific models as derived classes: Create derived classes that inherit from the base class or implement the interface. Each derived class represents a specific model implementation. These derived classes should provide the necessary details and logic specific to each model.
3. Encapsulate model-specific behavior: Encapsulate the behavior and logic specific to each model within its corresponding derived class. This ensures that modifications to existing models are limited to their respective classes, minimizing the impact on other parts of the codebase.
4. Use polymorphism to extend functionality: Leverage polymorphism to extend the behavior of models. Define additional methods or properties in the base class or interface that can be implemented differently in each derived class. This allows you to add new functionality to models without modifying the existing code.
5. Use dependency injection: Instead of directly instantiating specific model classes, use dependency injection to inject the appropriate model implementation into the code that depends on it. This allows you to easily replace or extend models without modifying the client code.

By following these guidelines, you ensure that your models can be extended to accommodate new requirements without modifying the existing code. This promotes code reusability, maintainability, and modularity, which are key principles of the OCP.

Resources:

https://methodpoet.com/open-closed-principle/#:~:text=The%20Open%2DClosed%20Principle%20(OCP,functionalities%20when%20introducing%20new%20features.