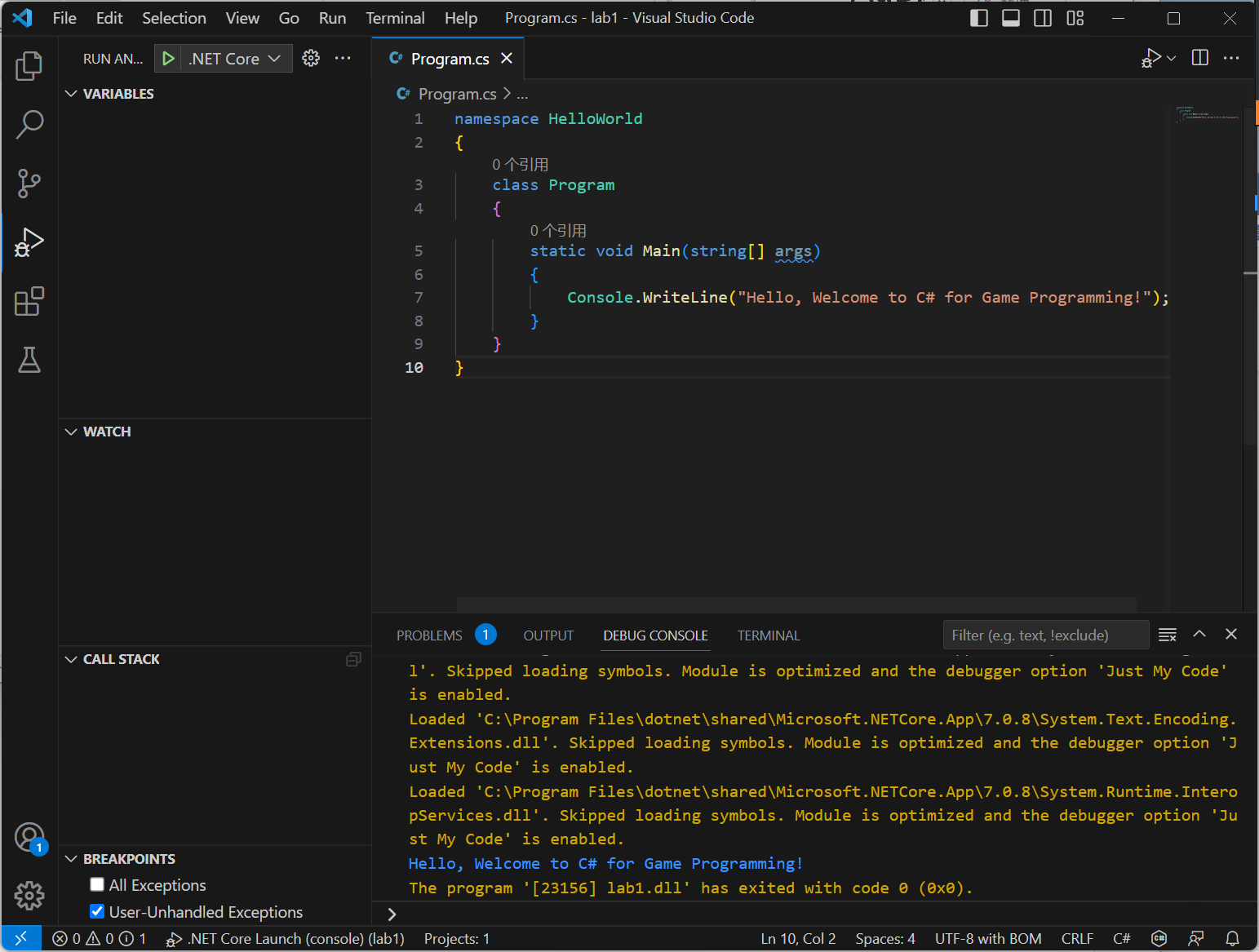
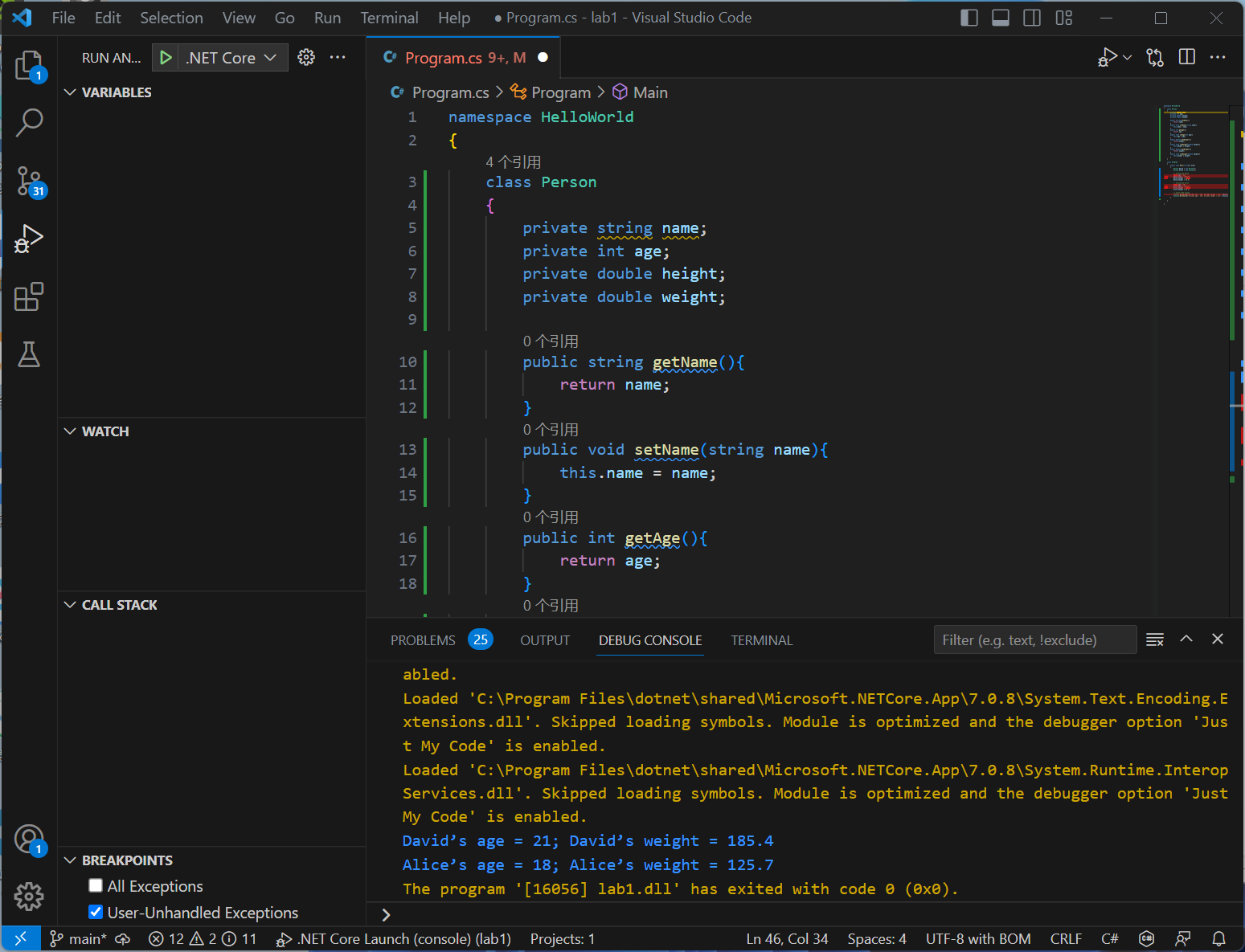
**Task1.1**

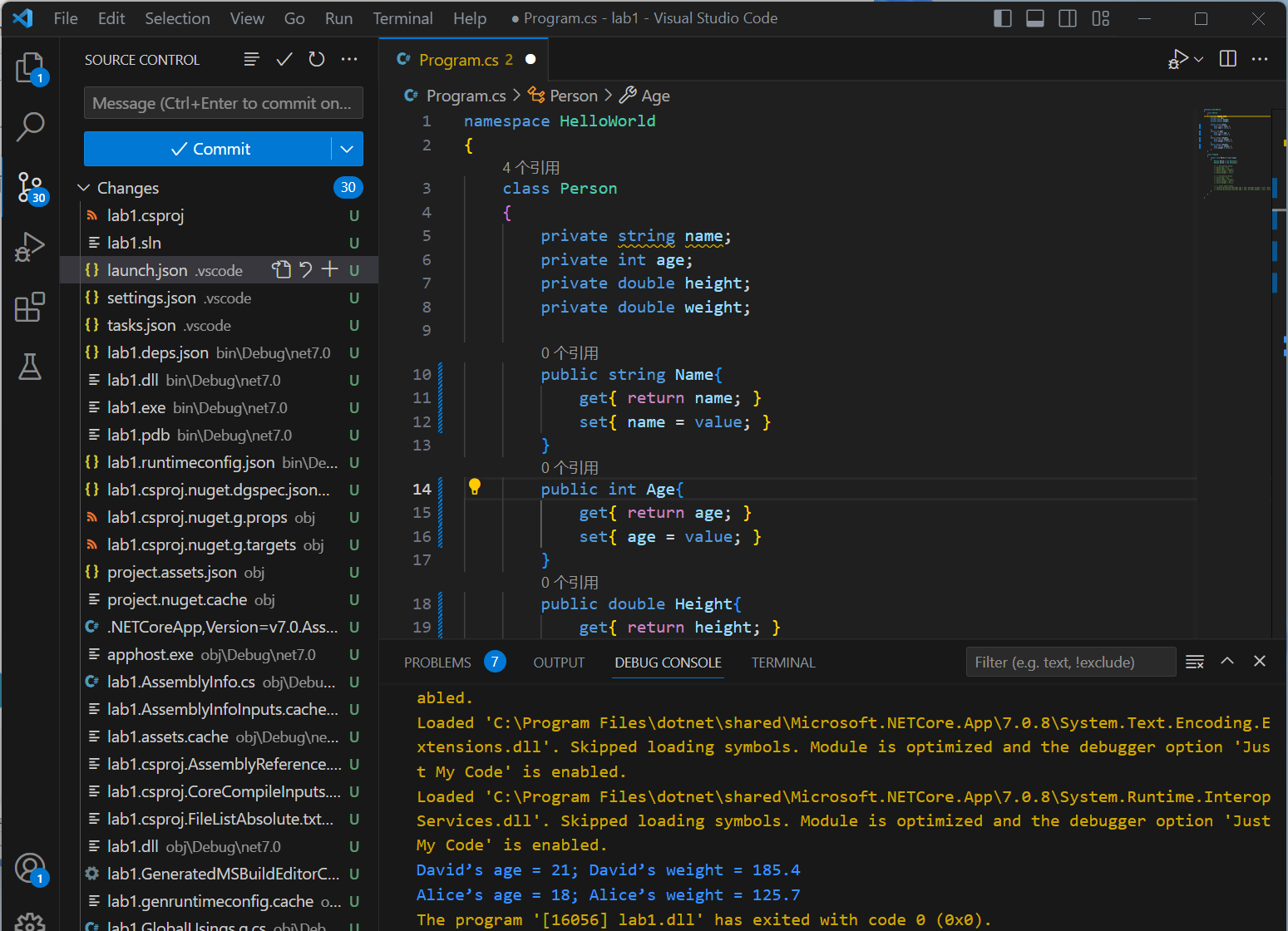
As shown in the figure below, the sentence we need appears in the console. Also, the output window doesn't disappear anytime soon, but is embedded in the VScode interface.



**1.2**



**1.3**



**1.4**

**(1) What's the purpose/role of a constructor?**

The purpose of a constructor is to initialize an object when it is created. It is a special method that is automatically called when an object is instantiated. Constructors are used to perform necessary initialization operations, such as setting initial values, allocating memory, or executing other required actions. Constructors ensure that objects are in a proper state before they are used.

**(2) What's the default constructor?**

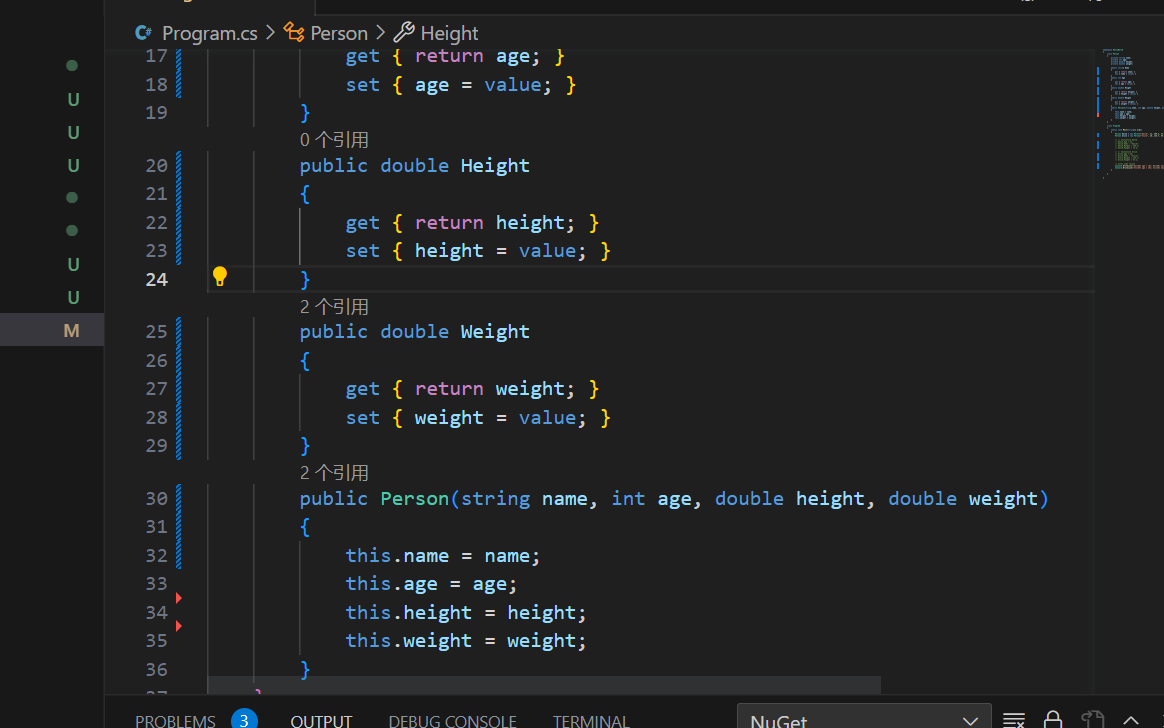
The default constructor is a parameterless constructor. It is automatically provided if no constructors are explicitly defined for a class. The default constructor takes no arguments and is typically used to create instances of the class. It performs some default initialization operations, such as setting fields to default values.

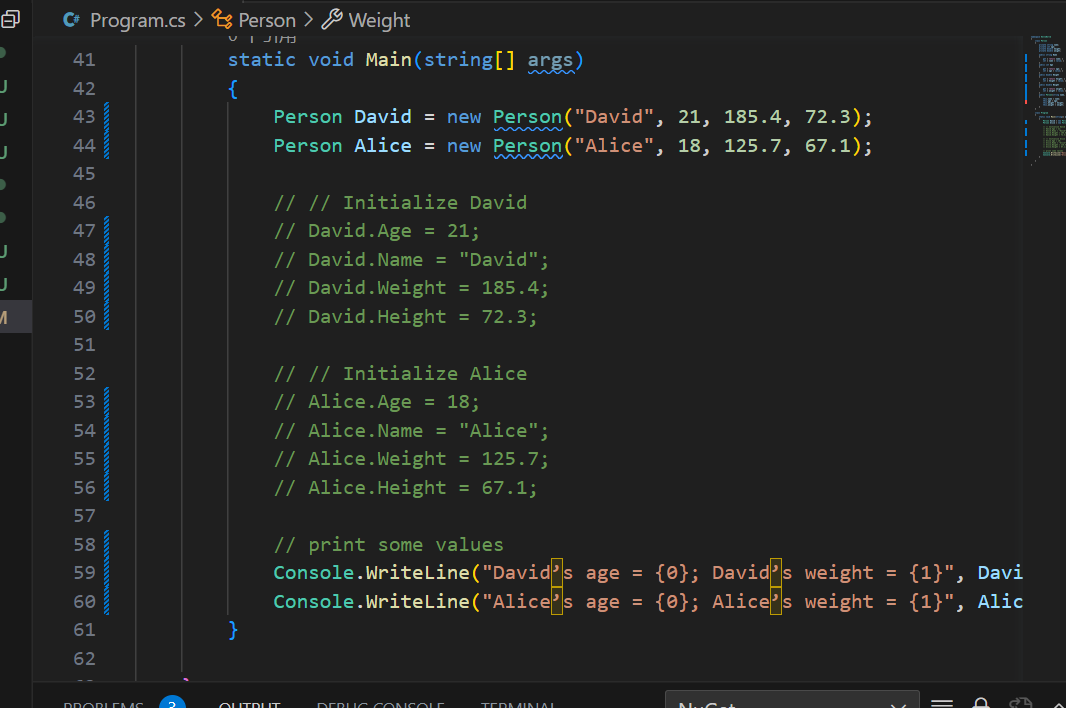
**(3) Do you (as the programmer) have to provide a constructor? If not, what happens?**

As a programmer, you can provide constructors as needed. If you do not explicitly provide any constructors, the compiler will automatically provide a default constructor as long as there are no other constructors defined. However, if you define at least one constructor, the compiler will no longer generate a default constructor.

If you define your own constructors and do not provide a default constructor, there won't be a parameterless constructor available when instantiating an object. This means that if you attempt to instantiate the class object without passing any arguments, the compiler will raise an error. Therefore, if you want to allow parameterless instantiation of the object, you need to explicitly provide a parameterless constructor.

**1.5**





**1.6**

**(1) What's the purpose to define an abstract class?**

The purpose of defining an abstract class is to provide a common blueprint or template for deriving other concrete classes. An abstract class itself cannot be instantiated; it is used to define a set of related methods, properties, and events as a base for other classes. Abstract classes can contain both concrete implementations and abstract members, providing a way to enforce necessary functionality in derived classes.

**(2) What happens if you try instantiating an abstract class?**

If you try to instantiate an abstract class, it will result in a compilation error. An abstract class is incomplete itself as it contains abstract members that need to be implemented in the derived classes. Therefore, you cannot create an instance of an abstract class directly; it can only be instantiated through derived classes.

**(3) Why is the property 'Area' defined as 'abstract'? And why is it read-only, i.e., why is only a 'get' accessor needed?**

The property 'Area' is defined as 'abstract' to require derived classes to provide their own implementation. The 'Area' property represents the area of a shape, but the specific calculation varies depending on the type of the shape. By defining it as an abstract property, it ensures that each derived class must provide its own implementation of 'Area'.

The 'Area' property is read-only because it only provides a 'get' accessor. This is because the area is calculated based on the characteristics of the shape and once calculated, it should not be changed. Therefore, only a 'get' accessor is needed to retrieve the area value, and there is no need for a 'set' accessor.

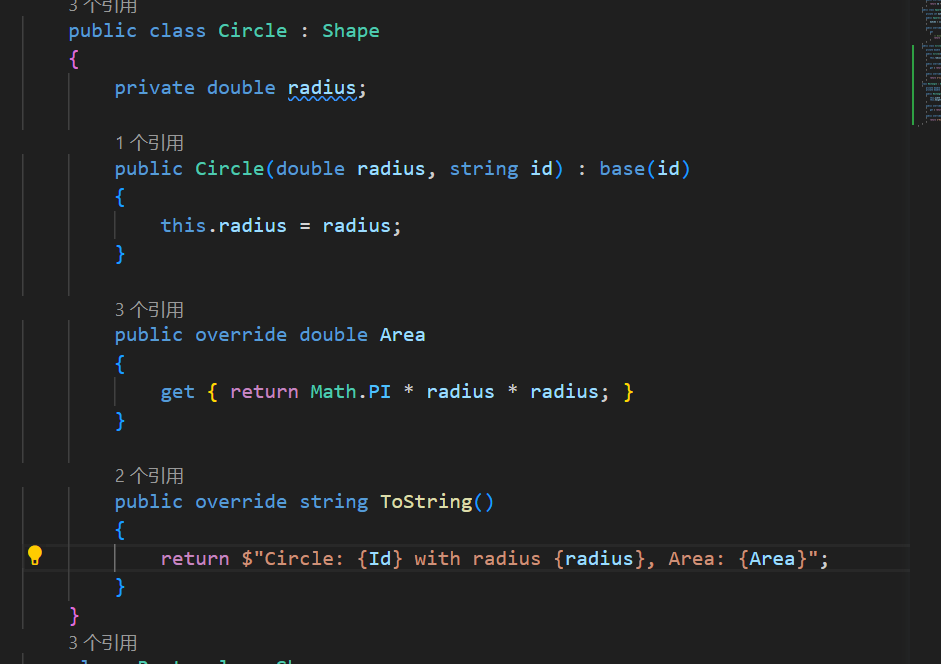
**(4) What's the purpose of the keyword 'override' for the method 'ToString()'?**

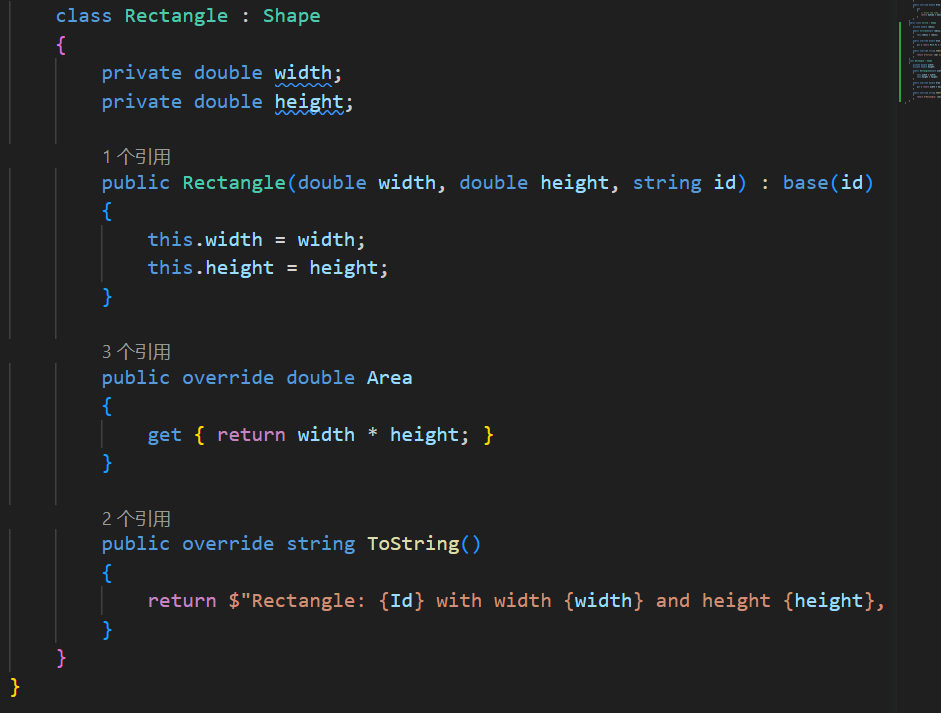
The keyword 'override' is used to indicate that a derived class is overriding (replacing) a base class's method with its own implementation. In the code, the derived class overrides the 'ToString()' method of the base class 'Shape'.

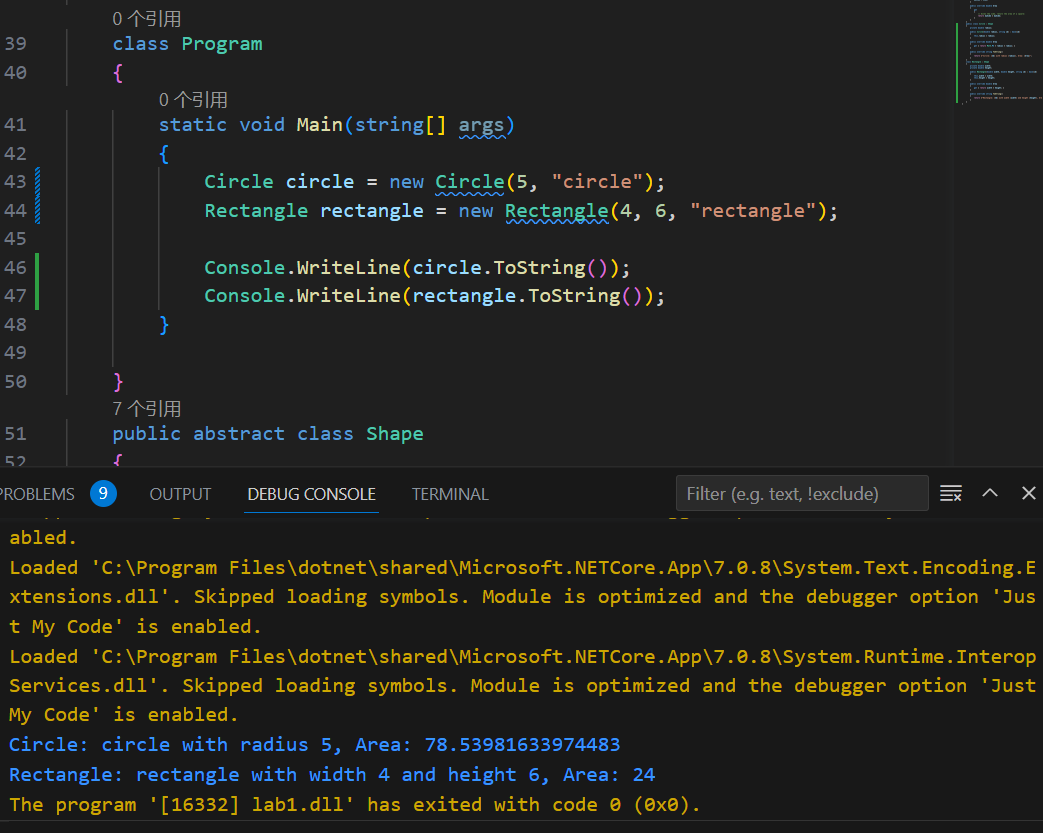
If the 'override' keyword is not used, the method in the derived class will be treated as a completely new method rather than overriding the base class method. This means that when calling the base class's 'ToString()' method, it will actually invoke the base class's implementation instead of the derived class's implementation. Using the 'override' keyword ensures that the base class method is correctly overridden, allowing the derived class's implementation to be called in the derived class.

**1.7**

**(1) Add another two shapes (classes), ‘Circle’ and ‘Rectangle’, to the programme above, which inherits from ‘Shape’, and modify ‘main()’ to print relevant information about them**







**(2) This programme demonstrates another most important characteristics of object oriented programming. What do we call it?**

The program showcases another crucial aspect of object-oriented programming known as inheritance. Inheritance allows a class to inherit properties and methods from another class, and it enables the derived class to add new functionality or modify the inherited functionality. In this example, the 'Circle' and 'Rectangle' classes inherit the properties and methods of the 'Shape' class while providing their own implementations. Through inheritance, we can better organize and reuse code, enhancing the maintainability and extensibility of the codebase.