03 Object-Oriented Programming  
Test your knowledge  
1. What are the six combinations of access modifier keywords and what do they do?

public: This access modifier makes a class, method, property, or field accessible from any other class, regardless of namespace.

private: This access modifier restricts access to a class, method, property, or field to only within the same class.

protected: This access modifier allows access to a class, method, property, or field within the same class or subclass, but not outside the class hierarchy.

internal: This access modifier allows access to a class, method, property, or field within the same assembly, but not from outside the assembly.

protected internal: This access modifier combines the protection of "protected" with the accessibility of "internal". A class, method, property, or field declared with this modifier is accessible within the same assembly and also within a subclass in a different assembly.

private protected: This access modifier is a combination of "private" and "protected", allowing access to a class, method, property, or field within the same class and within a subclass in the same assembly.

2.What is the difference between the static, const, and readonly keywords when applied to a type member?

const is used for values that will not change during the execution of the program, such as mathematical constants or enumerations. readonly is used for values that may change between instances of the class or between runs of the program, but should not be changed once set. static is used for values that are shared among all instances of the class and need to persist between method calls.

3. What does a constructor do?

Create an instance of the class, initialize the object's state and perform any necessary setup.

4. Why is the partial keyword useful?

The partial keyword in C# is useful because it allows a class, struct, or interface to be split into multiple parts in separate files, but still maintain the same type. Each part can contain methods, fields, properties, and events, and the parts are combined into a single type at compile time.

5. What is a tuple?

a data structure that allows you to store multiple values of different types within a single object

6. What does the C# record keyword do?

Records provide a simple and clean syntax for creating value-based classes

7. What does overloading and overriding mean?

Overloading: creating multiple methods with the same name, but with different parameters

Overriding allows you to provide a new implementation for a method that is already defined in a base class

8. What is the difference between a field and a property?

Fields: Fields are the basic member variables of a class. They hold the state of an object and can be accessed directly from within the class or from outside the class using an instance of the object. Fields can be public, private, or protected, and their access can be controlled by using access modifiers.

Properties: Properties are the more advanced and sophisticated way of declaring variables in a class. They are used to control access to the data stored in a field and can be used to enforce data validation and data transformation rules. Properties can be thought of as a combination of fields and methods that can be accessed as if they were fields, but actually they are methods that get and set the underlying fields.

9. How do you make a method parameter optional?

In C#, a method parameter can be made optional by assigning a default value to the parameter in the method definition. When a method is called and no value is provided for that parameter, the default value will be used instead.

10. What is an interface and how is it different from abstract class?

An interface is a blueprint for a class, defining a set of members (methods, properties, and events) that the class must implement. An interface provides a common set of members for multiple classes, allowing objects of different types to be treated the same way in certain contexts.

An abstract class, on the other hand, is a class that can contain both abstract and concrete members, and provides a common implementation for its derived classes. Unlike an interface, an abstract class can contain implementation details, and its members can have accessibility (such as public, private, etc.).

11. What accessibility level are members of an interface?

By default, members of an interface in C# are public. All members of an interface must be public because interfaces provide a blueprint for other classes to implement and their members must be accessible to those classes.

It is not possible to specify a different accessibility level for members of an interface. The idea behind this is that the members of an interface define a contract that must be followed, and accessibility level other than public would limit the usability of the interface.

12. True/False. Polymorphism allows derived classes to provide different implementations of the same method.

13. True/False. The override keyword is used to indicate that a method in a derived class is providing its own implementation of a method.

14. True/False. The new keyword is used to indicate that a method in a derived class is providing its own implementation of a method.

15. True/False. Abstract methods can be used in a normal (non-abstract) class.

16. True/False. Normal (non-abstract) methods can be used in an abstract class.

17. True/False. Derived classes can override methods that were virtual in the base class.

18. True/False. Derived classes can override methods that were abstract in the base class.

19. True/False. In a derived class, you can override a method that was neither virtual non abstract in the base class.

20. True/False. A class that implements an interface does not have to provide an  
implementation for all of the members of the interface.

21. True/False. A class that implements an interface is allowed to have other members that aren’t defined in the interface.

22. True/False. A class can have more than one base class.

23. True/False. A class can implement more than one interface.

Designing and Building Classes using object-oriented principles  
1. Write a program that that demonstrates use of four basic principles of  
object-oriented programming /Abstraction/, /Encapsulation/, /Inheritance/ and  
/Polymorphism/.

2. Use /Abstraction/ to define different classes for each person type such as Student and Instructor. These classes should have behavior for that type of person.

3. Use /Encapsulation/ to keep many details private in each class.

4. Use /Inheritance/ by leveraging the implementation already created in the Person class to save code in Student and Instructor classes.  
5. Use /Polymorphism/ to create virtual methods that derived classes could override to  
create specific behavior such as salary calculations.  
6. Make sure to create appropriate /interfaces/ such as ICourseService, IStudentService,  
IInstructorService, IDepartmentService, IPersonService, IPersonService (should have  
person specific methods). IStudentService, IInstructorService should inherit from  
IPersonService.  
Person:  
Calculate Age of the Person;  
Calculate the Salary of the person, Use decimal for salary;  
Salary cannot be negative number;  
Can have multiple Addresses, should have method to get addresses.  
Instructor:

Belongs to one Department and he can be Head of the Department;  
Instructor will have added bonus salary based on his experience, calculate his  
years of experience based on Join Date.  
Student:  
Can take multiple courses;  
Calculate student GPA based on grades for courses;  
Each course will have grade from A to F.  
Course:  
Will have list of enrolled students.  
Department:  
Will have one Instructor as head;  
Will have Budget for school year (start and end Date Time);  
Will offer list of courses.

Github Link:

https://github.com/Saki-Meow/SEPHomework