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

Private: the element used only in the current class and can’t access by outside code

Protect: can be accessed by class and class derived from class

Internal: can be accessed within the current project

Public: can be accessed everywhere

Private Internal: the element used only in the current class and can’t access by outside code in the current project

Protect Internal: can be accessed by class and class derived from class in the current project

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

Static shared by all object

Const and readonly are keywords that value can’t be change

Readonly is runtime and const is compile time

3. What does a constructor do?

Run when the class instantiated, most of the time, class value in the class needed by initialized will be initialized in here

4. Why is the partial keyword useful?

Partial keyword indicates that this block code is only part of a class and other parts of this class could be declared in other places. Partial keywords can let people work on a big project simulator.

5. What is a tuple?

Tuple is a store element object that can let us store 1-8 different type elements

6. What does the C# record keyword do?

Record gives a reference type for building in functionality for encapsulating data.

7. What does overloading and overriding mean?

Overload and overriding are for polymorphism. Overload is polymorphism used in a class while override is cross classes. If we want to create some functions that have the same name but different behavior. We can do overload. We want to rewrite a class inherited from the superclass. We will do override

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

Field is valuable in a class. Property is the set and get function to access field.

9. How do you make a method parameter optional?

We can set default value for optional parameters. However, all of them need be the latest parameters

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

Abstract contains states however interface can’t. Interface is used for common functions.

11. What accessibility level are members of an interface?

public

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

True

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.

True

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.

False

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

False

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

True

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

True

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

True

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

True

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

False

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

True

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

False

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

True

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/.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Assignment3

{

internal abstract class Courses

{

public abstract string ? Name { get; set; }

public abstract int ClassId { get; set; }

const int classType = -1;

}

class InDoorClass: Courses

{

const int classType = 0;

public override int ClassId { get; set;} = 0;

public override string ? Name { get; set; }

}

class Math : InDoorClass

{

int studentNumber;

//overloading

public Math() {

studentNumber = 0;

}

//overloading

public Math(int studentNumber)

{

this.studentNumber = studentNumber;

}

}

class OutDoorClass : Courses

{

const int classType = 1;

public override int ClassId { get; set; } = 1;

public override string? Name { get; set; }

}

}

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.

internal abstract class People

{

public abstract void DoSomeThing();

}

class Student : People

{

public override void DoSomeThing()

{

Console.WriteLine("I'm a student");

}

}

class Teacher : People

{

public override void DoSomeThing()

{

Console.WriteLine("I'm a Teacher");

}

}

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

internal abstract class People

{

public string? Name { get; set; }

public abstract void DoSomeThing();

}

4. Use /Inheritance/ by leveraging the implementation already created in the Person class to save code in Student and Instructor classes.

internal abstract class People

{

public string? Name { get; set; }

public abstract void DoSomeThing();

public void Type()

{

Console.WriteLine("I'm a persion");

}

}

5. Use /Polymorphism/ to create virtual methods that derived classes could override to create specific behavior such as salary calculations.

People people = new Student();

people.DoSomeThing();

people.GetType();

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

using System;

using System.Collections.Generic;

using System.Data;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

abstract class Person

{

public DateTime BirthDate { get; set; }

public int GetAge()

{

DateTime currDate = DateTime.Now;

var age = currDate - BirthDate;

return (int)(age.Days % 365.2425);

}

public double Salary { get; set; }

public abstract double GetSalary();

public List<string> Addresses { get; set; } = new List<string>();

}

class Instructor : Person

{

public bool IsHead { get; set; }

public string? Department { get; set; }

public DateTime JoinDate { get; set; }

public override double GetSalary()

{

DateTime currDate = DateTime.Now;

var age = currDate - JoinDate;

return Salary + age.Days;

}

}

class Student : Person

{

public List<Tuple<string, double, double>> courses { get; set; } = new List<Tuple<string, double, double>>();

public override double GetSalary()

{

return Salary;

}

public double GetGPA()

{

double totalGPA = 0;

double currGPA = 0;

foreach (Tuple<string, double, double> t in courses)

{

currGPA = (currGPA \* totalGPA + t.Item2) / (totalGPA + t.Item3);

totalGPA += t.Item3;

}

return currGPA;

}

public String GetCourseGPA(string courseName)

{

var course = courses.First(x => x.Item1.Equals(courseName));

double currGPA = course.Item3;

if (currGPA == 4.0)

{

return "A";

}

else if (currGPA >= 3.0)

{

return "B";

}

else if (currGPA >= 2.0)

{

return "C";

}

else if (currGPA >= 1.0)

{

return "D";

}

else

{

return "F";

}

}

}

class Course

{

public List<Student> StudentList { get; set; }=new List<Student>();

}

class Department

{

public Instructor? Head { get; set; }

public List<Course> CourseList { get; set; } = new List<Course>();

public List<Tuple<Double,DateTime,DateTime>> Buget { get; set; } = new List<Tuple<Double,DateTime,DateTime>>();

}