**Object-Oriented Application Development**

**Practical 7**

**Part A**

1. Fill in the blanks.

1. Packaging data attributes and behaviors into a single unit describes an object-oriented feature called encapsulation.
2. Polymorphism is useful in languages and can be achieved using override methods.
3. If you want to enable derived classes to access base class data members, the data members in the base class should be defined with access modifier protected.
4. To enable derived classes to override methods defined in a base class, methods of the base class should be defined using keyword virtual.
5. A class from which an object cannot be instantiated is defined with keyword abstract.

2. Choose the appropriate answer.

(a) Classes can extend or be derived from \_\_\_\_\_\_ class(es) and implement \_\_\_\_\_\_ interface(s).

A. one, one

B. many, one

C. many, many

D. one, many

(b) Consider the following definition:

class A : B, IA

{

. . .

}

Which of the following statements is true?

1. IA is an interface.
2. A is the derived class.
3. B is the base class.
4. All of the above.
5. None of the above.

(c) Abstract classes can include:

A. data members

B. abstract methods

C. non-abstract methods

D. properties

E. All of the above

(d) Interfaces can include:

A. data members

B. abstract methods

C. non-abstract methods

D. All of the above

E. None of the above.

3. State if the following statements are *true* or *false*:

1. If a class contains an abstract method, the class must be defined as abstract. T
2. A super class inherits the data members and methods of a sub class. F (sub class inherit)
3. The statement base() in a constructor is a call to the base class constructor. T
4. A class that implements an interface may implement some of the methods defined in the interface. F

4. What is the main difference between overloaded and overridden methods?

* Overriding a method is different from overloading a method
* **Overridden** **methods** must have exactly the same method signatures
* **Overloaded** **methods** must have different method signatures

5. How do abstract classes differ from interfaces?

Abstract classes can include:

A. data members

B. abstract methods

C. non-abstract methods

D. properties

E. All of the above

But interfaces can only accept abstract methods

1. Consider the class definitions given below:
2. Can with keyword *abstract* for the method DoWork() in class A be removed? Explain why.

A class that contain abstract method must be defined abstract class

1. Why must class A be defined with keyword *abstract*?

It contains one abstract method

1. Why must keyword *override* be used for method DoWork() in class B?

Because class b is trying to provide an implementation of abstract method dowork() in class A

|  |  |
| --- | --- |
| public abstract class A  {  public abstract void DoWork();  } | using System;  public class B : A  {  public override void DoWork()  {  Console.WriteLine("Doing work");  }  } |

1. Consider the interface and class defintions given below. Is it valid to remove method DoWork() from class C?

|  |  |
| --- | --- |
| public interface IExample  {  void DoWork();  } | using System;  public class C : IExample  {  public override void DoWork()  {  Console.WriteLine("Doing work");  }  } |

Cannot. It is invalid to remove the method DoWork() from class c.

**Part B**

1. Create a Student class as an abstract class that contains the data members and properties to represent name and test marks and an abstract method named IsPass(). The method indicates whether the student has passed the test.

Create Undergraduate and Postgraduate classes as sub classes of Student class. Override the IsPass() method in the Undergraduate and Postgraduate classes. The pass marks for an undergraduate and postgraduate is 50 and 60 respectivly.

Test your classes.

sing System;

public abstract class Student

{

public string name;

public string Name

{

get { return name; }

}

private int testMarks;

public int TestMarks

{

get { return testMarks; }

set { testMarks = value; }

}

public Student(string name)

{

this.name = name;

}

public bool IsPass()

{

if (TestMarks >= 50)

return true;

else

return false;

}

}

public class Undergraduate : Student

{

private int level;

public int Level

{

get { return level; }

set { level = value; }

}

public Undergraduate(string name, int level) : base(name)

{

this.level = level;

}

public bool IsPass()

{

if (TestMarks >= 50)

return true;

else

return false;

}

public override string ToString()

{

return String.Format("Name:{0}, Level:{1}",

name, level);

}

}

public class Postgraduate : Student

{

public Postgraduate(string name) : base(name)

{

return;

}

public bool IsPass()

{

if (TestMarks >= 60)

return true;

else

return false;

}

public override string ToString()

{

return String.Format("Name:{0}",

name);

}

}

public class StudentTest3

{

static void Main()

{

Postgraduate postgraduate = new Postgraduate("Tim");

Console.Write("Enter test marks for postgraduate student {0}: ",

postgraduate.Name);

int marks = Convert.ToInt32(Console.ReadLine());

postgraduate.TestMarks = marks;

if (postgraduate.IsPass())

Console.WriteLine("Pass");

else

Console.WriteLine("Fail");

}

}

1. Refer to question 1. Can an interface be used instead of a class to represent a student?
2. Create an abstract class called GeometricFigure. Each figure includes a height and a width. Provide properties with *get* and *set* members for each data member. Include an abstract method called ComputeArea() that computes the area of the GeometricFigure.

Create 3 additional classes:

* A Rectangle is a GeometricFigure whose area is determined by multiplying width by height.
* A Square is a Rectangle in which the width and height are the same. Provide a constructor that accepts both height and width but forces them to be equal. Provide a second constructor that accepts just one dimension and uses it for both height and width. The Square class uses the Rectangle’s ComputeArea() method.
* A Triangle is a GeometricFigure whose area is determined by multiplying the width by half the height.

Test the classes.

class Test

{

public static void Main()

{

GeometricFigure[] gf = new GeometricFigure[5];

gf[0] = new Triangle(10, 20);

displayinfo(gf[0]);

}

public static void DisplayInfo(GeometricFigure gf)

{

}

}

4. Refer to the Rectangle, Square, and Triangle classes in Question 3. Create an application that demonstrates creating objects of each of the classes. After each object is created, pass it to a method that accepts a GeometricFigure argument. The method displays the figure’s type (using a ToString() method), and the figure’s height, width, and area.