C# Programming: From Problem Analysis to Program Design, 4th edition

ISBN 978-1-285-09626-1

Chapter 11

1. c. . inheritance

2. d. encapsulation

3. a. overriding

4. d. .dll

5. a. inheritance

6. e. object

7. d. all of the above

8. e. protected

9. d. public, private, public

10 a. virtual

11. d. assemblies

12. a. each project must consist of code in one language only

13. c. add a using directive

14. e. abstract class

15. d. one, many

16. e. all of the above

17. b. abstract methods

18. b. Abstraction

19. d. partial classes

20. e. all of the above

21. Explain the difference between an overloaded and an overridden method. Give an example of each.

Overridden methods must have exactly the same signature as the method it is overriding. New functionality is normally defined with overridden methods. Overloaded methods must have a different signature than others with the same name. Each of the different overloaded methods must have a different number and/or type of parameter from the others. Thus, by definition the constructors fall into the category of being overloaded methods. Overloaded methods are usually defined to perform a similar behavior, but with different data types. ToString( ) is example of an overridden method.

22. How do abstract classes differ from interface classes?

When you use the abstract modifier with a class, you are indicating it is intended to be used only as a base class. Normally that class is incomplete in that one or more of its methods are declared as abstract. No objects can be constructed from an abstract class. Think of an interface as a class that is totally abstract. Interfaces contain no implementation details for any of their members. All of their members are considered abstract.

23. a.

public decimal Pay

{

get

{

return pay;

}

}

b.

public class Employee( )

{

empNumber = 99999999;

pay = 0M;

}

c.

public class Employee(string empNo, decimal payAmt )

{

empNumber = empNo;

pay = payAmt;

}

d.

public class HourlyEmployee : Employee

{

protected double hours;

protected decimal payRate;

}

e.

public HourlyEmployee(string empNo, decimal payAmt, decimal rate, double hrs)

: base (empNo, payAmt)

{

hours = hrs;

payRate = rate;

}

f.

public virtual decimal ComputePayAmount( )

{

return pay;

}

g.

public override decimal ComputePayAmount( )

{

return hours \* payRate;

}

h.

public void ProcessData<T>(T data)