2015 Final Exam close book

I. Choice problem (one point for each):

1. Method overloading allows the programmer to:
a) create multiple methods with the same name but different arguments
b) create methods with different variables of the same type
c) create methods with the same parameters but different names
d) None of the above.

2. When the garbage collector executes, a chain of destructors execute in an order:
a) equivalent to the order of a chain of constructors
b) reverse of the order of a chain of constructors
c) specified by the finalizer method
d) None of the above.

3. Polymorphism specifically enables the creation of programs that handle:
a) classes that are containers for other classes
b) large amounts of data with efficiency
c) a wide variety of classes in a generic manner
d) None of the above

4. When C# encounters an overridden method, it determines which method to call from the:
a) number of different types on which the method is called
b) type of object on which the method is called
c) type of reference that refers to the object
d) None of the above.

5. Which of the following may result in a compiler error?
a) referring to a derived class object with a base class reference
b) referring to a base-class object with a derived-class reference
c) referring to a base-class object with a base-class reference
d) referring to a derived-class object with a derived-class reference

6. Abstract classes are classes which may not be:
a) inherited
b) accessed by derived-classes
c) instantiated
d) None of the above.

7. The purpose of an interface is to:
a) provide similar objects with the same functionality, even though each will implement the functionality differently
b) provide different objects with the same functionality, even though each will implement the functionality differently
c) provide default implementations of methods and properties
d) None of the above.

7. What are delegates?
a) classes that combine sealed classes
b) classes that encapsulate sets of references to methods
c) classes that provide methods to manipulate data
d) None of the above.

8. An exception is:
a) a problem a computer has during construction
b) a problem that a program has during runtime
c) something that the computer does not understand
d) the way a computer signals to the users that it is about to terminate

9. If an exception is thrown in a catch handler, any code in the handler that follows the thrown exception will:
a) generate a syntax error
b) generate a logic error
c) never be executed
d) run after the finally block is done

10. Re-throwing is used to:
a) catch even more errors in the program.
b) provide addition debugging information.
c) prevent the program from terminating from a fatal error.
d) All of the above

11. According to Microsoft, programmer-defined exceptions should contain 3 constructors:
a) a default constructor, a constructor that receives a string for the error message and a constructor that receives an exception argument of the inner exception object.
b) a default constructor, a constructor that receives a string and a constructor that receives both a string and an exception.
c) a constructor that receives a string, a constructor that receives an exception and a constructor that receives both.
d) a default constructor, a constructor that receives a string and a constructor that receives a number indicating the line number where the exception occurred.

12. Multicast event delegates must reference methods:
a) with the same name, but a different signature
b) that should all be raised by the same event
c) defined earlier in the program
d) with the different signature

13. Events in C# can be:
a) generated within the code of the program
b) started with a click on a button, or other control
c) generate by keyboard input
d) All of the above

14. In order to add or remove an event from a delegate, the programmer would use:
a) the Add and Remove methods
b) the Add and Subtract methods
c) the += and the - = operators
d) events can only be added with the += and not removed from a delegate

15. In order to find out which key the user is pressing one should use the:
a) KeyCode property to return the pressed key
b) Key property to return the pressed key
c) KeyPress property to return the pressed key
d) GetKey property to return the pressed key

16. In order to have a program close use:
a) the reserved word exit
b) the reserved word unload
c) Application.Exit()
d) Application.Unload()

17. A reasonal example of a TreeView is:
a) all of your e-mail
b) a Web site in Internet Explorer
c) the My Computer folder
d) the left side of Windows Explorer
18. Which is the following is true?
a) Inheritance helps with program organization by modeling the "has a" relationship.
b) Inheritance exactly replicates human family trees. c) Inheritance allows a class to consist of objects of other classes.
d) program can call a derived class function from a base class object.
e) derived class can call a non-private base class function if the derived class has no function with the same as its parent class;
f) If you want the functions in a derived class to access the data in a base class directly, the only one way is to declare the data in base class public.
19. If a class C is derived from the a class B, which is derived from a class A, then a class C member function can not access
a) protected and public data in C and B.
b) protected and public data in C.
c) private data in C.
d) protected data in A or B.
e) private data in A B.
f) private data in A
20. The purpose of an interface is to:
a) provide similar objects with the same functionality, even though each will implement the functionality differently.
b) provide different objects with the similar functionality, even though each will implement the functionality differently.
c) provide default implementations of methods and properties.
d) provide a class to be inherited by different classes.
e) provide a static member function that different classes can use it.
f) provide the functions having no inheritance property but with the same reaction as the inheritance.
21. Abstract classes are classes which can <u>not</u> be:
a) inherited
b) accessed by sub classes
c) instantiated
d) acting as a base class
e) consisted of concrete method
f) the one with no abstract method in it.
22. Method overloading allows the programmer to:
a) create multiple methods with the same name but different arguments;
b) create multiple methods with the same name but return the same data type;
c) create multiple methods with the same parameters but different names;
d) Cause the method to carry out several operations at once;
e) make the method carry out the same operation on different objects.
f) write several method definitions to solve the similar operation but with different parameters.
23. Which of the following is true for a constructor and destructor (i.e., finalize) of the same class?
a) for a derived and case classes, their execution order of the constructor and destructor in the above two classes are of the same
b) They are both called once per object (in general).
c) They both are able to accept default arguments.
d) Both are called automatically, even if not defined in the class.
e) destructor runs immediately when no variable has a link pointing to it;
f) constructor runs immediately when a variable has a link pointing to it.
24. Multicast event delegates must reference methods:
a) with the same name, but a different signature
b) that should all be raised by the same event
c) defined earlier in the program
d) with different names and different signatures
25. In order to add or remove an event from a delegate, you would use:
a) the Add and Remove methods
b) the Add and Subtract methods
c) the += and the -= operators
d) events can only be added with the += and not removed from a delegate
26. The difference between textboxes and labels is:
a) textboxes have a drop=down feature
b) labels cannot be changed during runtime
c) textboxes allow the user to enter information into them
d) nothing
27. What does the method InitializeComponent contain?
a) code provided by the user which sets the properties of controls that were added
b) code provided by Visual Studio which sets the properties of controls that were added
c) a and b
d) None of the above
28. In order to insert a control into a Panel or GroupBox in Design mode one must:
a) manually create the control inside the Panel or GroupBox
b) create a control and then drag it into the Panel or GroupBox
c) create a control and then set it as inside the Panel or GroupBox
d) group several controls and set their MainControl property to the Panel or GroupBox
29. When adding a node to a tree use:
a) treeView.Nodes[parentIndex].Nodes.Add(new TreeNode(ChildLabel))
b)treeView.Node[parentIndex].Add(new TreeNode(ChildLabel))
c) treeView.Node[parentIndex].Add(ChildLabel)
d) treeView.Node[parentIndex].Nodes.Add(ChildLabel)

30. The Exists method of class Directory is used to:
a) check if a given directory has sub-directories
b) check if a given directory actually exists
c) make sure that the List has a beginning and an ending
d) check for an actual list by making sure the first and last elements are not the same, meaning the list only has one item in it.

II. Short Answer questions (15 points for each):

a. Please draw the memory allocation of objects (as the slides), from time1 to time 6, and its value for the following codes.

```
7       public class Time2      {
9             private int hour;       // 0-23
10          private int minute;    // 0-59
11          private int second;    // 0-59
15          public Time2()      {
17              SetTime( 0, 0, 0 );
18          }
22          public Time2( int hour )      {
24              SetTime( hour, 0, 0 );
25          }
29          public Time2( int hour, int minute )    {
31              SetTime( hour, minute, 0 );
32          }
35    public Time2( int hour, int minute, int second ){
37      SetTime( hour, minute, second );
38    }
41    public Time2( Time2 time ){
43      SetTime( time.Hour, time.Minute, time.Second );
44    }
48          public void SetTime(int hourValue, int
minuteValue, int secondValue ) {
51              Hour = hourValue;
52              Minute = minuteValue;
53              Second = secondValue;       }
57          public int Hour      {
59              get {
61                  return hour; }
64              set    {
66                  hour =(( value >= 0 && value < 24 ) ?
value : 0 );    }
68          }
72    public int Minute{
74      get{
76          return minute;}
79      Set {
81          minute =((value >= 0 && value<60 ) ?
value:0 );}
84    }
87    public int Second {
89      get    {
91          return second;}
94      set {
96        second=((value >= 0&& value < 60 ) ? value :
0 );}
99    }
```

```
8   class TimeTest2
9   {
10     // main entry point for application
11     static void Main( string[] args )
12     {
13        Time2 time1, time2, time3, time4, time5, time6;
14
15        time1 = new Time2();                // 00:00:00
16        time2 = new Time2( 2 );             // 02:00:00
17        time3 = new Time2( 21, 34 );        // 21:34:00
18        time4 = new Time2( 12, 25, 42 );    // 12:25:42
19        time5 = new Time2( 27, 74, 99 );    // 00:00:00
20        time6 = new Time2( time4 );         // 12:25:42
21
```

.....}

b. Please draw the memory allocation of object, e, and its value (as the slides) for the following two examples:

Class for Date:

```
7       public class Date {
9           private int month;    // 1-12
10          private int day;      // 1-31 based on month
11          private int year;     // any year
            ……}
```

Example 1:

```
8       public class Employee {
10        private string firstName;
11        private string lastName;
12        private Date birthDate;
13        private Date hireDate;
16        public Employee( string first, string last,
17              int birthMonth, int birthDay, int birthYear,
18              int hireMonth, int hireDay, int hireYear ) {
20          firstName = first;
21          lastName = last;
24          birthDate=new
Date(birthMonth,birthDay,birthYear);
25          hireDate= new Date(hireMonth, hireDay,
hireYear );
26          }
8       class CompositionTest {
11          static void Main( string[] args ) {
13            Employee e =
14              new Employee("Bob", "Jones", 7, 24, 1949,
3, 12,1988);
16            MessageBox.Show( e.ToEmployeeString(),
17                  "Testing Class Employee" );
19          }
21      }
```

Example 2:

```
4           public class Employee {
5               private String firstName;
6               private String lastName;
7               private Date birthDate;
8               private Date hireDate;
11            public Employee( String first, String last, Date
dateOfBirth, Date dateOfHire ) {
14                firstName = first;
15                lastName = last;
16                birthDate = dateOfBirth;
17                hireDate = dateOfHire;
18            }
```

```
21        public String toEmployeeString(){
22              ……
26          }
28      }
5       public class CompositionTest2 {
7           static void Main( String args[] ) {
9               Date birth = new Date( 7, 24, 1949 );
10              Date hire = new Date( 3, 25, 2013 );
11             Employee e = new Employee("Bob",
"Jones", birth, hire);
13              ……
17          }
19      }
```

c. For the following codes , where resultTextBox is assumed to be a textbox.

```
14    public class Box     {
16        private string[] names = { "length", "width",
"height" };
17        private double[] dimensions = new double[ 3 ];
20        public Box ( double length, double width,
double height )     {
22            dimensions[ 0 ] = length;
23            dimensions[ 1 ] = width;
24            dimensions[ 2 ] = height;
25        }
28        public double this[ int index]       {
30            get   {
32                return ( index < 0 || index >
dimensions.Length ) ?
33                    -1 : dimensions[ index ];
    34            }
36            set   {
38                if ( index >= 0 && index <
dimensions.Length )
39                    dimensions[ index ] = value;
40            }
42        } // end numeric indexer
45        public double this[ string name ]       {
47            get          {
50                int i = 0;
52                while ( i < names.Length &&
name.ToLower() != names[ i ] )
54                    i++;
56                return ( i == names.Length ) ? -1 :
dimensions[ i ];
57            }
59            set         {
62                int i = 0;
64                while ( i < names.Length &&
name.ToLower() != names[ i ] )
66                    i++;
68                if ( i != names.Length )
69                    dimensions[ i ] = value;
70            }
72        } // end indexer
74    } // end class Box
118       private void ShowValueAtIndex( string prefix,
int index )   {
120           resultTextBox.Text =
121               prefix + "box[ " + index + " ] = " +
box[ index ];
122       }
/// and this following codes
125       private void ShowValueAtIndex( string prefix,
string name )     {
127           resultTextBox.Text =
128               prefix + "box[ " + name + " ] = " +
box[ name ];
129       }
```

What is the meaning for the codes from lines 118-129? Please give the logic meaning in usage “indexer”.

d. Suppose class A inherits from base class B. Please list the calling order of the following four methods when an object of class A is instantiated then destroyed? These methods are: A’s constructor, B’s constructor, A’s destructor, B’s destructor.

e. Please write the execution results for the following codes.

```
5       class UsingExceptions    {
8           static void Main( string[] args ){
11           Console.WriteLine( "Calling
DoesNotThrowException" );
12           DoesNotThrowException();
15           Console.WriteLine("\nCallingThrow
ExceptionWithCatch");
16           ThrowExceptionWithCatch();
19           Console.WriteLine"\nCalling
ThrowExceptionWithoutCatch");
21              // call ThrowExceptionWithoutCatch
22           try {
24                ThrowExceptionWithoutCatch();
25             }
28           catch    {
30                Console.WriteLine( "Caught exception from "
+
31                    "ThrowExceptionWithoutCatch in
Main" );
32           }
35        Console.WriteLine("\nCalling
ThrowExceptionCatchRethrow");
38        try {
40            ThrowExceptionCatchRethrow();
41         }
44        catch    {
46            Console.WriteLine( "Caught exception from " +
47                    "ThrowExceptionCatchRethrow in
Main" );
48            }
49         } // end method Main
51         public static void DoesNotThrowException() {
54            try {
56                Console.WriteLine( "In
DoesNotThrowException" );
57            }
59            catch {
61                Console.WriteLine( "This catch never
executes" );
62            }
64            finally {
66                Console.WriteLine(
```

```
67                      "Finally executed in
DoesNotThrowException" );
68              }
69              Console.WriteLine( "End of
DoesNotThrowException" );
70          }
72          public static void ThrowExceptionWithCatch() {
75              try {
77                  Console.WriteLine( "In
ThrowExceptionWithCatch" );
78                  throw new Exception("Exception in

ThrowExceptionWithCatch" );
80              }
82              catch ( Exception error ) {
84                  Console.WriteLine( "Message: " +
error.Message );
85              }
87              finally {
89                  Console.WriteLine(
90                      "Finally executed in
ThrowExceptionWithCatch" );
91              }
92              Console.WriteLine( "End of
ThrowExceptionWithCatch" );
93          } // end method ThrowExceptionWithCatch
95          public static void ThrowExceptionWithoutCatch() {
98              try {
100                 Console.WriteLine("In
ThrowExceptionWithoutCatch" );
101                 throw new Exception(
102                     "Exception in
ThrowExceptionWithoutCatch" );
103             }
105             finally {
107                 Console.WriteLine( "Finally executed in " +
108                     "ThrowExceptionWithoutCatch" );
109             }
111             Console.WriteLine( "This will never be printed" );
112         }
114         public static void ThrowExceptionCatchRethrow() {
117             try {
119                 Console.WriteLine("In
ThrowExceptionCatchRethrow" );
120                 throw new Exception(
121                     "Exception in
ThrowExceptionCatchRethrow" );
122             }
124             catch ( Exception error ) {
126                 Console.WriteLine( "Message: " +
error.Message );
128                 throw error;
130             }
132             finally {
134                 Console.WriteLine( "Finally executed in " +
135                     "ThrowExceptionCatchRethrow" );
136             }
138             Console.WriteLine( "This will never be printed" );
139         } // end method ThrowExceptionCatchRethrow
140 } // end class UsingExceptions
```

f. For the following codes , please draw the memory allocation graph as the slides and write the execution result.

```
7       public class Point      {
10          private int x, y;
13          public Point()          {
16          }
19          public Point( int xValue, int yValue )    {
22              X = xValue;
23              Y = yValue;
24          }
7       public class Circle : Point {
9           private double radius;
12          public Circle() {
15          }
18          public Circle(int xValue, int yValue, double
radiusValue ): base( xValue, yValue) {
21              Radius = radiusValue;
22          }
8       class PointCircleTest    {
11          static void Main( string[] args )    {
13              Point point1 = new Point( 30, 50 );
14              Circle circle1 = new Circle( 120, 89, 2.7 );
16              string output = "Point point1: " +
point1.ToString() +    "\nCircle circle1: " +
circle1.ToString();
21              Point point2 = circle1;
23              output += "\n\nCCircle circle1 (via point2):
" +   point2.ToString();
28              Circle circle2 = ( Circle ) point2;
30              output += "\n\nCircle circle1 (via circle2): "
+   circle2.ToString();
33              output += "\nArea of circle1 (via circle2): "
+   circle2.Area().ToString( "F" );
37              if ( point2 is Circle )      {
39                  circle2 = ( Circle ) point2;
40                  output += "\n\ncast successful";
41              }
42              else       {
44                  output += "\n\npoint2 does not refer to a
Circle";
45              }
47              MessageBox.Show( output,
48                  "Demonstrating the 'is a' relationship" );
50          }
52      }
```