Question # 1

- Write a class ‘Shape’ that contains the following
  - Constructor takes one argument, string name
    - Constructor prints “Shape called <name> created”
  - Abstract method ‘calculateArea’
  - Method ‘ToString’ which prints “<count> This is a shape object called <name>”
  - Protected field ‘name’ and relevant properties for the same
  - Static field ‘count’ which gets incremented every time a new instance of shape is created or when an instance of its derived class is created.
  - Add a protected filed for ‘color’ and set it to Black
- Write a class ‘circle’ which inherits from above mentioned class ‘shape’ and have additional members as below
  - Constructor that takes one parameter ‘name’ and prints relevant message, sets ‘radius’ to default value
  - Constructor that takes two parameters ‘name’ and ‘radius’ and also prints relevant message
  - Protected field ‘radius’ and relevant properties
  - Create a protected field for ‘color’ by hiding base class color (using ‘new’ keyword) and set its value to Blue, create relevant properties for this field
  - Implementation for method ‘calculateArea’
  - Override ‘ToString’ method to print message: ‘This is a Circle object called <name> whose radius is <radius> and area is <area> and of color <color>”
- Write a class ‘rectangle’ which inherits from above mentioned class ‘shape’ and have additional members as below
  - Constructor that takes one parameter ‘name’ and prints relevant message, sets ‘width’ and ‘length’ to default value
  - Constructor that takes three parameters ‘name’, ‘length’ and ‘width’ and also prints relevant message
  - Protected fields ‘length’, ‘width’ and relevant properties
  - Create a protected field for ‘color’ by hiding base class color (using ‘new’ keyword) and set its value to Red , create relevant properties for this field
  - Implementation for method ‘calculateArea’
  - Override ‘ToString’ method to print: "This is a Rectangle object called <name> whose length is <length> and width is <width> and area is <area> of color <color>”

Question # 1 Contd.

- Write a class ‘square’ which inherits from above mentioned class ‘rectangle’ and have additional members as below
  - Constructor that takes one parameter ‘name’ and prints relevant message, sets ‘width’ to default value, ‘length’ to same value as ‘width’
  - Constructor that takes two parameters ‘name’ and ‘width’, ‘length’ to same value as ‘width’ and also prints relevant message
  - Protected fields ‘length’, ‘width’ and relevant properties **if needed**
  - Create a protected field for ‘color’ by hiding base class color (using ‘new’ keyword) and set its value to White , create relevant properties for this field
  - Implementation for method ‘calculateArea’ **if needed**
  - Override ‘ToString’ method to print: "This is a Square object called <name> whose length is <length> and width is <width> and area is <area> of color <color>”
- Use the following for the Main() method for your program

```
Circle circleA = new Circle();
Circle circleB = new Circle("Circle-B");
Circle circleC = new Circle("Circle-C", 5);
Rectangle rectangleA = new Rectangle();
Rectangle rectangleB = new Rectangle("Rectangle-B");
Rectangle rectangleC = new Rectangle("Rectangle-C", 5, 8);
Square squareA = new Square();
Square squareB = new Square("Square-B");
Square squareC = new Square("Square-C", 5);

Console.WriteLine();
Console.WriteLine(circleA);
Console.WriteLine(circleB);
Console.WriteLine(circleC);
Console.WriteLine(rectangleA);
Console.WriteLine(rectangleB);
Console.WriteLine(rectangleC);
Console.WriteLine(squareA);
Console.WriteLine(squareB);
Console.WriteLine(squareC);

Console.WriteLine();
Console.WriteLine(circleA.Name + "\t" + circleA.Radius + " has area: " + circleA.calculateArea() + "\t" + circleA.Color);
Console.WriteLine(circleB.Name + "\t" + circleB.Radius + " has area: " + circleB.calculateArea() + "\t" + circleB.Color);
Console.WriteLine(circleC.Name + "\t" + circleC.Radius + " has area: " + circleC.calculateArea() + "\t" + circleC.Color);
Console.WriteLine(rectangleA.Name + "\t" + rectangleA.Width + "\t" + rectangleA.Length + " has area: " + rectangleA.calculateArea() + "\t" + rectangleB.Color);
Console.WriteLine(rectangleB.Name + "\t" + rectangleB.Width + "\t" + rectangleB.Length + " has area: " + rectangleB.calculateArea() + "\t" + rectangleB.Color);
Console.WriteLine(rectangleC.Name + "\t" + rectangleC.Width + "\t" + rectangleC.Length + " has area: " + rectangleC.calculateArea() + "\t" + rectangleC.Color);
Console.WriteLine(squareA.Name + "\t" + squareA.Width + " has area: " + squareA.calculateArea() + "\t" + squareA.Color);
Console.WriteLine(squareB.Name + "\t" + squareB.Width + " has area: " + squareB.calculateArea() + "\t" + squareB.Color);
Console.WriteLine(squareC.Name + "\t" + squareC.Width + " has area: " + squareC.calculateArea() + "\t" + squareC.Color);
Console.Read();
```

Question # 2

- Modify the Library example for the following changes
- Create a new Interface
  - interface IDigital
    {
    uint LengthInSeconds { get; set; }
    DigitalDisc.DiscType MediaType { get; set; }

    // following method will return Hours, Minutes, Sec
    // for the duration of digital media (lengthSeconds)
    // e.g., 1:30:25
    string getHMS();
    }
- Implement IDigital interface with Music, Movie and AudioBook class
  - Make other necessary changes in application to retain the functionality
  - Add lines in Main() method to check-out and check-in a Music item, Movie item and AudioBook item – one each
  - From Main() method, print duration for each of Music item, Movie item and AudioBook item in HH:MM:SS format (no zero padding required)

Question # 3

- Modify the Library example for the following changes
- Create a new List<Item> as a private member of Item class
  - Implement necessary methods to get/set an element publicly for this List
- Implement IComparable interface which compares based on ‘year’ field of Item class
  - Hint: See ‘Simple Generics’ example which compares Persons based on their age
- Implement ItemCollection class which implements IEnumerable interface
  - Use Dictionary<uint, Item> to list all Items accessible by ‘year’
    - Dictionary requires unique key. So have only one item per year for this example
  - Implement Indexer to access items by value of ‘year’
    - Hint: See ‘String Indexer’ example
- At the end of Main() method
  - Print all (unsorted) elements of List<Item> using Iterator
  - Sort elements of List<Item> based on ‘year’
  - Print all (sorted) elements of List<Item> using Iterator
  - Print all elements in dictionary collection using Indexer

Question # 4

- Modify the Library example for the following changes
- Add a method to read Items from a text file using Regex to parse the entries where each entry is in the following format per line
  - Type: Book Year: 1991 Title: This is book 1 ISBN:1234567890 Pages: 500 Authors: Smith, John; Brown, Jack
  - …And likewise for other media types also
  - Using above data, populate List and Dictionary
- Add a method to write the LibraryCatalog.txt file which dumps all entries in plain text format (as per above mentioned format)
- (Optional) Add a method to write LINQ expression to do a query all Items based on ‘year’ and return the match
  - Modify Main() method to execute search and print the result

Question # 5 (Optional)

- Modify the Library example for the following changes
- Add a method to write the items in the library (iterate through List) into an XML file using XmlWriter class
- Sample Output:
  - <Library>
    <Item Type="Book">
    <Year>2009</Year>
    <Title>This is book 1</Title>
    <ISBN>1234567890</ISBN>
    …
    <Authors>
    <Author>
    <FirstName>John</FirstName>
    <LastName>Smith</LastName>
    </Author>
    …
    </Authors>
    </Item>
    <Item Type="AudioBook">
    …
    </Item>
    </Library>