Write an application that includes:

- A text box with a label Model Name
- A text box with a label Price
- A text box with a label Speed
- A text box with a label Results
- A control displaying error information
- A button Add that adds data for a car to a current sorted list
- A button Change Velocity that changes the speed for a car with a given model name
- A button List that displays the sorted list of cars
- A button Clear that clears the text fields

Define an interface **IVehicle** for object that presents a vehicle.

1. Declared a read/write property **Speed** that sets/returns the speed of a vehicle.

Define a class **Car** that implements the **IVehicle** and **IComparable** interfaces. The class contains private information for the car model, price, and speed.

1. Add a constructor that allows all variables to be given initial values.
2. Implement the **Speed** property of the **IVehicle** interface.
3. Implement the method **int CompareTo (Object obj)** of the IComparable interface that allows sorting by price; if prices are equals – sorting by speed; if speeds are equal – sorting by model.
4. Override the method **public virtual string ToString()** that returns the string representation of this object.

In the Form class:

1. Declare a private field for the **SortedList** of cars and initialize it.
2. Declare a method that handles the **Click** event for the **Add** button

- Adds a car to the sorted list – uses the model name as a key and the car itself as a value. The result text box displays the entered data for the new car.
- If there is a car in the sorted list with the entered data the result text box displays **ERROR!**.
- If the needed data is not entered the error provider control sets error for the corresponding text box.

3. Declare a method that handles the **Click** event for the **Change Velocity** button

- If the data for the model name is entered use the **Speed** property to change the car speed. The result text box displays the updated information for the car (successful search) or **There are not data!** (unsuccessful search).
- If the data for the car model is not entered the error provider control sets the error for this text box.

4. Declare a method that handles the **Click** event for the **List** button

- Displays the current contents of the sorted list of cars in the result text box using **IDictionaryEnumerator**.

5. Declare a method that handles the **Click** event for the **Clear** button

- Clears all text boxes and the error provider control.

1. Which types in the Common Type System clean up the memory using the garbage collector?

- ❑ All types
- [✓ Referent types only]
- ❑ All types except primitive data types
- ❑ Value types only

2. When the explicit casting is needed?

- [✓ Conversion from a base class to a derive class]
- ❑ Conversion from a derive class to a base class
- ❑ In both cases

3. Which of the following statements will you use if you know that the array element must be modified?

- [✓ for statement]
- ❑ foreach statement
- ❑ IEnumerator object returned from the GetEnumerator() method

4. Define a class, MyClass, with a private filed that presents an array of 100 integers and an indexer that throws InvalidOperationException when the array index is out of range.

```
class MyClass
{
   private int[] array = new int[100];
   public int this [int index]
   {
      get
      {  if (index<0 || index>=100)
            throw new InvalidOperationException("The index is out of range");
         else return array[index];
      }
      set
      {  if (index<0 || index>=100)
            throw new InvalidOperationException("The index is out of range");
         else array[index]=value;
      }
   }
}
```

5. Define an interface, IStorable, with two methods and a property. The first method, Read, does not have parameters and returns an integer; the second method, Write, has an Object parameter and does not return a value. The property, Status, is from integer data type.
Define a class, Document, that implements the IStorable.

```
interface IStorable
{  int Read ();
   void Write (Object o);
   int Status {   get;   set;   }
}
public class Document : IStorable
{  private int status;
   public int Status
   {  get {   return status;   }
      set {   status = value;   }
   }
   public int Read ()   {   …   }
   public void Write (Object o)   {   …   }
}
```

6. What will be the output of the following code?

```
try
{  int num = 100;
   int den = 0;
   try
   {
         MessageBox.Show("Message1");
          int res = num/den;
          MessageBox.Show("Message2");
   }
   catch(ArithmeticException ae)
   {
          MessageBox.Show("Message3");
   }
   MessageBox.Show("Message4");
}
```

- ❑ Message1
  Message4
  Message1
- ❑ Message2
  Message3
  Message4
- [✓ Message1
  Message3
  Message4]
- ❑ Message1
  Message3
  Message2
  Message4

7. Exceptions are:

- ☑ Means to stop error spreading in the program code
- ☑ Way to break away checking of error status codes returned by functions
- ☐ Way for error recovery
- ☑ Object-oriented mechanism for error handling

8. Define:

- A class that contains information for the Pick event
- A public delegate that handles the Pick event
- A public Pick event
- An event handler for the Pick event.

```
class PickEventArgs : EventArgs
{   …   }
public delegate void PickEventArgsHandler (object source,
                                            PickEventArgs e);
public event PickEventArgsHandler Pick;
void OnPick (object source, PickEventArgs e)
{   …   }
```

9. Which of the following form events will you use to include the initialization code for controls?

- ☐ Activated
- ☑ Load
- ☐ Closed

10. Which of the following events will you use to validate user input?

- ☐ LostFocus
- ☑ Validating
- ☐ Leave
- ☐ Validated