Why don’t these methods have a return type?

```
public Grade(int studentID, string assessmentDate, string subject, string
assessment, string comments)
{
    StudentID = studentID;
    AssessmentDate = assessmentDate;
    SubjectName = subject;
    Assessment = assessment;
    Comments = comments;
}

// Default constructor
public Grade()
{
    StudentID = 0;
    AssessmentDate = DateTime.Now.ToString("d");
    SubjectName = "Math";
    Assessment = "A";
    Comments = String.Empty;
}
```

What’s this?

```
public string Password {get; set;}
```

How would you explain this?

```
Subjects = new List<string>() { "Math", "English", "History", "Geography",  "Science" };
```

How would you explain this? What is it?

```
public string AssessmentDate
{
    get
    {
        return _assessmentDate;
    }

    set
    {
        DateTime assessmentDate;

        // Verify that the user has provided a valid date
        if (DateTime.TryParse(value, out assessmentDate))
        {
            // Check that the date is no later than the current date
            if (assessmentDate > DateTime.Now)
            {
                // Throw an ArgumentOutOfRangeException if the date is after the current date
                throw new ArgumentOutOfRangeException("AssessmentDate", "Assessment date must be on or before the current date");
            }

            // If the date is valid, then save it in the appropriate format
            _assessmentDate = assessmentDate.ToString("d");
        }
        else
        {
            // If the date is not in a valid format then throw an ArgumentException
            throw new ArgumentException("AssessmentDate", "Assessment date is not recognized");
        }
    }
}
```

How would you explain this?

```
[TestInitialize]
public void Init()
{
// Create the data source (needed to populate the Subjects collection)
GradesPrototype.Data.DataSource.CreateData();
}

[TestMethod]
public void TestValidGrade()
{
    GradesPrototype.Data.Grade grade = new GradesPrototype.Data.Grade(1, "01/01/2012", "Math", "A-", "Very good");
    Assert.AreEqual(grade.AssessmentDate, "01/01/2012");
    Assert.AreEqual(grade.SubjectName, "Math");
    Assert.AreEqual(grade.Assessment, "A-");
}
```

What is this method? Why is it here?

```
public int CompareTo(Student other)
{
    // Concatenate the LastName and FirstName of this student
    string thisStudentsFullName = LastName + FirstName;

    // Concatenate the LastName and FirstName of the "other" student
    string otherStudentsFullName = other.LastName + other.FirstName;

    // Use String.Compare to compare the concatenated names and return the result
    return(String.Compare(thisStudentsFullName, otherStudentsFullName));
}
```

Annotated slides follow

What's this?

```
public string Password {get; set;}
```

Property

this saves me time so I can do better things than work.

equivalent

```
private string _password;
public string Password{
   get { return _password;}
   set {_password = value;}
}
```

How would you explain this?

Creates the object

```
Subjects = new List<string>() { "Math", "English", "History", "Geography",  "Science" };
```

A List is a collection
this definition is a generic
Everything in this list is a string

these are the initial strings
in the list.

How would you explain this?

Creates the object

List<string> subjects;

```
Subjects = new List<string>() { "Math", "English", "History", "Geography",  "Science" };
```

A List is a collection
this definition is a generic
Everything in this list is a string

these are the initial strings
in the list.

How would you explain this? What is it?

```
public string AssessmentDate
{
    get
    {
        return _assessmentDate;
    }

    set
    {
        DateTime assessmentDate;

        // Verify that the user has provided a valid date
        if (DateTime.TryParse(value, out assessmentDate))
        {
            // Check that the date is no later than the current date
            if (assessmentDate > DateTime.Now)
            {
                // Throw an ArgumentOutOfRangeException if the date is after the current date
                throw new ArgumentOutOfRangeException("AssessmentDate", "Assessment date must be on or before the current date");
            }

            // If the date is valid, then save it in the appropriate format
            _assessmentDate = assessmentDate.ToString("d");
        }
        else
        {
            // If the date is not in a valid format then throw an ArgumentException
            throw new ArgumentException("AssessmentDate", "Assessment date is not recognized");
        }
    }
}
```

Property

```
Grade g = new Grade();
g.AssessmentDate=DateTime.Now;
Console.WriteLine(g.AssessmentDate);
```

How would you explain this? What is it?

```
public string AssessmentDate
{
    get
    {
        return _assessmentDate;
    }

    set
    {
        DateTime assessmentDate;

        // Verify that the user has provided a valid date
        if (DateTime.TryParse(value, out assessmentDate))
        {
            // Check that the date is no later than the current date
            if (assessmentDate > DateTime.Now)
            {
                // Throw an ArgumentOutOfRangeException if the date is after the current date
                throw new ArgumentOutOfRangeException("AssessmentDate", "Assessment date must be on or before the current date");
            }

            // If the date is valid, then save it in the appropriate format
            _assessmentDate = assessmentDate.ToString("d");
        }
        else
        {
            // If the date is not in a valid format then throw an ArgumentException
            throw new ArgumentException("AssessmentDate", "Assessment date is not recognized");
        }
    }
}
```

Property

"1/1/1990";

Grade g = new Grade();

g.AssessmentDate=~~DateTime.Now~~;

Console.WriteLine(g.AssessmentDate);

the unit testing framework looks for this and runs it befor running any tests.

How would you explain this?

```
[TestInitialize]
public void Init()
{
// Create the data source (needed to populate the Subjects collection)
GradesPrototype.Data.DataSource.CreateData();
}

[TestMethod]
public void TestValidGrade()
{
    GradesPrototype.Data.Grade grade = new GradesPrototype.Data.Grade(1, "01/01/2012", "Math", "A-", "Very good");
    Assert.AreEqual(grade.AssessmentDate, "01/01/2012");
    Assert.AreEqual(grade.SubjectName, "Math");
    Assert.AreEqual(grade.Assessment, "A-");
}
```

C# "attribute". It does not execute. It is 'Metadata'.

C# attribute. The unit testing framework looks for these and runs them as tests.

run the test

Expected test results.

What is this method? Why is it here?

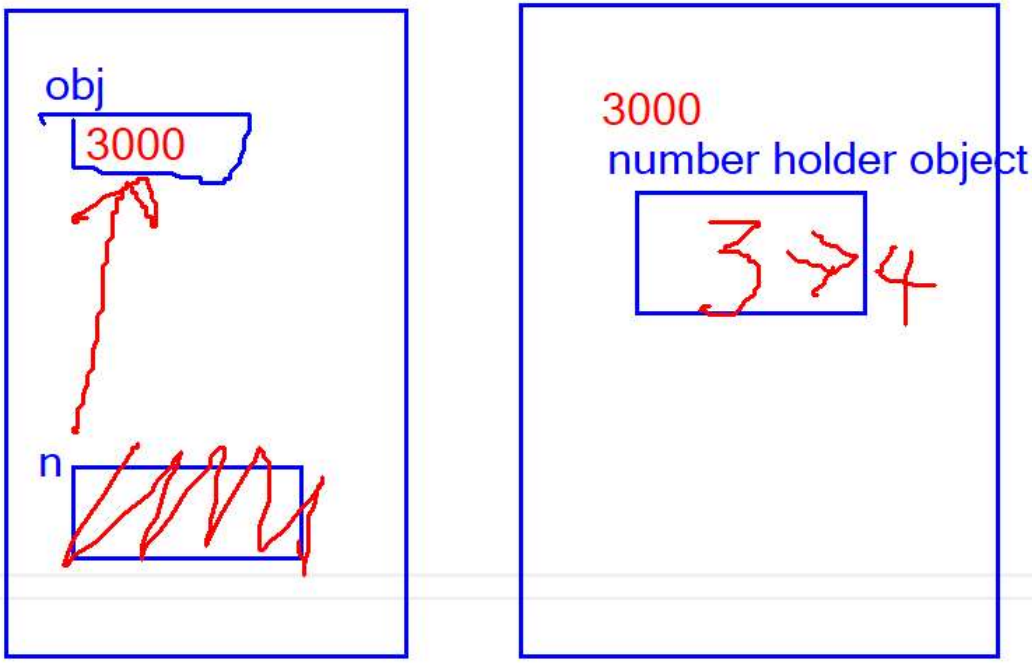
```
public int CompareTo(Student other)
{
    // Concatenate the LastName and FirstName of this student
    string thisStudentsFullName = LastName + FirstName;

    // Concatenate the LastName and FirstName of the "other" student
    string otherStudentsFullName = other.LastName + other.FirstName;

    // Use String.Compare to compare the concatenated names and return the result
    return(String.Compare(thisStudentsFullName, otherStudentsFullName));
}
```

Can be used as part of sorting.
This method is in the Student Class.
The Student class implements the ICompareable Interface.
If there is a list of students (List<Student> students;) you could execute
students.Sort();

```
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using System.Threading.Tasks;
6
7   namespace ConsoleApp3
8   {
9       class Program
10      {
11          static void Main(string[] args)
12          {
13              NumberHolder obj = new NumberHolder() { TheNumber = 3 };
14              F(ref obj);
15
16              Console.WriteLine(obj.TheNumber);
17
18              Console.ReadLine();
19
20          }
21          static void F(ref NumberHolder n)
22          {
23              n.TheNumber++;
24          }
25      }
26
27
28      class NumberHolder
29      {
30          public int TheNumber { get; set; }
31      }
32  }
33
```



```
namespace ConsoleApp3
{
    class Program
    {
        static void Main(string[] args)
        {
            NumberHolder obj = new NumberHolder() { TheNumber = 3 };
            F(obj);

            Console.WriteLine(obj.TheNumber);

            Console.ReadLine();

        }
        static void F(NumberHolder n)
        {
            n = new NumberHolder() { TheNumber = 0 };
            n.TheNumber++;
        }
    }


    class NumberHolder
    {
        public int TheNumber { get; set; }
    }
}
```