Telerik Academy

Defining Classes – Part 1

Classes, Fields, Constructors, Methods, Properties

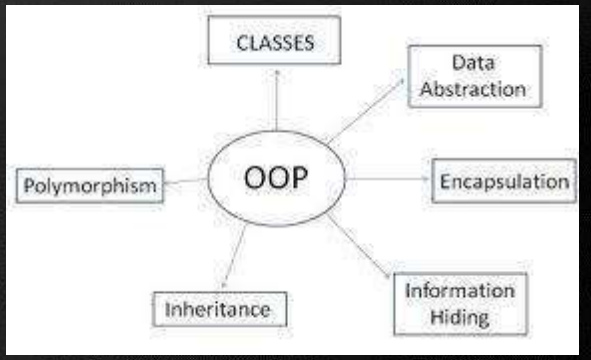
C# OOP

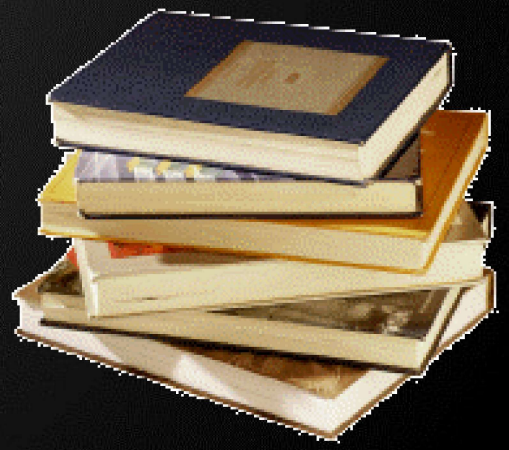
Telerik Software Academy

https://telerikacademy.com

Telerik Academy

Table of Contents





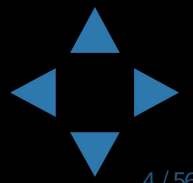
Telerik Academy

Defining Simple Classes

Telerik Academy

Classes in OOP

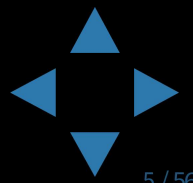
- Classes model real-world objects and define
  - Attributes (state, properties, fields)
  - Behavior (methods, operations)
- Classes describe the structure of objects
  - Objects describe particular instance of a class
- Properties hold information about the modeled object relevant to the problem
- Operations implement object behavior



Telerik Academy

Classes in C#

- Classes in C# can have members:
  - Fields, constants, methods, properties, indexers, events, operators, constructors, destructors, …
  - Inner types (inner classes, structures, interfaces, delegates, ...)
- Members can have access modifiers (scope)
  - public, private, protected, internal
- Members can be
  - static (common) or instance (specific for a given object)



Telerik Academy

Simple Class Definition

Begin of class definition

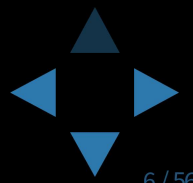
Inherited (base) class

```
public class Cat : Animal {
    private string name;
    private string owner;

    public Cat(string name, string owner)
    {
        this.name = name;
        this.owner = owner;
    }

    public string Name
    {
        get { return this.name; }
        set { this.name = value; }
    }
```

Fields

Constructor

Property



Telerik Academy

Simple Class Definition

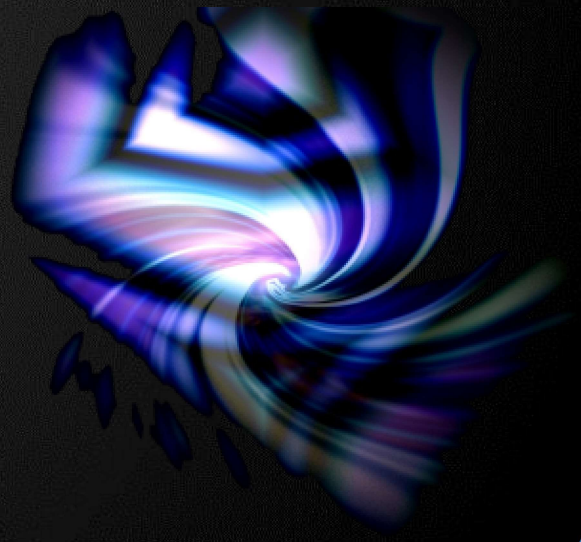
```
    public string Owner
    {
        get { return this.owner; }
        set { this.owner = value; }
    }

    public void SayMiau()
    {
        Console.WriteLine("Miauuuuuuu!");
    }
}
```

Method

End of class definition

Telerik Academy Class Definition and Members

- Class definition consists of:
  - Class declaration
  - Inherited class or implemented interfaces
  - Fields (static or not)
  - Constructors (static or not)
  - Properties (static or not)
  - Methods (static or not)
  - Events, inner types, etc.



Telerik Academy

Fields

Defining and Using Data Fields

Telerik Academy

Fields

- Fields are data members defined inside a class
  - Fields hold the internal object state
  - Can be static or per instance
  - Can be private / public / protected / …

```
class Dog
{
    private string name;
    private string breed;
    private int age;
    protected Color color;
}
```

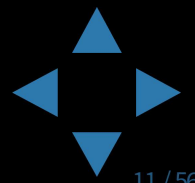
Field declarations

Telerik Academy

Constant Fields

- Constant fields are of two types:
  - Compile-time constants – const
    - Replaced by their value during the compilation
    - Can contain only values, known at compile time
  - Runtime constants – readonly
    - Assigned once only at object creation
    - Can contain values, calculated run time

```
class Math
{
   public const float PI = 3.14159;
   public readonly Color =
      Color.FromRGBA(25, 33, 74, 128);
}
```

Telerik Academy

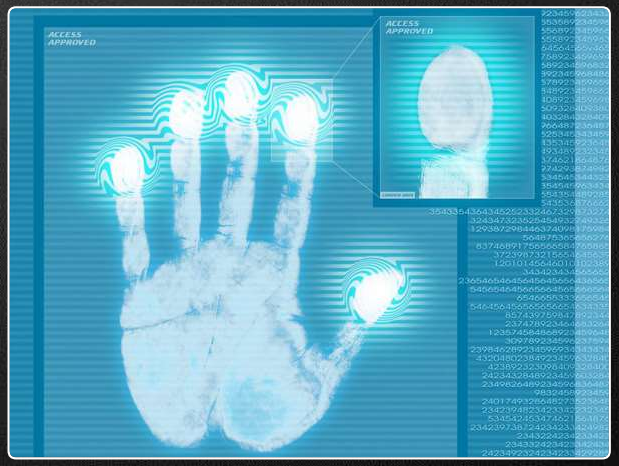
Constant Fields – *Example*

```
public class Constants
{
  public const double PI = 3.14159265358979323385;
  public readonly double Size;
  public Constants(int size)
  {
    this.Size = size; // Cannot be further modified!
  }
  static void Main()
  {
    Console.WriteLine(Constants.PI);
    Constants c = new Constants(5);
    Console.WriteLine(c.Size);
    c.Size = 10; // Compilation error: readonly field
    Console.WriteLine(Constants.Size); // compile error
  }
}
```

Telerik Academy

Access Modifiers

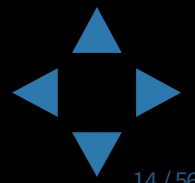
Public, Private, Protected, Internal



Telerik Academy

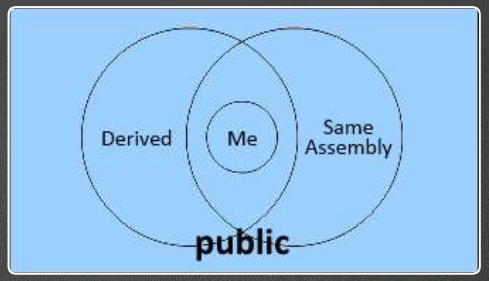
Access Modifiers

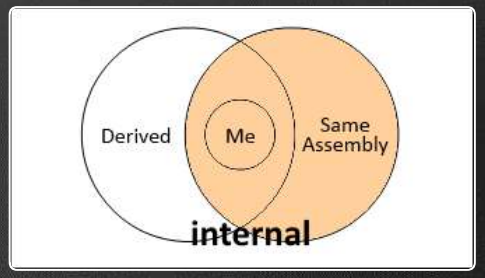
- Class members can have access modifiers
  - Restrict the access to them from outer sources
  - Supports the OOP principle "encapsulation"
- Class members can be:
  - public – accessible from any class
  - protected – accessible from the class itself and all its descendent classes
  - private – accessible from the class itself only
  - internal (default) – accessible from the current assembly, i.e. the current VS project

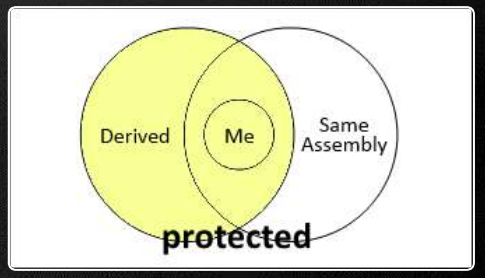


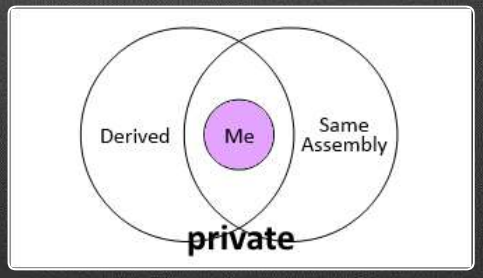
Telerik Academy

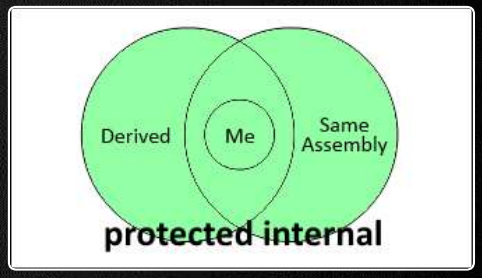
Access Modifiers Explained











Telerik Academy

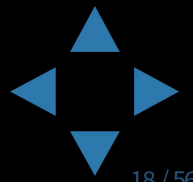
Defining Simple Classes

Example

Telerik Academy

Task: Define a Class "Dog"

- Our task is to define a simple class that represents information about a dog
  - The dog should have name and breed
    - Optional fields (could be null)
  - The class allows to view and modify the name and the breed at any time
  - The dog should be able to bark



Telerik Academy

Defining Class Dog – *Example*

```
public class Dog{
    private string name;
    private string breed;

    public Dog()
    {
    }

    public Dog(string name, string breed)
    {
        this.name = name;
        this.breed = breed;
    }
```

(the example continues)

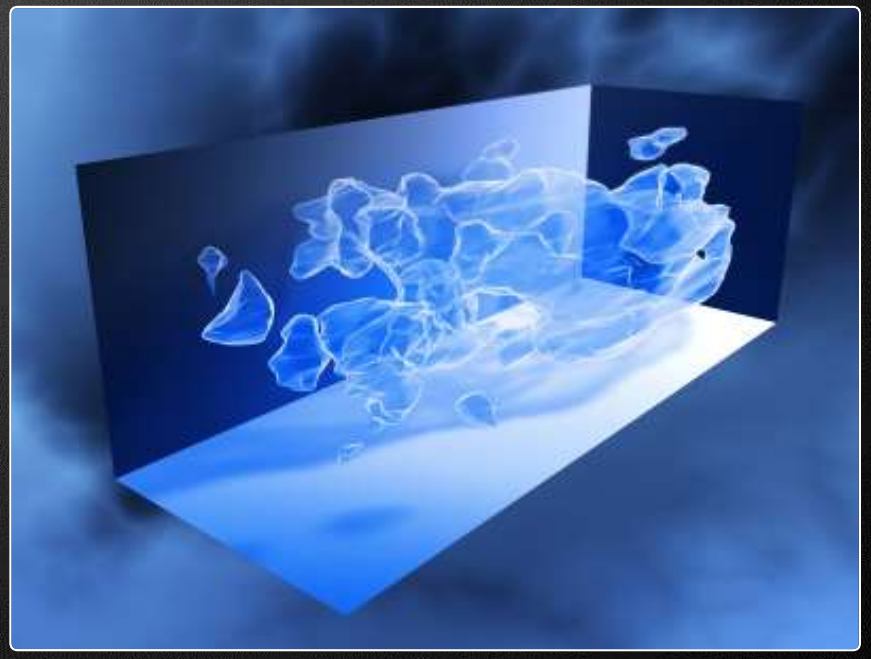
Telerik Academy

Defining Class Dog – *Example*

```
    public string Name
    {
        get { return this.name; }
        set { this.name = value; }
    }

    public string Breed
    {
        get { return this.breed; }
        set { this.breed = value; }
    }

    public void SayBau()
    {
        Console.WriteLine("{0} said: Bauuuuuu!",
            this.name ?? "[unnamed dog]");
    }
}
```

Telerik Academy

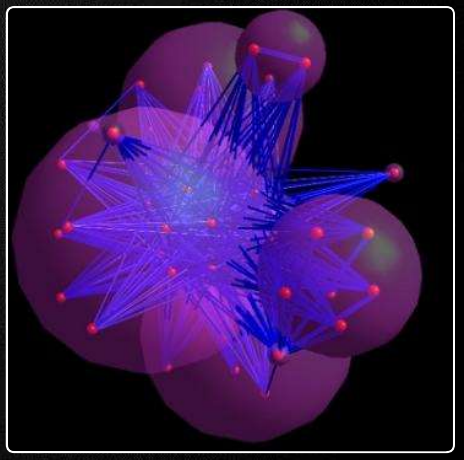
Using Classes and Objects



Telerik Academy

How to Use Classes (Non-Static)?

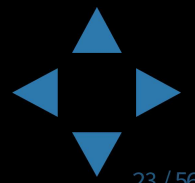
- Create an instance
  - Initialize its properties / fields
- Manipulate the instance
  - Read / modify its properties
  - Invoke methods
  - Handle events
- Release the occupied resources
  - Performed automatically in most cases



Telerik Academy

Task: Dog Meeting

- Our task is as follows:
  - Create 3 dogs
    - The first should be named “Sharo”, the second – “Rex” and the last – left without name
  - Put all dogs in an array
  - Iterate through the array elements and ask each dog to bark
  - *Note*:
    - Use the Dog class from the previous example!

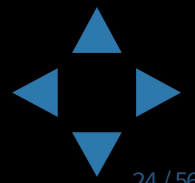


Telerik Academy

Dog Meeting – *Example*

```
static void Main()
{
    Console.Write("Enter first dog's name: ");
    string dogName = Console.ReadLine();
    Console.Write("Enter first dog's breed: ");
    string dogBreed = Console.ReadLine();
    // Use the Dog constructor to assign name and breed
    Dog firstDog = new Dog(dogName, dogBreed);
    // Use Dog's parameterless constructor
    Dog secondDog = new Dog();
    // Use properties to assign name and breed
    Console.Write("Enter second dog's name: ");
    secondDog.Name = Console.ReadLine();
    Console.Write("Enter second dog's breed: ");
    secondDog.Breed = Console.ReadLine();
```

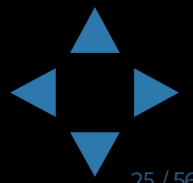
(the example continues)



Telerik Academy

Dog Meeting – *Example*

```
  // Create a Dog with no name and breed
  Dog thirdDog = new Dog();

  // Save the dogs in an array
  Dog[] dogs = new Dog[] {
    firstDog, secondDog, thirdDog };

  // Ask each of the dogs to bark
  foreach(Dog dog in dogs)
  {
      dog.SayBau();
  }
}
```

Telerik Academy

Dog Meeting

Demo

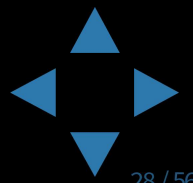
Telerik Academy

Constructors

Telerik Academy

What is a Constructor?

- Constructors are special methods
  - Invoked at the time of creating a new instance of an object
  - Used to initialize the fields of the instance
- Constructors has the same name as the class
  - Have no return type
  - Can have parameters
  - Can be private, protected, internal, public



Telerik Academy

Defining Constructors

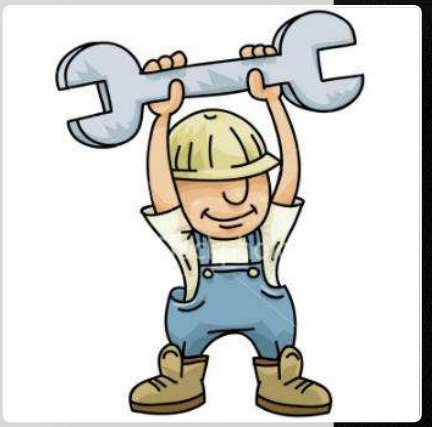
```
public class Point
{
    private int xCoord;
    private int yCoord;

    // Simple parameterless constructor
    public Point()
    {
        this.xCoord = 0;
        this.yCoord = 0;
    }

    // More code …
}
```

- Class Point with parameterless constructor:

Telerik Academy

Defining Constructors

```
public class Person
{
    private string name;
    private int age;
    // Parameterless constructor
    public Person()
    {
        this.name = null;
        this.age = 0;
    }
    // Constructor with parameters
    public Person(string name, int age)
    {
        this.name = name;
        this.age = age;
    }
    // More code …
}
```

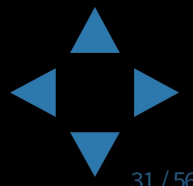


As rule constructors should initialize all own class fields.

Telerik Academy Constructors and Initialization

- Pay attention when using inline initialization!

```
public class AlarmClock
{
    private int hours = 9; // Inline initialization
    private int minutes = 0; // Inline initialization
    // Parameterless constructor (intentionally left empty)
    public AlarmClock()
    { }
    // Constructor with parameters
    public AlarmClock(int hours, int minutes)
    {
            this.hours = hours;       // Invoked after the inline
            this.minutes = minutes;   // initialization!
    }
    // More code …
}
```

Telerik Academy Chaining Constructors Calls

- Reusing constructors (chaining)

```
public class Point
{
    private int xCoord;
    private int yCoord;

    public Point() : this(0, 0) // Reuse the constructor
    {
    }

    public Point(int xCoord, int yCoord)
    {
        this.xCoord = xCoord;
        this.yCoord = yCoord;
    }

    // More code …
}
```

Telerik Academy

Constructors

Demo

Telerik Academy

Methods

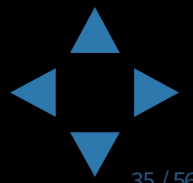
Defining and Invoking Methods

Telerik Academy

Methods

- Methods are class members that execute some action (some code, some algorithm)
  - Could be static or per instance
  - Could be public / private / protected / …

```
public class Point
{
  private int xCoord;
  private int yCoord;
  public double CalcDistance(Point p)
  {
    return Math.Sqrt(
      (p.xCoord - this.xCoord) * (p.xCoord - this.xCoord) +
      (p.yCoord - this.yCoord) * (p.yCoord - this.yCoord));
  }
}
```

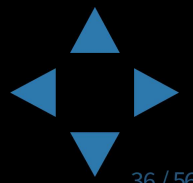



Telerik Academy

Using Methods

- Invoking instance methods is done through the object (class instance):

```
class TestMethods
{
  static void Main()
  {
    Point p1 = new Point(2, 3);
    Point p2 = new Point(3, 4);
    System.Console.WriteLine(p1.CalcDistance(p2));
    }
}
```

Telerik Academy

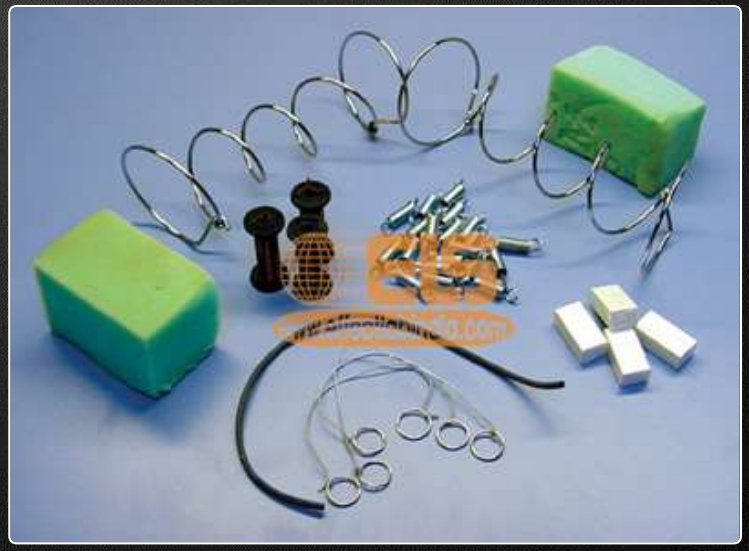
Methods

Demo

Telerik Academy

Properties

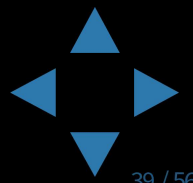
Defining and Using Properties



Telerik Academy

The Role of Properties

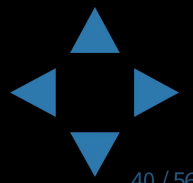
- Properties expose object's data to the world
  - Control how the data is manipulated
    - Ensure the internal object state is correct
    - E.g. price should always be kept positive
- Properties can be:
  - Read-only
  - Write-only (examples)
  - Read and write
- Simplify the writing of code



Telerik Academy

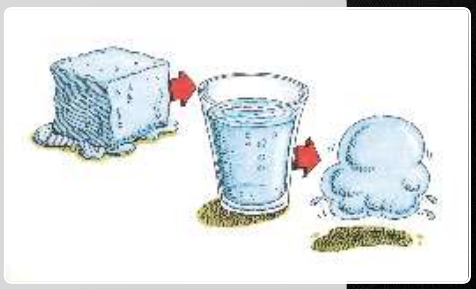
Defining Properties

- Properties work as a pair of methods
  - Getter and setter
- Properties should have:
  - Access modifier (public, protected, etc.)
  - Return type
  - Unique name
  - Get and / or Set part
  - Can contain code processing data in specific way, e.g. apply validation



Telerik Academy Defining Properties – *Example*

```
public class Point
{
    private int xCoord;
    private int yCoord;

    public int XCoord
    {
        get { return this.xCoord; }
            set { this.xCoord = value; }
    }

    public int YCoord
    {
        get { return this.yCoord; }
            set { this.yCoord = value; }
    }

    // More code ...
}
```



Telerik Academy

Dynamic Properties

- Properties are not obligatory bound to a class field – can be calculated dynamically:

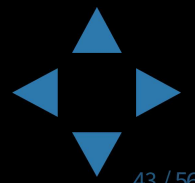
```
public class Rectangle
{
    private double width;
    private double height;

    // More code …

    public double Area
    {
            get
             {
                 return width * height;
             }
    }
}
```

Telerik Academy

Automatic Properties

- Properties could be defined without an underlying field behind them
  - It is automatically created by the compiler

```
class UserProfile
{
    public int UserId { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
}
…
UserProfile profile = new UserProfile() {
    FirstName = "Steve",
    LastName = "Balmer",
    UserId = 91112
};
```

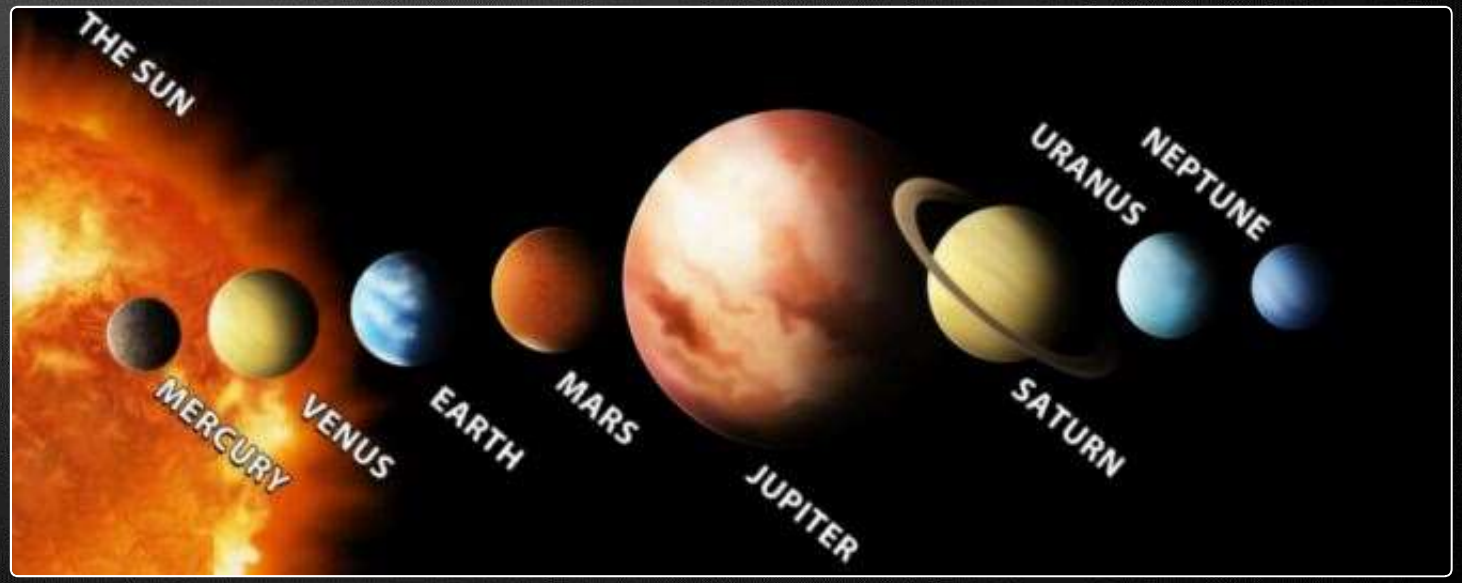
Telerik Academy

Properties

Demo

Telerik Academy

Enumerations

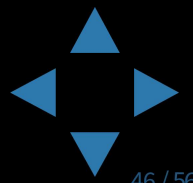


Telerik Academy

Enumerations in C#

- Enumerations are types that hold a value from a fixed set of named constants
  - Declared by enum keyword in C#

```
public enum DayOfWeek
{
  Mon, Tue, Wed, Thu, Fri, Sat, Sun
}
class Enum_Example_
{
  static void Main()
  {
    DayOfWeek day = DayOfWeek.Wed;
    Console.WriteLine(day); // Wed
  }
}
```

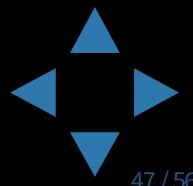


Telerik Academy

Enumerations – *Example*

```
public enum CoffeeSize
{
  Small = 100, Normal = 150, Double = 300
}
public class Coffee
{
  private CoffeeSize size;
  public Coffee(CoffeeSize size)
  {
    this.size = size;
  }
  public CoffeeSize Size
  {
    get { return this.size; }
  }
}
```

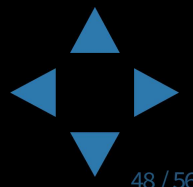
(the example continues)



Telerik Academy

Enumerations – *Example*

```
public class CoffeeMachine
{
  static void Main()
  {
    Coffee normalCoffee = new Coffee(CoffeeSize.Normal);
    Coffee doubleCoffee = new Coffee(CoffeeSize.Double);

    Console.WriteLine("The {0} coffee is {1} ml.",
      normalCoffee.Size, (int)normalCoffee.Size);
    // The Normal coffee is 150 ml.

    Console.WriteLine("The {0} coffee is {1} ml.",
      doubleCoffee.Size, (int)doubleCoffee.Size);
    // The Double coffee is 300 ml.
    }
}
```



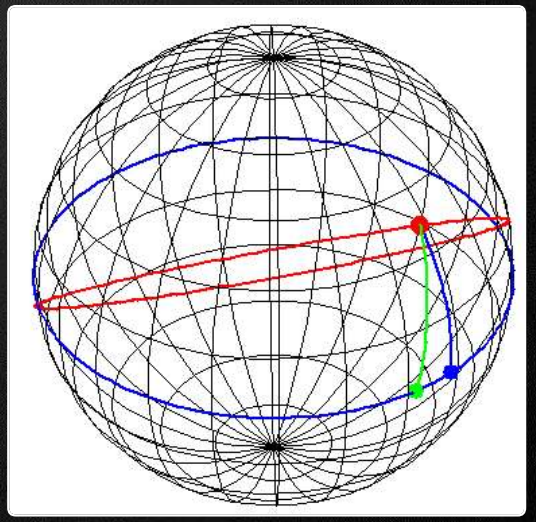
Telerik Academy

Enumerations

Demo

Telerik Academy

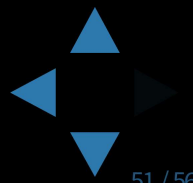
Keeping the Object State Correct



Telerik Academy

Keep the Object State Correct

- Constructors and properties can keep the object's state correct
  - This is known as encapsulation in OOP
  - Can force validation when creating / modifying the object's internal state
  - Constructors define which properties are mandatory and which are optional
  - Property setters should validate the new value before saving it in the object field
  - Invalid values should cause an exception



Telerik Academy

Keep the Object State – *Example*

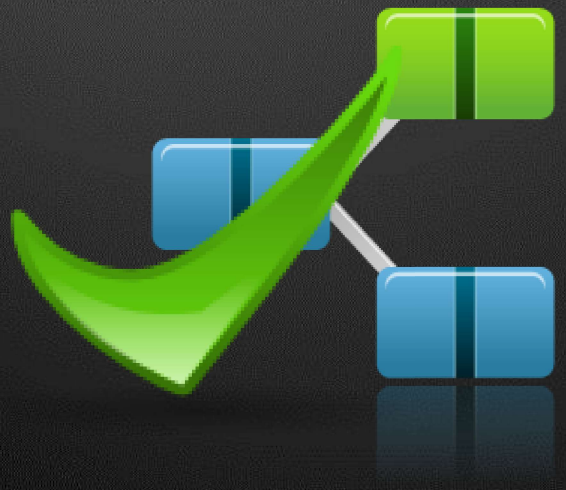
```
public class Person
{
    private string name;
    public Person(string name)
    {
        this.Name = name;
    }
    public string Name
    {
        get { return this.name; }
        set
        {
            if (String.IsNullOrEmpty(value))
                throw new ArgumentException("Invalid name!");
            this.name = value;
        }
    }
}
```

We have only one constructor, so we cannot create person without specifying a name.

Incorrect name cannot be assigned

Telerik Academy

Keeping the Object State Correct



Demo

Telerik Academy

Summary

- Classes define specific structure for objects
  - Objects are particular instances of a class
- Classes define fields, methods, constructors, properties and other members
  - Access modifiers limit the access to class members
- Constructors are invoked when creating new class instances and initialize the object's internal state
- Enumerations define a fixed set of constants
- Properties expose the class data in safe, controlled way