Telerik Academy

http://csharpfundamentals.telerik.com

Using Classes and Objects

Using the Standard .NET Framework Classes

Telerik Software Academy
Learning & Development Team
http://academy.telerik.com

Telerik Academy

Table of Contents

Telerik Academy

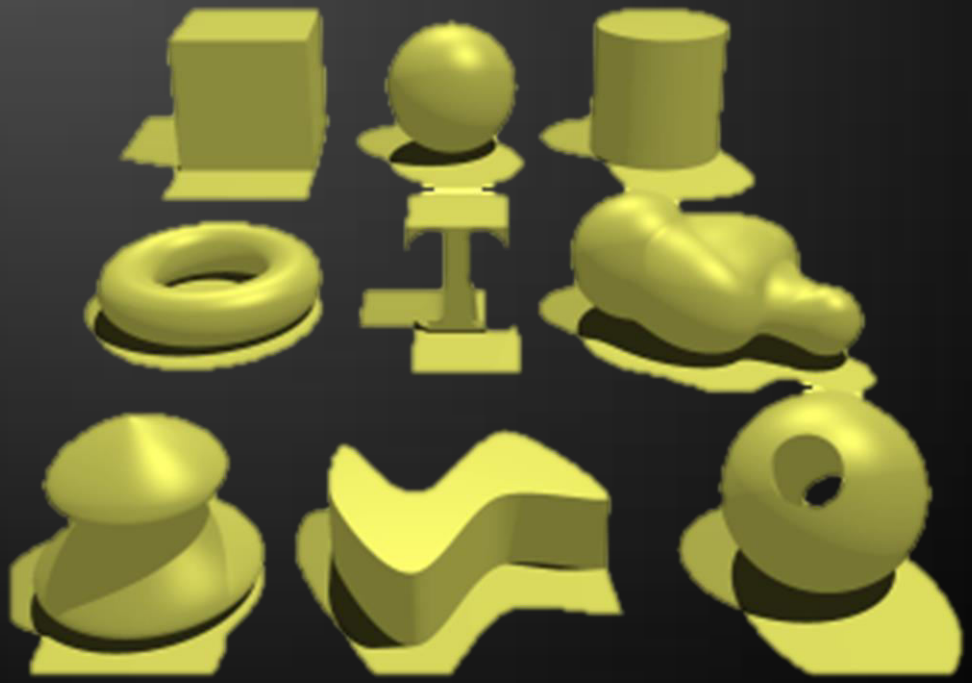
Table of Contents (2)

4. Structures
5. Namespaces
6. Random class
7. Introduction to .NET Common Type System

Telerik Academy

Classes and Objects

Modeling Real-world Entities with Objects

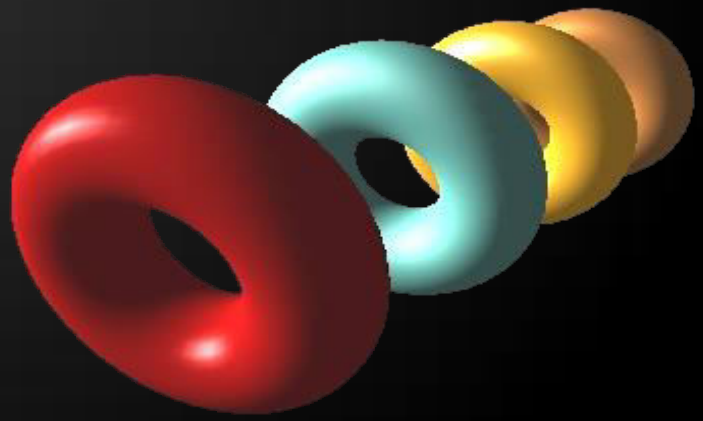


What are Objects?

- Software objects model real-world objects or abstract concepts
  - Examples:
    - bank, account, customer, dog, bicycle, queue
- Real-world objects have states and behaviors
  - Account' states:
    - holder, balance, type
  - Account' behaviors:
    - withdraw, deposit, suspend

What are Objects? (2)

- How do software objects implement real-world objects?
  - Use variables/data to implement states
  - Use methods/functions to implement behaviors
- An object is a software bundle of variables and related methods



Objects Represent

- checks
- people
- shopping list

Things from the real world

...

- numbers
- characters
- queues
- arrays

Things from the computer world

What is a Class?

- The formal definition of class:

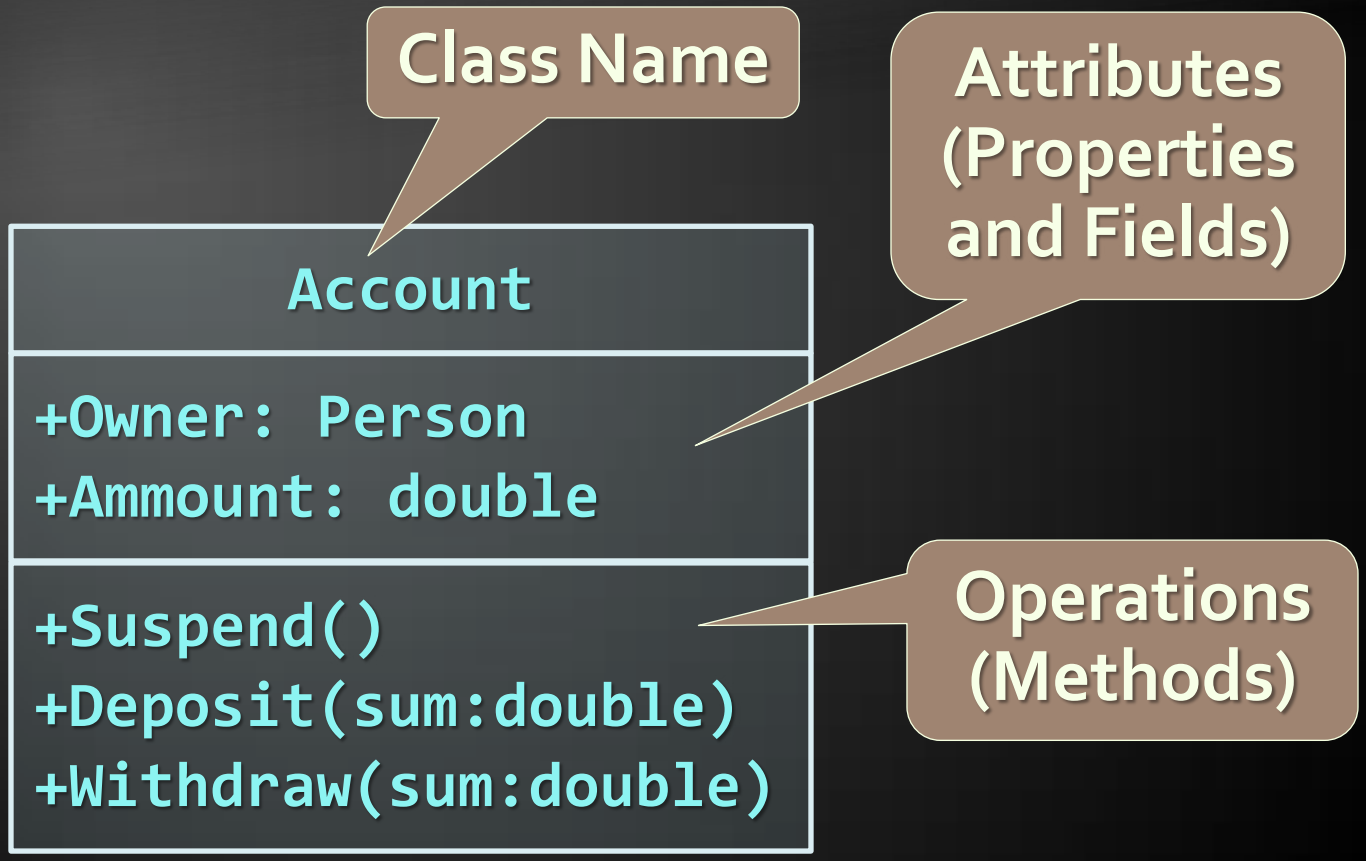
> Classes act as templates from which an instance of an object is created at run time. Classes define the properties of the object and the methods used to control the object's behavior.

Definition by Google

Classes

- Classes provide the structure for objects
  - Define their prototype, act as template
- Classes define:
  - Set of attributes
    - Represented by variables and properties
    - Hold their state
  - Set of actions (behavior)
    - Represented by methods
- A class defines the methods and types of data associated with an object

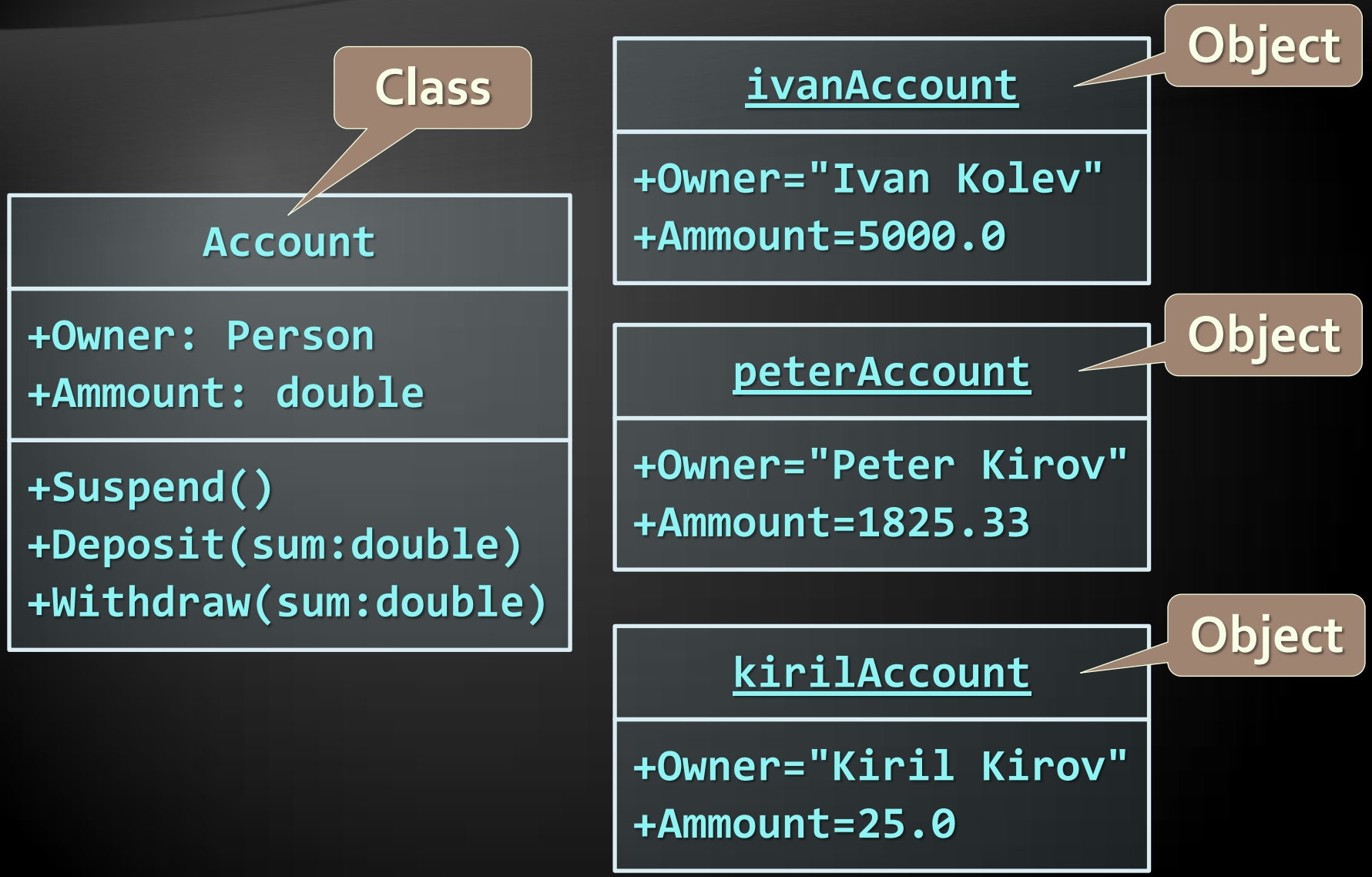
Classes – Example



Objects

- An object is a concrete instance of a particular class
- Creating an object from a class is called instantiation
- Objects have state
  - Set of values associated to their attributes
- Example:
  - Class: Account
  - Objects: Ivan's account, Peter's account

Objects – Example



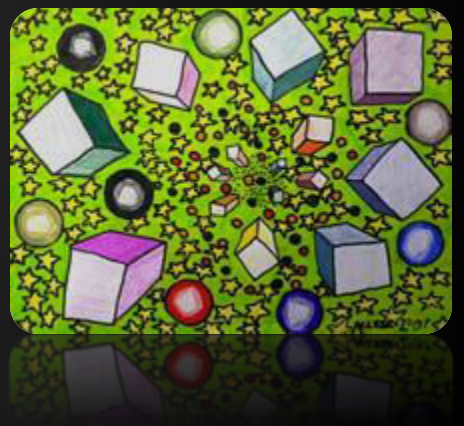
Telerik Academy

Classes in C#

Using Classes and their Class Members

Classes in C#

- Classes – basic units that compose programs
- Implementation is encapsulated (hidden)
- Classes in C# can contain:
  - Fields (member variables)
  - Properties
  - Methods
  - Constructors
  - Inner types
  - Etc. (events, indexers, operators, …)



Telerik Academy

Classes in C# – Examples

- Example of classes (structures):
  - `System.Console`
  - `System.String` (`string` in C#)
  - `System.Int32` (`int` in C#)
  - `System.Array`
  - `System.Math`
  - `System.Random`
  - `System.DateTime`
  - `System.Collections.Generics.List<T>`

Declaring Objects

- An instance of a class or structure can be defined like any other variable:

```
using System;
...
// Define two variables of type DateTime
DateTime today;
DateTime halloween;
```

- Instances cannot be used if they are not initialized

```
// Declare and initialize a structure instance
DateTime today = DateTime.Now;
```

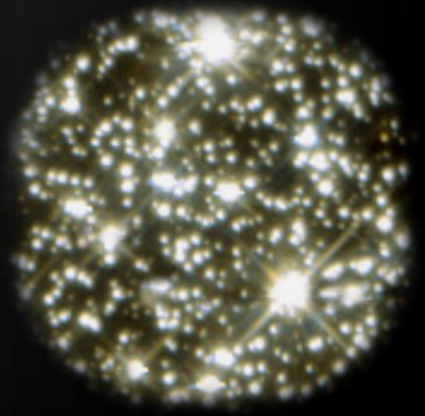
Telerik Academy

Fields and Properties

Accessing Fields and Properties

Fields

- Fields are data members of a class
  - Can be variables and constants (read-only)
- Accessing a field doesn't invoke any actions of the object
  - Just accesses its value
- Example:
  - `String.Empty` (the `""` string)



Accessing Fields

- Constant fields can be only read
- Variable fields can be read and modified
- Usually properties are used instead of directly accessing variable fields
- Examples:

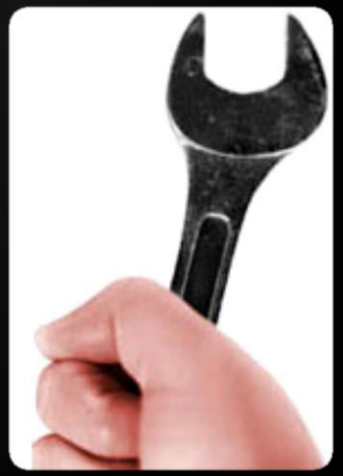
```
// Accessing read-only field
String empty = String.Empty;

// Accessing constant field
int maxInt = Int32.MaxValue;
```

Properties

- Properties look like fields
  - Have name and type
  - Can contain code, executed when accessed
- Usually used as wrappers
  - To control the access to the data fields
  - Can contain more complex logic
- Can have two components called accessors
  - get for reading their value
  - set for changing their value

Properties (2)

- According to the implemented accessors properties can be:
  - Read-only (get accessor only)
  - Read and write (both get and set accessors)
  - Write-only (set accessor only)
- Example of read-only property:
  - `String.Length`
- Example of read-write property:
  - `Console.BackgroundColor`



Accessing Properties and Fields – Example

```
using System;

...

DateTime christmas = new DateTime(2009, 12, 25);
int day = christmas.Day;
int month = christmas.Month;
int year = christmas.Year;
Console.WriteLine(
  "Christmas day: {0}, month: {1}, year: {2}",
  day, month, year);
Console.WriteLine(
  "Day of year: {0}", christmas.DayOfYear);
Console.WriteLine("Is {0} leap year: {1}",
  year, DateTime.IsLeapYear(year));
```

Telerik Academy

Accessing Properties and Fields

Live Demo

Telerik Academy

Instance and Static Members

Accessing Object and Class Members

Telerik Academy

Instance and Static Members

- Fields, properties and methods can be:
  - Instance (or object members)
  - Static (or class members)
- Instance members are specific for each object
  - Example: different dogs have different name
- Static members are common for all instances of a class
  - Example: `DateTime.MinValue` is shared between all instances of `DateTime`

Telerik Academy

Accessing Members – Syntax

- Accessing instance members
  - The name of the instance, followed by the name of the member (field or property), separated by dot (".")

```
<instance_name>.<member_name>
```

- Accessing static members
  - The name of the class, followed by the name of the member

```
<class_name>.<member_name>
```

Instance and Static Members – Examples

- Example of instance member
  - `String.Length`
    - Each string object has a different length
    - E.g. `"I like C#".Length` → `9`
- Example of static member
  - `Console.ReadLine()`
    - The console is only one (global for the program)
    - Reading from the console does not require to create an instance of it

Telerik Academy

Methods

Calling Instance and Static Methods

Methods

- Methods manipulate the data of the object to which they belong or perform other tasks
- Examples:
  - `Console.WriteLine(…)`
  - `Console.ReadLine()`
  - `String.Substring(index, length)`
  - `Array.GetLength(index)`
  - `List<T>.Add(item)`
  - `DateTime.AddDays(count)`

Instance Methods

- Instance methods manipulate the data of a specified object or perform any other tasks
  - If a value is returned, it depends on the particular class instance
- Syntax:
  - The name of the instance, followed by the name of the method, separated by dot

```
<object_name>.<method_name>(<parameters>)
```

Calling Instance Methods – Examples

- Calling instance methods of `String`:

```
String sampleLower = new String('a', 5);
String sampleUpper = sampleLower.ToUpper();

Console.WriteLine(sampleLower); // aaaaa
Console.WriteLine(sampleUpper); // AAAAA
```

- Calling instance methods of `DateTime`:

```
DateTime now = DateTime.Now;
DateTime later = now.AddHours(8);

Console.WriteLine("Now: {0}", now);
Console.WriteLine("8 hours later: {0}", later);
```

Telerik Academy

Calling Instance Methods

Live Demo

Static Methods

- Static methods are common for all instances of a class (shared between all instances)
  - Returned value depends only on the passed parameters
  - No particular class instance is available
- Syntax:
  - The name of the class, followed by the name of the method, separated by dot

```
<class_name>.<method_name>(<parameters>)
```

Telerik Academy

Calling Static Methods – Examples

```
using System;

double radius = 2.9;
double area = Math.PI * Math.Pow(radius, 2);
Console.WriteLine("Area: {0}", area);
// Area: 26,4207942166902

double precise = 8.7654321;
double round3 = Math.Round(precise, 3);
double round1 = Math.Round(precise, 1);
Console.WriteLine(
    "{0}; {1}; {2}", precise, round3, round1);
// 8,7654321; 8,765; 8,8
```

Constant field

Static method

Static method

Static method

Telerik Academy

Calling Static Methods

Live Demo

Constructors

- Constructors are special methods used to assign initial values of the fields in an object
  - Executed when an object of a given type is being created
  - Have the same name as the class that holds them
  - Do not return a value
- A class may have several constructors with different set of parameters

Constructors (2)

- Constructor is invoked by the new operator

```
<instance_name> = new <class_name>(<parameters>)
```

- Examples:

```
String s = new String(new char[]{'a','b','c'});
```

```
String s = new String('*', 5); // s = "*****"
```

```
DateTime dt = new DateTime(2009, 12, 30);
```

```
DateTime dt = new DateTime(2009, 12, 30, 12, 33, 59);
```

```
Int32 value = new Int32();
```

Telerik Academy

Parameterless Constructors

- The constructor without parameters is called default (parameterless) constructor
- Example:
  - Creating an object for generating random numbers with a default seed

```
using System;
...
Random randomGenerator = new Random();
```

Parameterless constructor call

The class `System.Random` provides generation of pseudo-random numbers

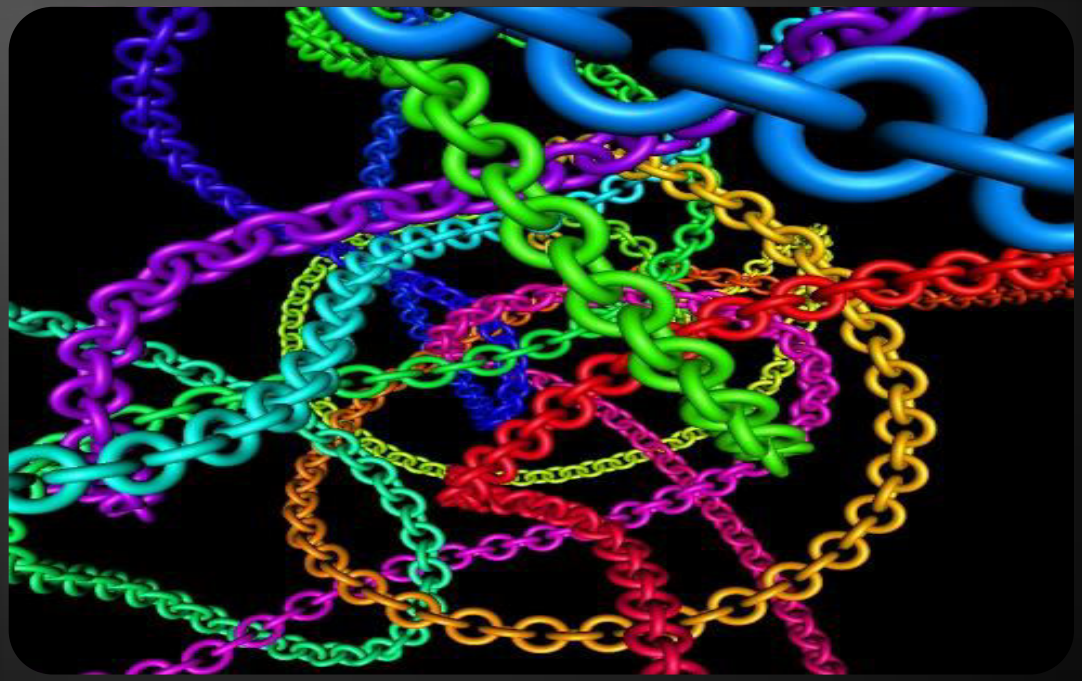
Constructor with Parameters

- Example
  - Creating objects for generating random values with specified initial seeds

```
using System;
...
Random randomGenerator1 = new Random(123);
Console.WriteLine(randomGenerator1.Next());
// 2114319875

Random randomGenerator2 = new Random(456);
Console.WriteLine(randomGenerator2.Next(50));
// 47
```

Telerik Academy

Generating Random Numbers



Live Demo

Telerik Academy

More Constructor Examples

- Creating a DateTime object for a specified date and time

```
using System;

DateTime halloween = new DateTime(2009, 10, 31);
Console.WriteLine(halloween);

DateTime julyMorning =
  new DateTime(2009,7,1, 5,52,0);
Console.WriteLine(julyMorning);
```

- Different constructors are called depending on the different sets of parameters

Telerik Academy

Creating DateTime Objects

Live Demo

Telerik Academy

Enumerations

Types Limited to a Predefined Set of Values

Enumerations

- Enumerations in C# are types whose values are limited to a predefined set of values
  - E.g. the days of week
  - Declared by the keyword enum in C#
  - Hold values from a predefined set

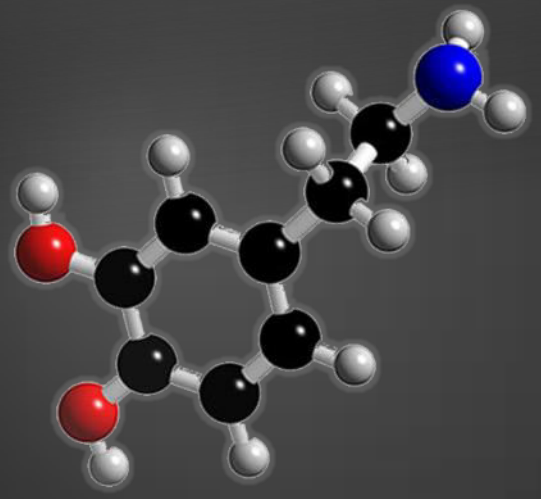
```
public enum Color { Red, Green, Blue, Black }
…
Color color = Color.Red;
Console.WriteLine(color); // Red
color = 5; // Compilation error!
```

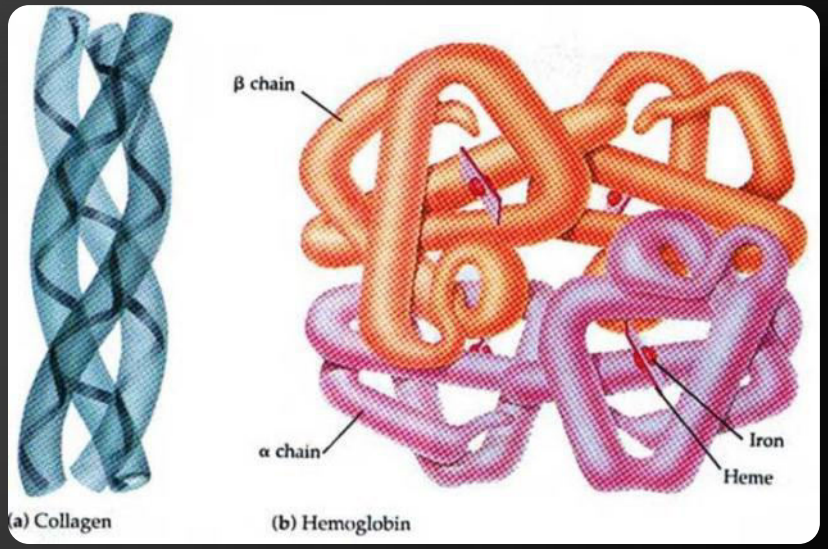
Telerik Academy

Enumerations

Live Demo

Telerik Academy





Structures

What are Structures? When to Use Them?

Structures

- Structures in C# are similar to classes
  - Structures are value types (directly hold a value)
  - Classes are reference types (pointers)
- Structures are usually used for storing data structures, without any other functionality
- Structures can have fields, properties, etc.
  - Using methods is not recommended
- Example of structure
  - `System.DateTime` – represents a date and time

Telerik Academy

Namespaces

Organizing Classes Logically into Namespaces

Telerik Academy

What is a Namespace?

- Namespaces are used to organize the source code into more logical and manageable way
- Namespaces can contain
  - Definitions of classes, structures, interfaces and other types and other namespaces
- Namespaces can contain other namespaces
- For example:
  - `System` namespace contains `Data` namespace
  - The name of the nested namespace is `System.Data`

Full Class Names

- A full name of a class is the name of the class preceded by the name of its namespace

```
<namespace_name>.<class_name>
```

- Example:
  - `Array` class, defined in the `System` namespace
  - The full name of the class is `System.Array`

Telerik Academy

Including Namespaces

- The using directive in C#:

```
using <namespace_name>
```

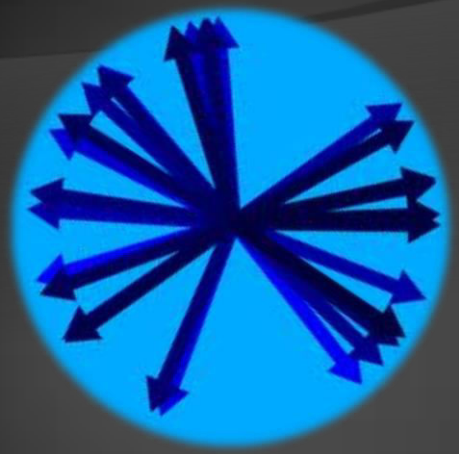
- Allows using types in a namespace, without specifying their full name

  Example:

```
using System;
DateTime date;
```

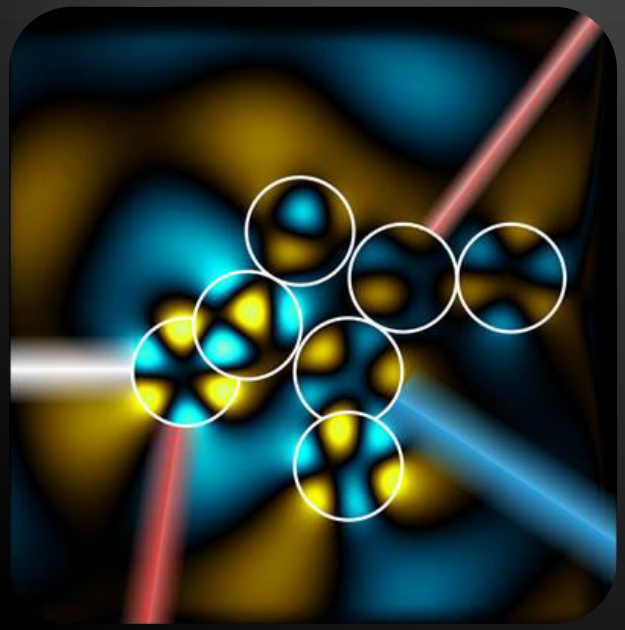
  instead of

```
System.DateTime date;
```



The Random Class

Password Generator Demo



The Random Class

- The Random class
  - Generates random integer numbers

```
Random rand = new Random();
for (int number = 1; number <= 6; number++)
{
    int randomNumber = rand.Next(49) + 1;
    Console.Write("{0} ", randomNumber);
}
```

- This generates 6 random int in range [1..49]
- Always use a single Random instance!
  - This will avoid abnormalities

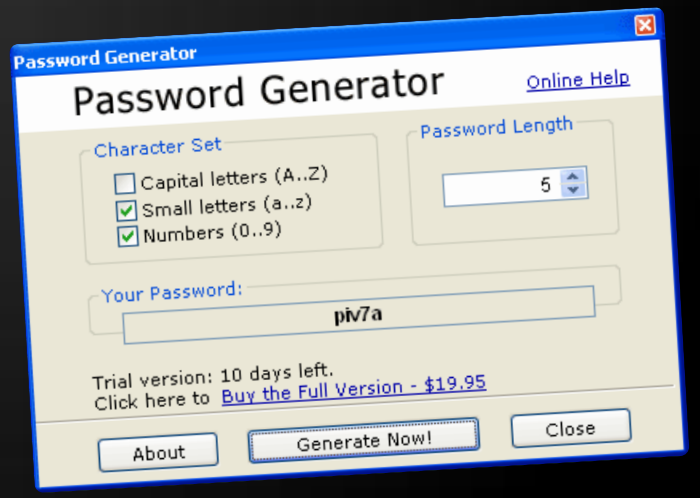
Telerik Academy

Password Generator – Example

- Write a program to generate a random password between 8 and 15 characters
  - The password contains of at least two capital letters, two small letters, one digit and three special characters
- Constructing the password generator class:
  - Start from an empty password
  - Place 2 random capital letters at random positions
  - Place 2 random small letters at random positions
  - Place 1 random digit at random positions
  - Place 3 special characters at random positions

Password Generator (2)

- Now we have exactly 8 characters
  - To make the password length between 8 and 15 we add between 0 and 7 random characters
    - Capital / small letters / digits / special character
    - Inserts each of them at random position



Telerik Academy

Password Generator Class

```
class RandomPasswordGenerator
{
   private const string CapitalLetters=
    "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
   private const string SmallLetters =
    "abcdefghijklmnopqrstuvwxyz";
   private const string Digits = "0123456789";
   private const string SpecialChars =
    "~!@#$%^&*()_+=`{}[]\\|':;.,/?<>";
   private const string AllChars =
    CapitalLetters + SmallLetters + Digits + SpecialChars;
   private static Random rnd = new Random();

 // the example continues…
```

Password Generator Class (2)

```
static void Main()
{
    StringBuilder password = new StringBuilder();
    for (int i = 1; i <= 2; i++)
    {
        char capitalLetter = GenerateChar(CapitalLetters);
        InsertAtRandomPosition(password, capitalLetter);
    }
    for (int i = 1; i <= 2; i++)
    {
        char smallLetter = GenerateChar(SmallLetters);
        InsertAtRandomPosition(password, smallLetter);
    }
    char digit = GenerateChar(Digits);
    InsertAtRandomPosition(password, digit);
    for (int i = 1; i <= 3; i++)
    {
        char specialChar = GenerateChar(SpecialChars);
        InsertAtRandomPosition(password, specialChar);
    }
// the example continues…
```

Password Generator Class (3)

```
    int count = rnd.Next(8);
    for (int i = 1; i <= count; i++)
    {
        char specialChar = GenerateChar(AllChars);
        InsertAtRandomPosition(password, specialChar);
    }
    Console.WriteLine(password);
}

private static void InsertAtRandomPosition(
    StringBuilder password, char character)
{
    int randomPosition = rnd.Next(password.Length + 1);
    password.Insert(randomPosition, character);
}

private static char GenerateChar(string availableChars)
{
    int randomIndex = rnd.Next(availableChars.Length);
    char randomChar = availableChars[randomIndex];
    return randomChar;
}
```

Telerik Academy

.NET Common Type System

Brief Introduction

Telerik Academy

Common Type System (CTS)

- CTS defines all data types supported in .NET Framework
  - Primitive types (e.g. `int`, `float`, `object`)
  - Classes (e.g. `String`, `Console`, `Array`)
  - Structures (e.g. `DateTime`)
  - Arrays (e.g. `int[]`, `string[,]`)
  - Etc.
- Object-oriented by design

Telerik Academy

CTS and Different Languages

- CTS is common for all .NET languages
  - C#, VB.NET, J#, JScript.NET, …
- CTS type mappings:

CTS Type	C# Type	VB.NET Type
System.Int32	int	Integer
System.Single	float	Single
System.Boolean	bool	Boolean
System.String	string	String
System.Object	object	Object

Telerik Academy

System.Object: CTS Base Type

- System.Object (object in C#) is a base type for all other types in CTS
  - Can hold values of any other type:

```
string s = "test";
object obj = s;
```

- All .NET types derive common methods from System.Object, e.g. ToString()

```
DateTime now = DateTime.Now;
string nowInWords = now.ToString();
Console.WriteLine(nowInWords);
```

Telerik Academy

Value and Reference Types

- In CTS there are two categories of types
  - Value types
  - Reference types
- Placed in different areas of memory
  - Value types live in the execution stack*
    - Freed when become out of scope
  - Reference types live in the managed heap (dynamic memory)
    - Freed by the garbage collector

* Note: this does not mean that value types, which are part of reference types live on the stack. E.g., integers in a List<int> do not live on the stack

Telerik Academy

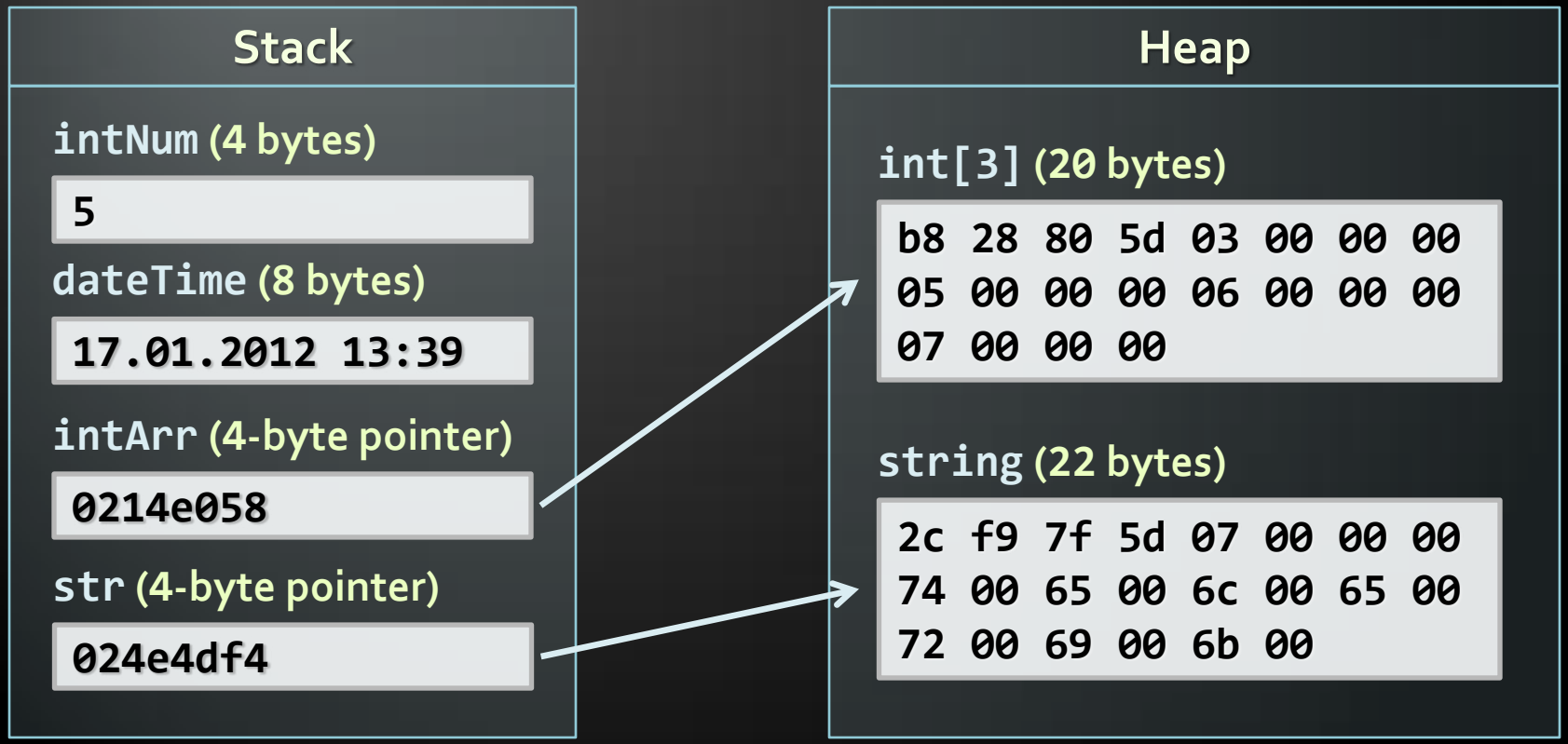
Value and Reference Types – Examples

- Value types
  - Most of the primitive types
  - Structures
  - Examples: `int`, `float`, `bool`, `DateTime`
- Reference types
  - Classes and interfaces
  - Strings
  - Arrays
  - Examples: `string`, `Random`, `object`, `int[]`

Telerik Academy

Value and Reference Types

```
int intNum = 5;
DateTime date = DateTime.Now;
int[] intArr = new int[] {5, 6, 7};
string str = "telerik";
```



Summary

- Classes provide the structure for objects
- Objects are particular instances of classes
- Classes have different members
  - Methods, fields, properties, etc.
  - Instance and static members
  - Members can be accessed
  - Methods can be called
- Structures are used for storing data
- Namespaces group related classes

Summary (2)

- Namespaces help organizing the classes
- Common Type System (CTS) defines the types for all .NET languages
  - Values types
  - Reference types

Telerik Academy

Using Classes and Objects

Questions?

http://csharpfundamentals.telerik.com

Exercises

1. Write a program that reads a year from the console and checks whether it is a leap. Use `DateTime`.
2. Write a program that generates and prints to the console 10 random values in the range [100, 200].
3. Write a program that prints to the console which day of the week is today. Use `System.DateTime`.
4. Write methods that calculate the surface of a triangle by given:
   - Side and an altitude to it; Three sides; Two sides and an angle between them. Use `System.Math`.

Exercises (2)

5. Write a method that calculates the number of workdays between today and given date, passed as parameter. Consider that workdays are all days from Monday to Friday except a fixed list of public holidays specified preliminary as array.
6. You are given a sequence of positive integer values written into a string, separated by spaces. Write a function that reads these values from given string and calculates their sum. Example:

   string = "43 68 9 23 318" → result = 461

Exercises (3)

7. * Write a program that calculates the value of given arithmetical expression. The expression can contain the following elements only:
   - Real numbers, e.g. `5`, `18.33`, `3.14159`, `12.6`
   - Arithmetic operators: `+`, `-`, `*`, `/` (standard priorities)
   - Mathematical functions: `ln(x)`, `sqrt(x)`, `pow(x,y)`
   - Brackets (for changing the default priorities)

   Examples:

   `(3+5.3) * 2.7 - ln(22) / pow(2.2, -1.7)` → `~ 10.6`

   `pow(2, 3.14) * (3 - (3 * sqrt(2) - 3.2) + 1.5*0.3)` → `~ 21.22`

   Hint: Use the classical "shunting yard" algorithm and "reverse Polish notation".

Free Trainings @ Telerik Academy

- "C# Programming @ Telerik Academy
  - csharpfundamentals.telerik.com

- Telerik Software Academy
  - academy.telerik.com

- Telerik Academy @ Facebook
  - facebook.com/TelerikAcademy

- Telerik Software Academy Forums
  - forums.academy.telerik.com