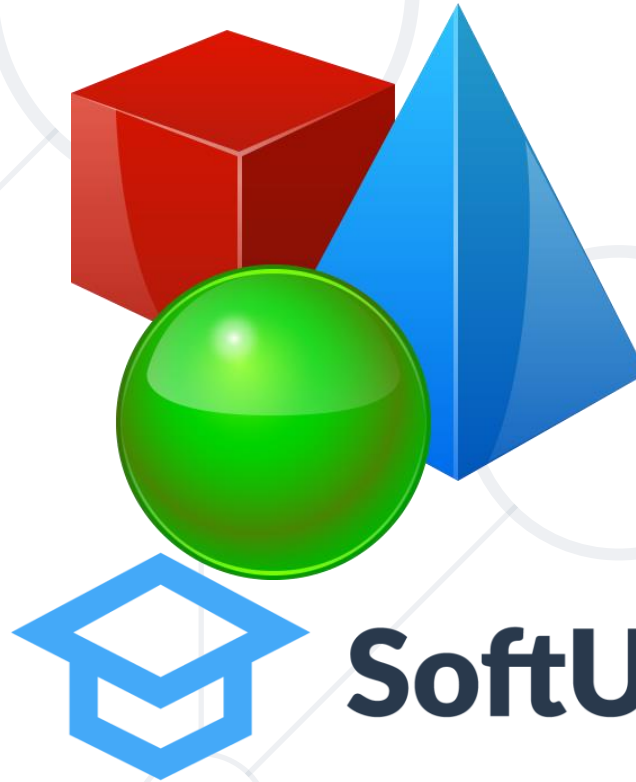


# Objects and Classes

Using Objects and Classes  
Defining Simple Classes



**SoftUni**

**SoftUni Team**  
**Technical Trainers**



**Software University**

<https://softuni.bg>

sli.do

**#fund-csharp**

1. Objects
2. Classes
3. Built-in Classes
4. Defining Simple Classes
  - Properties
  - Methods
  - Constructors





# **Objects and Classes**

What is an Object? What is a Class?

# Classes

- In programming, classes provide the structure for **objects**
  - Act as **templates** for **objects** of the same type
- Classes define
  - **Properties** (data), e.g., **Name**, **Age**
  - **Behaviors** (actions), e.g., **Bark()**
- One class may have many instances (objects)
  - Sample class: **Dog**
  - Sample objects: **sparky**, **rufus**



# Classes – Example

```
class Dog
```

Name

```
{
```

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

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

```
    public int Age { get; set; }
```

Properties

```
    public void Bark()
```

Method

```
{
```

```
        Console.WriteLine("Bark!");
```

```
}
```

```
}
```

# Objects

- An **object** holds a set of named values
  - Creating a **Dog** object

**Dog**

Name = "Sparky"

Breed = "Corgi"

Age = 3

Object  
name

Object  
properties

Create a **new** object of  
type Dog

```
var puppy = new Dog  
("Sparky", "Corgi", 3);  
Console.WriteLine(puppy);
```

The **new** operator creates  
a new object

```
var puppy = new Dog {Name = "Sparky", Breed = "Corgi", Age = 3 };
```



# Objects – Instances of Classes

- Creating the object of a defined class is called **instantiation**
- The **instance** is the object itself, which is created runtime
- All instances have common **behaviour**

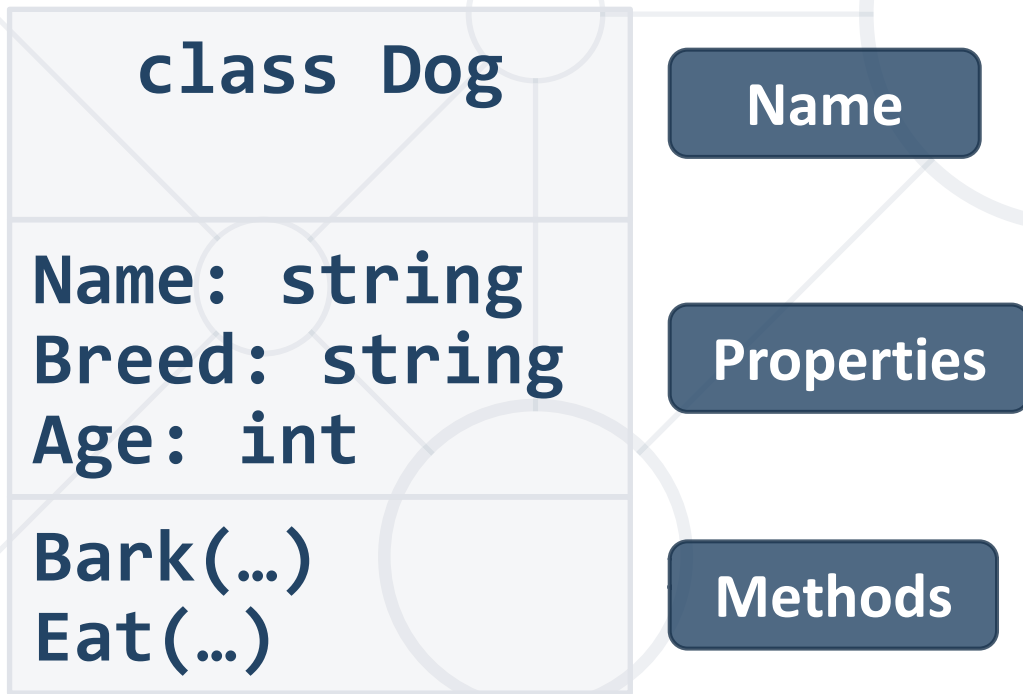


```
Dod sparky = new Dog("Sparky", "Corgi", 5);  
Dog rufus = new Dog("Rufus", "Shepherd", 3);  
Dog allie = new Dog("Allie", "Husky", 2);
```

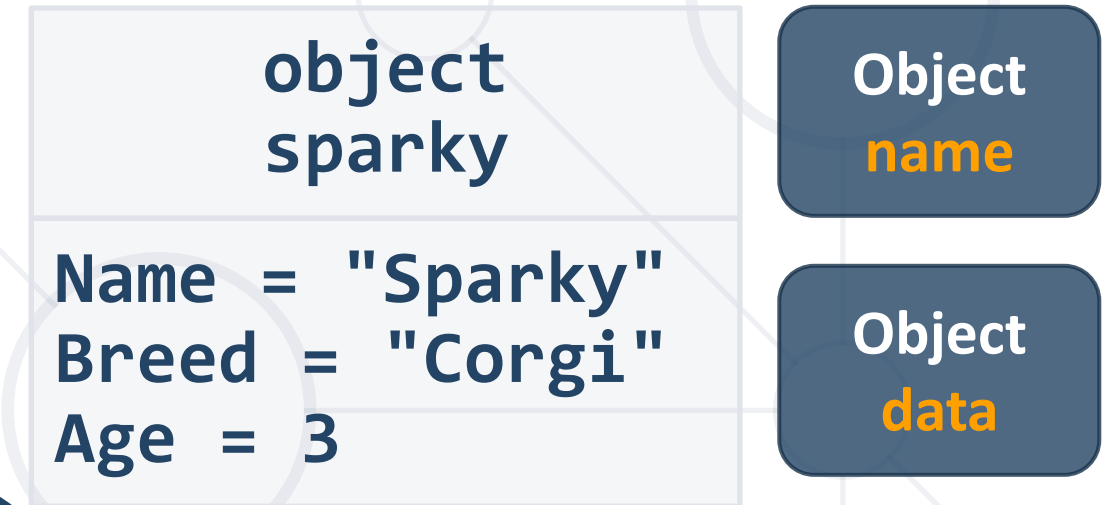


# Classes vs Objects

- Classes provide **structure** for creating objects



- An object is a single instance of a class





**Math.Max()**

# **Using the Built-in API Classes**

Math, Random, BigInteger, ...

- .NET Core provides thousands of ready-to-use classes
  - Packaged into namespaces like **System**, **System.Text**, **System.Collections**, **System.Linq**, **System.Net**, etc.

- Using static .NET class members

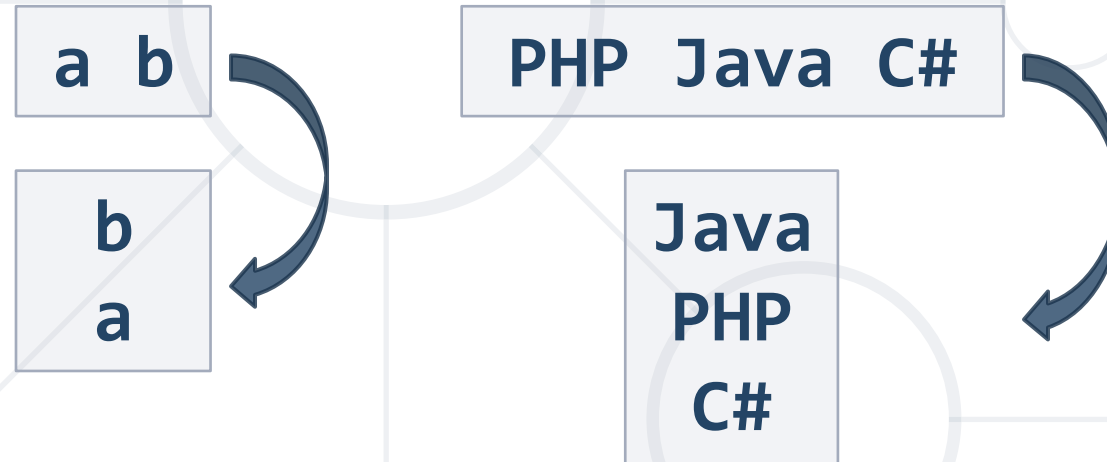
```
double cosine = Math.Cos(Math.PI);
```

- Using non-static .NET classes

```
Random rnd = new Random();  
int randomNumber = rnd.Next(1, 99);
```

# Problem: Randomize Words

- You are given a list of words
  - Randomize their order and print each word at a separate line



**Note: The output is a sample. It should always be different!**

Check your solution here: <https://judge.softuni.org/Contests/Practice/Index/1214#0>

# Solution: Randomize Words

```
string[] words = Console.ReadLine().Split(' ');
Random rnd = new Random();
for (int pos1 = 0; pos1 < words.Length; pos1++)
{
    int pos2 = rnd.Next(words.Length);
    // TODO: Swap words[pos1] with words[pos2]
}
Console.WriteLine(string.Join(Environment.NewLine, words));
```

Check your solution here: <https://judge.softuni.org/Contests/Practice/Index/1214#0>

# Problem: Big Factorial

- Calculate  $n!$  ( $n$  factorial) for very big  $n$  (e.g. 1000)

5 → 120      10 → 3628800      12 → 479001600

50 → 3041409320171337804361260816606476884437764156  
8960512000000000000

88 → 1854826422573984391147968456455462843802209689  
4939934668442158098688956218402819931910014124  
480450182841663351685120000000000000000000000000000

Check your solution here: <https://judge.softuni.org/Contests/Practice/Index/1214#1>

# Solution: Big Factorial

Use the .NET API class  
`System.Numerics  
.BigInteger`

```
using System.Numerics;

int n = int.Parse(Console.ReadLine());
BigInteger f = 1;
for (int i = 2; i <= n; i++)
    f *= i;
Console.WriteLine(f);
```

N!

Check your solution here: <https://judge.softuni.org/Contests/Practice/Index/1214#1>



# Creating Custom Classes

Defining Classes



# Defining Simple Classes

- Specification of a given type of objects from the real-world
- **Classes** provide structure for describing and creating objects



Keyword

Class **name**

```
class Dice
```

```
{
```

```
...
```

Class **body**

```
}
```

# Naming Classes

- Use **PascalCase** naming
- Use descriptive nouns
- Avoid abbreviations (except widely known, e.g., URL, HTTP, etc.)



```
class Dice { ... }  
class BankAccount { ... }  
class IntegerCalculator { ... }
```



```
class TPMF { ... }  
class bankaccount { ... }  
class intcalc { ... }
```

- Class is made up of **state** and **behaviour**
- Properties **store state**
- Methods **describe behaviour**

```
class Dice
```

```
{
```

```
    public int Sides { get; set; }
```

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

```
    public void Roll() { }
```

```
}
```

Properties

Method



- A class can have **many instances** (objects)

```
class Program
{
    public static void Main()
    {
        Dice diceD6 = new Dice();
        Dice diceD8 = new Dice();
    }
}
```

Use the **new** keyword



- Describe the characteristics of a given class

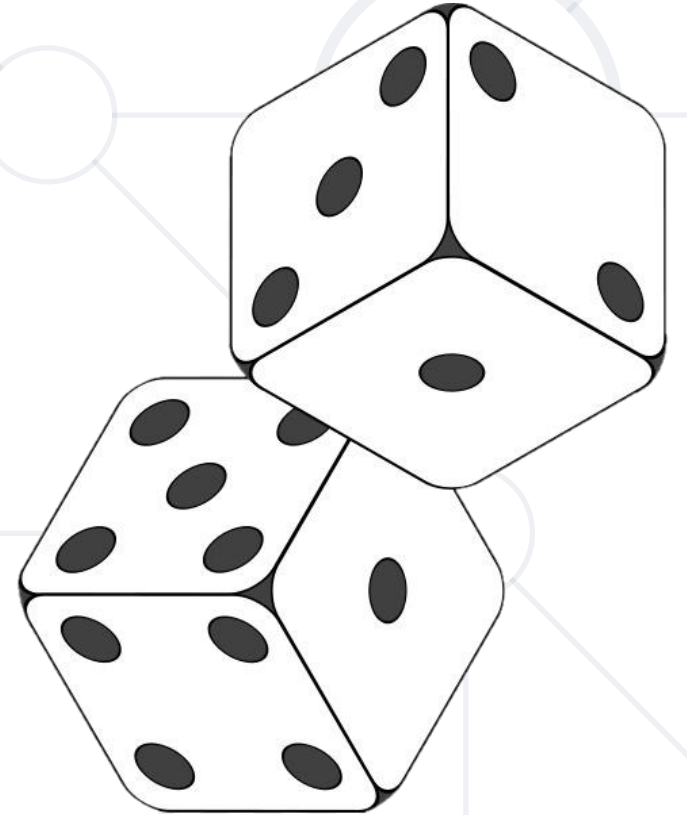
```
class Student
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public int Age { get; set; }
}
```

The **getter** provides access to the field

The **setter** provides field change

- Store **executable code** (algorithm)

```
class Dice
{
    public int Sides { get; set; }
    public int Roll()
    {
        Random rnd = new Random();
        return rnd.Next(1, Sides + 1);
    }
}
```

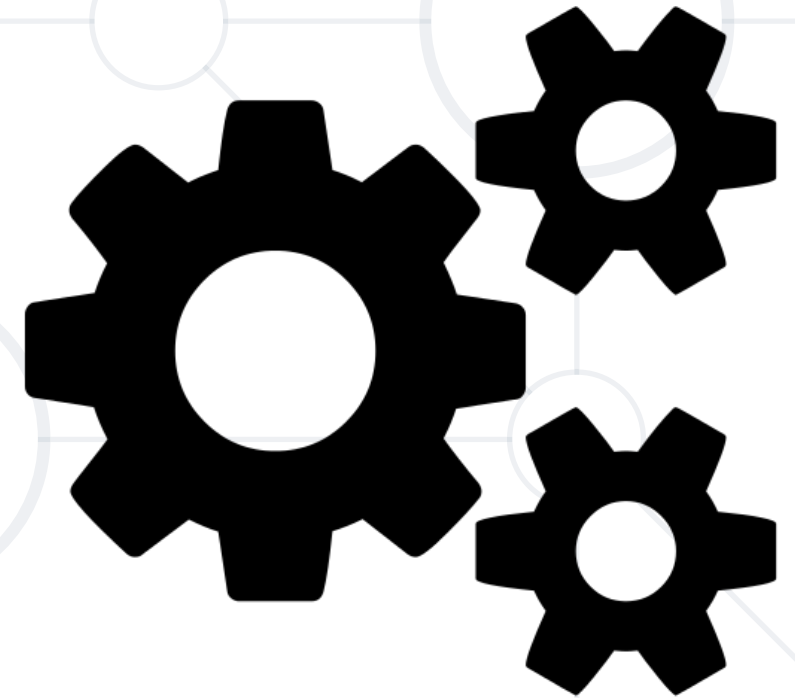


- Special methods, executed during object creation

```
class Dice
{
    public int Sides { get; set; }
    public Dice()
    {
        this.Sides = 6;
    }
}
```

**Constructor name** is the same as the name of the class

**Overloading** default constructor



- You can have multiple constructors in the same class

```
class Dice
{
    public Dice() { }
    public Dice(int sides)
    {
        this.Sides = sides;
    }
    p int Sides { get; set; }
}
```

```
class Program
{
    public static void Main()
    {
        Dice dice1 = new Dice();
        Dice dice2 = new Dice(7);
    }
}
```



- Classes can define **data** (state) and **operations** (actions)

```
class Rectangle
{
    public int Top { get; set; }
    public int Left { get; set; }
    public int Width { get; set; }
    public int Height { get; set; }

    int CalcArea()
    {
        return Width * Height;
    }
}
```

Classes may hold data (**properties**)

Classes may hold operations (**methods**)

```
public int Bottom
{
    get
    {
        return Top + Height;
    }
}
```

Calculated  
property

```
public int Right
{
    get
    {
        return Left + Width;
    }
}
```

Calculated  
property

```
public bool IsInside(Rectangle r)
{
    return (r.Left <= Left) && (r.Right >= Right) &&
           (r.Top <= Top) && (r.Bottom >= Bottom);
}
```

Boolean method

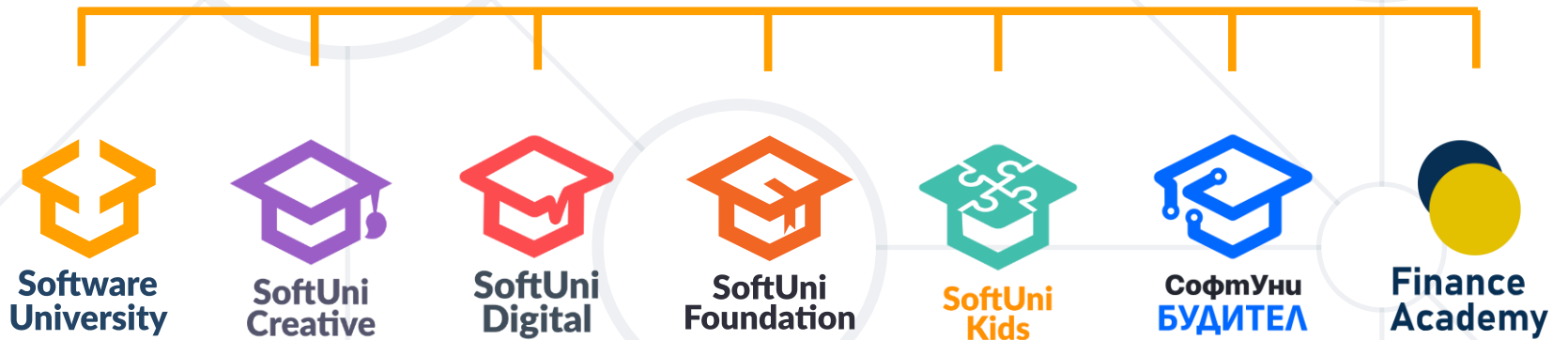
- Objects
  - Holds a set of **named values**
  - **Instance** of a class
- Classes define templates for object
  - **Methods**
  - **Constructors**
  - **Properties**



# Questions?



SoftUni



# SoftUni Diamond Partners



THE CROWN IS YOURS



- Software University – High-Quality Education, Profession and Job for Software Developers

- [softuni.bg](http://softuni.bg), [about.softuni.bg](http://about.softuni.bg)

- Software University Foundation

- [softuni.foundation](http://softuni.foundation)

- Software University @ Facebook

- [facebook.com/SoftwareUniversity](https://facebook.com/SoftwareUniversity)



- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – <https://about.softuni.bg/>
- © Software University – <https://softuni.bg>

