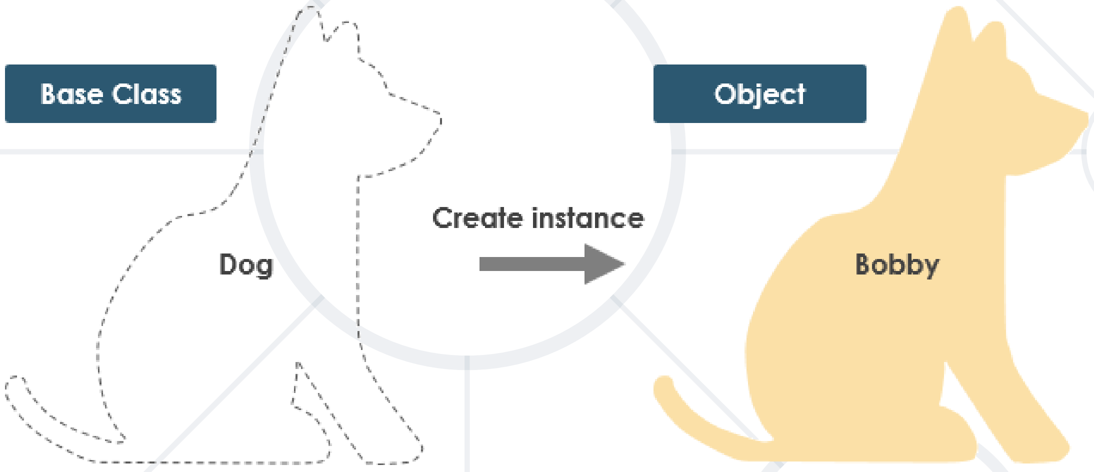
Classes and Objects

Objects, Classes and Class Members



Base Class

Dog

Create instance

Object

Bobby

Properties	Methods
Color	Sit
Eye Color	Lay Down
Height	Shake
Length	Come
Weight	

Property Values	Methods
Color: Yellow	Sit
Eye Color: Brown	Lay Down
Height: 17 in	Shake
Length: 35 in	Come
Weight: 24 pounds	

SoftUni

SoftUni Team

Technical Trainers

Software University

Software University

https://about.softuni.bg/

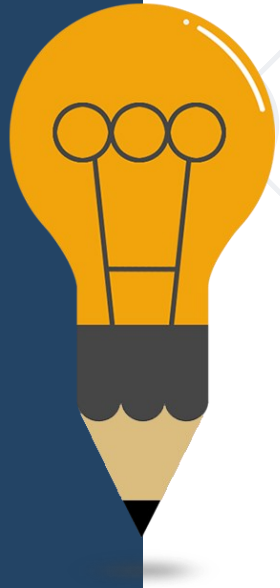
Table of Contents

1. Objects and Classes
2. Defining Simple Classes
3. Fields and Properties
4. Methods
5. Constructors

What is an Object? What is a Class?

Objects

- An **object** holds a set of named values
  - E.g. **birthday** object holds **day**, **month** and **year**
  - Creating a **birthday** object:



Birthday

Day = 22

Month = 6

Year = 1990

Object name

Object properties

Create a **new** object of type **DateTime**

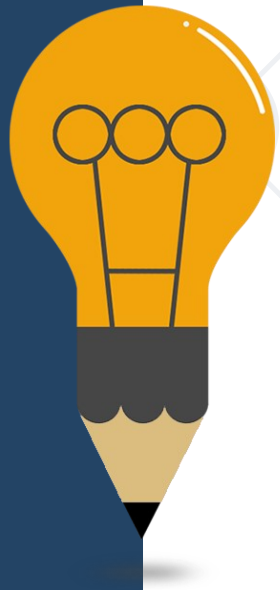
```
var day = new DateTime(
  2019, 2, 25);
Console.WriteLine(day);
```

The **new** operator creates a new object

```
var birthday = new { Day = 22, Month = 6, Year = 1990 };
```

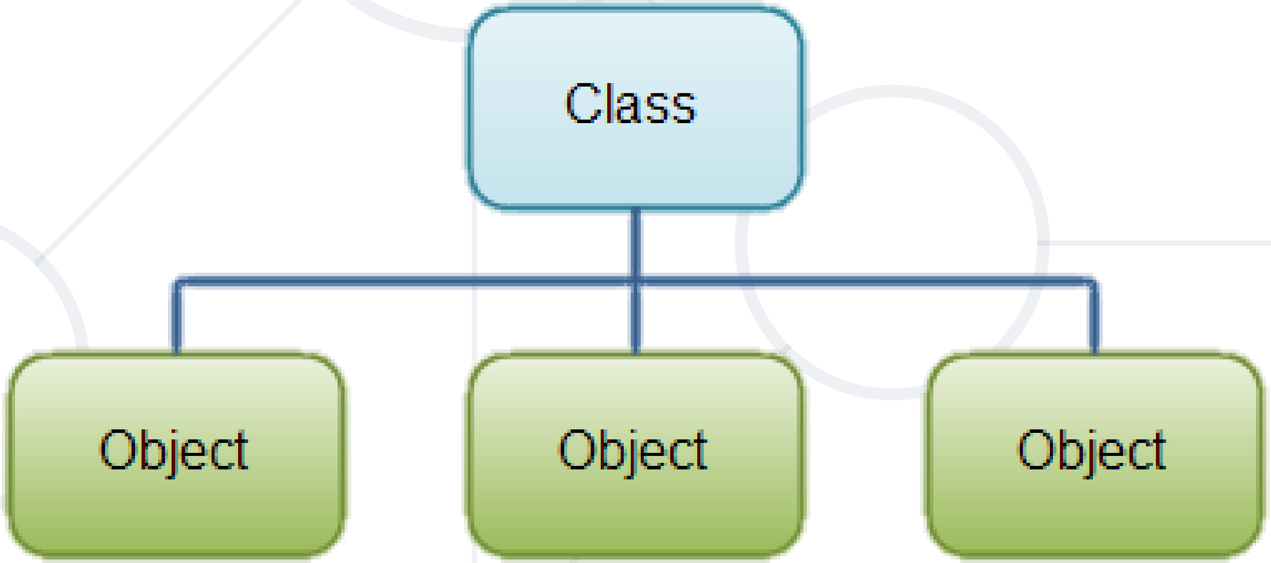
Classes

- In programming, **classes** provide the **structure** for **objects**
  - Act as **template** for **objects** of the same type
- Classes define:
  - **Data** (properties), e.g. `Day`, `Month`, `Year`
  - **Actions** (behavior), e.g. `AddDays(count)`, `Subtract(date)`



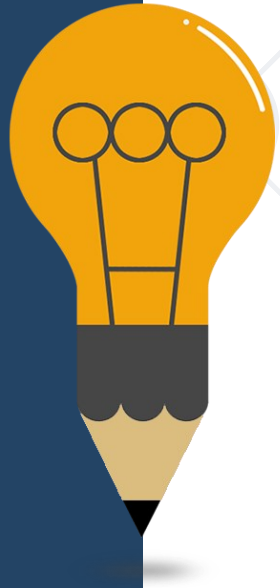
Classes

- One class may have many instances (objects)
  - Sample class: **DateTime**
  - Sample objects: **peterBirthday**, **mariaBirthday**



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**



```
DateTime date1 = new DateTime(2018, 5, 5);
DateTime date2 = new DateTime(2016, 3, 5);
DateTime date3 = new DateTime(2013, 12, 31);
```

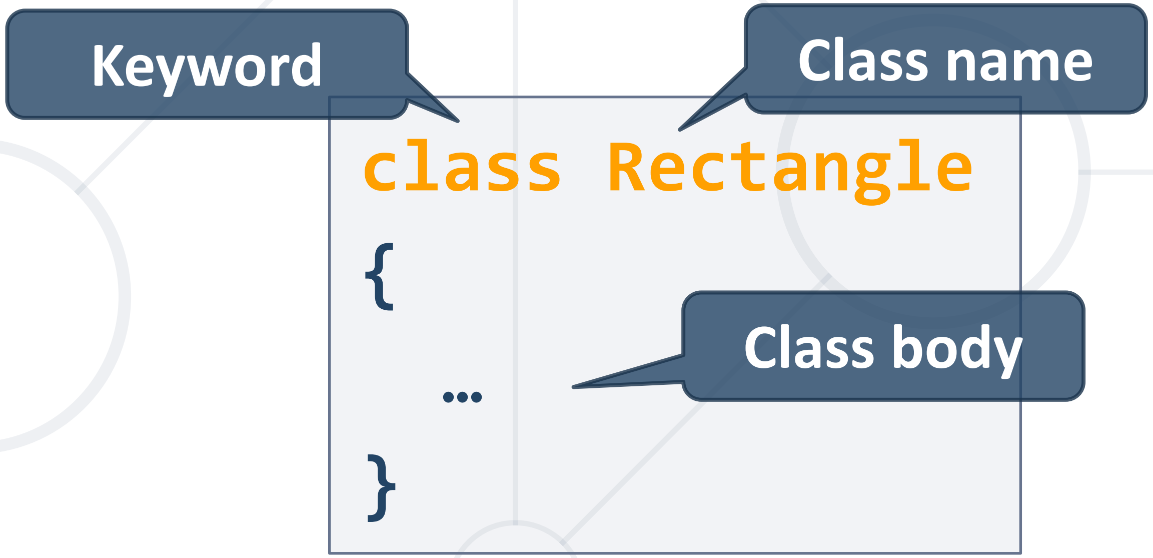
Objects and Classes – Example

```
DateTime peterBirthday = new DateTime(1996, 11, 27);
DateTime mariaBirthday = new DateTime(1995, 6, 14);
Console.WriteLine("Peter's birth date: {0:d-MMM-yyyy}",peterBirthday);
// 27-Nov-1996
Console.WriteLine("Maria's birth date: {0:d-MMM-yyyy}",mariaBirthday);
// 14-Jun-1995
var mariaAfter18Months = mariaBirthday.AddMonths(18);
Console.WriteLine("Maria after 18 months: {0:d-MMM-yyyy}", mariaAfter18Months);
// 14-Dec-1996
TimeSpan ageDiff = peterBirthday.Subtract(mariaBirthday);
Console.WriteLine("Maria older than Peter by: {0} days", ageDiff.Days);
// 532 days
```

Defining Simple Classes

Defining Simple Classes

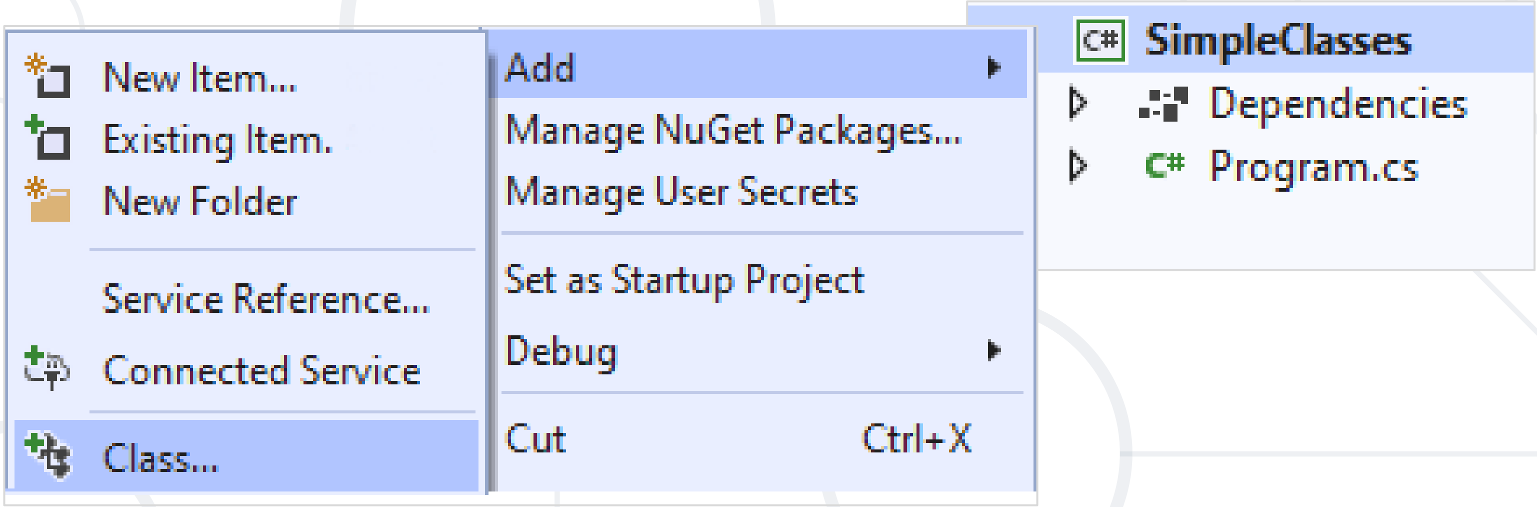
- Class is a **concrete implementation** of an ADT (abstract data type)
- Classes provide **structure** for **describing** and **creating** objects

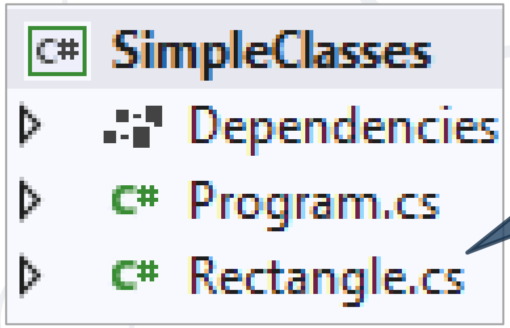


```
class Rectangle
{
    …
}
```

Defining Simple Classes Rectangle

- Create a file for this class: [**Project**] → [**Add Class**] or: right click on the project [**Add**]→ [**New Item**] → [**Class**]





Class stays in a separate file

Naming Classes

- Name classes with nouns using **PascalCasing**
- Use **descriptive nouns**
- **Avoid abbreviations** (except widely known, e.g. URL, HTTP, etc.)

```
class Dice { … }
class BankAccount { … }
```

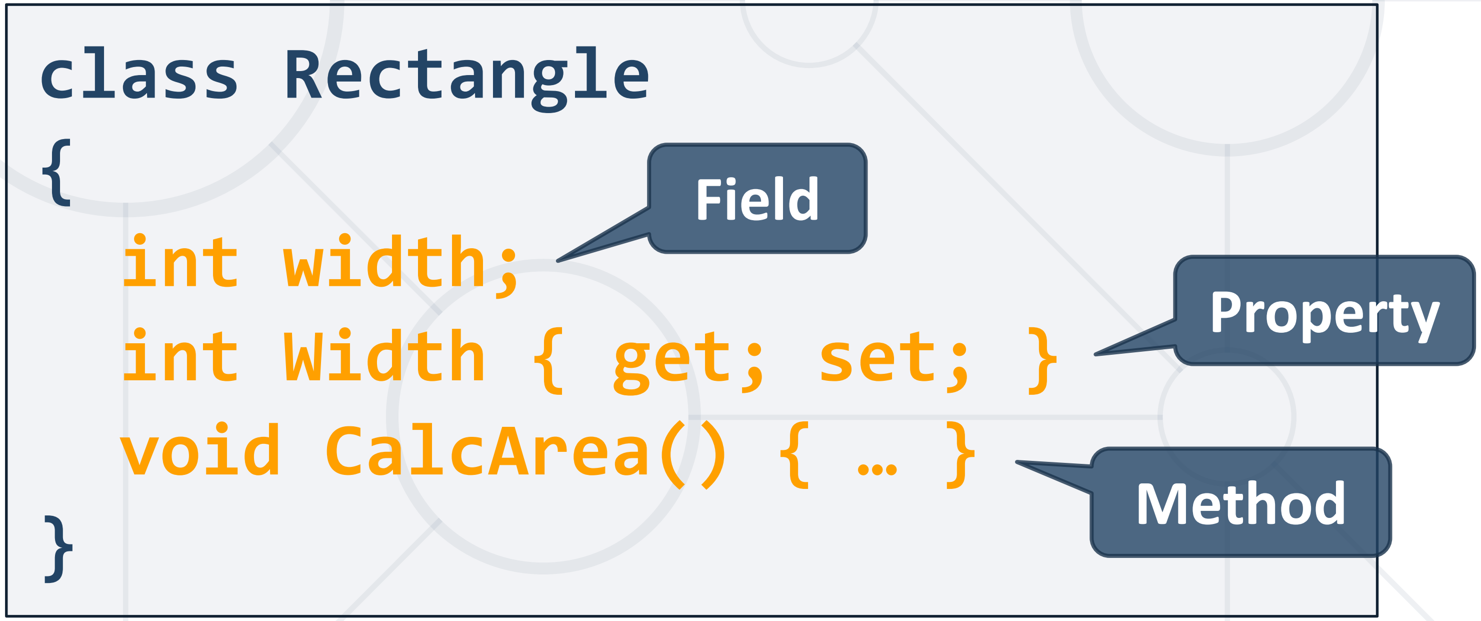
✓

```
class TPMF { … }
class bankaccount { … }
class intcalc { … }
```

✗

Class Members

- **Members** are **declared** inside the class
- Members can be:
  - **Fields** (data)
  - **Properties** (data + logic)
  - **Methods** (actions)
  - **Constructors**
  - Others



```
class Rectangle
{
   int width;
   int Width { get; set; }
   void CalcArea() { … }
}
```

Class Rectangle – Example

- Class **Rectangle** holds properties **Width** and **Height**

Rectangle.cs

```
class Rectangle
{
  public int Width { get; set; }
  public int Height { get; set; }
  public string Color { get; set; }
}
```

Creating an Object

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

```
class Program
{
  public static void Main()
  {
    Rectangle firstRect = new Rectangle();
    Rectangle secondRect = new Rectangle();
  }
}
```

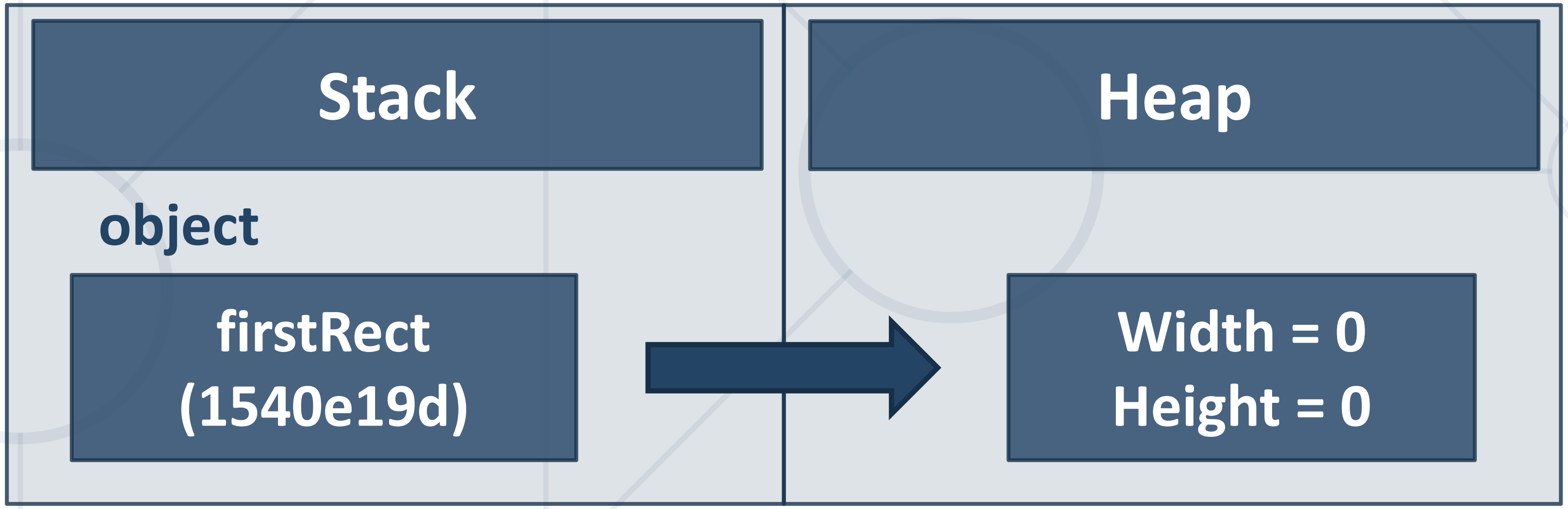
Use the **new** keyword to create an object

A variable stores an object **reference**

Object Reference

- Declaring a variable creates a **reference** in the stack
- The **new** keyword allocates memory on the heap

```
Rectangle firstRect = new Rectangle();
```



Defining a Simple Method in a Class

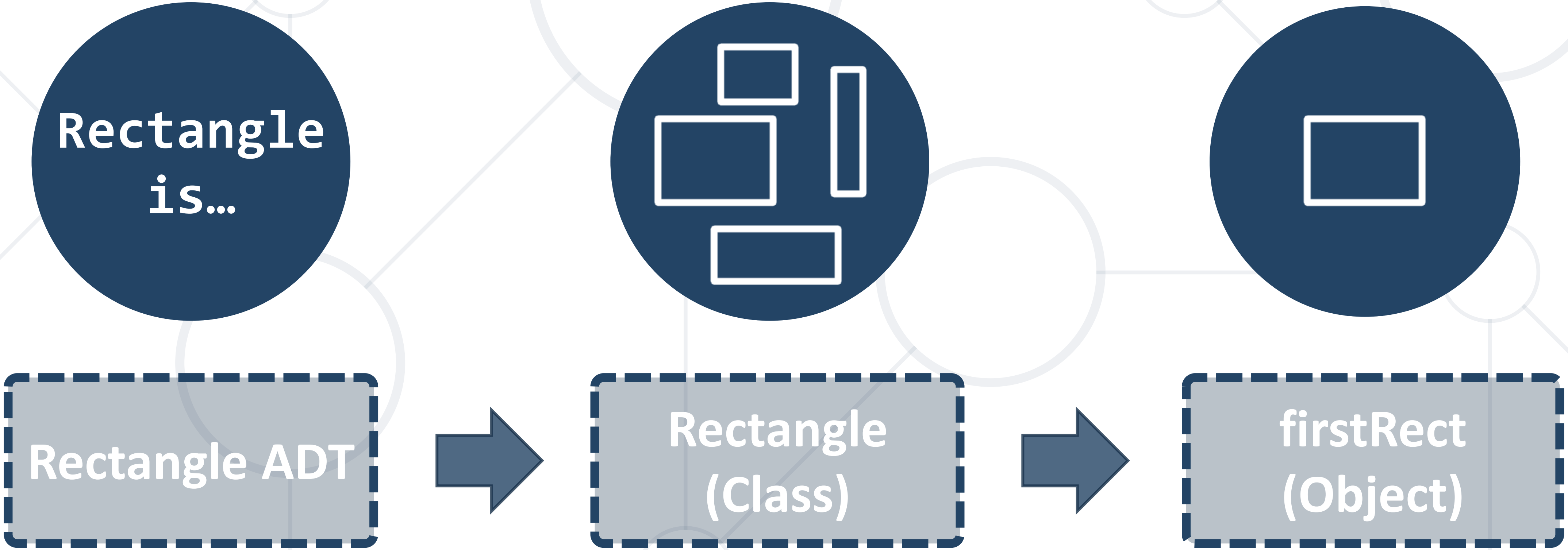
```
class Rectangle
{
  public int Width { get; set; }
  public int Height { get; set; }
  public string Color { get; set; }

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

Methods define actions in the classes

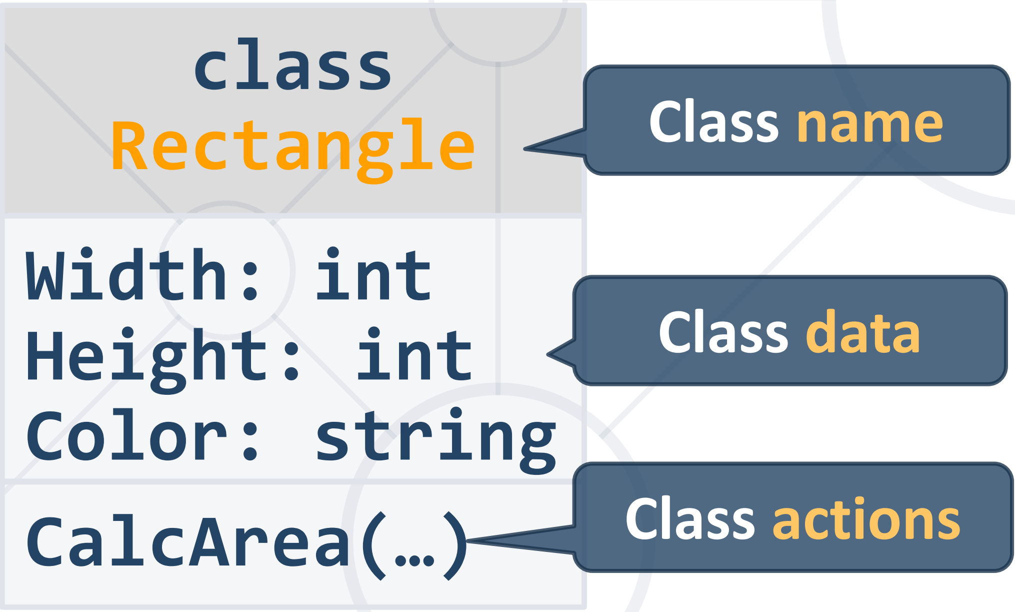
Class –> Object

- Classes provide **structure** for describing and creating objects
- An **object** is a **single instance of a class**



Classes vs. Objects

- Classes provide **structure** for creating **objects**



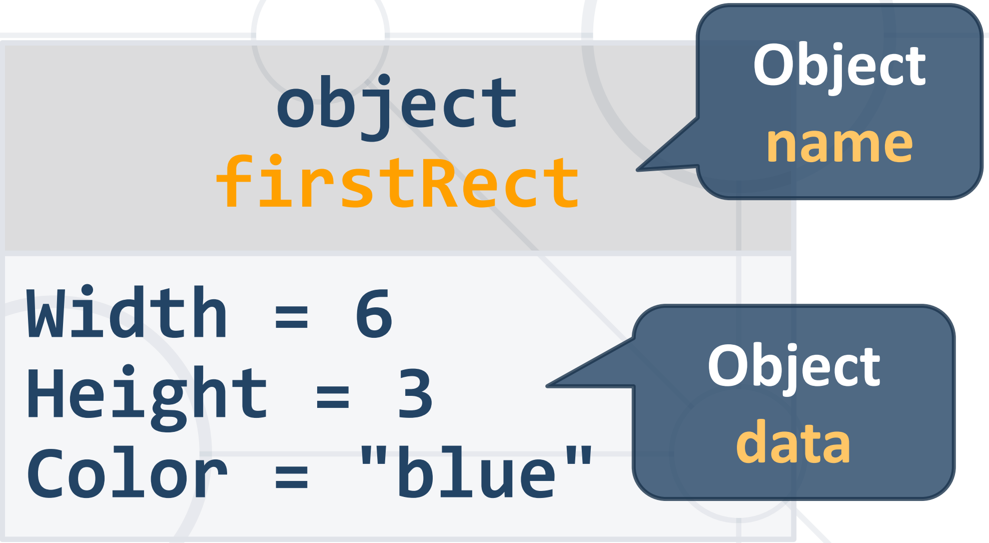
```
class
Rectangle
Width: int
Height: int
Color: string
CalcArea(…)
```

Class name

Class data

Class actions

- An **object** is a single **instance** of a class



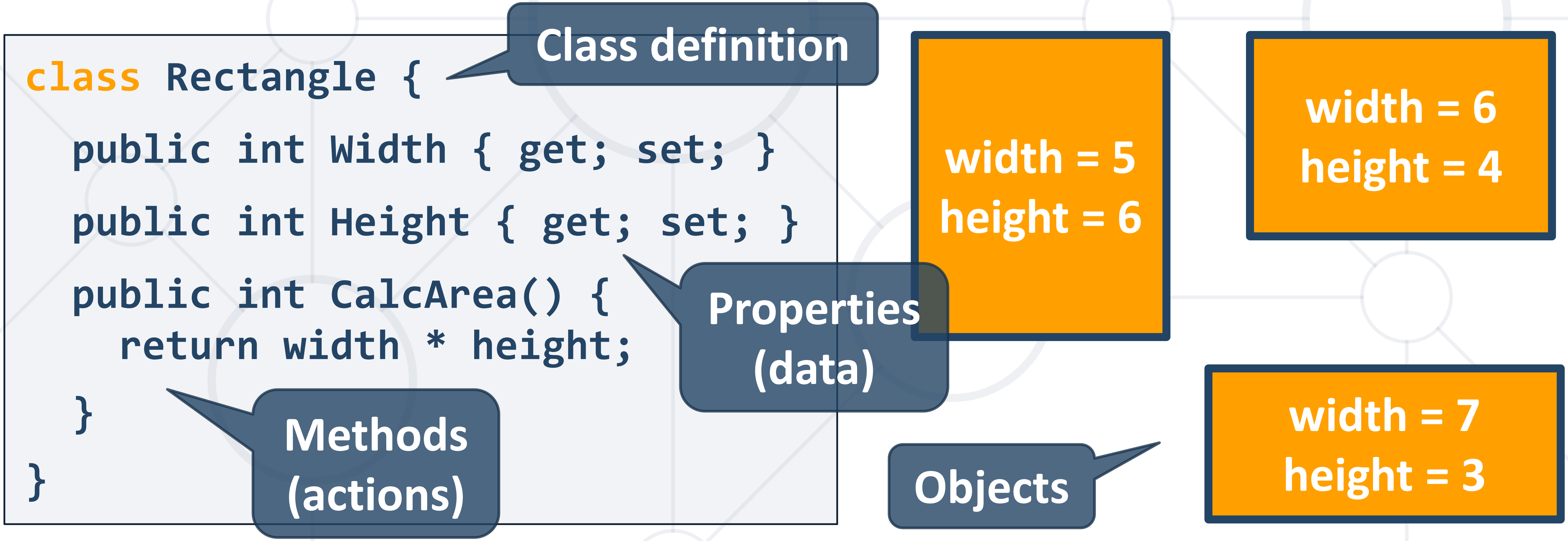
```
object
firstRect
Width = 6
Height = 3
Color = "blue"
```

Object name

Object data

Object-Oriented Programming (OOP)

- **Object-Oriented Programming (OOP)** is the concept of using **classes** and **objects** (class instances) to model the real world



```
class Rectangle {
  public int Width { get; set; }
  public int Height { get; set; }
  public int CalcArea() {
    return width * height;
  }
}
```

Class definition

Properties (data)

Methods (actions)

width = 5
height = 6

width = 6
height = 4

width = 7
height = 3

Objects

Storing Data Inside a Class

Fields and Modifiers

- Class fields have **type** and **name**
- Modifiers define accessibility

Class modifier

Fields should always be private

Fields can be of **any type**

```
public class Rectangle
{
  private string color;
  private int width;
  private int height;
  private int[] sections;
  private Shape type;
  public int CalcArea() { … }
}
```

Properties

- Used to create **accessors** and **mutators** (**getters** and **setters**)

```
public class Rectangle
{
  private int width;
  public int Width
  {
    public get { return this.width; }
    public set { this.width = value; }
  }
}
```

The field is hidden

The getter provides access to the field

The setter provides field change

Problem: Car

- Create a class **Car**

```
Car
-make:string
-model:string
-year:int
(no actions)
```

```
private string make;
private string model;
private int year;
public string Make
{
  get { return this.make; }
  set { this.make = value; }
}
// TODO: Balance and Year Getter & Setter
```

Check your solution here: https://judge.softuni.bg/Contests/Practice/Index/3161#0

Short Properties in C#

```
public string Brand { get; private set; }

public string Make { get; set; }

public string BrandAndMake
{
  get => Brand + " " + Make;
}
```

Defining a Class Behaviour

Methods

- Store **executable code** (an algorithm)

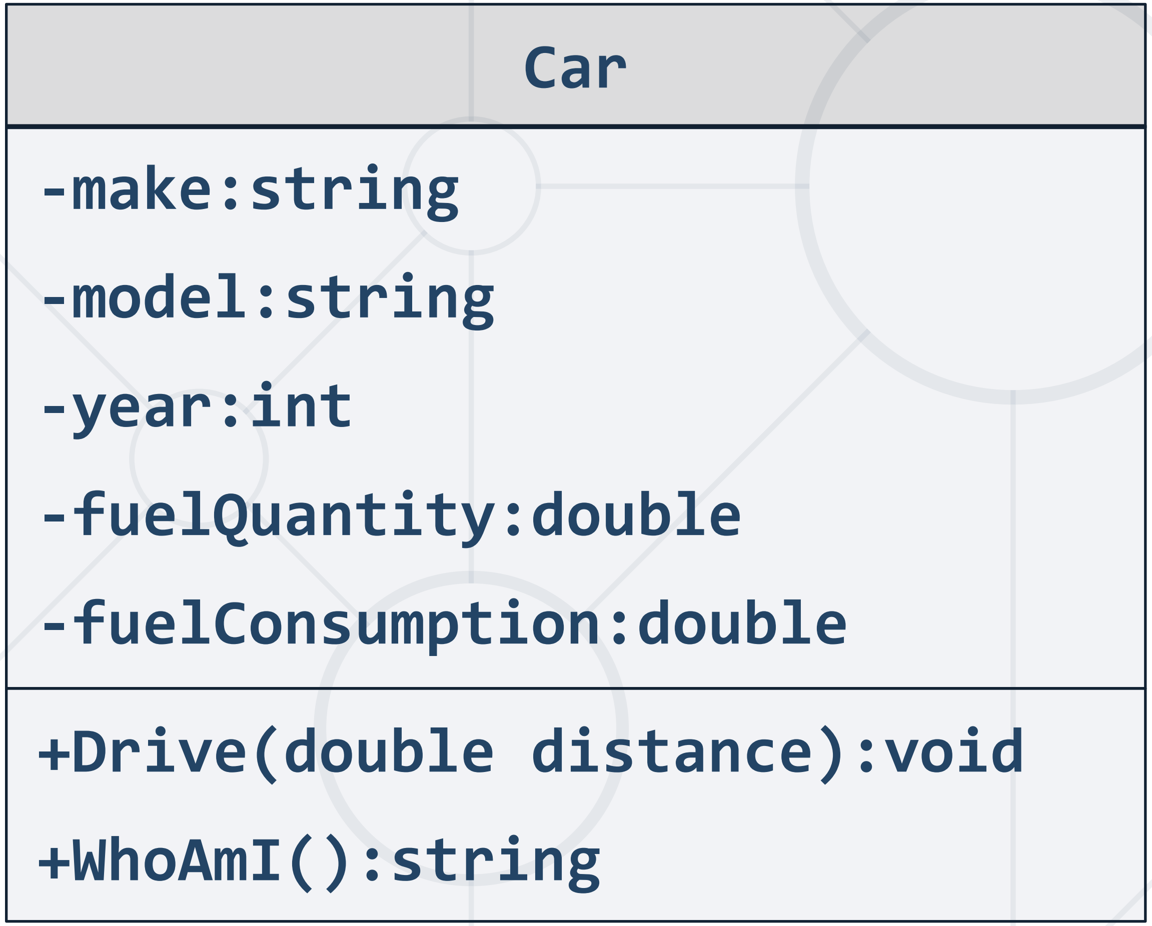
```
public class Rectangle
{
  public int Width { get; set; }
  public int Height { get; set; }

  public int CalcArea()
  {
     int area = this.Width * this.Height;
     return area;
  }
}
```

this points to the current instance

Problem: Car Extension

- Create a class **Car**



Car

-make:string

-model:string

-year:int

-fuelQuantity:double

-fuelConsumption:double

+Drive(double distance):void

+WhoAmI():string

Solution: Car Extension (1)

```
// TODO: Get the other fields from previous problem
private double fuelQuantity;
private double fuelConsumption;
// TODO: Get the other properties from previous problem
public double FuelQuantity {
  get { return this.fuelQuantity; }
  set { this.fuelQuantity = value; }}
public double FuelConsumption {
  get { return this.fuelConsumption; }
  set { this.fuelConsumption = value; }}
```

Solution: Car Extension (2)

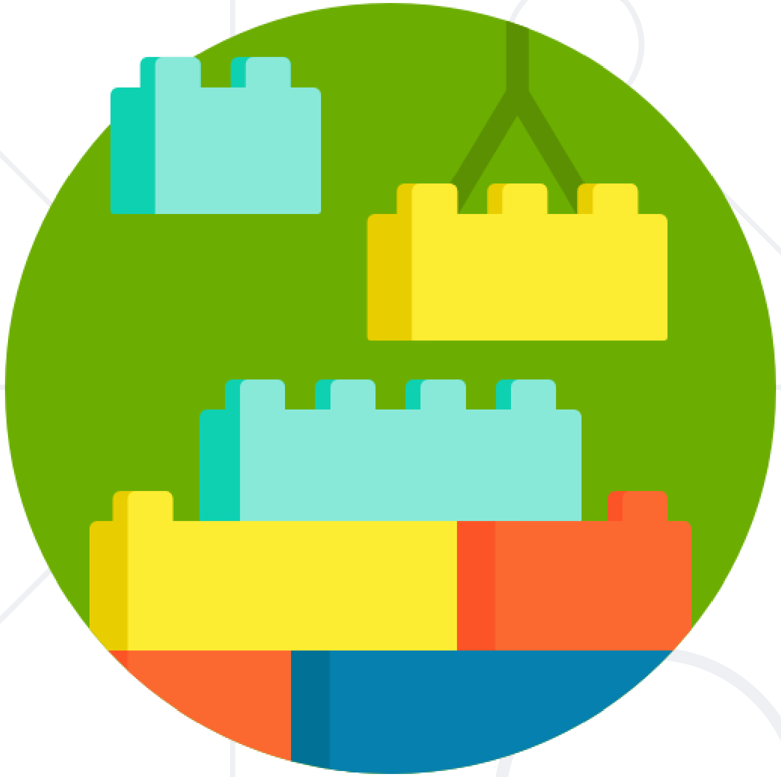
```
public void Drive(double distance)
{
  bool canContinue = this.FuelQuantity –
    (distance * this.FuelConsumption) >= 0;

  if (canContinue)
  {
    this.FuelQuantity -= distance * this.FuelConsumption;
  }
  else
  {
    Console.WriteLine("Not enough fuel to perform this trip!");
  }
}
```

Solution: Car Extension (3)

```
public string WhoAmI()
{
  StringBuilder sb = new StringBuilder();
  sb.AppendLine($"Make: {this.Make}");
  sb.AppendLine($"Model: {this.Model}");
  sb.AppendLine($"Year: {this.Year}");
  sb.Append($"Fuel: {this.FuelQuantity:F2}L");
  return sb.ToString();
}
```

Check your solution here: https://judge.softuni.bg/Contests/Practice/Index/3161#1



Object Initialization

Constructors

- When a constructor is invoked, it creates an instance of its class and usually initializes its members
- Classes in C# are instantiated with the **keyword new**

```
public class Rectangle
{
  public Rectangle() { }
}
```

```
public class StartUp
{
  static void Main()
  {
    var figure = new Rectangle();
  }
}
```

Object Initial State (1)

- Constructors **set object's initial state**

```
public class Rectangle {
  int width;
  int height;
  string color;

  public Rectangle(int width, int height, string color)
  {
    this.width = width;
    this.height = height;
    this.color= color;
  }
}
```

Object Initial State (2)

```
public class Rectangle {
  int width;
  int height;
  private int[] sections;

  public Rectangle(int width, int height, string color)
  {
    this.width = width;
    this.height = height;
    this.sections= new int[(width * height)/2];
  }
}
```

Always ensure **correct state**

Multiple Constructors

- You can have multiple constructors in the same class

```
public class Rectangle {
  private string color;

  public Rectangle()
  {
    this.color = "white";
  }

  public Rectangle(string color)
  {
    this.color = color;
  }
}
```

Constructor **without** parameters

Constructor **with** parameters

Constructor Chaining

- Constructors can call each other

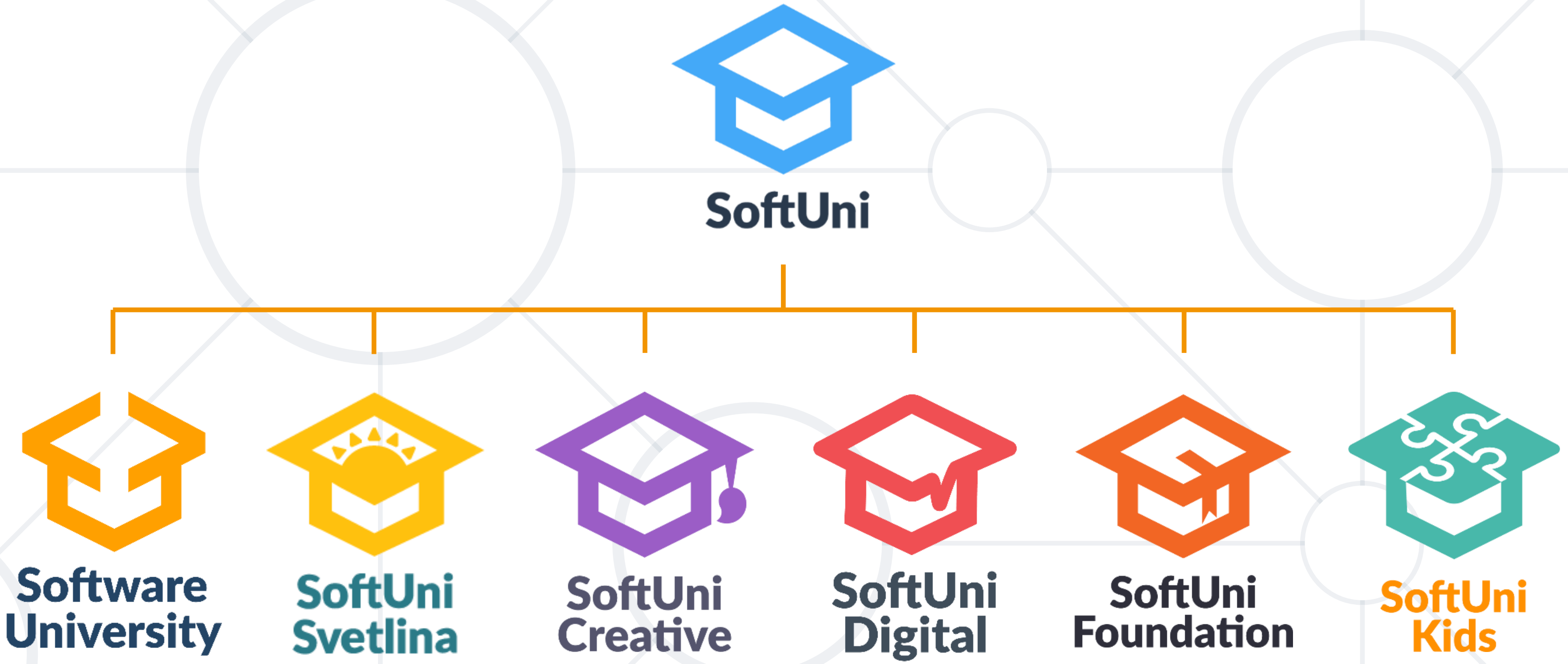
```
public class Person {
  private string name;
  private int age;
  public Person()
  {
    this.age = 18;
  }
  public Person(string name) : this()
  {
    this.name = name;
  }
}
```

Calls default constructor

Summary

- Classes define **structure** for objects
- Objects are **instances of a class**
- NET Core provides **thousands of ready-to-use classes**
- **Classes** provide structure for **describing** and **creating** objects
- Classes define **fields**, **methods**, **properties**, **constructors** and other members
- Constructors:
  - **Invoked** when creating **new instances**
  - **Initialize** the **object's state**

Questions?



SoftUni

Software University

SoftUni Svetlina

SoftUni Creative

SoftUni Digital

SoftUni Foundation

SoftUni Kids

License