Telerik Academy

Object-Oriented Programming Fundamental Principles – Part 1

Inheritance, Abstraction, Encapsulation

C# OOP

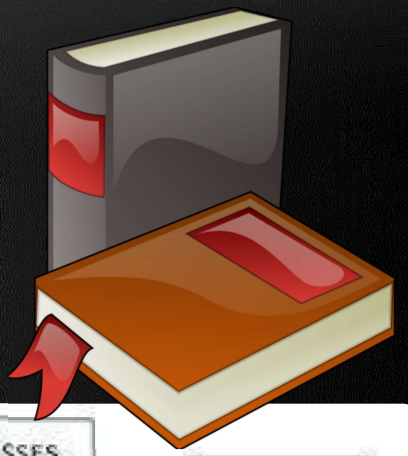
Telerik Software Academy

https://telerikacademy.com

Telerik Academy

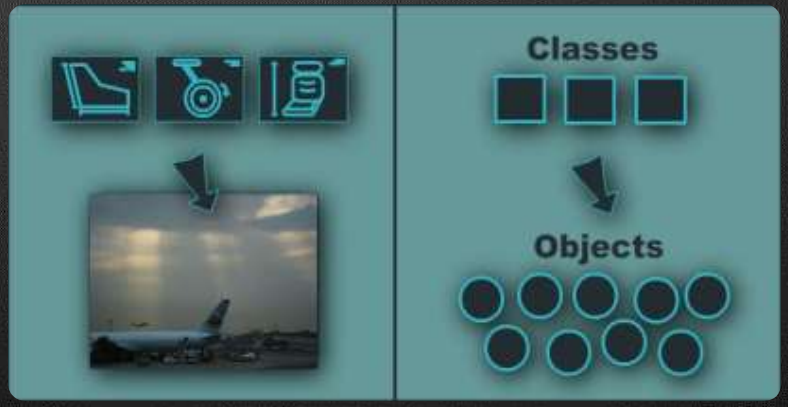
Table of Contents

- Fundamental Principles of OOP
- Inheritance
  - Class Hierarchies
  - Inheritance and Access Levels
- Abstraction
  - Interfaces
  - Abstract Classes
- Encapsulation



Telerik Academy

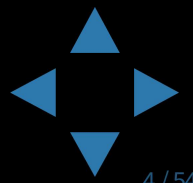
Fundamental Principles of OOP



Telerik Academy

Fundamental Principles of OOP

- Inheritance
  - Inherit members from parent class
- Abstraction
  - Define and execute abstract actions
- Encapsulation
  - Hide the internals of a class
- Polymorphism
  - Access a class through its parent interface



Telerik Academy

Inheritance

Telerik Academy

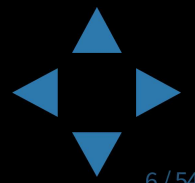
Classes and Interfaces

- Classes define attributes and behavior
  - Fields, properties, methods, etc.
  - Methods contain code for execution

```
public class Labyrinth { public int Size { get; set; } }
```

- Interfaces define a set of operations
  - Empty methods and properties, left to be implemented later

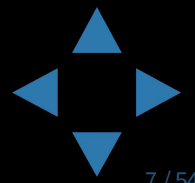
```
public interface IFigure { void Draw(); }
```

Telerik Academy

Inheritance

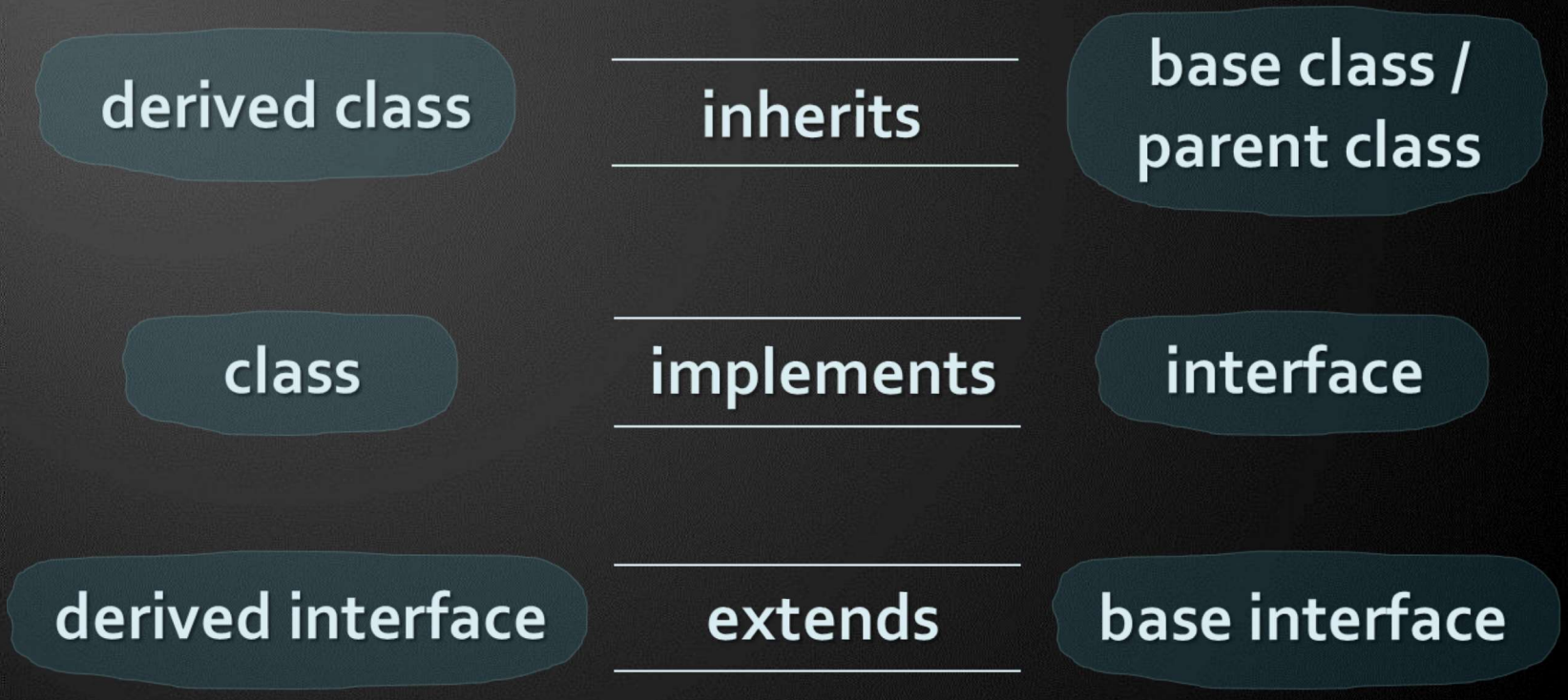
- Inheritance allows child classes to inherit the characteristics of an existing parent (base) class
  - Attributes(fields and properties)
  - Operations(methods)
- Child class can extend the parent class
  - Add new fields and methods
  - Redefine methods(modify existing behavior)
- A class can implement an interface by providing implementation for all its methods



Telerik Academy

Types of Inheritance

- Inheritance terminology



Telerik Academy

Inheritance – Benefits

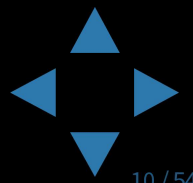
- Inheritance has a lot of benefits
  - Extensibility
  - Reusability (code reuse)
  - Provides abstraction
  - Eliminates redundant code

- Use inheritance for buidling is-a relationships
  - E.g.dog is-a animal(dogs are kind of animals)
- Don't use it to build has-a relationship
  - E.g.dog has-a name(dog is not kind of name)

Telerik Academy

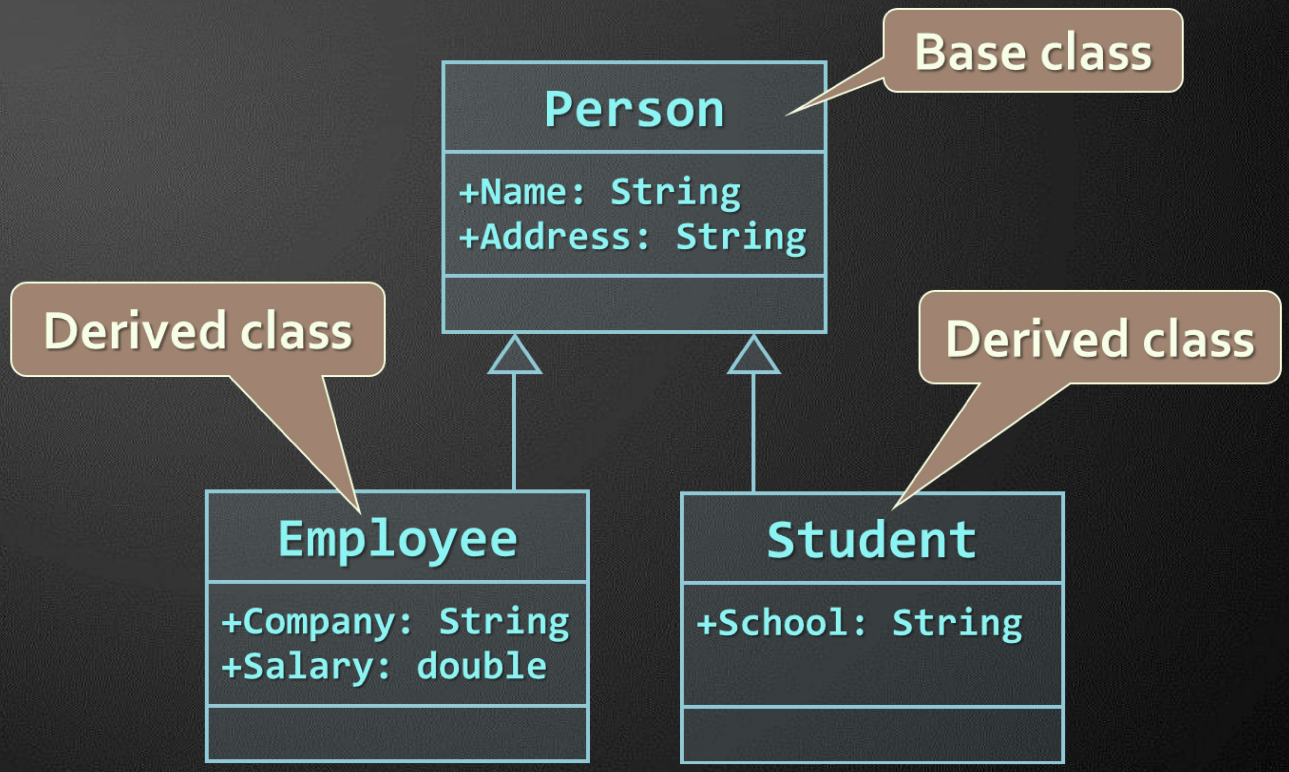
Inheritance

- Inheritance implicitly gains all members from another class
  - All fields, methods, properties, events, …
  - Some members could be inaccessible (hidden)
- The class whose methods are inherited is called base (parent) class
- The class that gains new functionality is called derived (child) class



Telerik Academy

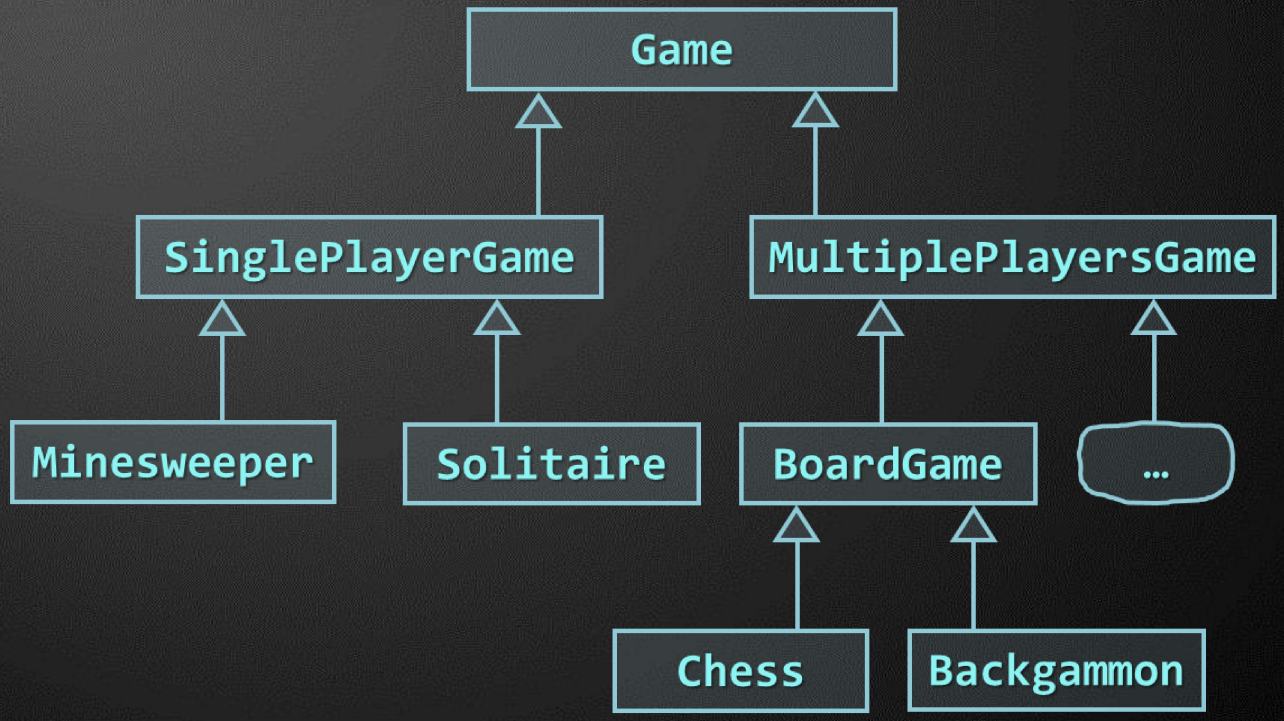
Inheritance – *Example*



Telerik Academy

Class Hierarchies

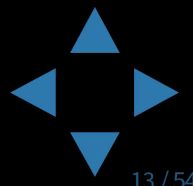
- Inheritance leads to a hierarchies of classes and / or interfaces in an application:



Telerik Academy

Inheritance in .NET

- A class can inherit only one base class
  - E.g. `IOException` derives from `SystemException` and it derives from `Exception`
- A class can implement several interfaces
  - This is .NET's form of multiple inheritance
  - E.g. `List<T>` implements `IList<T>`, `ICollection<T>`, `IEnumerable<T>`
- An interface can implement several interfaces
  - E.g. `IList<T>` implements `ICollection<T>` and `IEnumerable<T>`



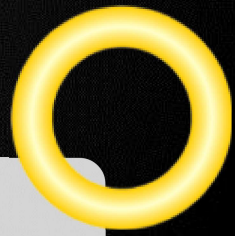
Telerik Academy

How to Define Inheritance?

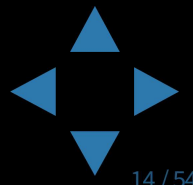
- Specify the name of the base class after the name of the derived(with colon)

```
public class Shape
{ … }
public class Circle : Shape
{ … }
```

- Use the keyword `base` to invoke the parent constructor



```
public Circle(int x, int y) : base(x)
{ … }
```

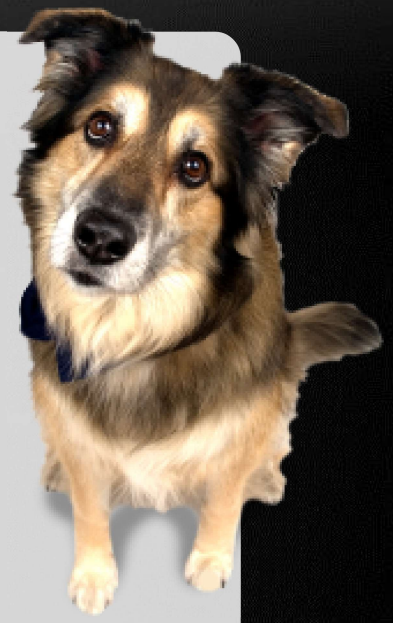


Telerik Academy Simple Inheritance *Example*

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

    public Mammal(int age)
    {
        this.Age = age;
    }

    public void Sleep()
    {
        Console.WriteLine("Shhh! I'm sleeping!");
    }
}
```

Telerik Academy Simple Inheritance *Example*

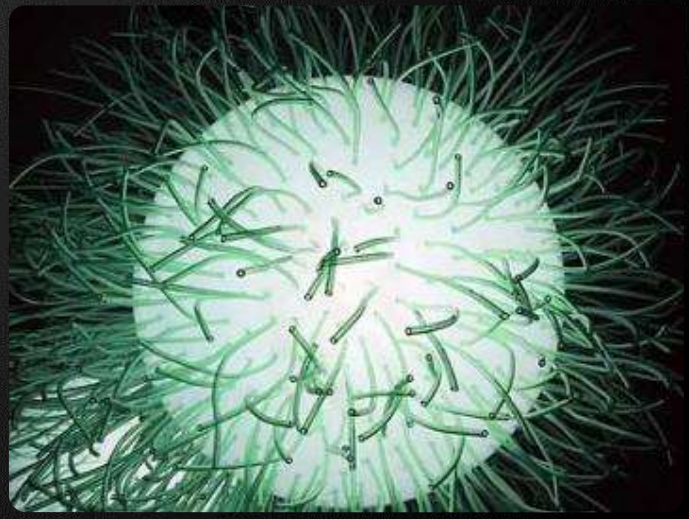
```
public class Dog : Mammal
{
    public string Breed { get; set; }

    public Dog(int age, string breed)
      : base(age)
    {
        this.Breed = breed;
    }

    public void WagTail()
    {
        Console.WriteLine("Tail wagging...");
    }
}
```



Telerik Academy

Simple Inheritance

Demo



Telerik Academy

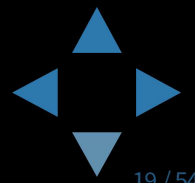
Access Levels

- Access modifiers in C#
  - `public` – access is not restricted
  - `private` – access is restricted to the containing type
  - `protected` – access is limited to the containing type and types derived from it
  - `internal` – access is limited to the current assembly
  - `protected internal` – access is limited to the current assembly or types derived from the containing class

Telerik Academy

Inheritance and Accessibility

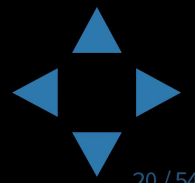
```
class Creature
{
    protected string Name { get; private set; }
    protected void Walk()
    {
        Console.WriteLine("Walking ...");
    }
    private void Talk()
    {
        Console.WriteLine("I am creature ...");
    }

}
class Mammal : Creature
{
    // base.Walk() can be invoked here
    // base.Talk() cannot be invoked here
    // this.Name can be read but cannot be modified here
}
```

Telerik Academy

Inheritance and Accessibility

```
class Dog : Mammal
{
    public string Breed { get; private set; }
    // base.Talk() cannot be invoked here (it is private)
}

class InheritanceAndAccessibility
{
    static void Main()
    {
        Dog joe = new Dog(6, "Labrador");
        Console.WriteLine(joe.Breed);
        // joe.Walk() is protected and can not be invoked
        // joe.Talk() is private and can not be invoked
        // joe.Name = "Rex"; // Name cannot be accessed here
        // joe.Breed = "Shih Tzu"; // Can't modify Breed
    }
}
```



Telerik Academy

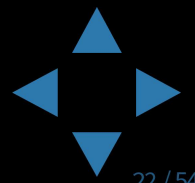
Inheritance and Accessibility

Demo

Telerik Academy

Inheritance: Important Aspects

- Structures cannot be inherited
- In C# there is no multiple inheritance
  - Only multiple interfaces can be implemented
- Static members are also inherited
- Constructors are not inherited
- Inheritance is transitive relation
  - If C is derived from B, and B is derived from A, then C inherits A as well



Telerik Academy

Inheritance: Important Features

- When a derived class extends its base class
  - It can freely add new members
  - Cannot remove derived ones
- Declaring new members with the same name or signature hides the inherited ones
- A class can declare `virtual` methods and properties
  - Derived classes can `override` the implementation of these members
  - E.g. `Object.ToString()` is virtual method

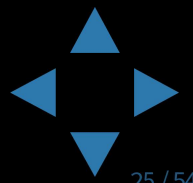
Telerik Academy

Abstraction

Telerik Academy

Abstraction

- Abstraction means ignoring irrelevant features, properties, or functions and emphasizing the relevant ones…
- … relevant to the given project
  - With an eye to future reuse in similar projects
- Abstraction helps managing complexity



Telerik Academy

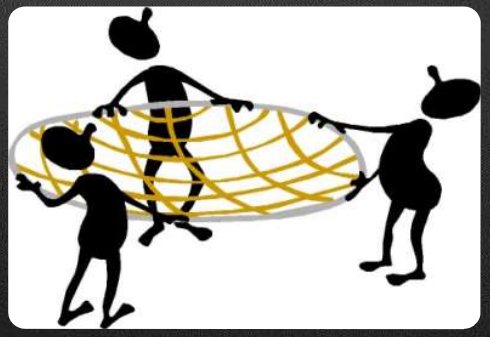
Abstraction (2)

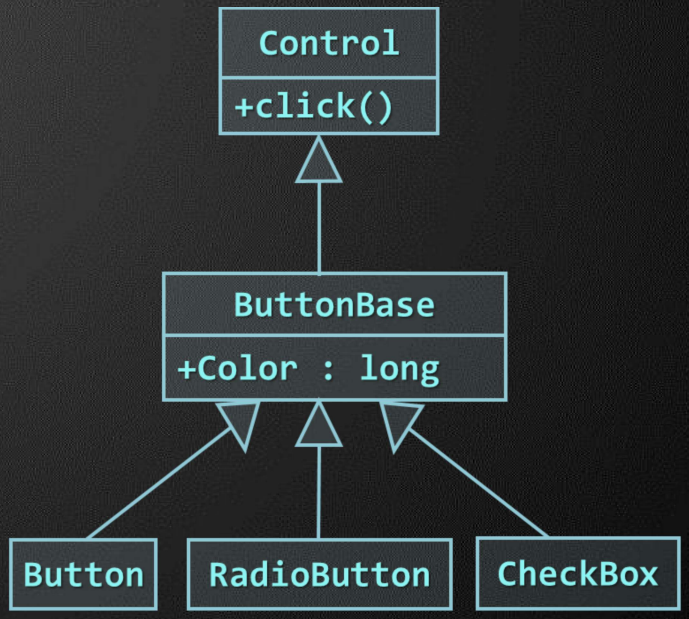
- Abstraction is something we do every day
  - Looking at an object, we see those things about it that have meaning to us
  - We abstract the properties of the object, and keep only what we need
  - E.g.students get "name" but not "color of eyes"
- Allows us to represent a complex reality in terms of a simplified model
- Abstraction highlights the properties of an entity that we need and hides the others

Telerik Academy

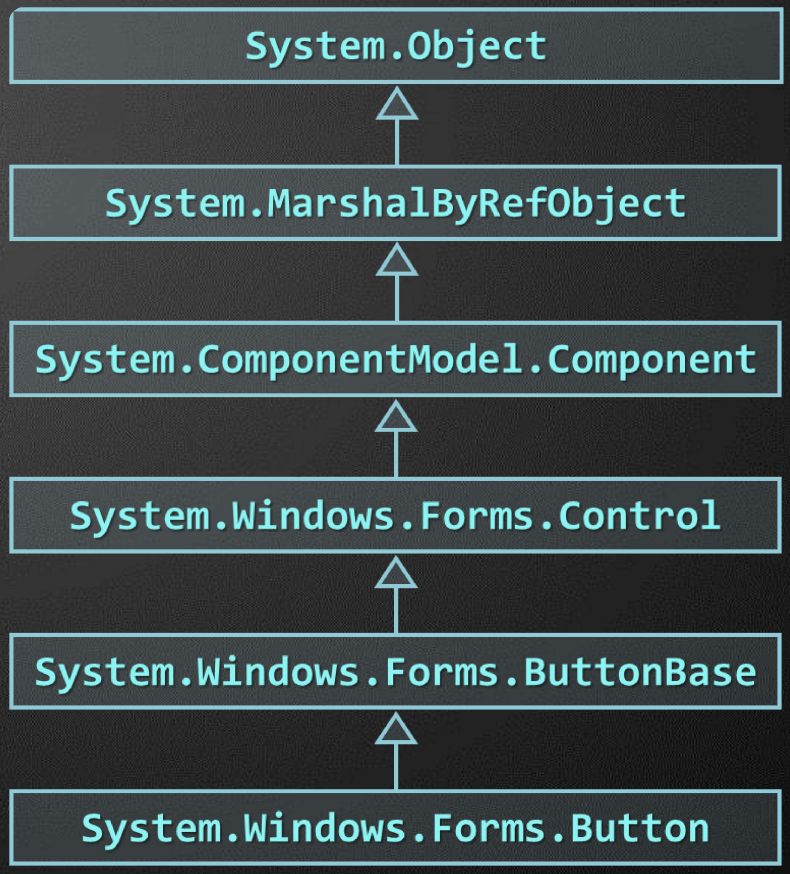
Abstraction in .NET

- In .NET object-oriented programming abstraction is achieved in several ways:
  - Abstract classes
  - Interfaces
  - Inheritance





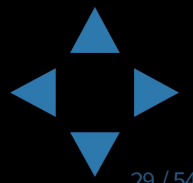
Abstraction in .NET – *Example*



Telerik Academy

Interfaces

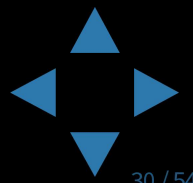
- An interface defines a set of operations(methods) that given object should perform
  - Also called "contract" for providing a set of operations
  - Defines abstract behavior
- Interfaces provide abstractions
  - You invoke the abstract actions
  - Without worrying how it is internally implemented



Telerik Academy

Interfaces (2)

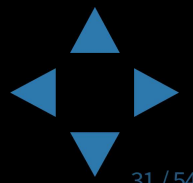
- Interfaces describe a prototype of group of methods(operations), properties and events
  - Can be implemented by a given class or structure
  - Define only the prototypes of the operations
    - No concrete implementation is provided
  - Can be used to define abstract data types
  - Can be inherited (extended) by other interfaces
  - Can not be instantiated



Telerik Academy

Interfaces – *Example*

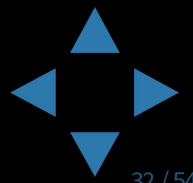
```
public interface IShape
{
    void SetPosition(int x, int y);
    int CalculateSurface();
}
public interface IMovable
{
    void Move(int deltaX, int deltaY);
}
public interface IResizable
{
    void Resize(int weight);
    void Resize(int weightX, int weightY);
    void ResizeByX(int weightX);
    void ResizeByY(int weightY);
}
```



Telerik Academy

Interfaces – *Example*

```
public interface IPerson
{
    DateTime DateOfBirth  // Property DateOfBirth
    {
        get;
        set;
    }

    int Age  // Property Age (read-only)
    {
        get;
    }

    void Print(); // Method for printing
}
```

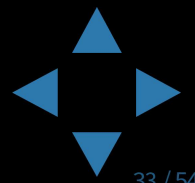
Telerik Academy

Interface Implementation

- Classes and structures can implement (support) one or several interfaces

```
class Rectangle : IShape
{
    public void SetPosition(int x, int y) { … }
    public int CalculateSurface() { … }
}
```

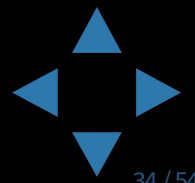
- Implementer classes must implement all interface methods
  - Or should be declared `abstract`



Telerik Academy

Interface Implementation

```
class Rectangle : IShape, IMovable
{
    private int x, y, width, height;
    public void SetPosition(int x, int y) // IShape
    {
        this.x = x;
        this.y = y;
    }
    public int CalculateSurface() // IShape
    {
        return this.width * this.height;
    }
    public void Move(int deltaX, int deltaY) // IMovable
    {
        this.x += deltaX;
        this.y += deltaY;
    }
}
```

Telerik Academy

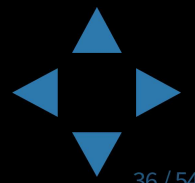
Interfaces and Implementation

Demo

Telerik Academy

Abstract Classes

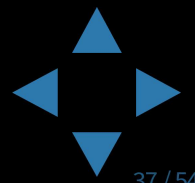
- Abstract classes are special classes defined with the keyword `abstract`
  - Mix between class and interface
  - Partially implemented or fully unimplemented
  - Not implemented methods are declared `abstract` and are left empty
  - Cannot be instantiated directly
- Child classes should implement all abstract methods or be declared as `abstract` too



Telerik Academy

Abstract Classes

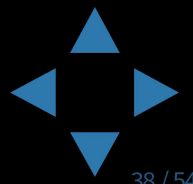
- Abstract methods are empty methods without implementation
  - The implementation is intentionally left for the descendent classes
- When a class contains at least one abstract method, it is called abstract class
- Abstract classes model abstract concepts
  - E.g. person, object, item, movable object



Telerik Academy

Abstract Class – *Example*

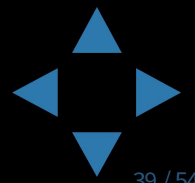
```
abstract class MovableShape : IShape, IMovable
{
    private int x, y;
    public void Move(int deltaX, int deltaY)
    {
        this.x += deltaX;
        this.y += deltaY;
    }
    public void SetPosition(int x, int y)
    {
        this.x = x;
        this.y = y;
    }
    public abstract int CalculateSurface();
}
```

Telerik Academy

Interfaces vs. Abstract Classes

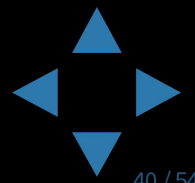
- C# interfaces are like abstract classes, but in contrast interfaces:
  - Can not contain methods with implementation
    - All interface methods are abstract
  - Members do not have scope modifiers
    - Their scope is assumed public
    - But this is not specified explicitly
  - Can not define fields, constants, inner types and constructors



Telerik Academy

Abstract Classes – *Example*

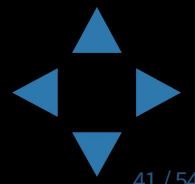
```
public abstract class Animal : IComparable<Animal>
{
    // Abstract methods
    public abstract string GetName();
    public abstract int Speed { get; }
    // Non-abstract method
    public override string ToString()
    {
        return "I am " + this.GetName();
    }
    // Interface method
    public int CompareTo(Animal other)
    {
        return this.Speed.CompareTo(other.Speed);
    }
}
```

Telerik Academy

Abstract Classes – *Example*

```
public class Turtle : Animal
{
    public override int Speed { get { return 1; } }

    public override string GetName()
    { return "turtle"; }
}

public class Cheetah : Animal
{
    public override int Speed { get { return 100; } }

    public override string GetName()
    { return "cheetah"; }
}
```

Telerik Academy

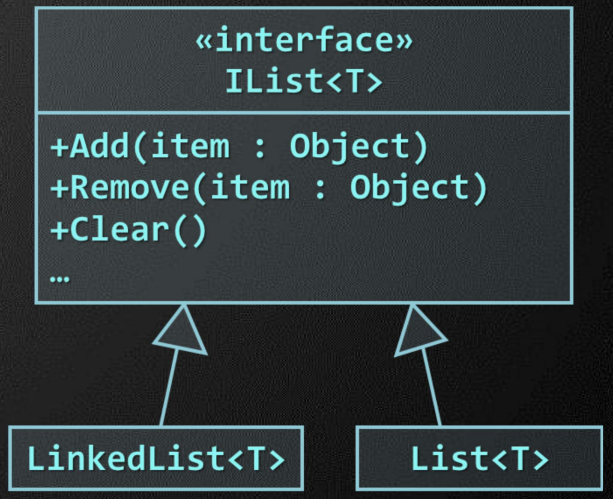
Abstract Classes

Demo

Telerik Academy

Abstract Data Types

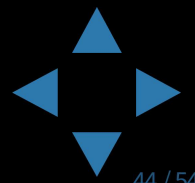
- Abstract Data Types(ADT) are data types defined by a set of operations (interface)
- *Example*: `IList<T>` in .NET Framework



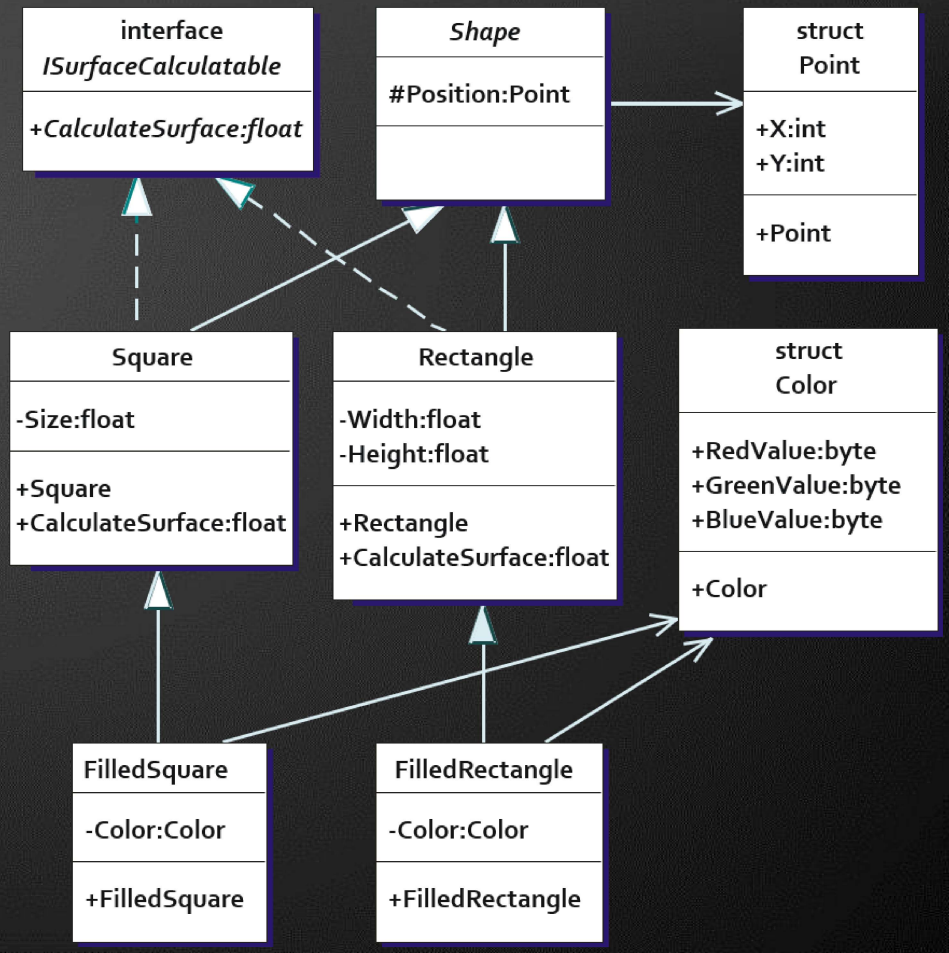
Telerik Academy

Inheritance Hierarchies

- Using inheritance we can create inheritance hierarchies
  - Easily represented by UML class diagrams
- UML class diagrams
  - Classes are represented by rectangles containing their methods and data
  - Relations between classes are shown as arrows
    - Closed triangle arrow means inheritance
    - Other arrows mean some kind of associations



UML Class Diagrams - Example



Telerik Academy

Class Diagrams in Visual Studio

Demo

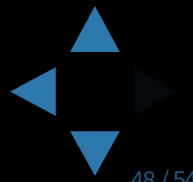
Telerik Academy

Encapsulation

Telerik Academy

Encapsulation

- Encapsulation hides the implementation details
- Class announces some operations (methods) available for its clients – its public interface
- All data members (fields) of a class should be hidden
  - Accessed via properties (read-only and read-write)
- No interface members should be hidden

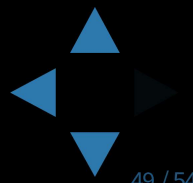


Telerik Academy

Encapsulation – *Example*

- Data fields are private
- Constructors and accessors are defined (getters and setters)

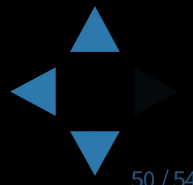
Person
-name : string -age : TimeSpan
+Person(string name, int age) +Name : string { get; set; } +Age : TimeSpan { get; set; }



Telerik Academy

Encapsulation in .NET

- Fields are always declared `private`
  - Accessed through properties in read-only or read-write mode
- Constructors are almost always declared `public`
- Interface methods are always `public`
  - Not explicitly declared with `public`
- Non-interface methods are declared `private` / `protected`



Telerik Academy

Encapsulation – Benefits

- Ensures that structural changes remain local:
  - Changing the class internals does not affect any code outside of the class
  - Changing methods' implementation does not reflect the clients using them
- Encapsulation allows adding some logic when accessing client's data
  - E.g. validation on modifying a property value
- Hiding implementation details reduces complexity → easier maintenance