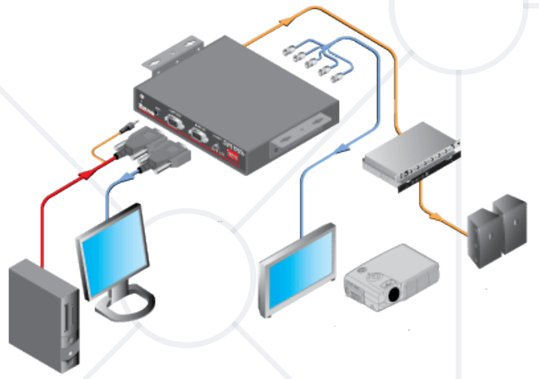
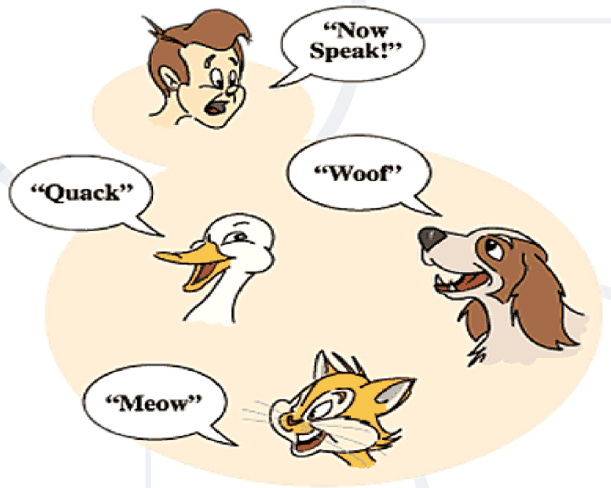
# Abstraction and Polymorphism





SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

## Have a Question?

Software University

sli.do

#prgm-for-qa

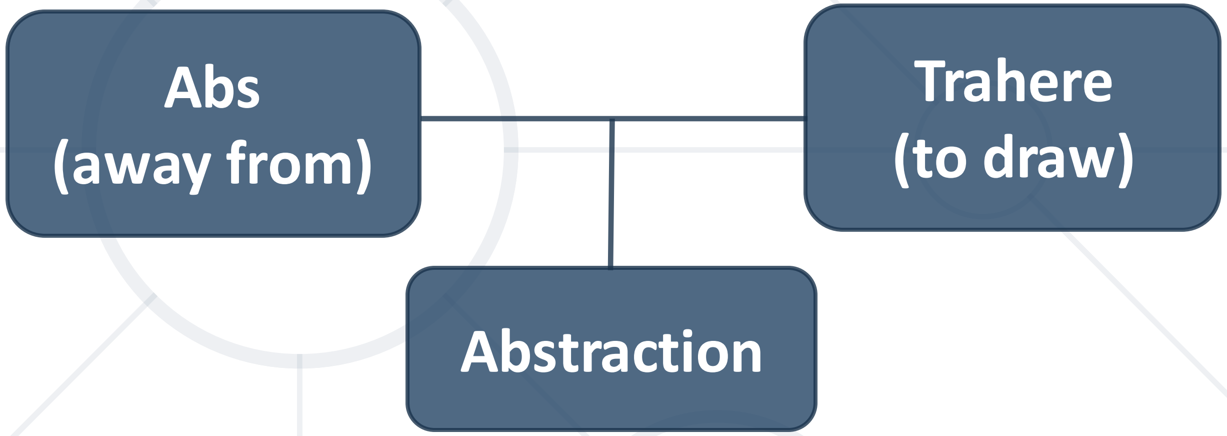
## Table of Contents

## Achieving Abstraction

Abstraction

# What is Abstraction?

- From the Latin



- **Preserving information**, **relevant** in a given context, and **forgetting information** that is **irrelevant** in that context

## Abstraction in OOP

- **Abstraction** means ignoring **irrelevant** features, properties, or functions and emphasizing the **ones …**

"Relevant" to what?

- **… relevant** to the **context** of the **project** we develop
- Abstraction helps **managing** complexity
- Abstraction lets you focus on **what the object does** instead of **how it does it**

## How Do We Achieve Abstraction?

- There are two ways to achieve abstraction
  - Interfaces
  - Abstract class

```
public interface IAnimal {}
public abstract class Mammal {}
public class Person : Mammal, IAnimal {}
```

## Abstraction vs Encapsulation

- Abstraction
  - Process of **hiding the implementation details** and showing only functionality to the user
  - Achieved with **interfaces** and **abstract classes**

- Encapsulation
  - Used to **hide the code** and **data** inside a **single unit to protect the data from the outside world**
  - Achieved with **access modifiers** (private, protected, public … )

# Working with Interfaces

Interfaces

## Interface

- Internal addition by compiler

```
public interface IPrintable {
  void Print();
}
```

Keyword

Name (starts with I per convention)

compiler

```
public interface IPrintable {
  public abstract void Print();
}
```

## Interface Example

- The implementation of **Print()** is provided in class **Document**

```
public interface IPrintable {
  void Print();
}
```

```
class Document : IPrintable {
  public void Print()
  { Console.WriteLine("Hello"); }
```

# Abstract Classes and Methods

## Abstract Classes

## Abstract Class

- **Cannot** be instantiated
- May contain **abstract methods** and **accessors**
- Must provide **implementation** for all **inherited** interface members
- Implementing an interface might map the interface methods onto **abstract** methods

## Abstract Methods

- An **abstract method** is implicitly a **virtual** method
- Abstract method declarations are only permitted in **abstract classes**
- An abstract method declaration provides no actual implementation:

```
public abstract void Build();
```

## Interfaces vs Abstract Classes

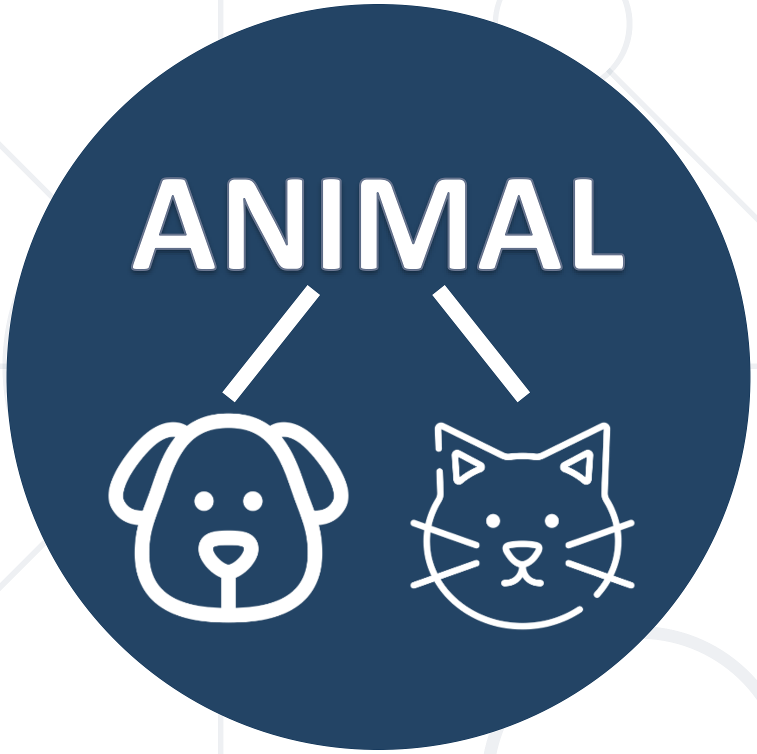
## Interface vs Abstract Class (1)

- Interface
  - A class may **implement several interfaces**
  - **Cannot have access modifiers**, everything is assumed as public
  - **Cannot provide any code**, just the signature
- Abstract Class (AC)
  - May **inherit only one abstract** class
  - Can **provide implementation** and/or just the **signature** that have to be overridden
  - **Can contain access modifiers** for the fields, functions, properties

# Interface vs Abstract Class (2)

- Interface
  - Fields and constants **can't be defined**
  - If we add **a new method we have to track down all the implementations** of the interface and **define implementation** for the new method

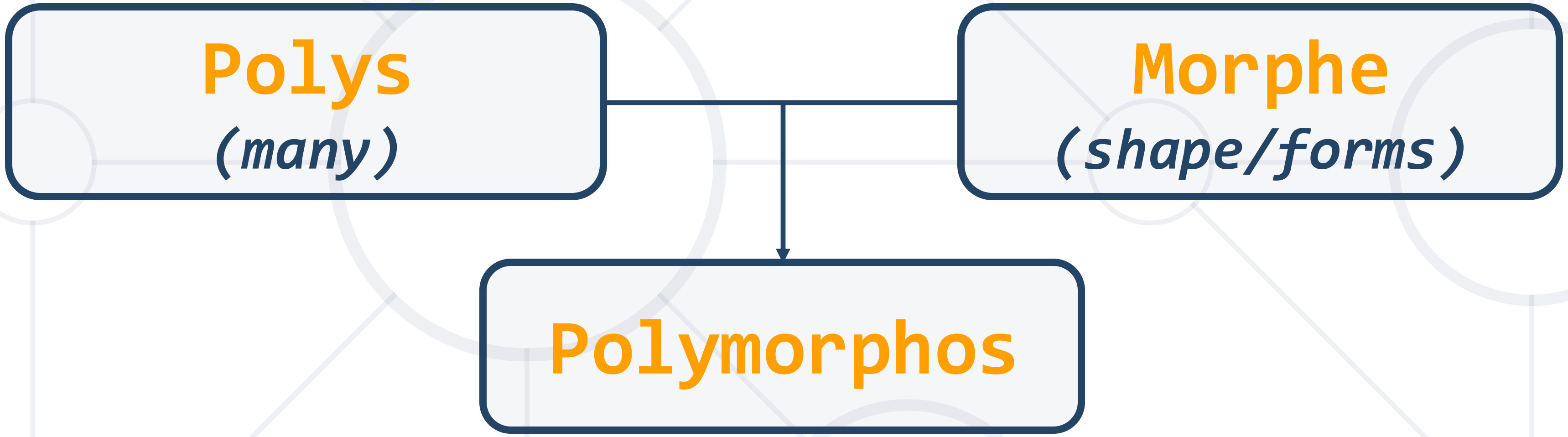
- Abstract Class
  - Fields and constants **can be defined**
  - If we add a **new method we** have the option of **providing default implementation** and therefore all the existing code might work properly



## Polymorphism

# What is Polimorphism?

- From the **Greek**



- This is something similar to a **word having several different meanings depending on the context**

# Polymorphism in OOP

- Ability of an **object** to take on **many forms**

```
public interface IAnimal {}
public abstract class Mammal {}
public class Person : Mammal, IAnimal {}
```

```
Person IS-A Person
```

```
Person IS-AN Animal
```

```
Person IS-AN Object
```

```
Person IS-A Mammal
```

## Variable Type and Data Type

- **Variables Type** is the compile-time type of the variable
- **Data Type** is the actual runtime type of the variable
- If you need an **object method** you need to **cast it or override it**

```
public class Person : Mammal, IAnimal {}
object objPerson = new Person();
IAnimal person    = new Person();
Mammal mammal     = new Person();
Person person     = new Person();
```

Variable Type

Data Type

## Keyword – is

- Runtime check if an **object** is an **instance** of a specific **class**

```
public class Person : Mammal, IAnimal {}
IAnimal person = new Person();
Mammal personOne = new Person();
Person personTwo = new Person();
if (person is Person)
{
  ((Person)person).getSalary();
}
```

Check object type of person

Cast to object type and use its methods

## is Type Pattern

- **Type pattern** - tests whether an expression can be converted to a specified type and casts it to a variable of that type

```
public class Person : Mammal, IAnimal {}
Mammal personOne = new Person();
Person personTwo = new Person();
if (personOne is Person person)
{
  person.GetSalary();
}
```

Checks if object is of type person and casts it

Uses its methods

# is Constant Pattern

- When performing pattern matching with the constant pattern, **is** tests whether an expression equals a specified constant
- Checking for **null** can be performed using the constant pattern

```
int i = 0;
int min = 0, max = 10;
while(true)
{
  Console.WriteLine($"i is {i}");
  i++;
  if(i is max or min) break;
}
```

## Keyword – as

- You can use the as operator to perform certain types of conversions between compatible reference types

```
public class Person : Mammal, IAnimal {}

IAnimal person = new Person();
Mammal personOne = new Person();
Person personTwo;
personTwo = personOne as Person;
if (personTwo != null)
{
  // Do something specific for Person
}
```

Convert Mammal to Person

Check if conversion is successful

## Summary

- **Abstraction**
- How do we achieve abstraction?
- **Interfaces**
- Abstract classes
- Polymorphism - **Definition** and **Types**
- **is** Keyword
- **as** Keyword

Software University

# Questions?

SoftUni

Software University

SoftUni Creative

SoftUni Digital

SoftUni Foundation

SoftUni Kids

Finance Academy

## SoftUni Diamond Partners

Software University

SUPER HOSTING .BG

Coca-Cola HBC Bulgaria

Flutter™ International

INDEAVR Serving the high achievers

AMBITIONED

DRAFT KINGS

SOFTWARE GROUP

BOSCH

Postbank

Решения за твоето утре

PHAR VISION

SmartIT

DXC TECHNOLOGY

createX

# Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, about.softuni.bg
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

## License