# OOP Introduction

# Lesson Objectives

- OOP Introduction
  - ✓ Class Design
  - ✓ Properties
  - ✓ Methods
  - ✓ Constructor
- OOP Characteristics
  - ✓ Abstraction
  - ✓ Encapsulation

Section 1

# OOP INTRODUCTION

# OOP Introduction

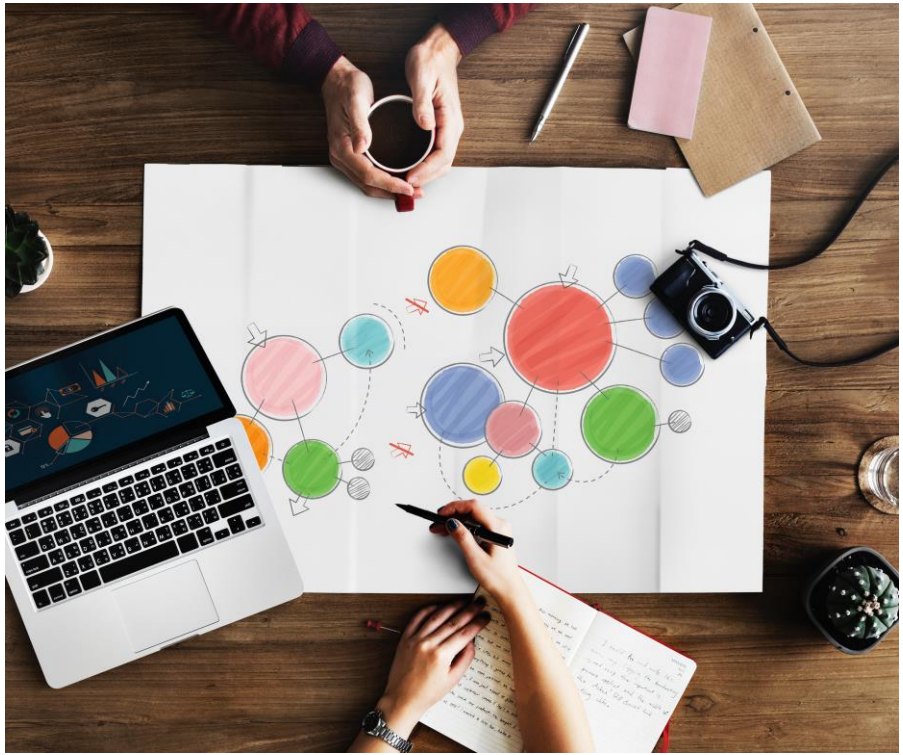
- Programming languages are based on two fundamental concepts: data and ways to manipulate data.
- Traditional languages such as Pascal and C used the procedural approach which focused more on ways to manipulate data rather than on the data itself.
- This approach had several drawbacks such as lack of re-use and lack of maintainability.

# OOP Introduction

## Traditional Programming = Spaghetti code

# OOP Introduction

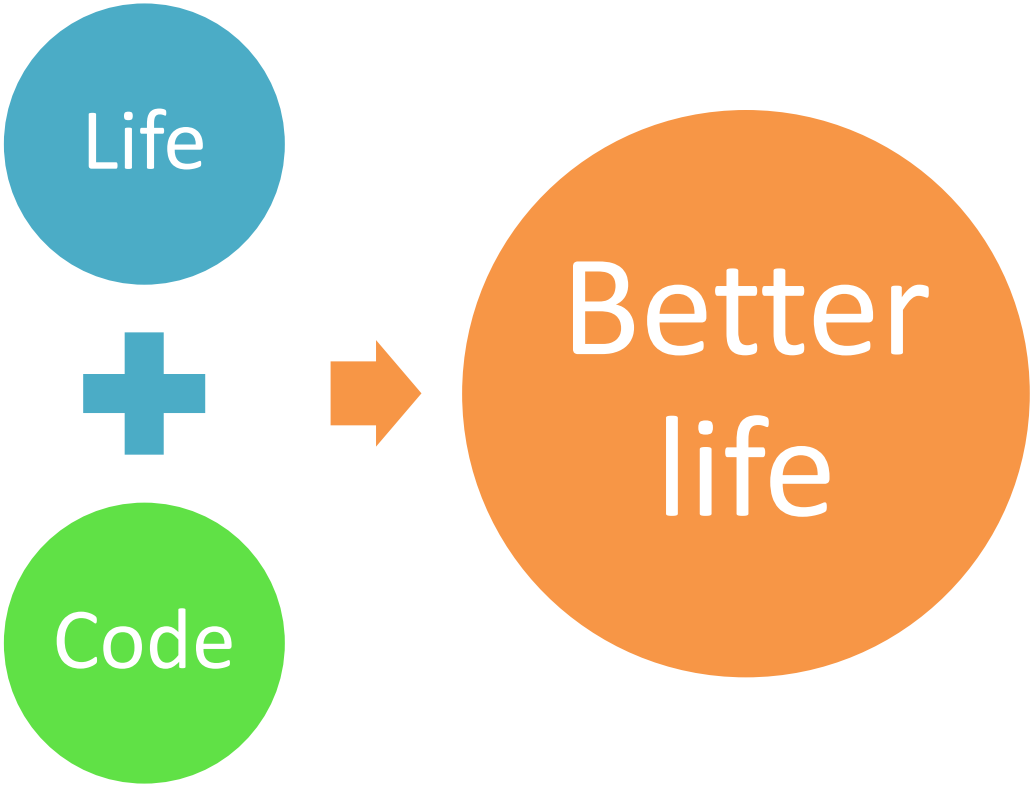
- Object Oriented Programming (OOP) was introduced, which focused on **data** rather than the **ways to manipulate data**.
- The object-oriented approach defines objects as entities having a defined set of values and a defined set of operations that can be performed on these values.



# OOP Introduction

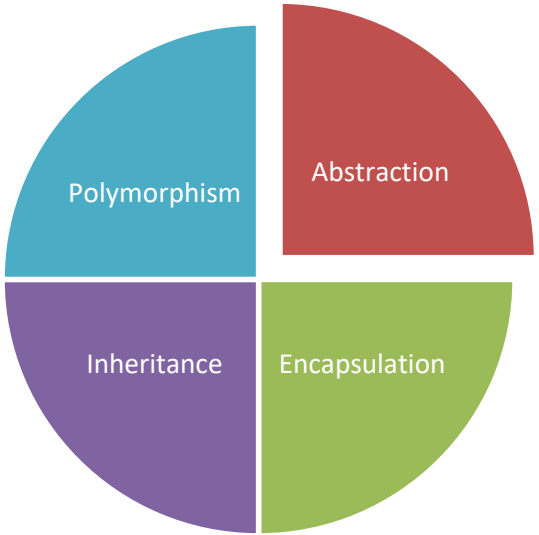
- Give an example, **but**:

# OOP Introduction



# OOP Introduction

- Object-oriented programming provides the following features:
- **Abstraction**:
  - ✓ extracting only the required information from objects
- **Encapsulation**
  - ✓ Details of what a class contains need not be visible to other classes and objects that use it
- **Inheritance**
  - ✓ creating a new class based on the attributes and methods of an existing class
- **Polymorphism**
  - ✓ the ability to behave differently in different situations



# Classes and Objects

- We actually write code for a class, not object
- Use class as a blue-print to create (instantiate) an object

# Classes and Objects

- An object is a tangible entity
- Every object has some characteristics and is capable of performing certain actions.
- An object in a programming language has a unique identity, state, and behavior.
- The state of the object refers to its characteristics or attributes
- The behavior of the object comprises its actions.

# Creating Classes

- The concept of classes in the real world can be extended to the programming world, similar to the concept of objects.
- In object-oriented programming languages like C#, a class is a template or blueprint which defines the state and behavior of all objects belonging to that class.
- A class comprises fields, properties, methods, and so on, collectively called data members of the class. In C#, the class declaration starts with the class keyword followed by the name of the class.

# Creating Classes

- **Syntax:**

```
class <class name>
{
    // class body
}
```

- **Class name should follow rules:**
  - ✓ Cannot be a C# keyword.
  - ✓ Cannot begin with a digit but can begin with the '@' character or an underscore (_).

# Create Classes

```
3 references
class Baby
{
    0 references
    public string Name { get; set; }
    0 references
    public decimal Weight { get; set; }
    0 references
    public string Hobby { get; set; }
    2 references
    public bool IsHungry { get; set; }

    1 reference
    public void Eat() { /* ... */ }

    1 reference
    public void Cry() { /* ... */  }

    0 references
    public void Crawl() { /* ... */ }
}
```

# Methods

- Methods are functions declared in a class and may be used to perform operations on class variables.
- They are blocks of code that can take parameters and may or may not return a value.
- A method implements the behavior of an object, which can be accessed by instantiating the object of the class in which it is defined and then invoking the method.
- Methods specify the manner in which a particular operation is to be carried out on the required data members of the class.

# Creating methods

- Cannot be a C# keyword, cannot contain spaces, and cannot begin with a digit
- Can begin with a letter, underscore, or the “@” character
- Should be a verb or verb phrase
- Should be in **PascalCasing**
- Should be simple, descriptive, and meaningful.
- Avoid using **Abbreviations**

# Instantiating Objects

- It is necessary to create an object of the class to access the variables and methods defined within it.
- In C#, an object is instantiated using the new keyword. On encountering the new keyword, the Just-in-Time (JIT) compiler allocates memory for the object and returns a reference of that allocated memory.

# Instantiating and Use Objects

- Syntax:

  <class name> <object name> = new <class name>();

  var  <object name> = new <class name>();

- Examples:

```
0 references
static void Main(string[] args)
{
    Baby peter = new Baby();
    peter.IsHungry = true;
    peter.Cry();

    var richy = new Baby();
    richy.IsHungry = true;
    richy.Eat();
}
```

# Constructor

- Whenever instantiating a object of any class, its constructor is called.
- A class or struct may have multiple constructors that take different arguments.
- Constructors enable the programmer to set default values, limit instantiation, and write code that is flexible and easy to read.

# Constructor

- As a method, you can:
  - ✓ Write constructor with zero, one or multiple parameters
  - ✓ Write multiple constructors as method overload
- As a special method:
  - ✓ If you don't provide a constructor for your class, C# creates one by default that instantiates the object and sets member variables to the default values
  - ✓ There are no return value
  - ✓ It usually used to set initial values for fields

# Constructor syntax

- A constructor is a method whose name is the same as the name of its type.
- Its method signature includes only the method name and its parameter list; it does not include a return type.

# Constructor syntax

```
2 references
public Baby()
{
    // Name = null
    // Weight = 0.0m
    // Hobby = null
    // IsHungry = false
}

0 references
public Baby(string name, decimal weight, string hobby, bool isHungry)
{
    Name = name;
    Weight = weight;
    Hobby = hobby;
    IsHungry = isHungry;
}
```

# The this keyword

- The this keyword is used to refer to the current object of the class to resolve conflicts between variables having same names and to pass the current object as a parameter.
  - ✓ Using ‘this’ keyword to refer current class instance members
  - ✓ Using this() to invoke the constructor in same class
  - ✓ Using ‘this’ keyword to invoke current class method
  - ✓ Using ‘this’ keyword as method parameter
  - ✓ Using this keyword to declare an indexer

# The this keyword

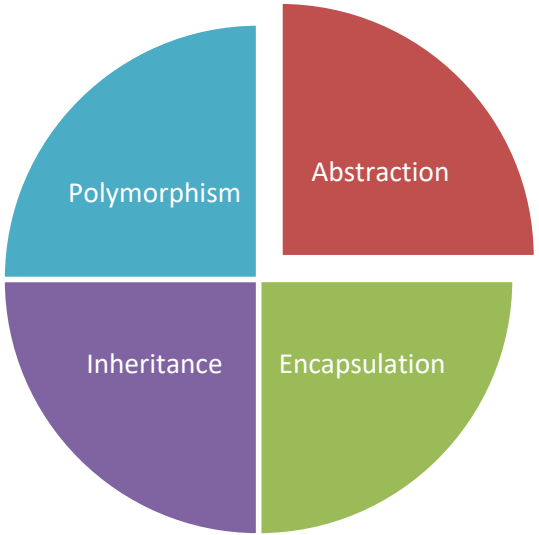
```
0 references
public Baby(string name, decimal weight, string hobby, bool isHungry)
{
    this.Name = name;
    this.Weight = weight;
    this.Hobby = hobby;
    this.IsHungry = isHungry;
}

1 reference
public void Eat(string food)
{
    if (this.IsHungry)
    {
        this.Cry();
        if (!string.IsNullOrEmpty(food) && food.Equals(this.Hobby))
        {
            Console.WriteLine($"Baby name {this.Name} is eating {food}");
            this.IsHungry = false;
        }
    }
}
```

Section 2

# OOP CHARACTERISTICS

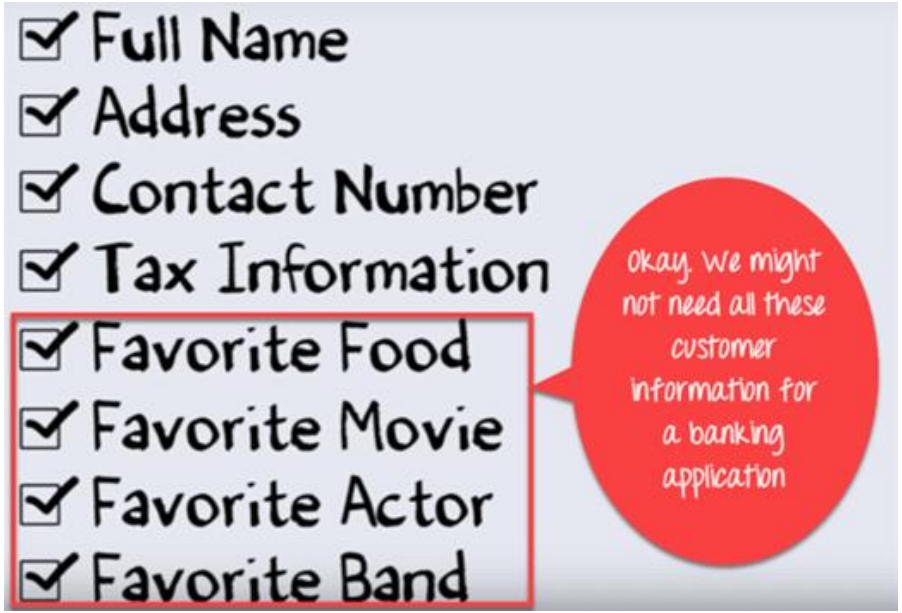
# OOP Characteristics

- **Abstraction**:
  - ✓ extracting only the required information from objects
- **Encapsulation**
  - ✓ Details of what a class contains need not be visible to other classes and objects that use it
- **Inheritance**
  - ✓ creating a new class based on the attributes and methods of an existing class
- **Polymorphism**
  - ✓ the ability to behave differently in different situations



# Abstraction

- Suppose you want to create a banking application and you are asked to collect all the information about your customer. There are chances that you will come up with following information about the customer



# Abstraction

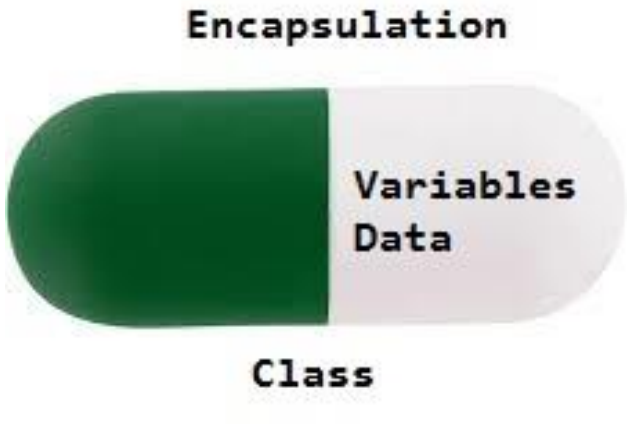
- But, not all of the above information is required to create a banking application.
- So, you need to select only the useful information for your banking application from that pool. Data like name, address, tax information, etc. make sense for a banking application

# Abstraction

- Abstraction is selecting data from a **larger pool** to show only the **relevant details** of the object to the user.
- Abstraction “shows” only the essential attributes and “hides” unnecessary information.
- It helps to reduce programming complexity and effort.
- Abstraction is accomplished using Abstract classes, Abstract methods, and interfaces.

# Encapsulation

- Encapsulation is defined as the wrapping up of data under a single unit.
- It is the mechanism that binds together code and the data it manipulates.
- Encapsulation is a protective shield that prevents the data from being accessed by the code outside this shield.



# Encapsulation

- The variables or data of a class are hidden from any other class and can be accessed only through any member function of own class in which they are declared.
- The data in a class is hidden from other classes, so it is also known as **data-hiding**.
- **Encapsulation can be achieved by:** Declaring all the variables in the class as private and using **C# Properties** in the class to set and get the values of variables.

# Advantages of Encapsulation

- Data Hiding
  - ✓ The user will have no idea about the inner implementation of the class
- Increased Flexibility
  - ✓ We can make the variables of the class as read-only or write-only depending on our requirement.
- Reusability
  - ✓ Improves the re-usability and easy to change with new requirements
- Testing code is easy
  - ✓ Easy to test for unit testing.

# Access Modifiers

- public
  - ✓ The type or member can be accessed by any other code in the same assembly or another assembly that references it.
- private
  - ✓ The type or member can be accessed only by code in the same class or struct.
- protected
  - ✓ The type or member can be accessed only by code in the same class, or in a class that is derived from that class.
- internal
  - ✓ The type or member can be accessed by any code in the same assembly, but not from another assembly.

# Access Modifiers

- protected internal
  - ✓ The type or member can be accessed by any code in the assembly in which it's declared, or from within a derived class in another assembly.
- private protected
  - ✓ The type or member can be accessed only within its declaring assembly, by code in the same class or in a type that is derived from that class.

# Access Modifiers

| | class | class member |
|---|---|---|
| public | v | v |
| private | | v |
| protected | | v |
| internal | v | v |
| protected internal | | v |
| private internal | | v |
| <default> | internal | private |

# Practice time

# Lesson Summary

# Thank you