## Application of OOP

In terms of Benefits, OOP offers various benefits to both the designer and the user of the program. The various benefits of OOP are as follows:

- Through the process of inheritance, redundant codes can be eliminated and the use of classes can be extended.

- Programs can be built from standard working modules that communicate with each other. This saves development time and increase productivity.

- Data hiding helps the programmer to build secure programs that cannot be touched by the code of other components of the program.

## Application of OOP

- OOP allows multiple instances of an object to co-exist without any interference.
- The data-centered design approach in OOP capture more details of a model.
- It helps in proper communication between objects by various message passing techniques, simplifying the interface description.

## Classes and Objects

- Class
  - Is a conceptual representation of all the entities that share common attributes and behaviors.
  - Defines the attributes and behaviors of all the instances of the class.
  - Is like a blueprint or a model.
- Object
  - Is an instance of a class.
  - Has individual copy of the common attributes and share a common set of behaviors.
  - Is a runtime entity.

## Advantages of Using Classes and Objects

- Maintenance of code by introducing modularity.
- Encapsulation of internal complexities in code from end-users.
- Reuse of code across applications.
- Support for a single interface to implement multiple methods.

## Constructors

- A constructor is a special type of method that is invoked when you create a new instance of a class.
- A constructor is used to initialize the members of the class.
- The name of a constructor is the same as the name of the class that contains it.
- A constructor is special member function within the class which is executed when an object of the class is created.
- A constructor can be modified to accept the user-supplied values at run time.
- Objects can be initialized using the default constructor with values hard-coded in the program. But there might be a requirement where the variables need to be initialized with user supplied values.

## Types of Constructors

- **Instance constructors**: They are called whenever an instance of a class is created. These constructors are used to initialize the data members of the class.

- **Static constructors**: They are used to initialize the static variables of a class. These variables are created using `static` keyword and they store values that can be shared by all the instances of a class.

## Destructors

- Destructors are special methods that are used to release the instance of a class from memory.
- A class can have only one destructor.
- The purpose of the destructor is to perform the required memory cleanup action.
- The .NET Framework automatically runs the destructor to destroy objects in the memory.

## Destructors

- A destructor has the same name as its class but is prefixed with a ~ , which is the symbol of tilde.
- Destructors cannot be inherited or overloaded.
- Garbage collection is a process that automatically frees the memory of objects that are no more in use.
- The decision to invoke the destructor is made by a special program of C# known as the garbage collector.
- The process of garbage collection happens automatically. It ensures that:
  - Objects get destroyed
  - Only unused objects are destroyed

Example of Constructor and Destructor

```
using System;
    public class Calculator
    {
      public Calculator()
       {
          Console.WriteLine("Constructor Invoked");
       }
      static Calculator()
      {
        Console.WriteLine("I am static ctor");
      }
     ~Calculator()
       {
          Console.WriteLine ("Destructor Invoked");
       }
       public static void Main(string[] args)
       {
          Calculator Calc1 = new Calculator();
       }
    }
```

## Polymorphism

- In Object-Oriented Programming (OOPs), polymorphism allows one interface to be used for multiple functions.

- Polymorphism reduces the complexity within the functions of a class of a program.

- Polymorphism can either be static or dynamic.

## Static Polymorphism

- Static polymorphism refers to an entity, which exists in various forms simultaneously.
- C# uses two approaches to implement static polymorphism. These are:
  - **Function overloading**: This approach allows using the same name for two or more functions. Each redefinition of a function must use different types of parameters, sequence of parameters, or a number of parameters.
  - **Operator overloading**: This approach allows user-defined types such as structures and classes, to use overloaded operators for easy manipulation of their objects.

## Dynamic Polymorphism

- In dynamic polymorphism, the decision about function execution is made at run time.
- Dynamic polymorphism is more useful than static polymorphism as it provides much more flexibility for manipulating the objects.
- C# uses two approaches to implement dynamic polymorphism:
  - **Abstract classes**: Are the special type of base classes that consist of abstract class members.
  - **Virtual functions**: Are the functions that do not really exist, however, appear to be present in some parts of the program.

## Function Overloading

- Function overloading is implemented by defining two or more functions in a class sharing the same name.
- In function overloading, each definition of a function must differ in its function signature.
- The signature of a function is defined by:
  - The number of parameters
  - The data types of parameters
  - The sequence of the parameters

# Constructor Overloading

- Constructors can also be parameterized, and therefore, they can be overloaded.
- Overloaded constructors are commonly used in C# to provide flexibility while creating an object.

**Example**

```
public class Book
{
        public int price;
        public string title;
        public Book() { price=0;  title="”; }
        public Book(int p,string t) {   price=p;  title=t; }
}
```

## Operator Overloading

- Operator overloading provides additional capabilities to C# operators when they are applied to user-defined data types.
- To use operators with user-defined data types, they need to be overloaded according to a programmer's requirement.
- Only the predefined set of C# operators can be overloaded.

## Properties

- A property is a member that represents an item of data in a class instance or class
- Is a type of variable that stores the values of an object of a class or a structure.
- Can help you define a property as read-only, write-only, or read/write type.
- Is of two types:
  - Get accessor is used to retrieve the values from a property.
  - Set accessor is used to assign values to a property.

**Property Example**

```
public class Emp
{
        private int code;
        public int CODE
       {
                get {     return code;  }
                set {     code=value;  }
      }
}
```

## Automatically Implemented Properties

*Automatically implemented properties*, or *auto-implemented properties*, which allow you to just declare the property, without declaring a backing field.

The important points are the following:

- You do not declare the backing field—the compiler allocates the storage for you, based on the type of the property.
- You cannot supply the bodies of the accessors—they must be declared simply as semicolons.
- You cannot access the backing field other than through the accessors. Read-only or Write-only auto implemented properties are not allowed. But you can do so by marking get or set as private.

**Auto Implemented Property Example**

```
public class Emp
{
        public int CODE
       {
                get;
                set;
      }
}
```