**Gaming - Platform**

* Unity engine
* Unreal engine
* Go dot engine
* Build box
* Construct 2
* Cry engine 3

# **MonoBehaviour**

* MonoBehaviour is a base class that many Unity scripts derive from.
* Without monobehaviour we can not attach script in to the gameobject.
* MonoBehaviours always exist as a [Component](https://docs.unity3d.com/ScriptReference/Component.html) of a GameObject, and can be instantiated with [GameObject.AddComponent](https://docs.unity3d.com/ScriptReference/GameObject.AddComponent.html).

## **C# Programming**

1. C# (C-Sharp) is a programming language developed by Microsoft that runs on the .NET Framework.
2. C# is used to develop web apps, desktop apps, mobile apps, games and much more.

**Datatype, variable, keyword**

A data type specifies the size and type of variable values.

| **Data Type** | **Size** |
| --- | --- |
| int | 4 Byte |
| long | 8 Byte |
| float | 4 Byte |
| double | 8 Byte |
| bool | 1 Byte |
| char | 2 Byte |
| string | 2 Byte Per character |

## **Difference Between C++ and C#**

| **Feature** | **C++** | **C#** |
| --- | --- | --- |
| **Multiple Inheritance** | C++ support multiple inheritance through classes. Means that a class can extend more than one class at a time. | C# does not support any multiple inheritances through classes. |
| **Pointers** | In C++ pointers can be used anywhere in the program. | In C# pointers can be used only in unsafe mode. |
| **Language Type** | C++ is a low level language. | C# is high level object oriented language. |
| **Level of Difficulty** | C++ includes very complex features. | C# is quite easy because it has the well-defined hierarchy of classes. |
| **Application Types** | C++ is typically used for console applications. | C# is used to develop mobile, windows, and console applications. |
| **Object Oriented** | C++ is not a pure object-oriented programming language due to the primitive data types. | C# is a pure object-oriented programming language. |
| **Access Specifiers** | The access modifiers are public, private, protected. It does not contain internal & protected internal access modifiers. | In C# public, private, protected, internal & protected internal are used for access specifiers. |
| **Control statement** | It does not contain such extra flow control statement. | In addition to for, while and do while; it has another flow control statement called for each. |
| **Garbage Collection** | C++ do not support garbage collection. | Garbage collection is supported by C# |
| **Types of Projects** | It is mainly used for such projects that focus on accessing the hardware and better performance. | It is mainly used in modern application development. |

## 

## **Classes and Objects**

**Class**

* Collection of method and datatypes. Virtual attribute that define only properties.
* Class is simple a representation of a type of object. It is the blue/plan/template that describe the details of an object.

**public** **class** class\_name

{

//field or data member

//field or data member

}

**Object**

* Collection object is termed as an instance of a class and it has it’s own state,behaviour and identity.
* An object is created from a class.
* In C#, Object is a real world entity, for example, chair, car, pen, mobile, laptop etc.
* Object is an instance of a class. All the members of the class can be accessed through object.

**ClassName** Object\_name = new **ClassName**();

## **C# Type Casting**

Type casting is when you assign a value of one data type to another type.

* **Implicit Casting** (automatically) - converting a smaller type to a larger type size  
  char -> int -> long -> float -> double
* **Explicit Casting** (manually) - converting a larger type to a smaller size type  
  double -> float -> long -> int -> char

## **C# Control Statements and Loops**

* if statement

if (*condition*)

{

*// block of code to be executed if the condition is True*

}

* if-else statement

if (*condition*)

{

*// block of code to be executed if the condition is True*

}

else

{

*// block of code to be executed if the condition is False*

}

* if-else-if ladder

if (*condition1*)

{

*// block of code to be executed if condition1 is True*

}

else if (*condition2*)

{

*// block of code to be executed if the condition1 is false and condition2 is True*

}

else

{

*// block of code to be executed if the condition1 is false and condition2 is False*

}

* While Loop

while (*condition*)

{

*// code block to be executed*

}

* Do While Loop

do

{

*// code block to be executed*

}

while (*condition*);

* For Loop.

for (*statement 1*; *statement 2*; *statement 3*)

{

*// code block to be executed*

}

* Foreach Loop.

foreach (*type* *variableName* in *arrayName*)

{

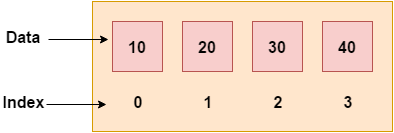
*// code block to be executed*

}

**Array and List**

**Array**

* Arrays are used to store multiple values in a single variable.
* To declare an array, define the variable type with square brackets.
* The size of an array is determined at the time of its creation and cannot be changed later.
* Array elements can be accessed using an index, starting from 0 for the first element and increasing by one for each subsequent element.
* Arrays are part of the C# language and are available as a built-in data type.



**List**

* Lists are dynamic-size collections that can store elements of different data types and automatically resize as needed.
* The List<T> class is part of the System.Collections.Generic namespace in C# and provides a more flexible alternative to arrays.
* Elements in a List are also accessed using an index, just like arrays.
* List<T> provides many useful methods for adding, removing, and manipulating elements, making it more convenient for dynamic data storage.

**C# Different between List and Array**

| **Array** | **List** |
| --- | --- |
| Arrays are continuous in memory that makes it exhausting (in a performance sense) to insert parts within the middle of the list. The advantage is that the ability to perform random access. | Lists, on the opposite hand, are parts that unfold concerning in memory, link along. This enables straightforward insertion within the list; however, random access while not further data structures isn’t doable. |
| An array could be a system; that’s to mention, it’s a particular approach to organising data within the memory device. | A list is an abstract data type; that is to say, it is any data structure that supports a specific bunch of operations. |
| An array is a collection of homogeneous parts. | A list is a collection of heterogeneous elements. |
| Array memory allocated is static and continuous. | List memory allocated is dynamic and Random. |
| A user needn’t ought to confine track of the next memory allocation. | A user must confine Track of the next location wherever memory is allotted. |

## **C# Methods**

* method is a block of code which only runs when it is called.
* You can pass data, known as parameters, into a method.
* Methods are used to perform certain actions, and they are also known as functions.

<access-specifier><**return**-type>FunctionName(<parameters>)

{

// function body

// return statement

}

# **C# Access Modifiers**

* Access modifiers are keywords used to control the visibility and accessibility of types (classes, structs, enums), members (fields, methods, properties, events), and nested types within a program.
* Access modifiers define which parts of a program can access certain elements and which parts cannot.
* Provides several access modifiers, each serving a specific purpose.

| Modifier | Description |
| --- | --- |
| public | The code is accessible for all classes. |
| private | The code is only accessible within the same class. |
| protected | The code is accessible within the same class, or in a class that is inherited from that class. You will learn more about inheritance in a later chapter |
| internal | The code is only accessible within its own assembly, but not from another assembly. You will learn more about this in a later chapter |

# **C# Structure and Enum**

**Structure**

* Structures (also called structs) are a way to group several related variables into one place.
* Each variable in the structure is known as a member of the structure.
* A structure can implement one or more interfaces.
* Structures cannot be used as a base for other structures or classes.
* Unlike classes, structures cannot inherit other structures or classes.

struct myStructure {

int myNum;

char myLetter;

char myString[30];

};

**Enum**

* An enum is a special type that represents a group of constants (unchangeable values).

enum Level {

LOW,

MEDIUM,

HIGH

};

# **C# Inheritance and polymorphism**

**Inheritance**

* Inheritance allows us to define a class in terms of another class, which makes it easier to create and maintain an application.
* Inheritance is a concept where one class share the structure and behaviour define in another class.
* If inheritance apply one one class is called single inheritance and if it depends on multiple classes then it is called multiple inheritance.
* In C#, it is possible to inherit fields and methods from one class to another. We group the "inheritance concept" into two categories:

**Derived Class (child,super)** - the class that inherits from another class.

**Base Class (parent,sub)** - the class being inherited from.

<access-specifier> class <base\_class> {

...

}

class <derived\_class> : <base\_class> {

...

}

* Inheritance allows us to define a class in terms of another class, which makes it easier to create and maintain an application.
* When creating a class, instead of writing completely new data members and member functions, the programmer can designate that the new class should inherit the members of an existing class. This existing class is called the base class, and the new class is referred to as the derived class.

**Polymorphism**

* The word polymorphism means having many forms.
* In object-oriented programming paradigm, polymorphism is often expressed as 'one interface, multiple functions'.
* Polymorphism is nothing but assigning behaviour or value in a subclass to something that was already declared in the main class.

C# provides two techniques to implement static polymorphism. They are −

1. **Function overloading**
2. **Operator overloading**

## **Function Overloading**

* multiple methods can have the same name with different parameters.
* The **advantage** of method overloading is that it increases the readability of the program because you don't need to use different names for same action.

You can perform method overloading in C# by two ways:

1. By changing number of arguments.
2. By changing data type of the arguments.

## **Function Overriding**

* If derived class defines same method as defined in its base class, it is known as method overriding in C#. It is used to achieve runtime polymorphism.
* It enables you to provide specific implementation of the method which is already provided by its base class.
* To perform method overriding in C#, you need to use **virtual** keyword with base class method and **override** keyword with derived class method.
* Method overriding is a feature that allows sub class to provide implementation of a method that is already defined in the main class.
* This will override the implementation in the superclass by proving the same method name same parameter and same return type.

# **C# Abstraction and Interface**

## **Abstraction**

* Data abstraction is the process of hiding certain details and showing only essential information to the user.
* Abstraction can be achieved by two ways:

1. **Abstract class**
2. **Interface**

he abstract keyword is used for classes and methods:

* **Abstract class**: is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).
* **Abstract method**: can only be used in an abstract class, and it does not have a body. The body is provided by the derived class (inherited from).
* abstract class is a class which is declared abstract.
* It can have abstract and non-abstract methods. It cannot be instantiated. Its implementation must be provided by derived classes. Here, derived class is forced to provide the implementation of all the abstract methods.

abstract class Classname

{

public abstract void method\_name();

public void method\_Name()

{

// body of method;

}

}

# **Interface**

* An interface is a completely "abstract class", which can only contain abstract methods and properties (with empty bodies):
* Interface in C# is a blueprint of a class. It is like abstract class because all the methods which are declared inside the interface are abstract methods. It cannot have method body and cannot be instantiated.

interface ClassName

{

Void Method\_name(); // interface method (does not have a body)

Void Method\_name(); // interface method (does not have a body)

}

## 