BridgeLabz

Employability Delivered

.NET

.NET

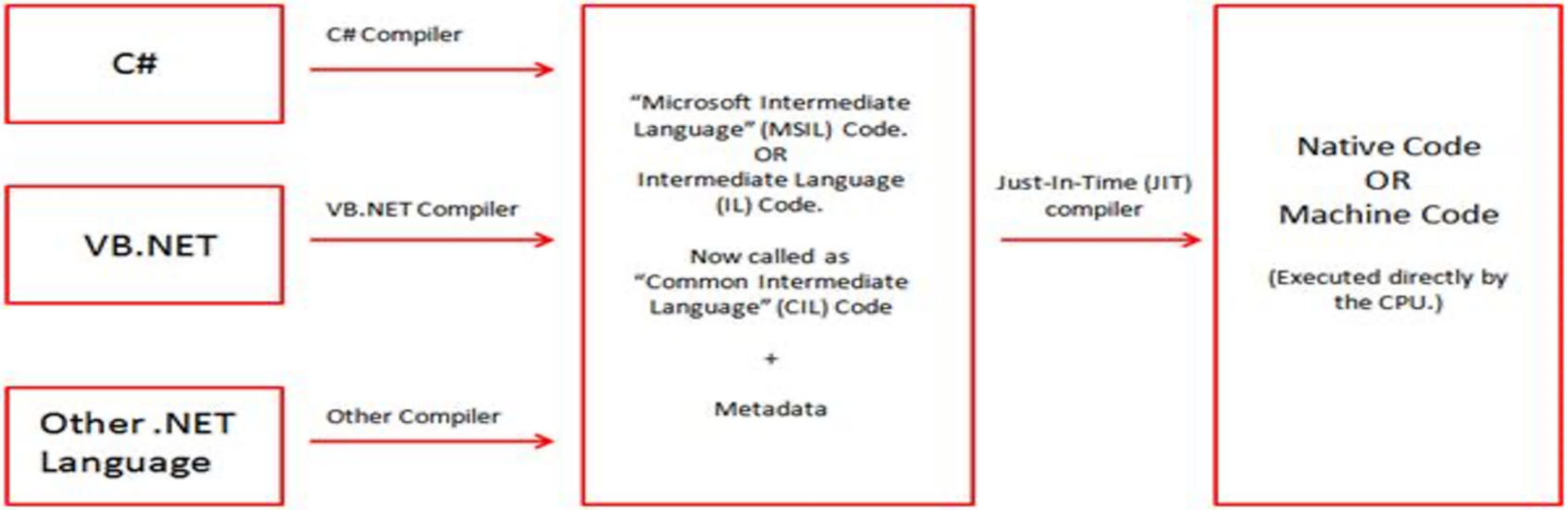
1. .NET Terminology
2. .NET is a Platform
3. CLR Architecture
4. C# Features

.NET Terminology

- **.NET Installation:** Install Visual studio
- **.NET Program Execution:** It starts with we writing the program, then compiling the program and at last running the program.
- **Common Language Runtime(CLR):** Common Language Runtime (CLR) is a managed execution environment that is part of Microsoft's . NET framework. CLR manages the execution of programs written in different supported languages. CLR transforms source code into a form of bytecode known as Common Intermediate Language (CIL)
- **Bytecode:** JITcompiler of .NET compiles the source code into bytecode so that it can be executed by CLR.

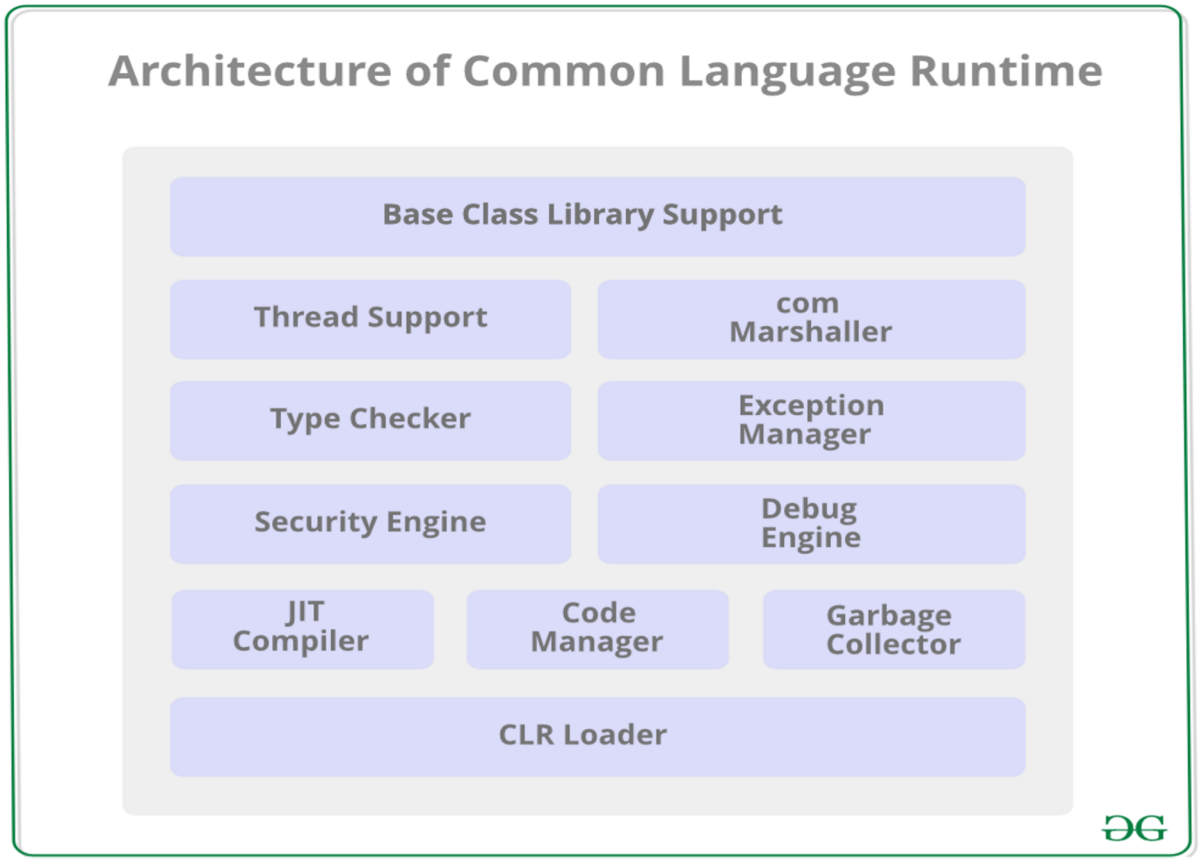
.NET is a platform dependent language

NET is Platform Dependent because once the code is written it is complied into Microsoft Intermediate Language(MSIL) code which is independent of platform but it is half compiled code,then Common Language Runtime(CLR) convert it into device specific code i.e it is platform dependent. (MSIL is send to JIT through CLR).



CLR Architecture

BridgeLabz
Employability Delivered



Base Class Library Support: The Common Language Runtime provides support for the base class library. The BCL contains multiple libraries that provide various features such as ***Collections***, ***I/O***, ***XML***, ***DataType definitions***, etc. for the multiple ***.NET*** programming languages.

Thread Support: The CLR provides thread support for managing the parallel execution of multiple threads. The System.Threading class is used as the base class for this.

COM Marshaller: Communication with the COM (Component Object Model) component in the .NET application is provided using the COM marshaller. This provides the COM interoperability support.

Type Checker: Type safety is provided by the type checker by using the Common Type System (CTS) and the Common Language Specification (CLS) that are provided in the CLR to verify the types that are used in an application.

Exception Manager: The exception manager in the CLR handles the exceptions regardless of the .NET Language that created them. For a particular application, the catch block of the exceptions are executed in case they occur and if there is no catch block then the application is terminated.

Security Engine: The security engine in the CLR handles the security permissions at various levels such as the code level, folder level, and machine level. This is done using the various tools that are provided in the .NET framework.

Debug Engine: An application can be debugged during the run-time using the debug engine. There are various ICorDebug interfaces that are used to track the managed code of the application that is being debugged.

JIT Compiler: The JIT compiler in the CLR converts the Microsoft Intermediate Language (MSIL) into the machine code that is specific to the computer environment that the JIT compiler runs on. The compiled MSIL is stored so that it is available for subsequent calls if required.

Code Manager: The code manager in CLR manages the code developed in the .NET framework i.e. the managed code. The managed code is converted to intermediate language by a language-specific compiler and then the intermediate language is converted into the machine code by the Just-In-Time (JIT) compiler.

Garbage Collector: Automatic memory management is made possible using the garbage collector in CLR. The garbage collector automatically releases the memory space after it is no longer required so that it can be reallocated.

CLR Loader: Various modules, resources, assemblies, etc. are loaded by the CLR loader. Also, this loader loads the modules on demand if they are actually required so that the program initialization time is faster and the resources consumed are lesser.

C# Features

- **C# is an Object Oriented language:** C# is a pure Object oriented language where even writing the smallest of the Program starts with writing a class. Its essentially a way of organizing programs as collection of objects, each of which represents an instance of a class. And Classes are Organized using Object Oriented Concepts of Abstraction, Encapsulation, Inheritance, Polymorphism, and Association.
- **Robust Language:** Robust means reliable. Its made reliable by checking possible errors at compile time and runtime and supporting runtime features like Garbage Collection, Exception Handling and memory management which also makes it Secure.
- **Interoperability:** Interoperability process enables the C# programs to do almost anything that a native C++ application can do.
- **Multithreading:** Multithreading is a C# feature that allows concurrent execution of two or more parts of a program for maximum utilization of CPU.
- **Scalable and Updateable:** C# is automatic scalable and updateable programming language. For updating our application we delete the old files and update them with new ones.

C# Programming Constructs

1. Sequences, Selection & Repetition
2. Class, Object and Methods
3. Key Concepts
4. Patterns

Note:  C# cheatsheet

1. Sequences, Selection & Repetition

- Sequences are simple C# Statement
- A selection statement provides for selection between alternatives. Consists of if, else & switch statements
- A repetition construct causes a group of one or more program statements to be invoked repeatedly until some end condition is met. Consists of Fixed for loop and Variable while loop

UC 1

Check Employee is Present or Absent

Check Employee Presence UC 1

```
class Program
{
    0 references
    static void Main(string[] args)
    {
        //Constants
        int IS_FULL_TIME = 1;
        Random random = new Random();
        //Computation
        int empCheck = random.Next(0, 2);
        if (empCheck == IS_FULL_TIME)
        {
            Console.WriteLine("Employee is Present");
        }
        else
        {
            Console.WriteLine("Employee is Present");
        }
    }
}
```

BridgeLabz
Employability Delivered

UC 2

Calculate Daily Employee Wage

Calculating Employee Wage UC 2

```
class Program
{
    0 references
    static void Main(string[] args)
    {
        //Constants
        int IS_FULL_TIME = 1;
        int EMP_RATE_PER_HOUR = 20;
        //Variables
        int empHrs = 0;
        int empWage = 0;
        Random random = new Random();
        //Computation
        int empCheck = random.Next(0, 2);
        if (empCheck == IS_FULL_TIME)
        {
            empHrs = 8;
        }
        else
        {
            empHrs = 0;
        }
        empWage = empHrs * EMP_RATE_PER_HOUR;
        Console.WriteLine("Emp Wage : " + empWage);
    }
}
```

BridgeLabz
Employability Delivered

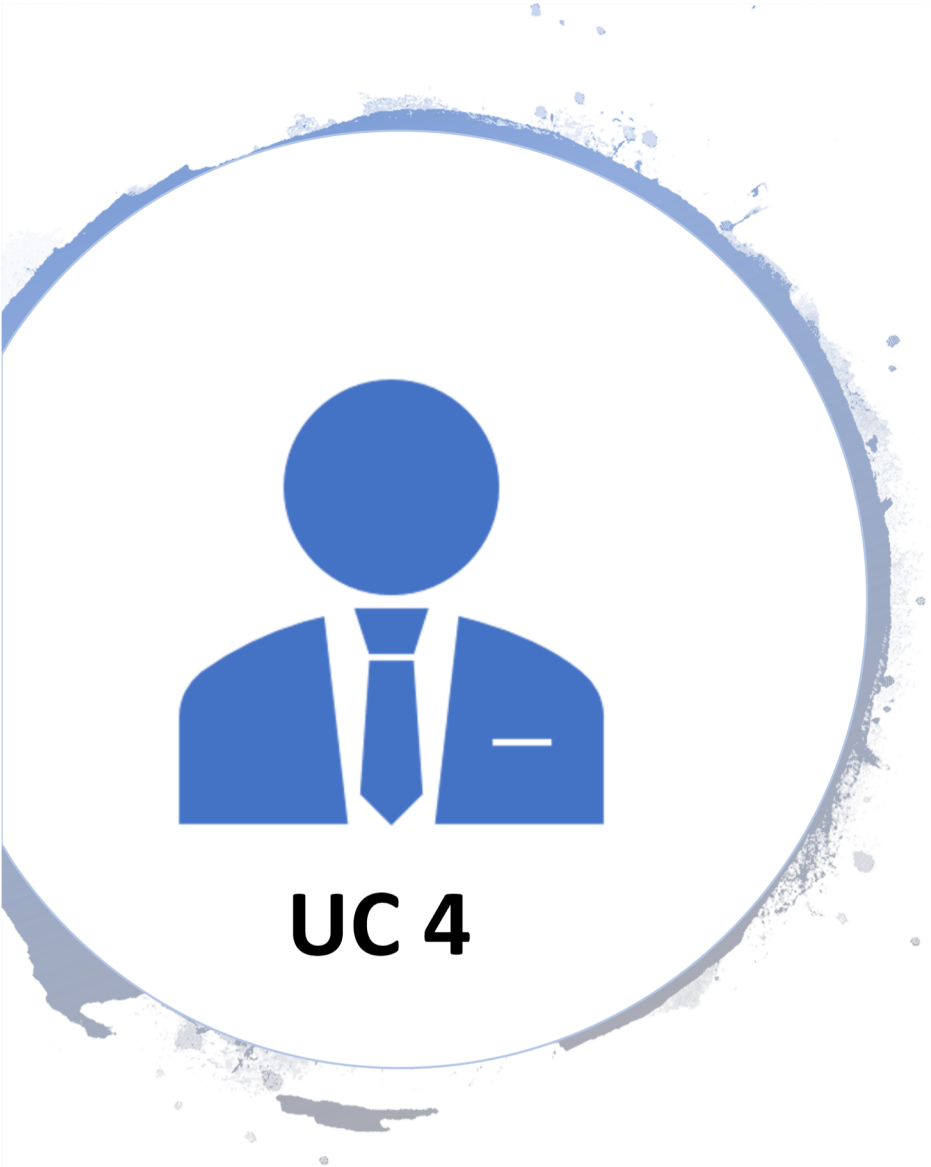
UC 3

Add Part time Employee & Wage

Add Part time Employee

```
class Program
{
    0 references
    static void Main(string[] args)
    {
        //Constants
        int IS_PART_TIME = 1;
        int IS_FULL_TIME = 2;
        int EMP_RATE_PER_HOUR = 20;
        //Variables
        int empHrs = 0;
        int empWage = 0;
        Random random = new Random();
        //Computation
        int empCheck = random.Next(0, 3);
        if (empCheck == IS_PART_TIME)
        {
            empHrs = 4;
        }
        else if (empCheck == IS_FULL_TIME)
        {
            empHrs = 8;
        }
        else
        {
            empHrs = 0;
        }
        empWage = empHrs * EMP_RATE_PER_HOUR;
        Console.WriteLine("Emp Wage : " + empWage);
    }
}
```

BridgeLabz
Employability Delivered



Solving using Case Statement

Calculating Employee Wage Using Switch

```
class Program
{
    public const int IS_PART_TIME = 1;
    public const int IS_FULL_TIME = 2;
    public const int EMP_RATE_PER_HOUR = 20;
    0 references
    static void Main(string[] args)
    {
        //Variables
        int empHrs = 0;
        int empWage = 0;
        Random random = new Random();
        //Computation
        int empCheck = random.Next(0, 3);
        switch (empCheck)
        {
            case IS_PART_TIME:
                empHrs = 4;
                break;
            case IS_FULL_TIME:
                empHrs = 8;
                break;
            default:
                empHrs = 0;
                break;
        }
        empWage = empHrs * EMP_RATE_PER_HOUR;
        Console.WriteLine("Emp Wage : " + empWage);
    }
}
```

BridgeLabz
Employability Delivered

UC 5

Calculating Wages for a Month assuming 20 Working Days in a Month

Calculating Wages for a Month

```
class Program
{
    public const int IS_PART_TIME = 1;
    public const int IS_FULL_TIME = 2;
    public const int EMP_RATE_PER_HOUR = 20;
    public const int NUM_OF_WORKING_DAYS = 2;
    0 references
    static void Main(string[] args)
    {
        //Variables
        int empHrs = 0, empWage = 0, totalEmpWage=0;
        //Computation
        for(int day = 0; day < NUM_OF_WORKING_DAYS; day++)
        {
            Random random = new Random();
            int empCheck = random.Next(0, 3);
            switch (empCheck)
            {
                case IS_PART_TIME:
                    empHrs = 4;
                    break;
                case IS_FULL_TIME:
                    empHrs = 8;
                    break;
                default:
                    empHrs = 0;
                    break;
            }
            empWage = empHrs * EMP_RATE_PER_HOUR;
            totalEmpWage += empWage;
            Console.WriteLine("Emp Wage : " + empWage);
        }
        Console.WriteLine("Total Emp Wage : " + totalEmpWage);
    }
}
```

BridgeLabz
Employability Delivered

UC 6

Calculate Wages till a condition of total working hours of 100 or max days os 20 is reached for a month

Calculating Wages till Number of Working Days or Total Working Hours per month is Reached

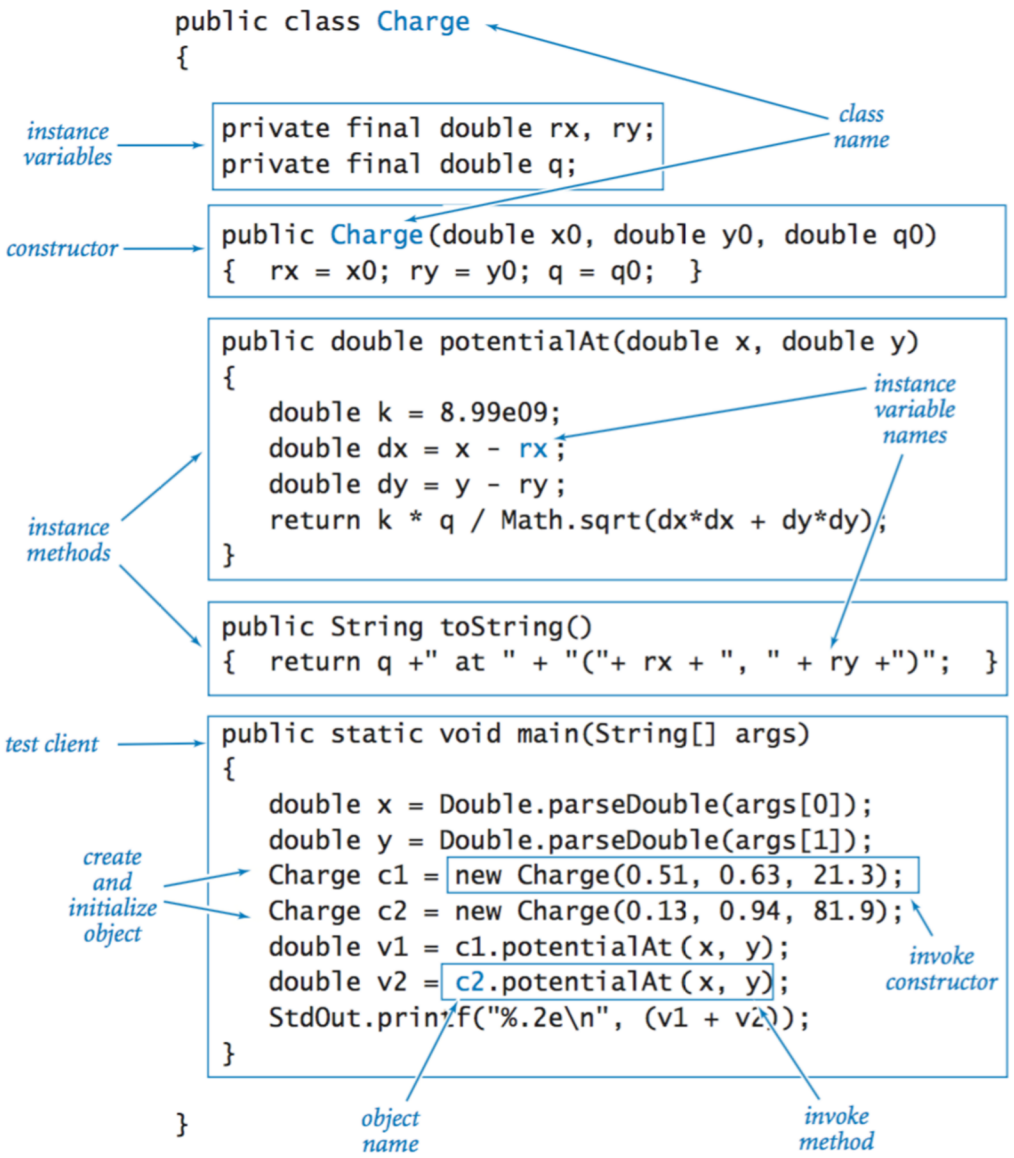
```
class Program
{
    public const int IS_PART_TIME = 1;
    public const int IS_FULL_TIME = 2;
    public const int EMP_RATE_PER_HOUR = 20;
    public const int NUM_OF_WORKING_DAYS = 2;
    public const int MAX_HRS_IN_MONTH = 10;
    0 references
    static void Main(string[] args)
    {
        //Variables
        int empHrs = 0, totalEmpHrs = 0, totalWorkingDays = 0;
        //Computation
        while(totalEmpHrs <= MAX_HRS_IN_MONTH && totalWorkingDays < NUM_OF_WORKING_DAYS)
        {
            totalWorkingDays++;
            Random random = new Random();
            int empCheck = random.Next(0, 3);
            switch (empCheck)
            {
                case IS_PART_TIME:
                    empHrs = 4;
                    break;
                case IS_FULL_TIME:
                    empHrs = 8;
                    break;
                default:
                    empHrs = 0;
                    break;
            }
            totalEmpHrs += empHrs;
            Console.WriteLine("Day#:" + totalWorkingDays + " Emp Hrs : " + empHrs);
        }
        int totalEmpWage = totalEmpHrs * EMP_RATE_PER_HOUR;
        Console.WriteLine("Total Emp Wage : " + totalEmpWage);
    }
}
```

BridgeLabz
Employability Delivered

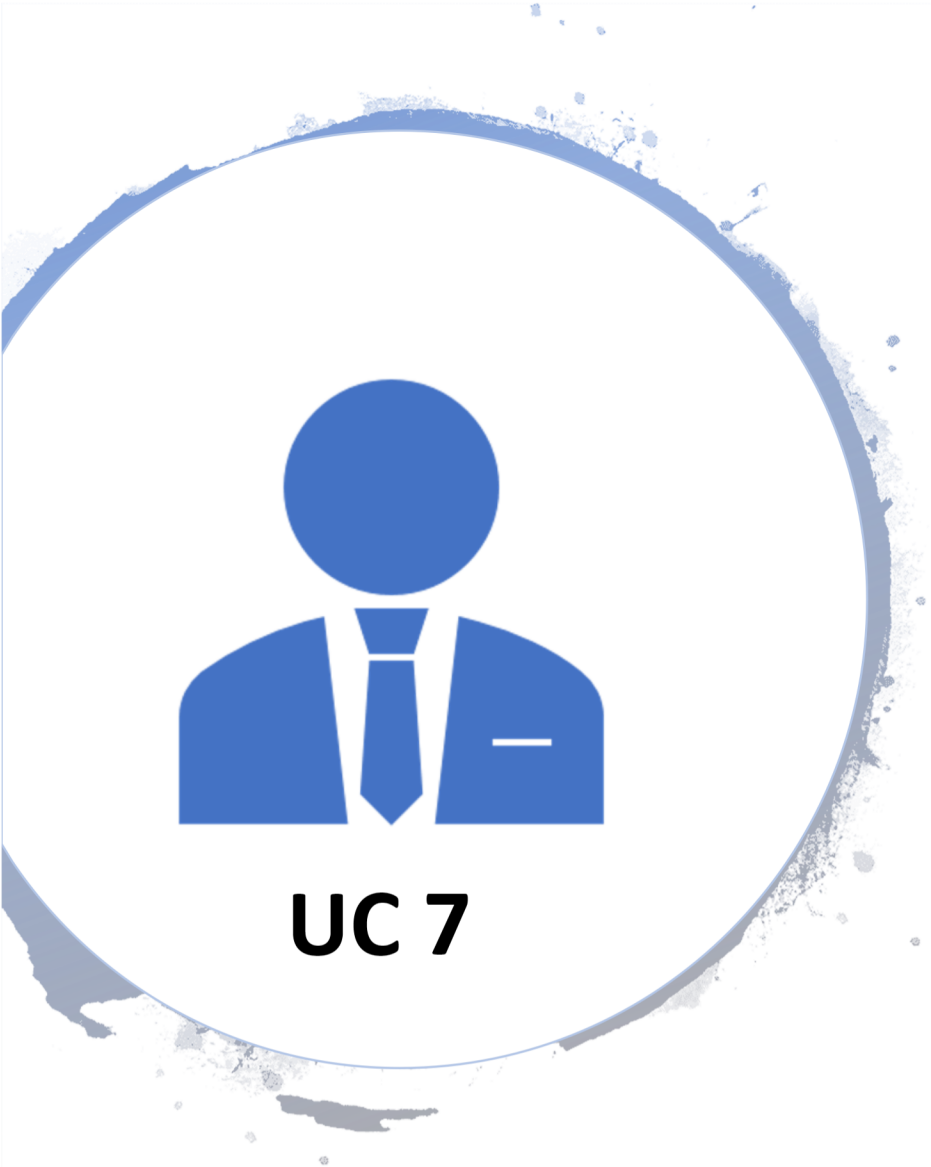
2. Class, Objects & Methods

- A **Class** can be considered as a blueprint using which you can create as many objects as you like.
- **Objects** have state and behaviour
- **Abstraction** is a process where you show only “relevant” data and “hide” unnecessary details of an object from the user.
- **Encapsulation** simply means binding object state(fields) and behaviour (methods) together. If you are creating class, you are doing encapsulation.
- **Association** establishes relationships between two Objects so as to enable Method Invocation

Class Specification



BridgeLabz
Employability Delivered



Refactor the Code to write a Class Method to Compute Employee Wage

Compute Employee Wage Using Class Methods

```
class Program
{
    public const int IS_PART_TIME = 1;
    public const int IS_FULL_TIME = 2;
    public const int EMP_RATE_PER_HOUR = 20;
    public const int NUM_OF_WORKING_DAYS = 2;
    public const int MAX_HRS_IN_MONTH = 10;
    1 reference
    public static int computeEmpWage()
    {
        //Variables
        int empHrs = 0, totalEmpHrs = 0, totalWorkingDays = 0;
        //Computation
        while (totalEmpHrs <= MAX_HRS_IN_MONTH && totalWorkingDays < NUM_OF_WORKING_DAYS)
        {
            totalWorkingDays++;
            Random random = new Random();
            int empCheck = random.Next(0, 3);
            switch (empCheck)
            {
                case IS_PART_TIME:
                    empHrs = 4;
                    break;
                case IS_FULL_TIME:
                    empHrs = 8;
                    break;
                default:
                    empHrs = 0;
                    break;
            }
            totalEmpHrs += empHrs;
            Console.WriteLine("Day#:" + totalWorkingDays + " Emp Hrs : " + empHrs);
        }
        int totalEmpWage = totalEmpHrs * EMP_RATE_PER_HOUR;
        Console.WriteLine("Total Emp Wage : " + totalEmpWage);
        return totalEmpWage;
    }
    0 references
    static void Main(string[] args)
    {
        computeEmpWage();
    }
}
```

UC 8

Ability to compute Employee Wage for multiple companies

Compute Employee Wage for Multiple Company in Procedural Way Using Class Methods

```
class Program
{
    public const int IS_PART_TIME = 1;
    public const int IS_FULL_TIME = 2;
    2 references
    public static int computeEmpWage(string company, int empRatePerHour, int numOfWorkingDays,int maxHoursPerMonth)
    {
        //Variables
        int empHrs = 0, totalEmpHrs = 0, totalWorkingDays = 0;
        //Computation
        while (totalEmpHrs <= maxHoursPerMonth && totalWorkingDays < numOfWorkingDays)
        {
            totalWorkingDays++;
            Random random = new Random();
            int empCheck = random.Next(0, 3);
            switch (empCheck)
            {
                case IS_PART_TIME:
                    empHrs = 4;
                    break;
                case IS_FULL_TIME:
                    empHrs = 8;
                    break;
                default:
                    empHrs = 0;
                    break;
            }
            totalEmpHrs += empHrs;
            Console.WriteLine("Day#:" + totalWorkingDays + " Emp Hrs : " + empHrs);
        }
        int totalEmpWage = totalEmpHrs * empRatePerHour;
        Console.WriteLine("Total Emp Wage for company : " + company + " is: " + totalEmpWage);
        return totalEmpWage;
    }
    0 references
    static void Main(string[] args)
    {
        computeEmpWage("DMart", 20, 2, 10);
        computeEmpWage("Reliance", 10, 4, 20);
    }
}
```

UC 9

Ability to save the Total Wage for Each Company

Compute Employee Wage and Save Total Wage by Company

```
0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        EmpWageBuilderObject dMart = new EmpWageBuilderObject("DMart", 20, 2, 10);
        EmpWageBuilderObject reliance = new EmpWageBuilderObject("Reliance", 10, 4, 20);
        dMart.computeEmpWage();
        Console.WriteLine(dMart.toString());
        reliance.computeEmpWage();
        Console.WriteLine(reliance.toString());
    }
}
```

```
public class EmpWageBuilderObject
{
    public const int IS_PART_TIME = 1;
    public const int IS_FULL_TIME = 2;

    private  string company;
    private  int empRatePerHour;
    private int numOfWorkingDays;
    private int maxHoursPerMonth;
    private int totalEmpWage;

    2 references
    public EmpWageBuilderObject(string company, int empRatePerHour, int numOfWorkingDays, int maxHoursPerMonth)
    {
        this.company = company;
        this.empRatePerHour = empRatePerHour;
        this.numOfWorkingDays = numOfWorkingDays;
        this.maxHoursPerMonth = maxHoursPerMonth;
    }
    2 references
    public  void computeEmpWage()
    {
        //Variables
        int empHrs = 0, totalEmpHrs = 0, totalWorkingDays = 0;
        //Computation
        while (totalEmpHrs <= this.maxHoursPerMonth && totalWorkingDays < this.numOfWorkingDays)
        {
            totalWorkingDays++;
            Random random = new Random();
            int empCheck = random.Next(0, 3);
            switch (empCheck)
            {
                case IS_PART_TIME:
                    empHrs = 4;
                    break;
                case IS_FULL_TIME:
                    empHrs = 8;
                    break;
                default:
                    empHrs = 0;
                    break;
            }
            totalEmpHrs += empHrs;
            Console.WriteLine("Day#:" + totalWorkingDays + " Emp Hrs : " + empHrs);
        }
        totalEmpWage = totalEmpHrs * this.empRatePerHour;
        Console.WriteLine("Total Emp Wage for company : " + company + " is: " + totalEmpWage);
    }

    2 references
    public string toString()
    {
        return "Total Emp Wage for company : " + this.company + " is: " + this.totalEmpWage;
    }
}
```

BridgeLabz
Employability Delivered

UC 10

Ability to manage Employee Wage of multiple companies

Support Employee Wage for Multiple Company using Interface approach

```
public class CompanyEmpWage
{
    public string company;
    public int empRatePerHour;
    public int numOfWorkingDays;
    public int maxHoursPerMonth;
    public int totalEmpWage;

    1 reference
    public CompanyEmpWage(string company, int empRatePerHour, int numOfWorkingDays, int maxHoursPerMonth)
    {
        this.company = company;
        this.empRatePerHour = empRatePerHour;
        this.numOfWorkingDays = numOfWorkingDays;
        this.maxHoursPerMonth = maxHoursPerMonth;
    }

    1 reference
    public void setTotalEmpWage(int totalEmpWage)
    {
        this.totalEmpWage = totalEmpWage;
    }

    1 reference
    public string toString()
    {
        return "Total Emp Wage for company : " + this.company + " is: " + this.totalEmpWage;
    }
}
```

```
0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        EmpWageBuilderArray empWageBuilder = new EmpWageBuilderArray();
        empWageBuilder.addCompanyEmpWage("DMart", 20, 2, 10);
        empWageBuilder.addCompanyEmpWage("Reliance", 10, 4, 20);
        empWageBuilder.computeEmpWage();
    }
}
```

```
public class EmpWageBuilderArray
{
    public const int IS_PART_TIME = 1;
    public const int IS_FULL_TIME = 2;

    private int numOfCompany = 0;
    private CompanyEmpWage[] companyEmpWageArray;

    1 reference
    public EmpWageBuilderArray()
    {
        this.companyEmpWageArray = new CompanyEmpWage[5];
    }

    2 references
    public  void addCompanyEmpWage(string company, int empRatePerHour, int numOfWorkingDays, int maxHoursPerMonth)
    {
        companyEmpWageArray[this.numOfCompany] = new CompanyEmpWage(company, empRatePerHour, numOfWorkingDays, maxHoursPerMonth);
        numOfCompany++;
    }

    1 reference
    public void computeEmpWage()
    {
        for (int i = 0; i < numOfCompany; i++)
        {
            companyEmpWageArray[i].setTotalEmpWage(this.computeEmpWage(this.companyEmpWageArray[i]));
            Console.WriteLine(this.companyEmpWageArray[i].toString());
        }
    }

    1 reference
    private int computeEmpWage(CompanyEmpWage companyEmpWage)
    {
        //Variables
        int empHrs = 0, totalEmpHrs = 0, totalWorkingDays = 0;
        //Computation
        while (totalEmpHrs <= companyEmpWage.maxHoursPerMonth && totalWorkingDays < companyEmpWage.numOfWorkingDays)
        {
            totalWorkingDays++;
            Random random = new Random();
            int empCheck = random.Next(0, 3);
            switch (empCheck)
            {
                case IS_PART_TIME:
                    empHrs = 4;
                    break;
                case IS_FULL_TIME:
                    empHrs = 8;
                    break;
                default:
                    empHrs = 0;
                    break;
            }
            totalEmpHrs += empHrs;
            Console.WriteLine("Day#:" + totalWorkingDays + " Emp Hrs : " + empHrs);
        }
        return totalEmpHrs * companyEmpWage.empRatePerHour;
    }
}
```

3. Key Concepts

1. Objects and Constructors
2. Static and final Keyword
3. Interface
4. Collection Library – ArrayList and HashMap
5. Compile time and Run time
6. Generics
7. super and this Keyword

UC 11

Ability to manage Employee Wage of multiple companies using API approach

UC 12

Refactor to have list of multiple companies to manage Employee Wage.

UC 13

Store the Daily Wage along with the Total Wage.

UC 14

Ability to get the Total Wage when queried by Company.

Final Solution

```
public interface IComputeEmpWage
{
    3 references
    public void addCompanyEmpWage(string company, int empRatePerHour, int numOfWorkingDays, int maxHoursPerMonth);
    2 references
    public void computeEmpWage();
    2 references
    public int getTotalWage(string company);
}
9 references
public class CompanyEmpWage
{
    public string company;
    public int empRatePerHour;
    public int numOfWorkingDays;
    public int maxHoursPerMonth;
    public int totalEmpWage;

    1 reference
    public CompanyEmpWage(string company, int empRatePerHour, int numOfWorkingDays, int maxHoursPerMonth)
    {
        this.company = company;
        this.empRatePerHour = empRatePerHour;
        this.numOfWorkingDays = numOfWorkingDays;
        this.maxHoursPerMonth = maxHoursPerMonth;
        this.totalEmpWage = 0;
    }

    1 reference
    public void setTotalEmpWage(int totalEmpWage)
    {
        this.totalEmpWage = totalEmpWage;
    }

    1 reference
    public string toString()
    {
        return "Total Emp Wage for company : " + this.company + " is: " + this.totalEmpWage;
    }
}
```

```
public class EmpWageBuilder: IComputeEmpWage
{
    public const int IS_PART_TIME = 1;
    public const int IS_FULL_TIME = 2;

    private LinkedList<CompanyEmpWage> companyEmpWageList;
    private Dictionary<string, CompanyEmpWage> companyToEmpWageMap;

    1 reference
    public EmpWageBuilder()
    {
        this.companyEmpWageList = new LinkedList<CompanyEmpWage>();
        this.companyToEmpWageMap = new Dictionary<string, CompanyEmpWage>();
    }

    3 references
    public  void addCompanyEmpWage(string company, int empRatePerHour, int numOfWorkingDays, int maxHoursPerMonth)
    {
        CompanyEmpWage companyEmpWage = new CompanyEmpWage(company, empRatePerHour, numOfWorkingDays, maxHoursPerMonth);
        this.companyEmpWageList.AddLast(companyEmpWage);
        this.companyToEmpWageMap.Add(company, companyEmpWage);
    }

    2 references
    public void computeEmpWage()
    {
        foreach (CompanyEmpWage companyEmpWage in  this.companyEmpWageList)
        {
            companyEmpWage.setTotalEmpWage(this.computeEmpWage(companyEmpWage));
            Console.WriteLine(companyEmpWage.toString());
        }
    }

    1 reference
    private int computeEmpWage(CompanyEmpWage companyEmpWage)...

    2 references
    public int getTotalWage(string company)
    {
        return this.companyToEmpWageMap[company].totalEmpWage;
    }
}

0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        EmpWageBuilder empWageBuilder = new EmpWageBuilder();
        empWageBuilder.addCompanyEmpWage("DMart", 20, 2, 10);
        empWageBuilder.addCompanyEmpWage("Reliance", 10, 4, 20);
        empWageBuilder.computeEmpWage();
        Console.WriteLine("Total Wage for DMart company : " + empWageBuilder.getTotalWage("DMart"));
    }
}
```

BridgeLabz

Employability Delivered

Thank You