

Taazaa Training

“Assignment - 5”

Topics:-

- **Day5A (Collection with Naming convention)**
- **Dictionary**
(Insert, Update, Delete and Search)
- **Hash Table**
(Insert, Update, Delete and Search)
- **Private Constructor**

Submitted by: -

Gurpreet Singh

Day5A (Collection with Naming convention)

Source Code:-

Recipe1.cs under Models folder

```
namespace Day5A.Models
{
    public class Recipe1//Pascal Case
    {
        public int step{get;set;}//camelcase
        public string rName{get;set;}//camelcase
    }
}
```

Recipe2cs.cs

```
using System.Collections;
using System.Collections.Generic;
using Day5A.Models;
namespace Day5A.Recipe
{
    public class Recipe2cs //pascal case
    {
        public ArrayList mangoShake()//camel case
        {
            ArrayList arrayList=new ArrayList();//camel case
            arrayList.Add("Mango");
            arrayList.Add("Ice Cubes");
            arrayList.Add("Milk");
            arrayList.Add("Sugar");
            return arrayList;
        }

        public List<Recipe1> bananaShake()//camel case
        {
            List<Recipe1> Obj=new List<Recipe1>();
            Obj.Add(new Recipe1{
                step=1,//camel case
                rName="Banana"//camel case
            });
            Obj.Add(new Recipe1{
                step=2,//camel case
```

```

        rName="Milk">//camel case
    });
    Obj.Add(new Recipe1{
        step=3,//camel case
        rName="Sugar">//camel case
    });
    Obj.Add(new Recipe1{
        step=4,//camel case
        rName="Ice Cubes">//camel case
    });

    return Obj;
}

}
}

```

Progarm.cs

```

using System;
using System.Collections;
using Day5A.Recipe;

namespace Day5A
{
    class Program//pascal case
    {
        static void Main()
        {
            Console.WriteLine("for Mango Shake");
            var obj=new Recipe2cs();//camel case
            var arrayList=obj.mangoShake();//camel case
            int count=arrayList.Count;//camel case
            for(int i=0;i<count;i++)
            {
                Console.WriteLine(arrayList[i]);
            }
            Console.WriteLine("for Banana Shake");
            var list=obj.bananaShake();//camel case
            count=list.Count;//camel case
            for(int i=0;i<count;i++)
            {

```

```

        Console.WriteLine(list[i].step + " " + list[i].rName); //camel case
    }
}
}
}
}

```

Output:-

The screenshot displays the Visual Studio Code interface with the following components:

- EXPLORER:** Shows the project structure with files like `Recipe1.cs`, `Recipe2.cs`, and `Program.cs`.
- EDITOR:** Displays the code for `Program.cs`, which includes:
 - Imports: `using System;`, `using System.Collections;`, and `using Day5A.Recipe;`
 - Namespace: `namespace Day5A`
 - Class: `class Program` with a `pascal case` comment.
 - Method: `static void Main()` containing the logic to create a `Recipe2cs` object, call `mangoShake()`, and iterate through the resulting list to print each item's step and name in camel case.
- TERMINAL:** Shows the command `C:\Users\hnp\Desktop\Assignment 5\Day5A>dotnet run` and its output:


```

for Mango Shake
Mango
Ice Cubes
Milk
Sugar
for Banana Shake
1 Banana
2 Milk
3 Sugar
4 Ice Cubes

```

Dictionary

(Insert, Update, Delete and Search)

Theory:-

The **Dictionary<TKey, TValue>** is a generic collection that stores key-value pairs in no particular order.

- It stores key-value pairs.
- Comes under **System.Collections.Generic** namespace.
- Implements **IDictionary<TKey, TValue>** interface.
- Keys must be unique and cannot be null.
- Values can be null or duplicate.
- Values can be accessed by passing associated key in the indexer e.g. **myDictionary[key]**
- Elements are stored as **KeyValuePair<TKey, TValue>** objects.

Source Code:-

```
using System;
using System.Collections.Generic;

namespace dictionaryProgram
{
    class Program //implementation of all operations of dictionary
    {
        static void Main(string[] args)
        {
            //declaration of Dictionary
            var dt=new Dictionary<string,object>(); //here key is string
            type and value is of object type

            //Insertion operation in Dictionary
            //inserting value in dt
            dt.Add("Eid",101);
            dt.Add("Ename","Gurpreet Singh");
            dt.Add("Ephno",9717983635);
            dt.Add("Esalary",1000000.00);

            //update operation in Dictionary
            //update Ephno value
```

```

        Console.WriteLine("ephno old value :- "+dt["Ephno"]);
        dt["Ephno"]=7827656364;
        Console.WriteLine("ephno new value :- "+dt["Ephno"]);

        //delete operation in Dictionary
        int count=dt.Count;
        Console.WriteLine("old dt contains "+count+" elements");
        dt.Remove("Ephno");
        count=dt.Count;
        Console.WriteLine("new dt contains "+count+" elements");

        //search operation in Dictionary
        var val="Gurpreet Singh";
        if(dt.ContainsValue(val))
        {
            Console.WriteLine(val+"is found in dt");
        }
    }
}

```

Output:-

The screenshot shows the Visual Studio Code interface with the C# code from the previous block. The terminal at the bottom displays the output of the program:

```

C:\Users\vip\Desktop\Assignment 5\dictionaryProgram>dotnet run
ephno old value :- 9717983635
ephno new value :- 7827656364
old dt contains 4 elements
new dt contains 3 elements
Gurpreet Singhis found in dt

```

The status bar at the bottom indicates the file is 'dictionaryProgram.csproj', the cursor is at line 32, column 28, and the encoding is UTF-8 with BOM.

Hash Table

(Insert, Update, Delete and Search)

Theory:-

The **Hashtable** is a non-generic collection that stores key-value pairs, similar to generic Dictionary<TKey, TValue> collection.

- **Hashtable** stores key-value pairs.
- Comes under **System.Collection** namespace.
- Implements **IDictionary** interface.
- Keys must be unique and cannot be null.
- Values can be null or duplicate.
- Values can be accessed by passing associated key in the indexer e.g. **myHashtable[key]**
- Elements are stored as **DictionaryEntry** objects.

Source Code:-

```
using System;
using System.Collections;

namespace hashTableProgram
{
    class Program
    {
        static void Main(string[] args)
        {
            // declaring hashtable
            Hashtable ht = new Hashtable();

            // insertion in hashtable
            ht.Add(1, "Gurpreet Singh");
            ht.Add(2, "Sandeep sir");
            ht.Add(3, "Ramesh");
            ht.Add(4, 4.00);

            //update operation in hashtable
            Console.WriteLine("old value of 4 key : "+ht[4]);
            ht[4]=400;
            Console.WriteLine("new value of 4 key : "+ht[4]);

            //deletion operation in hashtable
            int n=ht.Count;
```

```

        Console.WriteLine("old ht contains "+n+" values");
        ht.Remove(4);
        n=ht.Count;
        Console.WriteLine("new ht contains "+n+" values");

        //search operation in hashtable
        if(ht.ContainsValue("Gurpreet Singh"))
        {
            Console.WriteLine("ht contains Gurpreet Singh value");
        }
    }
}
}

```

Output:-

The screenshot displays the Visual Studio Code interface. The editor window shows the source code for `Program.cs` in the `hashTableProgram` namespace. The code includes a `Program` class with a `Main` method that performs operations on a `Hashtable`. The terminal window at the bottom shows the output of the program after running the command `dotnet run`.

Source Code:

```

1 using System;
2 using System.Collections;
3
4 namespace hashTableProgram
5 {
6     class Program
7     {
8         static void Main(string[] args)
9         {
10             // declaring hashtable
11             Hashtable ht = new Hashtable();
12
13             // insertion in hashtable
14             ht.Add(1,"Gurpreet Singh");
15             ht.Add(2,"Sandeep sir");
16             ht.Add(3,"Ramesh");
17             ht.Add(4,4.00);
18
19             //update operation in hashtable
20             Console.WriteLine("old value of 4 kev : "+ht[4]);

```

Terminal Output:

```

C:\Users\Vip\Desktop\Assignment 5\hashTableProgram>dotnet run
old value of 4 key : 4
new value of 4 key : 400
old ht contains 4 values
new ht contains 3 values
ht contains Gurpreet Singh value

```


Private Constructor

Theory:-

- Private constructor is a special instance constructor which is used in a class that contains static member only.
- If a class has one or more private constructor and no public constructor then other classes are not allowed to create instance of this class; this means you can neither create the object of the class nor can it be inherited by other classes.
- The main purpose of creating private constructor is to restrict the class from being instantiated when it contains every member as static.

Source Code:-

```
using System;

namespace privateConstructorProgram
{
    class Program
    {
        public static string name;
        public static int num;
        // Creating private Constructor
        // using private keyword
        private Program()
        {
            Console.WriteLine("Welcome to Private Constructor");
        }
        // Default Constructor
        // with parameters
        public Program(string a, int b) {

            name = a;
            num = b;
        }
        public static void Main() {

            // This line raises error because
            // the constructor is inaccessible
            // Program obj1 = new Program();

            // Here, the only default
            // constructor will invoke
            Program obj2 = new Program("Gurpreet", 101);
        }
    }
}
```

```

// Here, the data members of Geeks
// class are directly accessed
// because they are static members
// and static members are accessed
// directly with the class name
Console.WriteLine(Program.name + ", " + Program.num);
}

}
}

```

Output:-

The screenshot displays the Visual Studio Code interface. The Explorer pane on the left shows the project structure with 'Program.cs' open. The main editor shows the following C# code:

```

1 using System;
2
3 namespace privateConstructorProgram
4 {
5     4 references
6     class Program
7     {
8         2 references
9         public static string name;
10        2 references
11        public static int num;
12        // Creating private Constructor
13        // using private keyword
14        0 references
15        private Program()
16        {
17            Console.WriteLine("Welcome to Private Constructor");
18        }
19        // Default Constructor
20        // with parameters
21        1 reference
22        public Program(string a, int b) {

```

The bottom panel shows the TERMINAL output:

```

Microsoft Windows [Version 10.0.19042.1165]
(c) Microsoft Corporation. All rights reserved.

C:\Users\hp\Desktop\Assignment 5\privateConstructorProgram>dotnet run
Gurpreet, 101

C:\Users\hp\Desktop\Assignment 5\privateConstructorProgram>

```

The status bar at the bottom indicates the file is 'Ln 2, Col 1', uses 'UTF-8 with BOM' encoding, and the system clock shows 10:30 PM on 18-08-2021.