DotNet Framework

Compilation in DotNet happens in 2 steps

Functions of CLR>

We can write program using any Editor

Visual Studio : IDE

Statements that we write in program

Methods :

Arrays :

**2th Feb :**

What are the advantages of Arrays :

1.No need to declare multiple elements of same type

What are the disadvantages of Arrays :

1. Searching is Sequential
2. **Only one type of element in the array : Its not disadvantage, ITS ADVANTAGE**
3. Fixed size
4. Wastage of Memory

Int[] num = new int[10];

1 2 3

1. Time consuming while performing insertion & deletion, because it requires lots of reshuffling of elements up or down

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 5 | 6 | 8 | 9 | 19 | 89 | 7 |  |  |

**To remove /overcome these limitations of Arrays, we use Collections**

Collections : Collections are structures which store elements . But the type of elements could be of different types.

Different type of collections are

1. **ArrayList : Insertion / Deletion can be done anywhere**
2. **Stack : It follows LIFO , Insertion / Deletion are done at end**
3. **Queue : It follows FIFO Insertion is done at end , Deletion is done in beginning**
4. **HashSet : The values are stored as key,value pair**

These all classes are present in System.Collections namespace

What are the advantages of Collections:

1.Collections are dynamic in size , no wastage of Memory

2. Insertion / Deletion is easier

What are the disadvantages of Collections:

1. Elements are not type safe , they are not of same type, which will take time

Insertion in collection is not a problem, problem is while retrieving the elements, it will take time

1. Elements are stored in object form in collections

(BOXING / UNBOXIG HAPPENS)

Boxing : Converting varaibe from value type to reference type

**--------------------------------------------------**

using System;

using System.Collections;

class Program

{

static void Main()

{

ArrayList list = new ArrayList();

list.Add(1);

list.Add(2);

list.Add(3);

foreach(int temp in list)

Console.WriteLine(temp);

}

}

**--------------------------------------------------**

using System;

using System.Collections;

class Program

{

static void Main()

{

ArrayList list = new ArrayList();

list.Add(1);

list.Add(2);

list.Add(3);

foreach(int temp in list)

Console.WriteLine(temp);

list.Insert(0, 100);

list.Insert(2, 200);

Console.WriteLine("After Insertion, elements are ");

foreach (int temp in list)

Console.WriteLine(temp);

list.RemoveAt(0);

list.Remove(200);

Console.WriteLine("After Deletion, elements are ");

foreach (int temp in list)

Console.WriteLine(temp);

}

}

-----------------------------------------------

Stack

using System;

using System.Collections;

class Program

{

static void Main()

{

ArrayList list = new ArrayList();

list.Add(1);

list.Add(2);

list.Add(3);

list.Add("ajay");

list.Add("deepak");

list.Add(90.89);

foreach(var temp in list)

Console.WriteLine(temp);

list.Insert(0, 100);

list.Insert(2, 200);

Console.WriteLine("After Insertion, elements are ");

foreach (var temp in list)

Console.WriteLine(temp);

list.RemoveAt(0);

list.Remove(200);

Console.WriteLine("After Deletion, elements are ");

foreach (var temp in list)

Console.WriteLine(temp);

}

}

Queue

using System;

using System.Collections;

class Program

{

static void Main()

{

//ArrayList list = new ArrayList();

//list.Add(1);

//list.Add(2);

//list.Add(3);

//list.Add("ajay");

//list.Add("deepak");

//list.Add(90.89);

//list.Add(100);

//foreach(var temp in list)

// Console.WriteLine(temp);

//list.Insert(0, 100);

//list.Insert(2, 200);

//Console.WriteLine("After Insertion, elements are ");

//foreach (var temp in list)

// Console.WriteLine(temp);

//list.RemoveAt(0);

//list.Remove(200);

//Console.WriteLine("After Deletion, elements are ");

//foreach (var temp in list)

// Console.WriteLine(temp);

Stack stack = new Stack();

stack.Push(1);

stack.Push(2);

stack.Push(3);

stack.Push("Ajay");

Console.WriteLine("Elements of Stack are");

foreach (var temp in stack)

Console.WriteLine(temp);

stack.Pop();

Console.WriteLine("Elements of Stack after Deletion are");

foreach (var temp in stack)

Console.WriteLine(temp);

Queue queue = new Queue();

queue.Enqueue(1);

queue.Enqueue(2);

queue.Enqueue(3);

queue.Enqueue("Ajay");

Console.WriteLine("Elements of Queue are");

foreach (var temp in queue)

Console.WriteLine(temp);

queue.Dequeue();

Console.WriteLine("Elements of Queue after Deletion are");

foreach (var temp in queue)

Console.WriteLine(temp);

}

}

Hashtable

Hashtable hashtable = new Hashtable();

hashtable[1] = 90;

hashtable[2] = 98;

hashtable[3] = 87;

hashtable["Ajay"] = "Delhi";

foreach(var temp in hashtable.Keys)

{

Console.WriteLine(hashtable[temp]);

}

Console.WriteLine(hashtable[3]);

Advantages of Arrays:

1.No need to declare multiple elements of same type

2. Type Safe, elements are of same type

Advantages of Collections:

1.Collections are dynamic in size , no wastage of Memory

2. Insertion / Deletion is easier

We will combine benefits of Arrays & Collections , we get **Generic Collections**

**Generic Collections are type safe**

* **List<int>**
* **Stack<int>**
* **Queue<string>**
* **Dictionary<int,int>**

**These classes are present in** System.Collections.Generic namespace

using System;

using System.Collections.Generic;

namespace ConsoleApp2

{

class GenericCollectionsDemo

{

static void Main()

{

// ArrayList list = new ArrayList();

List<int> list = new List<int>();

list.Add(1);

list.Add(2);

list.Add(3);

foreach (int temp in list)

Console.WriteLine(temp);

Stack<string> stack = new Stack<string>();

stack.Push("Ajay");

Queue<float> queue = new Queue<float>();

queue.Enqueue(20.89f);

Dictionary<int, int> dictionary = new Dictionary<int, int>();

dictionary[1] = 89;

dictionary[2] = 89;

dictionary[3] = 89;

dictionary[4] = 89;

}

}

}

using System;

using System.Collections.Generic;

namespace ConsoleApp2

{

class GenericCollectionsDemo

{

static void Main()

{

// ArrayList list = new ArrayList();

List<int> list = new List<int>();

list.Add(1);

list.Add(2);

list.Add(3);

foreach (int temp in list)

Console.WriteLine(temp);

Stack<string> stack = new Stack<string>();

stack.Push("Ajay");

Queue<float> queue = new Queue<float>();

queue.Enqueue(20.89f);

Dictionary<int, string> dictionary = new Dictionary<int, string>();

dictionary[1] = "89";

dictionary[2] = "89";

dictionary[3] = "89";

dictionary[4] = "89";

}

}

}

Concepts of OOPS

1. Class
2. Object
3. Encapsulation
4. Abstraction
5. Polymorphism
6. Inheritance

1.Class : Blueprint based on which multiple objects are created

User defined type

Data type 2 types

1.Primitive / inbuilt 2. User defined

1. Primitive / inbuilt > int char float decimal boolen

2. User defined > class struct enum

2. Object : Instance / Variable of a class

What does a data type indicates

Amount of Memory

Range , values that can be stored

What can we store

What operations are allowed

**Int > We can only store numbers , range**

**We can perform + - / \* max min**

**Amount of Memory**

Class is a user defined

Class Student

{

Int rn ;

String name;

Void GetDetails()

{

}

Void DisplayDetails()

{

}

}

------------------------------------

Student student = new Student();

Int x;

All the types get their default value

Value type variable will get their default value

Int > 0

Boolean = false

Reference type variables they get initialized to null by default;

Object : It is an instance of a class

Student student = new Student();

New keyword does 2 things

1. Allocates Memory on Heap
2. It calls / invokes default constructor

default constructor : It will initialize variables to their default values

rn = 0

name = null;

3.Encapsulation : Combining members of a class . hiding the details which are not needed.

4.Abstraction : Showing the details which are needed.

How do we achieve them , by using Access Specifiers

1. Private
2. Public
3. Protected
4. Internal
5. Internal protected

By default , access specifier of class is internal

By default , access specifier of class members is private

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp2

{

class Student

{

int rn;

string name;

public void GetDetails()

{

Console.WriteLine("Enter RollNo");

rn = Convert.ToByte(Console.ReadLine());

Console.WriteLine("Enter Name");

name = Console.ReadLine();

}

public void DisplayDetails()

{

Console.WriteLine("RollNo is " + rn);

Console.WriteLine("Name is " + name);

}

}

class Program

{

static void Main()

{

Student student = new Student();

student.GetDetails();

student.DisplayDetails();

}

}

}