**Advantages of Arrays** :

* All the elements share common name
* We can access elements thru their position(index)
* The elements are stored in a contiguous location
* No need to declare different elements
* Similar data type, arrays are type-safe

**Disadvantages of Arrays**

* Size is fixed , memory is static

Int[] num = new int[20];

* Insertion and deletion of elements is time consuming because it requires lots of reshuffling of elements

Collection

Advantages

* Size is dynamic , memory is dynamic which means memory is not wasted
* Insertion and deletion of elements is easy. We can call inbuilt methods to do these operations

ArrayList list = new ArrayList();

List.Add(10);

Disadvantages of Collections

1. Elements are not of same type , Collections are not type-safe

// int[] num = new int[10];

ArrayList list = new ArrayList();

list.Add(1); // Boxing is done

list.Add(2);

list.Add("Deepak");

list.Add(100);

list.Add(1.9);

Console.WriteLine("Elements are ");

foreach(var temp in list) // Unboxing is done

{

Console.WriteLine(temp);

}

Int x;

String name;

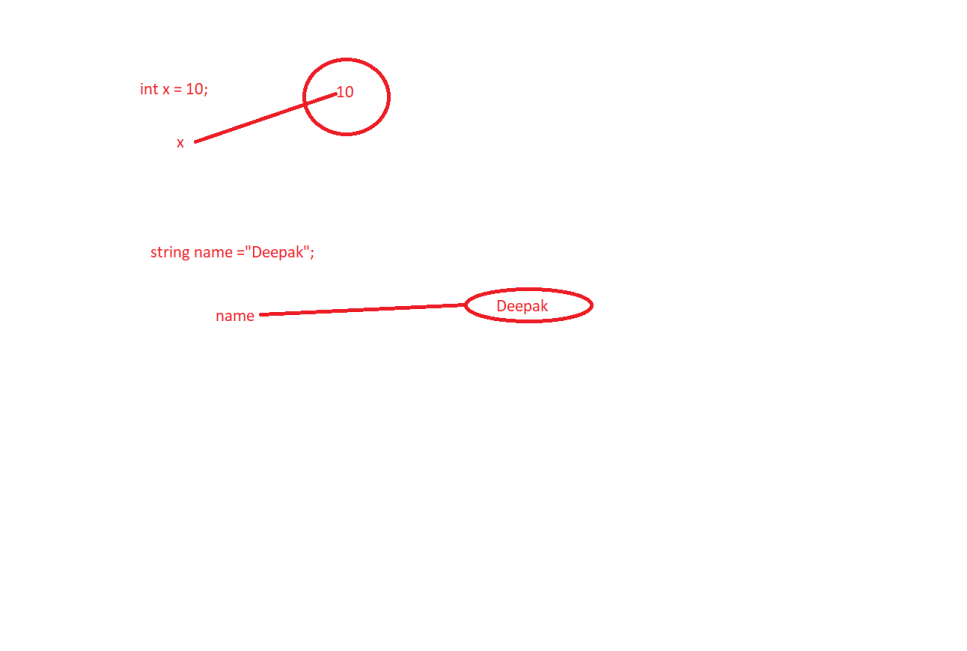
In RAM, we have 2 structures > Stack Heap

Variables of how many types

* Value Type : which stores the value . int , float , Boolean, char, struct, enum
* Reference Type : which stores the reference of the variable , string , class

When we declare varibale , they get memory and are stored somewhere in memory

Value type varibales are stored in stack

Reference type variables are stored in heap

Value type variables are stored in stack Reference type variables are stored in heap

Int x= 40

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
| + |
| 40 |
| 10  Deepak |
| 90 |

String name = “Deepak”

Garbage Collection : Removing unused memory blocks

Boxing > Converting value type variable to reference type

Unboxing > Converting reference type variable to value type

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FirstProject

{

class BoxingUnboxingDemo

{

static void Main()

{

int x = 20;

// Boxing

Object obj = x;

// Unboxing

int y = (int)obj;

}

}

}

**for vs foreach**

for loop

int[] num = new int[10];

for(int i=0;i<num.Length;i++)

{

Console.Write(num[i]);

}

// foreach

int[] num = new int[10];

foreach(int temp in num)

{

Console.Write(temp);

}

Out of these 2 loops, for loop is faster

* Because in foreach , it may require boxing unboxing sometimes
* Purpose of foreach loop is to only get information , it does not allow to change something in the loop

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FirstProject

{

class BoxingUnboxingDemo

{

static void Main()

{

int x = 20;

// Boxing

Object obj = x;

// Unboxing

int y = (int)obj;

int[] num = new int[] { 1, 2, 3, 4 };

Console.WriteLine("Using For loop");

for (int i = 0; i < num.Length; i++)

{

num[i]++;

Console.WriteLine(num[i]);

}

Console.WriteLine("Using Foreach loop");

foreach (int temp in num)

{

// temp++; // It will give compile time error

Console.WriteLine(temp);

}

}

}

}

**We take advantages of Arrays & Collections**

Arrays are type-safe, elements are of same type

Collections are dynamic , insertion/ deletion is easier

When we combine both of their advantages , We get GENERIC COLLECTIONS

Generic collections are collection only but they are type-safe , which means they can store elements of same type

**Classes for generic collections are present in System.Collections.Generic;**

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FirstProject

{

class GenericCollectionsDemo

{

static void Main()

{

// ArrayList list = new ArrayList();

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

list.Add(1);

list.Add(2);

list.Add(100);

foreach(int temp in list)

Console.WriteLine(temp);

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

queue.Enqueue("Ajay");

queue.Enqueue("Deepak");

foreach (string temp in queue)

Console.WriteLine(temp);

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

stack.Push(10.9f);

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

marks["Ajay"] = 90;

marks["Deeapk"] = 89;

}

}

}