// Oops

// Class

// Object

//Encapsulation

//Abstraction

// Inheritance

//Polymorphism

// Class > It is a user defined type

// Data type indicates

Int char float string can store only numbers > What data can be stored

What functions we can perform on variable of that type

What does a type indicate

Int >

enum struct

class > user defined type, blueprint for several objects

class <classname>

{

Members

Variables

methods

}

Int > What we can store . only numbers

What we can perform > .Addition subtraction , functions like these we can perform

Student > rn , name , marks , batch

What we can perform > .GetDetails DisplayDetails

Int num; num is the variable

Student x; Here x is a variable of class / instance of a class , object

Enacapsulation : Hiding the details which are not needed

Abstrction : Display the details which are required

We achieve them by using Access Specifiers

Private, public, protected, internal, internal protected

By default, class is internal , available in the assembly

By default , class members are private

student s = new student();

new keyword will do two things

1. allocate memory for the object from heap
2. it calls / invokes constructor (default constructor)

By default, in a class , default constructor is there

using System;

class student

{

int rn;

string name;

string batch;

int marks;

public void GetDetails()

{

Console.WriteLine("Enter RollNo");

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

Console.WriteLine("Enter Name");

name = Console.ReadLine();

Console.WriteLine("Enter Batch");

batch = Console.ReadLine();

Console.WriteLine("Enter Marks");

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

}

public void DisplayDetails()

{

Console.WriteLine("Roll No is " + rn);

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

Console.WriteLine("Batch is " +batch);

Console.WriteLine("Marks are " + marks);

}

}

class Program

{

static void Main()

{

student s = new student();

s.GetDetails();

s.DisplayDetails();

}

}

using System;

class student

{

int rn;

string name;

string batch;

int marks;

public void GetDetails()

{

Console.WriteLine("Enter RollNo");

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

Console.WriteLine("Enter Name");

name = Console.ReadLine();

Console.WriteLine("Enter Batch");

batch = Console.ReadLine();

Console.WriteLine("Enter Marks");

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

}

public void DisplayDetails()

{

Console.WriteLine("Roll No is " + rn);

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

Console.WriteLine("Batch is " +batch);

Console.WriteLine("Marks are " + marks);

}

}

class Program

{

static void Main()

{

//int x, y;

student s1 = new student();

s1.GetDetails();

s1.DisplayDetails();

student s2 = new student();

s2.GetDetails();

s2.DisplayDetails();

}

}

Array of Objects

using System;

class student

{

int rn;

string name;

string batch;

int marks;

public void GetDetails()

{

Console.WriteLine("Enter RollNo");

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

Console.WriteLine("Enter Name");

name = Console.ReadLine();

Console.WriteLine("Enter Batch");

batch = Console.ReadLine();

Console.WriteLine("Enter Marks");

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

}

public void DisplayDetails()

{

Console.WriteLine("Roll No is " + rn);

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

Console.WriteLine("Batch is " +batch);

Console.WriteLine("Marks are " + marks);

}

}

class Program

{

static void Main()

{

//int x, y , a, b, c ;

//student s1 = new student();

//s1.GetDetails();

//s1.DisplayDetails();

//student s2 = new student();

//s2.GetDetails();

//s2.DisplayDetails();

// Array of Objects

student[] student = new student[10];

for(int i=0;i<10;i++)

{

student[i] = new student();

student[i].GetDetails();

student[i].DisplayDetails();

}

}

}

Disadvantages/Limitations of Arrays

1. **We can not enter elements of different type : Advantage** , Type safe
2. Fixed Memory

Num[10]

Memory is allocated at compile time, memory can be wasted

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 5 | 6 | 8 | 90 | 7 |  |  |  |  |

1. we cannot change array size dynamically
2. insertion / deletion is time consuming because it requires lots of element reshuffling

Q. WAP to insert some elements in Array

Q. WAP to delete some elements in Array

TO solve problems posed by Arrays , we use collection

Collection is a structure in which we can store elements of different types

To use collections , we have to use System.Collections namespace

Collections could be

* ArrayList > insertion / deletion can be done anywhere
* Stack > Follows LIFO , insertion / deletion both are done at end
* Queue > Follows FIFO , insertion at end , deletion from beginning
* HashSet > Here we use Key, value
* SortedList

// ArrayList

using System;

using System.Collections;

class CollectionsDemo

{

static void Main()

{

// Array

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

// Fixed Memory is solved

ArrayList list = new ArrayList();

list.Add(10);

list.Add(20);

list.Add(30);

for(int i=0;i<list.Count;i++)

Console.WriteLine(list[i]);

list.Insert(0, 100);

Console.WriteLine("After inserting at position 0");

for (int i = 0; i < list.Count; i++)

Console.WriteLine(list[i]);

list.Remove(10);

Console.WriteLine("After Deleting 10");

for (int i = 0; i < list.Count; i++)

Console.WriteLine(list[i]);

}

}

To iterate thru collection / or an array, we also use foreach loop

foreach(int x in stack)

Console.WriteLine(x);

Foreach( type rangevaribale in collection/array name)

{

Console.Write(rangevaribale);

}

Stack

using System;

using System.Collections;

class CollectionsDemo

{

static void Main()

{

// Array

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

// Fixed Memory is solved

// ArrayList list = new ArrayList();

// list.Add(10);

// list.Add(20);

// list.Add(30);

//for(int i=0;i<list.Count;i++)

// Console.WriteLine(list[i]);

// list.Insert(0, 100);

// Console.WriteLine("After inserting at position 0");

// for (int i = 0; i < list.Count; i++)

// Console.WriteLine(list[i]);

// list.Remove(10);

// Console.WriteLine("After Deleting 10");

// for (int i = 0; i < list.Count; i++)

// Console.WriteLine(list[i]);

Stack stack = new Stack();

stack.Push(10);

stack.Push(20);

stack.Push(30);

stack.Push(40);

foreach (int x in stack)

Console.WriteLine(x);

stack.Pop();

Console.WriteLine("Elements after Deletion from stack");

foreach (int x in stack)

Console.WriteLine(x);

}

}

Queue

using System;

using System.Collections;

class CollectionsDemo

{

static void Main()

{

// Array

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

// Fixed Memory is solved

// ArrayList list = new ArrayList();

// list.Add(10);

// list.Add(20);

// list.Add(30);

//for(int i=0;i<list.Count;i++)

// Console.WriteLine(list[i]);

// list.Insert(0, 100);

// Console.WriteLine("After inserting at position 0");

// for (int i = 0; i < list.Count; i++)

// Console.WriteLine(list[i]);

// list.Remove(10);

// Console.WriteLine("After Deleting 10");

// for (int i = 0; i < list.Count; i++)

// Console.WriteLine(list[i]);

Queue queue = new Queue();

queue.Enqueue(10);

queue.Enqueue(20);

queue.Enqueue(30);

queue.Enqueue(40);

foreach (int x in queue)

Console.WriteLine(x);

queue.Dequeue();

Console.WriteLine("Elements after Deletion from queue");

foreach (int x in queue)

Console.WriteLine(x);

}

}

Boxing / Unboxing

Boxing > Convert from value type to reference type

UnBoxing > Convert from reference type to value type

using System;

using System.Collections;

class CollectionsDemo

{

static void Main()

{

// Array

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

// Fixed Memory is solved

ArrayList list = new ArrayList();

list.Add(10);

list.Add(20);

list.Add(30);

list.Add("Ajay");

list.Add(90.98);

//list.Add()

foreach(var x in list)

Console.WriteLine(x);

list.Insert(0, 100);

Console.WriteLine("After inserting at position 0");

foreach (var x in list)

Console.WriteLine(x);

list.Remove(10);

Console.WriteLine("After Deleting 10");

foreach (var x in list)

Console.WriteLine(x);

Queue queue = new Queue();

queue.Enqueue(10);

queue.Enqueue(20);

queue.Enqueue(30);

queue.Enqueue(40);

queue.Enqueue("Ajay");

foreach (var x in queue)

Console.WriteLine(x);

queue.Dequeue();

Console.WriteLine("Elements after Deletion from queue");

foreach (var x in queue)

Console.WriteLine(x);

}

}

Arrays : Advantage : Elements are of same type

Collections : Dynamic Size, memory is not wasted

Insertion / Deletion is not time consuming

When we combine advantages of both of them , we get Generic Collections

Generic means , type is known at compile time

ArrayList list = new ArrayList();

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

using System;

using System.Collections.Generic;

class GenericCollectionsDemo

{

static void Main()

{

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

list.Add(1);

list.Add(2);

list.Add(3);

list.Add(4);

foreach(int x in list)

Console.WriteLine(x);

list.Insert(0, 100);

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

stack.Push("Ajay");

stack.Push("Ajay");

stack.Push("Ajay");

stack.Push("Ajay");

foreach(string x in stack)

Console.WriteLine(x);

}

}

Hashtable hashtable = new Hashtable();

hashtable[0] = 90;

hashtable[2] = 89;

hashtable[3] = 78;

for (int i = 0; i <= hashtable.Keys.Count; i++)

Console.WriteLine(hashtable[i]);

//foreach (var x in hashtable)

// Console.WriteLine(x);

hashtable["Ajay"] = 90;

hashtable["Deepak"] = "Delhi";

Console.WriteLine(hashtable["Ajay"]);

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

keyValues[9] = 90;