.net framework

.net framework = tools + resources

which are useful for creating,testing,debugging,and deploying wide verities of application

using .net we can create different types of applications

ASP.Net Web applications

Windows applications

Web services

Windows services

Console applications

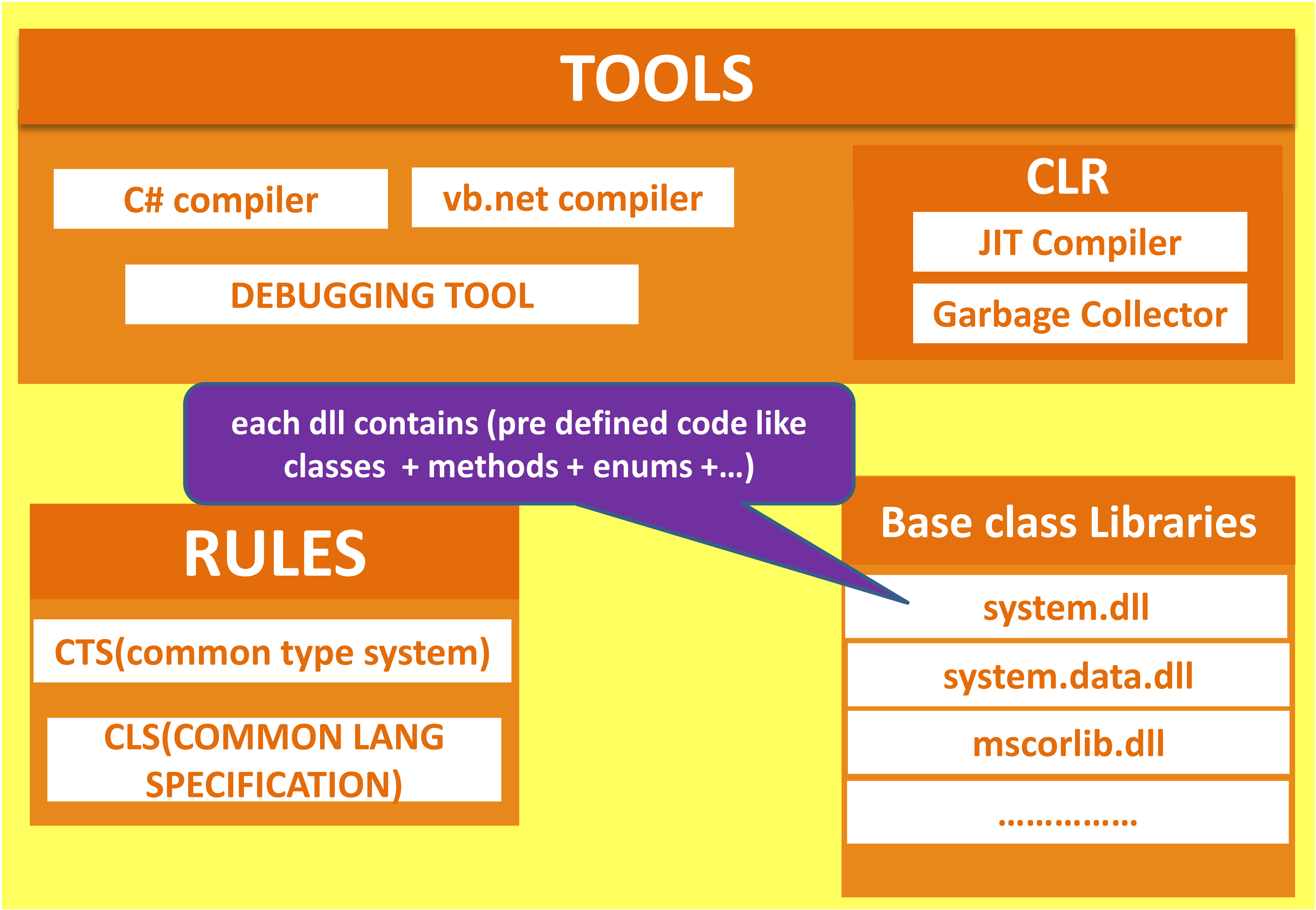
Mobile applications

ClassLibrary applications

WCF applications

WPF applications

.NET FRAMEWORK COMPONENTS



TOOLS

C# compiler

vb.net compiler

DEBUGGING TOOL

CLR

JIT Compiler

Garbage Collector

each dll contains (pre defined code like classes + methods + enums +…)

RULES

CTS(common type system)

CLS(COMMON LANG SPECIFICATION)

Base class Libraries

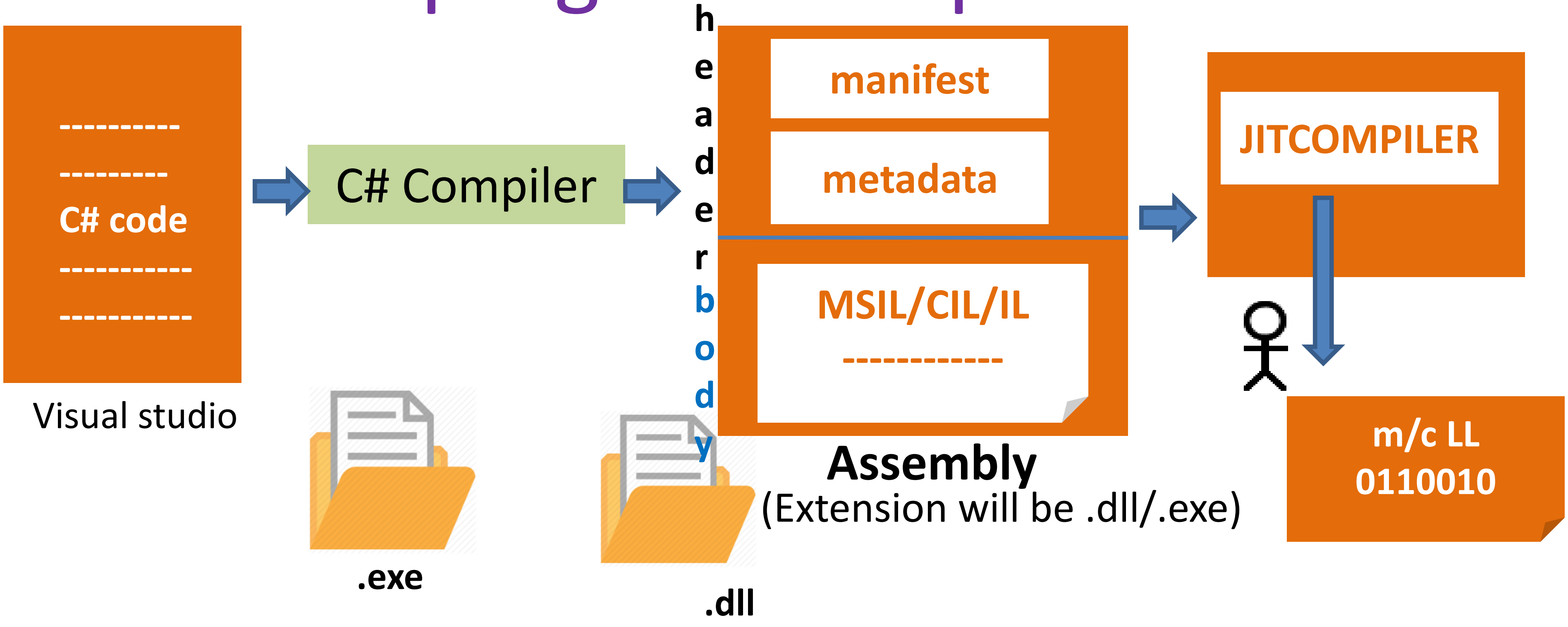
system.dll

system.data.dll

mscorlib.dll

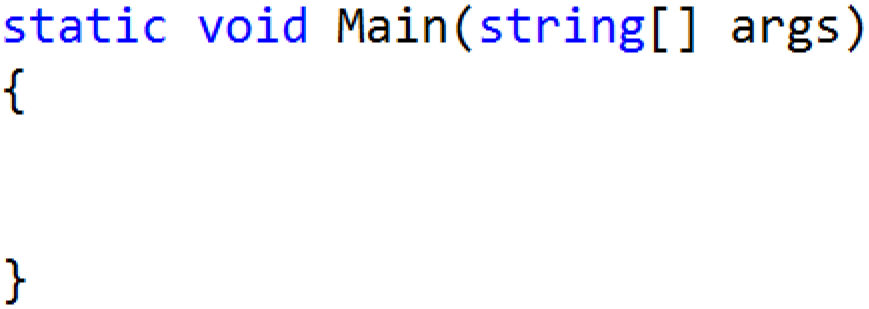
……………

.net program compilation



D:\xyz\-----\project1

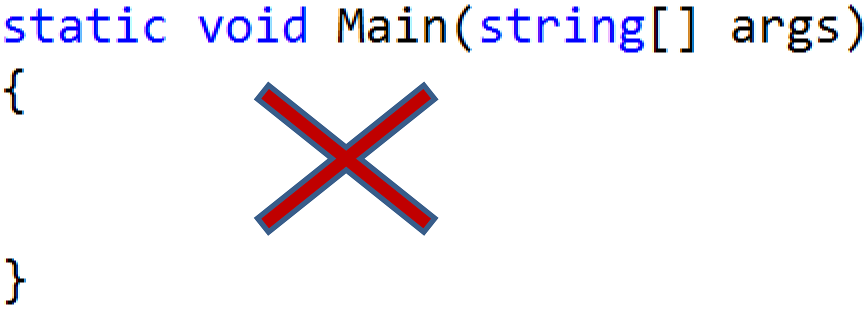
Console Application



```
static void Main(string[] args)
{


}
```

D:\xyz\project1\bin\Debug

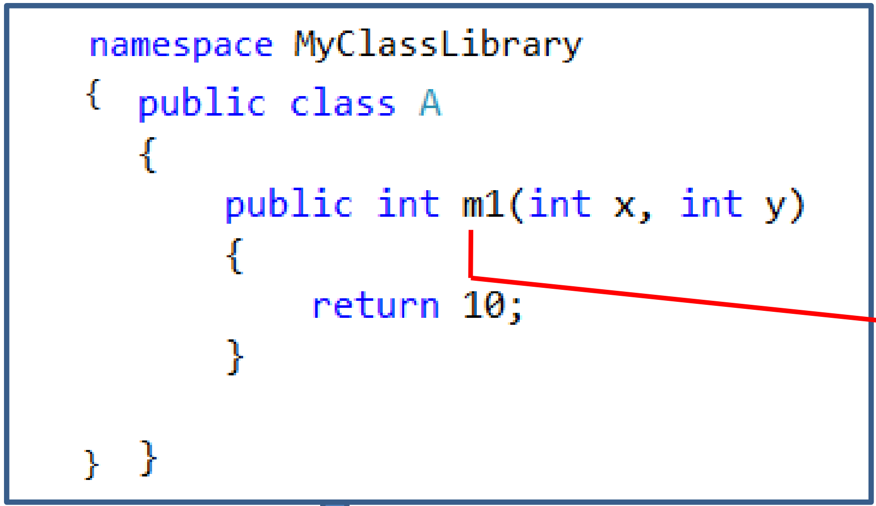
Class Library



```
static void Main(string[] args)
{


}
```

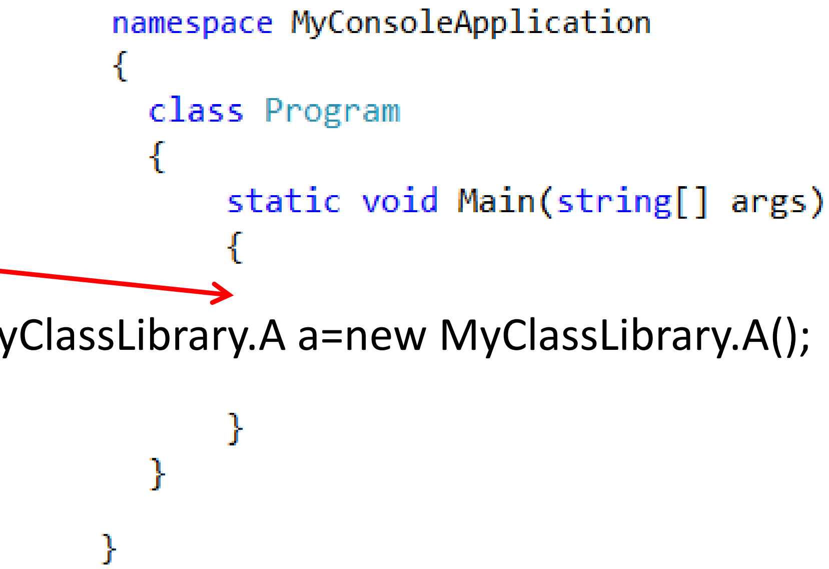
using one project code in another project

C:\MyClassLibrary



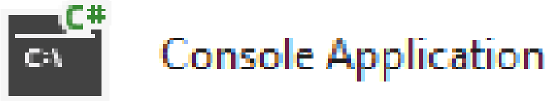
```
namespace MyClassLibrary
{  public class A
   {
        public int m1(int x, int y)
        {
             return 10;
        }

}  }
```

?



```
namespace MyConsoleApplication
{
   class Program
   {
        static void Main(string[] args)
        {
MyClassLibrary.A a=new MyClassLibrary.A();

        }
   }

}
```

C# Compiler



MyClassLibrary.dll

Trainer : must show this concept directly in visual studio.

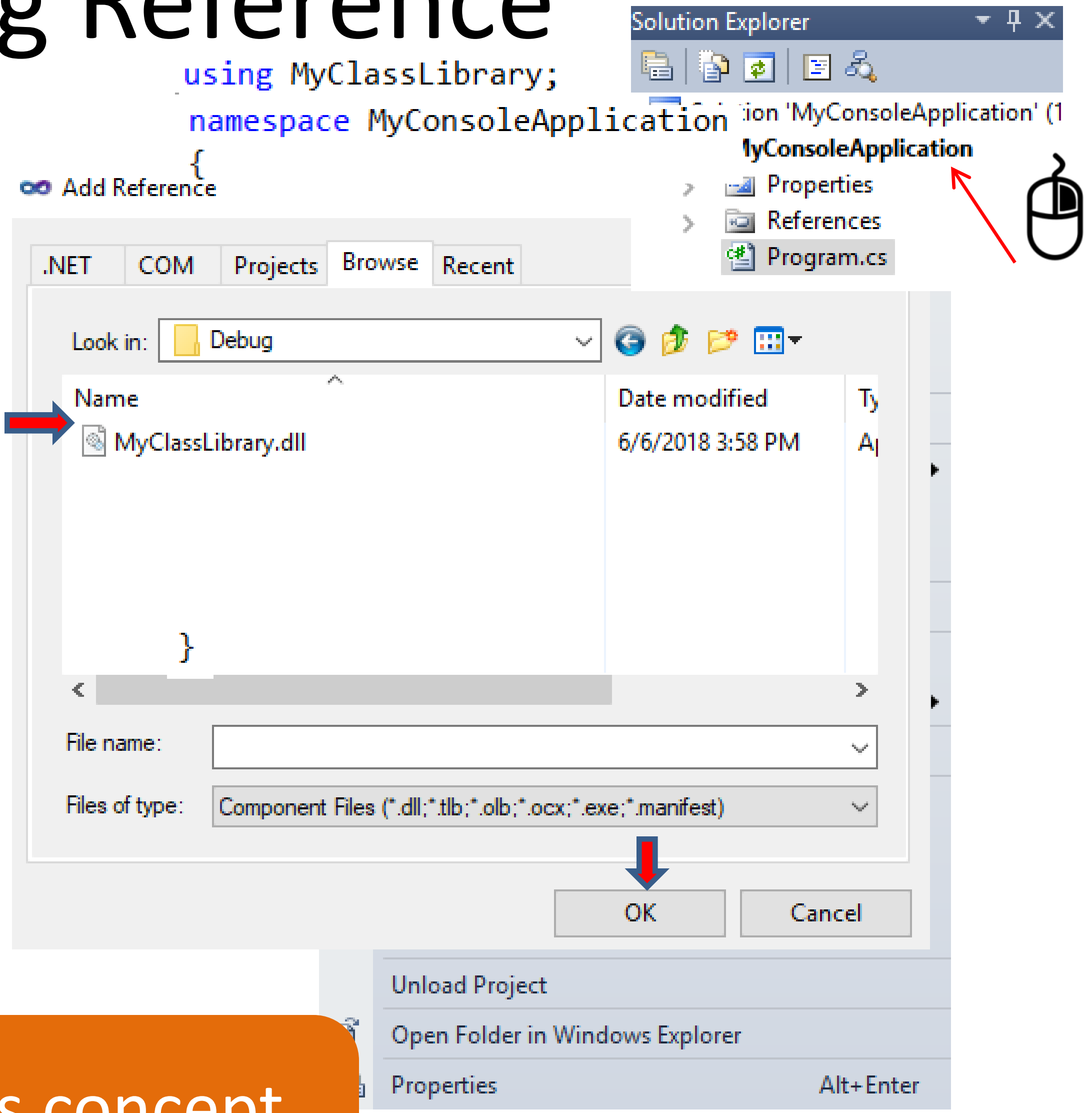
By Adding Reference

Req:Call m1() method in main()

C:\MyClassLibrary\bin\Debug\MyClassLibrary.dll

Adding Reference

```
namespace MyClassLibrary
{
public class A
 {
     public int m1(int x, int y)
     {
         return 10;
     }

 }
}
```



```
using MyClassLibrary;
namespace MyConsoleApplication
{

}
```

Add Reference

Look in: Debug

Name	Date modified	Ty
MyClassLibrary.dll	6/6/2018 3:58 PM	A

File name:

Files of type: Component Files (*.dll;*.tlb;*.olb;*.ocx;*.exe;*.manifest)

trainer : must show this concept directly in visual studio.

IMP Points

- MSIL is platform independent
- CLR & JIT Compilers are platform dependent

IMP Points

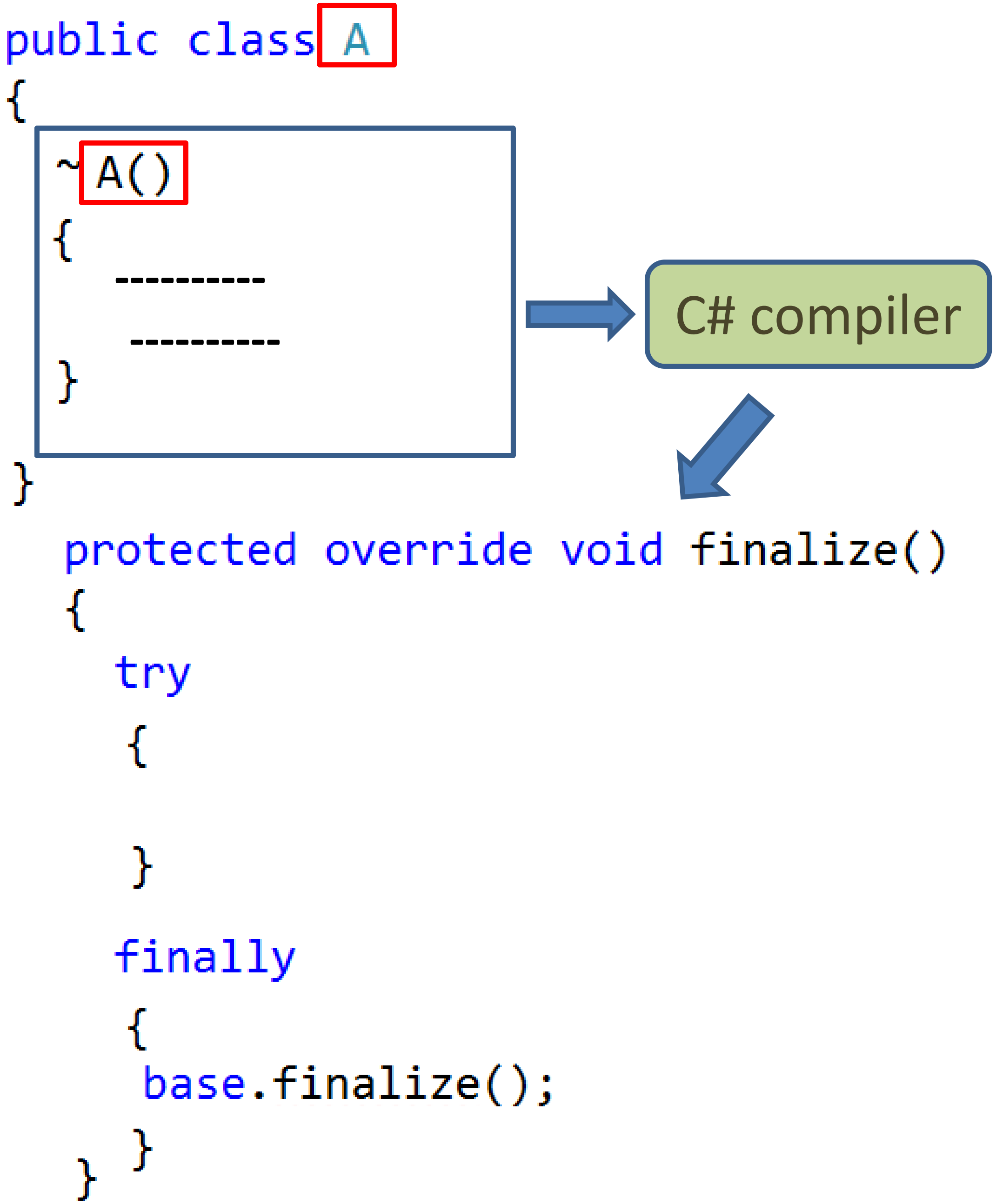
Garbage collector is responsible for deleting the dead or un-used objects present in the memory.

destructor

destructor is used for destroying objects

destructor-syntax

```
public class class_name
{
  ~ class_name( )
     {
        Code for releasing
        unmanaged resource
     }
}
```



```
public class A
{
  ~A()
  {
    ----------
     ----------
  }
}
```

C# compiler

```
   protected override void finalize()
   {
     try
      {

      }

     finally
      {
       base.finalize();
      }
   }
```

destructor execution sequence

destructor will be usually executed from child to parent (you cant kill parent without killing child)

```
public class A
{
    ~A()     2
    {

    }
}
```

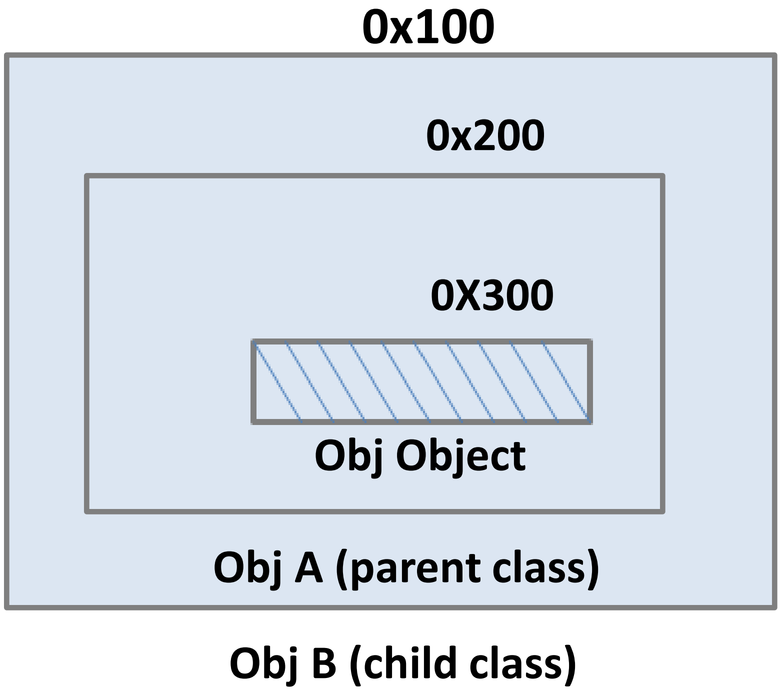
```
class Program
{
    static void Main(string[] args)
    {
        B b = new B();
    }
}
```

```
public class B :A
{
    ~B()     1
    {

    }
}
```

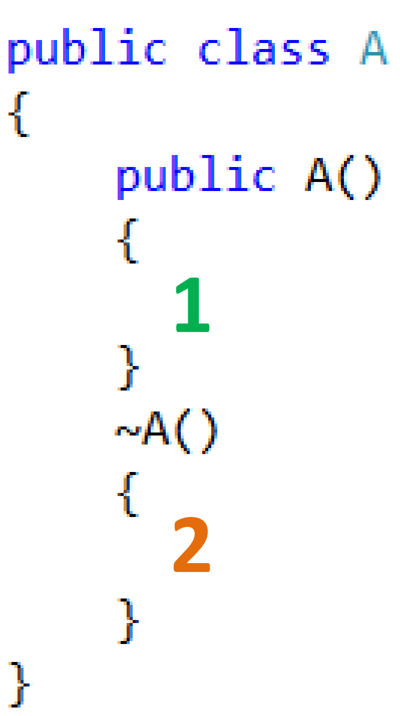
OK

Now tell me which destructor will be executed first?

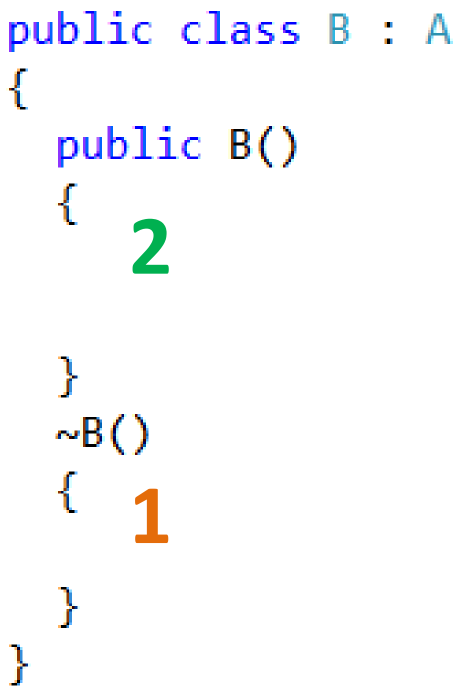
The correct destructor execution sequence is child class and then parent class



Destructor- Execution sequence



```
public class A
{
    public A()
    {
      1
    }
    ~A()
    {
      2
    }
}
```



```
public class B : A
{
   public B()
   {
      2

   }
   ~B()
   {  1

   }
}
```

IDisposable Interface

when a class is implementing IDisposable interface it is recommended to create the object for that class inside the using block

```
Public class A : IDisposable
{
        ................
        ................
}
```

```
using (A a  = new A())
{


}
```

Student need to remember this point(we use this in ADO.Net)

Execute all code and do all Exercises at Home

End of DAY 1

Collections Basics

Collections are used to store collection of related data

Normal collections/Non Generic collection

- ArrayList
- Hashtable
- ListDictionary
- Stack
- Queue
- Hybrid dictionary
- StringDictionary
- SortedList
- Array

dynamic

Array → **Fixed**

```
using System.Collections;
using System.Collections.Specialized;
```

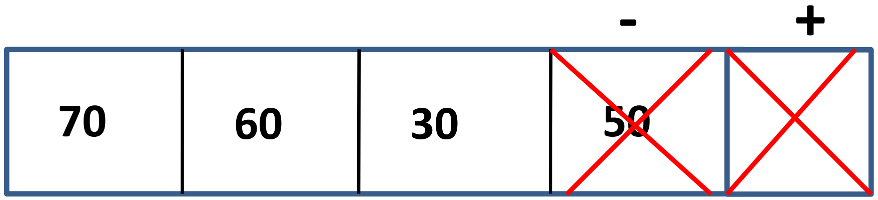
Generic collection

- List<>
- dictionary<>
- SortedList<>
- Stack<>
- Queue<>

```
using System.Collections.Generic;
static void Main(string[] args)
{
}
```

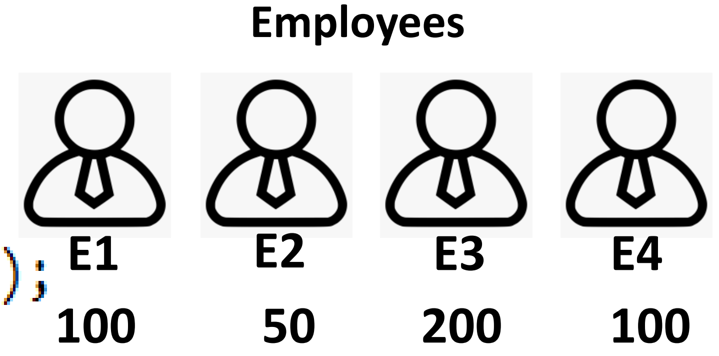
It is bad idea to use non generic collections (since casting conversion are required while using these collections)

Int[ ] marks=new int[ ]{70,60,30,50}



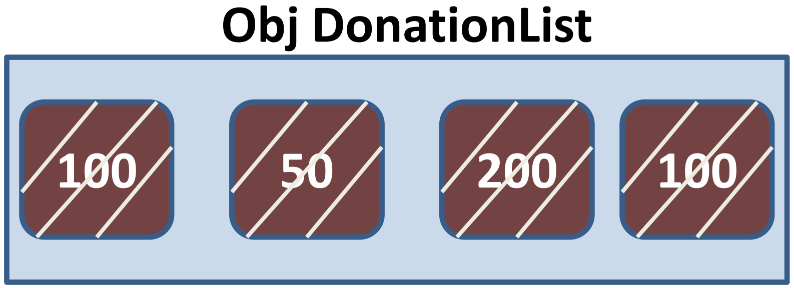
ArrayList

Req 1: store donation amount of each employee in a collection



```
ArrayList DonationList = new ArrayList();
DonationList.Add(100);
DonationList.Add(50);
DonationList.Add(200);
DonationList.Add(100);
```

Boxing



Req 2: Before donating this amount, Employer will add extra 20 to the donation amount of each employee

```
DonationList[0] = (int)DonationList[0] + 20;
DonationList[1] = (int)DonationList[1] +
DonationList[2] = (int)DonationList[2] +
DonationList[3] = (int)DonationList[3] + 20;
```

Unboxing

ype 'object' and 'int'

Execute all code and do all Exercises at Home

End of DAY 2

Generics

- Generics are useful for achieving reusability and type safety.
- Using generics we can avoid casting conversions
- we can apply generics to classes, Interface, Methods, Structures & Delegates

Generic classes

```
public class A
{
   ----------
   ----------
   ----------
}
 A a=new A();
 A a2=new A();
 ----------
 ----------
```

Type parameter

```
public class B<T>
{


}

 B<int> b1=new B<int>();

 B<int> b2=new B<int>();
 ----------
 ----------
```

```
B<char> b2=new B<char>(); ✓

B<int[]> b4 = new B<int[]>();✓
B<A> b5 = new B<A>(); ✓
B<B<int>> b6 = new <B<int>>();✓
```

```
D< int,int>  d1=new  D<int, int > ();

D<int, char> d2= new  D<int, cha>();

D<bool, string>  d3= new  D<bool,string>();

D<D, int[]> d4= new D<D, int[]>();
```

```
p  c   D<T1,T2>
{
   =
}
```

D → int, int ✓

D → int, cha ✓

D → bool, str /

D → $\underline{D}$, $\underline{int[]}$ /

Generic class-Assignment

```
public class D<T1, T2>
{

}
```

D type

```
public class E<T3>
{

}
```

E type

```
public class F<T4>
{

}
```

F type

```
public class G<T5>
{

}
```

Create an object for generic class G?

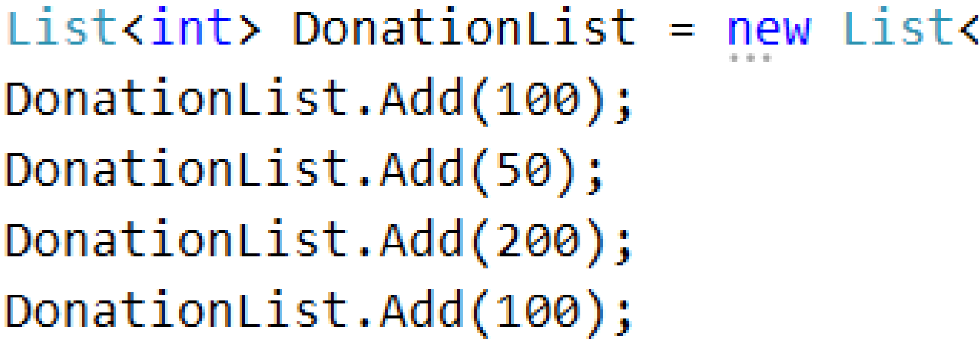
Using Generic Parameter in Method

```
public class A<T>
{
   public T m1(T t1, int x)
   {

      return---------;

    }

 }
A<char> a1 = new A<char>();
char c1= a1.m1('$', 20); ✓
A<bool> a2 = new A<bool>();
bool b1 = a2.m1(false, 200);✓
```

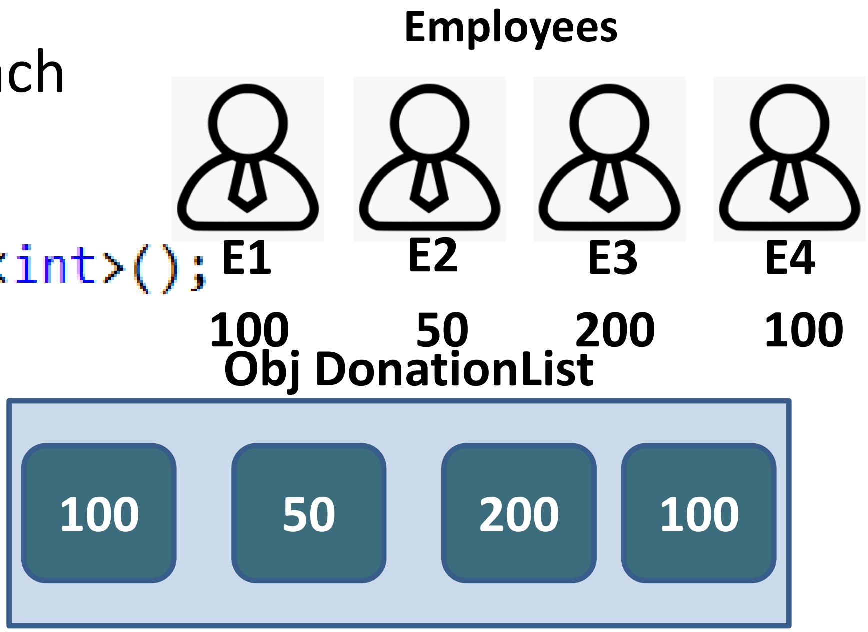
What Is T expecting

List<>

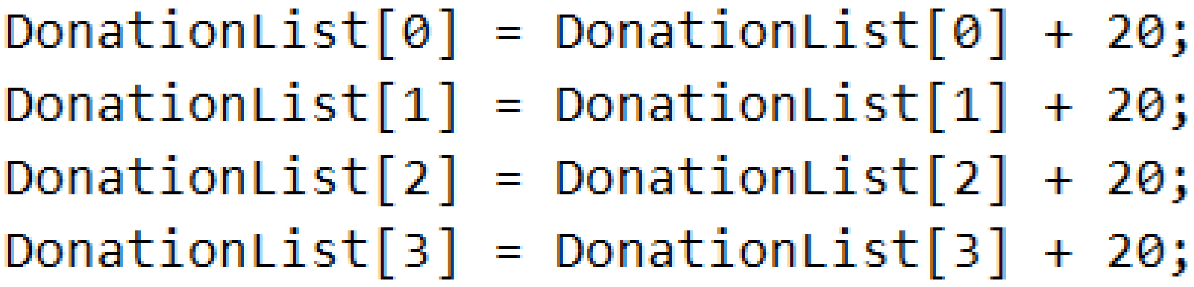
Req 1: store donation amount of each employee in a collection



```
List<int> DonationList = new List<int>();
DonationList.Add(100);
DonationList.Add(50);
DonationList.Add(200);
DonationList.Add(100);
```



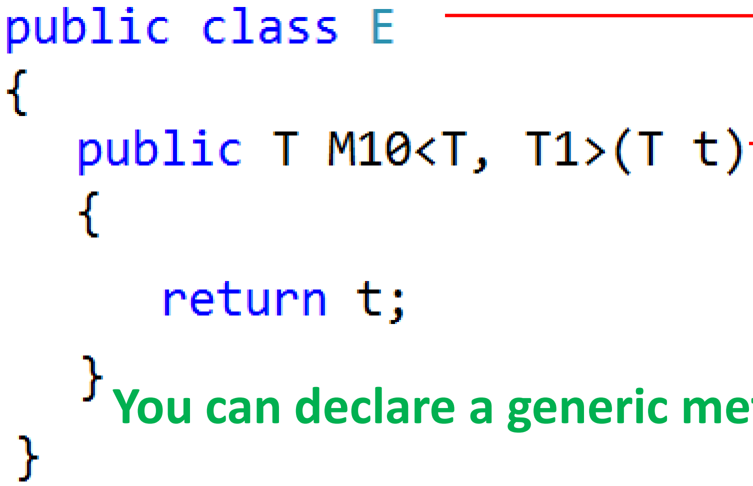
Req 2: Before donating this amount, Employer will add extra 20 to the donation amount of each employee



```
DonationList[0] = DonationList[0] + 20;
DonationList[1] = DonationList[1] + 20;
DonationList[2] = DonationList[2] + 20;
DonationList[3] = DonationList[3] + 20;
```

Note: Boxing / UnBoxing operations will not be performed when we are using Generic collections.

Generic Methods



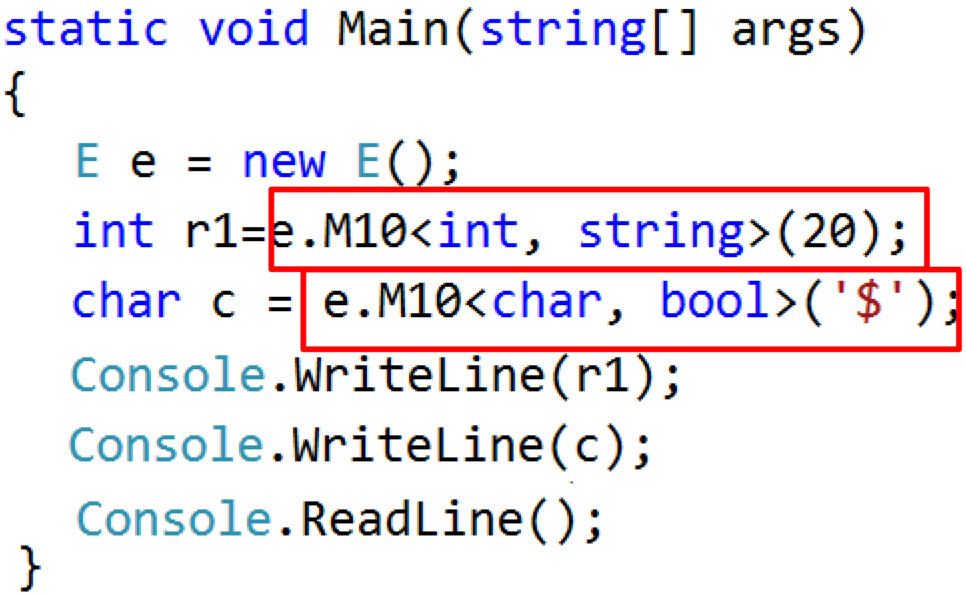
```
public class E
{
   public T M10<T, T1>(T t)
   {
      return t;
   }
}
```

Normal class

Generic method

You can declare a generic method either in a generic class or in non generic class

How to call Generic method??

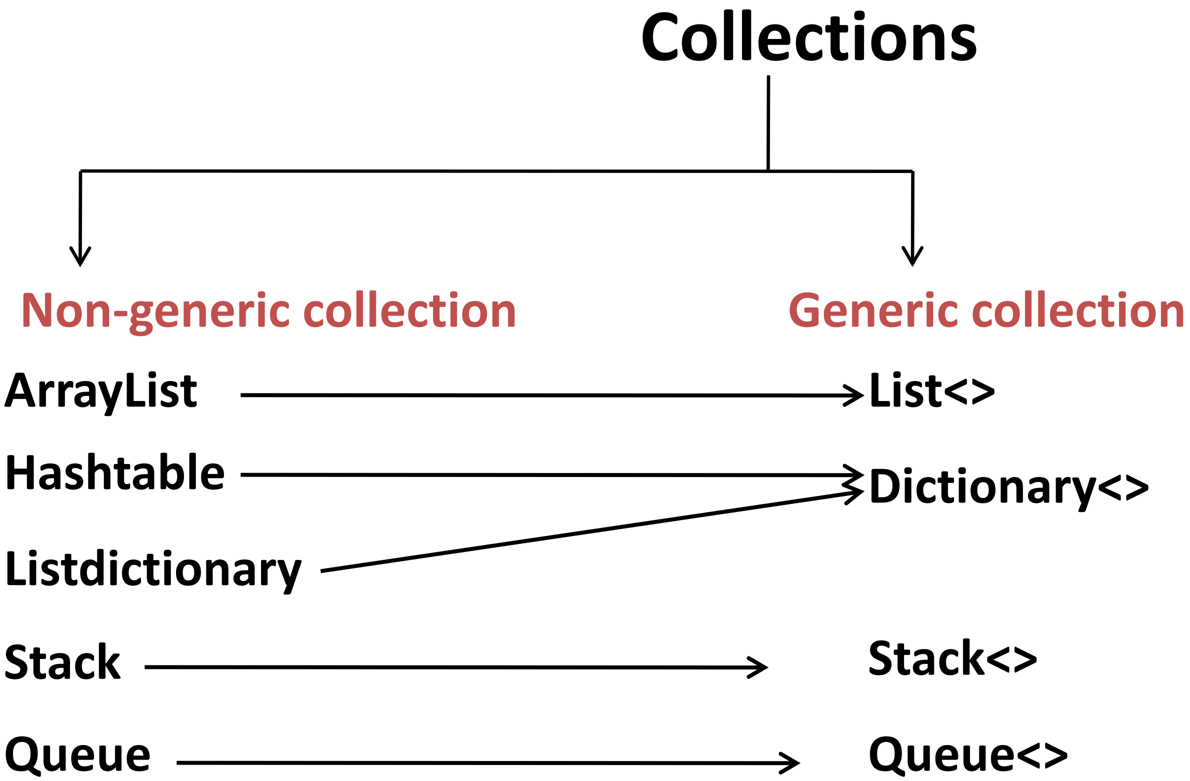


```
static void Main(string[] args)
{
   E e = new E();
   int r1=e.M10<int, string>(20);
   char c = e.M10<char, bool>('$');
   Console.WriteLine(r1);
   Console.WriteLine(c);
   Console.ReadLine();
}
```

Execute all code and do all Exercises at Home

End of DAY 3

Generic Collections



Collections

Non-generic collection

Generic collection

ArrayList

List<>

Hashtable

Dictionary<>

Listdictionary

Stack

Stack<>

Queue

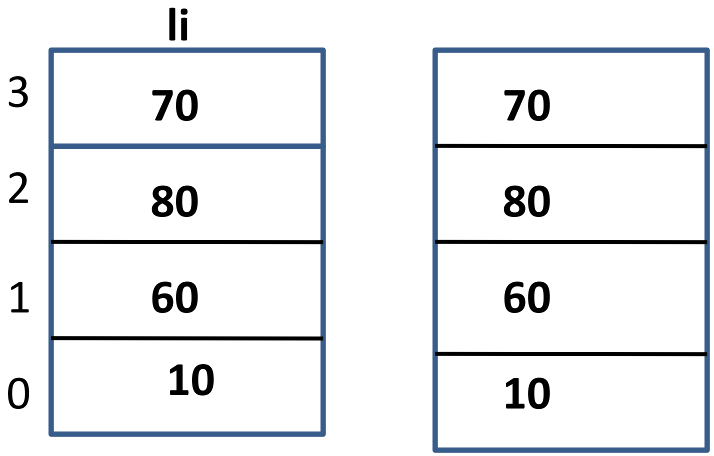
Queue<>

List< >

```
List<int> l1 = new List<int>();
l1.Add(10);
l1.Add(60);
l1.Add(80);
l1.Add(70);
l1.Add("10"); ✗
```

List<int>.Add(int item)

Method expecting the input in int format



Index based retrieval is possible

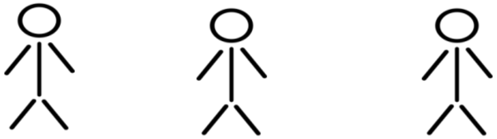
```
int x1 = l1[0]; //Retreiving data from List<>collection
```

```
foreach (int i in l1)
{
    Console.WriteLine(i);
}
```

OR

```
for (int i = 0;i<l1.Count; i++)
{
    Console.WriteLine(l1[i]);
}
```

List Collection-Assignment



kiran	Mahesh	Veena
39	40	30
O+ve	AB+ve	O-ve

```
public class Patient
{
  public string name { get; set; }
   public int Age { get; set; }
  public string Bg { get; set; }
 }
 public class PatientFactory
 {
  public List<Patient> GetPatient()
   {
   Patient p1= new Patient();
    p1.name = "Kiran";
    p1.Age = 39;
    p1.Bg = "O+ve";
    -----------    ?
    -----------
   }
 }
```

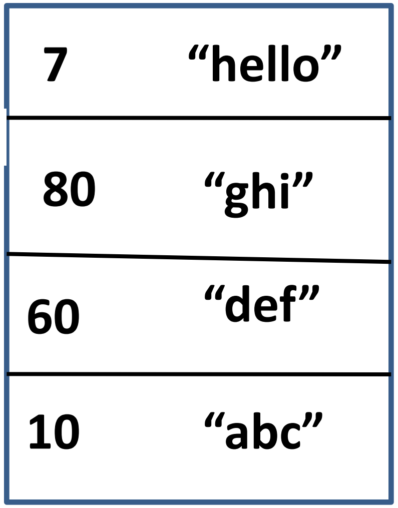
List Collection Assignment-Solution

```
public class PatientFactory
{
 public List<Patient> GetPatient()
 {
    Patient p1= new Patient();
    p1.name = "Kiran";
    p1.Age = 39;
    p1.Bg = "O+ve";
    Patient p2 = new Patient();
    p2.name = "Mahesh";
    p2.Age = 40;
    p2.Bg = "AB+ve";
    Patient p3 = new Patient();
    p3.name = "veena";
    p3.Age = 30;
    p3.Bg = "O-ve";
    List<Patient> l = new List<Patient>();
    l.Add(p1);
    l.Add(p2);
    l.Add(p3);
    return l;
  }
}
```

```
static void Main(string[] args)
{
PatientFactory pf = new PatientFactory();
List<Patient> l2 = pf.GetPatient();
  foreach (Patient p in l2)
  {
      Console.WriteLine(p.name);
      Console.WriteLine(p.Age);
      Console.WriteLine(p.Bg);
  }
  Console.ReadLine();
}
```

Dictionary<>Assignment

```
static void Main(string[] args)
{
 Dictionary<int, string> d = new Dictionary<int, string>();
  d.Add(10, "abc");
  d.Add(60, "def");
  d.Add(80, "ghi");
  d.Add(7, "hello");

foreach (KeyValuePair<int,string> K in d)
{
   Console.WriteLine(K.Key);

   Console.WriteLine(K.Value);
 }
   Console.ReadLine();
 }
```

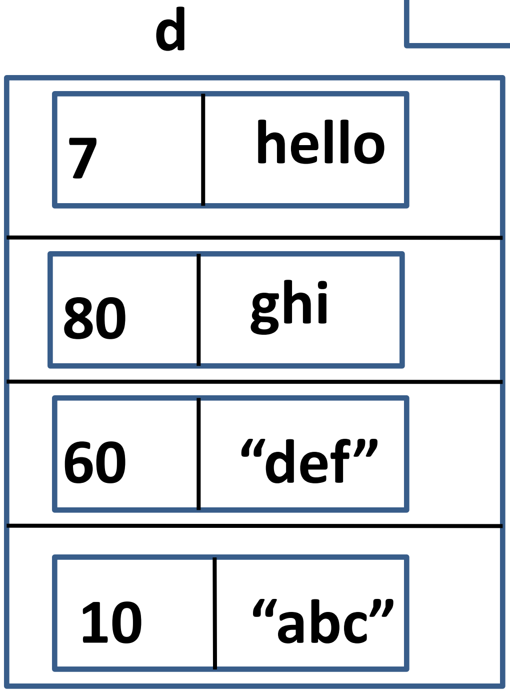


7 “hello”

80 “ghi”

60 “def”

10 “abc”



d

7 hello

80 ghi

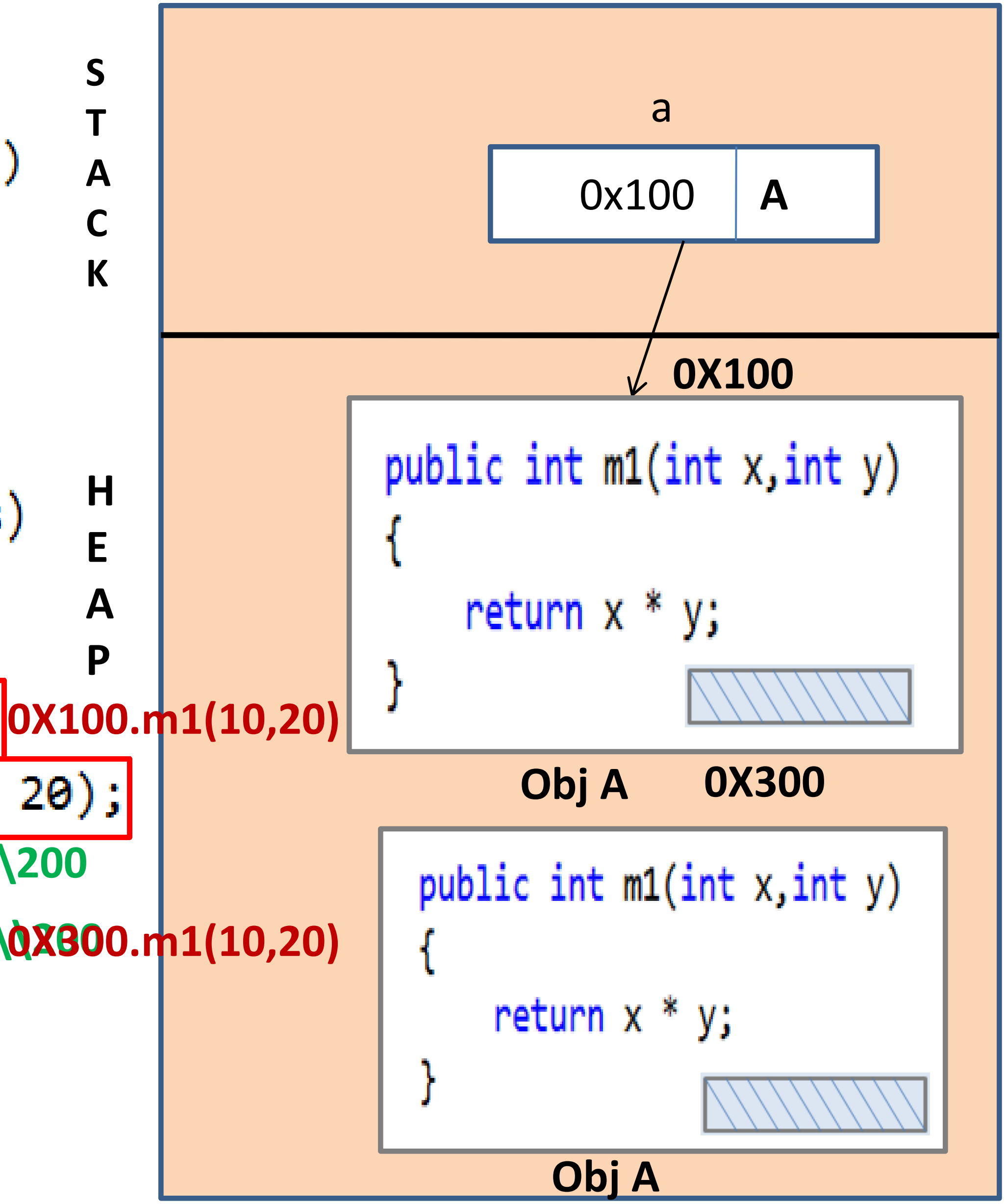
60 “def”

10 “abc”

Key,value will stored in KeyValuePair object

Calling instance method (Delegate)

```
public class A
{
    public int m1(int x,int y)
    {
        return x * y;
    }
}

static void Main(string[] args)
{
 A a = new A();
 int res = a.m1(10, 20);
 int res1 = new A().m1(10, 20);
 Console.WriteLine(res); \\200
 Console.WriteLine(res1); \\200
 Console.ReadLine();
}
```



delegates

- Delegate is a function pointer (It stores a function name as well as object address)
- Delegates are useful for implementing call back methods
- Delegate is internally treated as a class (but it appears like a method) hence we can create object for the delegate
- Delegate constructor always takes object address and method name.
- Delegates are usually useful for calling un-known methods present in un-known classes whose return type and input is known

delegate syntax

Syntax:

```
AM delegate RT <delegatename> (dt v1,dt v2,..);
```

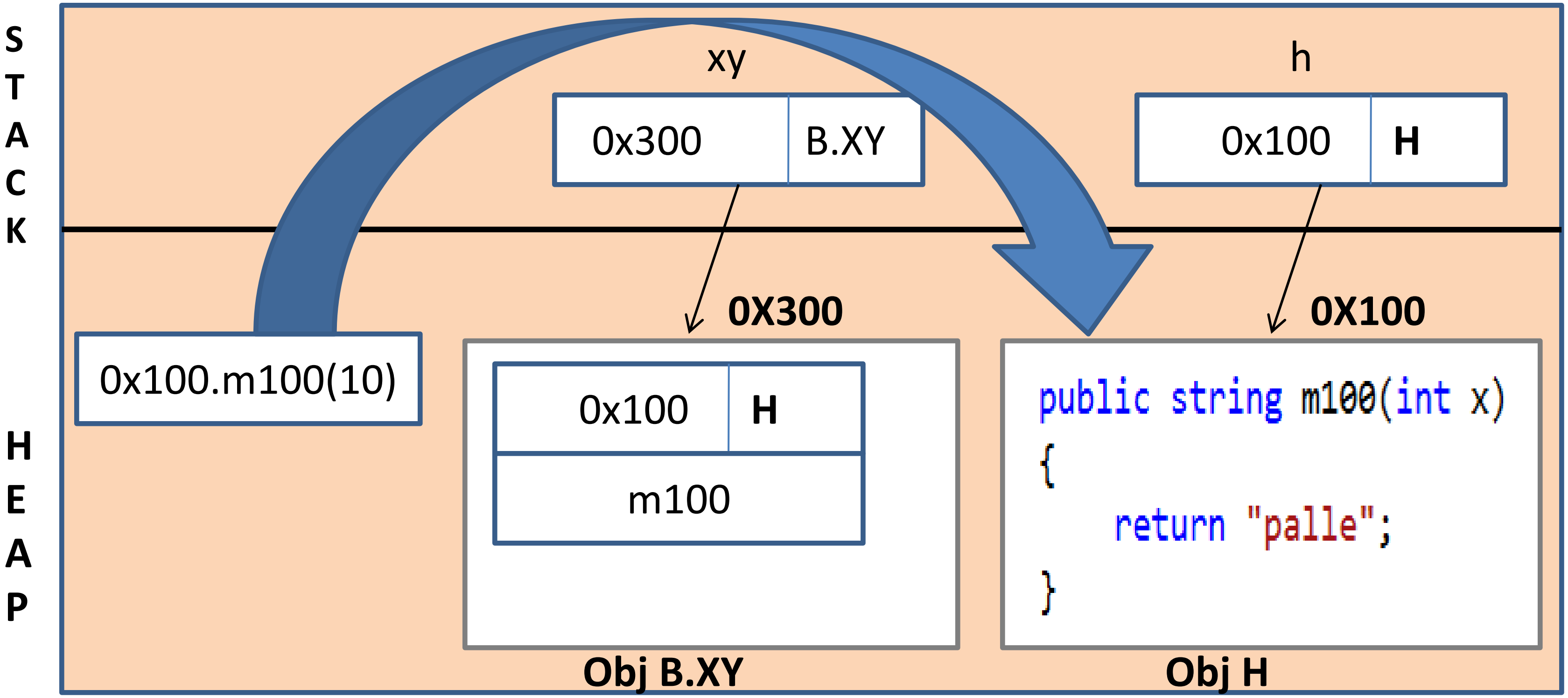
Creating Delegate Object:

```
delegatename variable = new
  delegatename(objaddress.methodname);
```

delegates sample

```
public class B
{
    public delegate string XY(int y);
}
public class H
{
    public string m100(int x)
    {
        return "palle";
    }
}
```

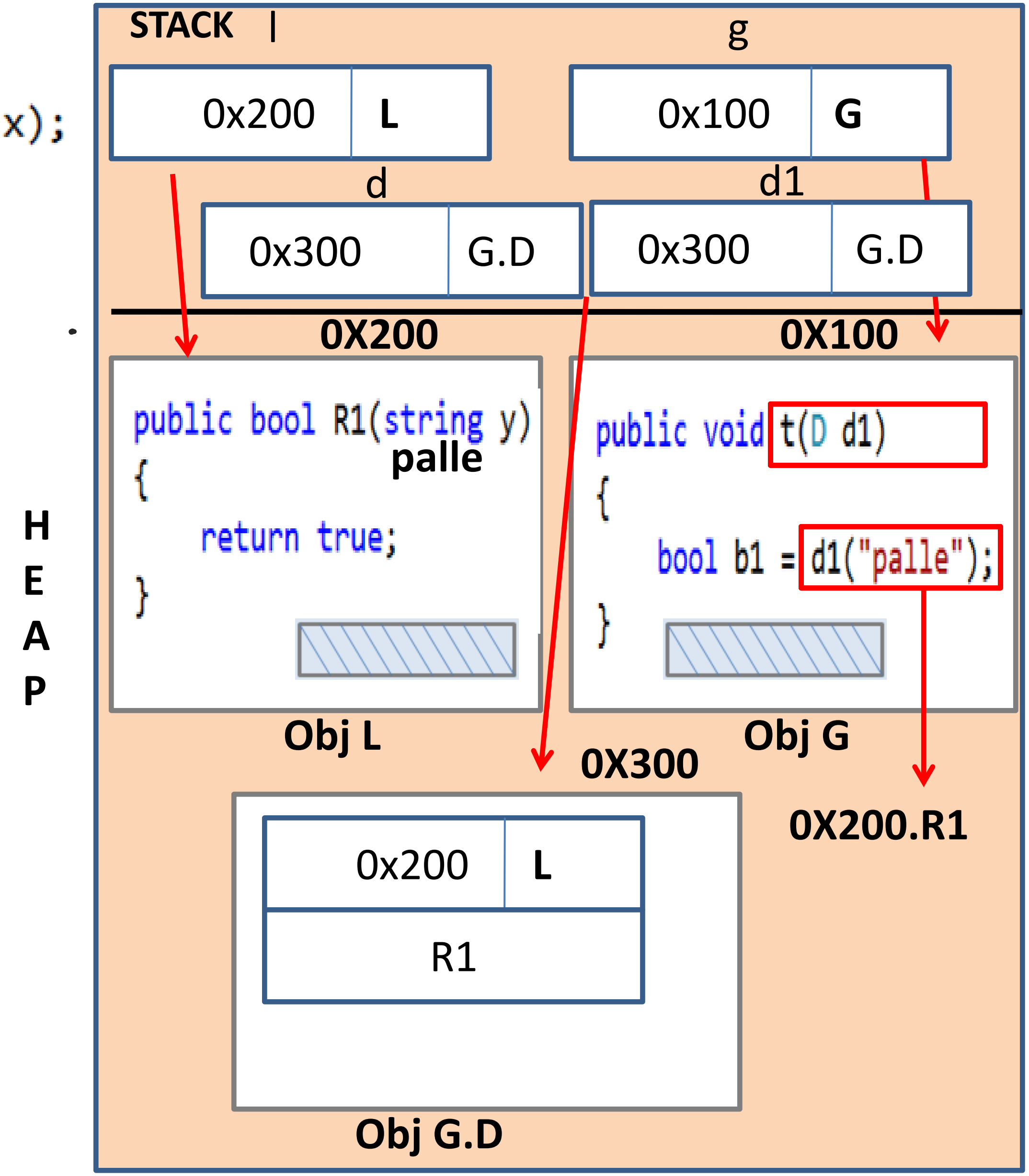
```
static void Main(string[] args)
{
    H h = new H();
    B.XY xy = new B.XY(h.m100);
    string r = xy(10);
}
```



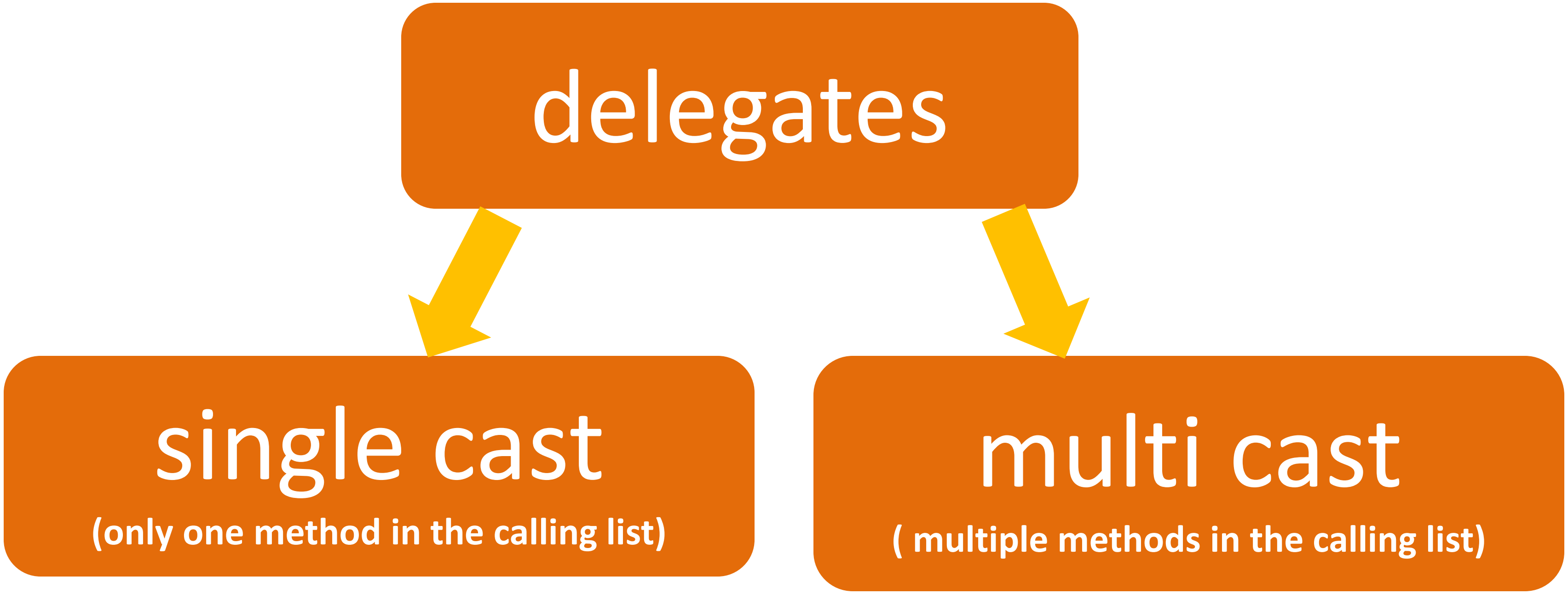
```
public string m100(int x)
{
    return "palle";
}
```

delegates assignment

```
public class G
{
    public delegate bool D(string x);
    public void t( D d1)
    {
      bool b1 = d1("palle");
    }
}
 public class L
 {
     public bool R1(string y)
     {
         Console.WriteLine(y);
         return true;
     }
 }

static void Main(string[] args)
{
    G g = new G();
    L l = new L();
    G.D d = new G.D(l.R1);
    g.t(d);
}
```



types of delegates



delegates

single cast

(only one method in the calling list)

multi cast

(multiple methods in the calling list)

Events

- Events are usually useful for **implementing notification mechanism.**
- Events follows **Publisher and Subscriber mechanism**.
- Events are usually used in **GUI programming**.
- Events are declared by **using delegates**.
- Event is a **global variable** and will **store address of a delegate**.

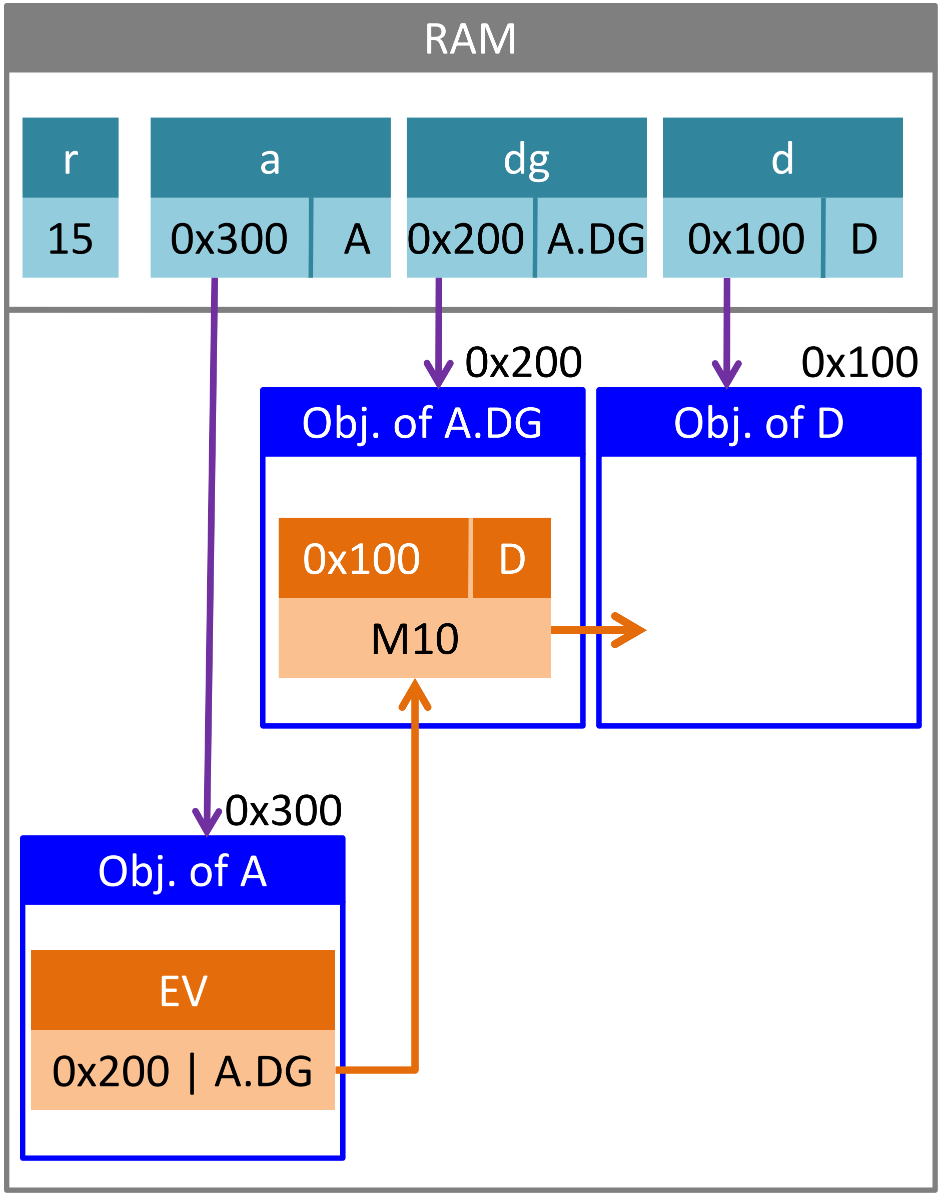
Syntax:

```
public event <existing_delegate_name> <event_name>;
```

Example:

```
public delegate void D(int x, int y); //Delegate
public event D e; //Event
```

Events sample



RAM

r	a		dg		d	
15	0x300	A	0x200	A.DG	0x100	D

```
public class A
{
  public delegate void DG(int n);
  public event DG EV;
  public void M2()
  {
   0x100.M10(10);
  }
}

Public class D
{
 Public void M10(int r)
 {
   r= r + 5;
   Console.WriteLine(r);
 }
}

class Program
{
 static void Main(string[] args)
 {
   D d = new D();
   A.DG dg = new A.DG(d.M10);
   A a = new A();
   a.EV += dg;
   a.M2();
 }
}
```

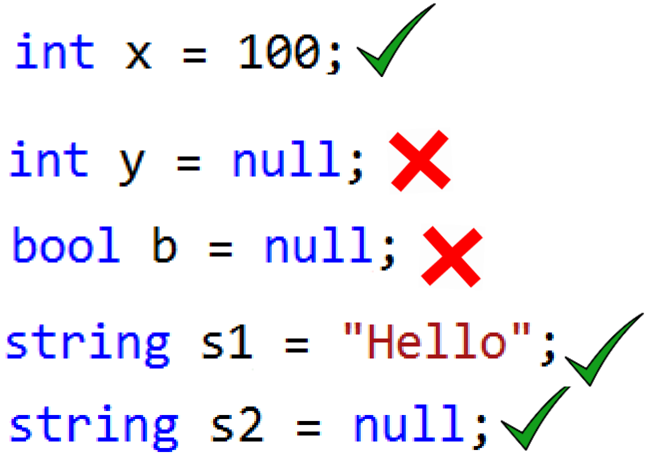
10

15

15

nullable type

Nullable Types are value types which are capable of storing null values along with other valid value type data

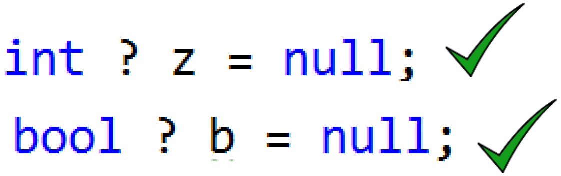


```
int x = 100; ✓
int y = null; ✗
bool b = null; ✗
string s1 = "Hello"; ✓
string s2 = null; ✓
```

Value type cant store null values

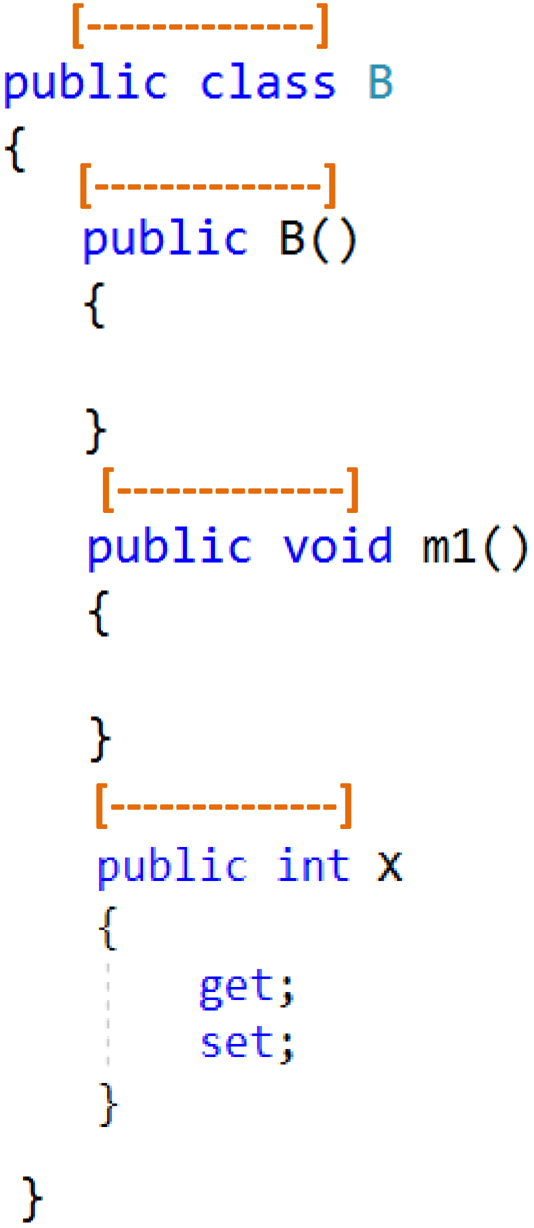
How to make it possible?

By using Nullable type



```
int ? z = null; ✓
bool ? b = null; ✓
```

attributes



```
   [--------------]
public class B
{
    [--------------]
    public B()
    {

    }
     [--------------]
    public void m1()
    {

    }
     [--------------]
     public int X
     {
         get;
         set;
     }
 }
```

using attributes we can add custom metadata into the metadata session of the assembly.

Execute all code and do all Exercises at Home

End of
DAY 4 & 5