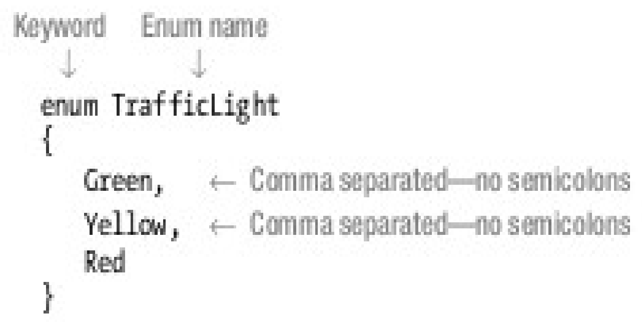
CLASSES

Asma Kanwal
Lecturer
Department of Computer Science
GC University, Lahore

ENUMERATION

An enumeration, or enum, is a programmer-defined type, such as a class or a struct.

- Like structs, enums are value types and therefore store their data directly, rather than separately, with a reference and data.
- Enums have only one type of member: named constants with integer values.



```
enum TrafficLight
{
    Green,
    Yellow,
    Red
}
```

Enumeration

Every enum type has an underlying integer type, which by default is int.

- Each enum member is assigned a constant value of the underlying type.
- By default, the compiler assigns 0 to the first member and assigns each subsequent member the value one more than the previous member.

Example

```
enum TrafficLight
  {
     Green,
     Yellow,
     Red
  }
  Class Program {
                  static void Main( ){
  TrafficLight t1 = TrafficLight.Green;
  TrafficLight t2 = TrafficLight.Yellow;
  TrafficLight t3 = TrafficLight.Red;

  Console.WriteLine("{0},\t{1}",   t1, (int) t1);
  Console.WriteLine("{0},\t{1}",   t2, (int) t2);
  Console.WriteLine("{0},\t{1}\n", t3, (int) t3);
}
}
```

Output:

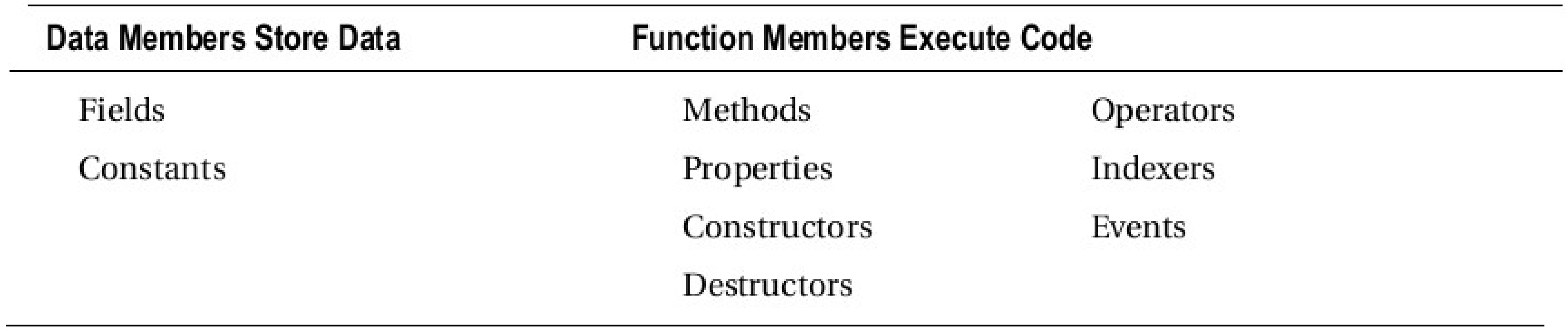
```
Green,  0
Yellow, 1
Red,    2
```

Class

A class is a data structure that can store data and execute code. It contains data members and function members:

- Data members store data associated with the class or an instance of the class. Data members generally model the attributes of the real-world object the class represents.
- Function members execute code. These generally model the functions and actions of the real-world object the class represents.

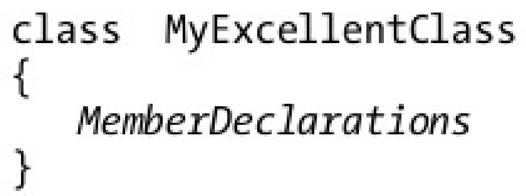
CLASS



Data Members Store Data	Function Members Execute Code	
Fields	Methods	Operators
Constants	Properties	Indexers
	Constructors	Events
	Destructors	

Class Declaration

- The class name
- The members of the class
- The characteristics of the class



```
class  MyExcellentClass
{
   MemberDeclarations
}
```

Class Members

Fields

- A field is a variable that belongs to a class.
- It can be of any type, either predefined or user-defined.

Like all variables, fields store data and have the following characteristics:

They can be written to.

They can be read from.

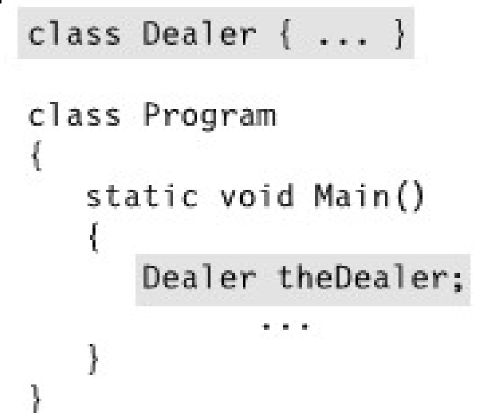
Class Members

Methods

- Return type: This states the type of value the method returns. If a method doesn’t return a value, the return type is specified as void.
- Name: This is the name of the method.
- Parameter list: This consists of at least an empty set of matching parentheses. If there are parameters they’re listed between the parentheses.
- Method body: This consists of a matching set of curly braces containing the executable code.

INSTANCES OF A CLASS

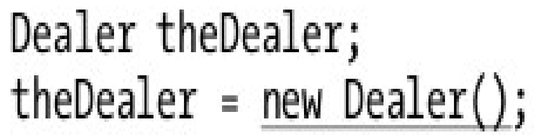
- Classes are reference types, which, as they require memory both for the reference to the data and for the actual data.
- The reference to the data is stored in a variable of the class type. So, to create an instance of the class, start by declaring a variable of the class type. If the variable isn't initialize d, its value is undefined.



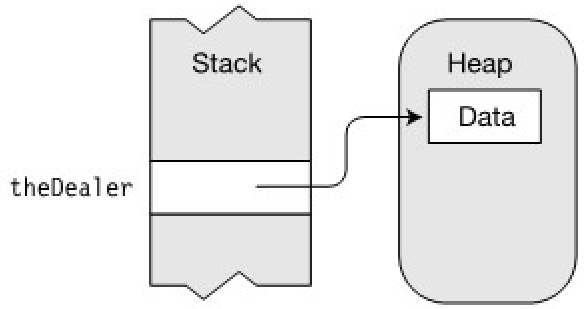
```
class Dealer { ... }

class Program
{
    static void Main()
    {
        Dealer theDealer;
                ...
    }
}
```

MEMORY ALLOCATION FOR DATA

- The keyword new.
- The name of the type of the instance for which memory is to be allocated.
- Matching parentheses, which might or might not include parameters.



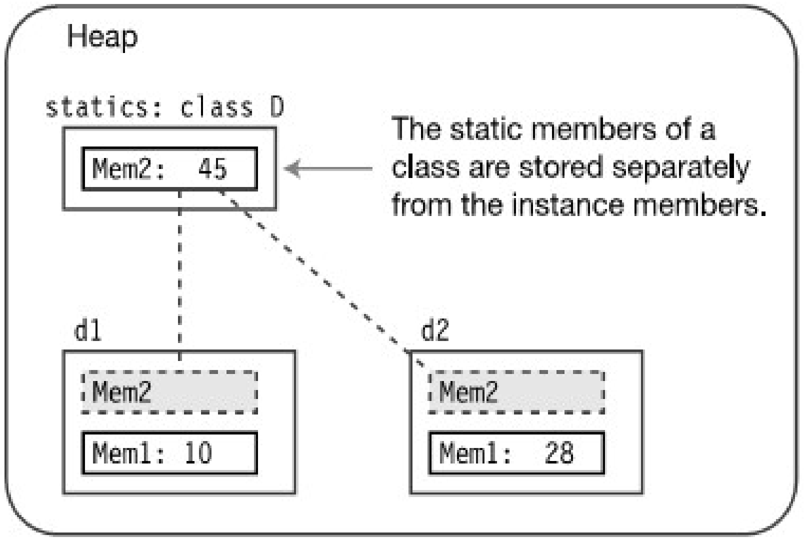
```
Dealer theDealer;
theDealer = new Dealer();
```



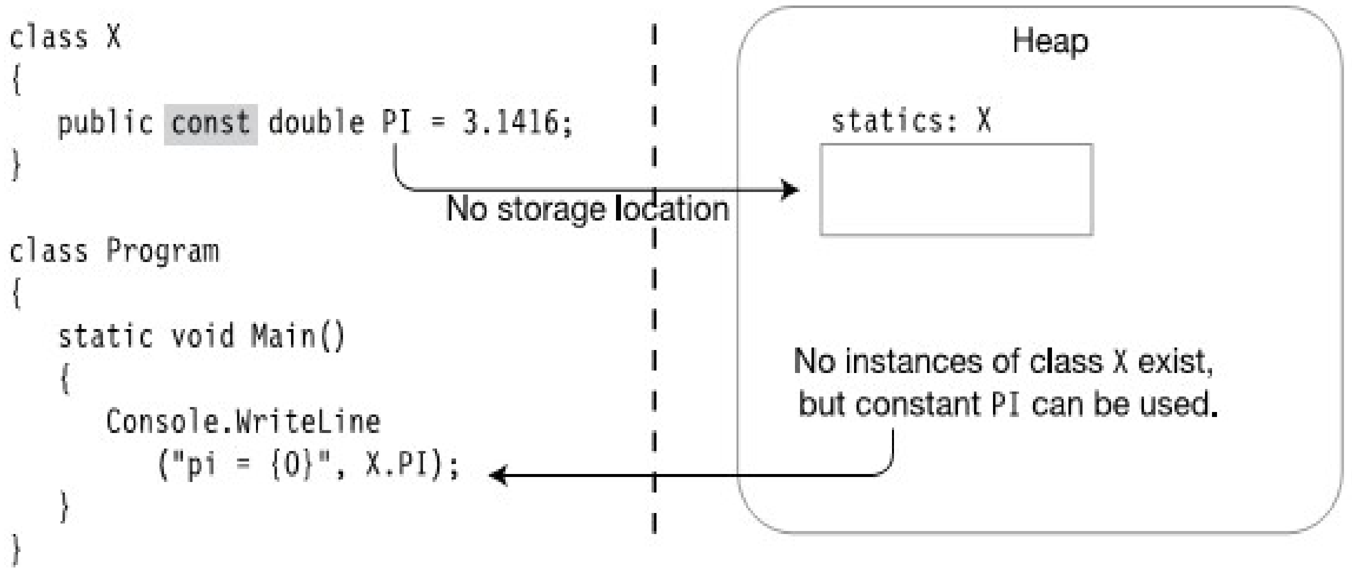
Static Fields

A static field is shared by all the instances of the class, and all the instances access the same memory location. Hence, if the value of the memory location is changed by one instance, the change is visible to all the instances.

```
class D
{
    int Mem1;
    static int Mem2;
    ...
}


static void Main()
{
    D d1 = new D();
    D d2 = new D();
    ...
}
```


Heap
statics: class D
Mem2: 45
The static members of a class are stored separately from the instance members.
d1
Mem2
Mem1: 10
d2
Mem2
Mem1: 28

CONSTANT FIELDS

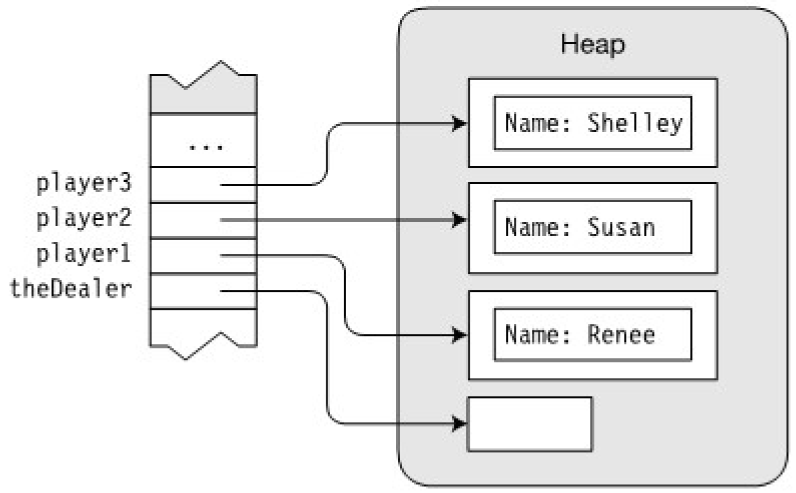


```
class X
{
   public const double PI = 3.1416;
}

class Program
{
   static void Main()
   {
      Console.WriteLine
         ("pi = {0}", X.PI);
   }
}
```

INSTANCE MEMBERS

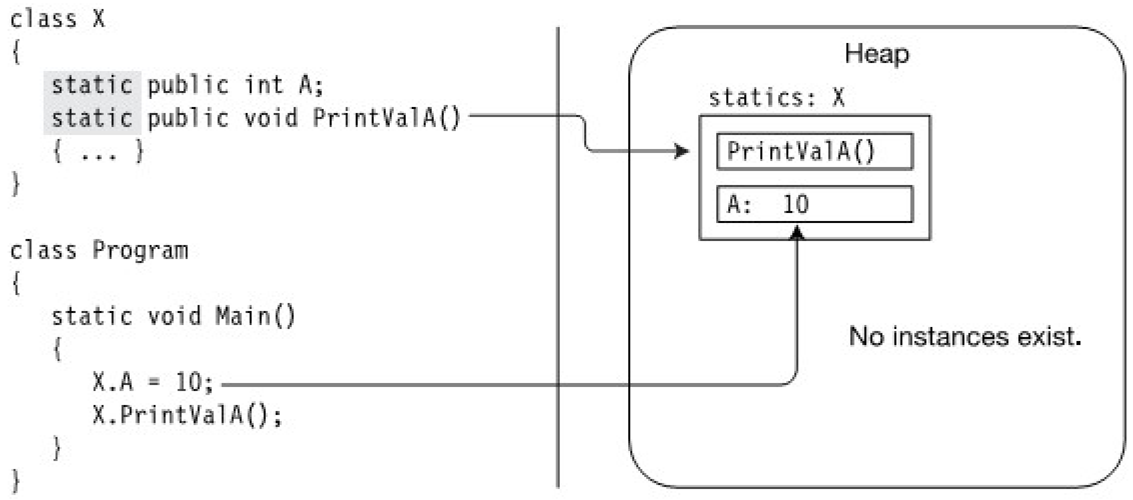
Each instance of a class is a separate entity that has its own set

of data members, distinct from the other instances of the same class. These are called instance members since they are associated with an instance of the class.



Static Function Members

- Static function members, like static fields, are independent of any class instance. Even if there are no instances of a class, you can still call a static method.
- Static function members cannot access instance members. They can, however, access other static members.

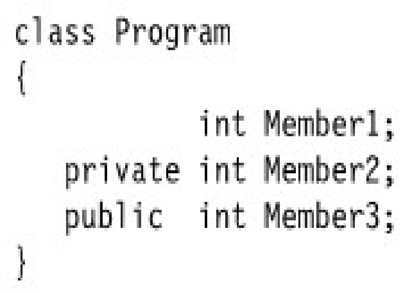


```
class X
{
    static public int A;
    static public void PrintValA()
    { ... }
}

class Program
{
    static void Main()
    {
        X.A = 10;
        X.PrintValA();
    }
}
```

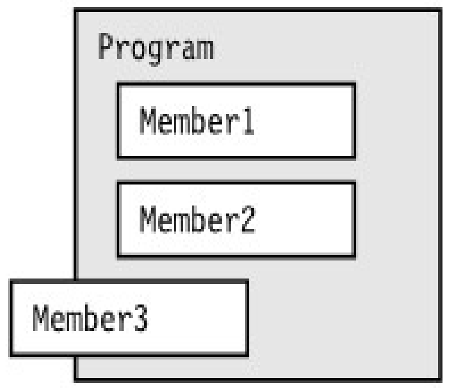
Access Modifiers

- private
- public
- protected
- internal
- protected internal

Public v/s Private

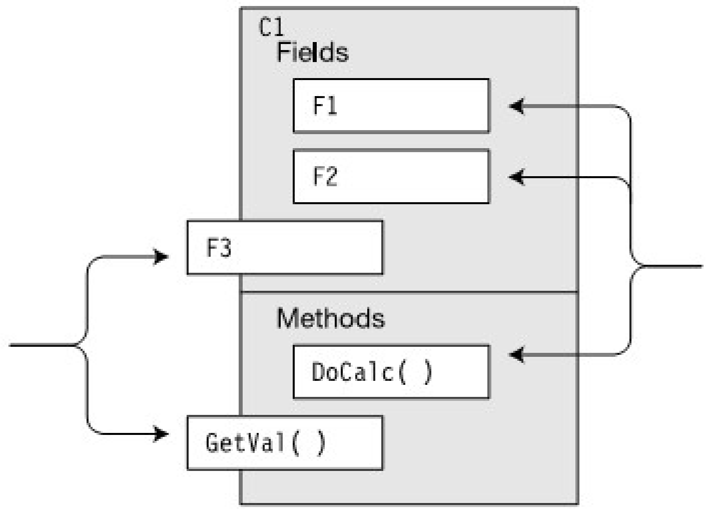


```
class Program
{
             int Member1;
   private int Member2;
   public  int Member3;
}
```



Example

```
class C1
{
    int         F1;
    private int F2;
    public  int F3;

    void DoCalc()
    {
        ...
    }

    public int GetVal()
    {
        ...
    }
}
```



Constructors

- Instance Constructor

  - ✓ A constructor is used to initialize the state of the class instance.
  - ✓ To create instances of class from outside the class, declare the constructor public

- Static Constructor

Parameterized Constructor

- A constructor can have parameters. The syntax for the parameters is exactly the same as for other methods.
- A constructor can be overloaded.

Example

```
class Class1
  {
    int Id;
    string Name;

    public Class1()
              { Id=28;    Name="Nemo"; }
    public Class1(int val)
               { Id=val;   Name="Nemo"; }
    public Class1(String name)
              { Name=name;           }

    public void SoundOff()
    { Console.WriteLine("Name {0},   Id {1}", Name,
Id); }
  }
```

```
class Program
  {
    static void Main()
    {
      Class1 a = new Class1(),
             b = new Class1(7),
             c = new Class1("Bill");

      a.SoundOff();
      b.SoundOff();
      c.SoundOff();
    }
  }
```

Default Constructors

- It takes no parameters.
- It has an empty body.

Static Constructor

- Class-level items are initialized
  - ✓ Before any static member is referenced
  - ✓ Before any instance of the class is created
- Static constructors are like instance constructors in the following ways:
  - ✓ The name of the static constructor must be the same as the name of the class.
  - ✓ The constructor cannot return a value.
- Static constructors are unlike instance constructors in the following ways:
  - ✓ Static constructors use the static keyword in the declaration.
  - ✓ There can only be a single static constructor for a class, and it cannot have parameters.
  - ✓ Static constructors cannot have accessibility modifiers.

STATIC CONSTRUCTOR

- A class can have both a static constructor and instance constructors.
- Like static methods, a static constructor cannot access instance members of its class and cannot use the this accessor.
- Cannot explicitly call static constructors from your program. They're called automatically by the system, at some time

✓ Before any instance of the class is created

✓ Before any static member of the class is referenced

Example

```
class RandomNumberClass
  {
    private static Random RandomKey;
    static RandomNumberClass()
    {
      RandomKey = new Random();
    }

    public int GetRandomNumber()
    {
      return RandomKey.Next();
    }
  }
```

```
class Program
  {
    static void Main()
    {
      RandomNumberClass a = new
RandomNumberClass();
      RandomNumberClass b = new
RandomNumberClass();


      Console.WriteLine("Next Random #:
{0}", a.GetRandomNumber());
      Console.WriteLine("Next Random #:
{0}", b.GetRandomNumber());
    }
  }
```