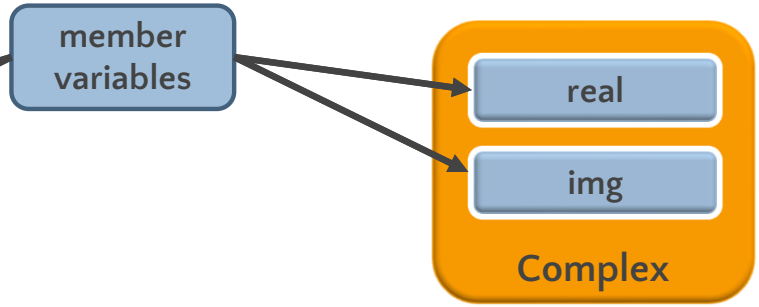
structure & enum

Structure

- Structure is a User Defined Data Type contains a collection of related data variables
- Structure is Value type Data Type
- Declare structure data type (*outside the class Program*)
  - member variables

```
struct complex
{
  public float real;
  public float img;
}
```

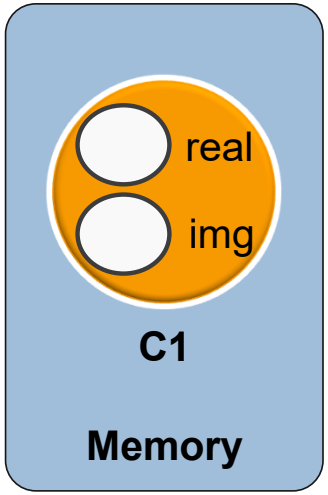


Structure

- Declare structure variable

```
complex c1;
```

Data Type — Variable Name



- Initialization Structure Variable
  - Initialize its *All* members

```
c1.real = 10.0f;
c1.img = 10.5f;
```

  - using ***new*** keyword (initialize members to their default values 0, null , false)

```
complex c2 = new complex();
```

Structure member methods

- Definition
  - Within the structure definition

```
struct complex
{
  public float real;
  public float img;
  public void Display()
  {
   Console.WriteLine($"real={real} \t img={img}");
  }
}
```

- Calling member Method

```
complex C2 = new complex();
C2.Dispaly();
```

Structure member methods

- Constructors
  - Special method its name is the name of the structure With no return
  - Called only one time at variable creation
  - Used for initialization member variables
  - ***Default Constructor***

```
complex c2 = new complex();
```

    - Automatically created and added to structure by the compiler
      - Initialize member variable to default values
    - Can't be added by developer (added by developer in C# 10.0+ (**.NET 6+**))

Structure member methods

- Constructor
  - ***Constructors with parameters***
    - Declared with the structure – member method
    - Overloading constructor
    - Calling Constructors with parameters

```
complex c2 = new complex(10);
```

```
complex c3 = new complex(10,15);
```

```
struct complex
{
 public float real;
 public float img;
 public complex (float x)
 {
   real = img = x;
 }
 public complex (float x, float y)
 {
   real=x;
    img = y;
 }
 public void Display()
 {
        …
 }
}
```

Structure

- Passing and returning structure(*value type*) to method
  - Ex: Addcomplex method

```
static complex AddComplex(complex c1, complex c2)
{
 complex total;
 total.real = c1.real + c2.real;
 total.img = c1.img + c2.img;
 return total;
}
```

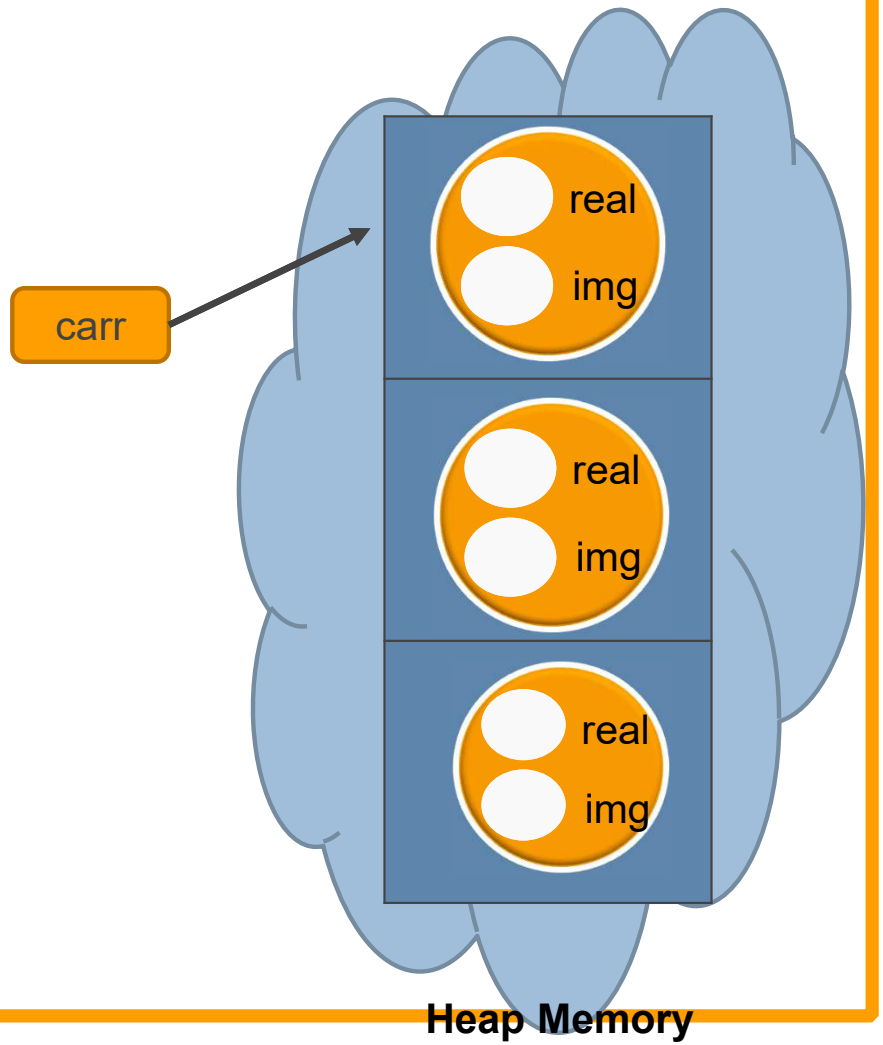
Array of Structures

- Declare Array reference

```
complex[] carr = new complex[3];
```

- Access structure member variables within an array

```
carr[0].real=15.7f;
carr[0].img=20.7f;
```



Array of Structures

- Initialization of array of structures

```
complex[] arr = new complex[3];
```

```
complex[] arr = new complex[]
            {
                new complex(10,15),
                new complex(),
                new complex(5,7),
            };
```

Enum

- Value type data type contains a set of named constants that represent integer values
  - Ex:
    - Days of the week (Sunday , Monday,… etc.)
    - Colors (Red , White,… etc.)
    - Gender (Male , Female)
- Declaring Enum data type

```
enum days
{
  Sat,
  Sun
}
```

```
enum Gender
{
   male,
   female
}
```

Enum

- Values of enum
  - First constant value equal to ***0*** otherwise state it

```
enum Day { Sat=1, Sun, Mon, Tue, Wed, Thu, Fri }
```

- Declare and initialization of Enum variable

```
days d = days.Sat;
days d2 =(days) 2; // sun
days d3 = 0; // 0
```

Assignment

- Write a program to add, subtract two complex using functions
- Add array of Structures of employee to menu program
  - In New
    - Add all employees
  - In Display
    - Display all employees

struct
Employee

Int ID;
String Name;
Float Salary;
gender g;

struct
Complex

float real;
float img;

Object Oriented Programming Using C#

Object Oriented Programming Using C#

Course Content

- Object Oriented Concepts and Terminologies
- Class :
  - Class members and access modifiers
  - Class constructors (overloading)
  - Array of Objects
  - static Modifier and Extension Method
- Polymorphism : Operator Overloading
- Inheritance I
- Inheritance II: Abstract class and Interface
- Association, Aggregation and Composition
- Exception Handling

Course Content

- Collections and Generics
- Delegates
- Anonymous Function and Lambda Expression
- Advanced Topics

Object Oriented Concepts and Terminologies

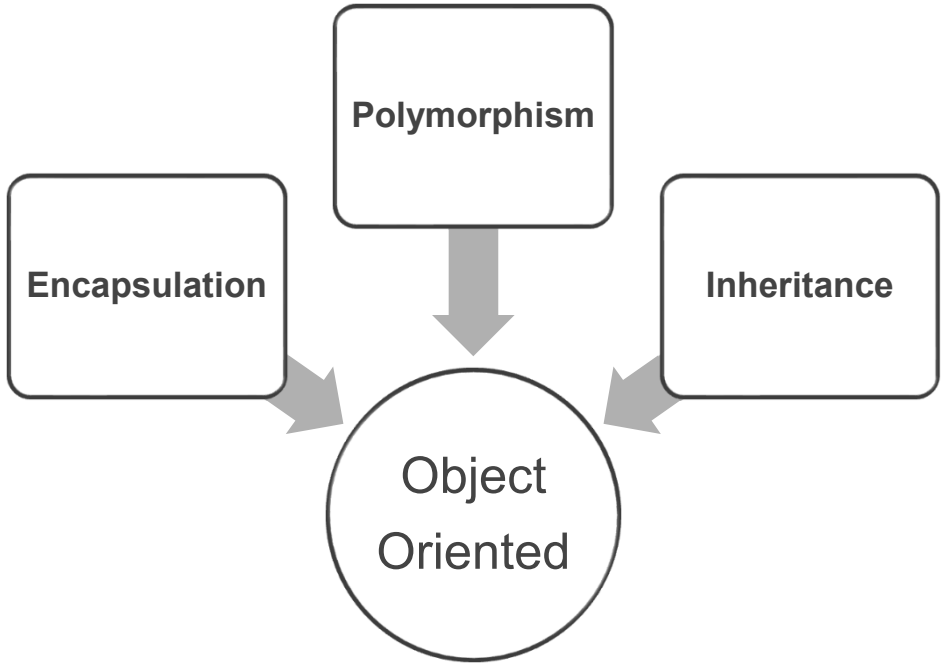
Programming Techniques

- Machine Language 50s-60s
- Structured (Modular) Programming 60s-70s
  - Ex: C , Basic , Fortran
- Object Oriented programming 90s
  - Ex: C++ , Java , C#

Object Oriented Paradigms

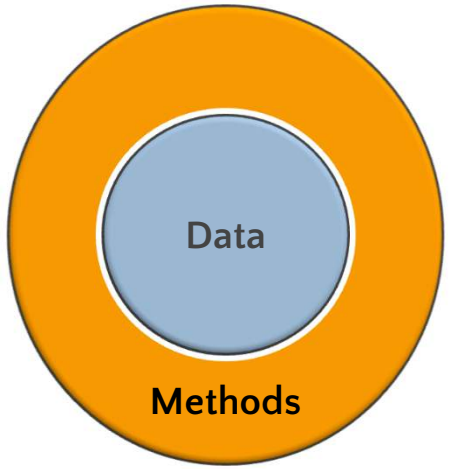
- Encapsulation
- Polymorphism
- Inheritance

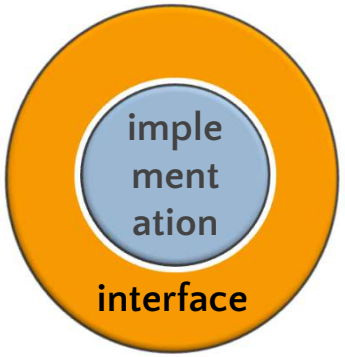
- Abstraction
- Composition
- Association



Encapsulation

- One Capsule contains
  - Data (Attributes or Properties)
  - Methods (Behavior)
- This capsule called ***Class***
  - Logically
    - Interface (what is visible to other classes)
    - Implementation
- Preventing data Access from outside the class ***Data Hiding***

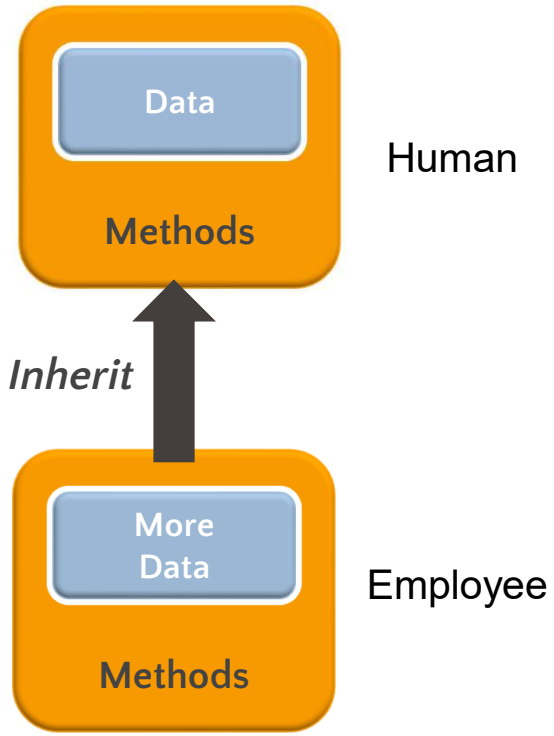




Inheritance

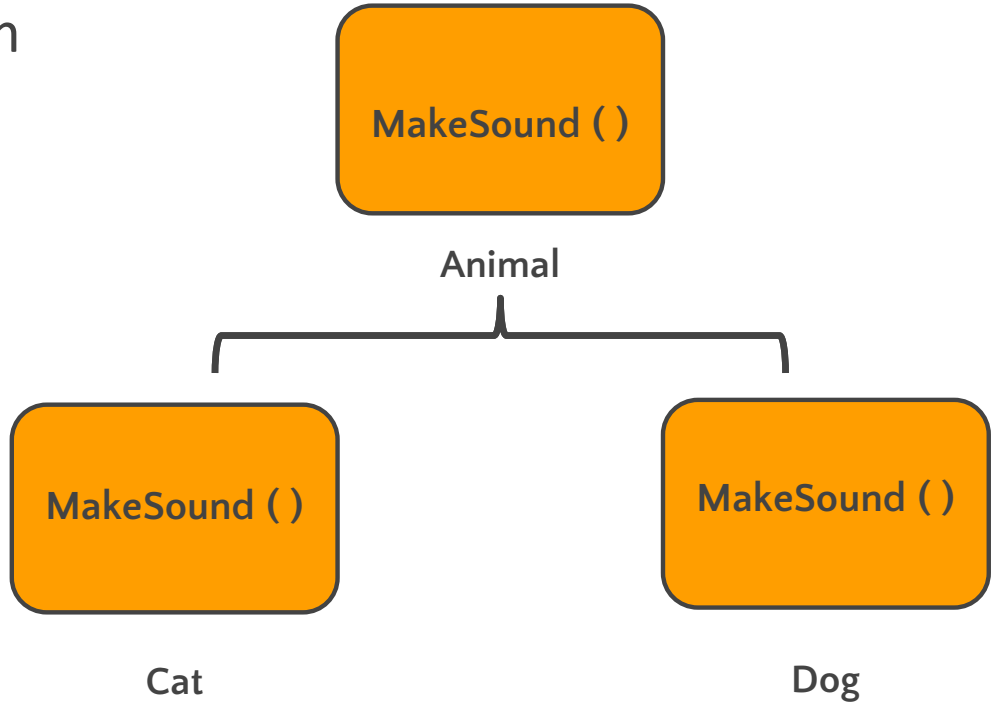
- The capability of creating new class from existing Class
- Used when *more details* are needed

  (extend the class)
- Achieve *Reusability* by reuse existing code (class)
- ***Is-a*** Relation



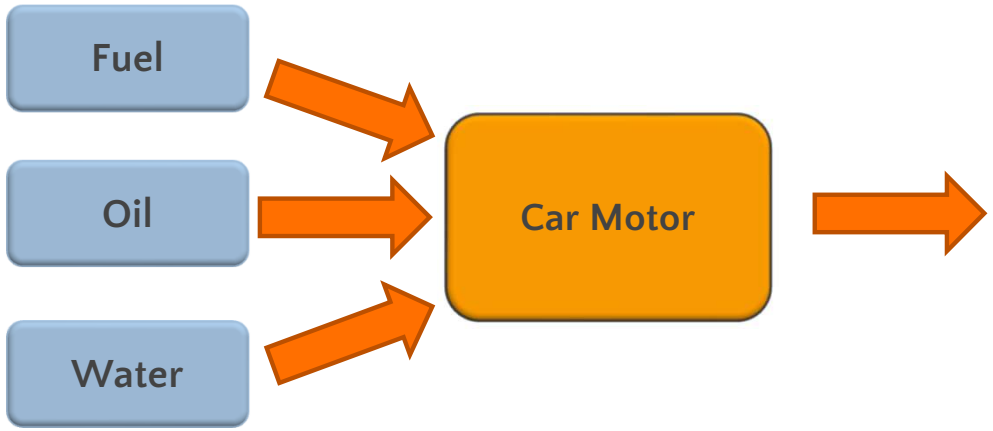
Polymorphism

- Latin word meaning many-forms or many-shapes
  - Ex :MakeSound Behavior
- Compile-time polymorphism
  - Method overloading
- Run-time polymorphism



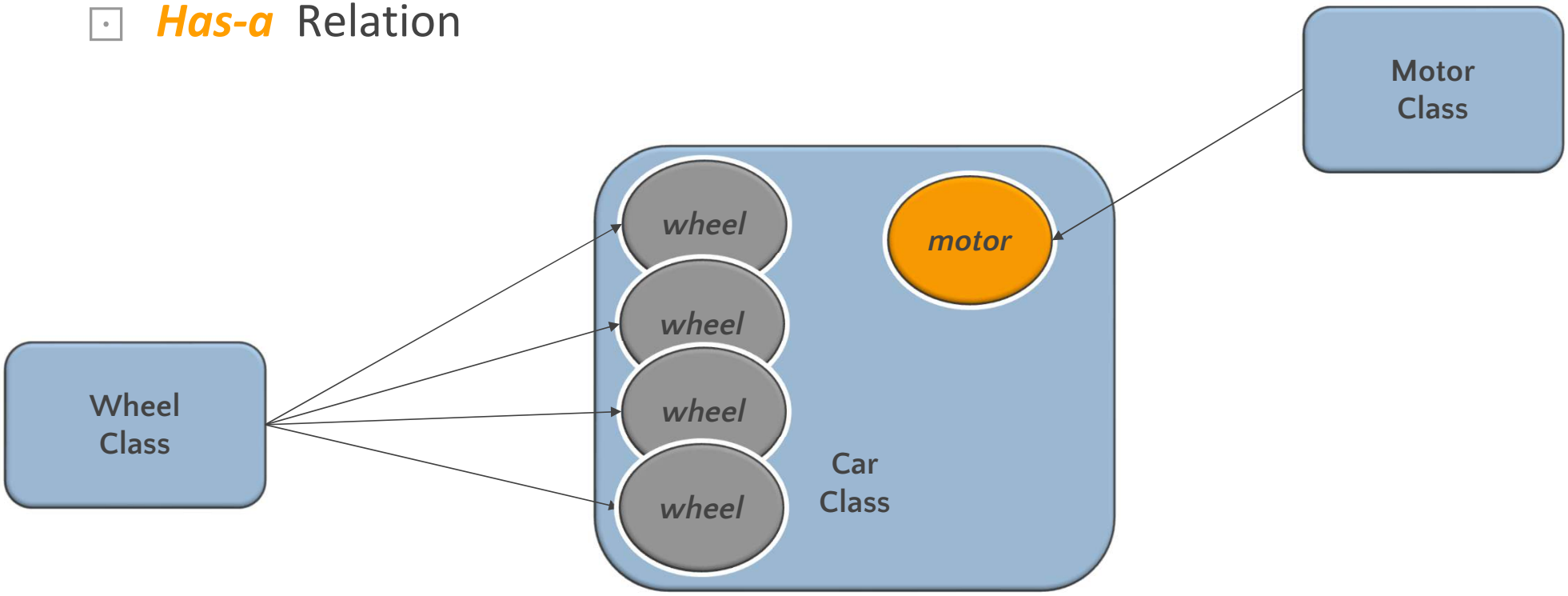
Abstraction

- Hiding internal Details from the class user
  - Ex: Car Motor (how does it work internally)
- Design a class contains only guidelines(abstract class)



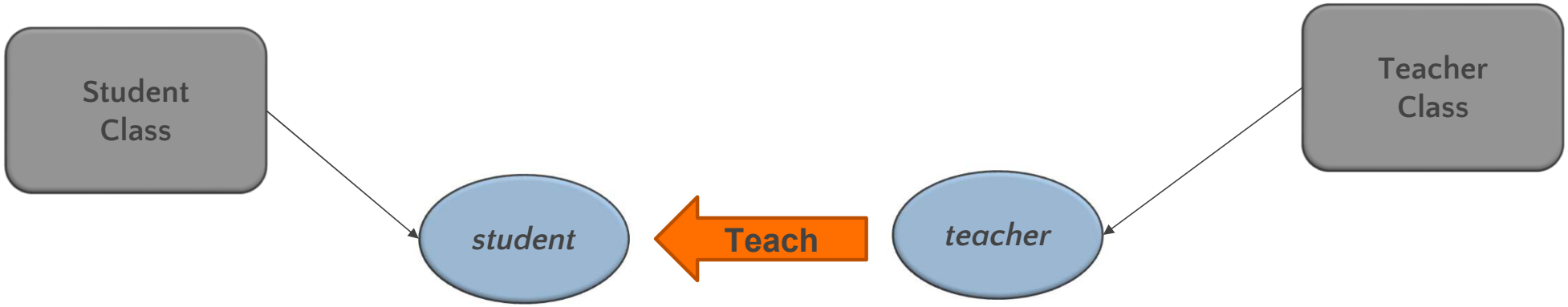
Composition

- Describe a relation between a class and object(s) (variables) from other classes
- ***Has-a*** Relation



Association

- Relation between two independent object



Class members and Access modifiers

Class

- A class is Reference type Data type
- Declare class data type
  - Ex: complex

```
class Complex
{
 public float real;
 public float img;
}
```

  - Ex: Employee

```
class Employee
{
 public int id;
 public string name;
 public float salary;
}
```

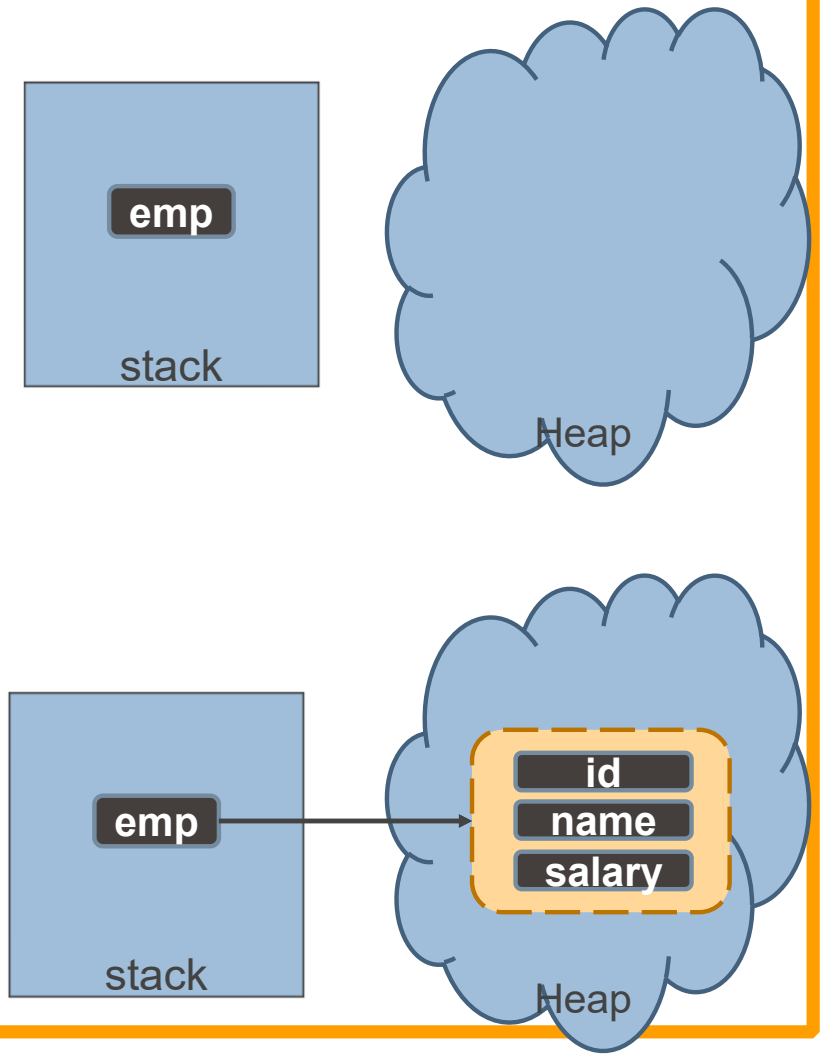
Class

- Declare a reference of class type

```
employee emp;
```

- Instantiate (creating) an Object

```
êņřnêxemployee
///or
employee emp = new employee();
```



Class Members

- ***Member variables*** (field /state)
  - Ex: real, img → Complex
  - Ex: id, name, salary →Employee
- ***Member methods*** (Actions / Behavior)
  - Ex: Display method

Members

```
class Employee
{
 public int id;
 public string name;
 public float salary;

 public void Display()
  {
    Console.WriteLine(id);
    Console.WriteLine(name);
  Console.WriteLine(salary);
  }
}
```

Class Members

- Instance Members
  - Called using reference to object

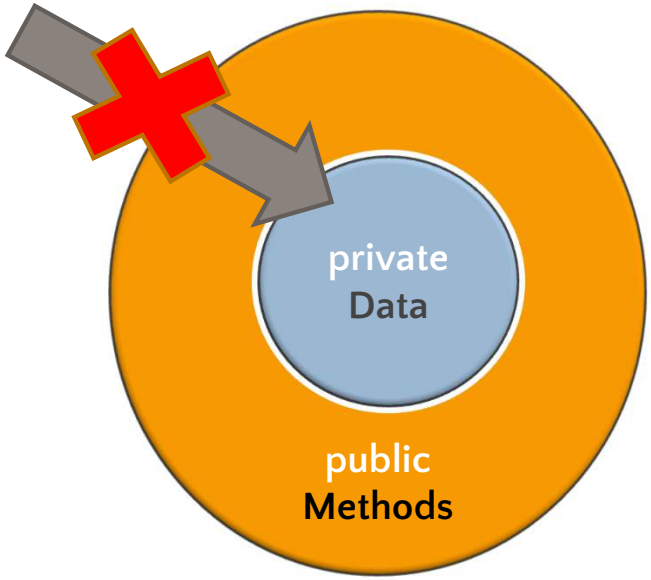
```
employee emp = new employee();
emp.id=10;
emp.Display();
```

- Static Members
  - Called using the class name

```
int x=100;
Console.WriteLine(x);
```

Encapsulation (data –hiding)

- Access Modifiers
  - Hiding data achieved by make it ***private*** preventing access from outside the class
    - Enforce some logic
    - Ex: age must be greater than 0
  - The only way for access private members achieved through public members

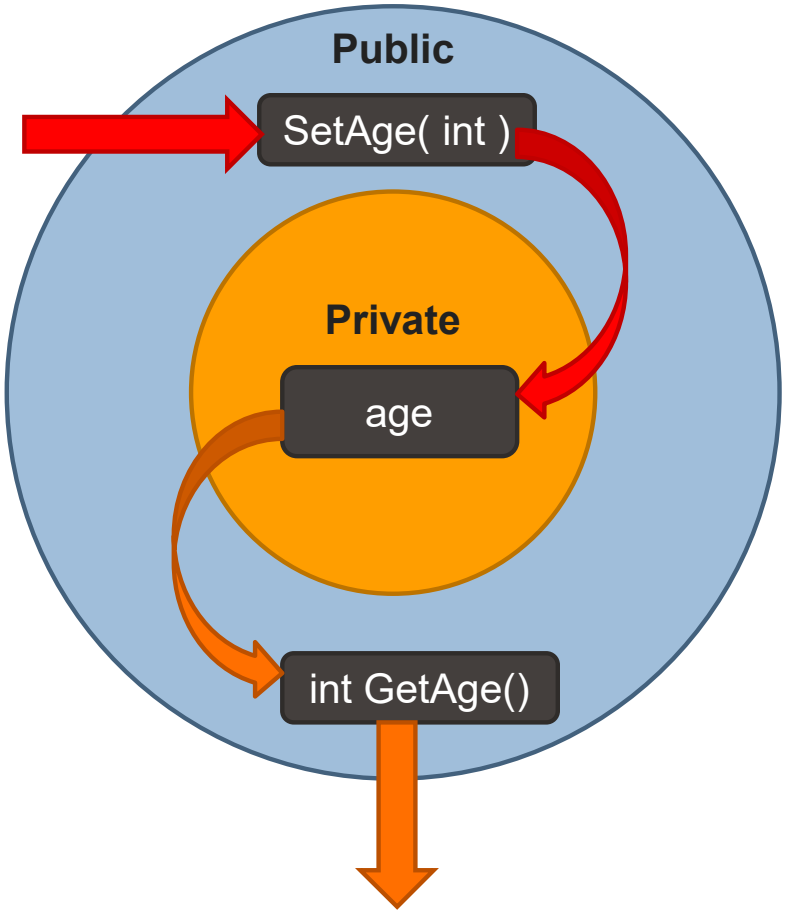


Encapsulation (data –hiding)

- Access modifiers
  - Public
    - Accessible from anywhere in the program
  - Private
    - Default for class members
    - Accessible from within the class Only
  - Protected
    - Explained Later in Inheritance
  - Internal
    - Explained Later in Class Library

Access Private Members

```
class employee
{
  private int age;
  public int GetAge()
  {
        return age;
  }
  public void SetAge(int a)
  {
        if((a>18)&&(age<60))
              age = a;
        else
              age=18;
  }
}
```



Class Members

- Member Properties
  - Explicitly
    - Read & Write
    - Read only (without set)
    - Write only (without get)

```
class employee
{
  private int _age;
  public int Age
  {
    set
    {
       _age = value;
    }
    get
    {
       return _age;
    }
  }
}
```

Class Members

- Member Properties
  - Implicitly (Automatic / Auto-Implemented)
    - No backing field

```
class employee
{
  public int ID {set; get; }
}
```

  - Properties can't pass to method as ref or out
  - Why using Automatic Property??
    - Some cases like ***Data-binding***(Windows Forms) / ***Model binding*** (MVC) properties are needed instead of fields

Automatically Implemented properties could be Initialized

```
public int ID {set; get; }=10;
```

Constructor

- Special Method used to Initialize member variables
- Its name like the class name
- Does not have a return type (no void)
- Called only one time per object at object creation
- The class could have many constructor (***overloading***-polymorphism)

```
Employee emp = new Employee();
```

Default Constructor

- Provided by the compiler (if not Defined and no other Constructor defined)
- That constructor initializes instance fields and properties according to the corresponding initializers. If a field or property has no initializer, its value is set to the default value
- Takes no parameters
- Used to initialize the member variables with specific values

```
class Employee
{

 public int id;
 public string name="";
 public float salary;
 public Employee()
 {
 }

}
```

Constructor with Parameter

- Takes parameters
- Used to initialize the member variables with given values
- When defined the compiler stop providing default Constructor

```
Employee emp = new Employee(10,”Aly”,10000);
Employee emp2 = new Employee(20,”Ahmed”,20000);
```

```
class Employee
{
 public int id;
 public string name;
 public float salary;
 public Employee(int ID,
   string Name, float Salary)
  {
    id=ID;
    name=Name;
    salary=Salary;
  }

}
```

Leaving member variables uninitialized in constructor(s) making them initialized with default values by the compiler

Object Initializer

- Instantiate An Object (create an Object)
  - Through Constructor

```
Employee emp = new Employee();
Employee emp2 = new Employee(20,”Ahmed”,20000);
```

  - Through Object Initializer
    - Default constructor Called first then setting member variable

```
Employee emp = new Employee{id=20 ,name=“Ahmed”, salary=20000};
```

Class Members

- Member Properties
  - **init** only Setter (C# 9)
    - Like read only properties (without no set) except it can be set at object initializer only

```
Employee emp = new
Employee{Age=30};
```

```
class Employee
{
  private int _age;
  public int Age
  {
    init
    {
       _age = value;
    }
    get
    {
       return _age;
    }
  }
}
```

```
class employee
{
  …
  public int ID {get; init; }
}
```

this Reference

- Current Object

```
public Employee(int id ,string name
,float salary)
{
        //x = x; warning
        id = id;
        this. name = name;
        this. salary = salary;
}
```

- Chaining Constructor

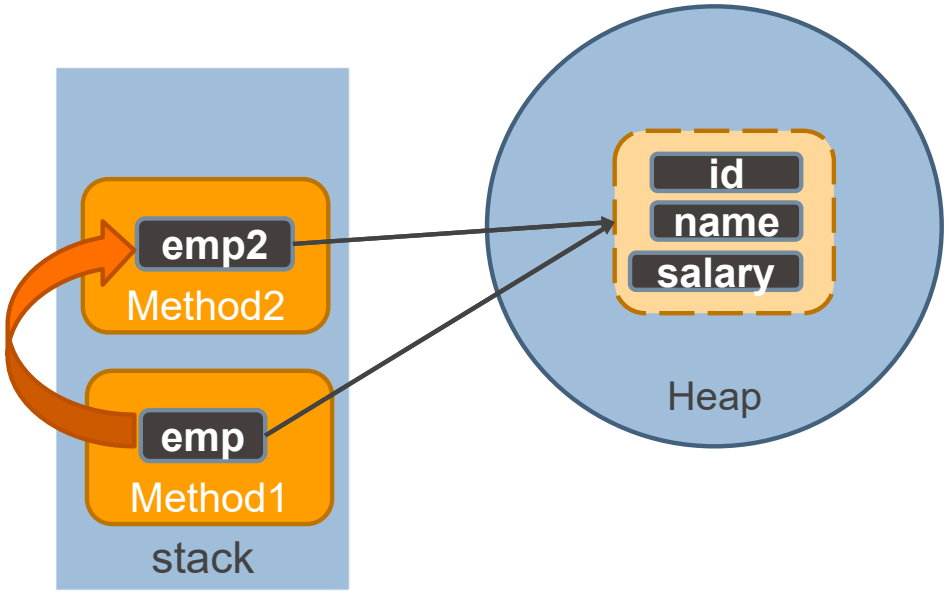
```
class myPoint
{
  public myPoint(): this(0, 0)
  {
   //X=0; y=0;
  }
  public myPoint(int x, int y)
  {
    X = x;
    Y = y;
  }

  public int X { get; set; }
  public int Y { get; set; }

}
```

Reference As Method parameter

```
class Employee
{
        …
}
```

```
řụčľîçŇêţḥộđ
   Employee emp;
   emp = new Employee{salary=3000};
   Method2(emp);
```

```
public Method2(Employee emp2)
{
  emp2.salary=4000;
}
```



Class Access Modifiers

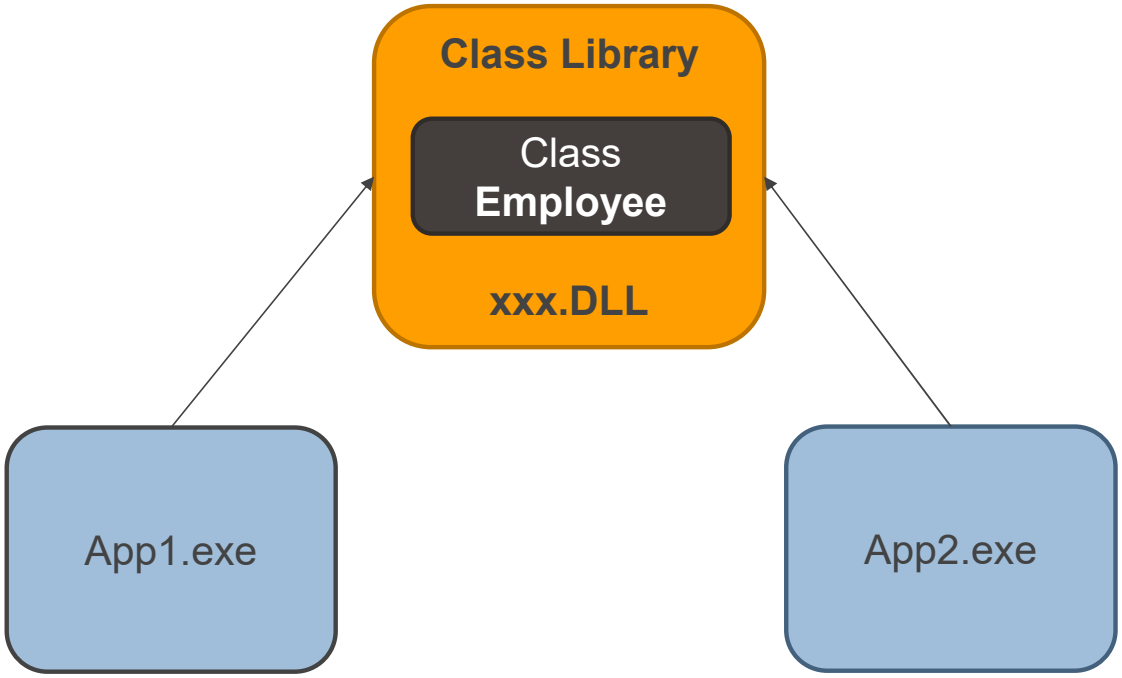
- Public
  - Could be used from anywhere
- Private
  - Mainly used with inner class (class within class)
- Internal
  - Default for a class
  - Could be used within the same assembly (EXE /DLL)

Namespace

- Container for related data types
- Assembly (exe or DLL) could contain One namespace (at least) or more
- Namespace could contain namespace(s)
- For use a data type contained in namespace other than current namespace
  - Full name of data type *namespace* **.** *DatatypeName*
  - Using namespace;

Class Library

- A class library is used to enable sharing Data Types (ex: classes)among applications
- Demo
  - Creating class library
  - Using class library



Class diagram

- Demo

Choosing Between Class and Struct

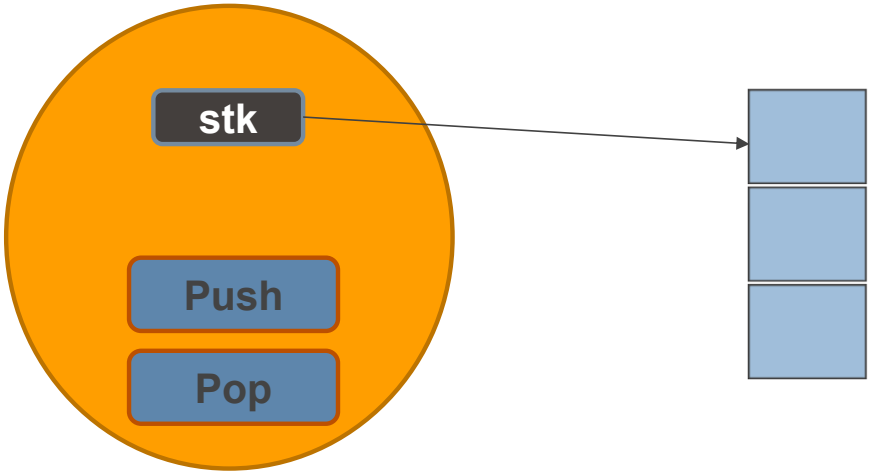
- Struct is value type (stored in stack)
- Class is reference type (stored in heap)
- Passing reference to method or assignment copy only the reference ,whereas Passing value to method or assignment copy the entire value (passing reference is cheaper)
- CONSIDER defining a struct instead of a class if instances of the type are small and commonly short-lived or are commonly embedded in other objects

Choosing Between Class and Struct

- AVOID defining a struct unless the type has all of the following characteristics:
  - It logically represents a single value, similar to primitive types (int , double , etc.).
  - It has an instance size under 16 bytes.
  - It is immutable.
  - It will not have to be boxed frequently

Assignment

- Design a class that represent a Stack Data Structure that contain
  - Data
    - Array of integers (to store values)
    - Size (init property)
    - Top_of_Stack
  - Actions
    - push
    - pop



Assignment

- Design a class represents Employee
  - Name
  - ID
  - Salary
  - DispalyData() method
  - Gender enum (init property)
  - Age as a property
- Adding the employee class to class library and used in menu program