Class (cont.)

- *static* modifier
  - Static class (ex: math class)
    - Class members
  - Static fields
  - Static properties

```
public class employee
    {
        public static int x=0;
    }
```

Class (cont.)

```
public class employee
{
    public static int x;
    static employee() // no access modifiers
    {
        x=0;
    }
}
```

- Static methods
  - Accessing normal variable
  - Static local variable
  - Static constructors
    - Initialization static member variables
- *const* keyword
  - (declaration – Design time)
- *readonly* keyword
  - (declaration or constructor – Run time)
- *static readonly* (static constructor)

Slide 69

w23

```
class train
{
    readonly datetime train_departure_time;
    public train()
    {
        train_departure_time=datetime.now;
    }
}

train Tr=new train();
```

wael, 11/2/2017

Class (cont.)

- Methods
  - Instance vs static methods
- *partial* keyword

w19

w20

- Array of objects
  - Default constructor
  - Other constructors
- Finalizer
  - Distructor

Slide 70

w19 public static void UseParams(params int[] list)

```
        UseParams(1, 2, 3, 4);
        int[] myIntArray = { 5, 6, 7, 8, 9 };
        UseParams(myIntArray);
```

wael, 1/13/2017

w20 operator + for complex

public static Complex operator +(Complex c1, Complex c2)

wael, 1/13/2017

Class (cont.)

- Overload operator Polymorphism

```
public static complex operator + (complex c1, complex c2)
{
}
```

Operators	Description
+, -, !, ~, ++, --	These unary operators take one operand and can be overloaded
+, -, *, /, %	These binary operators take one operand and can be overloaded.
==, !=, <, >, <=, >=	The comparison operators can be overloaded
&&, ||	The conditional logical operators cannot be overloaded directly.
+=, -=, *=, /=, %=	The assignment operators cannot be overloaded.
=, ., ?:, ->, new, is, sizeof, typeof	These operators cannot be overloaded.

Class (cont.)

- Indexer

```
public int this[int index]
{
get {   …}
set {   …}
}
```

- Finalizers

Anonymous Types

- *var* Keyword (Implicit Local Variable)
- Anonymous Type
  - Read only Properties

Object Initializer

```
var v2 = new { Price = 200f, name = "juice" };
v2.name = "milk"; //error readonly
```

  - Another use of var

```
employee em = new employee { ID = 10, Name = "Ahmed", Salary = 1000f };
em.ID = 2;
```

```
var v = new employee { ID = 10, Name = "Ahmed", Salary = 1000f };
v.ID = 2;
```

Anonymous Types (cont.)

- Change
  - Type
  - Name
  - Oreder
- CLR Generate another type

```
var patent1 =new
{
    Title = "Bifocals",
    YearOfPublication = "1784"
};
var patent2 =new
{
    Title = "Phonograph",
    YearOfPublication = "1877"
};
var patent3 = new
 {
    patent1.Title,
    // Renamed to show property naming.
    Year = patent1.YearOfPublication
}
```

Anonymous Types (cont.)

- Var vs Object
  - Var strongly type
  - Var read only (immutable)
  - Var cant be used as method parameter (Local variable)
  - Anonymous type associated with var contain ToString method override

Anonymous Types (cont.)

- Mainly used in Linq

```
var q = from emp1 in emparr
        select new { emp1.ID, emp1.Name };




foreach(var v3 in q)
{
    Console.WriteLine($"ID={v3.ID}\t Name={v3.Name}");
}
```

```
[] emparr;
emparr = new employee[3]
{
    new employee(),
    new employee(),
    new employee()
};
```

Assignments

- Write assignment for stack overload operator +
- Example of using indexer [int] [string] w22

Slide 77

w22

```
class IndexedNames
  {
    private string[] namelist = new string[size];
    static public int size = 10;
    public IndexedNames()
    {
      for (int i = 0; i < size; i++)
      {
        namelist[i] = "N. A.";
      }
    }

    public string this[int index]
    {
      get
      {
        string tmp;

        if( index >= 0 && index <= size-1 )
        {
          tmp = namelist[index];
        }
        else
        {
          tmp = "";
        }

        return ( tmp );
      }
      set
      {
        if( index >= 0 && index <= size-1 )
        {
          namelist[index] = value;
        }
      }
    }
```

```
public int this[string name]
{
   get
   {
      int index = 0;
      while(index < size)
      {
         if (namelist[index] == name)
         {
          return index;
         }
         index++;
      }
      return index;
   }

}

static void Main(string[] args)
{
   IndexedNames names = new IndexedNames();
   names[0] = "Zara";
   names[1] = "Riz";
   names[2] = "Nuha";
   names[3] = "Asif";
   names[4] = "Davinder";
   names[5] = "Sunil";
   names[6] = "Rubic";

   //using the first indexer with int parameter
   for (int i = 0; i < IndexedNames.size; i++)
   {
      Console.WriteLine(names[i]);
   }

   //using the second indexer with the string parameter
```

```
            Console.WriteLine(names["Nuha"]);
            Console.ReadKey();
        }
    }
}
```

wael, 10/30/2017