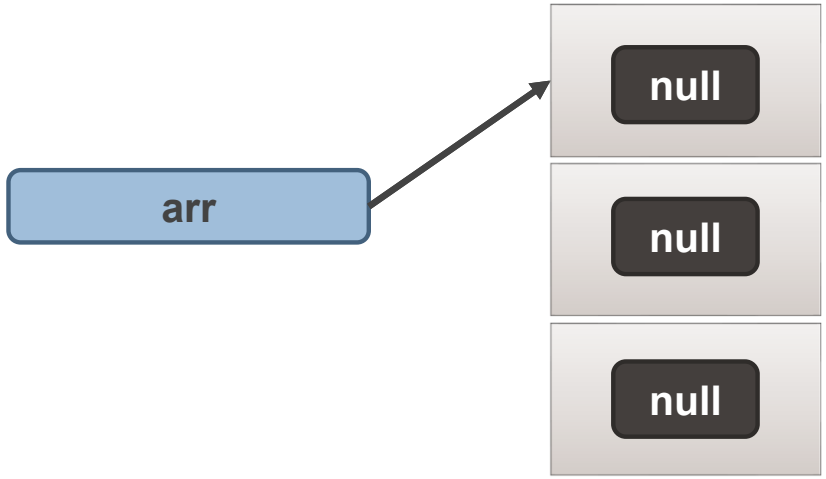
Array of Objects

- arrays of reference types are allocated out-of-line, meaning the array elements are just references to instances of the reference type residing on the heap
- Since the class is reference type so creating an array of the class type is creating an array of references initialized with null

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



- Demo

Array of Objects

- Each reference in array must initialized with an Object
  - Individual assignment

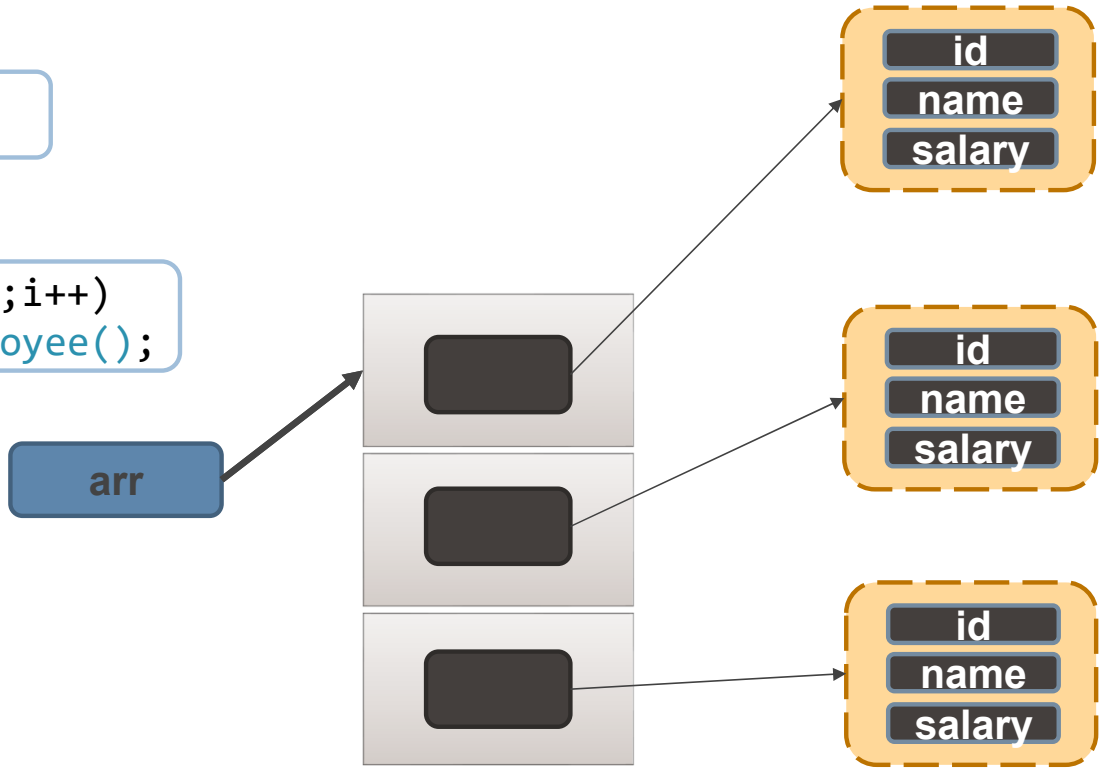
```
arr[1] = new Employee();
```

  - For loop

```
For(int i=0;i<arr.length;i++)
        arr[i] = new Employee();
```

  - Array initializer

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



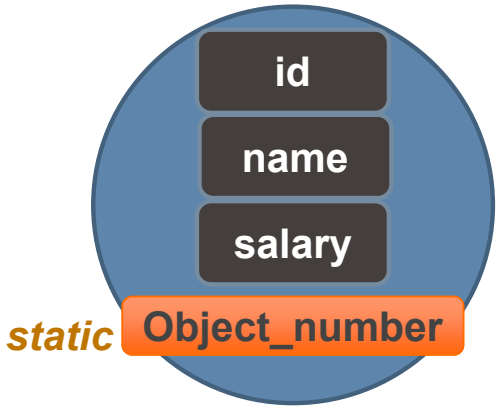
Assignment

- Modify menu program using array of 3 Employees
  - New
    - Adding 3 new employees
  - Display
    - Display 3 employees Data

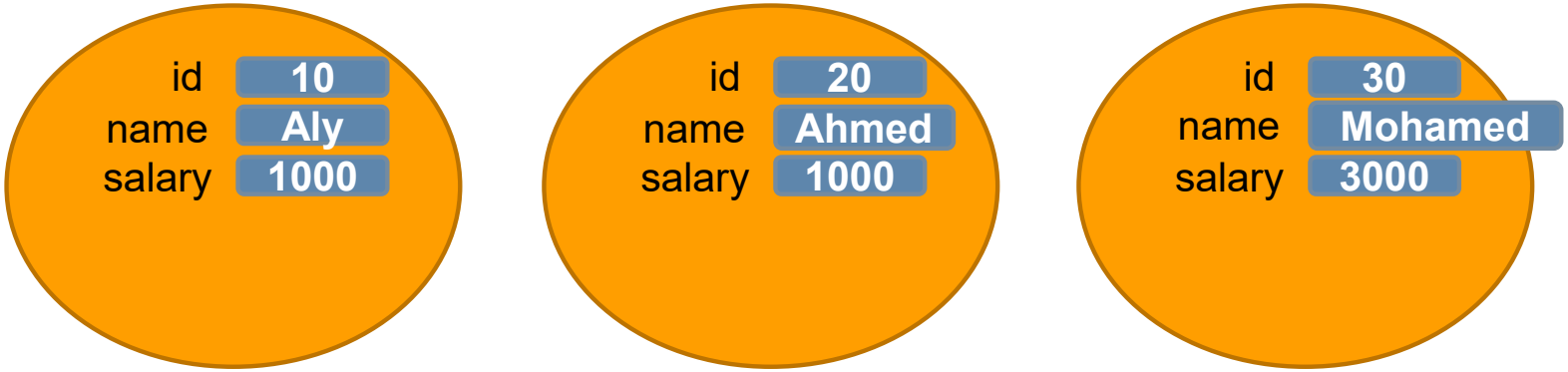
Static Modifier

Static Modifier

- Static member variable
  - Single copy of the variable Reside within the class
  - Ex: number of Objects



Class Employee



Emp1 Emp2 Emp3

Static Modifier

- Static member methods / properties
  - Access only static variables
- Static Constructor
  - Used for initialize static fields
  - No access modifiers allowed

Static Modifier

- Static class
  - Used as a container for related static methods
    - Ex: Math class
    - Ex: Console class
  - Creating an Object of static class is NOT allowed
  - Used as a Container for Extension Method

Extension Methods

- Used for extend the functionality of a class
- Used in LINQ

```
class ExClass
 {
   public static int ConvertStringToInt(string s)
    {
            int r = int.Parse(s);
            return r;
    }
 }
```

```
string s =10;
int x= ExClass.ConvertStringToInt(s);
```

Extension Methods

- Defining and calling Extension Method

```
static class ExClass
 {
   public static int ConvertStringToInt(this string s)
    {
            int r = int.Parse(s);
            return r;
    }
 }
```

```
string s =10;
int x= s.ConvertStringToInt();
```

Other Modifiers

- *const* keyword
  - Used for constant variable
  - EX: Math.PI
  - Initialized on Declaration (Design Time)
  - Used through class name (like static)
- *readonly* keyword
  - Once its value initialized it could not be changed
  - Ex: flight Takeoff Time
  - Initialized on Constructor (Run Time) or object initializer
- partial keyword
  - Used to indicate that the class defined in separate Files

Assignment

- In Menu Program:
- Modify Employee Class to achieve
  - Employee ID Auto Incremented (like primary key in database)
  - modify Gender as *readonly* field
- Write an Extension Method that extended an array of Employees to print its elements

Overload Operators (Design-time) Polymorphism

Overload Operator

- Redefine operator to take parameters other than it was predefined with
  - Ex: operator + takes 2 complex

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

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

```
complex total= AddComplex(cx,cy);
```

Overload Operator

- Define and calling Operator
  - Operator must be public and static
  - Operator method could not use ref , out , in

```
class complex
{
 public float real;
 public float img;
 public static complex operator +( complex c1,complex c2)
  {
   complex total=new complex() ;
   total.real = c1.real + c2.real;
   total.img = c1.img + c2.img;
   return total;
  }
}
```

```
complex total=cx + cy;
```

Overload Operator

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

Overload Operator

- Operator +=

```
complex c1=new complex { Real=10,Img=10 };
c1 += 5; // c1 = c1 + 5
```

```
 public static complex operator +(complex c1,int x)
{
  c1.Real += x;
  c1.Img += x;
  return c1;
}
```

Overload Operator

- Indexer [ ]
- Indexers provide a natural syntax for accessing elements in a class or struct that encapsulate a collection of values(array, list ,dictionary).
- Could be considered as a special property
  - Ex: access the elements of stk array within Stack class
- Declare and use indexer

```
Stack S1=new Stack();
S1[2]=10;
int x=S1[0];
```

```
class Stack
{
  …
 public int this[int index]
 {
   get { return stk[index];}
  □□ṣêʧ□□□ṣʧl□îŋđêẏ□□ŵăľụê□□
   }
}
```

Type Conversion

- Implicit Casting


```
int x=100;
float f=x;
```

Type Conversion

- Explicit Casting


```
float f=15.5f;
int x=(int) f;
```

Type Conversion

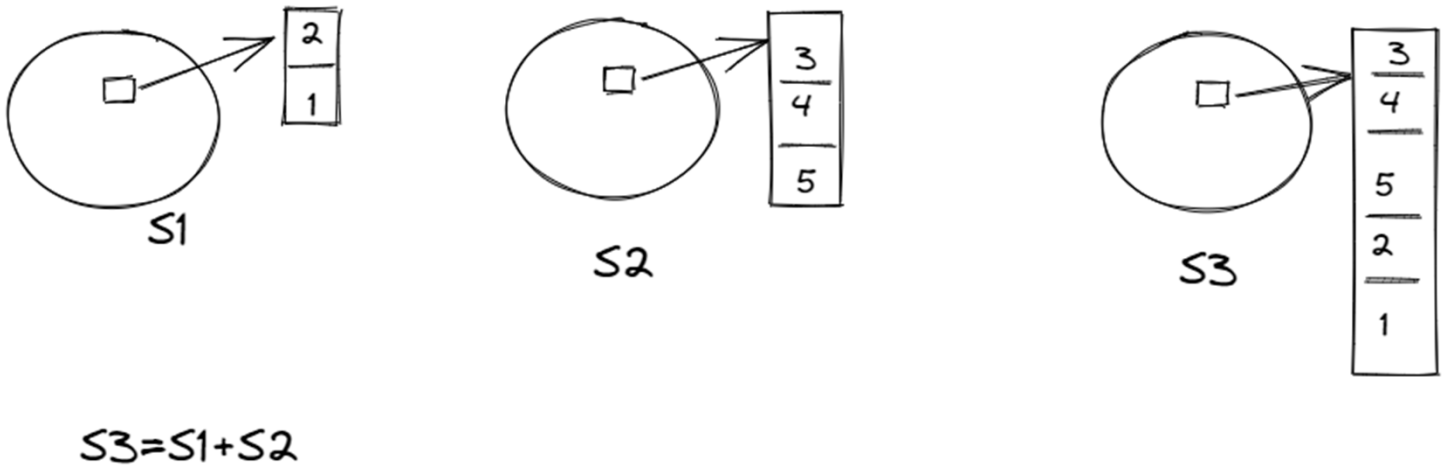
- User-defined Casting

```
class test1
 {
      public int x;
      public test1()
      { }
      public test1(int l)
      { x = l; }
      public static implicit operator int(test1 t)
      { return t.x; }
      public static explicit operator test1(int z)
      { return new test1(z); }
 }
```

```
test1 t = new test1();
int g = t;             //Implicit
test1 t2 = (test1)g; //Expilict
```

Assignment

- Modify class stack to add overload operator +



- Adding indexer [] to stack class for retrieve data Only(reading only)
- User defined casting between array and stack