

# ESc101 : Fundamental of Computing

I Semester 2008-09

## Lecture 25

### Object Oriented programming

- Static and non-static members of a class : Revision
- Arrays

**Note :** Arrays **WILL NOT** be part of the second mid semester exam. But whatever we discussed today on arrays will be required in the lab assignment in the week following the mid-semester break. So feel relaxed ... :-).

## Complete picture of a class

```
class class_name
```

```
{
```

```
    public static double pi=3.1426 ;  
    private static int count =0 ;
```

```
        ●  
        ●  
        ●
```

```
    public static void method1()  
    { .... }  
    private static int method2(int n)  
    { .... }
```

Static attributes  
and  
Static methods

```
    private double balance ;  
    public int id_num ;
```

```
        ●  
        ●  
        ●
```

```
    private void methodk()  
    { .... }  
    public int methodj(int n)  
    { .... }
```

Non-static attributes  
and  
non-static methods

```
}
```

## **Static and non-static methods**

- A static method is invoked on behalf of an entire class, not on a specific object. A static method might perform some general task (not specific to an object of the class), whereas, a non-static method is used for accessing, manipulating a specific object.

## Revisit Point class

```
public class Point
{
    double x;
    double y;

    public Point(double x1, double y1)
    {
        x = x1;
        y = y1;
    }

    public double distance_from_origin()
    {
        double dist;
        dist = Math.sqrt(x*x + y*y);
        return dist;
    }
}
```

## Example : Point class

Suppose we want to add more functionality to Point class :

- **Center(Point P,Point Q,Point R)** : Computing center of three points P,Q,R.
- **x\_MirrorImage(??)** : Update the current point such that it becomes the mirror image (with x-axis) of its existing position. For example, (4,5) becomes (4,-5) and (2,-45) becomes (2,45).

## Example : Point class

Suppose we want to add more functionality to Point class :

- **Center(Point P,Point Q,Point R)** : Computing center of three points P,Q,R.
- **x\_MirrorImage(??)** : Update the current point such that it becomes the mirror image (with x-axis) of its existing position. For example, (4,5) becomes (4,-5) and (2,-45) becomes (2,45).

**Question** : Should these methods be static methods or non-static ?

## Example : Point class

Suppose we want to add more functionality to Point class :

- **Center(Point P,Point Q,Point R)** : Computing center of three points P,Q,R.
- **x\_MirrorImage(??)** : Update the current point such that it becomes the mirror image (with x-axis) of its existing position. For example, (4,5) becomes (4,-5) and (2,-45) becomes (2,45).

**Question** : Should these methods be static methods or non-static ?

**Hint** : ponder over slide 3.

## Example : Point class

- **Center()** should be **static** method.

For the reason, read carefully the description of method Center(). If it were designed as a non-static method then **P.Center(A,B,C)** has got nothing to do with the point P as such. So there is no reason we should invoke it as P.Center(A,B,C). If you do so, it would not be an error but it is a very bad programming practice.

However if I had intended Center(Point A, Point B) to be a method which returns the center of the current point and the points A and B, then it should be defined as a non-static method.



## Example : Point class

- **x\_MirrorImage()** should be **non-static** method.

Look at the description of the method **x\_MirrorImage**. It is clear from its description that it manipulate the current point. For example, **P.x\_MirrorImage()** would change the point on which it is called. So it is advisable if we declare it as a non-static method.

## Example : Point class

Hence :

1. **Center()** should be **static** method.
2. **x\_MirrorImage()** should be **non-static** method.

So whenever you design a method in Point class, first ask yourself whether the method if invoked as P.method() involves the current point P in any way. If yes, design it as non-static, otherwise design it as a static method.

## Example : Point class

```
public class Point
{
    double x;
    double y;
    // constructors and the existing method
    // for Point are not shown here due to limited space

    public void x_MirrorImage()
    {
        y = -y;
    }

    public static Point Center(Point P, Point Q)
    {
        Point mid = new Point((P.x+Q.x)/2,(P.y+Q.y)/2);
        return mid;
    }

    public static Point Center(Point P, Point Q, Point R)
    {
        Point mid = new Point((P.x+Q.x+R.x)/3,(P.y+Q.y+R.y)/3);
        return mid;
    }
}
```

## How to invoke a static method, say *method1* ?

- **Within its own class** : you may invoke it in a method directly by calling *method1* and passing arguments if any.
- **Outside its class** : you may invoke it by *class\_name.method1*, where *class\_name* is the class in which *method1* is defined.

**Note :** Though you may invoke a static method as *refer.method1* where *refer* is a reference to an object of the same class (whose member is *method1*), but it is considered a bad programming practice.

## Example implementation

The files **program1.java** and **Geometry** package with altered **Point** class are on the course webpage. Try it out.

## Example2 of Static methods

Suppose you want to create a library of fundamental functions which can be used in your programs , so design a library of static methods.

```
public class My_math
{
    public static long GCD(long i, long g)
    {
        // write the statements of this method here
    }
    public static boolean IsPrime(long i, long g)
    {
        // write the statements of this method here
    }
    .
    . // add more functions here
}
```

It is meaningless to declare the above methods as non-static.

## Example1 of Static attributes

Recall the example of Bank\_account class where we had to assign different account number to each customer. We could achieve it using static attribute **NextN**. Please refer to the lecture held on Friday 26th September for its significance and implementation details. This example highlights the power of static attributes,

## Example2 of Static attributes

You might like to create a class which has all the constants of math and physics.

```
public class Constants
{
    private static double pi = 3.1426;
    private static double e = 2.31;
    private static double G = 9.8;
    ...

    public static double get_pi()
    { return pi; }
    public static double get_e()
    { return e; }
    ...
}
```

You may use them as **Constants.get\_pi()**, **Constants.get\_e()**, ...



# Arrays

What if you have a large collection of identical data items which you want to process.

**Examples :**

- Sort  $n$  numbers.
- Compute the diameter of a set of 100 points in 2-D space.
- Compute the smallest enclosing sphere for a set of 10000 points in 3-D space.

What if you have a large collection of identical data items which you want to process.

**Examples :**

- Sort  $n$  numbers.
- Compute the diameter of a set of 100 points in 2-D space.
- Compute the smallest enclosing sphere for a set of 10000 points in 3-D space.

We need an easy way to identify and manipulate large number of variables

## **Arrays offers a solution**

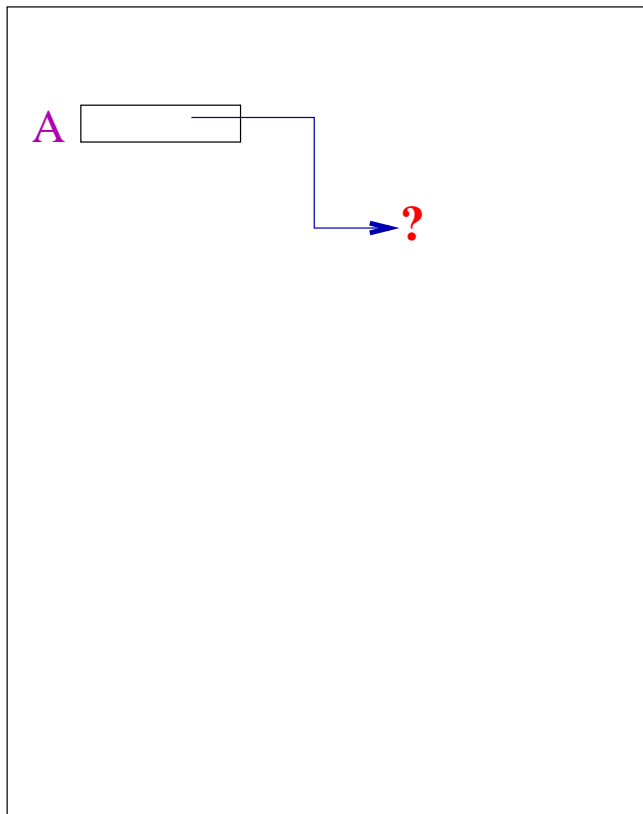
Array : An Object which is an ordered collection of data items. These data items could be

- primitive types.
- references to objects of a class.

## Array : declaration and creation

int[ ] A ;

Memory

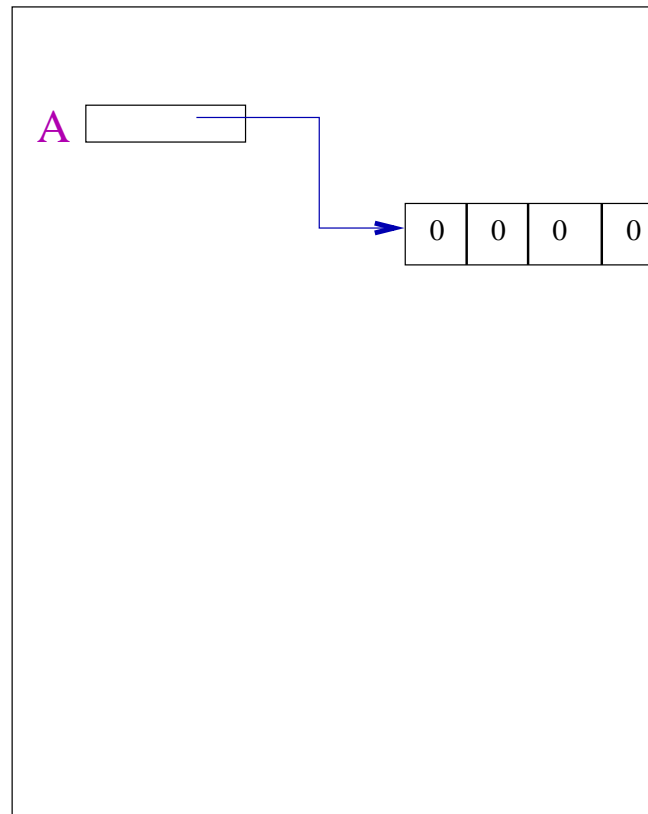


## Array : declaration and creation

```
int[ ] A ;
```

```
A = new int[4];
```

Memory



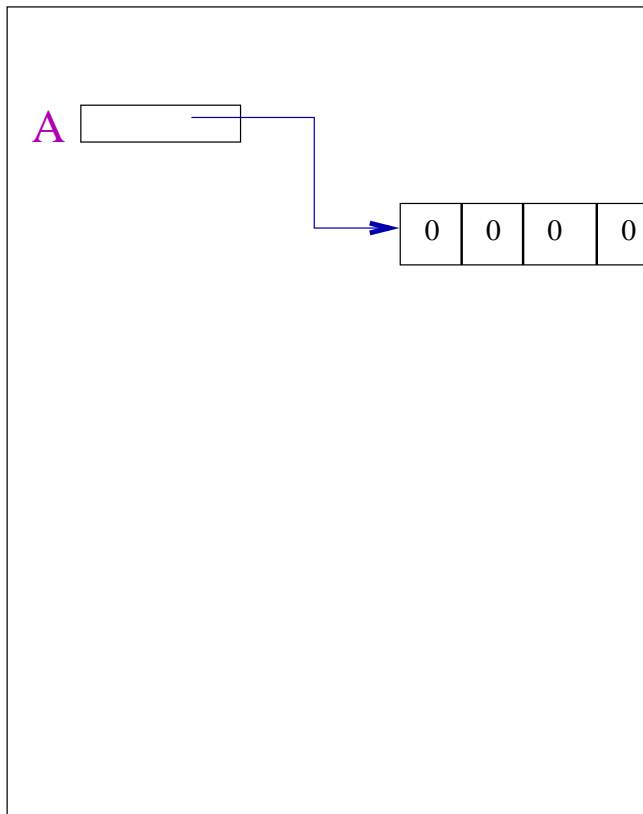
## Array : declaration and creation

int[ ] A ;

A = new int[4];

Expression of type int

Memory



## Manipulating the contents of array $A$

- $A$  has four variables of type `int`
- $A[0]$  represent the first variable,  $A[1]$  the second variable, ...,  $A[3]$  the last variable.
- Each  $A[i]$ , for  $0 \leq i \leq 3$ , can be treated as a single independent variable.



## Array : Accessing and Manipulating elements of an array

```
int[ ] A ;
```

```
A = new int[4];
```

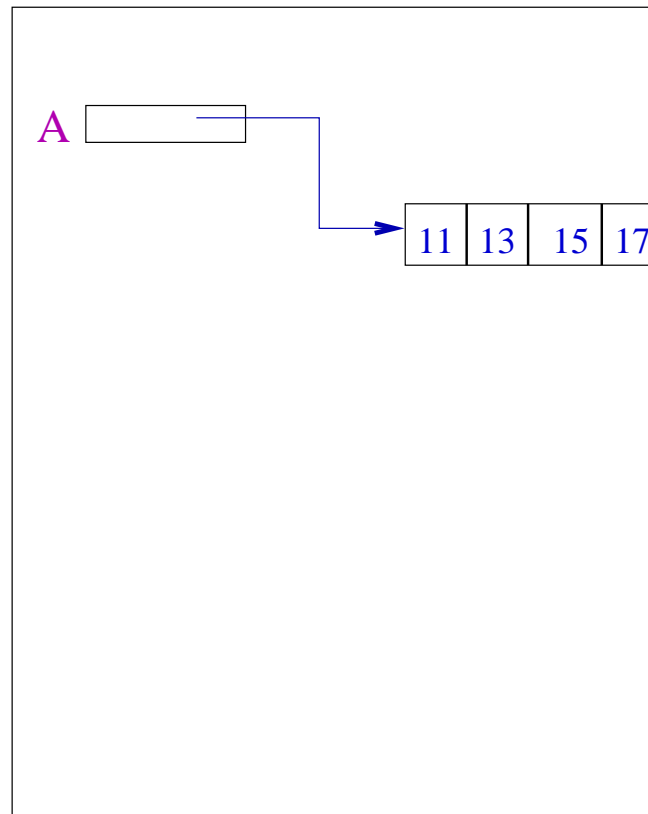
```
A[0] = 11;
```

```
A[1] = 13;
```

```
A[2] = 15;
```

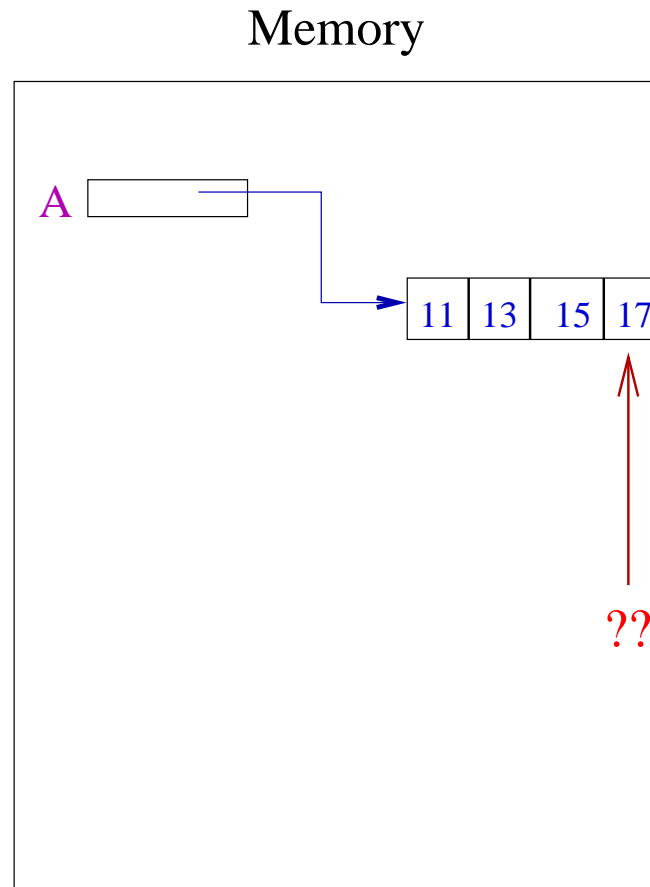
```
A[3] = 17;
```

Memory



## What will be the result of execution of last statement ?

```
int[ ] A ;  
A = new int[4];  
A[0] = 11;  
A[1] = 13;  
A[2] = 15;  
A[3] = 17;  
A [3] = A[2] -A[1]+ 7;
```



## Treat each element of array as a variable

```
int[ ] A ;
```

```
A = new int[4];
```

```
A[0] = 11;
```

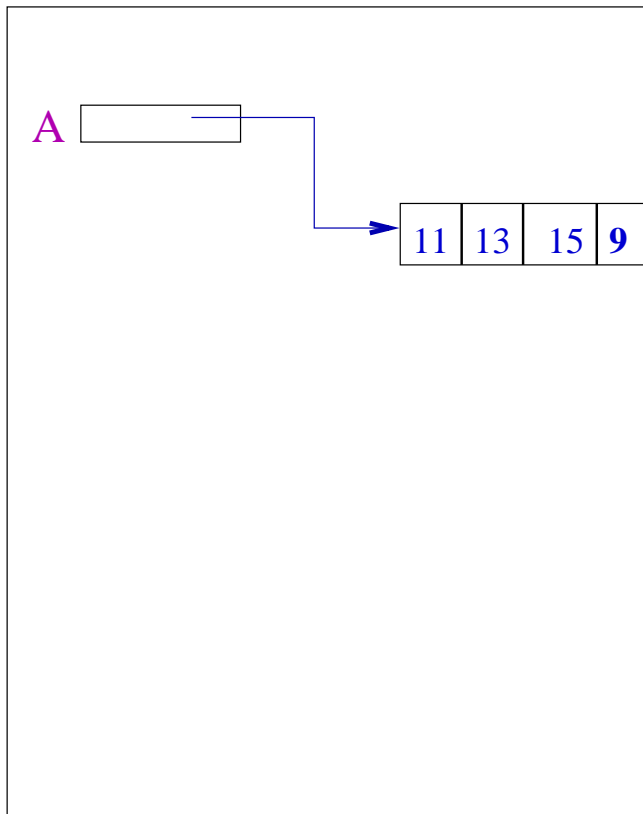
```
A[1] = 13;
```

```
A[2] = 15;
```

```
A[3] = 17;
```

```
A[3] = A[2] - A[1] + 7;
```

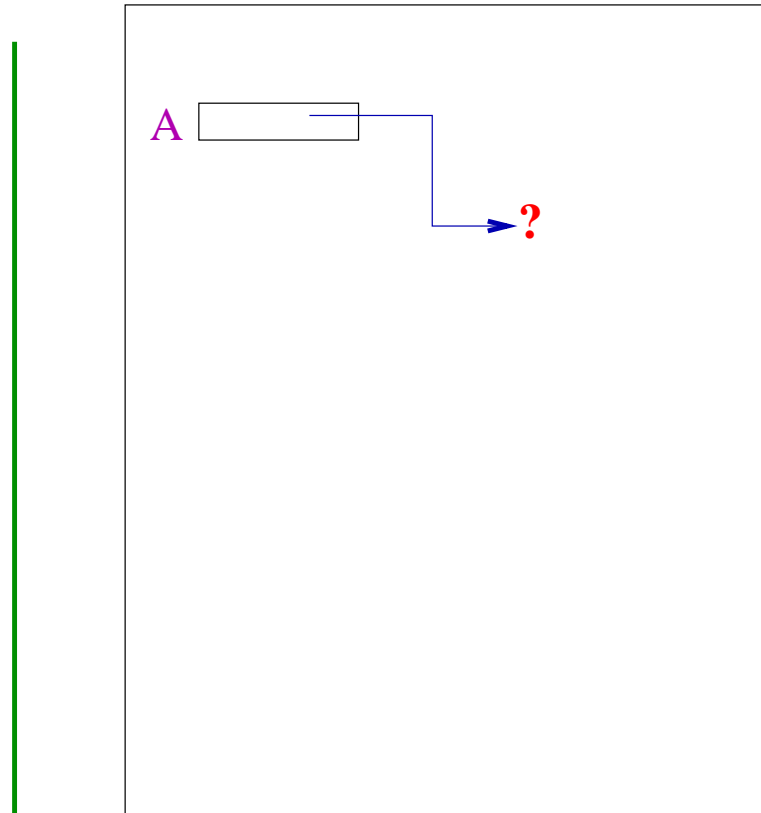
Memory



## Array of Points

Point[ ] A ;

Memory

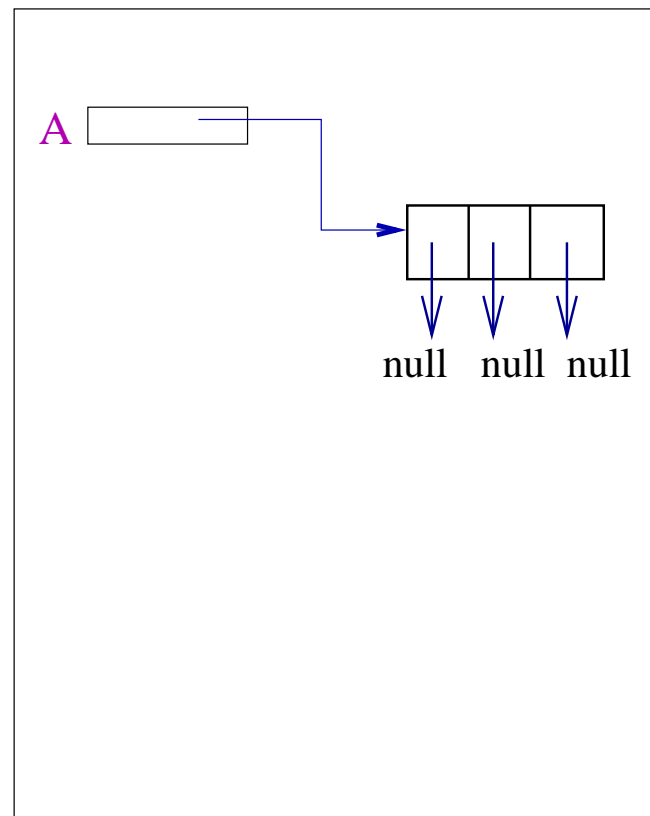


## Array of Points

Point[ ] A ;

A = new Point[3];

Memory



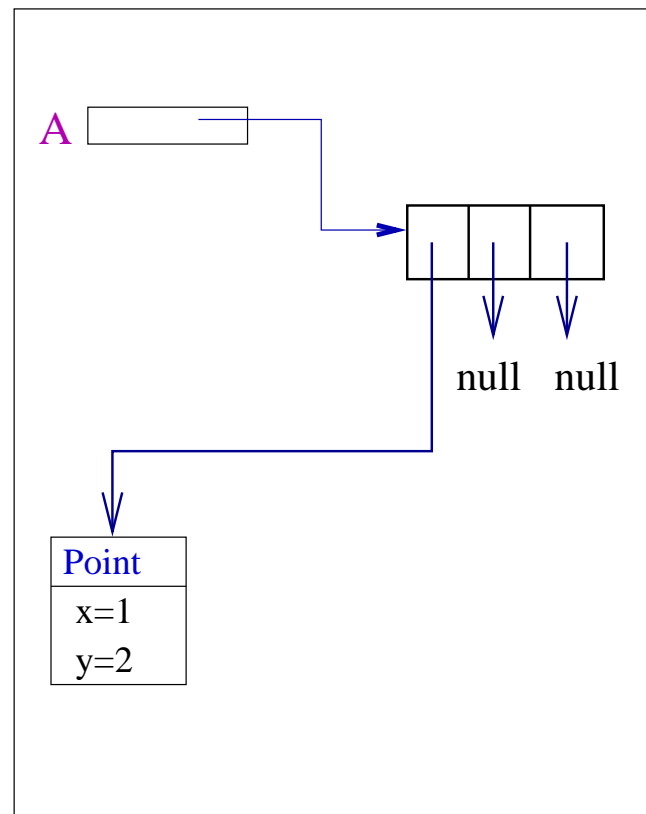
## Array of Points

Point[ ] A ;

A = new Point[3];

A[0] = new Point(1,2);

Memory



## More on arrays

1. the number of elements in an array is called **length** of an array.
2. **array\_name.length** returns the length of the array **array\_name**.
3. For example, A.length will be equal to 3.
4. Length of an array object once created can never be altered.

**A nice problem to work during vacations**



## Design a class My\_integers

which could be used for storing all integers in the range of **long** and could support the following arithmetic operations **without worrying about overflow**.

- Addition of My\_integers
- Multiplication of My\_integers
- Subtraction of My\_integers.
- Computing  $m^t$  where  $m$  is a My\_integer number and  $t$  is an integer of type byte.