Fundamentals of Java

Created by :

Sangeeta Joshi

Agenda

- Types of variables in Java

- static

- Memory Layout

- Packages

- Static Imports

- Arrays

Variables

- A variable is a name given to memory location.

- That memory is associated to a data type and can be assigned a value.

```
int n;
float f1;
char ch;
double d;
```

Variables conti…

- Assigning a value to a variable

- Initialization of a variable with a primary value

```
1. int  n1;
2. n1 =21 ;              // assignment
3. int   i2 = 18;    // initialization
4. char  ch = 'S';   // initialization
5. double d = 21.8;  // initialization
6. d = n1;                   // assignment
7. float f1 = 16.13F;
```

Types of variables in java

- Instance Variables : Copy exists per instance

- Static Variables   : Class level variable i.e. copy exists per class

- Local Variables   : Variables declared within methods or blocks.
   They are local to the block where they are
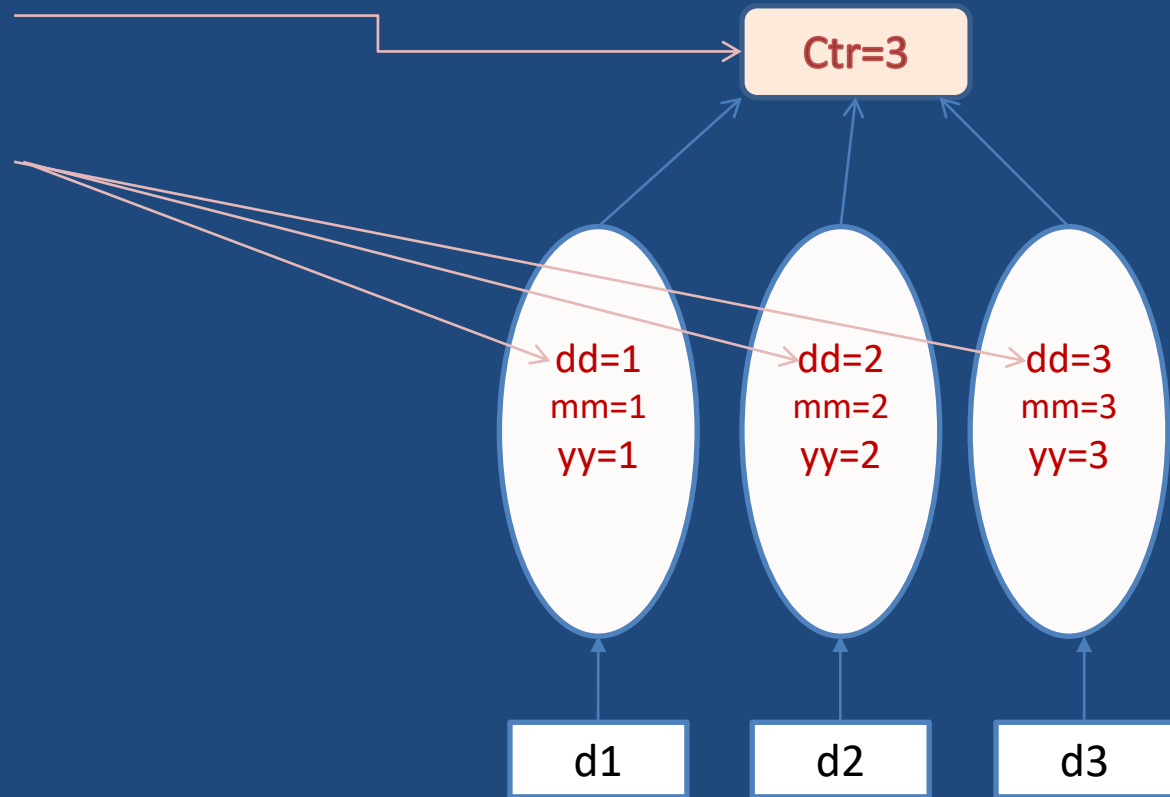   declared

static

- class Date

```
{
    static int ctr;
    int dd;
    int mm;
    int yy;
}
```

Ctr=3

dd=1
mm=1
yy=1

dd=2
mm=2
yy=2

dd=3
mm=3
yy=3

d1

d2

d3

More on static

Static members are associated with the class as a whole rather than
with a particular instance

static variable : It's a class level variable.

static method : If a method is declared as static ,it becomes a class
level method & thus can be invoked using class name. (where as
non static methods require an instance of a class for invocation.)

' this' is never passed to any static data member or function

WHY main() is static ?

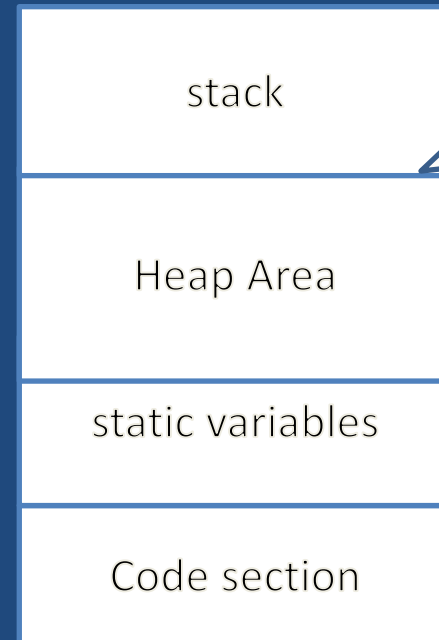Static blocks can be used for initialization of static variables

Static blocks get executed even before main() method

Memory Layout

- Local variables $\longrightarrow$

- Dynamically allocated Area $\longrightarrow$

stack
Heap Area
static variables
Code section

Movable boundary

8

Memory

Local variables are created on stack where as objects are created on heap.

```
class Demo {
   public static void main (String args[])
      {
            Date d = new Date ();
      }
   }
```

d1d   ⟶   Date object

stack   heap

Packages

- A *package* is a grouping of related classes & interfaces providing access protection and name space management

- Programmers bundle groups of related types into packages to make it easier to find & use related classes & interfaces

- Packages help to avoid naming conflicts

- There can be only one package declaration per source file

- In case if no package is declared, then the class is placed into the default package i.e., the current folder

Packages …..continued

- If you want to create a package ,the package statement should be the first statement.

- The fully qualified name of a class is :
  packageName.className

- Fully qualified name of class Date in java is

java.util.Date

package   subpackage   class name
name   name

Packages …..continued

```
package p1;

import  java.util.Date;
import java.sql.*;

class MyClass
{



}
```

Basic Structure Of Java File

➢ Package statement……optional
➢ Import statements …..optional
➢ Standard class writing

Packages .....continued

- Import statement imports public classes within the package
  It does not import sub packages.

- Classes from java.lang package are by default imported.

    How Compiler locates classes?

  ➢   first checks current directory
  ➢   all directories in the class path for
              the actual class file
                    or
              the subdirectory that has same name of imported package

  ➢   then looks for the file in one of the imported packages
  ➢   then finally looks for file in java.lang package
  ➢   if still unable to locate then raises error

		same class	subclass same pkg	diff class same pkg	subclass diffpkg	diff class diffpkg
public		Y	Y	Y	Y	Y
protected		y	y	Y	y	N
default		Y	Y	Y	N	N
private		Y	N	N	N	N

Static Imports

- There are situations where you need frequent access to static final fields (constants) and static methods from one or two classes.

- Prefixing the name of these classes over and over can result in cluttered code.

- The *static import* statement gives you a way to import the constants and static methods that you want to use so that you do not need to prefix the name of their class.

Static Imports

- The static import declaration is analogous to the normal import declaration

- Normal import declaration: imports classes from packages.

- Advantage : classes can be used without package qualification

- static import declaration : imports static members from classes

- Advantage: static members can be used without class qualification

Static Imports

- Before Java 5.0

  Required to fully qualify every static member referenced from external classes

  ```
  Math.sin(x)
  ```

- Now :

  ```
  import static TypeName.Identifier;
  import static Typename.*;
  ```

  Also works for static methods and enums

  ```
  sin(x)
  ```

Static Imports

- Caution : Use Static import Very sparingly!
- If you overuse the static import feature, it can make your program unreadable and unmaintainable

- Importing *all* of the static members from a class : Not a good idea

- use it when you require frequent access to   static members from one or two classes

- Used appropriately, static import can make your program *more* readable

Array(contd)

- An *array* is a container object that holds a fixed number of values of a single type. The length of an array is established when the array is created. After creation, its length is fixed.

- An array is contiguous block of memory locations referred by a common name.

- Each item in an array is called an *element*, and each element is accessed by its numerical *index*.

Declaring Arrays

- Collection of values of a single type
- Declare arrays of primitive or class types.

    - char s[];   or   char [] s;
    - Point p[];   or   Point [] p;

- An array is an object in java
- Array reference is stored on stack whereas actual
- array is created on heap

Creating Arrays

Arrays can be created either using new keyword or without
using new keyword

    int arr [] = { 1,2,3,4,5,6 }   ……..primitive data type

        OR

    int arr[] =new int[6];

        arr[0]=1;

    Date d[]= new Date[4];   …….Date type

        d[0]=new Date();

        d[1]=new Date();

Initializing Arrays

- Initialize an array element

- Create an array with initial values
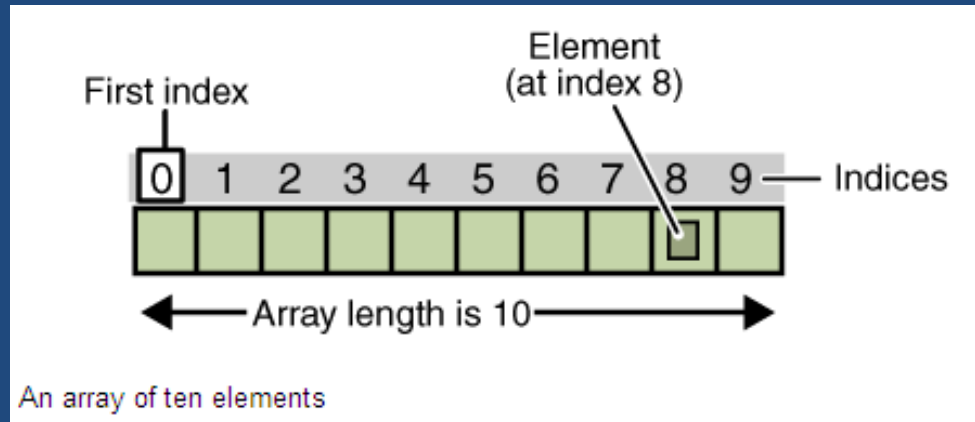 ```
  String names[] = new String[3];
  names [0] = "Jack";
  names [1] = "Jill";
  names [2] = "Tom";

  MyClass array[] = {new MyClass(), new MyClass() };
  ```

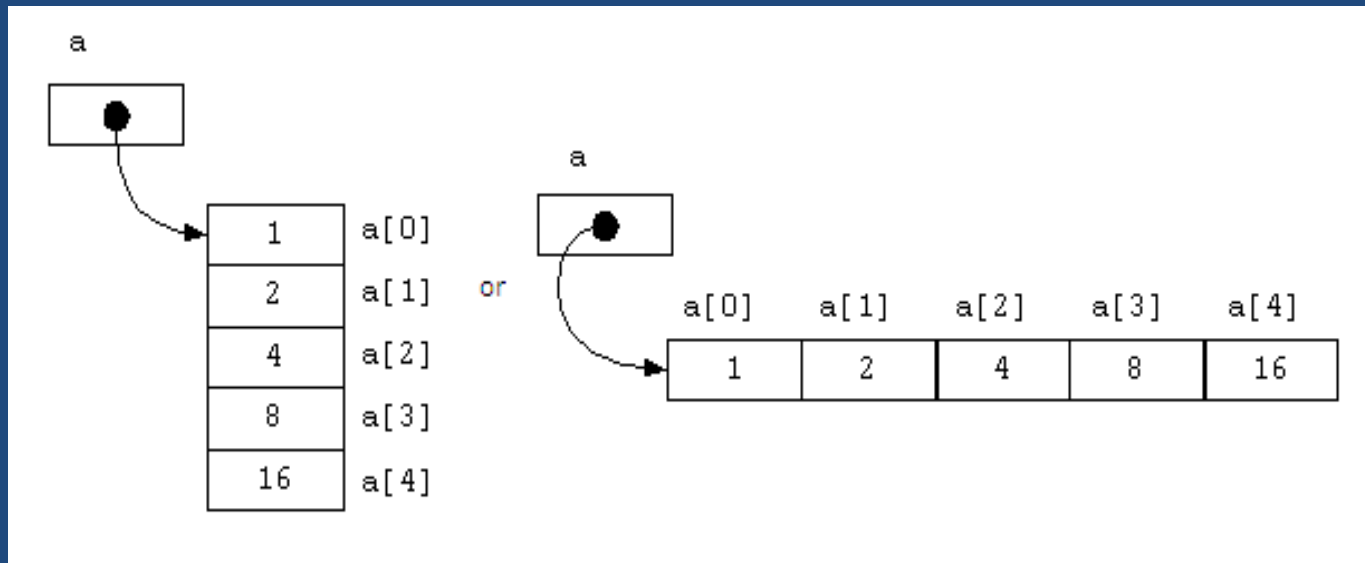# Initializing Arrays (contd)



An array of ten elements

# One-dimensional Arrays

- One-dimensional array is a list of variables of the same data type.

- Syntax to declare a one-dimensional array type

- 

- array_name []; //type is the datatype of the array.


-   For Example:
-    String designations []; // designations is name of the array.

# One-dimensional Arrays (contd)

- int [] a = {1, 2, 4, 8, 16};

# Two-dimensional **Arrays**

- In additions to one-dimensional arrays, you can create two-dimensional arrays. To declare two-dimensional arrays, you need to specify multiple square brackets after the array name.

- Syntax to declare a two dimensional array

- type array_name = new type[rows][cols];

- For Example:
- int multidim[] = new int[3][5];

# Multi-dimensional **Arrays**

To store data in more dimensions a multi-dimensional array is used. A multi-dimensional array of dimension n is a collection of items. These items are accessed via **n** subscript expressions.

# Multi-dimensional Arrays (contd)

```
Example
int [][][] x = new int [2] [3] [4];
                for(int i = 0;i <2 ;i++)
                {
                        for(int j = 0;j <3 ;j++)
                        {
                                for(int k = 0;k <4 ;k++)

        {System.out.print(x[i][j][k]);}
                                System.out.println(" ");
                        }
                }
```
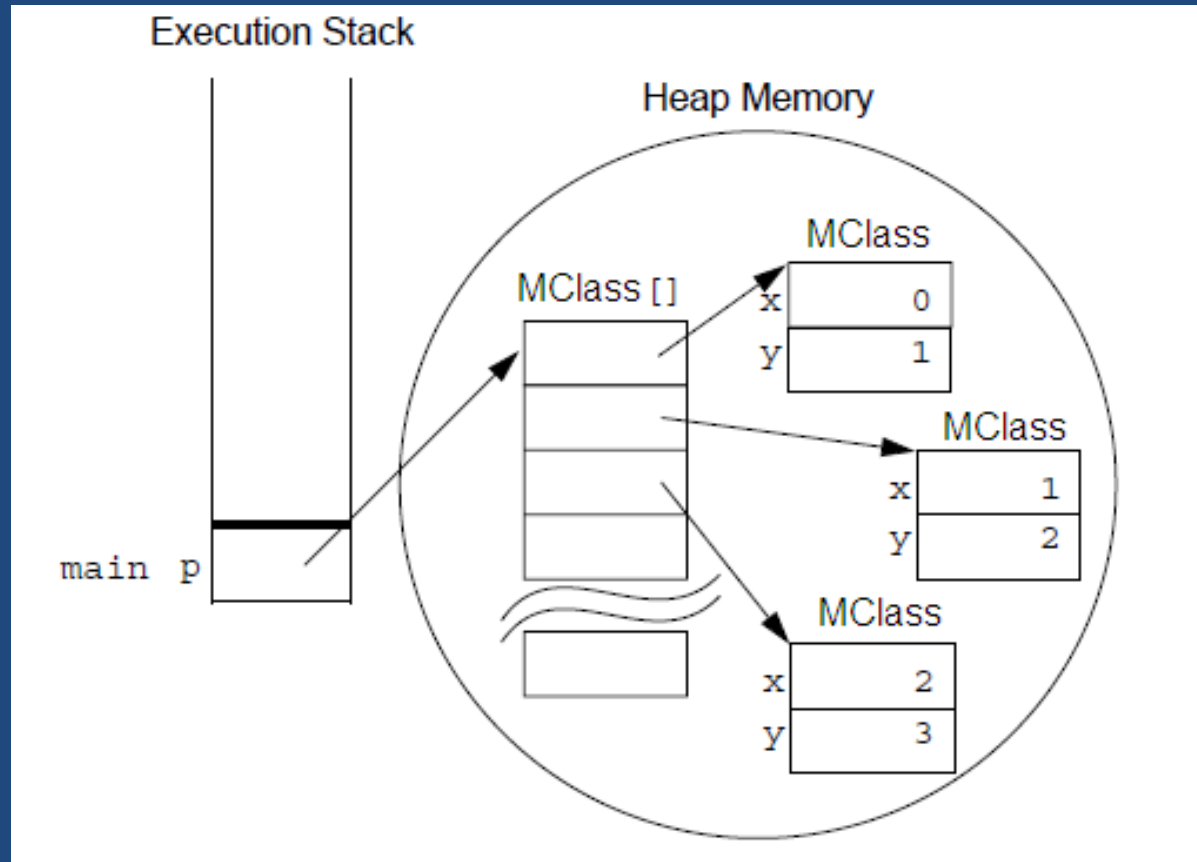
# Creating Reference Arrays

```
Example:
public class MClass{
        public MClass(int x , int y){
                System.out.println("constr" + x + y );
        }
        public static void main(String [] a){
                MClass[] p;
                p = new MClass[10];
                for ( int i=0; i<10; i++ ) {
                p[i] = new MClass(i, i+1);
                }
        }
}
```

Creating Reference Arrays (contd)

Array Bounds

All array index begin at 0:

```
public void disp()
{

        Int [] x = new int[5];
    for (int i = 0; i < x.length; i++)
    {
      System.out.println(x[i]);
    }
}
```

Enhanced for Loop

Enhanced for loop can be used for iterating over arrays:

```
public void disp()
{
    int x = new int[5];
     for ( int i : x )
     {
     System.out.println(i);
      }
}
```

The for loop can be read as *for each* x *in i*.

Array Resizing

You cannot resize an array as it is a static data structure.

You can use the same reference variable to refer to an

entirely new array, such as:

```
int [] x = new int [5];
x = new int[8];
```

Any Questions?