Core Java Foundation Day02

- Akshita Chanchlani

null

If reference variable contains null value then it is called null reference variable / null object.

```
class Program{
    public static void main(String[] args) {
      Employee emp; //Object reference / reference
      emp.printRecord(); //error: variable emp might not have been initialized
    }
}
```

```
public static void main(String[] args) {
    Employee emp = null; //null reference varibale / null object
    emp.printRecord();    //NullPointerException
}
```

Value type VS Reference Type

Sr. No.	Value Type	Reference Type
1	primitive type is also calle as value type.	Non primitive type is also called as reference type.
2	boolean, byte, char, short, int, float, double, long are primitive/value type.	Interface, class, type variable and array are non primitive/reference type.
3	Variable of value type contains value.	Variable of reference type contains reference.
4	Variable of value type by default contains 0 value.	Variable of reference type by default contain null reference.
5	We can not create variable of value type using new operator.	It is madatory to use new operator to create instance of reference type.
6	variable of value type get space on Java stack.	Intstance of reference type get space on heap section.
7	We can not store null value inside variable of value type.	We can store null value inside variable reference type.
8	In case of copy, value gets copied.	In case of copy, reference gets copied.

Reference

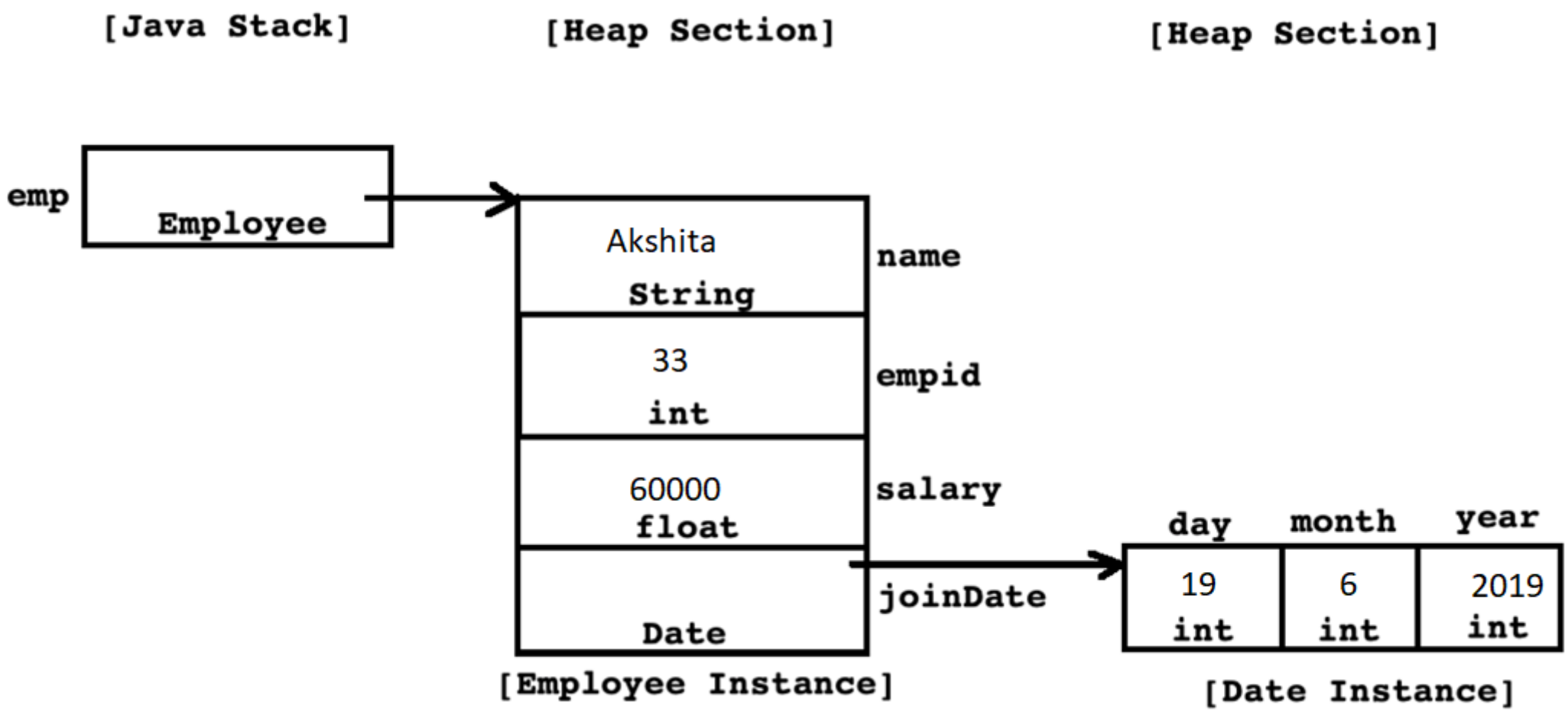
- Class scope reference variable get space on heap.
- Local Reference Variables gets space on java Stack.

```
class Employee{
    private String name;
    private int empid;
    private float salary
    private Date joinDate;  //joinDate : Field
    public Employee( String name, int empid, float salary, Date joinDate ){
        this.name = name;
        this.empid = empid;
        this.salary = salary;
        this.joinDate = joinDate;
    }
}
```

Class Program
{
public static void main(String[] args)
{
Employee emp = new Employee (“Akshita”,33,60000,new Date(19,06,2019));
}
}

- In above code, emp is main method local reference variable hence it gets space on Java Stack. But name, empid, salary and joinDate are fields of Employee class hence it will get space inside instance on Heap.

Reference


[Java Stack]
[Heap Section]
[Heap Section]
emp
Employee
Akshita
String
name
33
int
empid
60000
float
salary
Date
joinDate
day
month
year
19
int
6
int
2019
int
[Employee Instance]
[Date Instance]

Object class

- It is a non final and concrete class declared in java.lang package.
- In java all the classes(not interfaces)are directly or indirectly extended from java.lang.Object class.
- In other words, java.lang.Object class is ultimate base class/super cosmic base class/root of Java class hierarchy.
- Object class do not extend any class or implement any interface.
- It doesn't contain nested type as well as field.
- It contains default constructor.
  - `Object o = new Object("Hello");  //Not OK`
  - `Object o = new Object( );  //OK`

- Object class contains 11 methods.

Object class

- Consider the following code:

```
class Person{
}
class Employee extends Person{
}
```

- In above code, java.lang.object is direct super class of class Person.

- In case class Employee, class Person is direct super class and class object is indirect super class.

toString( ) method

- It is a non final method of java.lang.Object class.
- Syntax:
  - **public String toString( );**
- If we want to return state of Java instance in String form then we should use toString() method.
- Consider definition of toString inside Object class:

```
public String toString() {
    return this.getClass().getName() + "@" + Integer.toHexString(this.hashCode());
}
```

toString( ) method

- If we do not define toString() method inside class then super class's toString() method gets call.
- If we do not define toString() method inside any class then object class's toString() method gets call.
- It return String in following form:
  - **F.Q.ClassName@HashCode**
  - **Example : test.Employee@6d06d69c**
- If we want state of instance then we should override toString() method inside class.
- The result in toString method should be a concise but informative that is easy for a person to read.
- It is recommended that all subclasses override this method.

Final variable

- In java we do not get const keyword. But we can use final keyword.
- After storing value, if we don't want to modify it then we should declare variable final.

```
public static void main(String[] args) {
    final int number = 10;  //Intialization
    //number = number + 5;    //Not OK
    System.out.println("Number  :   "+number );
}
```

```
    public static void main(String[] args) {
        final int number;
        number = 10;    //Assignment
        //number = number + 5;    //Not OK
        System.out.println("Number  :   "+number );
    }
```

Final variable

- We can provide value to the final variable either at compile time or run time.

```
public static void main(String[] args) {
    Scanner sc = new Scanner( System.in);
    System.out.print("Number    :   ");
    final int number = sc.nextInt();    //OK
    //number = number + 5;    //Not OK
    System.out.println("Number   :   "+number );
}
```

Final field

- once initialized, if we don't want to modify state of any field inside any method of the class(including constructor body) then we should declare field final.

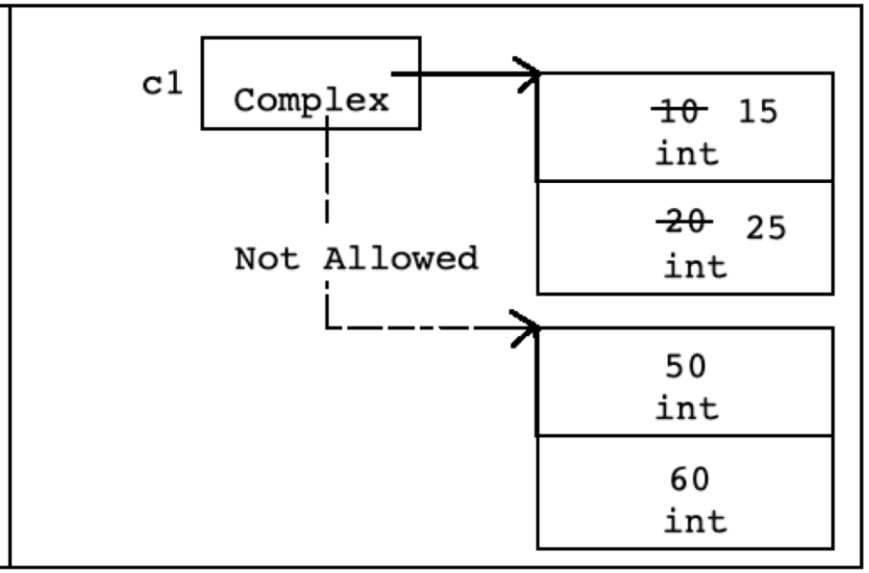
```
class Circle{
    private float area;
    private float radius = 10;
    public static final float PI = 3.142f;
    public void calculateArea( ){
        this.area = PI * this.radius * this.radius;
    }
    public void printRecord( ){
        System.out.println("Area    :   "+this.area);
    }
}
```

- If we want to declare any field final then we should declare it static also.

Final Reference Variable

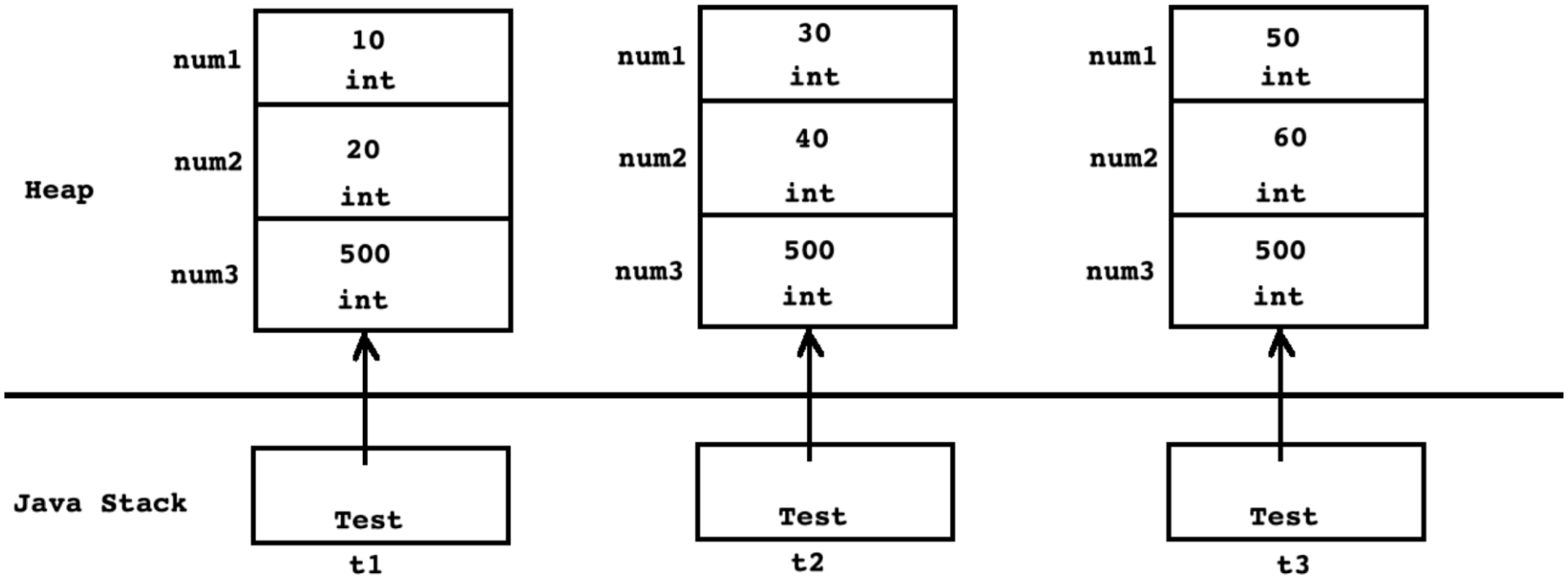
- In Java, we can declare reference final but we can not declare instance final.

```
public static void main(String[] args) {
    final Complex c1 = new Complex( 10, 20 );
    c1.setReal(15);
    c1.setImag(25);
    //c1 = new Complex(50,60);    //Not OK
    c1.printRecord( );      //15, 25
}
```



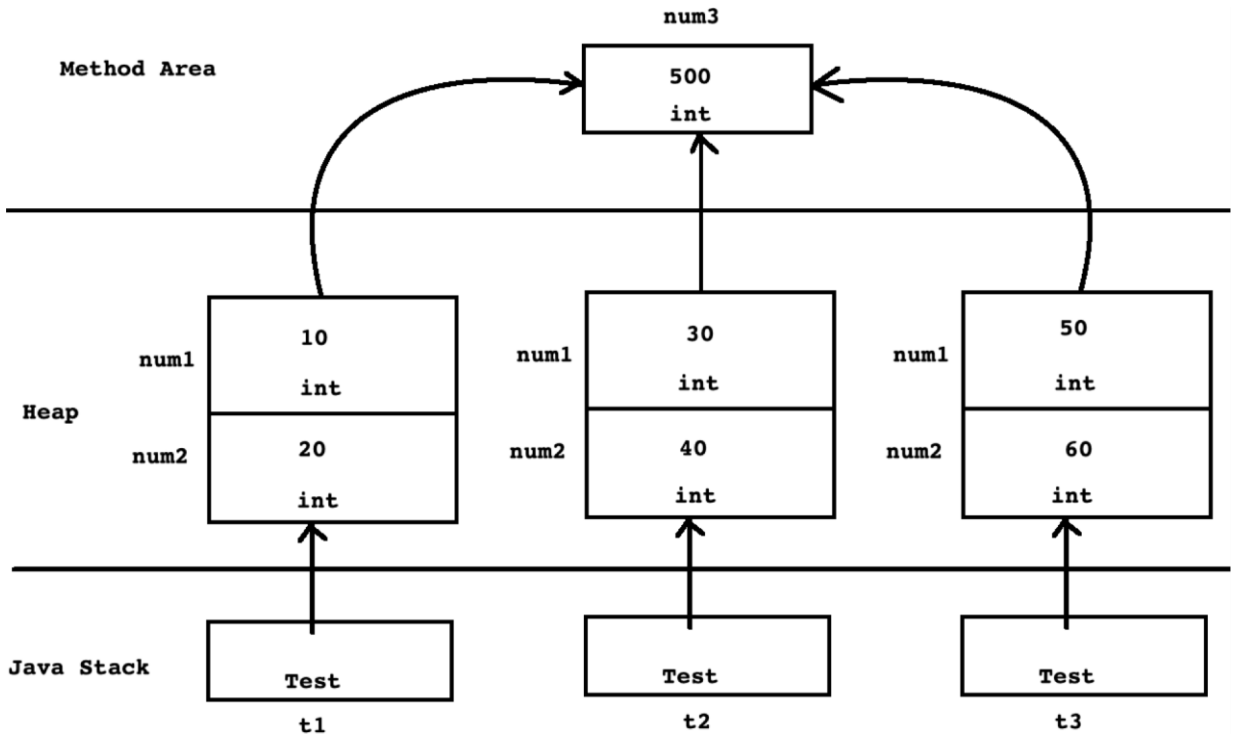
- We can declare method final. It is not allowed to override final method in sub class.
- We can declare class final. It is not allowed to extend final class.

Static Field



Static Field

- If we want to share value of any field inside all the instances of same class then we should declare that field static.



Static Field

- Static field do not get space inside instance rather all the instances of same class share single copy of it.
- Non static Field is also called as instance variable. It gets space once per instance.
- Static Field is also called as class variable. It gets space once per class.
- Static Field gets space once per class during class loading on method area.
- Instance variables are designed to access using object reference.
- Class level variable can be accessed using object reference but it is designed to access using class name and dot operator.

Static Initialization Block

- A *static initialization block* is a normal block of code enclosed in braces, { }, and preceded by the static keyword. Here is an example:

```
static {
    // whatever code is needed for initialization goes here
}
```

- A class can have any number of static initialization blocks, and they can appear anywhere in the class body.
- The runtime system guarantees that static initialization blocks are called in the order that they appear in the source code.
- There is an alternative to static blocks – you can write a private static method:

Instance Initializer Block

- Normally, you would put code to initialize an instance variable in a constructor.
- There are two alternatives to using a constructor to initialize instance variables: initializer blocks and final methods.
- Initializer blocks for instance variables look just like static initializer blocks, but without the static keyword:

```
{
    // whatever code is needed for initialization goes here
}
```

- The Java compiler copies initializer blocks into every constructor. Therefore, this approach can be used to share a block of code between multiple constructors.

Instance Initializer Block

- A *final method* cannot be overridden in a subclass.

```
class Whatever {
    private varType myVar = initializeInstanceVariable();

    protected final varType initializeInstanceVariable() {

        // initialization code goes here
    }
}
```

- This is especially useful if subclasses might want to reuse the initialization method. The method is final because calling non-final methods during instance initialization can cause problems.

Static Method

- To access non static members of the class, we should define non static method inside class.
- Non static method/instance method is designed to call on instance.
- To access static members of the class, we should define static method inside class.
- static method/class level method is designed to call on class name.
- static method do not get this reference:
    1. If we call, non static method on instance then method get this reference.
    2. Static method is designed to call on class name.
    3. Since static method is not designed to call on instance, it doesn't get this reference.

Static Method

- this reference is a link/connection between non static field and non static method.
- Since static method do not get this reference, we can not access non static members inside static method directly. In other words, static method can access static members of the class only.
- Using instance, we can use non static members inside static method.

```
class Program{
    public int num1 = 10;
    public static int num2 = 10;
    public static void main(String[] args) {
        //System.out.println("Num1    :   "+num1);    //Not OK
        Program p = new Program( );
        System.out.println("Num1    :   "+p.num1);    //OK
        System.out.println("Num2    :   "+num2);
    }
}
```

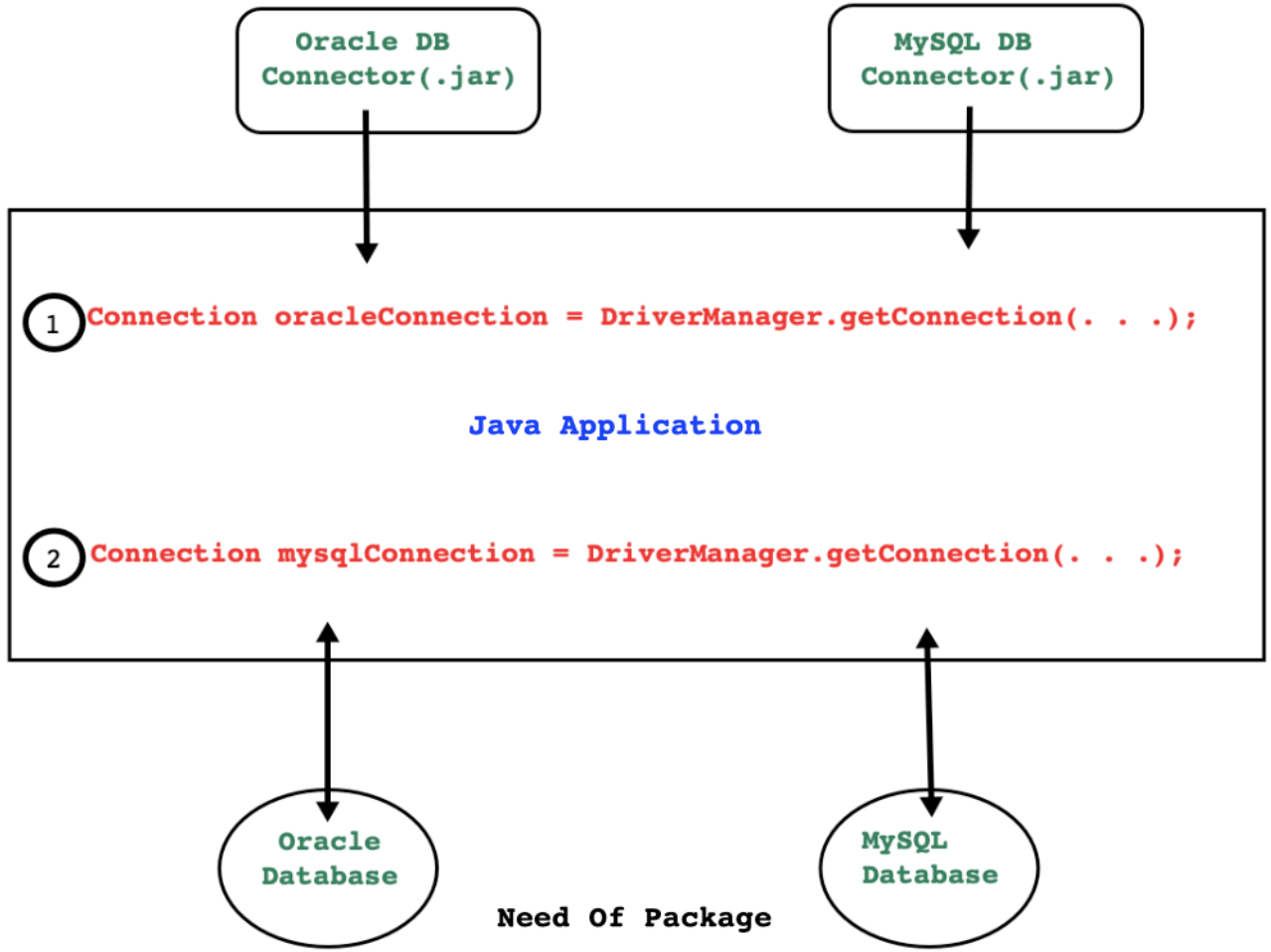
Static Method

- Inside method, If we are going to use this reference then method should be non static otherwise it should be static.

```
class Math{
    public static int power( int base, int index ){
        int result = 1;
        for( int count = 1; count <= index; ++ count ){
            result = result * base;
        }
        return result;
    }
}
```

```
class Program{
    public static void main(String[] args) {
        int result = Math.power(10, 2);
        System.out.println("Result  :   "+result);
    }
}
```

Package


Oracle DB
Connector(.jar)
MySQL DB
Connector(.jar)
1
Connection oracleConnection = DriverManager.getConnection(. . .);
Java Application
2
Connection mysqlConnection = DriverManager.getConnection(. . .);
Oracle
Database
MySQL
Database
Need Of Package

Package

- Package is a Java language feature which helps developer to:
    1. To group functionally equivalent or related types together.
    2. To avoid naming clashing/collision/conflict/ambiguity in source code.
    3. To control the access to types.
    4. To make types easier to find(from the perspective of java docs).

- Consider following class:
    - java.lang.Object
        - Here java is main package, lang is sub package and Object is type name.

Package

- Not necessarily but as shown below, package can contain some or types.
    1. Sub package
    2. Interface
    3. Class
    4. Enum
    5. Exception
    6. Error
    7. Annotation Type

Package Creation

- package is a keyword in Java.
- To define type inside package, it is mandatory write package declaration statement inside .java file.
- Package declaration statement must be first statement inside .
- If we define any type inside package then it is called as packaged type otherwise it will be unpackaged type.
- Any type can be member of single package only.

```
package p1;   //OK
class Program{
    //TODO
}
```

```
package p1, p2; //NOT OK
class Program{
    //TODO
}
```

```
package p1; //OK
package p2; //NOT OK
class Program{
    //TODO
}
package p3; //Not OK
```

Un-named Package

- If we define any type without package then it is considered as member of unnamed/default package.
- Unnamed packages are provided by the Java SE platform principally for convenience when developing small or temporary applications or when just beginning development.
- An unnamed package cannot have sub packages.
- In following code, class Program is a part of unnamed package.

```
class Program{

    public static void main(String[] args) {

        System.out.println("Hello");

    }

}
```

Naming Convention

- For small programs and casual development, a package can be unnamed or have a simple name, but if code is to be widely distributed, unique package names should be chosen using qualified names.
- Generally Package names are written in all lower case to avoid conflict with the names of classes or interfaces.
- Companies use their reserved internet domain name to begin their package names. For example : com.example.mypackage
- Following examples will help you in deciding name of package:
    1. java.lang.reflect.Proxy
    2. oracle.jdbc.driver.OracleDriver
    3. com.mysql.jdbc.cj.Driver
    4. org.cdac.sunbeam.dac.utils.Date

How to use package members in different package?

- If we want to use types declared inside package anywhere outside the package then
    1. Either we should use fully qualified type name or
    2. import statement.
- If we are going to use any type infrequently then we should use fully qualified name.
- Let us see how to use type using package name.

```
class Program{
    public static void main(String[] args) {
        java.util.Scanner sc = new java.util.Scanner( System.in );
    }
}
```

How to use package members in different package?

- If we are going to use any type frequently then we should use import statement.
- Let us see how to import Scanner.

```
import java.util.Scanner;
class Program{
    public static void main(String[] args) {
        Scanner sc = new Scanner( System.in );
    }
}
```

How to use package members in different package?

- There can be be any number of import statements after package declaration statement
- With the help of(*) we can import entire package.

```
import java.util.*;
class Program{
    public static void main(String[] args) {
        Scanner sc = new Scanner( System.in );
    }
}
```

How to use package members in different package?

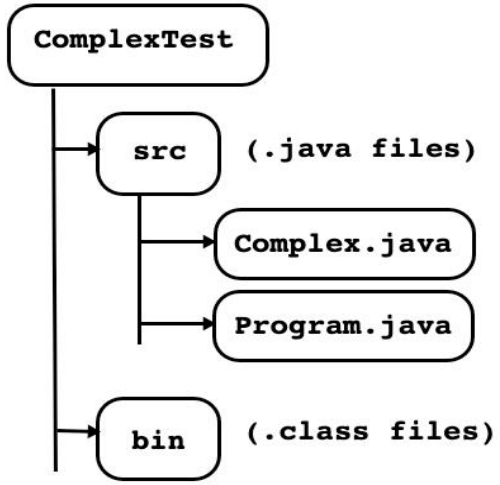
- Another, less common form of import allows us to import the public nested classes of an enclosing class. Consider following code.

```
import java.lang.Thread.State;

class Program{

    public static void main(String[] args) {

        Thread thread = Thread.currentThread( );

        State state = thread.getState( );

    }

}
```

- Note : java.lang package contains fundamental types of core java. This package is by default imported in every .java file hence to use type declared in java.lang package, import statement is optional.

Example 1

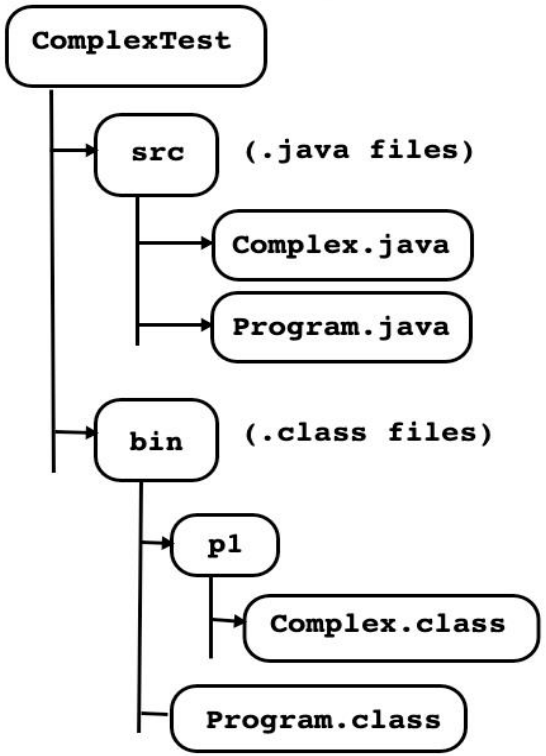
[Compile from here]



[Before Compilation]

1. Set path(if not set)
2. Compile “Complex.java”
3. set classpath
4. Compile “Program.java”
5. execute “Program.class”

[execute from here]



[After Compilation]

Example 1

```java
//Location : ./src/Complex.java
package p1;
class TComplex{
    @Override
    public String toString( ){
        return "TComplex.toString()";
    }
}
```

```java
//Location : ./src/Program.java
import p1.TComplex;
class Program{
    public static void main(String[] args) {
        //p1.TComplex c1 = new p1.TComplex( );  //or
        TComplex c1 = new TComplex( );
        System.out.println( c1.toString( ) );
    }
}
```

```
javac -d ./bin ./src/Complex.java    //output : p1/TComplex.class
export CLASSPATH=./bin
javac -d ./bin ./src/Program.java    //Output : Error
//TComplex is not public in p1; can not be accessed from outside package
```

Use export CLASSPATH in case of MAC
and use set CLASSPATH in case of windows.

Example 1

- Package name is physically mapped to the folder.
- Point to Remember : default access modifier of any type is package level private which is also called as default.

```
//location  :   ./src/Complex.java
??? class TComplex{ //here ??? means package level  private / default
    //TODO
}
```

- If we want to use any type inside same package as well as in different package then access modifier of type must be public.
- Access modifier of type(class/interface) can be either package level private/public only. In other words, type can not be private or protected.

```
//location  :   ./src/Complex.java
public class TComplex{
}
```

- If we compile then compiler generates error.

Example 1

- According to Java Language Specification(JLS), name of public type and name of .java file must be same. It means that, .java file can contain multiple non public types but only one public type.

```
//location  :   ./src/Complex.java
public class Complex{   //Now OK
     @Override
    public String toString( ){
        return "Complex.toString()";
    }
}
```

- Let us recompile above code:
  - javac -d ./bin ./src/Complex.java //output : p1/Complex.class
  - export CLASSPATH=./bin
  - javac -d ./bin ./src/Program.java //Output : Program.class
  - java Program //Output : Complex.toString()
- **Conclusion : It is possible to use packaged type from unpackaged type.**

Example 2

```
//Location : ./src/Complex.java
public class Complex{ //Unpackaged Type
    @Override
    public String toString( ){
        return "Complex.toString()";
    }
}
```

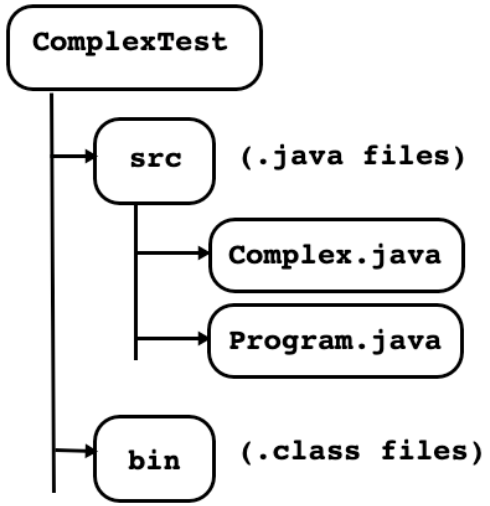
```
//Location : ./src/Program.java
package p1;
public class Program{
    public static void main(String[] args) {
        Complex c1 = new Complex( );
        System.out.println( c1.toString( ) );
    }
}
```

```
javac -d ./bin ./src/Complex.java    //OK:Complex.class
export CLASSPATH=./bin
javac -d ./bin ./src/Program.java    //error : cannot find symbol
```

- If we define any type without package then it is considered as a member of default package.
- Conclusion : Since we can not import default package, it is not possible to use unpackaged type from packaged type.

Example 3

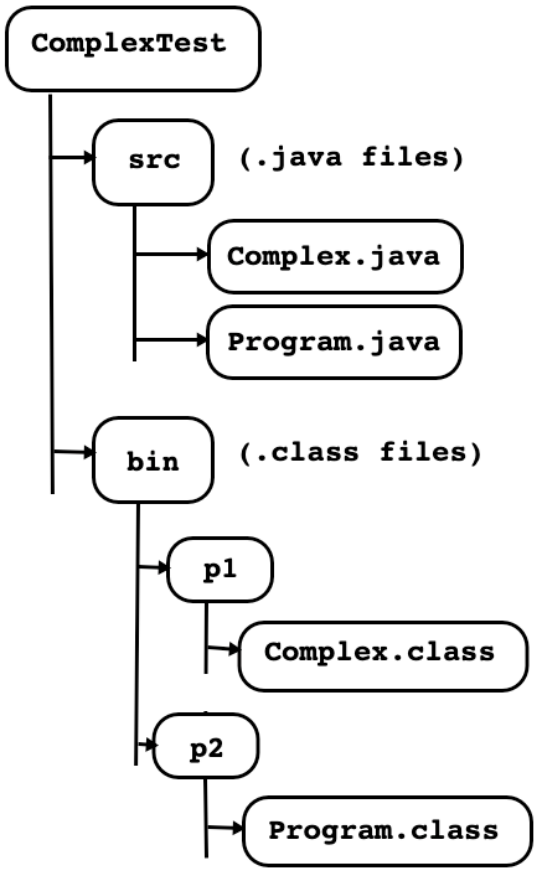
[Compile from here]



[Before Compilation]

1. Set path(if not set)
2. Compile “Complex.java”
3. set classpath
4. Compile “Program.java”
5. execute “Program.class”

[execute from here]



[After Compilation]

Example 3

```java
//Location : ./src/Complex.java
package p1;
public class Complex{
    @Override
    public String toString( ){
        return "Complex.toString()";
    }
}
```

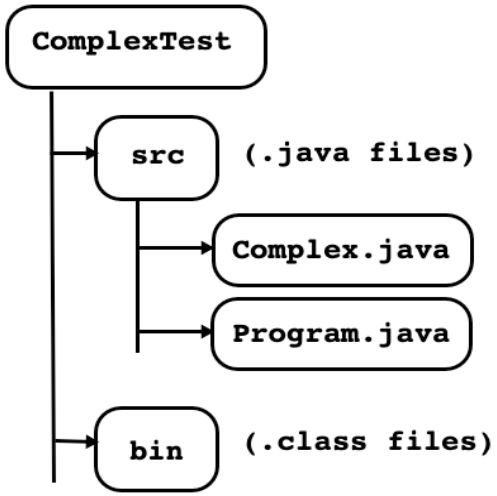
```java
//Location : ./src/Program.java
package p2;
import p1.Complex;
public class Program{
    public static void main(String[] args) {
        Complex c1 = new Complex( );
        System.out.println( c1.toString( ) );
    }
}
```

```
javac -d ./bin ./src/Complex.java   //OK:p1/Complex.class
export CLASSPATH=./bin
javac -d ./bin ./src/Program.java   //OK:p2/Program.class
//java Program  //Error
java p2.Program //Complex.toString()
```

- Conclusion : We can define types into separate package. It is possible to use packaged type from another packaged type.

Example 4

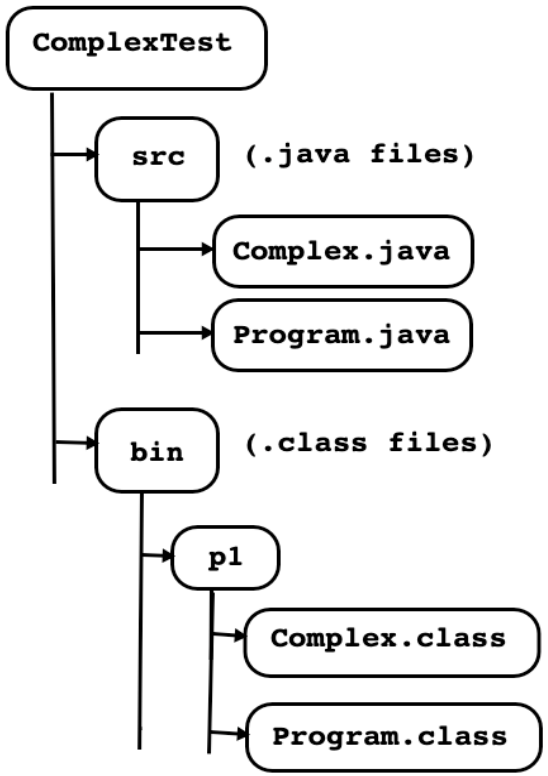
[Compile from here]



[Before Compilation]

1. Set path(if not set)
2. Compile “Complex.java”
3. set classpath
4. Compile “Program.java”
5. execute “Program.class”

[execute from here]



[After Compilation]

Example 4

```java
//Location : ./src/Complex.java
package p1;
public class Complex{
    @Override
    public String toString( ){
        return "Complex.toString()";
    }
}
```

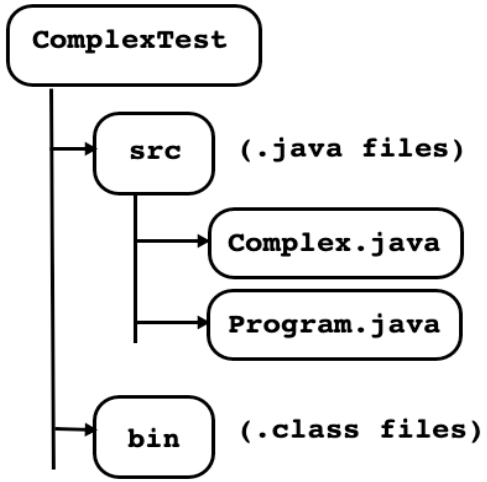
```java
//Location : ./src/Program.java
package p1;
import p1.Complex;  //Optional
public class Program{
    public static void main(String[] args) {
        Complex c1 = new Complex( );
        System.out.println( c1.toString( ) );
    }
}
```

```
javac -d ./bin ./src/Complex.java    //OK:p1/Complex.class
export CLASSPATH=./bin
javac -d ./bin ./src/Program.java    //OK:p1/Program.class
java p1.Program //Complex.toString()
```

- Conclusion : We can define all types into single package. In this case, to use one type into another type, use of import statement is optional.

Example 5

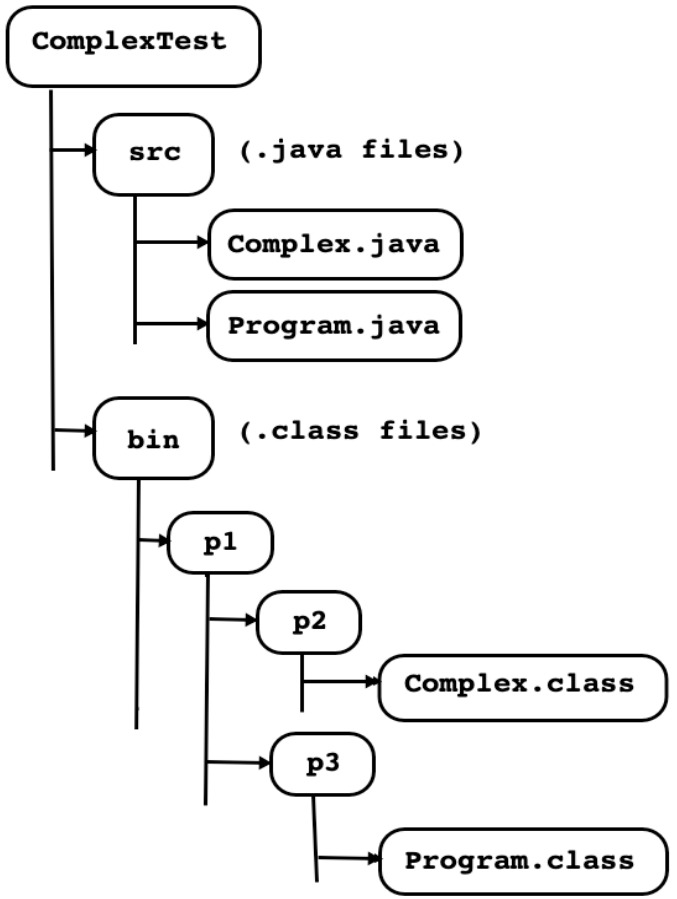
[Compile from here]



[Before Compilation]

1. Set path(if not set)
2. Compile “Complex.java”
3. set classpath
4. Compile “Program.java”
5. execute “Program.class”

[execute from here]



[After Compilation]

Example 5

```java
//Location : ./src/Complex.java
package p1.p2;
public class Complex{
    @Override
    public String toString( ){
        return "Complex.toString()";
    }
}
```

```java
//Location : ./src/Program.java
package p1.p3;
import p1.p2.Complex;
public class Program{
    public static void main(String[] args) {
        Complex c1 = new Complex( );
        System.out.println( c1.toString( ) );
    }
}
```

```
javac -d ./bin ./src/Complex.java   //OK:p1/p2/Complex.class
export CLASSPATH=./bin
javac -d ./bin ./src/Program.java   //OK:p1/p3/Program.class
java p1.p3.Program //Complex.toString()
```

- Conclusion : We can create package inside package. It is also called as sub package.

Static Import

- If static members belonging to the same class then use of type name and dot operator is optional.

```
package p1;
public class Program{
    private static int number = 10;
    public static void main(String[] args) {
        System.out.println("Number  :   "+Program.number);  //OK    :   10
        System.out.println("Number  :   "+number);  //OK    :   10
    }
}
```

Static Import

- If static members belonging to the different class then use of type name and dot operator is mandatory.
- PI and pow are static members of java.lang.Math class. To use Math class import statement is not required.
- Consider Following code:

```
package p1;
public class Program{
    public static void main(String[] args) {
        float radius = 10.5f;
        float area = ( float )( Math.PI * Math.pow( radius, 2 );
        System.out.println( "Area    :    "+area );
    }
}
```

Static Import

- 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.
- Consider code:

```
package p1;
import static java.lang.System.out;
import static java.lang.Math.*;
public class Program{
    public static void main(String[] args) {
        float radius = 10.5f;
        float area = ( float )( PI * pow( radius, 2 );
        out.println( "Area   :   "+area );
    }
}
```

Array

- Array, stack, queue, LinkedList are data structures.
- In Java, data structure is called collection and value stored inside collection is called element.
- Array is a sequential/linear container/collection which is used to store elements of same type in continuous memory location.

In C/C++

Static Memory allocation for array

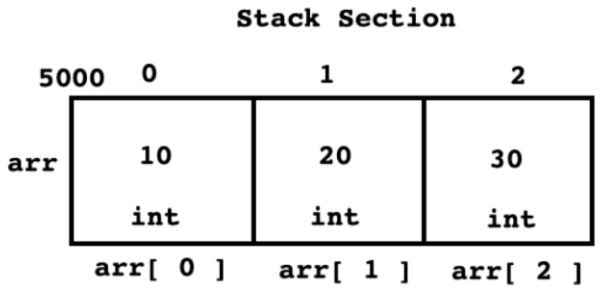
```
int arr1[ 3 ];    //OK

int size = 3;
int arr2[ size ];     //OK
```

In C/C++

Dynamic Memory allocation for array

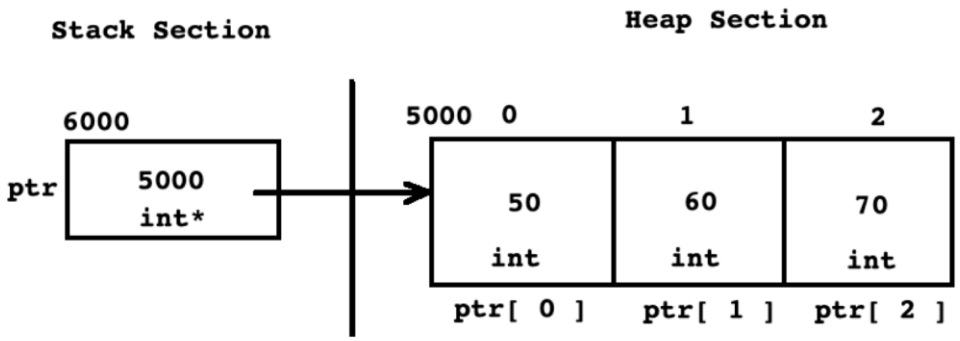
```
int *arr = ( int* )malloc( 3 * sizeof( int ));
//or
int *arr = ( int* )calloc( 3, sizeof( int ));
```

Static v/s Dynamic Memory Allocation In C/C++



[Static Memory Allocation]

```
arr[ 0 ]  =>  10
arr[ 1 ]  =>  20
arr[ 2 ]  =>  30
```



[Dynamic Memory Allocation]

```
ptr[ 0 ]  =>  50
ptr[ 1 ]  =>  60
ptr[ 2 ]  =>  70
```

Array Declaration and Initialization In C

- `int arr[ 3 ]; //OK : Declaration`
- `int arr[ 3 ] = { 10, 20, 30 }; //OK : Initialization`
- `int arr[ ] = { 10, 20, 30 }; //OK`
- `int arr[ 3 ] = { 10, 20 }; //OK : Partial Initialization`
- `int arr[ 3 ] = { }; //OK : Partial Initialization`
- `int arr[ 3 ] = { 10, 20, 30, 40, 50 }; //Not recommended`

Accessing Elements Of Array

- If we want to access elements of array then we should use integer index.
- Array index always begins with 0.

```
int arr[ 3 ] = { 10, 20, 30 };
printf("%d\n", arr[ 0 ] );
printf("%d\n", arr[ 1 ] );
printf("%d\n", arr[ 2 ] );
```

```
int arr[ 3 ] = { 10, 20, 30 };
int index;
for( index = 0; index < 3; ++ index )
    printf("%d\n", arr[ index ] );
```

Advantage and Disadvantages Of Array

- **Advantage Of Array**

  1. We can access elements of array randomly.

- Disadvantage Of Array

  1. We can not resize array at runtime.
  2. It requires continuous memory.
  3. Insertion and removal of element from array is a time consuming job.
  4. Using assignment operator, we can not copy array into another array.
  5. Compiler do not check array bounds(min and max index).

Array In Java

- Array is a reference type in Java. In other words, to create instance of array, new operator is required. It means that array instance get space on heap.
- **There are 3 types of array in Java:**
    1. Single dimensional array
    2. Multi dimensional array
    3. Ragged array
- **Types of loop in Java:**
    1. do-while loop
    2. while loop
    3. for loop
    4. for-each loop
- **To perform operations on array we can use following classes:**

    java.util.Arrays

Single Dimensional Array

Reference declaration

```
int arr[ ]; //OK
int [ arr ]; //NOT OK
int[ ] arr; //OK
```

Instantiation

```
int[ ] arr1 = new int[ 3 ];
//or
int size = 3;
int[ ] arr2 = new int[ size ];
```

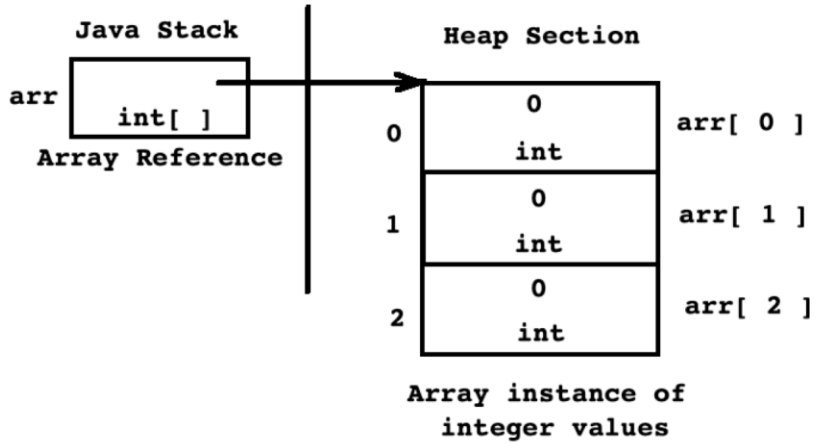
```
int[] arr1 = new int[ -3 ];  //NegativeArraySizeException
//or
int size = -3;
int[] arr2 = new int[ size ];    //NegativeArraySizeException
```

Initialization

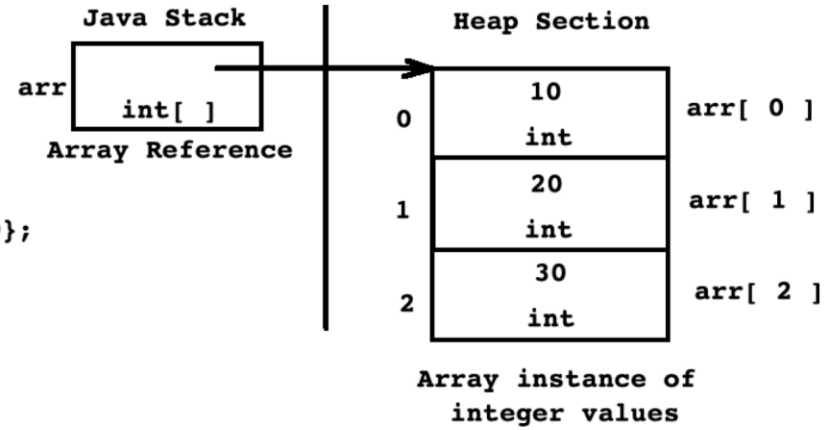
```
int[] arr = new int[ size ]{ 10, 20, 30 };  //Not OK
int[] arr = new int[  ]{ 10, 20, 30 };  //OK
int[] arr = { 10, 20, 30 };  //OK
```

Single Dimensional Array

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




```
int[] arr = new int[]{10,20,30};
```



Using length Field

```
public class Program {
    public static void printRecord( int[] arr ) {
        for( int index = 0; index < arr.length; ++ index )
            System.out.print( arr[ index ] +"   ");
        System.out.println();
    }
    public static void main(String[] args) {
        int[] arr1 = new int[  ] { 10, 20, 30 };
        Program.printRecord(arr1);

        int[] arr2 = new int[  ] { 10, 20, 30, 40, 50 };
        Program.printRecord(arr2);

        int[] arr3 = new int[  ] { 10, 20, 30, 40, 50, 60, 70  };
        Program.printRecord(arr3);
    }
}
```

Using toString( ) Method

```
public static void main(String[] args) {
    int[] arr = new int[   ] { 10, 20, 30, 40, 50 };
    System.out.println(arr.toString()); //[I@6d06d69c
}
public static void main(String[] args) {
    double[] arr = new double[   ] { 10.1, 20.2, 30.3, 40.4, 50.5 };
    System.out.println(arr.toString()); //[D@6d06d69c
}
//Check the documentation of getName() method of java.lang.Class.
public static void main(String[] args) {
    int[] arr = new int[   ] { 10, 20, 30, 40, 50 };
    System.out.println(Arrays.toString(arr));   //[10, 20, 30, 40, 50]
}
```

ArrayIndexOutOfBoundsException

- Using illegal index, if we try to access elements of array then JVM throws ArrayIndexOutOfBoundsException. Consider following code:

```
public static void main(String[] args) {
    int[] arr = new int[  ] { 10, 20, 30, 40, 50 };
    //int element = arr[ -1 ]; //ArrayIndexOutOfBoundsException
    //int element = arr[ arr.length ]; //ArrayIndexOutOfBoundsException
    //int element = arr[ 7 ]; //ArrayIndexOutOfBoundsException
}
```

Sorting Array Elements

```
public class Program {
    public static void main(String[] args) {
        int[] arr = new int[] { 50, 10, 40, 20, 30 };
        System.out.println(Arrays.toString(arr));
        Arrays.sort(arr);   //The sorting algorithm is a Dual-Pivot Quicksort
        System.out.println(Arrays.toString(arr));
    }
}
```

Reference Copy and Instance Copy

Array Reference copy

```
int[] arr1 = new int[   ] { 10, 20, 30, 40, 50 };
int[] arr2 = arr1;  //Reference Copy
```

Array Instance Copy(Using Arrays.copyOf())

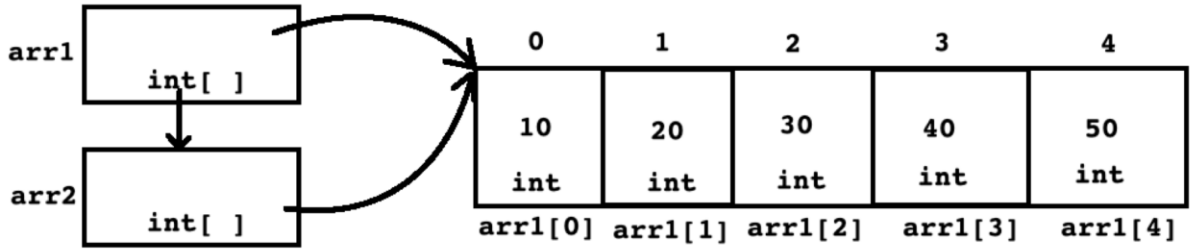
```
int[] arr1 = new int[   ] { 10, 20, 30, 40, 50 };
int[] arr2 = Arrays.copyOf(arr1, arr1.length);  //Array instance copy
```

Implementation of Arrays.copyOf method:

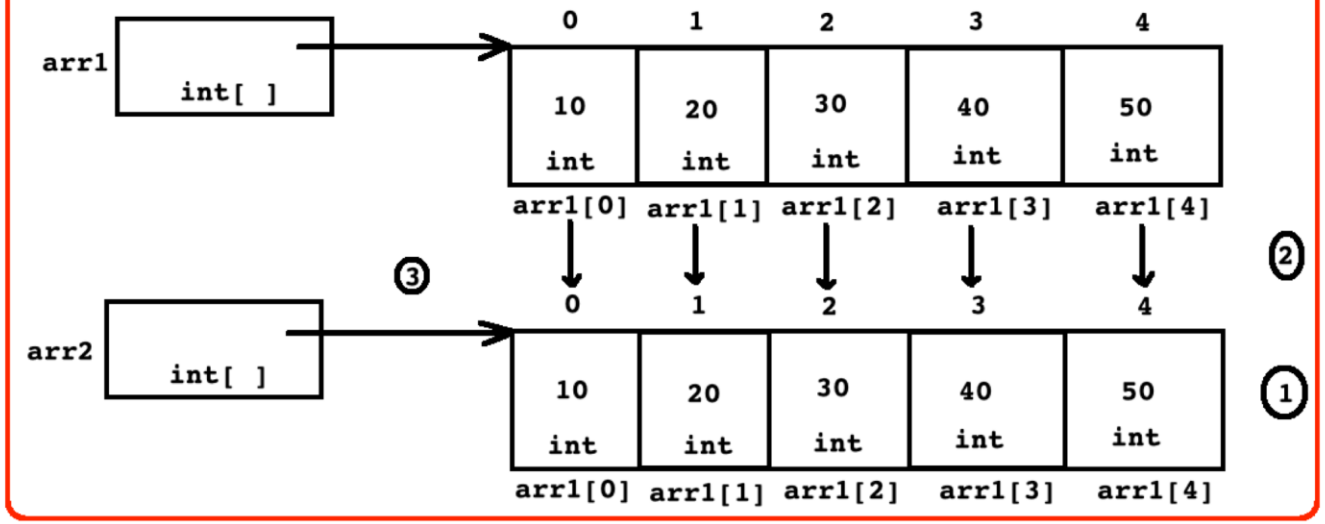
```
public static int[] copyOf(int[] original, int newLength) {
    int[] copy = new int[newLength];
    //public static void arraycopy(Object src, int srcPos, Object dest, int destPos, int length);
    System.arraycopy(original, 0, copy, 0, Math.min(original.length, newLength));
    return copy;
}
```

Reference Copy and Instance Copy

```
int[] arr1 = new int[  ] { 10, 20, 30, 40, 50 };
int[] arr2 = arr1;  //Reference Copy
```




```
int[] arr1 = new int[  ] { 10, 20, 30, 40, 50 };
int[] arr2 = Arrays.copyOf(arr1, arr1.length);  //Array instance copy
```

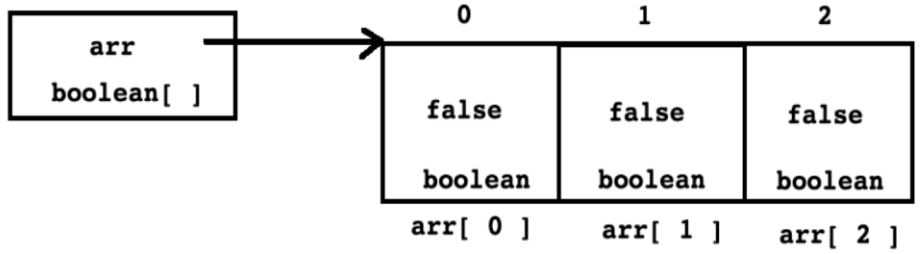


Array Of Primitive Values

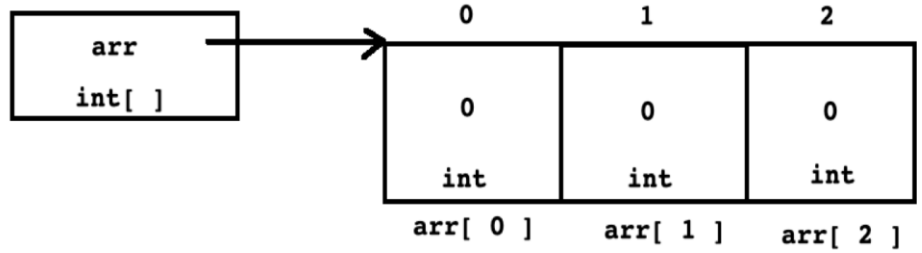
```
public class Program {
    public static void main(String[] args) {
        boolean[] arr = new boolean[ 3 ];   //contains all false
        int[] arr = new int[ 3 ];   //contains all 0
        double[] arr = new double[ 3 ]; //contains all 0.0
    }
}
```

Array Of Primitive Values

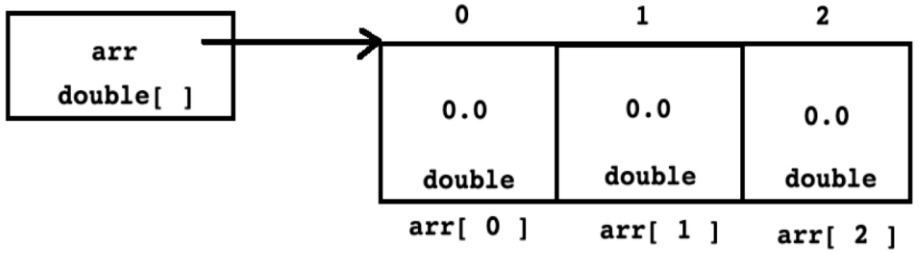
```
boolean[] arr = new boolean[3];
```




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




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



If we create array of primitive values then it's default value depends of default value of data type.

Array Of References

```
public class Program {
    public static void main(String[] args) {
        Date[] arr = new Date[ 3 ]; //Contains all null
    }
}
```

Array Of References and Instances

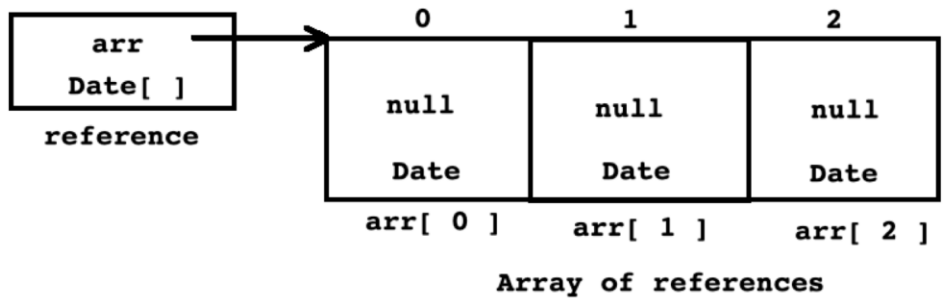
```
public class Program {
    public static void main(String[] args) {
        Date[] arr = new Date[ 3 ]; //Contains all null
    }
}
```

- Let us see how to create array of instances of non primitive type

```
public class Program {
    public static void main(String[] args) {
        Date[] arr = new Date[ 3 ];
        arr[ 0 ] = new Date( );
        arr[ 1 ] = new Date( );
        arr[ 2 ] = new Date( );
    }
    //or
    public static void main(String[] args) {
        Date[] arr = new Date[ 3 ];
        for( int index = 0; index < arr.length; ++ index )
        arr[ index ] = new Date( );
    }
}
```

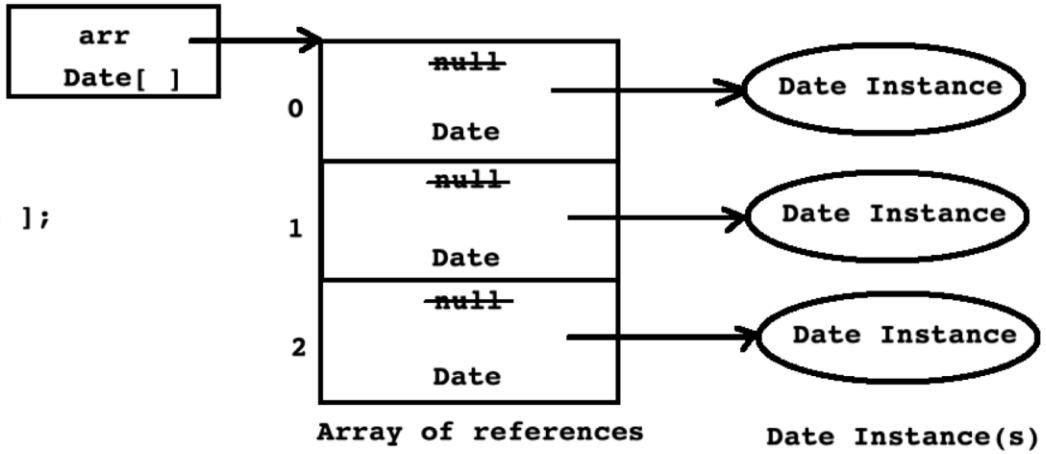
Array Of References and Instances

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



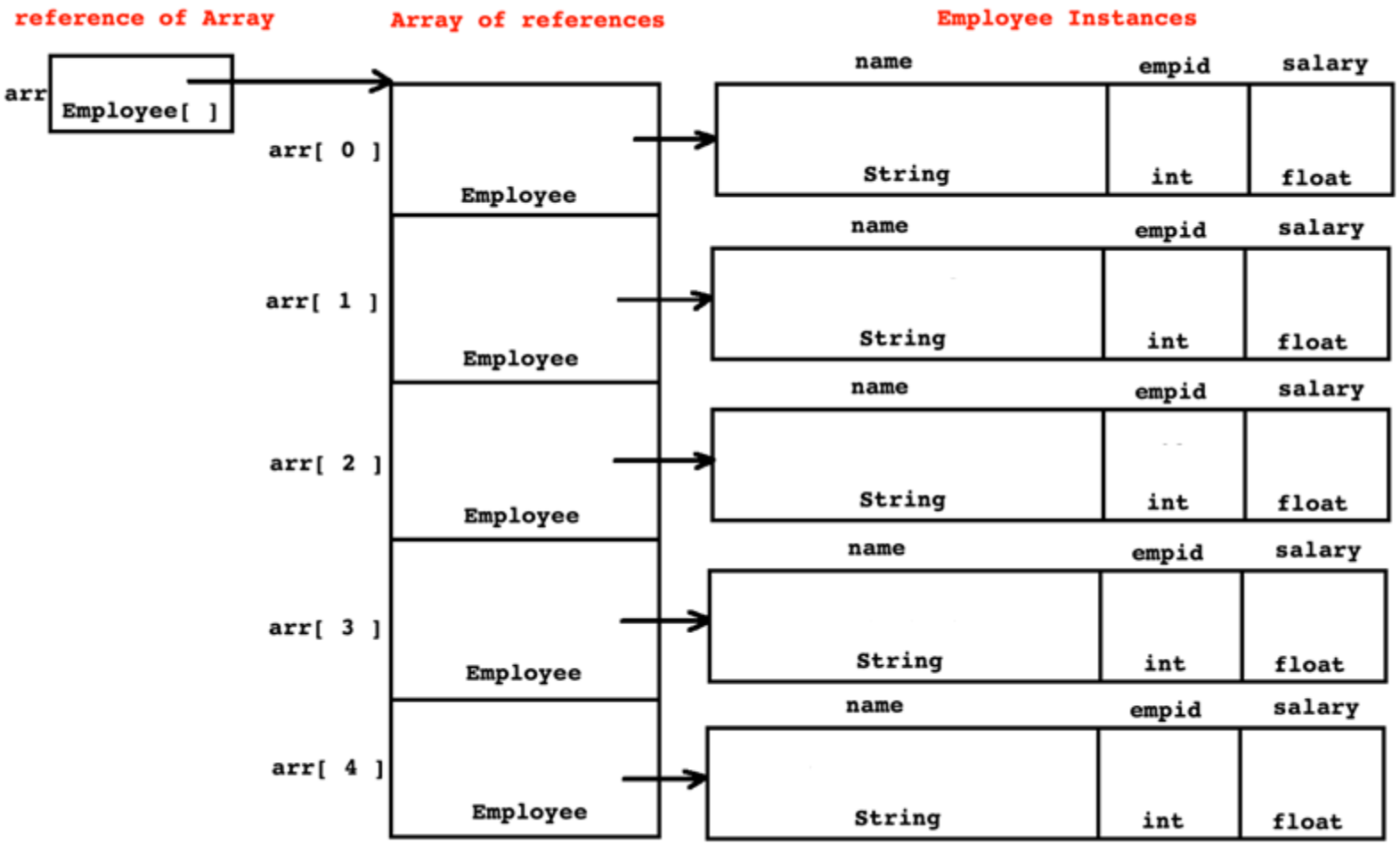
If we create an array of references then by default it contains null.

```
Date[] arr = new Date[ 3 ];
arr[ 0 ] = new Date( );
arr[ 1 ] = new Date( );
arr[ 2 ] = new Date( );
```



[Array Of Instances]

Array Of Instances



Multi Dimensional Array

- Array of elements where each element is array of same column size is called as multi dimensional array.

Reference declaration:

```
int arr[ ][ ];   //OK
int [ ]arr[ ]    //OK
int[ ][ ] arr;   //OK
```

Array Creation:

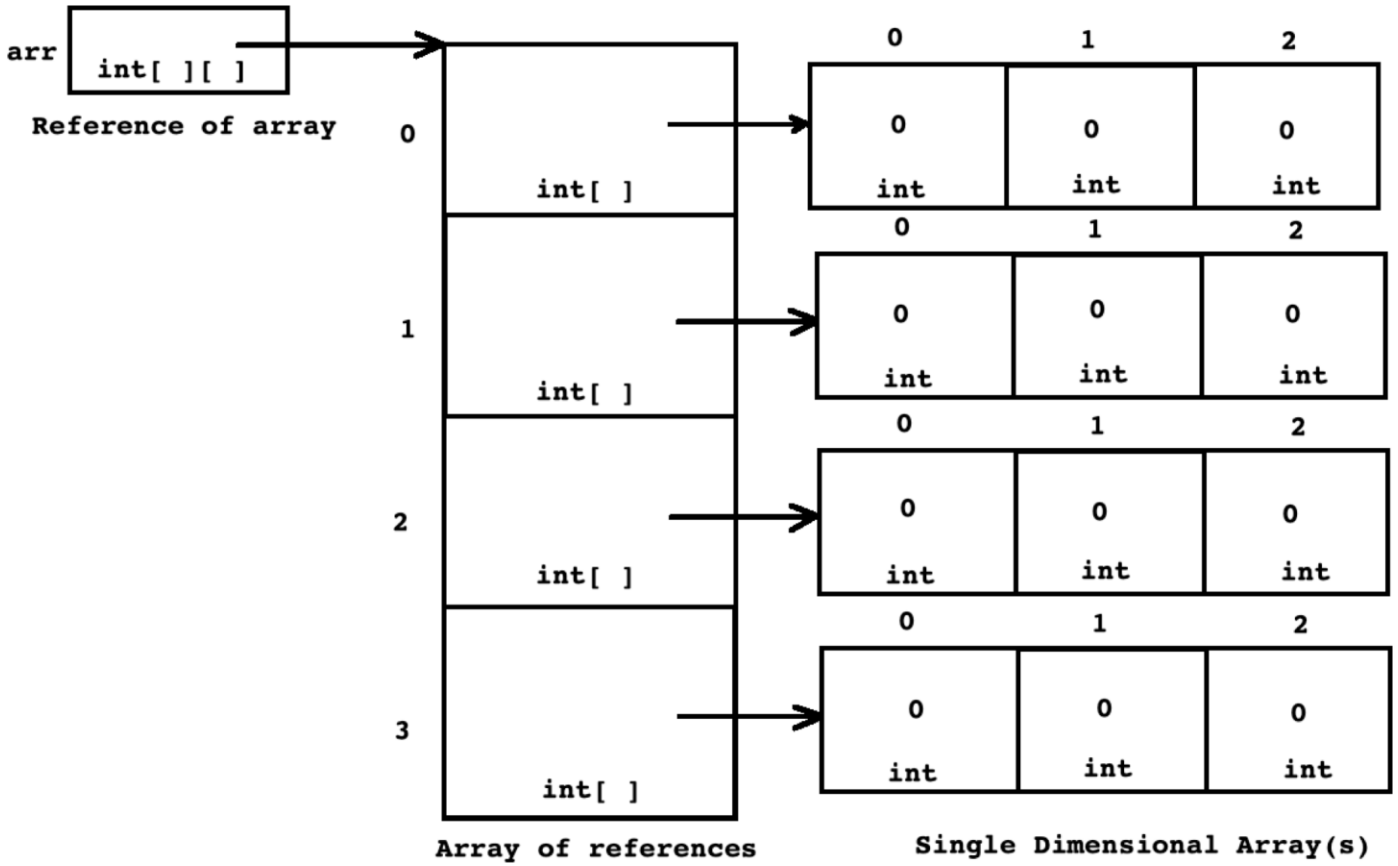
```
int[][] arr = new int[ 2 ][ 3 ];
```

Initialization

```
int[][] arr = new int[ ][ ]{{10,20,30},{40,50,60}}; //OK
int[][] arr = { {10,20,30}, {40,50,60} };    //OK
```

Multi Dimensional Array

+ Multi Dimensional Array



Ragged Array

- A multidimensional array where column size of every array is different.

Reference declaration

```
int arr[][];
int []arr[];
int[][] arr;
```

Array creation

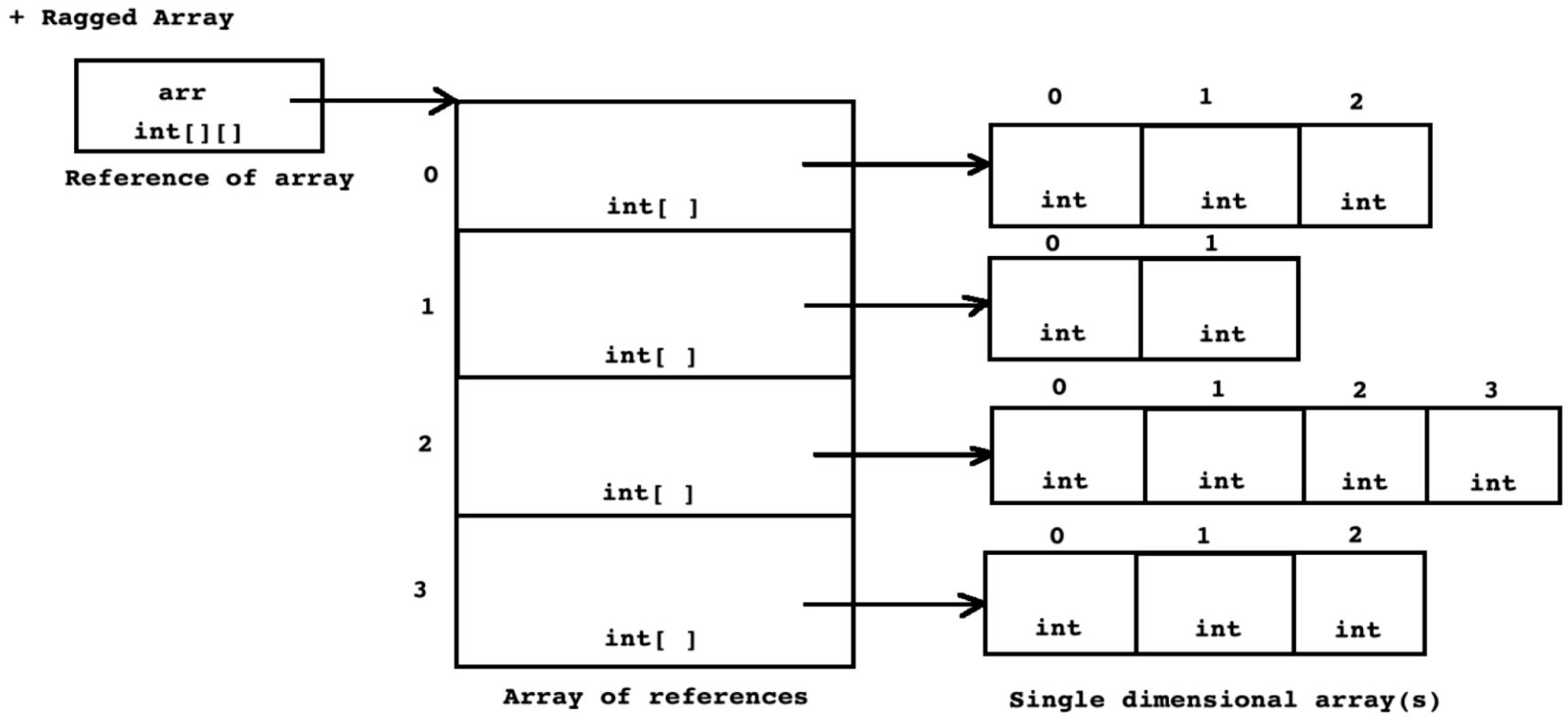
```
int[][] arr = new int[3][];
arr[ 0 ] = new int[ 2 ];
arr[ 1 ] = new int[ 3 ];
arr[ 2 ] = new int[ 5 ];
```

Array Initialization

```
int[][] arr = new int[3][];
arr[ 0 ] = new int[  ]{ 10, 20 };
arr[ 1 ] = new int[  ]{ 10, 20, 30 };
arr[ 2 ] = new int[  ]{ 10, 20, 30, 40, 50 };
```

```
int[][] arr = { { 1, 2 }, { 1, 2, 3 }, {1,2,3,4,5}};
```

Ragged Array



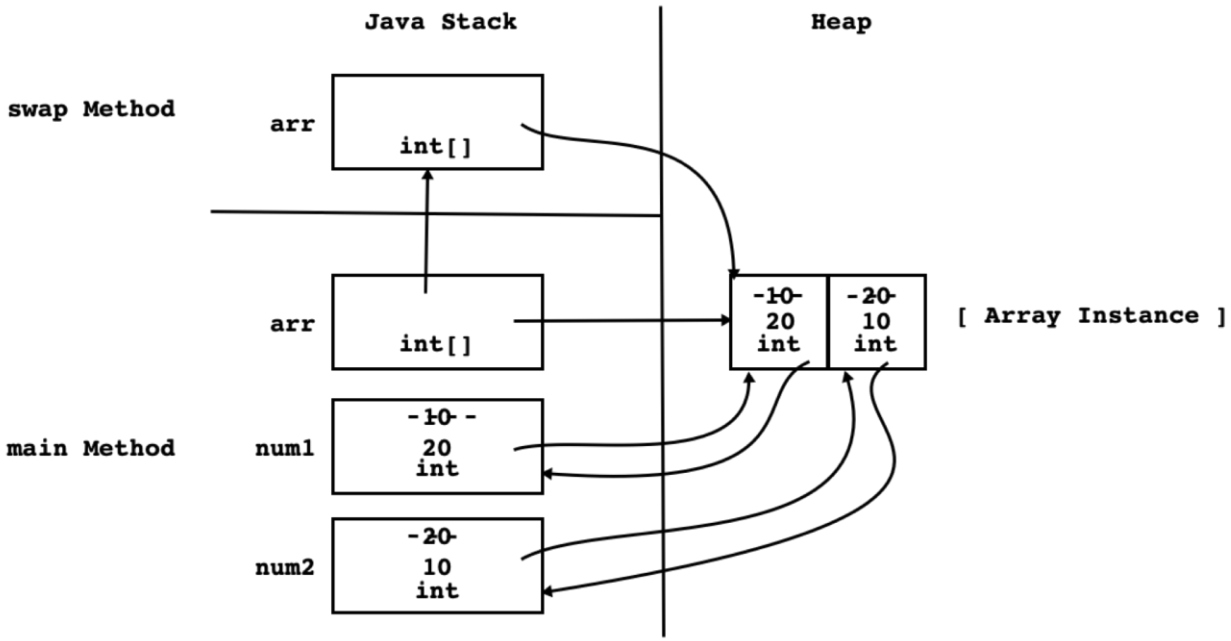
Argument Passing Methods

- In C programming language, we can pass argument to the function using 2 ways:
  1. By value.
  2. By address
- In C++ programming language, we can pass argument to the function using 3 ways:
  1. By value.
  2. By address
  3. By reference
- In Java programming language, we can pass argument to the method using a way:
  1. By value.
     - In other word, every variable of primitive type/non primitive type is pass to the method by value only.

Simulation Of Pass By Reference in Java

```
public class Program {
    private static void swap(int[] arr) {
        int temp = arr[0];
        arr[0] = arr[1];
        arr[1] = temp;
    }

    public static void main(String[] args) {
        int num1 = 10, num2 = 20;
        int[] arr = new int[] { num1, num2 };
        Program.swap(arr);  //passing arr as a argument to the method by value.
        num1 = arr[0];  num2 = arr[1];
        System.out.println("Num1    :    " + num1);  //20
        System.out.println("Num2    :    " + num2);  //10
    }
}
```

Simulation Of Pass By Reference in Java



Thank you.