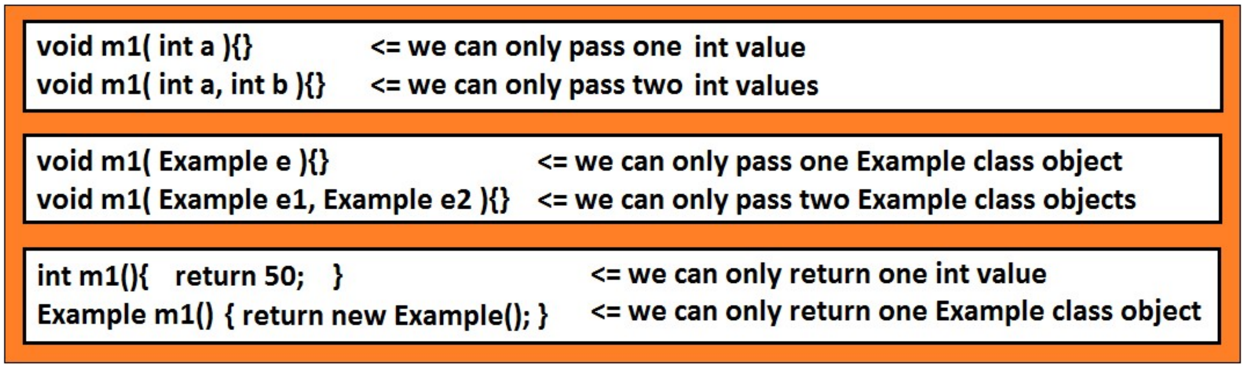
Definition of an Array
An array is a referenced data type used to create fixed number of multiple variables of same type as a group to store multiple values of similar type in continuous memory locations with single variable name.

Need of array
In projects array is used for storing similar type of multiple values or objects and to send all those multiple values or objects at time as single value from one method to another method either as an argument or as a return value.

<u>Let us first understand the problem of variables</u>
Using variable we cannot store multiple values in continuous memory locations. Duet to this limitation we have below two problems

1) Using primitive or class type variables we cannot pass multiple values or objects as one value to a method as argument and also
2) We cannot return multiple values or objects from a method at a time.

For passing multiple values or objects to a method we must define we must define overloaded methods with required number of parameters as shown below.



```
void m1( int a ){}              <= we can only pass one  int value
void m1( int a, int b ){}       <= we can only pass two  int values

void m1( Example e ){}                    <= we can only pass one Example class object
void m1( Example e1, Example e2 ){}       <= we can only pass two Example class objects

int m1(){   return 50;   }                <= we can only return one int value
Example m1() { return new Example(); }    <= we can only return one Example class object
```

Solution
To solve above two problems, we must group all values or objects to send them as a single unit from one application to another application as method argument or return type. To group them as a single unit we must store them in continuous Memory Locations. This can be possible by using referenced data type array.

In Java, Array is a reference data type. It is used to store <u>fixed number</u> of multiple values and objects <u>*of same type*</u> in continuous memory locations.

Note: Like other data types Array is not a keyword rather it is a concept.
It creates continuous memory locations using other primitive or reference types.

Array limitation
Its size is fixed, means we cannot increase or decrease its size after its creation.

Array declaration syntax:
<Accessibility modifier> <Execution-level Modifiers> <datatype>[] <array variable name>;

For Example:

    public static int[] i;
    public static Example[] e;

Rule #1: we can place []
1. after data type
2. before variable name
3. after variable name
4. But not before data type

Rule #2: Like in C or C++, in Java we cannot mention array size in declaration part. It leads CE.
For Example:

    int[5] i;   CE: illegal start of expression
    int[] i;

Find out valid array declaration statements
1. int[] i;
2. int []i;
3. int [] i;
4. int []i;
5. int i[];
6. int[5] i;
7. int i[5] ;
8. []int i;

Below syntax shows multidimensional array declaration

    int[][] i;   <---- two dimensional
    int[][][] i;   <---- three dimensional

Find out valid multidimensional array declarations
1. int[][] i;
2. int[] []i;
3. int[] i[];
4. int [][]i;
5. int []i[];
6. int i[][];

Find out variables type from the below list
1. int i, j; <---- both i, j are of type int
2. int[] i, j; <---- both i, j are of type int[]
3. int i[], j; <---- i is of type int[], and j is of type int
4. int []i1, i2; <---- both i, j are of type int[]
   if we place [] before variable that is applicable to data type not to variable. In this case Compiler moves [] to after data type.
5. int[][] i, j; <---- both i, j are of type int[][]
6. int[] i[], j; <---- i is of type int[][], and j is of type int[]
7. int[] i[][], j; <---- i is of type int[][][], and j is of type int[]

Rule #3: [] is allowed before the variable only for first variable that is placed immediately after data type.

    Ex: int   []p, q[];
           int   []p, []q;

Array object creation
We have three ways to create array object

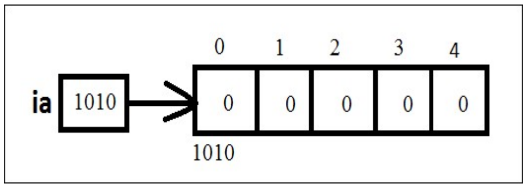
Syntax #1: Array object creation without explicit values or with default values

```
<Accessibility Modifier> <Modifier> <data type>[] <array name> = new <data type>[<size>];
```

Ex #1: Array creation with primitive type
int[] i = new int[5];

From the above statement, array object is created with five int type locations.
All locations are initialized with default value ZERO, because the array object is created with primitive data type int.

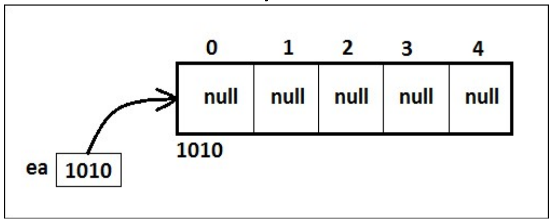
Below diagram shows the memory structure of array object with 5 locations.



Ex #2: Array creation with referenced type
Example[] ea = new Example[5];

From the above statement array object is created with five Example type variables. All locations are initialized with default value ***null***, because the array object is created with referenced data type Example.

Below is the memory location structure



The point to be remembered is in this array object creation statement, Example class objects are not created; rather only Example class type referenced variables are created to store Example class objects further.

Q) How many String objects are created from the below statement?
String[] s = new String[5];

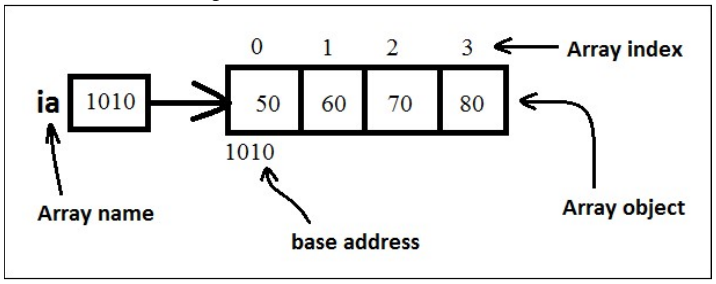
A) **ZERO** String objects are created. It creates ONE String array object with 5 variables of type String with default value "null".

Syntax #2: Array creation with values

```
<Accessibility Modifier> <Modifier> <data type>[] <array name> = {<list of values with , separator>};
```

Example #1: Array creation with primitive data type
int[] ia = {50, 60, 70, 80};

In this array object creation, array contains 4 continuous memory locations with some starting base address assume 1010, and that address is stored in "ia" variable as shown in the below diagram.



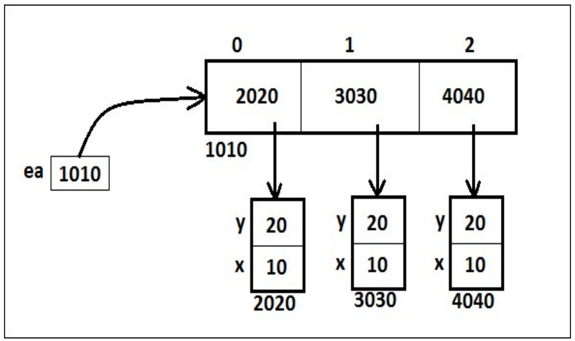
As you noticed, every array location has index starts with ZERO. This array index is used for storing, reading, and modifying array values.

Example #2: Array creation with referenced data type.

```
class Example {
        int x = 10;
        int y = 20;
}
```

Example[] ea = {new Example(), new Example(), new Example()};

In this array object creation, array contains 3 continuous memory locations with some starting base address assume 1010, and that address is stored in "ea" variable as shown in the below diagram.



What is the difference in creating array object with primitive types and referenced types?
As shown in the above diagrams

- If we create array object with primitive type, all its memory locations are of primitive type variables, so values are stored directly in those locations.
- If we create array object with referenced type, all its memory locations are of referenced type variables, so object reference is stored in those locations.

Rules in creating array object

Rule #1: array size is mandatory. If we do not pass array size it leads to **CE:** ***array dimension missing***

Find out CE from the below list

```
int[]    ia1 = new int[5];
int[]    ia2 = new int[];
int[5]   ia3 = new int[];
```

Rule #2: We must specify array size only at array object creation side not at declaration side, it leads CE. Find out CE from the below list

```
int[] i1 = new int[5];
int[5] i2 = new int[];
```

Rule #3: size should be “0” or “+ve int number”. If we pass int data type greater range value or incompatible type value it leads to CE, If we pass “-ve number” program compiles fine but leads to exception "***java.lang.NegativeArraySizeException***"

Find out errors in the below lines of code

```
int[] i1 = new int[3];
int[] i2 = new int[0];
int[] i3 = new int[-5];
int[] i4 = new int['a'];
int[] i5 = new int[10L];
int[] i6 = new int[10.345];
int[] i7 = new int[true];
int[] i8 = new int[“a”];
```

Rule #4: While storing, reading and modifying array values, we must pass array index within the range of [0, array length-1]. If we pass index negative value or value >= length,
it leads to RE: "***java.lang.ArrayIndexOutBoundsException***"

Find out errors in the below lines of code

```
int[] i = new int[5];

i[0]      = 6;
i[1]      = 5;
i[2]      = 4;
i[3]      = 7;
i[4]      = 8;
i[5]      = 9;
i[-3]     = 10;
i[10L]    = 10;
i['a']    = 10;
i[true]   = 10;
```

Find out errors in the below lines of code

```
int[] i1 = new int[2];
int[] i2 = new int[-4];
int[] i3 = new int['a'];
int[] i4 = new int["a"];
int[] i5 = new int[34.5];
int[] i9 = new int[(int)45.34];

int[] i6 = new int[0];
int[] i7 = {};
int[3] i8 = {1,2,3};

System.out.println(i3[91]);
System.out.println(i6[0]);
System.out.println(i3[34.56]);
System.out.println(i3['a']);
System.out.println(i3["a"]);
System.out.println(i1[-1]);
```

Rule #5: Source data type and destination data type must be compatible, else it leads to CE: *incompatible types*

For Example

```
Example[] ea1 = new Example[5];
Example[] ea2 = new Sample[5];
Example[] ea3 = new String[5];
```

"*length*" property

length is a non-static final int type variable. It is created in every array object to store array size. We must use this variable for finding array object size dynamically for retrieving its elements.

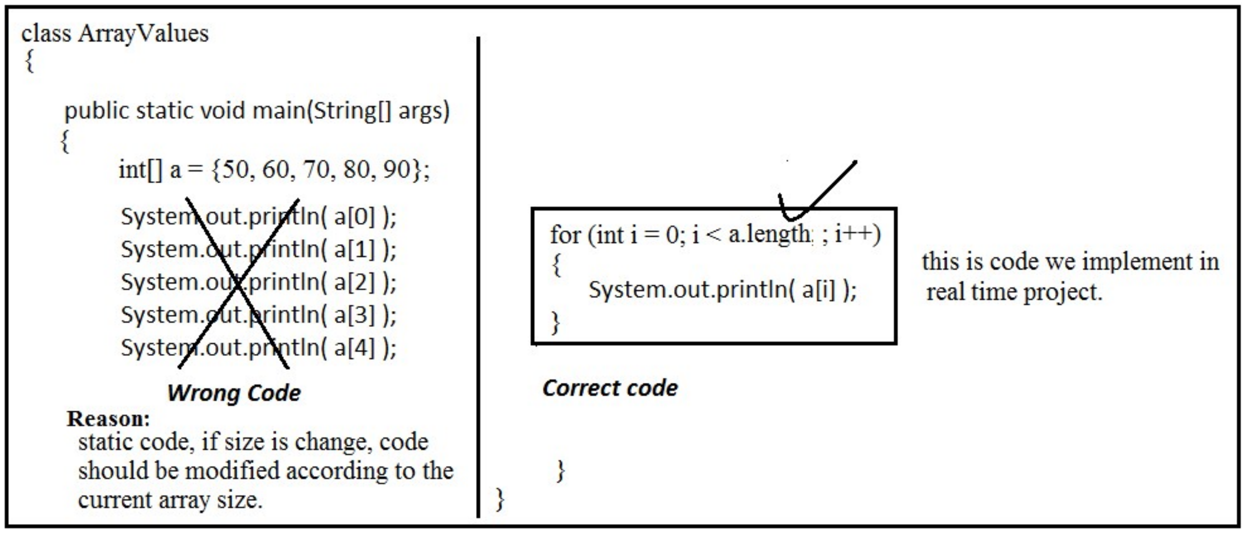
Q) What is the output from the below statement?

```
Thread[] th = new Thread['a'];
System.out.println(th.length); //97
```

Q) How many Thread objects are created from above program?

Zero Thread objects are created. Only one Thread array object is created with 97 locations.

Write a program to create int type array with size 5. Then print its values on console.



```
class ArrayValues
{
    public static void main(String[] args)
    {
        int[] a = {50, 60, 70, 80, 90};

        ~~System.out.println( a[0] );~~
        ~~System.out.println( a[1] );~~
        ~~System.out.println( a[2] );~~
        ~~System.out.println( a[3] );~~
        ~~System.out.println( a[4] );~~

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

Wrong Code

Reason:
static code, if size is change, code should be modified according to the current array size.

Correct code

this is code we implement in real time project.

Q) What is the output of below programs?

```
class Example{
        public static void main(String[] args){
            int[] a;
            for (int i = 0; i < 10; i++){
              a[i] = i * i;
            }
        }
}
```

```
class Example{
        int[] a;
        public static void main(String[] args){
            for (int i = 0; i < 10; i++){
                a[i] = i * i;
            }
        }
}
```

```
class Example{
        static int[] a;
        public static void main(String[] args){
            for (int i = 0; i < 10; i++){
                a[i] = i * i;
            }
        }
}
```

```
class Example{
        static int[] a = new int[5];
        public static void main(String[] args){
            for (int i = 0; i < 10; i++){
                a[i] = i * i;
            }
        }
}
```

Passing Array as Argument
To pass array object as an argument the method parameter type must be the passing array object type or its super class type.

Below program shows passing int[] object as argument
//Example.java
```
class Example{
        static void m1(int[] ia){
                System.out.println("Array Size: "+ia.length);
                System.out.pritln("Its elements");
                for(int i = 0 ; i < ia.length ; i++){
                        System.out.print(a[i] + "\t");
                }
        }
}
```
//Test.java
```
class Test{
        public static void main(String[] args){

                int[] i1= {5, 3, 6, 7};
                Example.m1(i1);

                int[] i2 = new int[5];
                Example.m1(i2);

                Example.m1( new int[7] ); ✓
              //Example.m1( {3, 4 , 5} ); ✗ CE:
```
calling m1() method by passing array object with explicit values

calling m1() method by passing array object with default values

calling m1() method by passing un-referenced array objects

Q) How can we pass an array with user values without referenced variable?
A) Using **Anonymous array**

Syntax to create anonymous array
Combine both array creation syntaxes. **Rule**: Do not mention array size in [].
Its size will be the number of values passing in {} as shown below

```
                Example.m1(new int[] {3, 4, 5} ); ✓

                /* anonymous array can also be created with referenced variable. */
                int[] ia = new int[]{3, 4, 5};
                Example.m1( ia );

                /* But it is not recommended. It is only recommended to pass array as argument with
                explicit values without referenced variable*/
        }
}
```

Q) When we pass array object as argument into a method, if we modify its values, will that modification effect to the original passed-in variable?
A) If we modify array object values with method parameter, the modification is effected to original referenced variable.

What is the output from the below program?

```
class Example{

        static void m1(int[] ia){
                ia[2] = 5;
        }

        public static void main(String[] args){

                int[] ia = {10, 20, 30, 40};
                m1(ia);

                for(int i = 0; i < ia.length ; i++){
                        System.out.print(ia[i] + "\t");
                }
        }
}
```

Q) What is the output if we call m1() method by passing below array?

```
        int[] ia2 = {1,2};
        m1(ia2);
```

A) It leads to AIOBE in m1() method.

Q) Why will we get this exception?

Q) What is the output from the below program?

```
class Example{
        int x = 10;
        int y = 20;

        void m1(){
                x = 5;
        }
}
```

```
class Sample
        static void m2(Example[] e){
                e[2].m1();
        }
}
```

```
class Test{
        public static void main(String[] args){
                Example[] e = {new Example(),
                                        new Example(),
                                        new Example(),
                                        new Example() };
                Sample.m2(e);

                for(int i = 0 ; i < e.length ; i ++){
                        System.out.println(e[i].x);
                        System.out.println(e[i].y);

                        System.out.println();
                }
        }
}
```

Declaring array as final
It is possible to declare array as final
For example:

```
//normal array, means non-final array
int[] ia1 = new int[5];

//final array
final int[] ia2 = new int[5];
```

Q) If we declare array as final, will all its locations are also final?
A) No, only array object referenced variable is final. It means in the above example only "ia2" is final not its array locations. It means we can modify array locations value, but we cannot assign new array object reference to this final referenced variable. It leads compile time error.

Find out CE in the below program. Comment the CE, execute and print output.

```
//ArrayAsFinal.java
class ArrayAsFinal {
	public static void main(String[] args) {

		final int[] ia = new int[5];

		//modifying array referenced variable
		//ia = new int[6]; CE: cannot assign a value to final variable ia

		//modifying array locations value
		ia[1] = 5;
		ia[2] = 6;

		//printing array locations value
		for (int i = 0; i < ia.length ; i++){
			System.out.println( "ia[" + i + "] --> " +  ia[i] );
		}
	}
}
```

Output

```
ia[0] --> 0
ia[1] --> 5
ia[2] --> 6
ia[3] --> 0
ia[4] --> 0
```

Q) Can we declare array locations as final?
A) No, because we are not creating array locations.

Q) Can we declare a class referenced variable as final?
A) Yes, in this case also only that referenced variable is final but not the instance variables available in object.

```
	final Example e = new Example();
	- Here "e" is only final, but not "x and y" variables
```

Q) Can we declare a class object's variables as final, in the above case x, y can we declare as final?
A) Yes it is possible, because those variables are created by us. We must declare them as final in the class definition.

```
class Example {
  int x = 10;
  final int y = 20;
 public static  void main(String[] args) {
    final Example e1 = new Example();
    e1 = new Example();
    e1.x = 5;
    e1.y = 6;
 }
}
```

Types of array referenced variables

Like primitive variables and other referenced variables, we can also create array referenced variable as

1) static → for storing an array object common to all objects & all methods of current class
2) non-static →for storing an array object separately for each object of this class
3) parameter →for receiving an array object as an argument of this method/ this constructor
4) local → for storing an array object specific to one method or one constructor

- If we create array object with static referenced variable, it is created at the time of class loading. That static referenced variable is created in method area and its array objet is created in heap area.
- If we create array object with non-static referenced variable, it is created at the time of that enclosing class object creation. That referenced variable is created in heap area in that enclosing class object and array object is also created in heap area.
- If we create array object with parameter or with local referenced variable, it is created when that enclosing method is called. That referenced variable is created in that method's stack frame and object is created in heap area.

Check below program, it has CE, comment it, then run and print out also draw JVM Architecture

```
//Test.java
class Test{
        static int[] ia1 = new int[5];
        int[] ia2 = {40, 50, 60, 70};

        public static void main(String[] args) {
                int[] ia3 = new int[3];

                System.out.println(ia1[1]);
                System.out.println(ia2[1]);
                System.out.println(ia3[1]);

                Test t = new Test()
                System.out.println(t.ia2[1]);
        }
}
```

Q) If we create array object of a class is its class byte codes are loaded into JVM?

Yes, but SV, SBs are not executed. If we create array object from class Example, Example class byte codes is loaded into JVM, but its SVs and SBs are not executed. If at all in that array object if you create "Example object", then Example class SVs an SBs are executed.

For example:

```
class Example{
        static{
            Sopln("Example is loaded");
        }
        Example(){
            Sopln("Example object is created");
        }
}
```

Below statement ***loads*** Example class but doesn't execute SV and SB from class Example

Example[] e = new Example[5];

Output: no output

Below statement ***loads*** Example class

Example[] e = {new Example(), new Example()};

Output: Example is loaded

```
Example object is created
Example object is created
```

Check below program, give output with JVM architecture. Find out CE, RE in Test.java

```
//Example.java
class Example{

	int x = 10; int y = 20;

	static{
		System.out.println("Example is loaded");
	}

	Example(){
		System.out.println("Example object is created");
	}
}

//Test.java
class Test{
	static Example[] e1 = new Example[5];
	Example[] e2 = {new Example(), new Example()};

	public static void main(String[] args)  {

		System.out.println("Test main");
		Example[] e3 = new Example[2];

		System.out.println(
			"e3 array object is created");

		e1[1] = new Example();
		e3[1] = new Example();
		System.out.println(e1[1].x);
		System.out.println(e2[1].x);
		System.out.println(e3[1].x);

		Test t = new Test()
		System.out.println(t.e2[1].x);

		System.out.println(e1[0].x);
		System.out.println(t.e2[0].x);
		System.out.println(e3[0].x);
		System.out.println(t.e1[1].y);
	}
}
```

Types of Arrays based on dimensions

Java supports two types of arrays

1. Single dimensional arrays <- stores normal objects and values
2. Multidimensional arrays <- stores array objects

- Single dimensional array is also called "*array of objects or values*"
- Multi dimensional array is also called "*array of arrays*".

Note: In Java, multidimensional array is *array of arrays*.
Basically multidimensional arrays are used for representing data in table format.

Below is the syntax to create two dimensional array.

Two dimensional array

int[][] ia = new int[3][2];

From this statement we can say

- one parent array is created with 3 locations and
- three child arrays are created with 2 locations.

Parent array size gives below information

1. parent array's number of locations
2. number of child arrays

Below is the array object diagram for this two dimensional array.

Below is the table format diagram for the above two dimensional array object.

Below syntax is two dimensional array object creation with explicit values

Int[][] ia = { {5, 6, 7}, {8, 9, 10}, {11, 12, 13} };

Below is the array object diagram for this two dimensional array.

Below is the table format diagram for the above two dimensional array object.

Rule: Base array size is mandatory where as child array size is not mandatory.

For Example

```
int[][] ia = new int[3][];
//int[][] ia = new int[][2]; ->CE: missing array dimension
```

- If *we do not pass* child array size, child array objects are not be created, only parent array object is created. Later developer has to pass array objects.
- If *we pass* child array size, all child arrays are created with same size. If we want to create child arrays with different sizes we must not pass child array size.

Jogged arrays
Multidimensional array with different sizes of child arrays is called Jogged array. It creates a table with different sizes of columns in a row. To create jogged array, in multidimensional array creation we must not specify child array size instead we must assign child array objects with different sizes as shown in the below diagram.

For Example:

```
int[][] ia = new int[3][]; -> base array is created with size 3
ia[0] = new int[2];
ia[1] = new int[3];
ia[2] = new int[4];
ia[0][0] = 50;
ia[2][3] = 70;
```

Below is the array object diagram for this two dimensional array.

Below is the table format diagram for the above two dimensional array object.

Draw Memory location for below three dimensional array object

```
int[][][] ia = new int[4][3][2];
int[][][] ia = {  { {4, 5}, {3, 2}, null, {1, 2}}  };
```

Below program shows printing multi dimensional array elements in table format
//MultiDimentionalArrayPrinter.java

```
class MultiDimentionalArrayPrinter {
	public static void main(String[] args) {

		int[][] ia = { {5, 6, 7}, { 4, 3}, {1}, {4, 5, 7, 8, 9}};


		for (int i = 0; i < ia.length ; i++ ){
			for (int j = 0; j < ia[i].length ; j++ ){
				System.out.print(ia[i][j] + "\t");
			}
			System.out.println();
		}
	}
}
```

Array casting & exception in array casting

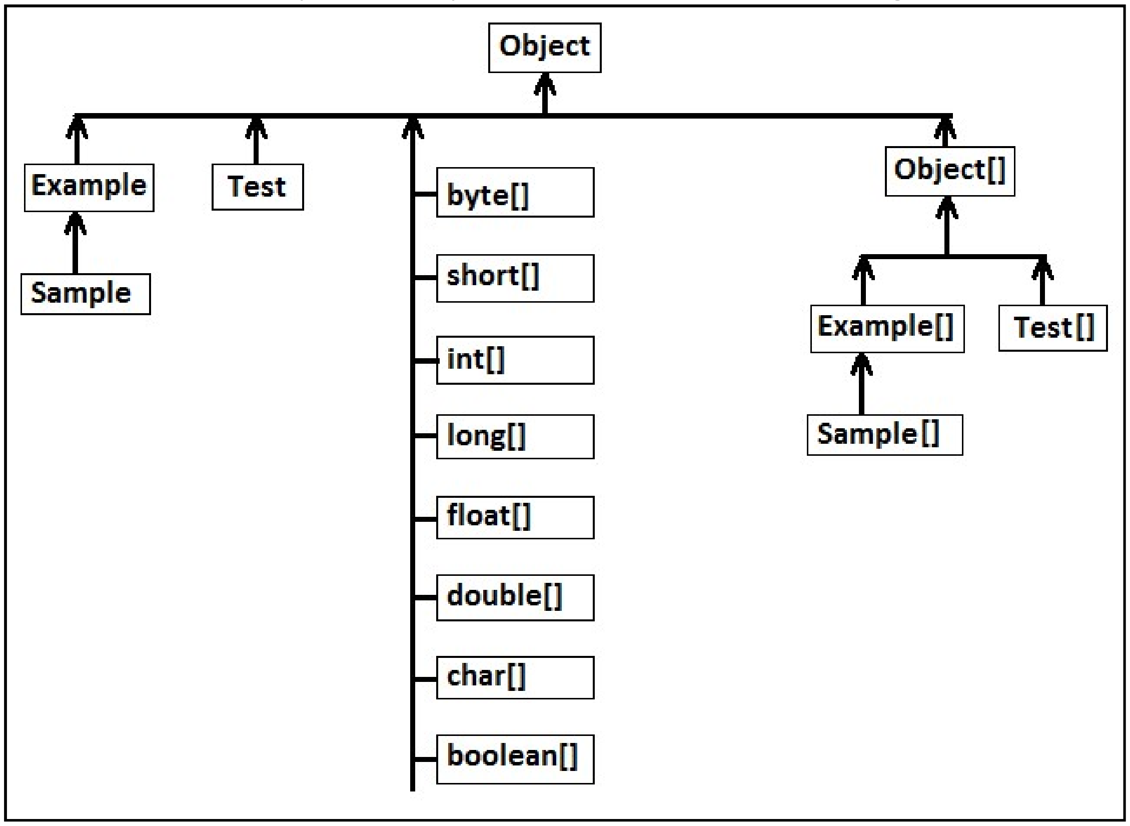
1. For all referenced types and arrays including primitive arrays java.lang.Object is super class.
2. For every array object, JVM internally creates a class with name "*datatype[]*".

For example

- the class for int type array is "int[]"
- the class for Example type array is "Example[]"
- the class for Object type array is "Object[]"

3. For all array objects created with referenced types the super class is "Object[]", which is subclass of "Object"
4. The array objects created with primitive types are not compatible with each other and their super class is "Object" not "Object[]".

Now observe below hierarchy and strictly remember and follow it in solving next bits.



Find out compile time errors in below list of array objects conversion

1.	int[]	i1	= new int[5];
2.	int[]	i2	= new short[5];
3.	Object[]	obj	= new short[5];
4.	Object	obj1	= new int[5];
5.	Object	obj2	= new Object[5];
6.	Example	e1	= new Example[5];
7.	Object[]	obj	= new Example[5];
8.	Object	obj	= new Example[5];
9.	Example[]	ea	= new Sample[5];
10.	Example[]	ea	= new Test[5];

SCJP Question:
Given:

```
11. public static void main(String[] args) {
12. Object obj = new int[] { 1, 2, 3 };
13. int[] someArray = (int[])obj;
14. for (int i : someArray) System.out.print(i + " ");
15. }
```

What is the result?
A. 1 2 3
B. Compilation fails because of an error in line 12.
C. Compilation fails because of an error in line 13.
D. Compilation fails because of an error in line 14.
E. A ClassCastException is thrown at runtime.

java.lang.ArrayStoreException

JVM throws this exception, if it identifies the incompatible object is passed to store in array location. If this problem is identified by compiler, it throws CE: "incompatible types".

Check below example

```
        Object[] obj = new Example[5];
```

In this statement compiler thinks "Example array object is created with five locations and is stored in Object[] variable". This assignment is allowed as Example[] is a subclass of Object[].

1. obj[0] = new Example();
2. obj[1] = new Sample();
3. obj[2] = new Test();

At third line we experience "*java.lang.ArrayStoreException*", because, Test object is not compatible with Example.
It is not identified by compiler because it checks only type of referenced variable. In this case it considered obj[0], obj[1], obj[2] variables are of type java.lang.Object not Example. So it allows assignment. Actually they are of type Example, it is only known to JVM.

Find out CE and RE in the below lines of code

1. Example[] e = new Sample[5];

2. e[1]   = new Sample();
3. e[2]   = new Test();
4. e[3]   = new Example();
5. e[4]   = new Example[2];

In the below program what is the argument we must pass to execute else block.

```
class Example{
        static void m1(Object obj){
                if (obj instanceof Object){
                        System.out.println("If");
                }
                else{
                        System.out.println("Else");
                }
        }
}
```

Var-arg parameter method

```
class Addition{

        static void add(){
                System.out.println("no-arg");
        }
        static void add(int a){
                System.out.println("1 int-arg");
        }
        static void add(int a, int b){
                System.out.println("2 int-arg");
        }
        static void add(int a, int b, int c){
                System.out.println("3 int-arg");
        }
        static void add(int a, int b, int c, int d){
                System.out.println("4 int-arg");
        }
}
```

```
class Test{
        public static void main(String[] args) {

                Addition.add();
                Addition.add(5);
                Addition.add(5, 6);
                Addition.add(5, 6, 7);
                Addition.add(5, 6, 7, 8);

        }
}
```

```
class Addition{

        static void add(int[] a){
            System.out.println(a.length +"
                            values array is passed");
            for(int i = 0; i < a.length;  i ++){
                System.out.println("   Value "
                                +(i + 1)+" is:"+a[i]);
            }
        }
}
```

```
class Test{
      public static void main(String[] args) {

          Addition.add();
          Addition.add( new int[]{            }   );
          Addition.add( new int[]{  5         }   );
          Addition.add( new int[]{  5, 6      }   );
          Addition.add( new int[]{  5, 6, 7   }   );
          Addition.add( new int[]{  5, 6, 7, 8 }  );
      }
}
```

```
class Addition{

        static void add(int… a){
            System.out.println(a.length +"
                            values array is passed");
            for(int i = 0; i < a.length;  i ++){
                System.out.println("   Value "
                                +(i + 1)+" is:"+a[i]);
            }
        }
}
```

```
class Test{
        public static void main(String[] args) {

          Addition.add();
          Addition.add(5);
          Addition.add(5, 6);
          Addition.add(5, 6, 7);
          Addition.add(5, 6, 7, 8);
          Addition.add( new int[]{5, 6, 7, 8, 9} );
        }
}
```

```
class AdditionWithArrayParam {
        static void add(int[] a){
                if ( a.length == 0){
                        System.out.println("Values are not passed");
                }else{
                        int sum = 0;
                        for (int i = 0; i < a.length; i++){
                                sum = sum + a[i];
                        }
                        System.out.println("Result: "+sum );
                }
        }
        public static void main(String[] args) {
                add( new int[]{} );
                add( new int[]{5} );
                add( new int[]{ 5, 6 } );
                add( new int[]{ 5, 6, 7 } );
                add( new int[]{ 5, 6, 7, 8 } );
        }
}
```

```
class AdditionWithVarArgParam {
        static void add(int... a){
                if ( a.length == 0){
                        System.out.println("Values are not passed");
                }else{
                        int sum = 0;
                        for (int i = 0; i < a.length; i++){
                                sum = sum + a[i];
                        }
                        System.out.println("Result: "+sum );
                }
        }
        public static void main(String[] args) {
                add(  );
                add( 5 );
                add( 5, 6  );
                add( 5, 6, 7  );
                add( 5, 6, 7, 8  );

                add(  new int[]{5,6,7, 8, 9});
        }
}
```

```
//PassingDynamicValues.java
import java.util.*;

class PassingDynamicValues {

        static void add( int... a ){
                if(a.length == 0){
                        System.out.println("Values are not passed");
                }
                else{
                        int sum = 0;
                        for(int i = 0 ; i < a.length ; i++){
                                sum += a[i];
                        }
                        System.out.println("Result: "+sum );
                }
        }

        public static void main(String[] args) {

                Scanner scn = new Scanner(System.in);

                System.out.print("Enter num1: ");
                int n1 = scn.nextInt();

                System.out.print("Enter num2: ");
                int n2 = scn.nextInt();

                System.out.print("Enter num3: ");
                int n3 = scn.nextInt();

                add( n1, n2, n3 );

        }
}
```

```
//PassingDynamicValues.java
import java.util.*;

class PassingDynamicValues {

        static void add( int... a ){
                if(a.length == 0){
                        System.out.println("Values are not passed");
                }
                else{
                        int sum = 0;
                        for(int i = 0 ; i < a.length ; i++){
                                sum += a[i];
                        }
                        System.out.println("Result: "+sum );
                }
        }

        public static void main(String... args) {

                Scanner scn = new Scanner(System.in);

                System.out.print("How many values you want input: ");
                int noOfValues = scn.nextInt();

                int[] inputValues = new int[noOfValues];

                for (int i = 0; i < noOfValues; i++){
                        System.out.print("Enter num"+(i+1)+": ");
                        inputValues[i] = scn.nextInt();
                }

                add( inputValues );     //inputValues ={n1, n2, n3, n4, n5, ...}

        }
}
```

```
//VarArgRTvaluesTest.java
import java.util.*;

class A{
        void m1(int... ia){
                System.out.println(ia.length + " values are passed");
                for ( int i : ia ){
                        System.out.println("  "+i);
                }
        }
}
class VarArgRTvaluesTest{
        public static void main(String[] args) {
                Scanner scn = new Scanner(System.in);

                System.out.println("Enter numbers with space separator: ");
                String input = scn.nextLine();
                int[] iNums      = null;

                if(input.isEmpty()){
                        iNums = new int[0];
                }
                else{
                        //Splitting given sting into individual numbers
                        String[] sNums = input.split(" ");

                        //Creating array with size equal to given nums
                        iNums                   = new int[sNums.length];

                        if(sNums.length != 0){
                                //Copying nums from sNums to iNums array
                                // by converting numbers from String form to int form
                                for(int i = 0; i < sNums.length; i ++){
                                        iNums[i] = Integer.parseInt( sNums[i] );
                                }
                        }
                }//if-else(empty)

                //invoking var-arg method by passing array with '0 - n' number of values
                A a1 = new A();
                a1.m1( iNums );

        }
}
```

```
//VarArgReadingNamesRT.java
import java.util.*;

class NamesPrinter{
        static void print(String... names) {
                System.out.println(names.length +" names are passed");
                System.out.println("They are: ");

                for (String name : names ){
                        System.out.println("   "+name);
                }
        }
}

class VarArgReadingNamesRT{
        public static void main(String[] args){
                Scanner scn = new Scanner(System.in);

                System.out.println("Enter names with space separator");
                String input = scn.nextLine();

                if( input.isEmpty() ){
                        NamesPrinter.print();
                }else{
                        String[] names = input.split(" ");
                        NamesPrinter.print( names );
                }
        }
}
```

```
//VarArgRules.java
class Employee{}

class VarArgRules{

//Rule #1:
        int... ia;
        void m2(int... ia){}

        void m2(String... ia){}
        void m2(Integer... ia){}
        void m2(Thread... ia){}
        void m2(Employee... ia){}
        void m2(Object... ia){}
        void m2(Class... ia){}

//Rule #2:
        void m1(int... ia){}
        void m1(int.. ia){}
        void m1(int.... ia){}

//Rule #3:
        void m3(int... ia){}
        void m4(int ...ia){}
        void m5(int ia...){}
        void m5(...int ia){}
        void m6(int...ia){}

//Rule #4:
        void m7(int... ia){}
        void m7(int...... ia){}
        void m7(int[]... ia){}
        void m7(int[][]... ia){}

        void m7(int...[] ia){}
        void m7(int[]...[] ia){}

        void m8(int[]ia[]){}
        void m9(int[]ia...){}
```

```
//Rule #5
        static void m10(int... ia, int a, int b){
        static void m10(int a, int... ia, int b){

        static void m10(int a, int b, int... ia){
                System.out.println("\na : "+a);
                System.out.println("b : "+b);
                System.out.println("ia: "+java.util.Arrays.toString(ia));
        }
        //m10();    m10(5);    m10(5, 6);    m10(5, 6, 7);    m10(5, 6, 7, 8);

//Rule #6:
        static void m11(int... a, int... b){}
        static void m11(int... a, String... b){}

//Rule #7:
        static void m12()       { System.out.println("no-param method");}
        static void m12(int a)  { System.out.println("int-param method");}
        static void m12(int... a){ System.out.println("int var-arg method");}

        //m12();    m12(5);    m12(5, 6);    m12(5, 6, 7);

//Rule #8:
        static void m13(int... a)       { System.out.println("int var-arg method");}
        static void m13(long... a)      { System.out.println("long var-arg method");}
        static void m13(float... a)     { System.out.println("float var-arg method");}

        //m13(5);    m13(5L);    m13(5F);    m13(5, 6);

        static void m14(int... a)       { System.out.println("int var-arg method");}
        static void m14(boolean... a)  { System.out.println("boolean var-arg method");}

        //m14();    m14(5);    m14(true);

//Rule #9:
        static void m15(int... a)       { System.out.println("int var-arg method");}
        static void m15(float... a)     { System.out.println("float var-arg method");}
        //m15();    m15(5);    m15(5L); m15(5F);

//Rule #10:
        static void m16(int a)          { System.out.println("int param method");}
        static void m16(int... a)       { System.out.println("int var-arg method");}
        //m16();  m16(5);  m16(5, 6); m15( new int[]{5});
//Rule #11:
        static void m17(int[] a){ System.out.println("int[] arg method");}
        static void m18(int... a){ System.out.println("int var-arg method");}

//Rule #12:
        static void m19(int[] a){ System.out.println("int[] arg method");}
        static void m19(int... a){ System.out.println("int var-arg method");}

        public static void main(String[]args) {
                m10();
                m10(5);
                m10(5, 6);              //=>m10(5, 6, new int[0]);
                m10(5, 6, 7);           //=>m10(5, 6, new int[]{7});
                m10(5, 6, 7, 8);        //=>m10(5, 6, new int[]{7, 8});

                m12();                                  m13();
                m12(5);                                 m13(5);
                m12(5,6);                               m13(5L);
                m12(5,6,7);                             m13(5F);
                                                        m13(5, 5L);
                                                        m13(5, 5L, 5F);
                m14();
                m14(5);
                m14(true);

                m15();                  //=>m15( new int[0]          );
                m15(5)                  //=>m15( new int[]{5});
                m15(5L);                //=>m15( new float[]{5.0F}    );

                m16(5);
                m16( new int[]{5});

                m17();
                m17(5);
                m17( new int[0] );
                m17( new int[]{5} );

                m18();
                m18(5);
                m18( new int[0] );
                m18( new int[]{5} );

        }//main close
}//class close
```

```
//Rule #13:
class A{
        void m1(int... ia){
                System.out.println(“A int…”);
        }
}
class B extends A{
        void m1(long... ia){ {
                System.out.println(“B long…”);
        }
}
class Test{
        public static void main(String[] args){
                B b1 = new B();
                b1.m1();
                b1.m1(5);
                b1.m1(5L);

                A a1 = new B();
                a1.m1();
                a1.m1(5);
                a1.m1(5L);
        }
}
```

```
//Rule #13:
class A{
        void m1(long... ia){
                System.out.println(“A long…”);
        }
}
class B extends A{
        void m1(int... ia){ {
                System.out.println(“B int…”);
        }
}
class Test{
        public static void main(String[] args){
                B b1 = new B();
                b1.m1();
                b1.m1(5);
                b1.m1(5L);

                A a1 = new B();
                a1.m1();
                a1.m1(5);
                a1.m1(5L);
        }
}
```

```
//Rule #14:
class A{
        void m1(int... ia){
                System.out.println(“A int…”);
        }
}
class B extends A{
        void m1(int... ia){ {
                System.out.println(“B int…”);
        }
}
```

```
class Test{
        public static void main(String[] args){
                B b1 = new B();
                b1.m1();
                b1.m1(5);
                b1.m1(5L);

                A a1 = new B();
                a1.m1();
                a1.m1(5);
                a1.m1(5L);
        }
}
```

```
//Rule #15:
class A{
        void m1(int... ia){
                System.out.println("A int…");
        }
}
class B extends A{
        void m1(int[] ia){ {
                System.out.println("B int[]");
        }
}
```

```
class Test{
        public static void main(String[] args){
                B b1 = new B();
                b1.m1();
                b1.m1(5);
                b1.m1( new int[]{5});

                A a1 = new B();
                a1.m1();
                a1.m1(5);
                a1.m1( new int[]{5});
        }
}
```

```
//Rule #16:
class A{
        void m1(int[] ia){
                System.out.println("A int[]");
        }
}
class B extends A{
        void m1(int… ia){ {
                System.out.println("B int…");
        }
}
```

```
class Test{
        public static void main(String[] args){
                B b1 = new B();
                b1.m1();
                b1.m1(5);
                b1.m1( new int[]{5});

                A a1 = new B();
                a1.m1();
                a1.m1(5);
                a1.m1( new int[]{5});
        }
}
```

Java7 annotation: @SafeVarArg

```
@SafeVarargs // Not actually safe!
 static void m(List<String>... stringLists) {
   Object[] array = stringLists;
   List<Integer> tmpList = Arrays.asList(42);
   array[0] = tmpList; // Semantically invalid, but compiles without warnings
   String s = stringLists[0].get(0); // Oh no, ClassCastException at runtime!
 }
```

Garbage collection with arrays

If array object is unreferenced, then all its internal objects are also unreferenced. So we can say when an array object is unreferenced not only array object, all its internal objects are eligible for garbage collection provided they do not have explicit references from other referenced variables.

For example:

```
Example[] e = {new Example(), new Example(), new Example(), new Example()};
```

e[1] = null;

- at this line number, e[1] referencing object is eligible for garbage collection.

e = null;

- at this line number, all 5 objects, including array object, are eligible for garbage collection.

Q) In the below program how many objects are eligible for gc?

```
class Test{
        static void m1(Example[] e){
                e[1] = null;
                e = null;
        }
        public static void main(String[] args){
                Example[] e = new Example[5];
                e[0] = new Example();
                e[1] = new Example();

                e[2] = new Example();
                Example e1 = new Example();
                e[3] = e1;

                line #1: e1 = null;
                line #2: m1(e);
                line #3: e = null;
        }
}
```