Build COMPETENCY
across your TEAM

SYNERGETICS
GET IT RIGHT

Microsoft Partner
Gold Cloud Platform
Silver Learning

Arrays, Collection and Generics

Google Cloud Platform

Windows Azure

amazon web services

APACHE kafka
A distributed streaming platform

Apache Software Foundation

APACHE Spark

Java

hadoop

Context And Dependency Injection

SYNERGETICS GET IT RIGHT | Microsoft Partner Gold Cloud Platform Silver Learning

- One dimensional array
- Multidimensional array
- Using varargs
- Using Arrays class
- Collections Framework
  - Collection Interfaces
  - Implementing Classes
  - Iterating Collections (using foreach & iterator)
  - Comparable and Comparator
- Generics
  - Writing Generic Classes
  - Using Generics with Collections

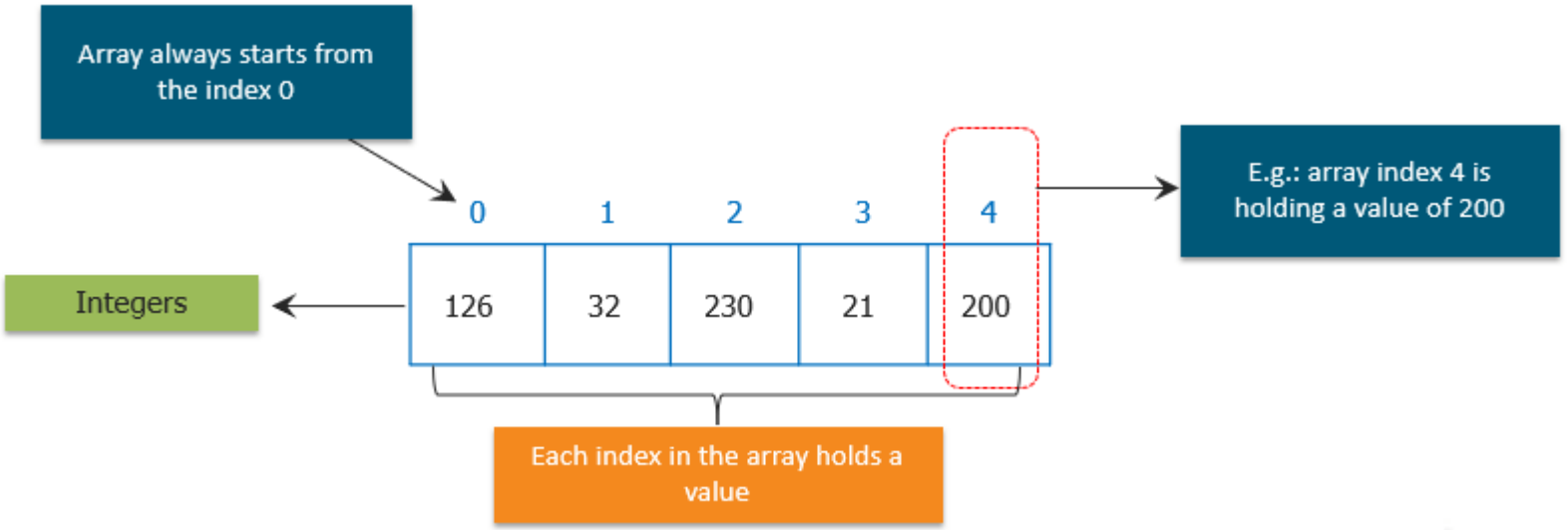
SYNERGETICS GET IT RIGHT | Microsoft Partner Gold Cloud Platform Silver Learning

One dimensional array

- Arrays are an important structure to hold data.
- Java allows us to hold many objects of the same type using arrays.
- It can be used with the help of a loop to access the elements by their index.

What are Java Arrays?

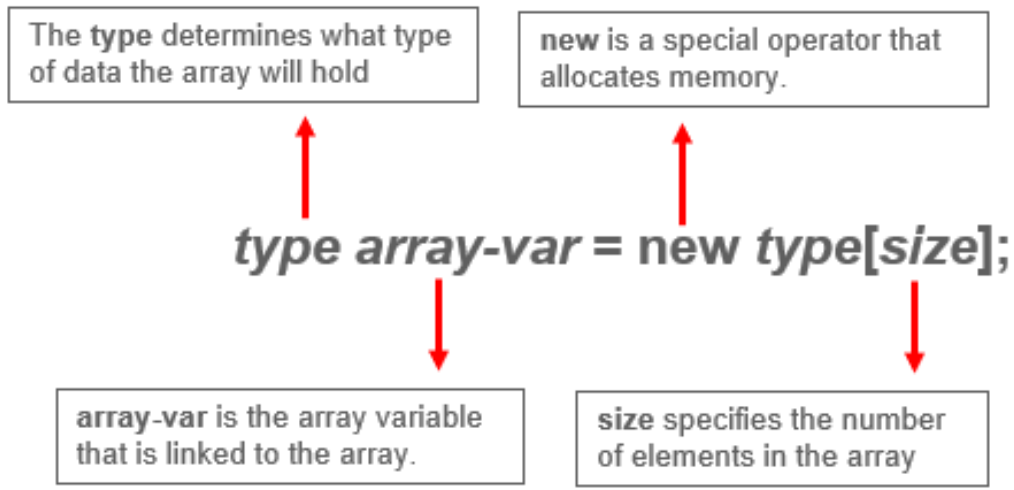
- Arrays in Java are homogeneous data structures implemented in Java as objects. Arrays store one or more values of a specific data type and provide indexed access to store the same.
- A specific element in an array is accessed by its index.
- Arrays offer a convenient means of grouping related information.



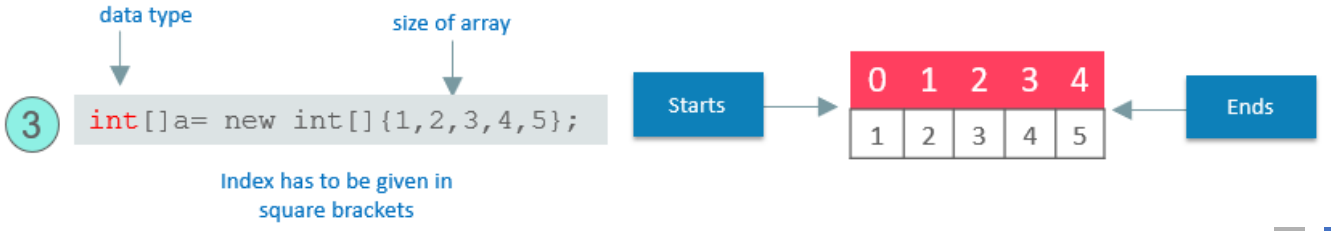
Obtaining an array is a two-step process.

- First, you must declare a variable of the desired array type
- Second, you must allocate the memory that will hold the array, using new, and assign it to the array variable

General Form of Java Array Initialization



- Arrays can be initialized when they are declared. The array will automatically be created large enough to hold the number of elements you specify in the array initializer. There is **no** need to use **new**

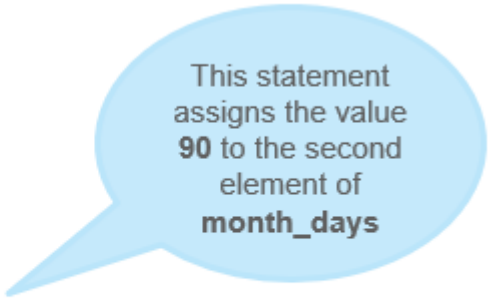


```
class MyArray{
        public static voide main(String args[]){
                int month_days[ ] = {31,28,31,30,31,30,31,30,31,30,31};
                System.out.println("April has " + month+days[3] + "days.");
        }
}
```

Accessing a Specific Element in a Java Array

- In arrays, we can access the specific element by its index within square brackets.

```
public static void main(String args[]) {
   int month_days[];
    month_days = new int[12];
    month_days[0] = 31;
    month_days[1] = 28;
    month_days[2] = 31;
    month_days[3] = 30;
    month_days[4] = 31;
    month_days[5] = 30;
    month_days[6] = 31;
    month_days[8] = 30;
    month_days[9] = 31;
    month_days[10] = 30;
    month_days[11] = 31;
     System.out.println("April has " + month_days[3] + " days.");
     }
}
```



month_days[1] = 90;

SYNERGETICS GET IT RIGHT | Microsoft Partner Gold Cloud Platform Silver Learning

Multidimensional array

- Multidimensional arrays are arrays of arrays.

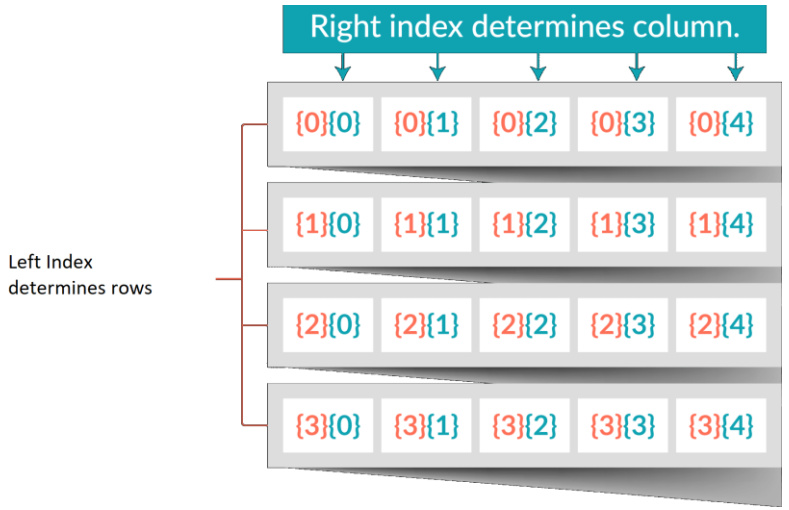
Declaring Multidimensional Array

- To declare it, we have to specify each additional index using another set of square brackets.

- Conceptually, the array declared above would be represented as shown in the figure

This allocates a **4** by **5** array and assigns it to **Mul.**

int Mul[ ][ ] = new int[4][5];



- The following program, numbers each element in the array from left to right, top to bottom, and then displays these values:

```
class Mul2D{
 public static void main(String args[]) {
       int mul2d[][]= new int[4][5];
        int i, j, k = 0;
     for(i=0; i<4; i++)
       for(j=0; j<5; j++) {
       Mul2D[i][j] = k;
       k++;
}
for(i=0; i<4; i++) {
    for(j=0; j<5; j++);
       System.out.print(mul2d[i][j] + " ");
       System.out.println();
       }
   }
}
```

These are other Multidimensional arrays representation of other data types.

```
int [][]a= new int [2][2];
```

	0	1
0	1	4
1	4	5

2 x 2 dimensional int array

```
char [][]a= new char[3][2];
```

	0	1
0	s	a
1	g	v
2	v	d

```
float [][]a= new float[5][5];
```

	0	1	2	3	4
0	2.2	3.4	5.0	3.3	1.2
1	7.8	9.0	1.1	2.9	5.5
2	2.0	3.0	7.8	9.8	9.9
3	5.7	6.6	8.8	5.3	2.7
4	1.8	4.4	7.6	1.0	1.1

5 x 5 dimensional float array

Using varargs

Variable Arguments

- In JDK 5, Java has included a feature that simplifies the creation of methods that need to take a variable number of arguments.
- This feature is called varargs and it is short-form for variable-length arguments.
- A method that takes a variable number of arguments is a varargs method.
- Prior to JDK 5, variable-length arguments could be handled two ways.
- One using overloaded method(one for each) and another put the arguments into an array, and then pass this array to the method. Both of them are potentially error-prone and require more code.
- The varargs feature offers a simpler, better option.

Variable Arguments

- A variable-length argument is specified by three periods(...)

```
public static void fun(int ... a)
{
   // method body
}
```

This syntax tells the compiler that fun( ) can be called with zero or more arguments. As a result, here a is implicitly declared as an array of type int[].

Variable Arguments

```
Class Calculator{
        public void add(int num1, int num2){
                return num1 + num2;
        }
        public void add(int num1, int num2, int num3){
                return num1 + num2 + num3;
        }
}
```

```
Class Calculator{
        public void add(int ...nums){
                int sum = 0;

                for( num : nums){
                        sum  += num;
                }
                return sum;
        }
}
```

Important points:

- Vararg Methods can also be overloaded but overloading may lead to ambiguity.
- Prior to JDK 5, variable length arguments could be handled into two ways : One was using overloading, other was using array argument.
- There can be only one variable argument in a method.
- Variable argument (varargs) must be the last argument.

Using Arrays class

Arrays class in Java

There are often times when we need to do following tasks on an array in Java.

- Fill an array with a particular value. We usually do it with the help of a for loop.
- Sort an array.
- Binary search in sorted array.
- And many more..

Arrays class in Java

- The Arrays class of the java.util package contains several static methods that we can use to fill, sort, search, etc in arrays.
- This class is a member of the Java Collections Framework and is present in java.util.arrays.

Arrays class in Java

- **public static String toString(int[] a)** The string representation consists of a list of the array's elements, enclosed in square brackets ("[]"). Adjacent elements are separated by the characters a comma followed by a space. Elements are converted to strings as by String.valueOf(int). Returns "null" if a is null.

- **public static void sort(int[] a)** – Sorts the specified array into ascending numerical order.

- **public static void sort(int[] a, int fromIndex, int toIndex)** If we wish to sort a specified range of the array into ascending order. we can use this. The range to be sorted extends from the index fromIndex, inclusive, to the index toIndex, exclusive. If fromIndex == toIndex, the range to be sorted is empty.

- **public static int binarySearch(int[] a, int key)** Returns an int value for the index of the specified key in the specified array. Returns a negative number if the specified key is not found in the array. For this method to work properly, the array must first be sorted by the sort method.

Arrays class in Java

- **public static int[] copyOf(int[] original, int newLength)** Copies the specified array and length. It truncates the array if provided length is smaller and pads if provided .

- **public static int[] copyOfRange(int[] original, int from, int to)** Copies the specified range of the specified array into a new array. The initial index of the range (from) must lie between zero and original.length, inclusive.

- **public static void fill(int[] a, int val)** Fills all elements of the specified array with the specified value.

- **public static void fill(int[] a, int fromIndex, int toIndex, int val)** – Fills elements of the specified array with the specified value from the fromIndex element, but not including the toIndex element.

SYNERGETICS GET IT RIGHT | Microsoft Partner Gold Cloud Platform Silver Learning

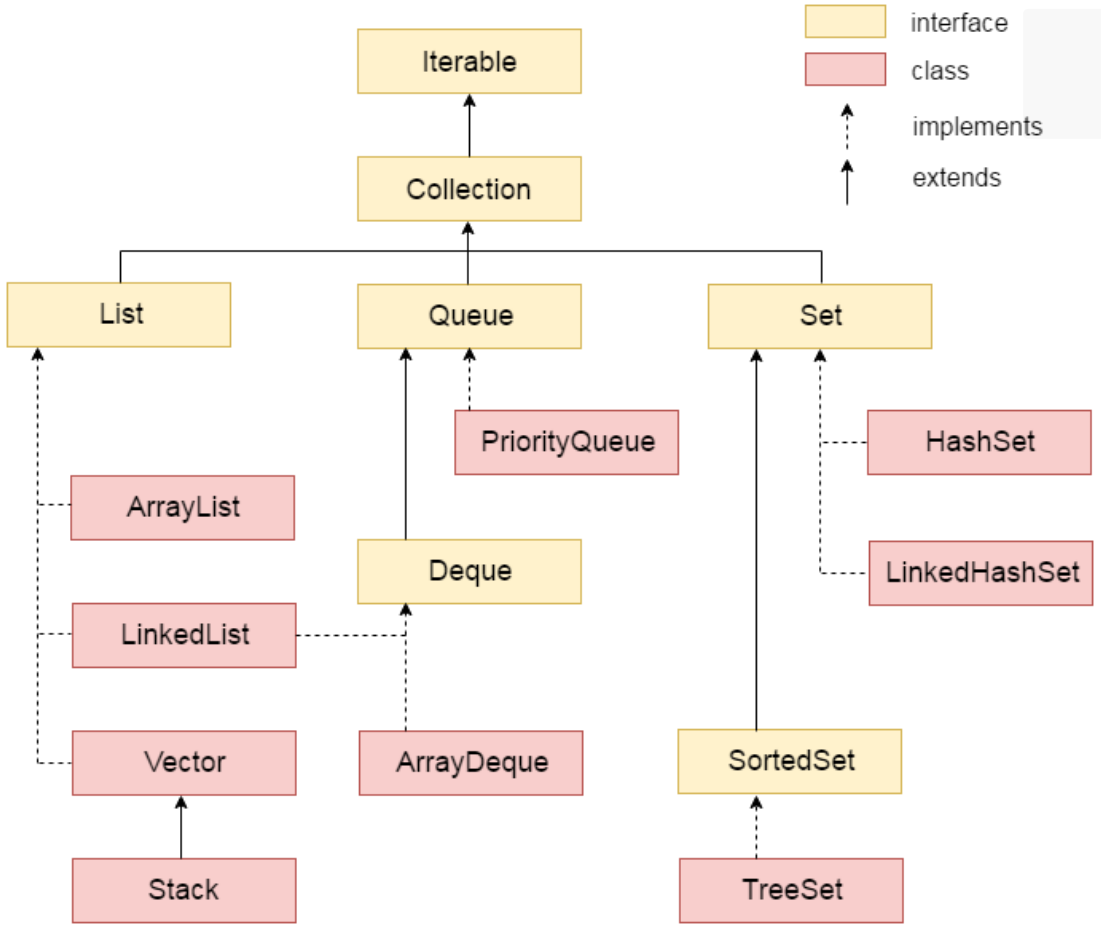
Collections Framework

Collections Framework

The Collection framework represents a unified architecture for storing and manipulating a group of objects. It has:

- Interfaces and its implementations, i.e., classes
- Algorithm

Collections Framework



Collection Interface

- The Collection interface is the foundation upon which the collections framework is built.
- It declares the core methods that all collections will have.
- These methods are summarized in the following table.

Collection Interface Methods

1	**boolean add(Object obj)** Adds obj to the invoking collection. Returns true if obj was added to the collection. Returns false if obj is already a member of the collection, or if the collection does not allow duplicates.
2	**boolean addAll(Collection c)** Adds all the elements of c to the invoking collection. Returns true if the operation succeeds (i.e., the elements were added). Otherwise, returns false.
3	**void clear( )** Removes all elements from the invoking collection.
4	**boolean contains(Object obj)** Returns true if obj is an element of the invoking collection. Otherwise, returns false.
5	**boolean containsAll(Collection c)** Returns true if the invoking collection contains all elements of **c**. Otherwise, returns false.
6	**boolean equals(Object obj)** Returns true if the invoking collection and obj are equal. Otherwise, returns false.

Collection Interface Methods

6	**boolean equals(Object obj)** Returns true if the invoking collection and obj are equal. Otherwise, returns false.
7	**int hashCode( )** Returns the hash code for the invoking collection.
8	**boolean isEmpty( )** Returns true if the invoking collection is empty. Otherwise, returns false.
9	**Iterator iterator( )** Returns an iterator for the invoking collection.
10	**boolean remove(Object obj)** Removes one instance of obj from the invoking collection. Returns true if the element was removed. Otherwise, returns false.
11	**boolean removeAll(Collection c)** Removes all elements of c from the invoking collection. Returns true if the collection changed (i.e., elements were removed). Otherwise, returns false.

Collection Interface Methods

12	**boolean retainAll(Collection c)** Removes all elements from the invoking collection except those in c. Returns true if the collection changed (i.e., elements were removed). Otherwise, returns false.
13	**int size( )** Returns the number of elements held in the invoking collection.
14	**Object[ ] toArray( )** Returns an array that contains all the elements stored in the invoking collection. The array elements are copies of the collection elements.
15	**Object[ ] toArray(Object array[ ])** Returns an array containing only those collection elements whose type matches that of array.

Implementation Classes

- Some of the implementation classes.

- ArrayList class
- LinkedList class
- List interface
- HashSet class
- LinkedHashSet class
- TreeSet class
- PriorityQueue class
- Map interface
- HashMap class
- LinkedHashMap class
- TreeMap class
- Hashtable class

Iterator interface

- Iterator interface provides the facility of iterating the elements in a forward direction only.
- Methods of Iterator interface
- There are only three methods in the Iterator interface. They are:

No	Method	Description
1	public boolean hasNext()	It returns true if the iterator has more elements otherwise it returns false.
2	public Object next()	It returns the element and moves the cursor pointer to the next element.
3	public void remove()	It removes the last elements returned by the iterator. It is less used.

Iterator interface

```
ArrayList al = new ArrayList();

Iterator itr = al.iterator();

        // checking the next element availabilty
        while (itr.hasNext())
        {
            //  moving cursor to next element
            int i = (Integer)itr.next();

            // getting even elements one by one
            System.out.print(i + " ");

            // Removing odd elements
            if (i % 2 != 0)
               itr.remove();
        }
```

Iterator interface

- remove() method can throw two exceptions
- UnsupportedOperationException : If the remove operation is not supported by this iterator
- IllegalStateException : If the next method has not yet been called, or the remove method has already been called after the last call to the next method

Java Comparable interface

- Java Comparable interface is used to order the objects of user-defined class.
- This interface is found in java.lang package and contains only one method named compareTo(Object).
- It provide single sorting sequence only i.e. you can sort the elements on based on single data member only. For example it may be rollno, name, age or anything else.

compareTo(Object obj) method

public int compareTo(Object obj): is used to compare the current object with the specified object.

We can sort the elements of:

- String objects
- Wrapper class objects
- User-defined class objects

Comparator Interface in Java

- Comparator interface is used to order the objects of user-defined classes.
- A comparator object is capable of comparing two objects of two different classes. Following function compare obj1 with obj2

public int compare(Object obj1, Object obj2):

- **Method 1:** One obvious approach is to write our own sort() function using one of the standard algorithms. This solution requires rewriting the whole sorting code for different criterion like Roll No. and Name.

- **Method 2:** Using comparator interface- Comparator interface is used to order the objects of user-defined class. This interface is present java.util package and contains 2 methods compare(Object obj1, Object obj2) and equals(Object element). Using comparator, we can sort the elements based on data members. For instance it may be on rollno, name, age or anything else.

Comparator Interface in Java

- Method of Collections class for sorting List elements is used to sort the elements of List by the given comparator.

// To sort a given list. ComparatorClass must implement

// Comparator interface.

public void sort(List list, ComparatorClass c)

Comparator Interface in Java

- **How does Collections.Sort() work?**
- Internally the Sort method does call Compare method of the classes it is sorting. To compare two elements, it asks “Which is greater?” Compare method returns -1, 0 or 1 to say if it is less than, equal, or greater to the other. It uses this result to then determine if they should be swapped for its sort.

```
class Sortbyroll implements Comparator<Student>
{
    // Used for sorting in ascending order of
    // roll number
    public int compare(Student a, Student b)
    {
        return a.rollno - b.rollno;
    }
}
```

Comparator Interface in Java

```
ArrayList<Student> ar = new ArrayList<Student>();
ar.add(new Student(111, "bbbb", "london"));
ar.add(new Student(131, "aaaa", "nyc"));
ar.add(new Student(121, "cccc", "jaipur"));

Collections.sort(ar, new Sortbyroll());
```

Generics in Java

Generics in Java

- Generics was added in Java 5 to provide compile-time type checking and removing risk of ClassCastException that was common while working with collection classes.
- The whole collection framework was re-written to use generics for type-safety.
- Let’s see how generics help us using collection classes safely.

Generics in Java

```
List list = new ArrayList();
list.add("abc");
list.add(new Integer(5)); //OK

for(Object obj : list){
        //type casting leading to ClassCastException at runtime
    String str=(String) obj;
}
```

Above code compiles fine but throws ClassCastException at runtime because we are trying to cast Object in the list to String whereas one of the element is of type Integer.

Generics in Java

- After Java 5, we use collection classes like below.

```
List<String> list1 = new ArrayList<String>(); // java 7 ? List<String> list1 = new ArrayList<>();
list1.add("abc");
//list1.add(new Integer(5)); //compiler error

for(String str : list1){
    //no type casting needed, avoids ClassCastException
}
```

Generics in Java

- We can define our own classes with generics type.
- A generic type is a class or interface that is parameterized over types.
- We use angle brackets (<>) to specify the type parameter.

```
public class GenericsType<T> {

        private T t;

        public T get(){
                return this.t;
        }

        public void set(T t1){
                this.t=t1;
        }
}
```

SYNERGETICS GET IT RIGHT | Microsoft Partner
Gold Cloud Platform
Silver Learning

Q & A

Contact: amitmahadik@synergetics-india.com

SYNERGETICS GET IT RIGHT | Microsoft Partner
Gold Cloud Platform
Silver Learning

Thank You