Arrays in JAVA

Dr Ilayaraja N

Introduction

- **“an array is a collection of similar type of elements which has contiguous memory location”**
- **Java array** is an object which contains elements of a similar data type.
- Additionally, The elements of an array are stored in a contiguous memory location. It is a data structure where we store similar elements.
- We can store only a fixed set of elements in a Java array.
- Array in Java is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on 1st index and so on.

Introduction

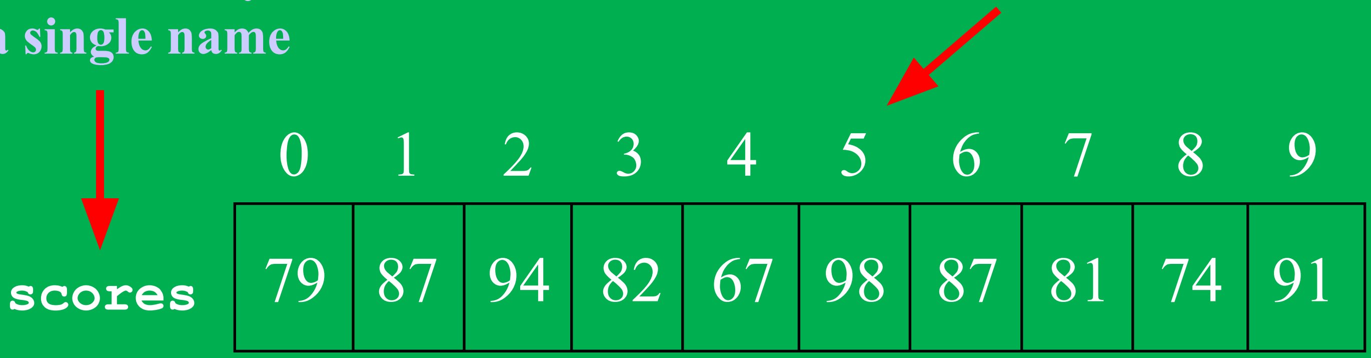
- Array: An ordered collection of values with two distinguishing characters:
  - Ordered and fixed length
  - Homogeneous. Every value in the array must be of the same type
- The individual values in an array are called **elements**.
- The number of elements is called the **length** of the array
- Each element is identified by its position number in the array, which is called **index**. In Java, the index numbers begin with 0.

Introduction

- An *array* is an ordered list of values

The entire array has a single name

Each value has a numeric *index*

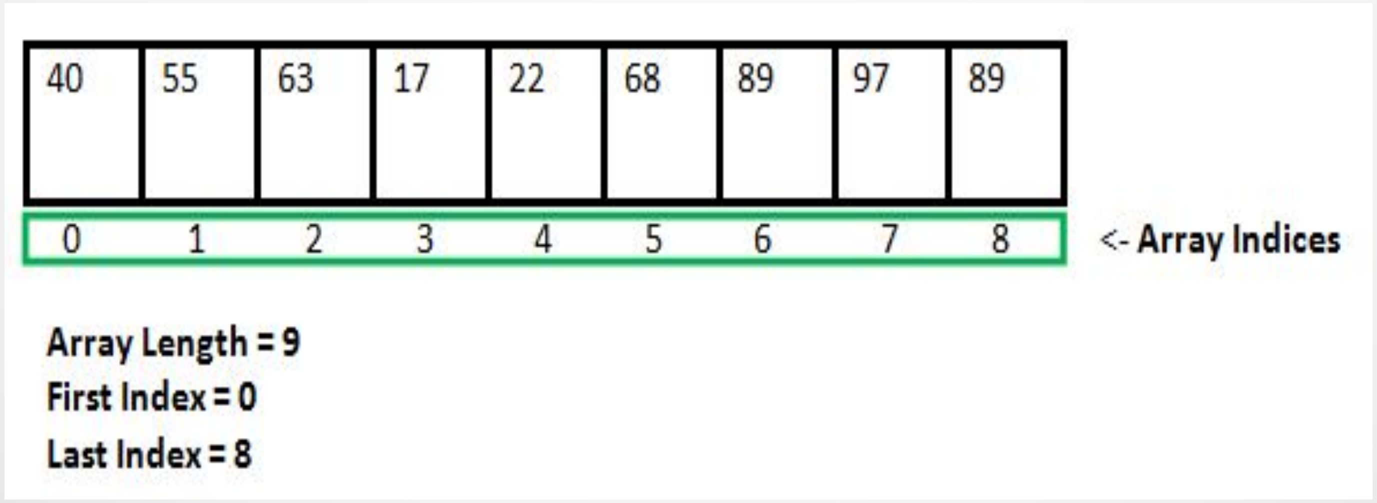


An array of size N is indexed from zero to N-1

This array holds 10 values that are indexed from 0 to 9

Introduction

int [ ] num = {40, 55, 63, 17, 22, 68, 89, 97, 89};



Arrays

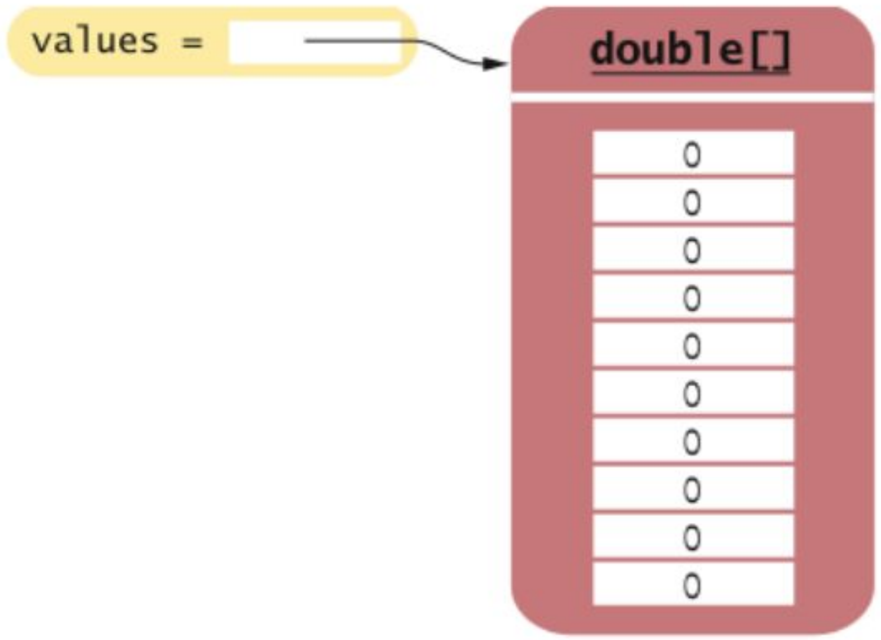


Figure 1
An Array Reference and an Array

Arrays

Use `[]` to access an element:

```
values[2] = 29.95;
```

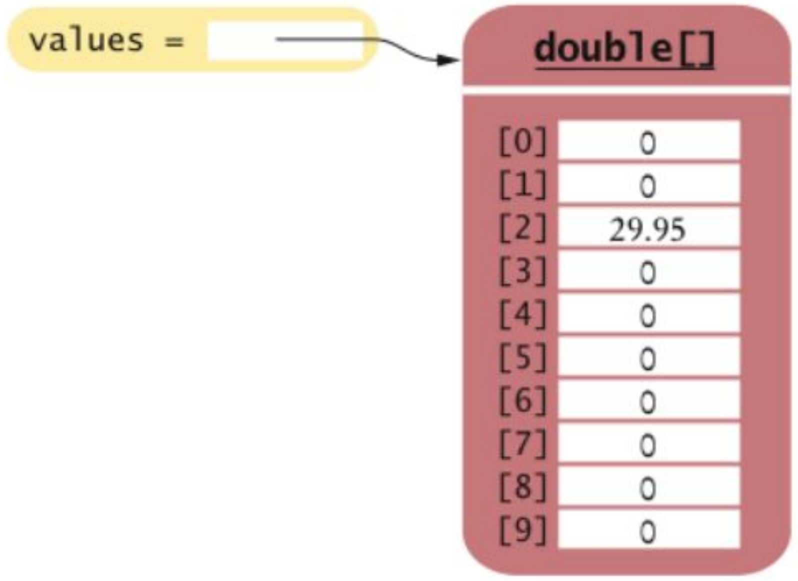
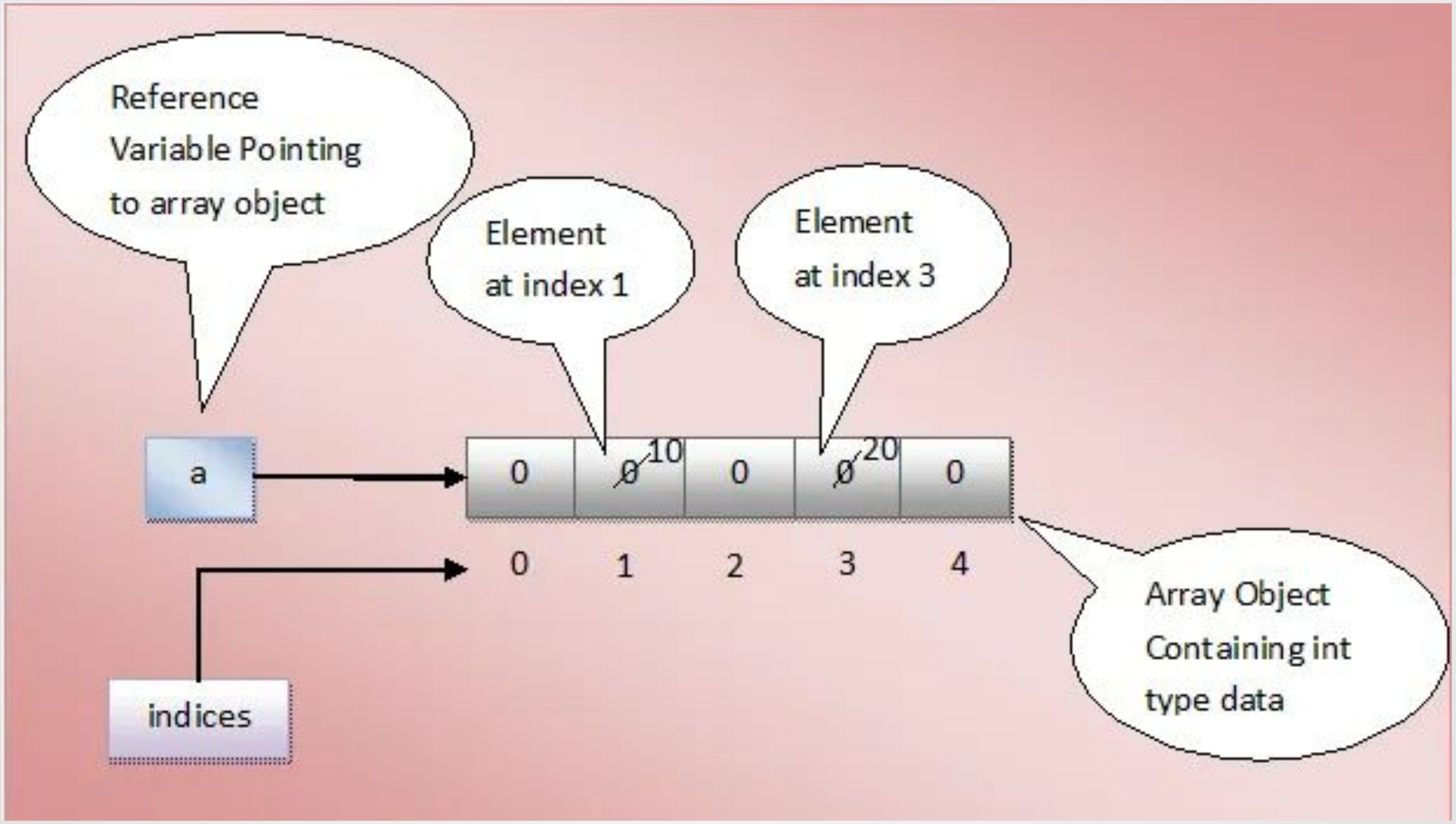


Figure 2
Modifying an Array Element

Introduction



Reference Variable Pointing to array object
Element at index 1
Element at index 3
a
0
~~0~~ 10
0
~~0~~ 20
0
0
1
2
3
4
indices
Array Object Containing int type data

Arrays

- Using the value stored:

```
System.out.println("The value of this data item is "
   + values[2]);
```

- Get array length as `values.length` (Not a method!)
- Index values range from `0` to `length - 1`
- Accessing a nonexistent element results in a **bounds error**:

```
double[] values = new double[10];
values[10] = 29.95; // ERROR
```

- Limitation: Arrays have fixed length

Introduction

- The values held in an array are called *array elements*
- An array stores multiple values of the same type (the *element type*)
- The element type can be a primitive type or an object reference
- Therefore, we can create an array of integers, or an array of characters, or an array of `String` objects, etc.
- **In Java, the array itself is an object**
- Therefore the name of the array is a object reference variable, and the array itself must be instantiated

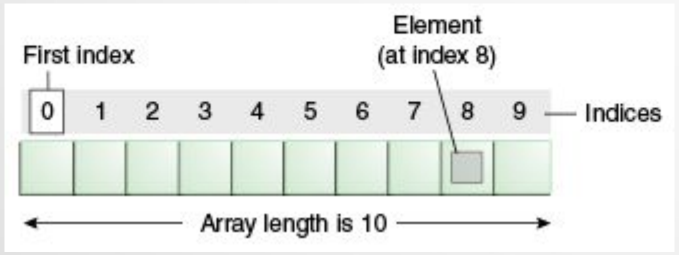
Introduction

Advantages

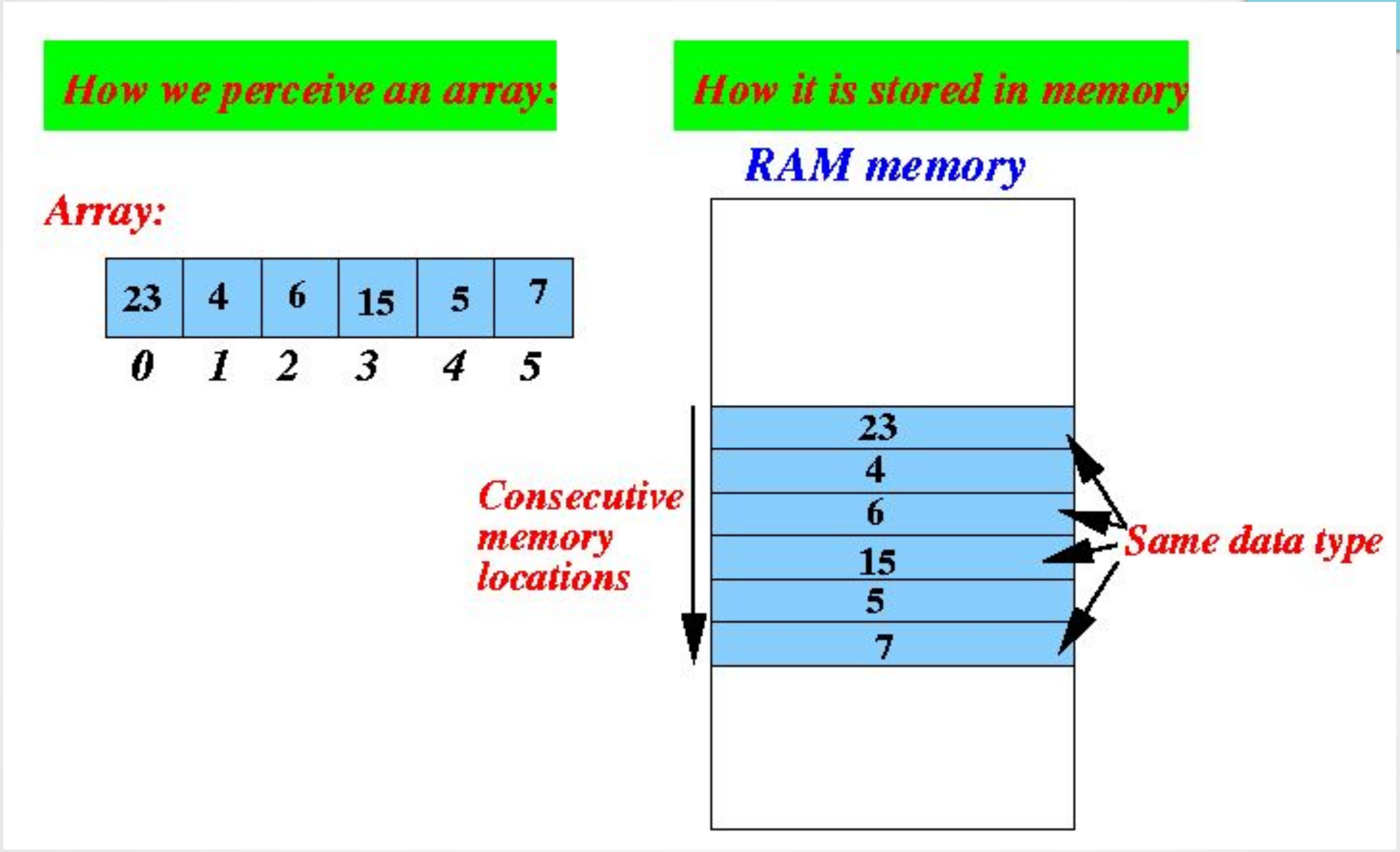
- **Code Optimization:** It makes the code optimized, we can retrieve or sort the data efficiently.
- **Random access:** We can get any data located at an index position.

Disadvantages

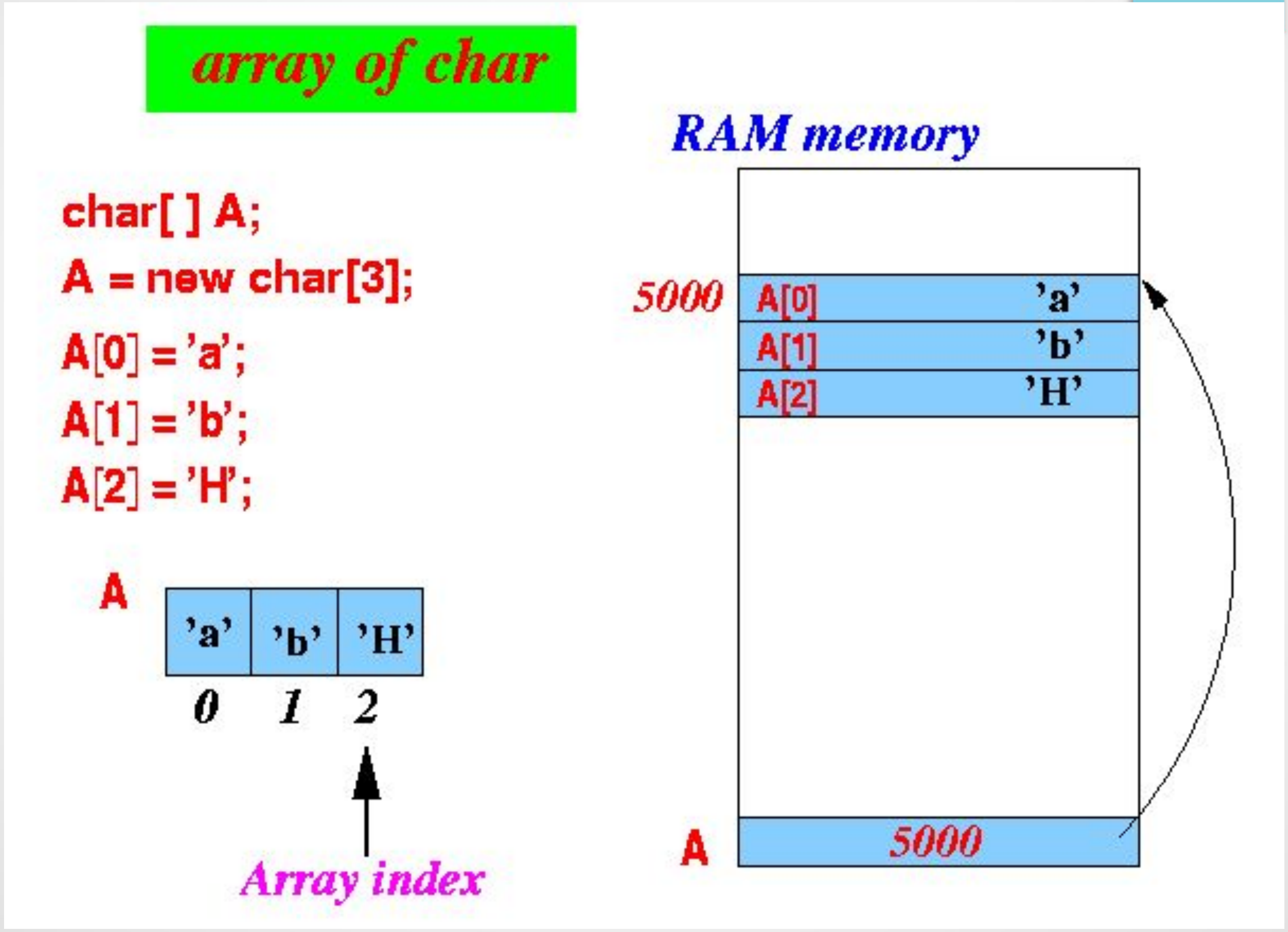
- **Size Limit:** We can store only the fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, collection framework is used in Java which grows automatically.



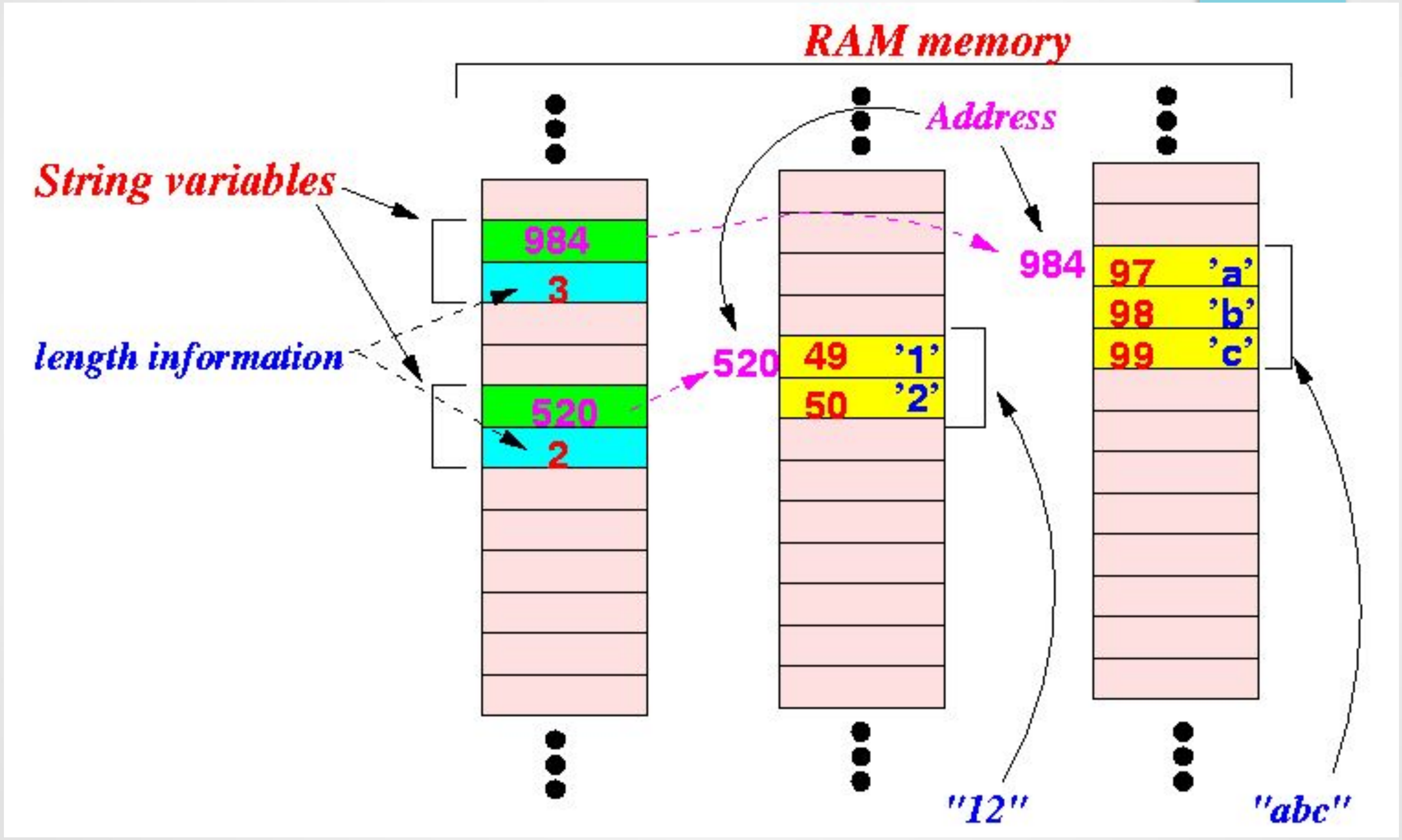
Memory Mapping


How we perceive an array:
How it is stored in memory
Array:
23
4
6
15
5
7
0
1
2
3
4
5
RAM memory
Consecutive memory locations
Same data type

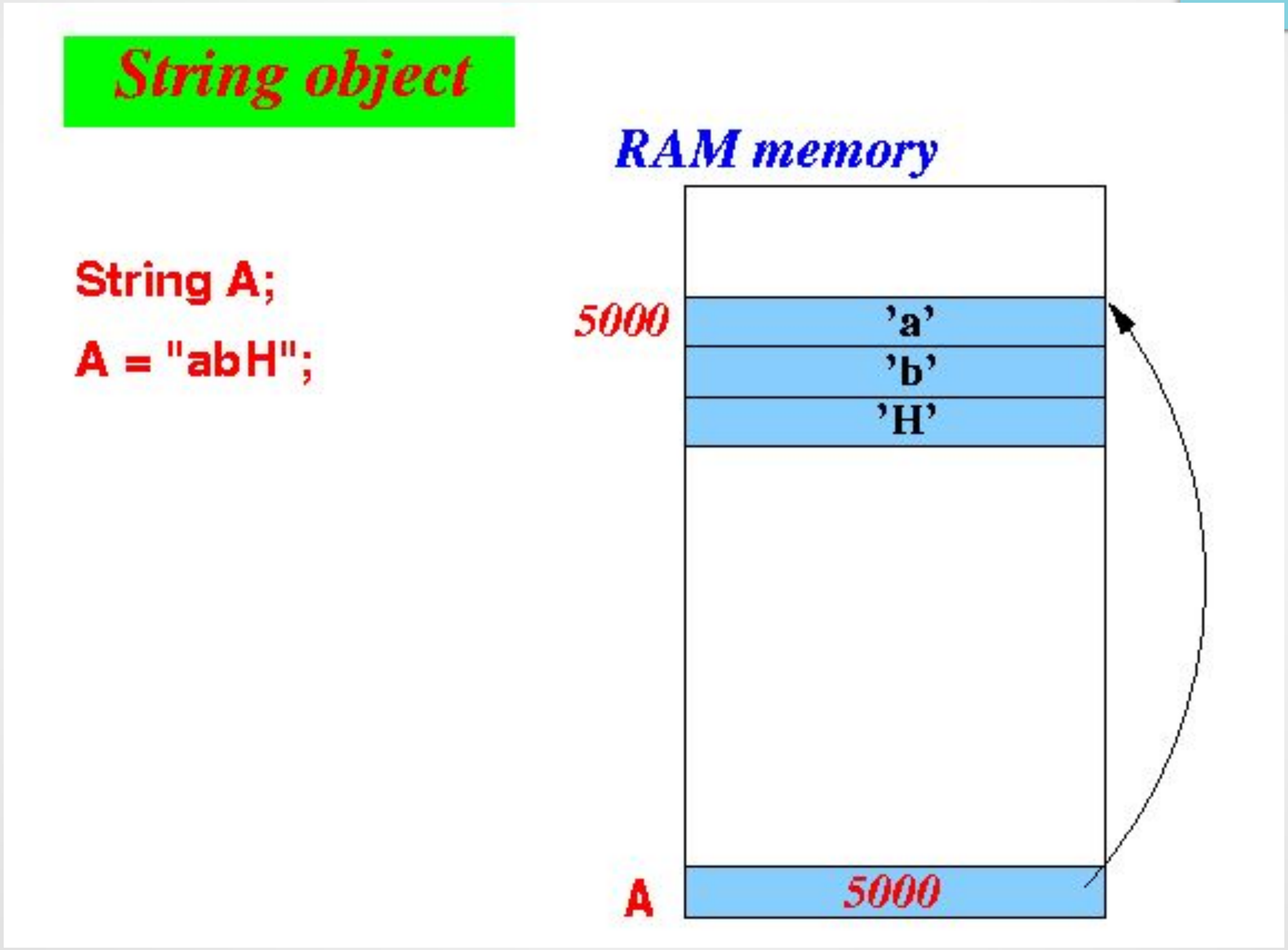
Memory Mapping



Memory Mapping



Memory Mapping



Declaring an Array Variable

- Array is collection of related data items
- Creating an array
  - Declare an array
  - Create memory location
  - Putting values to memory locations

Declaring an Array Variable

- Do not have to create an array while declaring array variable
  - *<type>* [ ] variable_name;
  - *double[ ] myList;*
  - *double myList[ ];*
- Both syntaxes are equivalent
- No memory allocation at this point

Alternate Array Syntax

- The brackets of the array type can be associated with the element type or with the name of the array
- Therefore the following declarations are equivalent:

```
float[] price;

float price[];
```

- The first format generally is more readable

Array declaration

An array is characterized by

- Element type
- Length

```
type[ ]  identifier  =  new type[length];
```

Default values in initialization

numerics	0
boolean	false
objects	null

Array declaration

- The elements in the array allocated by *new* will automatically be initialized to **zero** (for numeric types), **false** (for boolean), or **null** (for reference types).

- 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.
  - Thus, **in Java all arrays are dynamically allocated.**

Array Length Specified at Run-time

```
// array length specified at compile-time
int[] array1 = new int[10];



// array length specified at run-time
// calculate size…
int size = …;
int[] array2 = new int[size];
```

Creating Arrays

- General syntax for declaring an array:

```
Base_Type[] Array_Name = new Base_Type[Length];
```

- Examples:
  80-element array with base type `char`:
  `char[] symbol = new char[80];`

  100-element array of `double`s:
  `double[] reading = new double[100];`

  70-element array of `Species`:
  `Species[] specimen = new Species[70];`

Example:

- int intArray[ ];   //declaring array
- intArray = new int[20]; // allocating memory to array

- OR

- int[ ] intArray = new int[20]; // combining both statements in one

Programming Tip: Use Singular Array Names

- Using singular rather than plural names for arrays improves readability

- Although the array contains many elements the most common use of the name will be with a subscript, which references a *single* value.
- It is easier to read:
  - `score[3]` than
  - `scores[3]`

Declaring Arrays

- The `score` array could be declared as follows:

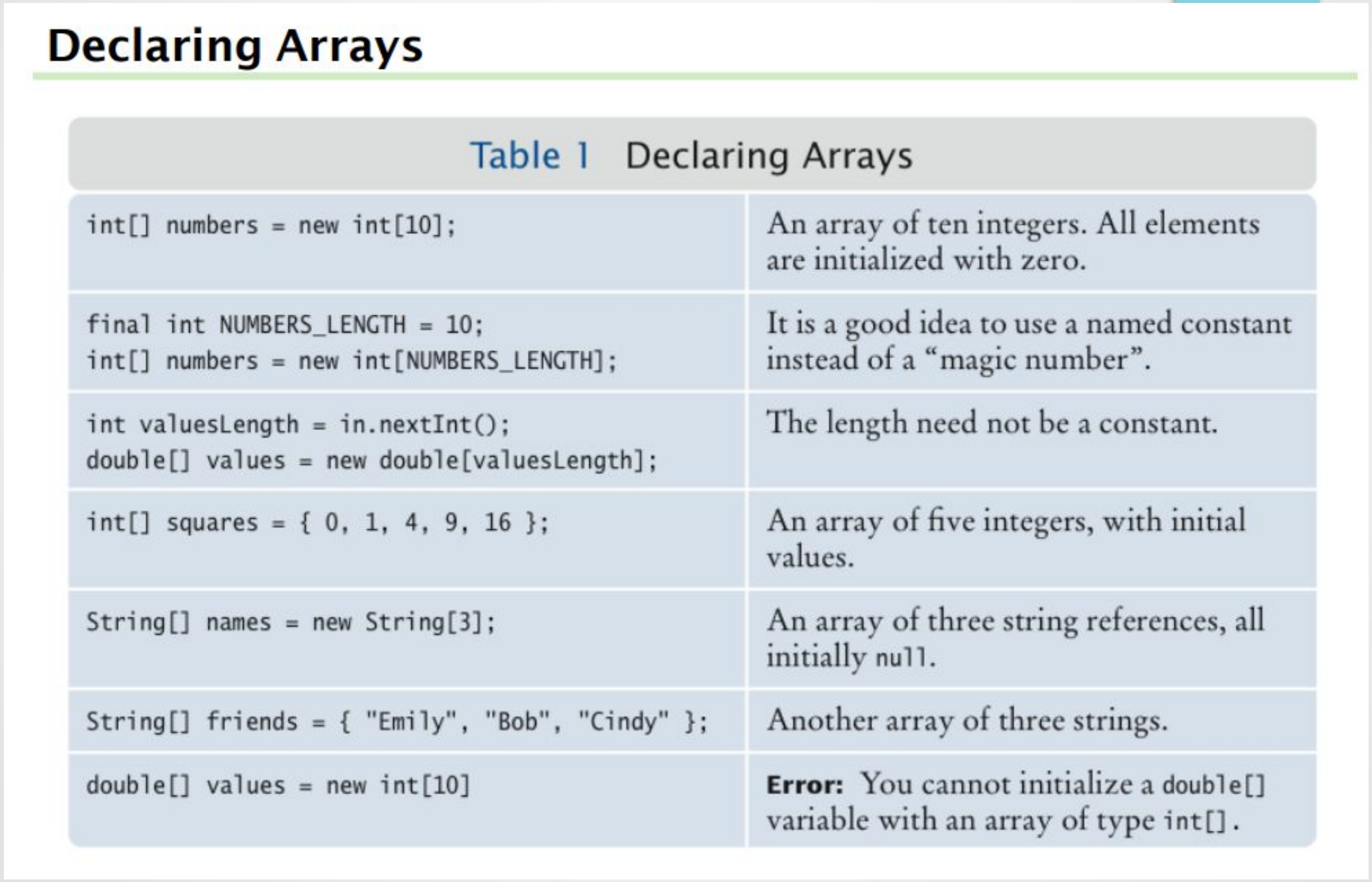
```
int[] score = new int[10];
```

- The type of the variable `score` is `int[]` (an array of integers)
- Note that the type of the array does not specify its size, but each object of that type has a specific size
- The reference variable `score` is set to a new array object that can hold 10 integers

Declaring Arrays

- Some examples of array declarations:

```
float[] price = new float[500];

boolean[] flag;

flag = new boolean[20];

char[] code = new char[1750];
```



Declaring Arrays

Table 1 Declaring Arrays

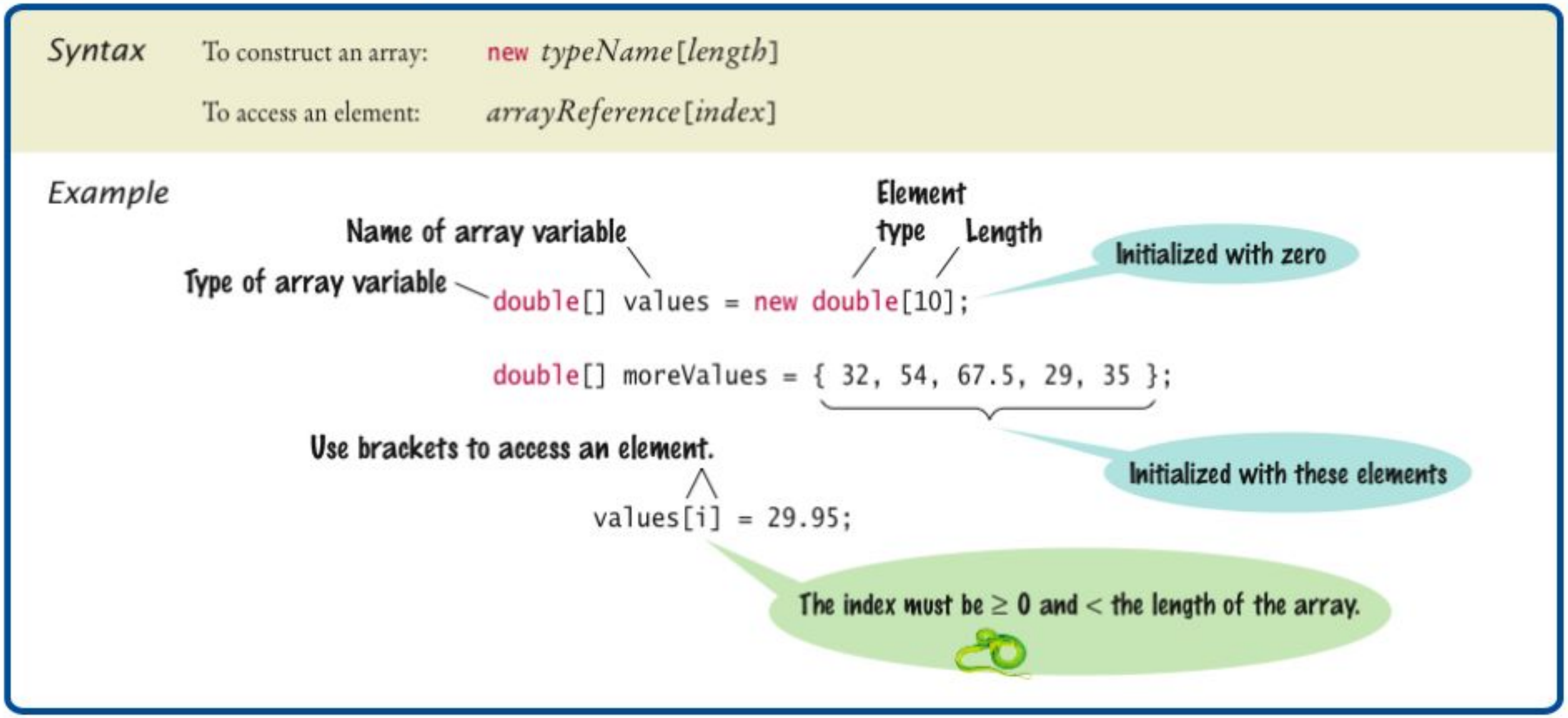
`int[] numbers = new int[10];`	An array of ten integers. All elements are initialized with zero.

`final int NUMBERS_LENGTH = 10;`
`int[] numbers = new int[NUMBERS_LENGTH];`
	It is a good idea to use a named constant instead of a "magic number".

`int valuesLength = in.nextInt();`
`double[] values = new double[valuesLength];`
	The length need not be a constant.

`int[] squares = { 0, 1, 4, 9, 16 };`	An array of five integers, with initial values.
`String[] names = new String[3];`	An array of three string references, all initially `null`.
`String[] friends = { "Emily", "Bob", "Cindy" };`	Another array of three strings.
`double[] values = new int[10]`	**Error:** You cannot initialize a `double[]` variable with an array of type `int[]`.

Syntax 7.1 Arrays



Syntax To construct an array: `new` *typeName*[*length*]

To access an element: *arrayReference*[*index*]

Example

```
double[] values = new double[10];

double[] moreValues = { 32, 54, 67.5, 29, 35 };

values[i] = 29.95;
```

Name of array variable
Type of array variable
Element type
Length
Initialized with zero
Initialized with these elements
Use brackets to access an element.
The index must be ≥ 0 and < the length of the array.

Three Ways to Use [ ] (Brackets) with an Array Name

1. Declaring an array: `int[] pressure`

   creates a name of type "`int` array"

   types `int` and `int[]` are different

   - `int[]`: type of the array
   - `int` : type of the individual values

2. To create a new array, e.g. `pressure = new int[100];`

3. To refer to a specific element in the array
   - also called *an indexed variable*, e.g.

```
pressure[3] = keyboard.nextInt();
System.out.println("You entered" + pressure[3]);
```

Some Array Terminology

```
temperature[n + 2]
```

Array name

```
temperature[n + 2]
```

Index - also called a *subscript*
- must be an `int`,
- or an expression that evaluates to an `int`

Indexed variable - also called an *element* or *subscripted variable*

```
temperature[n + 2]
```

Value of the indexed variable
- also called an element of the array

```
temperature[n + 2] = 32;
```

Note that "element" may refer to either a single indexed variable in the array or the *value* of a single indexed variable.

Subscript Range

- Array subscripts use zero-numbering
  - the first element has subscript 0
  - the second element has subscript 1
  - etc. - the nth element has subscript n-1
  - the last element has subscript `length-1`
- For example: an int array with 4 elements

Subscript:	0	1	2	3
Value:	97	86	92	71

Arrays

- A particular value in an array is referenced using the array name followed by the index in brackets
- For example, the expression

```
score[2]
```

  refers to the value `94` (the 3rd value in the array)
- That expression represents a place to store a single integer and can be used wherever an integer variable can be used

Arrays

- For example, an array element can be assigned a value, printed, or used in a calculation:

```
score[2] = 89;

score[first] = score[first] + 2;

mean = (score[0] + score[1])/2;

System.out.println ("Top = " + score[5]);
```

Defining an Array

Define an array as follows:

- variable_name=new <*type*> [arraySize];
- Number = new int[5];
- Mylist = new int[10];

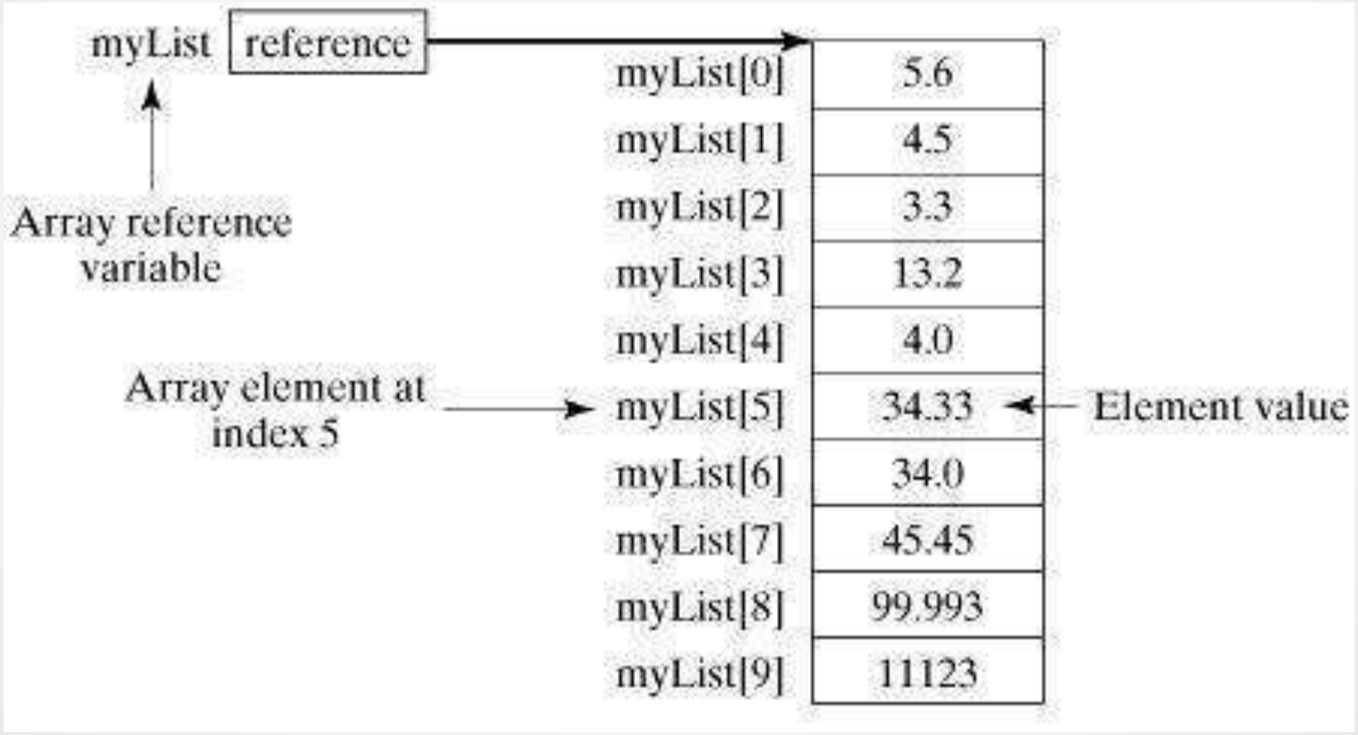
It creates an array using new dataType[arraySize];

- It assigns the reference of the newly created array to the variable variable_name.
- dataType arrayname[ ] = {list of values};
- Int a [ ]={1,2,3,4,5,6,7,};
- Array index starts from 0 to arraySize-1;
- int is of 4 bytes, total space=4*10=40 bytes

Declaring and defining in the same statement:

- *Double* [ ] mylist = new *double*[10];

Creating arrays cntd...



Selecting elements

Identifying an element

```
array[index]
```

- Index can be an expression
- Cycling through array elements

```
for (int i = 0; i < array.length; i++) {
    operations involving the ith element
}
```

- Accessing Java Array Elements using for Loop
- Each element in the array is accessed via its index. The index begins with 0 and ends at (total array size)-1. All the elements of array can be accessed using Java for Loop.
- // accessing the elements of the specified array

```
for (int i = 0; i < arr.length; i++)
   System.out.println("Element at index " + i + " : "+ arr[i]);
```

Defining length

- Use named constant to declare the length of an array.

```
private static final in N_JUDGES  =  5;
double[ ]  score  =  new double[N_JUDGES];
```

- Or read the length of an array from the user.

Bounds Checking

- Once an array is created, it has a fixed size
- An index used in an array reference must specify a valid element
- That is, the index value must be in bounds (0 to N-1)
- The Java interpreter throws an `ArrayIndexOutOfBoundsException` if an array index is out of bounds
- This is called *automatic bounds checking*
- The Java Virtual Machine (JVM) throws an ArrayIndexOutOfBoundsException if length of the array in negative, equal to the array size or greater than the array size while traversing the array.

Subscript out of Range Error

- Using a subscript larger than `length-1` causes a *run time* (not a compiler) error
  - an `ArrayOutOfBoundsException` is thrown
    - you do not need to catch it
    - you need to fix the problem and recompile your code
- Other programming languages, e.g. C and C++, do not even cause a run time error!
  - one of the most dangerous characteristics of these languages is that they allow out of bounds array indices.

Bounds Checking

- For example, if the array `codes` can hold 100 values, it can be indexed using only the numbers 0 to 99

- If `count` has the value 100, then the following reference will cause an exception to be thrown:

```
System.out.println (code[count]);
```

- It's common to introduce *off-by-one errors* when using arrays

problem

```
for (int index=0; index <= 100; index++)
code[index] = index*50 + epsilon;
```

Bounds Checking

- Each array object has a public constant called `length` that stores the size of the array
- It is referenced using the array name:

```
score.length
```

- Note that `length` holds the number of elements, not the largest index

Initializer Lists

- An *initializer list* can be used to instantiate and initialize an array in one step
- The values are delimited by braces and separated by commas
- Examples:

```
int[] unit = {147, 323, 89, 933, 540,
               269, 97, 114, 298, 476};

char[] letterGrade = {'A', 'B', 'C', 'D', ’F'};
```

Initializer Lists

- Note that when an initializer list is used:
  - the `new` operator is not used
  - no size value is specified
- The size of the array is determined by the number of items in the initializer list
- An initializer list can only be used only in the array declaration

Initializer Lists

- // Declaring array literal with new operator

  int[ ] intArray = new int[ ] { 1,2,3,4,5,6,7,8,9,10 };

-
- In a situation, where the size of the array and variables of array are already known, array literals can be used.
- The length of this array determines the length of the created array.
- There is no need to write the new int[] part in the latest versions of Java

A convenient way of initializing an array:

```
int[ ] digit = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};

private static final String[ ] US_CITIES_OVER_ONE_MILLION = {
    “New York”,
    “Los Angeles”,
    “Chicago”,
    “Huston”,
    “Philadelphia”,
    “Phoenix”,
    “San Diego”,
    “San Antonio”,
    “Dallas”,
}
```

Initializing an Array's Values - in Its Declaration

- can be initialized by putting a comma-separated list in braces
- Uninitialized elements will be assigned some default value, e.g. 0 for `int` arrays (**explicit initialization** is recommended)
- The length of an array is automatically determined when the values are explicitly initialized in the declaration
- For example:

```
double[] reading = {5.1, 3.02, 9.65};
System.out.println(reading.length);
```

- displays `3`, the length of the array `reading`

Initializing Array Elements in a Loop

- A `for` loop is commonly used to initialize array elements
- For example:

```
int i;//loop counter/array index
int[] a = new int[10];
for(i = 0; i < a.length; i++)
   a[i] = 0;
```

  - note that the loop counter/array index goes from `0` to `length - 1`
  - it counts through `length = 10` iterations/elements using the zero-numbering of the array index

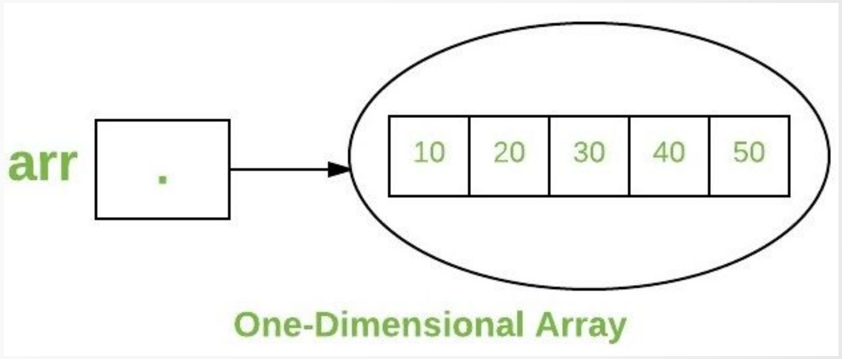
Programming Tip:

Do not count on default initial values for array elements

- explicitly initialize elements in the declaration or in a loop

```
class GFG
{
    public static void main (String[] args)
    {
      // declares an Array of integers.
      int[] arr;

      // allocating memory for 5 integers.
      arr = new int[5];

      // initialize the first elements of the array
      arr[0] = 10;

      // initialize the second elements of the array
      arr[1] = 20;

      //so on...
      arr[2] = 30;
      arr[3] = 40;
      arr[4] = 50;

      // accessing the elements of the specified array
      for (int i = 0; i < arr.length; i++)
         System.out.println("Element at index " + i +
                                       " : "+ arr[i]);
    }
}
```

```
Element at index 0 : 10
Element at index 1 : 20
Element at index 2 : 30
Element at index 3 : 40
Element at index 4 : 50
```



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

int a[]=new int[ ]{10,90,70,40,50};//declaration and instantiation

//a[0]=10;//initialization
//a[1]=20;
//a[2]=70;
//a[3]=40;
//a[4]=50;

//printing array
for(int i=0;i<a.length;i++)//length is the property of array
System.out.println(a[i]);
}
}
```

Anonymous Array in Java

Java supports the feature of an anonymous array, so you don't need to declare the array while passing an array to the method.

```
//Java Program to demonstrate the way of passing an anonymous array
//to method.
public class TestAnonymousArray{
//creating a method which receives an array as a parameter
static void printArray(int arr[]){
for(int i=0;i<arr.length;i++)
System.out.println(arr[i]);
}
public static void main(String args[]){
printArray(new int[]{10,22,44,66});//passing anonymous array to method
}}
```

Returning Array from the Method

```
//Java Program to return an array from the method
class TestReturnArray{
//creating method which returns an array
static int[] get(){
return new int[]{10,30,50,90,60};
}
public static void main(String args[]){
//calling method which returns an array
int arr[]=get();
//printing the values of an array
for(int i=0;i<arr.length;i++)
System.out.println(arr[i]);
}}
```

An array of objects

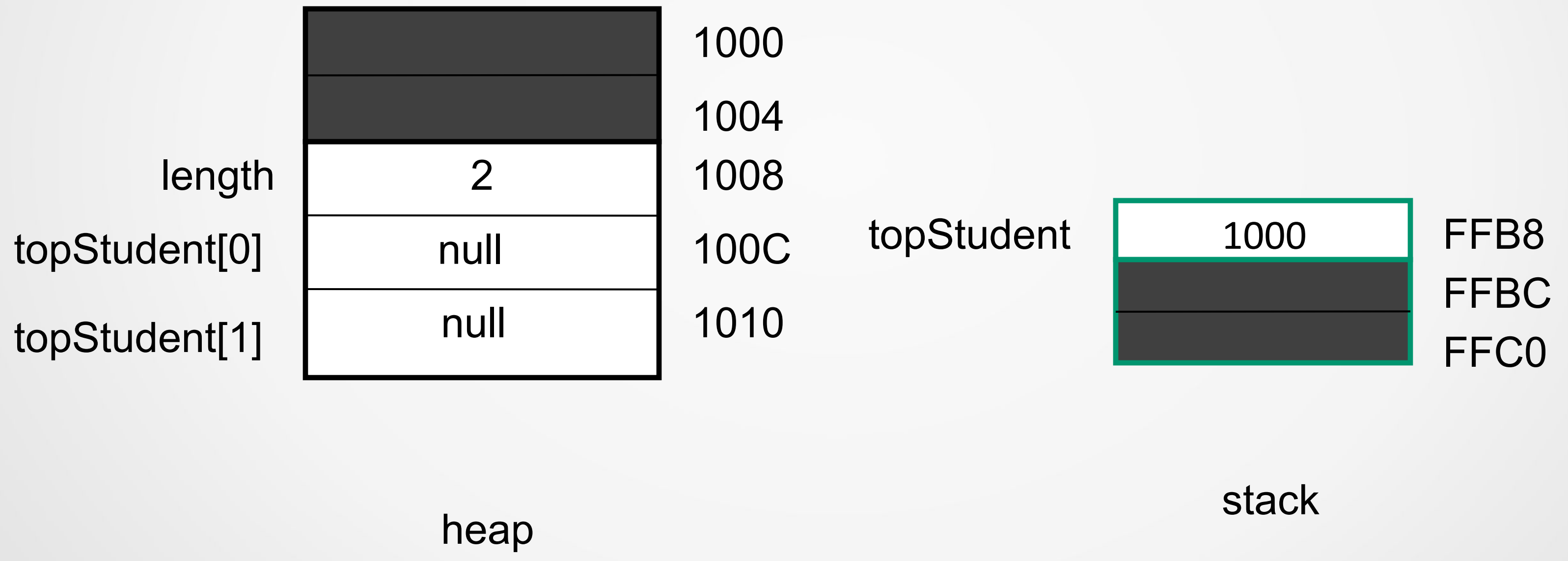
Elements of an array can be objects of any Java class.

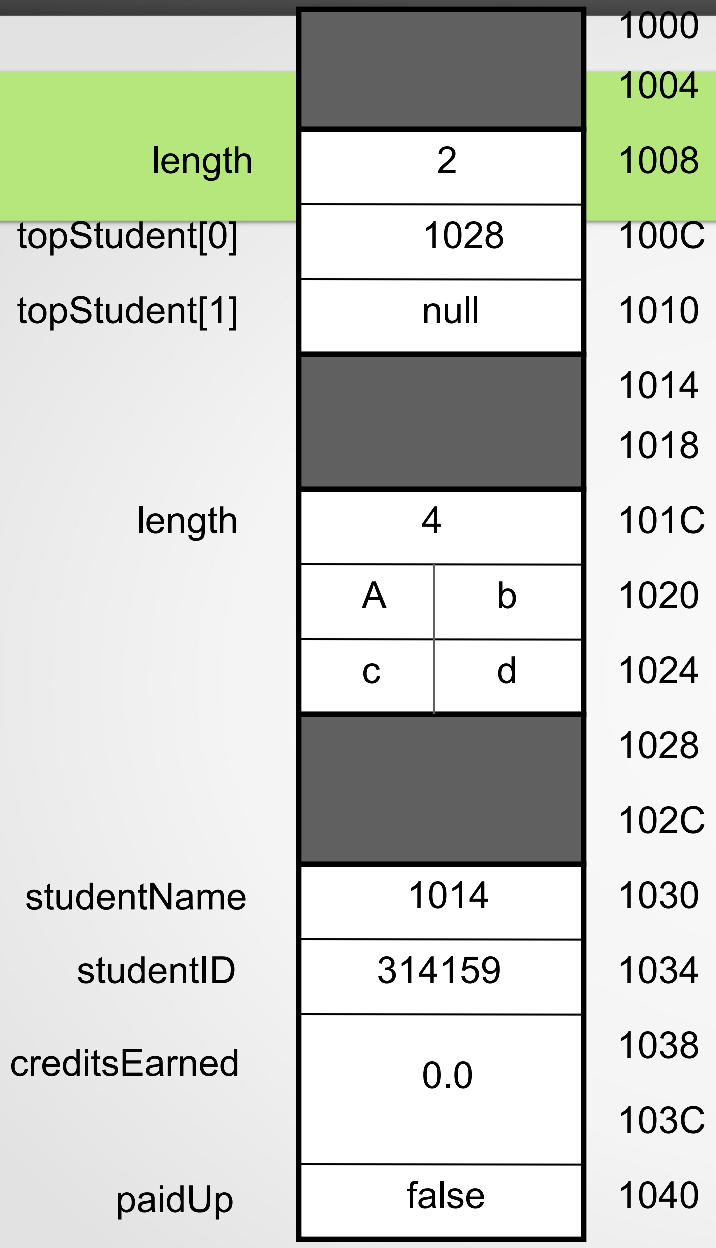
Example: An array of 5 instances of the student class

```
Student[ ] topStudent = new Student[5];
```

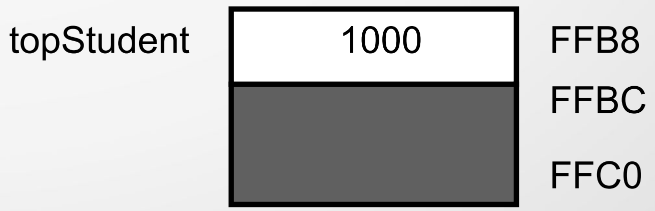
Internal representation of arrays

```
Student[ ]  topStudent  =  new Student[2];
topStudent[0] = new Student("Abcd", 314159);
```





```
Student[]  topStudent = new Student[2];
topStudent[0] = new Student("Abcd", 314159);
```



What happens if we define diffrent type…

- We define

  int[ ] a=new long[20];

  incompatible types

  found: long[]

  required: int[]

- The right hand side defines an array, and thus the array variable should refer to the same type of array

  The C++ style is not permitted in JAVA syntax

  Example:
  int prime[100];
  error=> ']' expected
  long primes[20];

Because you need to declare it as reference variable such as,

long[] primes = new long[20];

primes[25]=33;

*Runtime Error:*Exception in thread “main”
java.lang.ArrayIndexOutOfBoundsException

Array Size through Input

```
….
BufferedReader stdin = new BufferedReader (new InputStreamReader(System.in));
String inData;
int    num;
System.out.println("Enter a Size for Array:");
inData = stdin.readLine();
num    = Integer.parseInt( inData ); // convert inData to int
long[] primes = new long[num];
System.out.println(“Array Length=”+primes.length);
….
```

SAMPLE RUN:

```
Enter a Size for Array:
4
Array Length=4
```

Example for array

```
public class TestArray {
   public static void main(String[] args) {
      double[] myList = {1.9, 2.9, 3.4, 3.5};
      // Print all the array elements
      for (double element: myList) {
         System.out.println(element);
      }
   }
}
```

Otput:

```
1.9
2.9
3.4
3.5
```

Reusing Array Variables

- int[] primes=new int[10];

  ……

  primes=new int[50];
- Previous array will be discarded
- Cannot alter the type of array

Demonstration

```
long[] primes = new long[20];
primes[0] = 2;
primes[1] = 3;
System.out.println(primes[0]);
System.out.println(primes[1]);
```

Output:

```
2
3
```

Array Length

- Refer to array length using *length() method*
  - A data member of array object
  - array_variable_name.length
  - for(int k=0; k<primes.length;k++)
- Sample Code:

```
long[ ] primes = new long[20];
System.out.println(primes.length);
```

- Output: 20
- If number of elements in the array are changed, JAVA will automatically change the length attribute!

Sample Program

```
class MinArray
{
  public static void main ( String[] args )
  {
      int[] array = { 20, 19, 1, 5, 71, 27, 19, 95 } ;
      int min=array[0]; // initialize the current minimum
      for ( int index=0; index < array.length; index++ )
       if ( array[ index ] < min )
           min = array[ index ] ;
      System.out.println("The minimum of this array is: " +
      min );
  }
}
```

Two dimenssional array

- Representing 2D arrays
  - Int myarray[][];
  - Myarray = new int[3][4];
  - Int myarray [][] = new int[3][4];
- Example
- Int myarray[2][3]={0,0,0,1,1,1};

2 columns and 3 rows

- Multidimensional Arrays

- Multidimensional arrays are arrays of arrays with each element of the array holding the reference of other array. These are also known as Jagged Arrays. A multidimensional array is created by appending one set of square brackets ([]) per dimension. Examples:

- int[][] intArray = new int[10][20]; //a 2D array or matrix
- int[][][] intArray = new int[10][20][10]; //a 3D array

- int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };
- int x = myNumbers[1][2];
- System.out.println(x); // Outputs 7

Character Array in Java

- Character Array in Java is an Array that holds character data types values. In Java programming, unlike C, a character array is different from a string array, and neither a string nor a character array can be terminated by the NUL character.
- The Java language uses UTF-16 representation in a character array, string, and StringBuffer classes.
- The Character arrays are very advantageous in Java. They are very efficient and faster. Also, the data can be manipulated without any allocations.

- Java Strings are immutable means we can not change their internal state once they are created.
- However, char arrays allow us to manipulate after the creation. Even data structures List and Set are also acceptable.

What is a character in Java

- In Java, the characters are primitive data types. The char keyword is used to declare the character types of variables and methods. The default value of a char data type **'\u0000'**. The character values are enclosed with a single quote. Its default size is 2 bytes.
- The char data type can sore the following values:
  - Any alphabet
  - Numbers between 0 to 65,535 (Inclusive)
  - special characters (@, #, $, %, ^, &, *, (,), ¢, £, ¥)
  - 16-bit Unicode characters.

How to declare Character Arrays

char[ ] JavaCharArray;

How to Initialize Character Array

We can initialize the character array with an initial capacity. For example, to assign an instance with size 5, initialize it as follows:

char[] JavaCharArray = new char[5];

The values will be assign to this array as follows:

```
char[] JavaCharArray = new char[5];
JavaCharArray[0] = 'a';
JavaCharArray[1] = 'b';
JavaCharArray[2] = 'c';
JavaCharArray[3] = 'd';
JavaCharArray[4] = 'e';
```

Loops in Character Array

```
public class CharArrayDemo {
    public static void main(String[] args) {
        char[] JavaCharArray = {'a', 'b', 'c', 'd', 'e'};
        for (char c:JavaCharArray) {
        System.out.println(c);
        }
    }
}
```

Loops in Character Array

```
public class CharArrayDemo1 {
    public static void main(String[] args) {
        char[] JavaCharArray = {'a', 'b', 'c', 'd', 'e'};
        for (int i=0; i<JavaCharArray.length; i++) {
        System.out.println(JavaCharArray[i]);
        }
        }
}
```

Sorting a Character Array

```
import java.util.Arrays;
public class CharArrayDemo2 {
    public static void main(String[] args) {
        char[] JavaCharArray = {'e', 'b', 'c', 'a', 'd'};
        Arrays.sort(JavaCharArray);
        System.out.println(Arrays.toString(JavaCharArray));
        }
        }
```

Length of a Character Array

```
public class CharArrayDemo3 {
    public static void main(String[] args) {
        char[] JavaCharArray = {'a', 'b', 'c', 'd', 'e','f'};
        System.out.println(JavaCharArray.length);
        }
        }
```

How to Convert a String Array into Character Array

```
public class CharArrayDemo4 {
public static void main(String[] args) {
String value = "JavaTPoint"; //Enter String
//Convert string to a char array.
char[] array = value.toCharArray(); // Conversion to character from string
for(char c : array) //Iterating array values
{
System.out.println(c);
}
}
}
```

Equals() and Hashcode() in Java

- The equals() and hashcode() are the two important methods provided by the **Object** class for comparing objects.
- Since the Object class is the parent class for all Java objects, hence all objects inherit the default implementation of these two methods.
- How equals() and hashcode() methods, are related to each other, and how we can implement these two methods in Java.

Java equals()

- The java equals() is a method of *lang.Object* class, and it is used to compare two objects.
- To compare two objects that whether they are the same, it compares the values of both the object's attributes.
- By default, two objects will be the same only if stored in the same memory location.
    - » public boolean equals(Object obj)
- **Parameter:**
- **obj**: It takes the reference object as the parameter, with which we need to make the comparison.
- **Returns:**
- It returns the true if both the objects are the same, else returns false.

General Contract of equals() method

- There are some general principles defined by Java SE that must be followed while implementing the equals() method in Java. The equals() method must be:
- ***reflexive***: An object x must be equal to itself, which means, for object x, **equals(x)** should return true.
- ***symmetric***: for two given objects x and y, *x.equals(y)* must return true if and only if *equals(x)* returns true.
- ***transitive***: for any objects x, y, and z, if x.equals(y) returns true and y.equals(z) returns true, then x.equals(z) should return true.
- ***consistent***: for any objects x and y, the value of x.equals(y) should change, only if the property in equals() changes.
- For any object x, the *equals(null)* must return false.

Java hashcode()

- A **hashcode** is an integer value associated with every object in Java, facilitating the hashing in hash tables.
- To get this hashcode value for an object, we can use the hashcode() method in Java. It is the means ***hashcode() method that returns the integer hashcode value of the given object***.
- Since this method is defined in the Object class, hence it is inherited by user-defined classes also.
- The hashcode() method returns the same hash value when called on two objects, which are equal according to the equals() method. And if the objects are unequal, it usually returns different hash values.

Java hashcode()

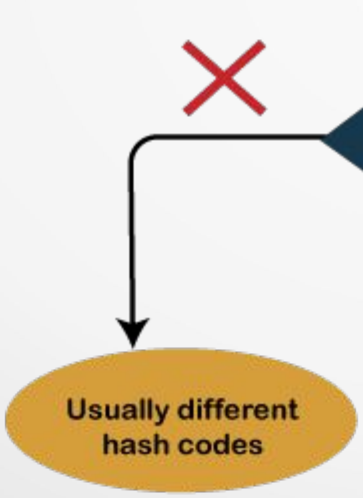
- public int hashCode()

- **Returns:**
- It returns the hash code value for the given objects.

Java hashcode()

Contract for hashcode() method in Java

- If two objects are the same as per the equals(Object) method, then if we call the hashCode() method on each of the two objects, it must provide the same integer result.



```
class Test_hash_equal{
    public static void main(String[] args){
        String a = "Andrew";
        String b = "Andrew";

        if(a.equals(b)){   //checking the equality of objects using equals() methods
            System.out.println("a & b are equal variables, and their respective hashvalues are:" + " "+ a.hash
Code() + " & " + b.hashCode());

        }

        String c = "Maria";
        String d= "Julie";

        if(!c.equals(d)){    //checking  the equality of objects using equals() method
            System.out.println("\nc & d are Un-equal variables, and their respective hashvalues are:" + " "+ c.
hashCode() + " & " + d.hashCode());

        }
    }
}
```

Output:

```
a & b are equal variables, and their respective hash values are: 1965574029 & 1965574029
c & d are Un-equal variables, and their respective hash values are: 74113750 & 71933245
```