More Datatypes and Operator

5.1 Arrays

- Arrays are collections of similar type of elements that have contiguous memory location.
- Java Array is an object that contains elements of similar data type.
- It is a data structure where we store similar elements. We can store only fixed set of elements in java array
- Array in java is index based; first element of the array is stored at 0 index.

5.1.1 Advantage of Java Array

- Code optimization: It makes code optimized, we can retrieve or sort the data easily.
- Random access: -We can get any data located at any index.

5.1.2 Disadvantage of Java Array

- Size Limit: we can store fixed size of elements in the array. It does't grow its size at runtime. To solve the problem, collection framework is used in java.

5.1.3 Types of Array

- Single dimensional or one dimensional
- Multidimensional

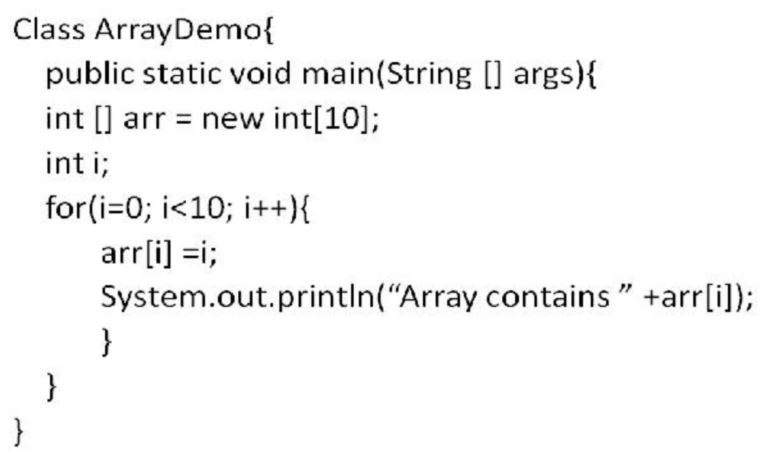
Single dimensional Array

- A One Dimensional Array is a list of related variables. Such lists are common in programming .
- General form:
  - **type[] array-name = new type[size];**

    here, type declares the element type
- The element type determines the data type of each element contained in the array.
- The number of elements that the array will hold is determined by ***size***.
- Since arrays are implemented as objects, the creation of array is two-step process
  - First declare an array reference variable.
  - Second allocate memory for the array using *new* operator.

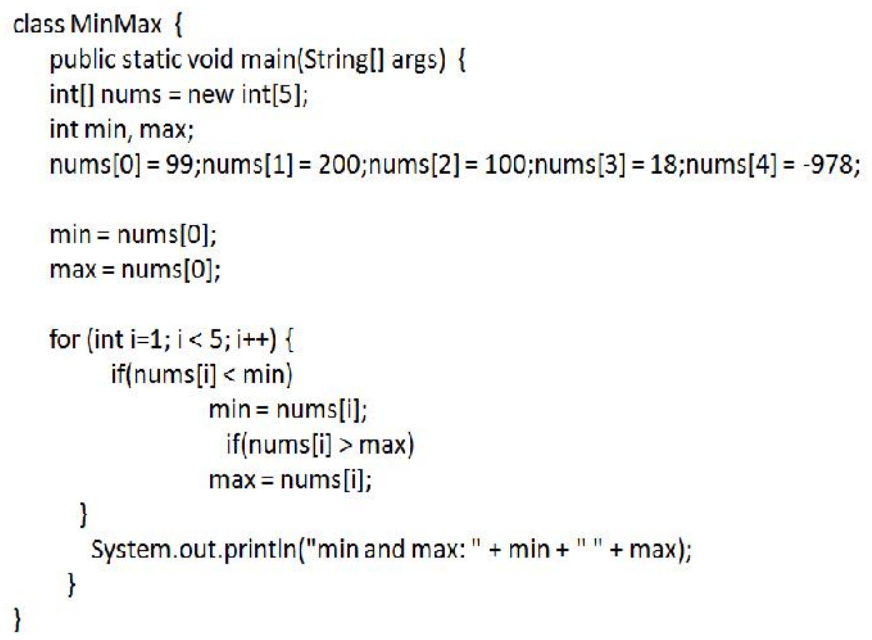
  Example:
  1. **int [ ] array_name;** //creates array reference
  2. **array_name = new int [10];** //allocate memory for 10 integer element

Example 1:



```
Class ArrayDemo{
   public static void main(String [] args){
   int [] arr = new int[10];
   int i;
   for(i=0; i<10; i++){
        arr[i] =i;
        System.out.println("Array contains " +arr[i]);
        }
   }
}
```

Example2: Finding max value



```
class MinMax {
    public static void main(String[] args) {
    int[] nums = new int[5];
    int min, max;
    nums[0] = 99;nums[1] = 200;nums[2] = 100;nums[3] = 18;nums[4] = -978;

    min = nums[0];
    max = nums[0];

    for (int i=1; i < 5; i++) {
           if(nums[i] < min)
                      min = nums[i];
                        if(nums[i] > max)
                      max = nums[i];
       }
       System.out.println("min and max: " + min + " " + max);
      }
}
```

5.1.4 Initialization

- The general form for initializing a one-dimensional array is
    - **type[] arrayName = {** ***val1, val2, val3 ..... valn*** **} ;**
- Here, the initial value are specified by val1 through valN
- They are assigned in sequence, left to right, in index order.
- Java automatically allocates an array large enough to hold the initializers that you specify.
- There is no need to explicitly use the new operator.

Example

Initializing an array

```
class MinMax2     {
        public static void main(String[] args)  {
                int[] nums = { 99,5623, 463, 287, 49 };
                int min, max;

                min = nums[0];
                max = nums[0];

                for ( int i = 1;  i < 5; i++)  {
                        if(nums[i] < min)
                                min = nums[i];
                        if(nums[i] > max)
                                max = nums[i];
              System.out.println ("Min and max: " + min + " " + max);
        }
    }
 }
```

5.2 Multidimensional Arrays

- In java, multidimensional array or two dimensional arrays are array of arrays.
- A two-dimensional array can be thought of as creating a table of data, with the data organized by row and column.
- An individual item of data is accessed by specifying its row and column position.
- To declare a two-dimensional array, you must specify the size of both dimensions.
- Example:
    - **int [][] table=new int[10][20];**
- here, table is used to declare two dimensional array of int with size 10 and 20

Example:

```
class ArrayTwoD  {
        public static void main(String[] args)  {
                int [][] table = new int [3][4];

                for(int i=0; i < 3; i++){
                        for(int j =0; j< 4; j++){
                                table[i][j]=j;
                        System.out.println(table[i][j]);}
                }
        }
}
```

5.2.1 Initializing multidimensional array

- A multidimensional array can be initialized by enclosing each dimension's initializer list within its own set of braces.
- The general form:

```
type[][] array-name = {
                  { val, val, val…..val },
                  { val, val, val…..val },
                          ….
                          …..
                   { val, val, val ….. val}
     };
```

Example:

```
class ArrayTwoD  {
        public static void main(String[] args)  {
                int [ ][ ] table = {{1, 2, 3},  {4,5,6},  {7, 8,9}};
                for(int i=0; i <= table.length; i++){
                        for(int j =0; j < table.length; j++){
                                int values=table[i][j];
                        System.out.println(values);
                        }
                }
        }
}
```

5.3 Alternative Array Declaration Syntax

- There is a second form that can be used to declare an array.
    - *type var-name*[ ];
- Here, the square brackets follow the name of the array variable, not the type specifier. For example two declaration are equivalent;
    - int counter [ ] = new int[3];
    - int [ ] counter = new int[3];
- The following declarations are also equivalent:
    - char table [ ] =new char[3] [4];
    - char [ ] table = new char [3] [4]
- This alternative declaration lets you declare both the array and nonarray variables of the same type in an single declaration, for example;
    - int alpha, beta[ ], gamma;
    - Here, alpha, gamma are of type int, but beta ia an array of int.

5.4 Assigning Array References

- As with other objects, when you assign one array reference variable to another, you are simply changing what object that variable refers to.
- Example:

```
int a [ ] = {1, 3, 8};   //create an array b [] of same size as a []

int b [ ] = new int[a.length];      //Does not copy elements of a[ ] to b [ ], only
makes  b refers to same location

b =a;
```

- When we do "b = a", we actually assigning reference of array. Hence if we make any change to one array, it would be reflected in other array as well because both a and b refer to same location.

Example

```
class Test
{
    public static void main(String[] args)
    {
        int a[ ] = {1, 8, 3};

        int b[ ] = new int[a.length];
        b = a;

        b[0]++;

        System.out.println("Contents of a[] ");
        for (int i=0; i<a.length; i++)
            System.out.print(a[i] + " ");

        System.out.println("\n\nContents of b[] ");
        for (int i=0; i<b.length; i++)
            System.out.print(b[i] + " ");
    }
}
```

Output:

```
Contents of a []
2 8 3
Contents of b []
2 8 3
```

5.5 Using the Length Member

- Arrays are implemented as objects, each array has associated with it a length instance variable that contains the number of elements that the array can hold.
- In other words, length contains **the size of the array.**

- Example

```
class JavaArrayLengthTest
{
            public static void main(String[] args) {
                        String[] testArray = { "Apple", "Banana", "Carrots" };
                        int arrayLength = testArray.length;
                        System.out.println("The length of the array is: " + arrayLength);
            }
 }
Output:
The length of the array is: 3
```

5.6 The For-Each Style for Loop

- A for-each loop cycles through a collection of objects, such as an array, in strictly sequential fashion from start to finish.
- The general form of the **for-each** style is
    - ***for(type itr-var:collection)***
      statement-block
- Here, ***type*** specifies the type.
- ***itr-val*** specifies the name of an iteration variable that will receive the elements from a collection, one at a time, from beginning to end.
- The collection being cycled through is specified by **collection.**

```
class Big {
          public static void main (String[ ] arg)
                {
                    int  [] myArr = {  45, 5, 34, 8 } ;
                    int  big = myArr[0];
                      for( int num : myArr)
                             if (  big<num )
                                      big = num;
                    System.out.println("Bigest =  " + big) ;
                }
}
```

```
class SumofArray{
      public static void main(String[] args){
      int [ ] arr ={10,20,30,40,50,10};
      int sum =0;
      for( int num : arr){
      sum=sum+num;
      }
      System.out.println(“Sum of array element is”+sum);
      }

}
```

5.6.1 Applying the enhanced for

- Since the for-each can only cycle through an array sequentially, from start to finish.
- One of the most common is searching.

```
class Search {
        public static void main(String[] args)
          {
                int[ ] nums = { 6, 8, 3, 7, 5, 6, 1, 4 };
                int val = 5;
                boolean found = false;

                for(int x : nums)
                {
                        if(x == val)
                        {
                                found = true;
                                break;
                        }
                }
                if(found)
                System.out.println("Value found!");
        }
}
```

5.7 Strings

- One of the most important of Java’s data type is String.
- String defines and supports character strings.
- In many other programming languages a string is an array of characters. But in java, strings are objects.
- **Constructing Strings:**

- You can construct a String just like you construct any other type of object: by using new and calling the String constructor.
- **Ex:**

  String str=new String("hello");
- This creates a **String** object called **str** that contains the character string **"hello".**
- You can also construct a String from another String.
  - **String str=new String("rnsit");**
  - **String str2=new String("MCA");**
- Another easy way to create a **String is:**
  - **String str="Java strings are powerful";**

5.7.1 Array of Strings

- Once you have created a String object, you can use it anywhere that quoted string is allowed.

```
class StringArrays {
        public static void main(String[] args)
           {
               String[] strs = { "This", "is", "a", "test." };
                 for(String s : strs)

                 System.out.print(s + " ");
                System.out.println("\n");

               strs[1] = "was";
               strs[3] = "test, too!";

               System.out.println("Modified array: ");
                   for(String s : strs)
                     System.out.print(s + " ");
           }
      }
```

5.7.2 Strings are immutable

- The contents of a **String** object are immutable. that is, once created, the character sequence that makes up the **string cannot be altered.**
- This restriction allows java to implement strings more efficiently.
- When you need a string that is a variation on one that already exists, simply create a **new string** that contains the desired changes.
- **It is just that the contents of a specific String object cannot be changed after it is created.**

- The **substring()** method returns a new string that contains a specified portion of the invoking string.
- The form of substring() is
    - **String substring(int startIndex,int endIndex);**
- Here, **startIndex** specifies the beginning index and
- **endIndex** specifies the stopping point.
- The string returned contains all the characters from the beginning index to the ending index.

5.7.3 Using a string to control a switch statement

- One good use for a String-controlled switch is when some action must be taken based on a command given in string form.

```
class StringSwitch
     {
      public static void main(String[] args)
      {
            String command = "cancel";
            switch(command)
               {
                  case "connect":
                  System.out.println("Connecting");
                  break;

                  case "cancel":
                   System.out.println("Canceling");
                   break;

                  case "disconnect":
                   System.out.println("Disconnecting");
                   break;

                  default: System.out.println("Command Error!");
                   break;
               } } }
```

5.8 The Bitwise operators.

- Java defines several bitwise operator which can applied to integer types, long, int, short, char and byte.
- Bitwise operator works on bits and perform bit-by-bit operation
- Assume integer variable A holds 60 and variable B holds 13 then

Operator	Description	Example
& (bitwise and)	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12 which is 0000 1100
| (bitwise or)	Binary OR Operator copies a bit if it exists in either operand.	(A | B) will give 61 which is 0011 1101
^ (bitwise XOR)	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49 which is 0011 0001
~ (bitwise compliment)	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -61 which is 1100 0011 in 2's complement form due to a signed binary number.
<< (left shift)	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000
>> (right shift)	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 1111
>>> (zero fill right shift)	Shift right zero fill operator. The left operands value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros.	A >>>2 will give 15 which is 0000 1111

Example: Getting uppercase for a letter

- The following program demonstrates **&** by turning any lowercase letters into uppercase by resetting the bit to 0.
- As the UNICODE/ASCII character set is defined, the lowercase letters are the same as the uppercase ones except that the lowercase ones are greater in value by exactly 32.

- Therefore, to transform a lowercase letter to uppercase, just turn off the 6th bit, as this program illustrates.
- The value 65,503 used in the AND statement is the decimal representation of 1111 1111 1101 1111. Thus, the AND operation leaves all bits in ch unchanged except for the 6th one, which is set to 0

```
class UpCase {
 public static void main(String args[]) {
   char ch;
         for(int i=0; i < 10; i++)
         {
                  ch = (char) ('a' + i);
                  System.out.print(ch);
                  ch = (char) ((int) ch & 65503);
                  System.out.print(ch + " ");
          }
  }
Output:
aA bB cC dD eE fF
```

- **Iterating through bits**

```
class DisplayBits{
            public static void main(String[ ] args){
                        int t;
                        byte val;
                        val = 123;

                        for(t=128; t < 0; t = t/2) {
                                    if((val & t )! =0)
                                                System.out.println("1 ");
                                    else
                                                 System.out.println("0 ");

                                    }

                        }

}
```

- The for loop successively tests each bit in **val**, using bitwise AND, to determine whether it is on or off.
- If the bit is on, the digit 1 is displayed; otherwise, 0 is displayed
- Bitwise OR, to change the uppercase to lowercase.

```
class LowCase {
            public static void main(String args[]) {
             char ch;
            for(int i = 0; i < 10; i++) {
                        ch = (char) ('A' + i);
                         System.out.print(ch);
                        ch = (char) ((int) ch | 32);
                        System.out.print(ch + " ");
            }
       }
}
Output:
Aa Bb Cc Dd Ee Ff
```

5.8.1 Shift Operators

- In java it is possible to shift the bits that make up a value to left to right by a specified amount.
- Java defines the three bit-shift operator shown here:

<<	Left shift
>>	Right shift
>>>	Unsigned right shift

- The general forms for these operator.

value << num-bits
value >> num-bits
value >>> num-bits

- Here, value is the value being shifted by the number of bit position specified by *num-bits.*
- *Example*

```
    class   ShiftOperators
    {
             public  static  void  main  (String  args[ ] )
             {
                       int  x  =  7 ;
                       System.out.println ("x  =  "  +  x) ;
                       System.out.println ("x  >>  2  =  "  +   (x  >>  2) ) ;
                       System.out.println ("x  <<  1  =  "  +  (x  <<  1) ) ;
                      System.out.println ("x  >>>  1  =  "  +  (x  >>>  1) ) ;
                }
    }
output:
x >> 2 = 1
x<< 1 = 14
x >>> 1 = 3
```

5.8.2 Bitwise shorthand Assignments

- All of the binary bitwise operator have a shorthand form that combines an assignment with the bitwise opearation.
- For example:
  - X =X ^ 127;
  - X ^= 127;