Object Oriented Programming with Java (Subject Code: BCS-403)

Unit 1

Lecture 5

Lecture 5

- Arrays
- Strings

Java Array

- Array is a collection 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 a java array.
- Array in java is index based, first element of the array is stored at 0 index.

Advantage of Java Array

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

Disadvantage of Java Array

- **Size Limit:** We can store only 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.

Types of Array in java

There are two types of array.

- Single Dimensional Array
- Multidimensional Array

Single Dimensional Array in java

Syntax to Declare an Array in java

dataType[] arr; (or)

dataType []arr; (or)

dataType arr[];

Instantiation of an Array in java

Array RefVar=**new** datatype[size];

Example

int a[]=**new int**[5];

Example of single dimensional java array

```
class Testarray{
public static void main(String args[]){
  int a[]=new int[5];
a[0]=10;
a[1]=20;
a[2]=70;
a[3]=40;
a[4]=50;
for(int i=0;i<a.length;i++)
System.out.println(a[i]);
 }}
```

Declaration, Instantiation and Initialization of Java Array

We can declare, instantiate and initialize the java array together by:

```
int a[]={33,3,4,5};//declaration, instantiation and initialization
class Testarray1{
public static void main(String args[]){
 int a[]={33,3,4,5};//declaration, instantiation and initialization
 //printing array
for(int i=0;i<a.length;i++)//length is the property of array
System.out.println(a[i]);
}}
```

Passing Array to method in java

```
class Testarray2{
static void min(int arr[])
{
int min=arr[0];
for(int i=1;i<arr.length;i++)
 if(min>arr[i])
  min=arr[i];
  System.out.println(min);
}
  public static void main(String args[]){
  int a[]={33,3,4,5};
min(a);//passing array to method
  }}
```

Multidimensional array in java

In such case, data is stored in row and column based index (also known as matrix form).

Syntax to Declare Multidimensional Array in java.

- dataType[][] arrayRefVar; (or)
- dataType [][]arrayRefVar; (or)
- dataType arrayRefVar[][]; (or)
- dataType []arrayRefVar[];

Example to instantiate Multidimensional Array in java

```
int[][] arr=new int[3][3];//3 row and 3 column
```

Example to initialize Multidimensional Array in java

```
arr[0][0]=1;
arr[0][1]=2;
arr[0][2]=3;
arr[1][0]=4;
arr[1][1]=5;
arr[1][2]=6;
arr[2][0]=7;
arr[2][1]=8;
arr[2][2]=9;
```

Example of Multidimensional java array

```
class Testarray3{
public static void main(String args[]){
  //declaring and initializing 2D array
int arr[][]={{1,2,3},{2,4,5},{4,4,5}};
 //printing 2D array
for(int i=0;i<3;i++){
 for(int j=0;j<3;j++){
   System.out.print(arr[i][j]+" ");
 }
 System.out.println();
}
 }}
```

```
class Testarray5{
public static void main(String args[]){
int a[][]={{1,3,4},{3,4,5}};
int b[][]={{1,3,4},{3,4,5}};
int c[][]=new int[2][3];
for(int i=0;i<2;i++){
for(int j=0;j<3;j++){
c[i][j]=a[i][j]+b[i][j];
System.out.print(c[i][j]+" ");
}
System.out.println();//new line
}
}}
```

Example of Multidimensional java array

```
class Testarray3{
public static void main(String args[]){
  //declaring and initializing 2D array
int arr[][]={{1,2,3},{2,4,5},{4,4,5}};
 //printing 2D array
for(int i=0;i<3;i++){
 for(int j=0;j<3;j++){
   System.out.print(arr[i][j]+" ");
 }
 System.out.println();
}
 }}
```

```
class Testarray5{
public static void main(String args[]){
int a[][]={{1,3,4},{3,4,5}};
int b[][]={{1,3,4},{3,4,5}};
int c[][]=new int[2][3];
for(int i=0;i<2;i++){
for(int j=0;j<3;j++){
c[i][j]=a[i][j]+b[i][j];
System.out.print(c[i][j]+" ");
}
System.out.println();//new line
}
}}
```

Final arrays in Java

```
class Test
{
    public static void main(String args[])
    {
      final int arr[] = {1, 2, 3, 4, 5};  // Note: arr is final
      for (int i = 0; i < arr.length; i++)
      {
          arr[i] = arr[i]*10;
          System.out.println(arr[i]);
      }
    }
}
```

The array *arr* is declared as final, but the elements of array are changed without any problem.

Arrays are objects and object variables are always references in Java.

So, when we declare an object variable as final, it means that the variable cannot be changed to refer to anything else.

Jagged Array in Java

Jagged array is array of arrays such that member arrays can be of different sizes, i.e., we can create a 2-D arrays but with variable number of columns in each row. These type of arrays are also known as Jagged arrays.

```
class Main
{
   public static void main(String[] args)
   {
            int arr[][] = new int[2][];
            arr[0] = new int[3];
            arr[1] = new int[2];
            int count = 0;
      for (int i=0; i<arr.length; i++)
         for(int j=0; j<arr[i].length; j++)
            arr[i][j] = count++;
       System.out.println("Contents of 2D Jagged Array");
      for (int i=0; i<arr.length; i++)
      {
         for (int j=0; j<arr[i].length; j++)
            System.out.print(arr[i][j] + " ");
         System.out.println();
      }
   }}
```

Java String

- **Java String** provides a lot of concepts that can be performed on a string such as compare, concat, equals, split, length, replace, compareTo, intern, substring etc.
- In java, string is basically an object that represents sequence of char values.
- An array of characters works same as java string.

For example:

```
char[] ch={'j','a','v','a'};
String s=new String(ch);
```

is same as:

```
String s="java";
```

- The java.lang.String class implements *Serializable*, *Comparable* and *CharSequence* interfaces.
- The java String is immutable i.e. it cannot be changed but a new instance is created.
- For mutable class, you can use StringBuffer and StringBuilder class.

There are two ways to create String object:

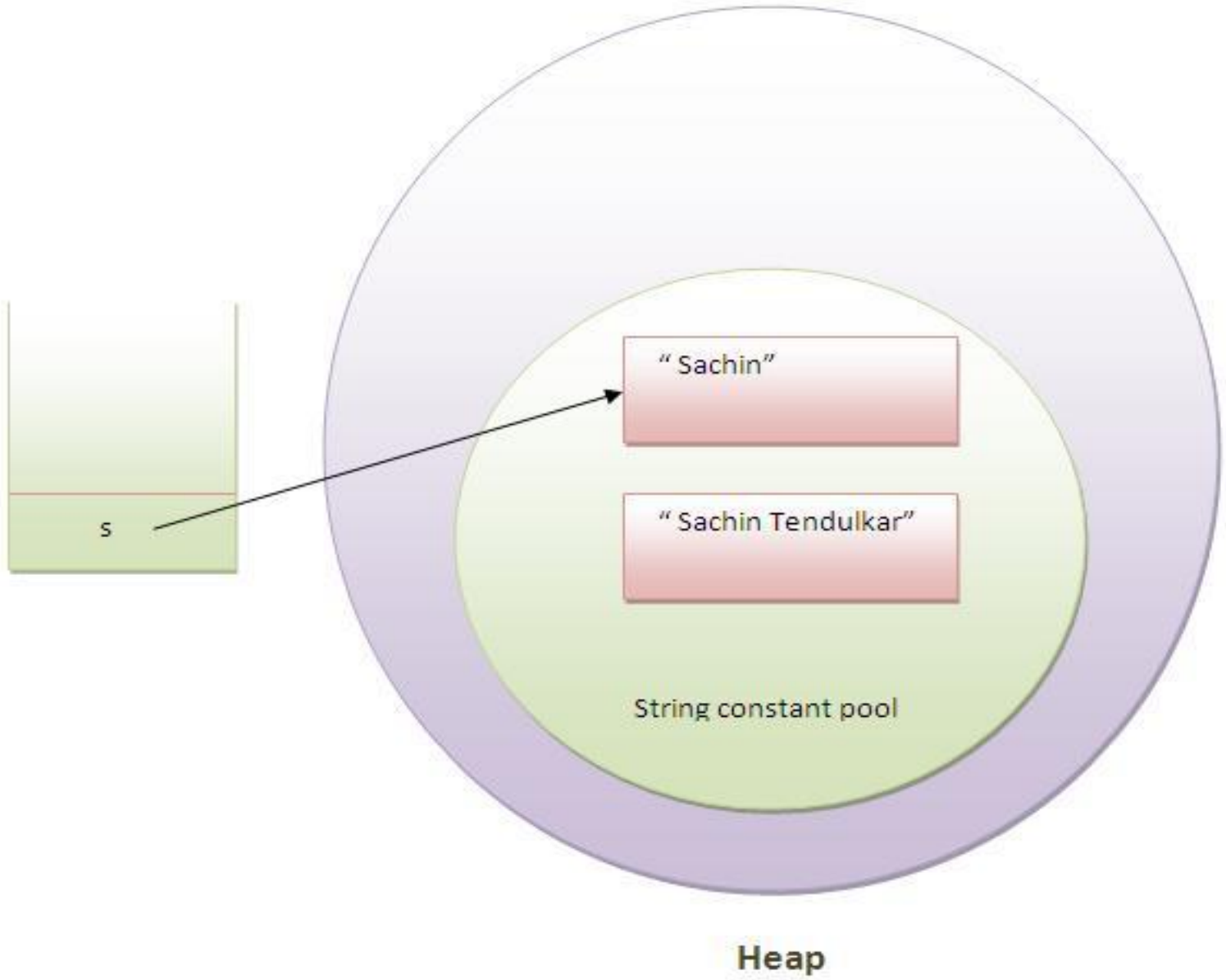
- By string literal
- By new keyword

Method	Description
char charAt(int index)	returns char value for the particular index
int length()	returns string length
String substring(int beginIndex)	returns substring for given begin index
String substring(int beginIndex, int endIndex)	returns substring for given begin index and end index
boolean contains(CharSequence s)	returns true or false after matching the sequence of char value
boolean equals(Object another)	checks the equality of string with object
boolean isEmpty()	checks if string is empty
String concat(String str)	concatinates specified string
String replace(char old, char new)	replaces all occurrences of specified char value
String replace(CharSequence old, CharSequence new)	replaces all occurrences of specified CharSequence
String trim()	returns trimmed string omitting leading and trailing spaces
String toUpperCase()	returns string in uppercase.
String toUpperCase(Locale l)	returns string in uppercase using specified locale.
int indexOf(int ch)	returns specified char value index
int indexOf(int ch, int fromIndex)	returns specified char value index starting with given index

Immutable String in Java

- In java, **string objects are immutable**. Immutable simply means unmodifiable or unchangeable.
- Once string object is created its data or state can't be changed but a new string object is created.

```
class Testimmutablestring{
 public static void main(String args[]){
   String s="Sachin";
   s.concat(" Tendulkar");//concat() method appends the stri
    ng at the end
   System.out.println(s);//will print Sachin because strings ar
    e immutable objects
 }
}
```


" Sachin"
" Sachin Tendulkar"
s
String constant pool
Heap

1) String compare by equals() method

- The String equals() method compares the original content of the string. It compares values of string for equality. String class provides two methods:
- **public boolean equals(Object another)** compares this string to the specified object.
- **public boolean equalsIgnoreCase(String another)** compares this String to another string, ignoring case.

```
class Teststringcomparison1{
 public static void main(String args[]){
   String s1="Sachin";
   String s2="Sachin";
   String s3=new String("Sachin");
   String s4="Saurav";
   System.out.println(s1.equals(s2));//true
   System.out.println(s1.equals(s3));//true
   System.out.println(s1.equals(s4));//false
 } }
```

Output: true true false

```
class Teststringcomparison2{
 public static void main(String args[]){
   String s1="Sachin";
   String s2="SACHIN";

   System.out.println(s1.equals(s2));//false
   System.out.println(s1.equalsIgnoreCase(s2));//true
 }
}
```

Output: false true

2) String compare by == operator

The = = operator compares references not values.

```
class Teststringcomparison3{
 public static void main(String args[]){
   String s1="Sachin";
   String s2="Sachin";
   String s3=new String("Sachin");
   System.out.println(s1==s2);//true
                                (because both refer to same instance)
  System.out.println(s1==s3);
                        //false(because s3 refers to instance created in non
    pool)
 }
}
```

3) String compare by compareTo() method

The String compareTo() method compares values lexicographically and returns an integer value that describes if first string is less than, equal to or greater than second string.

Suppose s1 and s2 are two string variables. If:

s1 == s2 :0

s1 > s2 :positive value

s1 < s2 :negative value

```
class Teststringcomparison4{
 public static void main(String args[]){
   String s1="Sachin";
   String s2="Sachin";
   String s3="Ratan";
   System.out.println(s1.compareTo(s2));//0
   System.out.println(s1.compareTo(s3));//1(because s1>s3)
   System.out.println(s3.compareTo(s1));//-1(because s3 < s1 )
 } }
```

Output:0 1 -1

String Concatenation in Java

In java, string concatenation forms a new string *that is* the combination of multiple strings.

There are two ways to concat string in java:

By + (string concatenation) operator

By concat() method

1) String Concatenation by + (string concatenation) operator

Java string concatenation operator (+) is used to add strings. For Example:

```
class TestStringConcatenation1{
 public static void main(String args[]){
   String s="Sachin"+" Tendulkar";
   System.out.println(s);//Sachin Tendulkar
 } }
```

Output:Sachin Tendulkar

Java String split

```
public class SplitExample{
public static void main(String args[]){
String s1="java string split method by javatpoint";
String[] words=s1.split(“ ");//splits the string based on whitespace
for(String w:words){
System.out.println(w);
}
}}
```

Java String endsWith

```
public class EndsWithExample{
public static void main(String args[]){
String s1="java by javatpoint";
System.out.println(s1.endsWith("t"));
System.out.println(s1.endsWith("point"));
}}
```

Output:

true true

substring() method

```
public class SubstringExample{
public static void main(String args[]){
String s1="javatpoint";
System.out.println(s1.substring(2,4));//returns va
System.out.println(s1.substring(2));//returns vatpoint
}}
```

Java String join() method example

```
public class StringJoinExample{
public static void main(String args[]){
String joinString1=String.join("-
   ","welcome","to","javatpoint");
System.out.println(joinString1);
}}
```

welcome-to-javatpoint