Array and String

Content

- **Single and Multidimensional Array**
- **String class**
- **StringBuffer class**
- **Operations on string**
- **Command line argument**
- **Use of Wrapper Class**

Java Arrays

- 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.
- We can store primitive values or objects in an array in Java. Like C/C++, we can also create single dimensional or multidimensional arrays in Java.

- Types of Array in java
  - 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**
  - arrayRefVar=**new** datatype[size];

```
class Testarray{
public static void main(String args[]){
int a[]=new int[5];//declaration and instantiation
a[0]=10;//initialization
a[1]=20;
a[2]=70;
a[3]=40;
a[4]=50;
//traversing array
for(int i=0;i<a.length;i++)//length is the property of array

System.out.println(a[i]);
}}
```

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

```
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();
}
}}
```

- **Copying a Java Array**
- We can copy an array to another by the arraycopy() method of System class.

```
//Java Program to copy a source array into a destination array in Java

class TestArrayCopyDemo {
  public static void main(String[] args) {
    //declaring a source array
    char[] copyFrom = { 'd', 'e', 'c', 'a', 'f', 'f', 'e',
            'i', 'n', 'a', 't', 'e', 'd' };
    //declaring a destination array
    char[] copyTo = new char[7];
    //copying array using System.arraycopy() method
    System.arraycopy(copyFrom, 2, copyTo, 0, 7);
    //printing the destination array
    System.out.println(String.valueOf(copyTo));
  }
}
```

Addition of 2 Matrices in Java

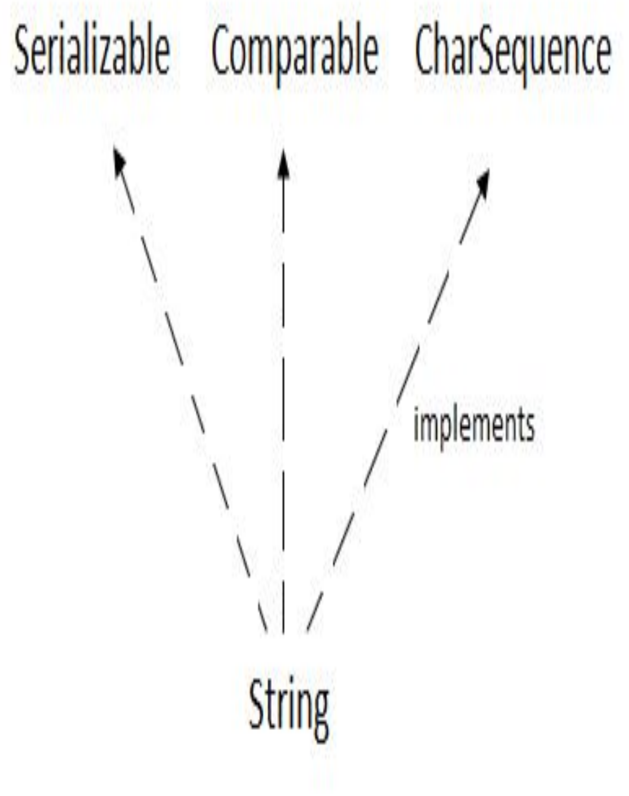
```
class Testarray5{
public static void main(String args[]){
//creating two matrices
int a[][]={{1,3,4},{3,4,5}};
int b[][]={{1,3,4},{3,4,5}};

//creating another matrix to store the sum of two matrices
int c[][]=new int[2][3];

//adding and printing addition of 2 matrices
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
}

}}
```

Java String

- 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={‘h’,’e’,’l’,’l’,’o’};
- String s=**new** String(ch);
- is same as:
- String s=“hello";
- **Java String** class provides a lot of methods to perform operations on strings such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), substring() etc.

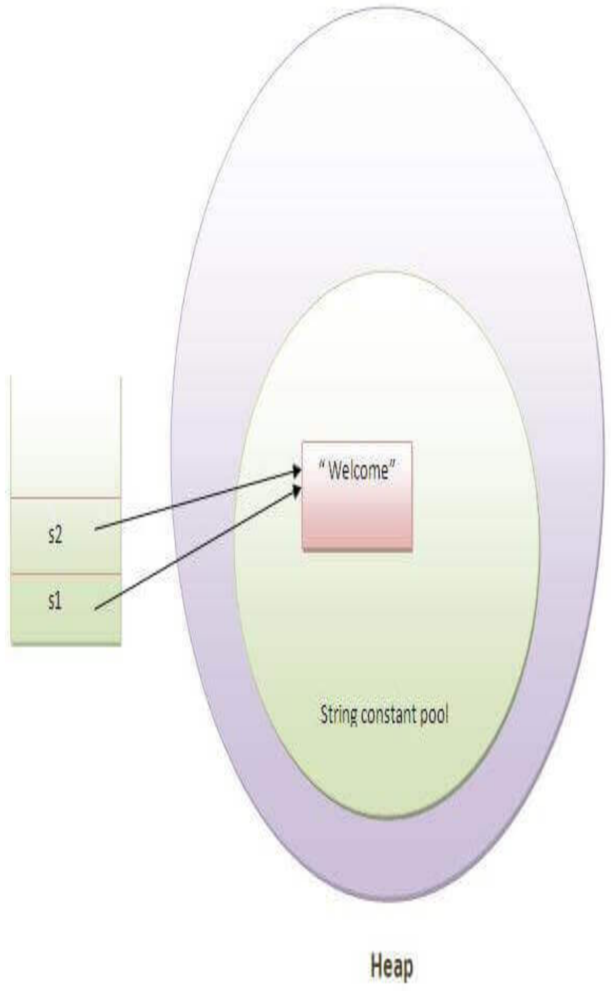
CharSequence Interface:

The CharSequence interface is used to represent the sequence of characters.
String, StringBuffer and StringBuilder classes implement it. It means, we can create strings in java by using these three classes.
The Java String is immutable which means it cannot be changed. Whenever we change any string, a new instance is created.
For mutable strings, you can use StringBuffer and StringBuilder classes.



- What is String in java
- Generally, String is a sequence of characters. But in Java, string is an object that represents a sequence of characters. The java.lang.String class is used to create a string object. The Java String is immutable .
- How to create a string object?
  - By string literal
  - By new keyword

- Java String literal is created by using double quotes. For Example:
  - String s="welcome";
- Each time you create a string literal, the JVM checks the "string constant pool" first.
- If the string already exists in the pool, a reference to the pooled instance is returned. If the string doesn't exist in the pool, a new string instance is created and placed in the pool.
- String objects are stored in a special memory area known as the "string constant pool”
- For example:
  - String s1="Welcome";
  - String s2="Welcome“



- Why Java uses the concept of String literal?
  - To make Java more memory efficient (because no new objects are created if it exists already in the string constant pool).

- 2) By new keyword
  - String s=**new** String("Welcome");//creates two obj ects and one reference variable
  - In such case, JVM will create a new string object in normal (non-pool) heap memory, and the literal "Welcome" will be placed in the string constant pool.
  - The variable s will refer to the object in a heap (non-pool).

Java String Example

```
public class StringExample{
public static void main(String args[]){
String s1="java";//creating string by java string literal
char ch[]={'s','t','r','i','n','g','s'};
String s2=new String(ch);//converting char array to string

String s3=new String("example");//creating java string by
    new keyword
System.out.println(s1);
System.out.println(s2);
System.out.println(s3);
}}
```

Java String class methods

1	char charAt(int index)	returns char value for the particular index
2	int length()	returns string length
3	static String format(String format, Object... args)	returns a formatted string.
4	static String format(Locale l, String format, Object... args)	returns formatted string with given locale.
5	String substring(int beginIndex)	returns substring for given begin index.
6	String substring(int beginIndex, int endIndex)	returns substring for given begin index and end index.
7	boolean contains(CharSequence s)	returns true or false after matching the sequence of char value.

8	static String join(CharSequence delimiter, CharSequence... elements)	returns a joined string.
9	static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements)	returns a joined string.
10	boolean equals(Object another)	checks the equality of string with the given object.
11	boolean isEmpty()	checks if string is empty.
12	String concat(String str)	concatenates the specified string.
13	String replace(char old, char new)	replaces all occurrences of the specified char value.
14	String replace(CharSequence old, CharSequence new)	replaces all occurrences of the specified CharSequence.
15	static String equalsIgnoreCase(String another)	compares another string. It doesn't check case.
16	String[] split(String regex)	returns a split string matching regex.

17	String[] split(String regex, int limit)	returns a split string matching regex and limit.
18	String intern()	returns an interned string.
19	int indexOf(int ch)	returns the specified char value index.
20	int indexOf(int ch, int fromIndex)	returns the specified char value index starting with given index.
21	int indexOf(String substring)	returns the specified substring index.
22	int indexOf(String substring, int fromIndex)	returns the specified substring index starting with given index.
23	String toLowerCase()	returns a string in lowercase.
24	String toLowerCase(Locale l)	returns a string in lowercase using specified locale.
25	String toUpperCase()	returns a string in uppercase.

26	String toUpperCase(Locale l)	returns a string in uppercase using specified locale.
27	String trim()	removes beginning and ending spaces of this string.
28	static String valueOf(int value)	converts given type into string. It is an overloaded method.

- **Java String compare**
- We can compare string in java on the basis of content and reference.
- It is used in
- **authentication** (by equals() method), **sorting** (by compareTo() method), **reference matching** (by == operator) etc.
- **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
 }
}
```

- 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 instan
     ce created in nonpool)
 }
}
```

- **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.
- Two strings are **lexicographically** equal if they are the same length and contain the same characters in the same positions. In this case, stringA. compareTo(stringB) returns 0. Otherwise, stringA. compareTo(stringB) returns a negative value if StringA comes first and a positive value if StringB comes first.
- 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 )
 }
}
```

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
- 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
 }
}
```

- String Concatenation by concat() method
- The String concat() method concatenates the specified string to the end of current string. Syntax:

  public String concat(String another)

```
class TestStringConcatenation3{
 public static void main(String args[]){
   String s1="Sachin ";
   String s2="Tendulkar";
   String s3=s1.concat(s2);   // Sachin Tendulakar
String s4 = s2.concat(s1);//Tendulkar Sachin
   System.out.println(s3);//Sachin Tendulkar
  }
}
```

Substring in Java

- A part of string is called **substring**. In other words, substring is a subset of another string. In case of substring startIndex is inclusive and endIndex is exclusive.
- Note: Index starts from 0.
- You can get substring from the given string object by one of the two methods:
- **public String substring(int startIndex):** This method returns new String object containing the substring of the given string from specified startIndex (inclusive).
- **public String substring(int startIndex, int endIndex):** This method returns new String object containing the substring of the given string from specified startIndex to endIndex
- In case of string:
  - **startIndex:** inclusive
  - **endIndex:** exclusive

```
public class TestSubstring{
 public static void main(String args[]){
   String s="SachinTendulkar";
   System.out.println(s.substring(6));//Tendulkar
   System.out.println(s.substring(0,6));//Sachin
 }
}
```

- **Java String toUpperCase() and toLowerCase() method**
  - The java string toUpperCase() method converts this string into uppercase letter and string toLowerCase() method into lowercase letter.

```
String s="Sachin";
System.out.println(s.toUpperCase());//SACHIN
System.out.println(s.toLowerCase());//sachin
System.out.println(s);//Sachin(no change in original)
```

- **Java String trim() method**
  - The string trim() method eliminates white spaces before and after string.

```
String s="  Sachin  ";
System.out.println(s);//  Sachin
System.out.println(s.trim());//Sachin
```

- **Java String startsWith() and endsWith() method**

```
String s="Sachin";
 System.out.println(s.startsWith("Sa"));//true
 System.out.println(s.endsWith(“n"));//true
```

- **Java String charAt() method**
  - The string charAt() method returns a character at specified index.

```
String s="Sachin";
System.out.println(s.charAt(0));//S
System.out.println(s.charAt(5));//n
```

- **Java String length() method**
  - The string length() method returns length of the string.

```
String s="Sachin";
System.out.println(s.length());//6
```

- **Java String valueOf() method**
  - The string valueOf() method coverts given type such as int, long, float, double, boolean, char and char array into string.

```
int a=10;
Ans = a + 10 // 20
String s=String.valueOf(a);
System.out.println(s+10);  //1010
```

- **Java String replace() method**
  - The string replace() method replaces all occurrence of first sequence of character with second sequence of character.

```
String s1="Java is a programming language. Java is a platform. Java i
   s an Island.";
String replaceString=s1.replace("Java","Kava");//replaces all occurr
   ences of "Java" to "Kava"
System.out.println(replaceString);
```

StringBuffer Class

- Java StringBuffer class is used to create mutable (modifiable) string. The StringBuffer class in java is same as String class except it is mutable i.e. it can be changed.

Constructor	Description
StringBuffer()	creates an empty string buffer with the initial capacity of 16.
StringBuffer(String str)	creates a string buffer with the specified string.
StringBuffer(int capacity)	creates an empty string buffer with the specified capacity as length.

- What is mutable string
  - A string that can be modified or changed is known as mutable string. StringBuffer and StringBuilder classes are used for creating mutable string.

- **StringBuffer append() method**
  - The append() method concatenates the given argument with this string

```
class StringBufferExample{
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello ");
sb.append("Java");//now original string is changed
sb.append(“world”)
System.out.println(sb);// Hello Java
}
}
```

- **StringBuffer insert() method**
  - The insert() method inserts the given string with this string at the given position.

```
class StringBufferExample2{
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello");
sb.insert(1,"Java");//now original string is changed
System.out.println(sb);//prints HJavaello
}
}
```

- **StringBuffer replace() method**
  - The replace() method replaces the given string from the specified beginIndex and endIndex.

```
class StringBufferExample3{
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello");
sb.replace(1,3,"Java");
System.out.println(sb);//prints HJavalo
}
}
```

- **StringBuffer delete() method**
  - The delete() method of StringBuffer class deletes the string from the specified beginIndex to endIndex.

```
class StringBufferExample4{
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello");
sb.delete(2,5);
System.out.println(sb);//prints Heo
}
}
```

- **StringBuffer reverse() method**
  - The reverse() method of StringBuilder class reverses the current string.

```
class StringBufferExample5{
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello");
sb.reverse();
System.out.println(sb);//prints olleH
}
}
```

- **StringBuffer capacity() method**
  - The capacity() method of StringBuffer class returns the current capacity of the buffer. The default capacity of the buffer is 16. If the number of character increases from its current capacity, it increases the capacity by (oldcapacity*2)+2. For example if your current capacity is 16, it will be (16*2)+2=34.

```
class StringBufferExample6{
public static void main(String args[]){
StringBuffer sb=new StringBuffer();
System.out.println(sb.capacity());//default 16
sb.append("Hello");
System.out.println(sb.capacity());//now 16
sb.append("java is my favourite language");
System.out.println(sb.capacity());//now (16*2)+2=34 i.e (oldcapacity*
    2)+2
}
}
```

StringBuilder class

- Java StringBuilder class is used to create mutable (modifiable) string. The Java StringBuilder class is same as StringBuffer class except that it is non-synchronized. It is available since JDK 1.5

Constructor	Description
StringBuilder()	creates an empty string Builder with the initial capacity of 16.
StringBuilder(String str)	creates a string Builder with the specified string.
StringBuilder(int length)	creates an empty string Builder with the specified capacity as length.

```
class StringBuilderExample{
public static void main(String args[]){
StringBuilder sb=new StringBuilder("Hello ");
sb.append("Java");//now original string is chang
    ed
System.out.println(sb);//prints Hello Java
}
}
```

Difference between String and StringBuffer

No.	String	StringBuffer
1)	String class is immutable.	StringBuffer class is mutable.
2)	String is slow and consumes more memory when you concat too many strings because every time it creates new instance.	StringBuffer is fast and consumes less memory when you cancat strings.
3)	String class overrides the equals() method of Object class. So you can compare the contents of two strings by equals() method.	StringBuffer class doesn't override the equals() method of Object class.

Difference between StringBuffer and StringBuilder

No.	StringBuffer	StringBuilder
1)	StringBuffer is *synchronized* i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously.	StringBuilder is *non-synchronized* i.e. not thread safe. It means two threads can call the methods of StringBuilder simultaneously.
2)	StringBuffer is *less efficient* than StringBuilder.	StringBuilder is *more efficient* than StringBuffer.

Command-line argument

- The java command-line argument is an argument i.e. passed at the time of running the java program.
- The arguments passed from the console can be received in the java program and it can be used as an input.
- So, it provides a convenient way to check the behavior of the program for the different values. You can pass **N** (1,2,3 and so on) numbers of arguments from the command prompt.

```
class CommandLineExample{
public static void main(String args[]){
System.out.println("Your first argument is: "+arg
   s[0]);
}
}
compile by > javac CommandLineExample.java
run by > java CommandLineExample Hello Worl
   d
Output: Your first argument is: Hello
```

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

for(int i=0;i<args.length;i++)
System.out.println(args[i]);

}
}
```

```
compile by > javac A.java
run by > java A abc xyz cd hjuk
o/p: abc
Xyz
Cd
hjuk
```

Use of Wrapper Class

- The **wrapper class in Java** provides the mechanism *to convert primitive into object and object into primitive*.
- Since J2SE 5.0, **autoboxing** and **unboxing** feature convert primitives into objects and objects into primitives automatically. The automatic conversion of primitive into an object is known as autoboxing and vice-versa unboxing.
- **Use of Wrapper classes in Java**
- Java is an object-oriented programming language, so we need to deal with objects many times like in Collections, Serialization, Synchronization, etc.

- **Change the value in Method:** Java supports only call by value. So, if we pass a primitive value, it will not change the original value. But, if we convert the primitive value in an object, it will change the original value.
- **Serialization:** We need to convert the objects into streams to perform the serialization. If we have a primitive value, we can convert it in objects through the wrapper classes.
- **Synchronization:** Java synchronization works with objects in Multithreading.
- **java.util package:** The java.util package provides the utility classes to deal with objects.
- **Collection Framework:** Java collection framework works with objects only. All classes of the collection framework (ArrayList, LinkedList, Vector, HashSet, LinkedHashSet, TreeSet, PriorityQueue, ArrayDeque, etc.) deal with objects only.

Primitive Type	Wrapper class
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

- **Autoboxing**
  - The automatic conversion of primitive data type into its corresponding wrapper class is known as autoboxing, for example, byte to Byte, char to Character, int to Integer and so on.

- **Wrapper class Example: Primitive to Wrapper**

```
//Autoboxing example of int to Integer
public class WrapperExample1{
public static void main(String args[]){
//Converting int into Integer
int a=20;   //variable
Integer i=Integer.valueOf(a);//converting int into Integer explicitly
Integer j=a;//autoboxing, now compiler will write Integer.valueOf(a) i
    nternally

System.out.println(a+" "+i+" "+j);
}}
```

- **Unboxing**
  - The automatic conversion of wrapper type into its corresponding primitive type is known as unboxing. It is the reverse process of autoboxing..
- **Wrapper class Example: Wrapper to Primitive**

```
public class WrapperExample2{
public static void main(String args[]){
//Converting Integer to int
Integer a=new Integer(3);
int i=a.intValue();//converting Integer to int explicitly
int j=a;//unboxing, now compiler will write a.intValue() internally

System.out.println(a+" "+i+" "+j);
}}
```