String

- It is a sequence of character.
- Strings are object in java.
- Objects of a String are immutable means it can not be change once created.

Creating a String in Java

Two ways to create a String in Java:

(1) String literal

String str = "kalibermind";

(2) By Using new Keyword

String str = new String("kalibermind);

Example:

```
public class StringDemo {
public static void main(String[] args)
{
char[] charArray = {'K','A','L','I','B','E','R','M','I','N','D'};
String str = new String(charArray);
System.out.println(str);
}
}
```

String Method

String str = "kaliberMind";

1. **int length():** Returns the number of characters in the String.

```
int len = str.length();
System.out.println(len); // return 11
```

2. **Char charAt(int i):** Returns the character at i^{th} index.

```
char c = str.charAt(3);
System.out.println(c); // return 'i'
```

3. **String substring (int i):** Return the substring from the i^{th} index character to end.

```
String substr = str.substring(3);
```

```
System.out.println(substr); // returns “ibermind”
```

4. **String substring (int i, int j):** Returns the substring from i to j-1 index.

```
String substr = str.substring(0, 8);
System.out.println(substr); // returns “kaliberm”
```

5. **String concat(String str):** Concatenates specified string to the end of this string.

```
String s1 = ”Kaliber”;
String s2 = ”mind”;
String s3 = s1.concat(s2);
System.out.println(s3); // returns “kalibermind”
```

6. **int indexOf (String s):** Returns the index within the string of the first occurrence of the specified string.

```
int index = str.indexOf(“Mind”); // returns 7
```

7. **int indexOf (String s, int i):** Returns the index within the string of the first occurrence of the specified string, starting at the specified index.

```
String s = ”Learn Share Learn”;
int output = s.indexOf(‘a’,3); // returns 8
```

8. **int lastindexOf(char c):** Returns the index within the string of the last occurrence of the specified string.

```
String s = ”Learn Share Learn”;
int output = s.lastindexOf(‘a’);
System.out.println(output);  // returns 14
```

9. **boolean equals(Object otherObj):** Compares this string to the specified object.

```
Boolean out = “Kaliber”.equals(“Kaliber”);
System.out.println(out); // returns true
Boolean out = “Kaliber”.equals(“kaliber”);
System.out.println(out); // returns false
```

10. **boolean equalsIgnoreCase (String anotherString):** Compares string to another string, ignoring case considerations.

```
Boolean out = “Kaliber”.equalsIgnoreCase(“Kaliber”);
System.out.println(out); // returns true
```

```
Boolean out = “Kaliber”.equalsIgnoreCase(“kaliber”);
System.out.println(out); // returns true
```

11. **int compareTo(String anotherString):** Compares two string lexicographically(Alphabetical Order)

    - It compares strings on the basis of Unicode value of each character in the strings.
    - If first string is lexicographically greater than second string, it returns positive number (difference of character value). If first string is less than second string lexicographically, it returns negative number and if first string is lexicographically equal to second string, it returns 0.

if s1 > s2, it returns positive number
if s1 < s2, it returns negative number
if s1 == s2, it returns 0

```
1. public class CompareToExample{
2. public static void main(String args[]){
3. String s1="hello";
4. String s2="hello";
5. String s3="meklo";
6. String s4="hemlo";
7. String s5="flag";
8. System.out.println(s1.compareTo(s2));//0 because both are equal
9. System.out.println(s1.compareTo(s3));//-5 because "h" is 5 times lower than "m"
10. System.out.println(s1.compareTo(s4));//-1 because "l" is 1 times lower than "m"
11. System.out.println(s1.compareTo(s5));//2 because "h" is 2 times greater than "f"
12. }}
```

Output:

```
0
-5
-1
2
```

12.int compareToIgnoreCase(String anotherString): Compares two string lexicographically, ignoring case considerations.

```
int out = s1.compareToIgnoreCase(s2);
// where s1 ans s2 are
// strings to be compared
```

This returns difference s1-s2. If :

```
out < 0  // s1 comes before s2
out = 0   // s1 and s2 are equal.
out >0   // s1 comes after s2.
```

Note- In this case, it will not consider case of a letter (it will ignore whether it is uppercase or lowercase).

13.String toLowerCase(): Converts all the characters in the String to lower case.

```
String str = "HeLLo";
String str2 = str.toLowerCase();
System.out.println(str2); // returns "hello"
```

14. String toUpperCase(): Converts all the characters in the String to upper case.

```
String str = "HeLlo";
String str2 = str.toUpperCase();
System.out.println(str2); // returns "HELLO"
```

15. String trim(): Returns the copy of the String, by removing whitespaces at both ends. It does not affect whitespaces in the middle.

```
String word1 = " Learn Share Learn ";
String word2 = word1.trim(); // returns "Learn Share Learn"
```

16. String replace (char oldChar, char newChar): Returns new string by replacing all occurrences of *oldChar* with *newChar*.

```
String s1 = "kaliberMind";
String s2 = "kaliberMind".replace('M' ,'m'); // returns "kalibermind"
```

Note:- s1 is still kaliberMind and s2 is kalibermind.

Immutable String in Java

In java, **string objects are immutable**. Immutable simply means unmodifiable or unchangeable.

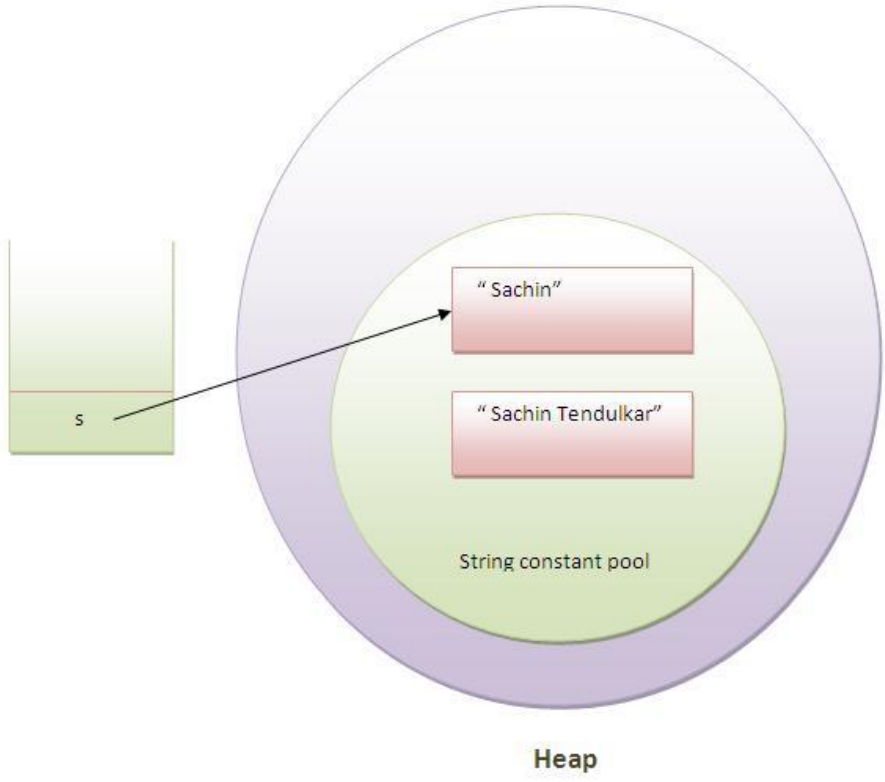
Once string object is created its data or state can't be changed but a new string object is created.

Let's try to understand the immutability concept by the example given below:

```
1. class Testimmutablestring{
2.  public static void main(String args[]){
3.    String s="Sachin";
4.    s.concat(" Tendulkar");//concat() method appends the string at the end
5.    System.out.println(s);//will print Sachin because strings are immutable objects
6.  }
7. }
```

Output: Sachin

Now it can be understood by the diagram given below. Here Sachin is not changed but a new object is created with sachintendulkar. That is why string is known as immutable.



- As you can see in the above figure that two objects are created but s reference variable still refers to "Sachin" not to "Sachin Tendulkar".

But if we explicitely assign it to the reference variable, it will refer to "Sachin Tendulkar" object. For example:

```
1. class Testimmutablestring1{
2.  public static void main(String args[]){
3.    String s="Sachin";
4.    s=s.concat(" Tendulkar");
5.    System.out.println(s);
6.  }
7. }
```

Output: Sachin Tendulkar

In such case, s points to the "Sachin Tendulkar". Please notice that still sachin object is not modified.

Q. Why string objects are immutable in java?

Because java uses the concept of string literal. Suppose there are 5 reference variables,all referes to one object "sachin".If one reference variable changes the value of the object, it will be affected to all the reference variables. That is why string objects are immutable in java.

- We know String can be created by two ways : One is String Literal and another by using new keyword.

- String Literal is memory efficient in Java that's why String is immutable.
  String str1 = "kalibermind";

  String str2 = "kalibermind;

  System.out.println(str1 == str2); //returns True.

Because, both acquire same memory location in Java.

- By using new keyword
  String obj1 = new String("kalibermind academy");
  String obj2 = new String("kalibermind academy");

  System.out.println(obj1== obj2); //returns false.

Because, both acquire different memory location.

Q.How to write immutable class?

There are many immutable classes like String, Boolean, Byte, Short, Integer, Long, Float, Double etc. In short, all the wrapper classes and String class is immutable. We can also create immutable class by creating final class that have final data members as the example given below:

Example to create Immutable class

In this example, we have created a final class named Employee. It have one final datamember, a parameterized constructor and getter method.

```
1. public final class Employee{
2. final String pancardNumber;
3.
4. public Employee(String pancardNumber){
5. this.pancardNumber=pancardNumber;
6. }
7.
8. public String getPancardNumber(){
9. return pancardNumber;
10.}
11.
12.}
```

The above class is immutable because:

- The instance variable of the class is final i.e. we cannot change the value of it after creating an object.
- The class is final so we cannot create the subclass.
- There is no setter methods i.e. we have no option to change the value of the instance variable.

These points makes this class as immutable.

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.

- StringBuffer represents growable and writable character sequence.
- StringBuffer size increases dynamically.
- It may have characters or String inserted or appended at the end. // append = joint.

Note: Java StringBuffer class is thread-safe i.e. multiple threads cannot access it simultaneously. So it is safe and will result in an order.

Example :

```
public class StringBufferExample {

 public static void main(String[] args)

{

        String str = “kalibermind”;

        StringBuffer sb = new StringBuffer(str);

                sb.append(“Academy”);

        System.out.println(sb.toString());

}

}
```

Output: kalibermind Academy

Important Constructors of StringBuffer class

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.

Important methods of StringBuffer class

Modifier and Type	Method	Description
public synchronized StringBuffer	append(String s)	is used to append the specified string with this string. The append() method is overloaded like append(char), append(boolean), append(int), append(float), append(double) etc.
public synchronized StringBuffer	insert(int offset, String s)	is used to insert the specified string with this string at the specified position. The insert() method is overloaded like insert(int, char), insert(int, boolean), insert(int, int), insert(int, float), insert(int, double) etc.
public synchronized StringBuffer	replace(int startIndex, int endIndex, String str)	is used to replace the string from specified startIndex and endIndex.
public synchronized StringBuffer	delete(int startIndex, int endIndex)	is used to delete the string from specified startIndex and endIndex.
public synchronized StringBuffer	reverse()	is used to reverse the string.
public int	capacity()	is used to return the current capacity.
public void	ensureCapacity(int minimumCapacity)	is used to ensure the capacity at least equal to the given minimum.

public char	charAt(int index)	is used to return the character at the specified position.
public int	length()	is used to return the length of the string i.e. total number of characters.
public String	substring(int beginIndex)	is used to return the substring from the specified beginIndex.
public String	substring(int beginIndex, int endIndex)	is used to return the substring from the specified beginIndex and endIndex.

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.

1) StringBuffer append() method

The append() method concatenates the given argument with this string.

```
1. class StringBufferExample{
2. public static void main(String args[]){
3. StringBuffer sb=new StringBuffer("Hello ");
4. sb.append("Java");//now original string is changed
5. System.out.println(sb);//prints Hello Java
6. }
7. }
```

2) StringBuffer insert() method

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

```
1. class StringBufferExample2{
2. public static void main(String args[]){
3. StringBuffer sb=new StringBuffer("Hello ");
4. sb.insert(1,"Java");//now original string is changed
5. System.out.println(sb);//prints HJavaello
6. }
7. }
```

3) StringBuffer replace() method

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

```
1. class StringBufferExample3{
2. public static void main(String args[]){
3. StringBuffer sb=new StringBuffer("Hello");
4. sb.replace(1,3,"Java");
5. System.out.println(sb);//prints HJavalo
6. }
7. }
```

4) StringBuffer delete() method

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

```
1. class StringBufferExample4{
2. public static void main(String args[]){
3. StringBuffer sb=new StringBuffer("Hello");
4. sb.delete(1,3);
5. System.out.println(sb);//prints Hlo
6. }
7. }
```

5) StringBuffer reverse() method

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

```
1. class StringBufferExample5{
2. public static void main(String args[]){
3. StringBuffer sb=new StringBuffer("Hello");
4. sb.reverse();
5. System.out.println(sb);//prints olleH
6. }
7. }
```

6) 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.

```
1. class StringBufferExample6{
```

```
2. public static void main(String args[]){
3. StringBuffer sb=new StringBuffer();
4. System.out.println(sb.capacity());//default 16
5. sb.append("Hello");
6. System.out.println(sb.capacity());//now 16
7. sb.append("java is my favourite language");
8. System.out.println(sb.capacity());//now (16*2)+2=34 i.e (oldcapacity*2)+2
9. }
10. }
```

7) StringBuffer ensureCapacity() method

The ensureCapacity() method of StringBuffer class ensures that the given capacity is the minimum to the current capacity. If it is greater than the 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.

```
1. class StringBufferExample7{
2. public static void main(String args[]){
3. StringBuffer sb=new StringBuffer();
4. System.out.println(sb.capacity());//default 16
5. sb.append("Hello");
6. System.out.println(sb.capacity());//now 16
7. sb.append("java is my favourite language");
8. System.out.println(sb.capacity());//now (16*2)+2=34 i.e (oldcapacity*2)+2
9. sb.ensureCapacity(10);//now no change
10. System.out.println(sb.capacity());//now 34
11. sb.ensureCapacity(50);//now (34*2)+2
12. System.out.println(sb.capacity());//now 70
13. }
14. }
```

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.

Important Constructors of StringBuilder class

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.

Important methods of StringBuilder class

Method	Description
public StringBuilder append(String s)	is used to append the specified string with this string. The append() method is overloaded like append(char), append(boolean), append(int), append(float), append(double) etc.
public StringBuilder insert(int offset, String s)	is used to insert the specified string with this string at the specified position. The insert() method is overloaded like insert(int, char), insert(int, boolean), insert(int, int), insert(int, float), insert(int, double) etc.
public StringBuilder replace(int startIndex, int endIndex, String str)	is used to replace the string from specified startIndex and endIndex.

public StringBuilder delete(int startIndex, int endIndex)	is used to delete the string from specified startIndex and endIndex.
public StringBuilder reverse()	is used to reverse the string.
public int capacity()	is used to return the current capacity.
public void ensureCapacity(int minimumCapacity)	is used to ensure the capacity at least equal to the given minimum.
public char charAt(int index)	is used to return the character at the specified position.
public int length()	is used to return the length of the string i.e. total number of characters.
public String substring(int beginIndex)	is used to return the substring from the specified beginIndex.
public String substring(int beginIndex, int endIndex)	is used to return the substring from the specified beginIndex and endIndex.

Java StringBuilder Examples

Let's see the examples of different methods of StringBuilder class.

1) StringBuilder append() method

The StringBuilder append() method concatenates the given argument with this string.

```
1. class StringBuilderExample{
2. public static void main(String args[]){
3. StringBuilder sb=new StringBuilder("Hello ");
4. sb.append("Java");//now original string is changed
5. System.out.println(sb);//prints Hello Java
6. }
```

```
7. }
```

2) StringBuilder insert() method

The StringBuilder insert() method inserts the given string with this string at the given position.

```
1. class StringBuilderExample2{
2. public static void main(String args[]){
3. StringBuilder sb=new StringBuilder("Hello ");
4. sb.insert(1,"Java");//now original string is changed
5. System.out.println(sb);//prints HJavaello
6. }
7. }
```

3) StringBuilder replace() method

The StringBuilder replace() method replaces the given string from the specified beginIndex and endIndex.

```
1. class StringBuilderExample3{
2. public static void main(String args[]){
3. StringBuilder sb=new StringBuilder("Hello");
4. sb.replace(1,3,"Java");
5. System.out.println(sb);//prints HJavalo
6. }
7. }
```

4) StringBuilder delete() method

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

```
1. class StringBuilderExample4{
2. public static void main(String args[]){
3. StringBuilder sb=new StringBuilder("Hello");
4. sb.delete(1,3);
5. System.out.println(sb);//prints Hlo
6. }
7. }
```

5) StringBuilder reverse() method

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

```
1. class StringBuilderExample5{
2. public static void main(String args[]){
3. StringBuilder sb=new StringBuilder("Hello");
4. sb.reverse();
5. System.out.println(sb);//prints olleH
6. }
7. }
```

6) StringBuilder capacity() method

The capacity() method of StringBuilder class returns the current capacity of the Builder. The default capacity of the Builder 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.

```
1. class StringBuilderExample6{
2. public static void main(String args[]){
3. StringBuilder sb=new StringBuilder();
4. System.out.println(sb.capacity());//default 16
5. sb.append("Hello");
6. System.out.println(sb.capacity());//now 16
7. sb.append("java is my favourite language");
8. System.out.println(sb.capacity());//now (16*2)+2=34 i.e (oldcapacity*2)+2
9. }
10.}
```

7) StringBuilder ensureCapacity() method

The ensureCapacity() method of StringBuilder class ensures that the given capacity is the minimum to the current capacity. If it is greater than the 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.

```
1. class StringBuilderExample7{
2. public static void main(String args[]){
3. StringBuilder sb=new StringBuilder();
4. System.out.println(sb.capacity());//default 16
5. sb.append("Hello");
6. System.out.println(sb.capacity());//now 16
7. sb.append("java is my favourite language");
8. System.out.println(sb.capacity());//now (16*2)+2=34 i.e (oldcapacity*2)+2
9. sb.ensureCapacity(10);//now no change
10.System.out.println(sb.capacity());//now 34
```

```
11.sb.ensureCapacity(50);//now (34*2)+2
12.System.out.println(sb.capacity());//now 70
13.}
14.}
```

Note:

(1)Objects of String are Immutable and Object of StringBuffer & Builder are mutable.

(2)StringBuffer and StringBuilder are similar but , a StringBuilder is faster and preffered over StringBuffer for single threaded program.

(3) If thread safety is needed then StringBuffer is used.

Difference between StringBuffer and StringBuilder

There are many differences between StringBuffer and StringBuilder. A list of differences between StringBuffer and StringBuilder are given below:

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.

StringTokenizer in Java

The **java.util.StringTokenizer** class allows you to break a string into tokens. It is simple way to break string.

It doesn't provide the facility to differentiate numbers, quoted strings, identifiers etc.

Constructors of StringTokenizer class

There are 3 constructors defined in the StringTokenizer class.

Constructor	Description
StringTokenizer(String str)	creates StringTokenizer with specified string.
StringTokenizer(String str, String delim)	creates StringTokenizer with specified string and delimeter.
StringTokenizer(String str, String delim, boolean returnValue)	creates StringTokenizer with specified string, delimeter and returnValue. If return value is true, delimiter characters are considered to be tokens. If it is false, delimiter characters serve to separate tokens.

Methods of StringTokenizer class

The 6 useful methods of StringTokenizer class are as follows:

Public method	Description
boolean hasMoreTokens()	checks if there is more tokens available.
String nextToken()	returns the next token from the StringTokenizer object.
String nextToken(String delim)	returns the next token based on the delimeter.
boolean hasMoreElements()	same as hasMoreTokens() method.
Object nextElement()	same as nextToken() but its return type is Object.
int countTokens()	returns the total number of tokens.

Simple example of StringTokenizer class

Let's see the simple example of StringTokenizer class that tokenizes a string "my name is khan" on the basis of whitespace.

```
1. import java.util.StringTokenizer;
2. public class Simple{
3.  public static void main(String args[]){
4.    StringTokenizer st = new StringTokenizer("my name is khan"," ");
5.     while (st.hasMoreTokens()) {
6.         System.out.println(st.nextToken());
7.     }
8.    }
9. }
```

Output: my
name
is
khan

Example of nextToken(String delim) method of StringTokenizer class

```
1. import java.util.*;
2.
3. public class Test {
4.    public static void main(String[] args) {
5.        StringTokenizer st = new StringTokenizer("my,name,is,khan");
6.
7.       // printing next token
8.       System.out.println("Next token is : " + st.nextToken(","));
9.    }
10.}
```

Output: Next token is : my