String Handling in Java

Introduction

String Handling provides a lot of concepts that can be performed on a string such as concatenating string, comparing string, substring etc.

In java, string is basically an *immutable* object. We will discuss about immutable string later. Let's first understand what is string and how we can create the string object.

String

Generally string is a sequence of characters. But in java, string is an object. String class is used to create string object.

How to create String object?

There are two ways to create String object:

1. By string literal
2. By new keyword

1) String literal

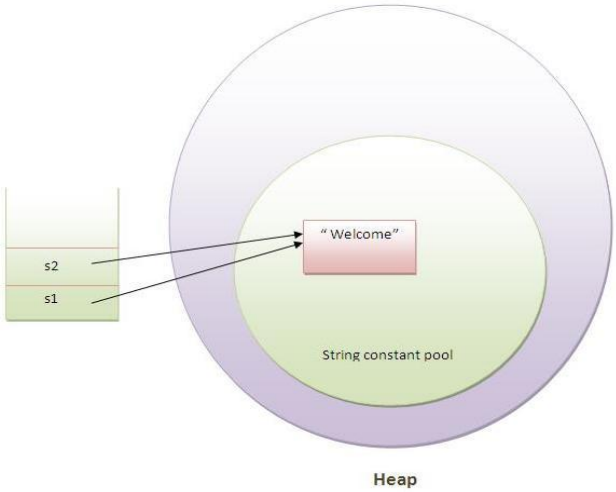
String literal is created by double quote. For Example:

1. String s="Hello";

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 returns. If the string does not exist in the pool, a new String object instantiates, then is placed in the pool. For example:

Note: In Java when an object or an array is created, memory is allocated to them from heap. The JVM through the use of **new** operator allocates memory from the heap for an object. String constant pool is part of such heap memory.

1. String s1="Welcome";
2. String s2="Welcome";//no new object will be created



In the above example only one object will be created. First time JVM will find no string object with the name "Welcome" in string constant pool, so it will create a new object. Second time it will find the string with the name "Welcome" in string constant pool, so it will not create new object whether will return the reference to the same instance.

Note: **String objects are stored in a special memory area known as string constant pool inside the Heap memory.**

Why java uses concept of string literal?

To make Java more memory efficient (because no new objects are created if it exists already in string constant pool).

2) By new keyword

1. String s=new String("Welcome");//creates two objects

In such case, JVM will create a new String object in normal(nonpool) Heap memory and the literal "Welcome" will be placed in the string constant pool. The variable *s* will refer to the object in Heap(nonpool).

What we will learn in String Handling ?

- Concept of String
- Immutable String
- String Comparison
- String Concatenation
- Concept of Substring

- String class methods and its usage
- StringBuffer class
- StringBuilder class
- Creating Immutable class
- toString() method
- StringTokenizer class

Immutable String in Java

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

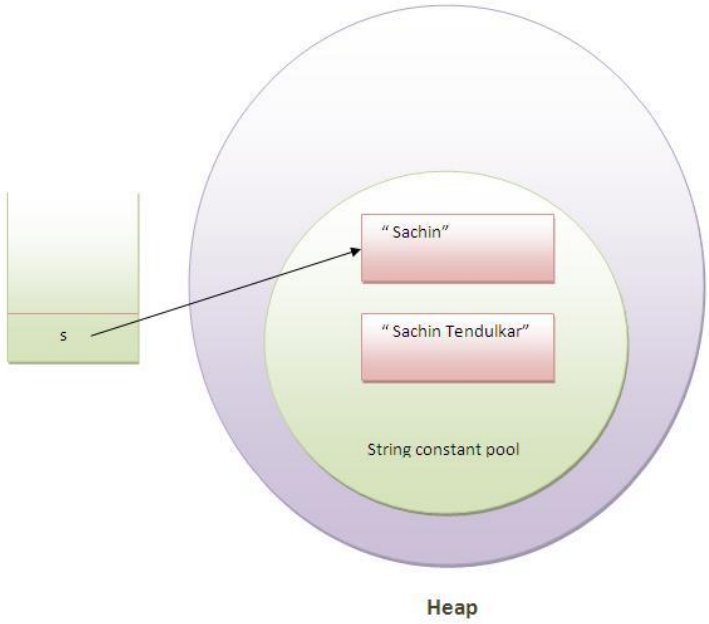
Once string object is created its data or state can't be changed but a new string object is created.

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

```
1. class Simple{
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 explicitly assign it to the reference variable, it will refer to "Sachin Tendulkar" object. For example:

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

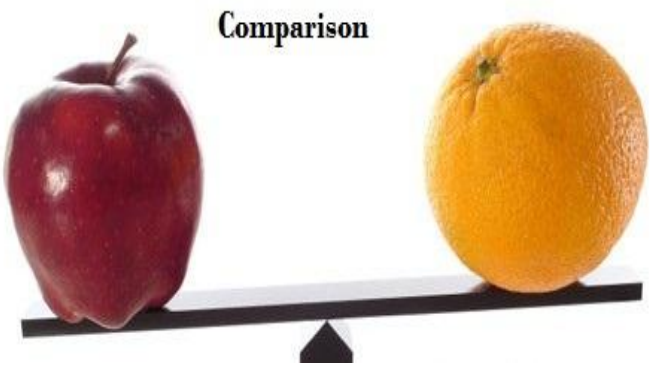
Output:Sachin Tendulkar

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

Why string objects are immutable in java?

Because java uses the concept of string literal. Suppose there are 5 reference variables, all refers 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.

String comparison in Java



We can compare two given strings on the basis of content and reference.

It is used in **authentication** (by equals() method), **sorting** (by compareTo() method), **reference matching** (by == operator) etc.

There are three ways to compare String objects:

1. By equals() method
2. By = = operator
3. By compareTo() method

1) By equals() method

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.

```
1. class Simple{
2.  public static void main(String args[]){
3.
4.    String s1="Sachin";
5.    String s2="Sachin";
6.    String s3=new String("Sachin");
7.    String s4="Saurav";
8.
9.    System.out.println(s1.equals(s2));//true
10.   System.out.println(s1.equals(s3));//true
11.   System.out.println(s1.equals(s4));//false
12. }
```

```
13. }
```

Output:
true
true
false

```
1.  //Example of equalsIgnoreCase(String) method
2.  class Simple{
3.   public static void main(String args[]){
4.
5.     String s1="Sachin";
6.     String s2="SACHIN";
7.
8.     System.out.println(s1.equals(s2));//false
9.     System.out.println(s1.equalsIgnoreCase(s2));//true
10.  }
11. }
```

Output:false
true

2) By == operator

The = = operator compares references not values.

```
1.  //Example of == operator
2.
3.  class Simple{
4.   public static void main(String args[]){
5.
6.     String s1="Sachin";
7.     String s2="Sachin";
8.     String s3=new String("Sachin");
9.
10.    System.out.println(s1==s2);//true (because both refer to same instance)
11.    System.out.println(s1==s3);//false(because s3 refers to instance created in nonpool)
12.  }
13. }
```

Output:true
false

3) By compareTo() method:

compareTo() method compares values and returns an int which tells if the values compare less

than, equal, or greater than.

Suppose s1 and s2 are two string variables.If:

- **s1 == s2** :0
- **s1 > s2** :positive value
- **s1 < s2** :negative value

```
1. //Example of compareTo() method:
2.
3. class Simple{
4.  public static void main(String args[]){
5.
6.    String s1="Sachin";
7.    String s2="Sachin";
8.    String s3="Ratan";
9.
10.   System.out.println(s1.compareTo(s2));//0
11.   System.out.println(s1.compareTo(s3));//1(because s1>s3)
12.   System.out.println(s3.compareTo(s1));//-1(because s3 < s1 )
13.  }
14. }
```

Output:

```
0
1
-1
```

String Concatenation in Java

Concating strings form a new string i.e. the combination of multiple strings.

There are two ways to concat string objects:

1. By + (string concatenation) operator
2. By concat() method

---

1) By + (string concatenation) operator

String concatenation operator is used to add strings.For Example:

```
1. //Example of string concatenation operator
2.
3. class Simple{
4.  public static void main(String args[]){
5.
6.    String s="Sachin"+" Tendulkar";
7.    System.out.println(s);//Sachin Tendulkar
8.  }
9. }
```

Output:Sachin Tendulkar

The compiler transforms this to:

```
1. String s=(new StringBuilder()).append("Sachin").append(" Tendulkar).toString();
```

String concatenation is implemented through the StringBuilder(or StringBuffer) class and its append method. String concatenation operator produces a new string by appending the second operand onto the end of the first operand. The string concatenation operator can concat not only string but primitive values also. For Example:

```
1. class Simple{
2.  public static void main(String args[]){
3.
4.    String s=50+30+"Sachin"+40+40;
5.    System.out.println(s);//80Sachin4040
6.  }
7. }
```

Output:80Sachin4040

Note: If either operand is a string, the resulting operation will be string concatenation. If both operands are numbers, the operator will perform an addition.

2) By concat() method

concat() method concatenates the specified string to the end of current string.

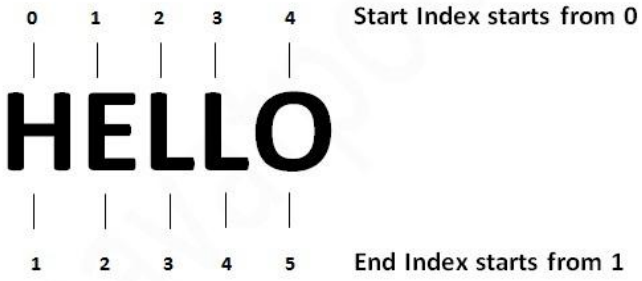
Syntax:public String concat(String another){}

```
1. //Example of concat(String) method
2.
3. class Simple{
```

```
4.    public static void main(String args[]){
5.
6.      String s1="Sachin ";
7.      String s2="Tendulkar";
8.
9.      String s3=s1.concat(s2);
10.
11.    System.out.println(s3);//Sachin Tendulkar
12.   }
13. }
```

Output: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 starts from 0 and endIndex starts from 1 or startIndex is inclusive and endIndex is exclusive.

You can get substring from the given String object by one of the two methods:

1. **public String substring(int startIndex):** This method returns new String object containing the substring of the given string from specified startIndex (inclusive).
2. **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:**starts from index 0(inclusive).

- **endIndex:**starts from index 1(exclusive).

Example of java substring

```
1. //Example of substring() method
2.
3. class Simple{
4.  public static void main(String args[]){
5.
6.    String s="Sachin Tendulkar";
7.    System.out.println(s.substring(6));//Tendulkar
8.    System.out.println(s.substring(0,6));//Sachin
9.  }
10. }
```

Output:Tendulkar
Sachin

Methods of String class in Java

java.lang.String class provides a lot of methods to work on string. By the help of these methods, we can perform operations on string such as trimming, concatenating, converting strings etc.

Let's see the important methods of String class.

Method	Description
1)public boolean equals(Object anObject)	Compares this string to the specified object.
2)public boolean equalsIgnoreCase(String another)	Compares this String to another String, ignoring case.
3)public String concat(String str)	Concatenates the specified string to the end of this string.
4)public int compareTo(String str)	Compares two strings and returns int
5)public int compareToIgnoreCase(String str)	Compares two strings, ignoring case differences.
6)public String substring(int beginIndex)	Returns a new string that is a substring of this

	string.
7)public String substring(int beginIndex,int endIndex)	Returns a new string that is a substring of this string.
8)public String toUpperCase()	Converts all of the characters in this String to upper case
9)public String toLowerCase()	Converts all of the characters in this String to lower case.
10)public String trim()	Returns a copy of the string, with leading and trailing whitespace omitted.
11)public boolean startsWith(String prefix)	Tests if this string starts with the specified prefix.
12)public boolean endsWith(String suffix)	Tests if this string ends with the specified suffix.
13)public char charAt(int index)	Returns the char value at the specified index.
14)public int length()	Returns the length of this string.
15)public String intern()	Returns a canonical representation for the string object.
16)public byte[] getBytes()	Converts string into byte array.
17)public char[] toCharArray()	Converts string into char array.
18)public static String valueOf(int i)	converts the int into String.
19)public static String valueOf(long i)	converts the long into String.
20)public static String valueOf(float i)	converts the float into String.
21)public static String valueOf(double i)	converts the double into String.
22)public static String valueOf(boolean i)	converts the boolean into String.

23)public static String valueOf(char i)	converts the char into String.
24)public static String valueOf(char[] i)	converts the char array into String.
25)public static String valueOf(Object obj)	converts the Object into String.
26)public void replaceAll(String firstString,String secondString)	Changes the firstString with secondString.

First seven methods have already been discussed. Now Let's take the example of other methods:

---

toUpperCase() and toLowerCase() method

```
1.  class Simple{
2.   public static void main(String args[]){
3.
4.     String s="Sachin";
5.     System.out.println(s.toUpperCase());//SACHIN
6.     System.out.println(s.toLowerCase());//sachin
7.     System.out.println(s);//Sachin(no change in original)
8.   }
9.  }
```

Output:SACHIN
sachin
Sachin

---

trim() method

```
1.  class Simple{
2.   public static void main(String args[]){
3.
4.     String s="  Sachin  ";
5.     System.out.println(s);//  Sachin
6.     System.out.println(s.trim());//Sachin
7.   }
8.  }
```

Output: Sachin
Sachin

---

startsWith() and endsWith() method

```
1. class Simple{
2.  public static void main(String args[]){
3.
4.    String s="Sachin";
5.    System.out.println(s.startsWith("Sa"));//true
6.    System.out.println(s.endsWith("n"));//true
7.  }
8. }
```

Output:true
true

---

charAt() method

```
1. class Simple{
2.  public static void main(String args[]){
3.
4.    String s="Sachin";
5.    System.out.println(s.charAt(0));//S
6.    System.out.println(s.charAt(3));//h
7.  }
8. }
```

Output:S
h

---

length() method

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

Output:6

---

intern() method

A pool of strings, initially empty, is maintained privately by the class String.

When the intern method is invoked, if the pool already contains a string equal to this String object as determined by the equals(Object) method, then the string from the pool is returned. Otherwise, this String object is added to the pool and a reference to this String object is returned.

```
1. class Simple{
2.  public static void main(String args[]){
3.
4.    String s=new String("Sachin");
5.    String s2=s.intern();
6.    System.out.println(s2);//Sachin
7.   }
8. }
```

Output:Sachin

StringBuffer class:

The StringBuffer class is used to created mutable (modifiable) string. The StringBuffer class is same as String except it is mutable i.e. it can be changed.

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

Commonly used Constructors of StringBuffer class:

1. **StringBuffer():** creates an empty string buffer with the initial capacity of 16.
2. **StringBuffer(String str):** creates a string buffer with the specified string.
3. **StringBuffer(int capacity):** creates an empty string buffer with the specified capacity as length.

Commonly used methods of StringBuffer class:

1. **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.
2. **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.

3. **public synchronized StringBuffer replace(int startIndex, int endIndex, String str):** is used to replace the string from specified startIndex and endIndex.
4. **public synchronized StringBuffer delete(int startIndex, int endIndex):** is used to delete the string from specified startIndex and endIndex.
5. **public synchronized StringBuffer reverse():** is used to reverse the string.
6. **public int capacity():** is used to return the current capacity.
7. **public void ensureCapacity(int minimumCapacity):** is used to ensure the capacity at least equal to the given minimum.
8. **public char charAt(int index):** is used to return the character at the specified position.
9. **public int length():** is used to return the length of the string i.e. total number of characters.
10. **public String substring(int beginIndex):** is used to return the substring from the specified beginIndex.
11. **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.

simple example of StringBuffer class by append() method

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

```
1. class A{
2. public static void main(String args[]){
3. 
4. StringBuffer sb=new StringBuffer("Hello ");
5. sb.append("Java");//now original string is changed
6. 
7. System.out.println(sb);//prints Hello Java
8. }
9. }
```

Example of insert() method of StringBuffer class

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

```
1. class A{
2. public static void main(String args[]){
3. 
4. StringBuffer sb=new StringBuffer("Hello ");
5. sb.insert(1,"Java");//now original string is changed
6. 
7. System.out.println(sb);//prints HJavaello
8. }
9. }
```

Example of replace() method of StringBuffer class

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

```
1. class A{
2. public static void main(String args[]){
3. 
4. StringBuffer sb=new StringBuffer("Hello");
5. sb.replace(1,3,"Java");
6. 
7. System.out.println(sb);//prints HJavalo
8. }
9. }
```

Example of delete() method of StringBuffer class

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

```
1. class A{
2. public static void main(String args[]){
3. 
4. StringBuffer sb=new StringBuffer("Hello");
5. sb.delete(1,3);
6. 
7. System.out.println(sb);//prints Hlo
8. }
9. }
```

Example of reverse() method of StringBuffer class

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

```
1. class A{
2. public static void main(String args[]){
3. 
4. StringBuffer sb=new StringBuffer("Hello");
5. sb.reverse();
6. 
7. System.out.println(sb);//prints olleH
8. }
9. }
```

Example of capacity() method of StringBuffer class

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 A{
2. public static void main(String args[]){
3. 
4. StringBuffer sb=new StringBuffer();
5. System.out.println(sb.capacity());//default 16
6. 
7. sb.append("Hello");
8. System.out.println(sb.capacity());//now 16
9. 
10. sb.append("java is my favourite language");
11. System.out.println(sb.capacity());//now (16*2)+2=34 i.e (oldcapacity*2)+2
12. }
13. }
```

Example of ensureCapacity() method of StringBuffer class

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.

```
class A{
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

sb.ensureCapacity(10);//now no change
System.out.println(sb.capacity());//now 34

sb.ensureCapacity(50);//now (34*2)+2
System.out.println(sb.capacity());//now 70

}
}
```

StringBuilder class:

The ***StringBuilder*** class is used to create mutable (modifiable) string. The *StringBuilder* class is same as *StringBuffer* class except that it is **non-synchronized.** It is available since JDK1.5.

Commonly used Constructors of StringBuilder class:

1. **StringBuilder():** creates an empty string Builder with the initial capacity of 16.
2. **StringBuilder(String str):** creates a string Builder with the specified string.
3. **StringBuilder(int length):** creates an empty string Builder with the specified capacity as length.

Commonly used methods of StringBuilder class:

1. **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.
2. **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.
3. **public StringBuilder replace(int startIndex, int endIndex, String str):** is used to replace the string from specified startIndex and endIndex.
4. **public StringBuilder delete(int startIndex, int endIndex):** is used to delete the string

from specified startIndex and endIndex.

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

---

simple program of StringBuilder class by append() method

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

```
1. class A{
2. public static void main(String args[]){
3.
4. StringBuilder sb=new StringBuilder("Hello ");
5. sb.append("Java");//now original string is changed
6.
7. System.out.println(sb);//prints Hello Java
8. }
9. }
```

Example of insert() method of StringBuilder class

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

```
1. class A{
2. public static void main(String args[]){
3.
4. StringBuilder sb=new StringBuilder("Hello ");
5. sb.insert(1,"Java");//now original string is changed
6.
7. System.out.println(sb);//prints HJavaello
8. }
9. }
```

Example of replace() method of StringBuilder class

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

```
1. class A{
2. public static void main(String args[]){
3. 
4. StringBuilder sb=new StringBuilder("Hello");
5. sb.replace(1,3,"Java");
6. 
7. System.out.println(sb);//prints HJavalo
8. }
9. }
```

Example of delete() method of StringBuilder class

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

```
1. class A{
2. public static void main(String args[]){
3. 
4. StringBuilder sb=new StringBuilder("Hello");
5. sb.delete(1,3);
6. 
7. System.out.println(sb);//prints Hlo
8. }
9. }
```

Example of reverse() method of StringBuilder class

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

```
1. class A{
2. public static void main(String args[]){
3. 
4. StringBuilder sb=new StringBuilder("Hello");
5. sb.reverse();
6. 
7. System.out.println(sb);//prints olleH
8. }
```

```
9. }
```

Example of capacity() method of StringBuilder class

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 A{
2. public static void main(String args[]){
3. 
4. StringBuilder sb=new StringBuilder();
5. System.out.println(sb.capacity());//default 16
6. 
7. sb.append("Hello");
8. System.out.println(sb.capacity());//now 16
9. 
10. sb.append("java is my favourite language");
11. System.out.println(sb.capacity());//now (16*2)+2=34 i.e (oldcapacity*2)+2
12. }
13. }
```

Example of ensureCapacity() method of StringBuilder class

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

```
12. sb.ensureCapacity(10);//now no change
13. System.out.println(sb.capacity());//now 34
14.
15. sb.ensureCapacity(50);//now (34*2)+2
16. System.out.println(sb.capacity());//now 70
17.
18. }
19. }
```

How to create 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.

Understanding toString() method

If you want to represent any object as a string, **toString() method** comes into existence.

The toString() method returns the string representation of the object.

If you print any object, java compiler internally invokes the toString() method on the object. So overriding the toString() method, returns the desired output, it can be the state of an object etc. depends on your implementation.

Advantage of the toString() method

By overriding the toString() method of the Object class, we can return values of the object, so we don't need to write much code.

Understanding problem without toString() method

Let's see the simple code that prints reference.

```
1.  class Student{
2.   int rollno;
3.   String name;
4.   String city;
5.
6.   Student(int rollno, String name, String city){
7.   this.rollno=rollno;
8.   this.name=name;
9.   this.city=city;
10.  }
11.
12.  public static void main(String args[]){
13.    Student s1=new Student(101,"Sachin","Dehradun");
14.    Student s2=new Student(102,"Vijay","Delhi");
15.
16.    System.out.println(s1);//compiler writes here s1.toString()
17.    System.out.println(s2);//compiler writes here s2.toString()
18.  }
19. }
```

Output:Student@1fee6fc
Student@1eed786

As you can see in the above example, printing s1 and s2 prints the hashcode values of the objects but I want to print the values of these objects. Since java compiler internally calls toString() method, overriding this method will return the specified values. Let's understand it with the example given below:

Example of toString() method

Now let's see the real example of toString() method.

```
1.  class Student{
2.   int rollno;
3.   String name;
4.   String city;
5.
6.   Student(int rollno, String name, String city){
7.   this.rollno=rollno;
8.   this.name=name;
9.   this.city=city;
10.  }
11.
12.  public String toString(){//overriding the toString() method
13.   return rollno+" "+name+" "+city;
14.  }
15.  public static void main(String args[]){
16.    Student s1=new Student(101,"Raj","Dehradun");
17.    Student s2=new Student(102,"Vijay","Delhi");
18.
19.    System.out.println(s1);//compiler writes here s1.toString()
20.    System.out.println(s2);//compiler writes here s2.toString()
21.  }
22. }
```