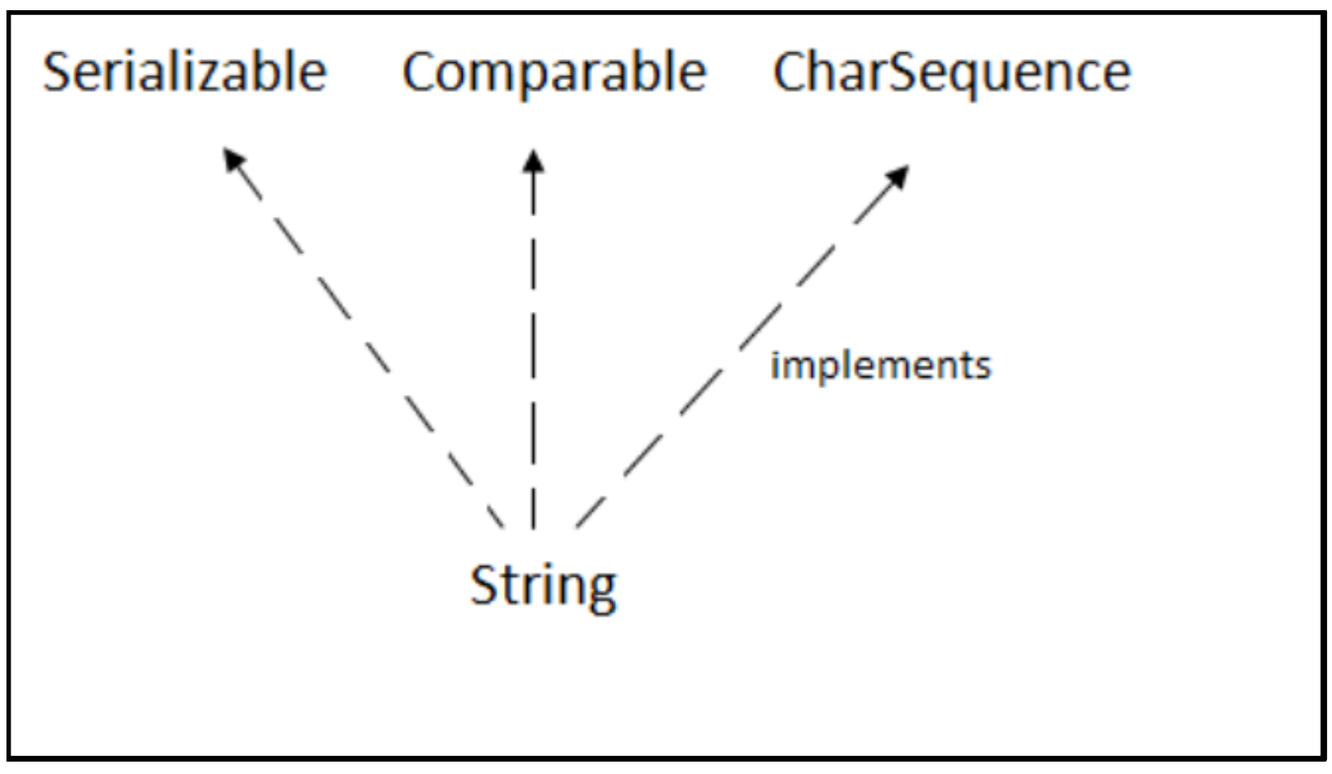
Strings

- 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','t','p','o','i','n','t'};
  String s=new String(ch);
  is same as:
  String s="javatpoint";
  ```

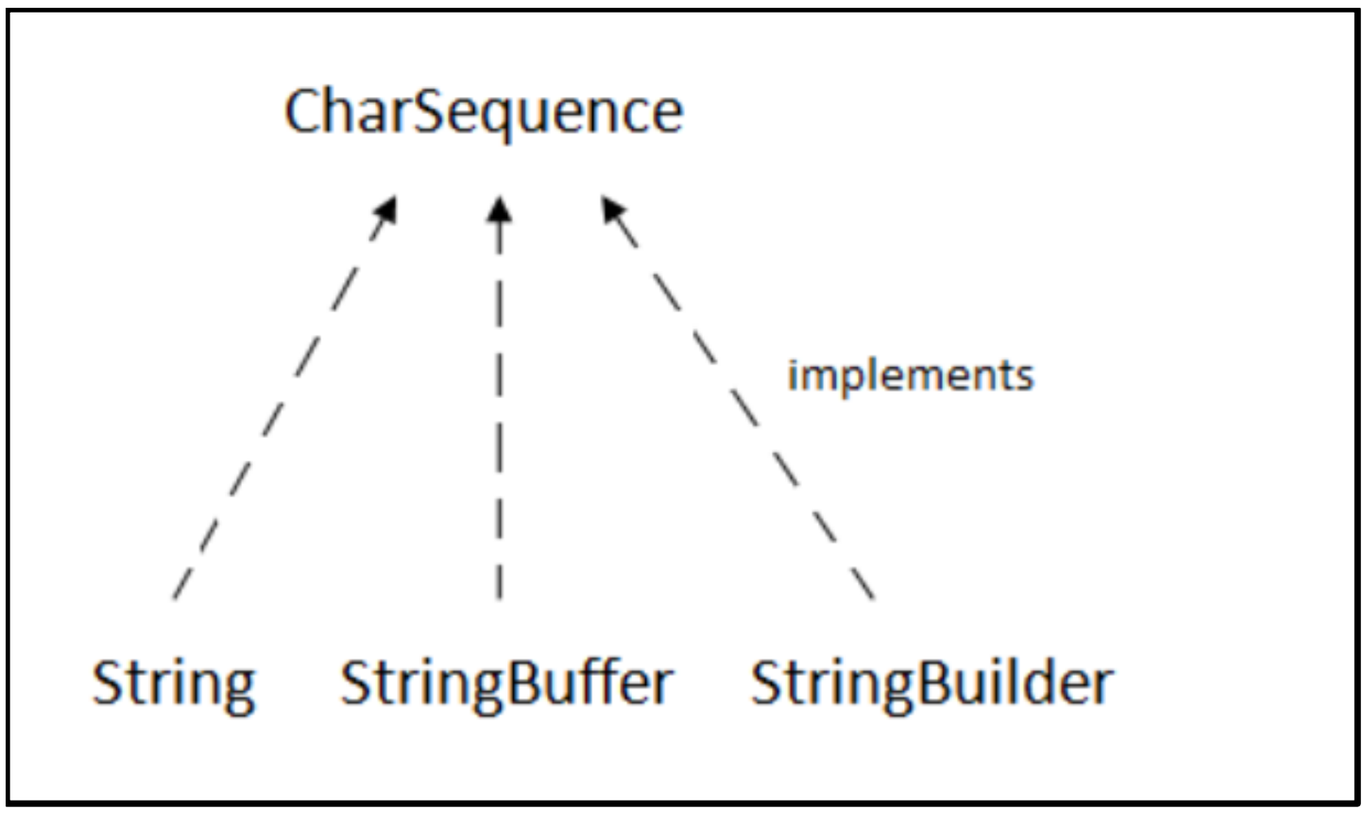
- Java String class provides a lot of methods to perform operations on strings such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.

- The java.lang.String class implements Serializable, Comparable and CharSequence interfaces.



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.

We will discuss immutable string later. Let's first understand what String in Java is and how to create the String object.

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.

How to create a string object?
There are two ways to create String object:

- By string literal
- By new keyword

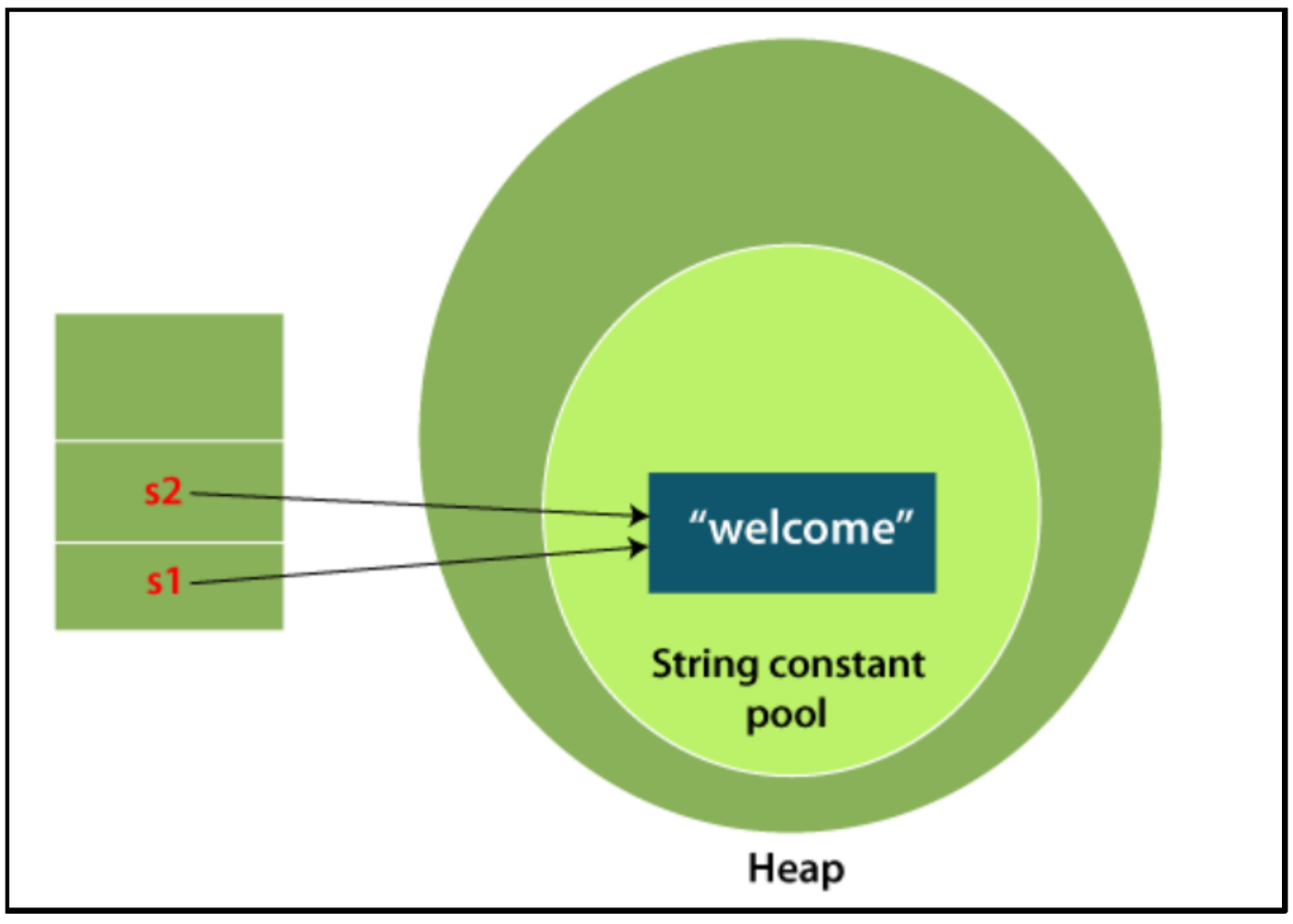
1) String Literal

- 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.
  For example:

  ```
  String s1="Welcome";
  String s2="Welcome";//It doesn't create a new instance
  ```



- In the above example, only one object will be created.
- Firstly, JVM will not find any string object with the value "Welcome" in string constant pool that is why it will create a new object.
- After that it will find the string with the value "Welcome" in the pool, it will not create a new object but will return the reference to the same instance.

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

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 objects 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);
}}
```

- The above code, converts a char array into a String object. And displays the String objects s1, s2, and s3 on console using println() method.

Java String class methods

The java.lang.String class provides many useful methods to perform operations on sequence of char values.

1	**char charAt(int index)**	It returns char value for the particular index
2	**int length()**	It returns string length
3	**static String format(String format, Object... args)**	It returns a formatted string.
4	**static String format(Locale l, String format, Object... args)**	It returns formatted string with given locale.
5	**String substring(int beginIndex)**	It returns substring for given begin index.
6	**String substring(int beginIndex, int endIndex)**	It returns substring for given begin index and end index.
7	**boolean contains(CharSequence s)**	It returns true or false after matching the sequence of char value.
8	**static String join(CharSequence delimiter, CharSequence... elements)**	It returns a joined string.
9	**static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements)**	It returns a joined string.
10	**boolean equals(Object another)**	It checks the equality of string with the given object.
11	**boolean isEmpty()**	It checks if string is empty.
12	**String concat(String str)**	It concatenates the specified string.
13	**String replace(char old, char new)**	It replaces all occurrences of the specified char value.
14	String replace(CharSequence old, CharSequence new)	It replaces all occurrences of the specified CharSequence.
15	static String equalsIgnoreCase(String another)	It compares another string. It doesn't check case.
16	String[] split(String regex)	It returns a split string matching regex.
17	String[] split(String regex, int limit)	It returns a split string matching regex and limit.
18	String intern()	It returns an interned string.
19	int indexOf(int ch)	It returns the specified char value index.
20	int indexOf(int ch, int fromIndex)	It returns the specified char value index starting with given index.

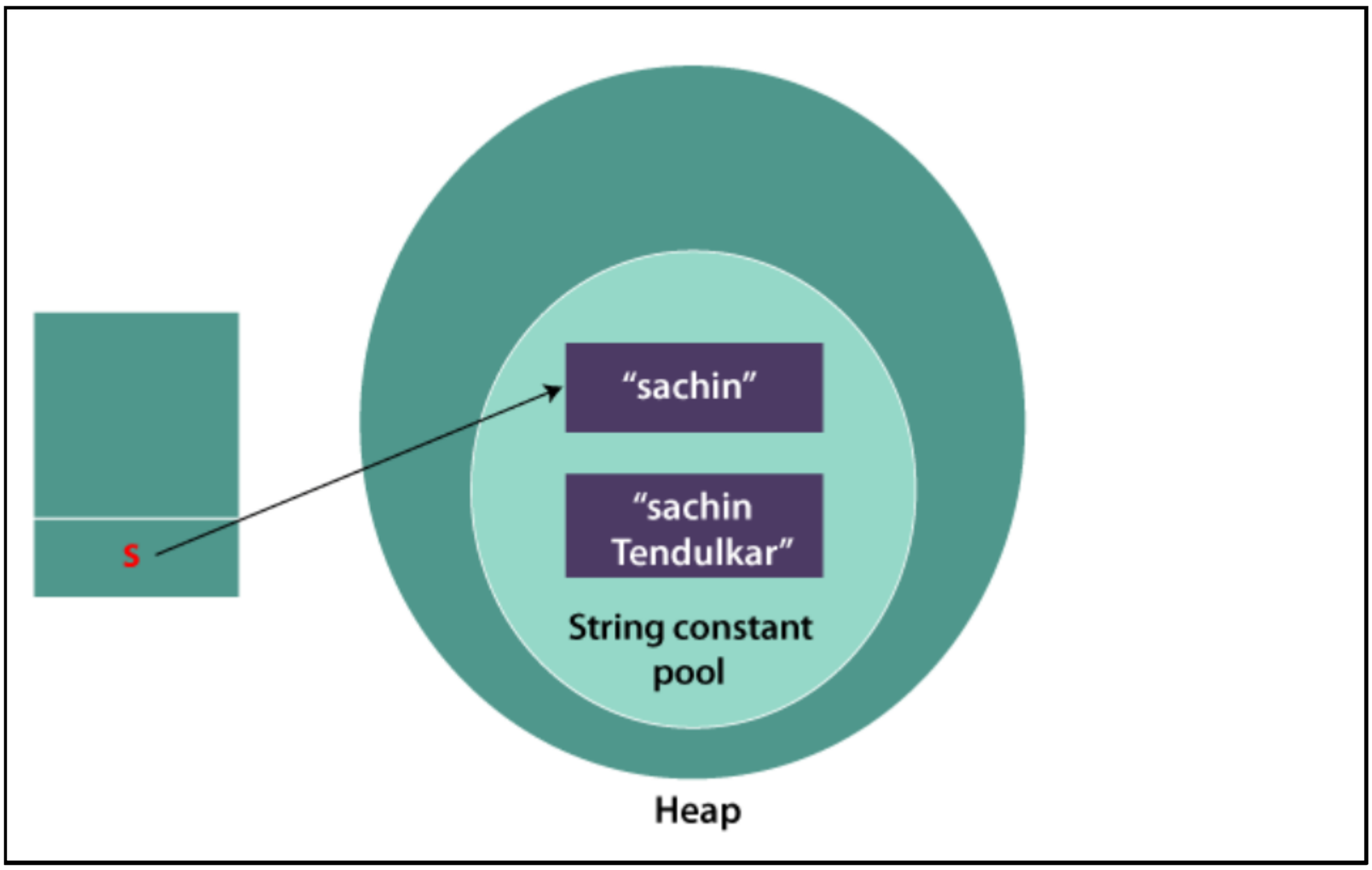
21	int indexOf(String substring)	It returns the specified substring index.
22	int indexOf(String substring, int fromIndex)	It returns the specified substring index starting with given index.
23	String toLowerCase()	It returns a string in lowercase.
24	String toLowerCase(Locale l)	It returns a string in lowercase using specified locale.
25	String toUpperCase()	It returns a string in uppercase.
26	String toUpperCase(Locale l)	It returns a string in uppercase using specified locale.
27	String trim()	It removes beginning and ending spaces of this string.
28	static String valueOf(int value)	It converts given type into string. It is an overloaded method.

Immutable String in Java

- String references are used to store various attributes like username, password, etc.
- 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 concept of immutability by the example given below:

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

- Now it can be understood by the diagram given below. Here Sachin is not changed but a new object is created with Sachin Tendulkar. 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:

Testimmutablestring1.java

```
class Testimmutablestring1{
 public static void main(String args[]){
   String s="Sachin";
   s=s.concat(" Tendulkar");
   System.out.println(s);
 }
}
```

Output:Sachin Tendulkar

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

Why String objects are immutable in Java?

- As Java uses the concept of String literal.
- Suppose there are 5 reference variables, all refer to one object "Sachin".
- If one reference variable changes the value of the object, it will be affected by all the reference variables.

- That is why String objects are immutable in Java.

Why String class is Final in Java?

The reason behind the String class being final is because no one can override the methods of the String class. So that it can provide the same features to the new String objects as well as to the old ones.

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.
- There are three ways to compare String in Java:

    - By Using equals() Method
    - By Using == Operator
    - By compareTo() Method
    -

1) By Using equals() Method

- The String class equals() method compares the original content of the string. It compares values of string for equality. String class provides the following 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

- In the above code, two strings are compared using equals() method of String class. And the result is printed as boolean values, true or false.

Teststringcomparison2.java

```
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
```

- In the above program, the methods of String class are used. The equals() method returns true if String objects are matching and both strings are of same case. equalsIgnoreCase() returns true regardless of cases of strings.

2) By Using == operator

- The == operator compares references not values.

Teststringcomparison3.java

```
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 nonpool)
 }
}
```

Output:

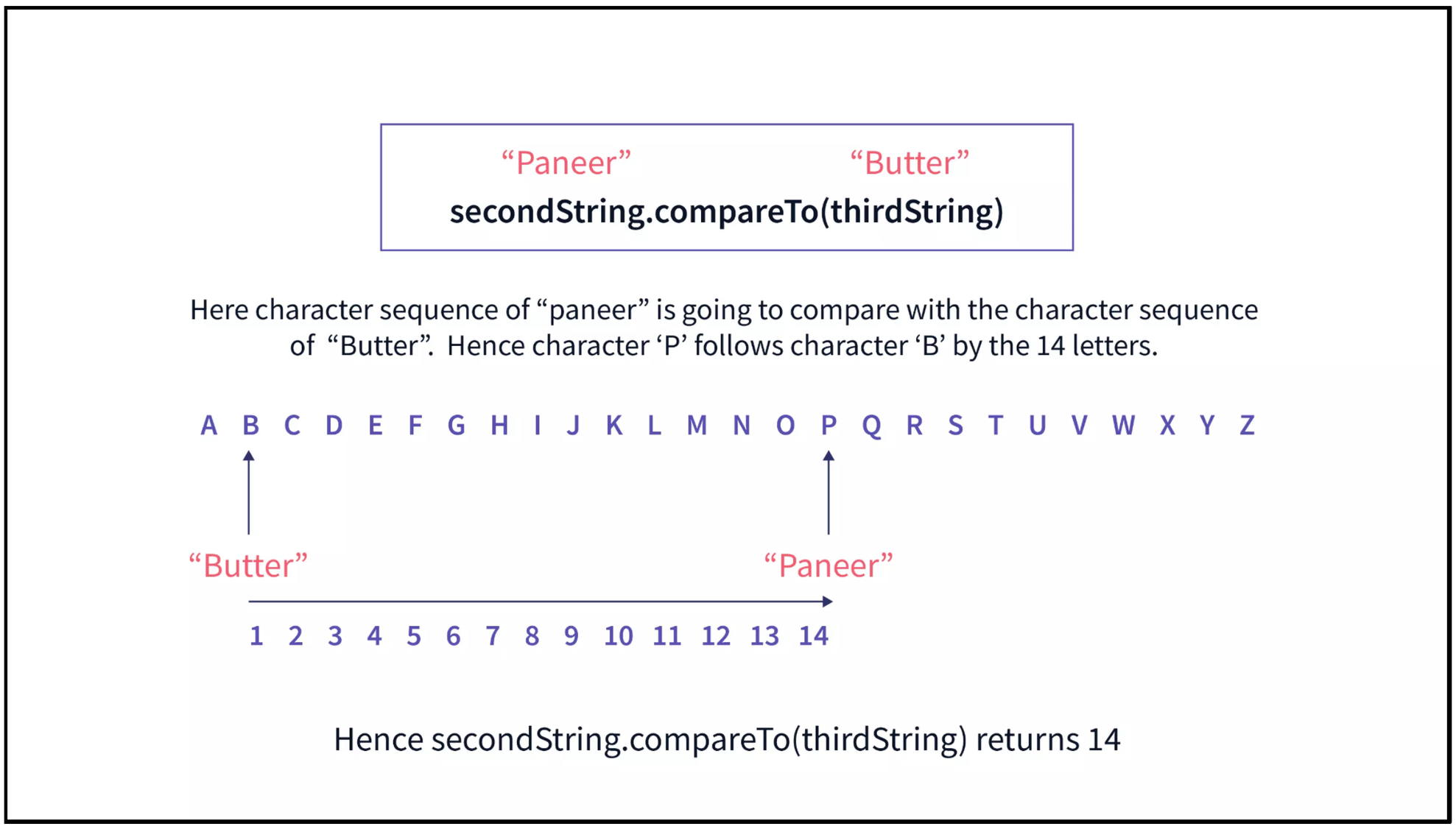
```
true
False
```

3) By Using compareTo() method

The String class 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 objects. If:

s1 == s2 : The method returns 0.
s1 > s2 : The method returns a positive value.
s1 < s2 : The method returns a 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 )
 }
}
```

Test it Now
Output:

```
0
1
-1
```

Java StringBuffer Class

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

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.

Important Constructors of StringBuffer Class

Constructor Description

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

1) StringBuffer Class append() Method

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

StringBufferExample.java

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

Output:

```
Hello Java
```

2) StringBuffer insert() Method

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

StringBufferExample2.java

```
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
}
}
```

Output:

```
HJavaello
```

3) StringBuffer replace() Method
The replace() method replaces the given String from the specified beginIndex and endIndex.

StringBufferExample3.java

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

Output:
HJavalo

4) StringBuffer delete() Method

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

StringBufferExample4.java

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

Output:
Hlo

5) StringBuffer reverse() Method
The reverse() method of the StringBuilder class reverses the current String.

StringBufferExample5.java

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

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

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

StringBuilderExample.java

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

Output:

```
Hello Java
```

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

StringBuilderExample2.java

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

Output:

```
HJavaello
```

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

StringBuilderExample3.java

```
class StringBuilderExample3{
public static void main(String args[]){
```

```
StringBuilder sb=new StringBuilder("Hello");
sb.replace(1,3,"Java");
System.out.println(sb);//prints HJavalo
}
}
```

Output:

```
HJavalo
```

4) StringBuilder delete() method

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

StringBuilderExample4.java

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

Output:

```
Hlo
```

5) StringBuilder reverse() method

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

StringBuilderExample5.java

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

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.
3)	StringBuffer was introduced in Java 1.0	StringBuilder was introduced in Java 1.5