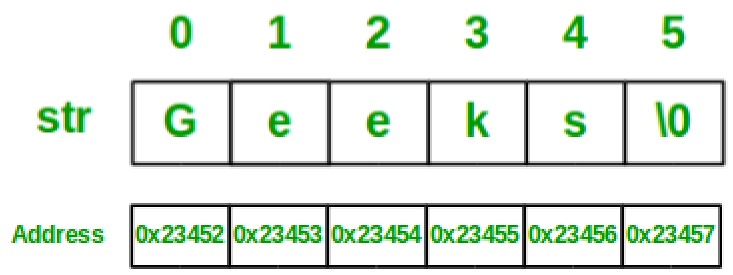
Strings in Java

Strings are defined as an sequence of characters. In java, String are immutable which means a constant and cannot be changed once created.
Below is the basic syntax for declaring a string in **Java programming** language.
Syntax:

> **String** *stringVariableName* = "sequence_of_characters";

Example:

```
String str = "Geeks";
```

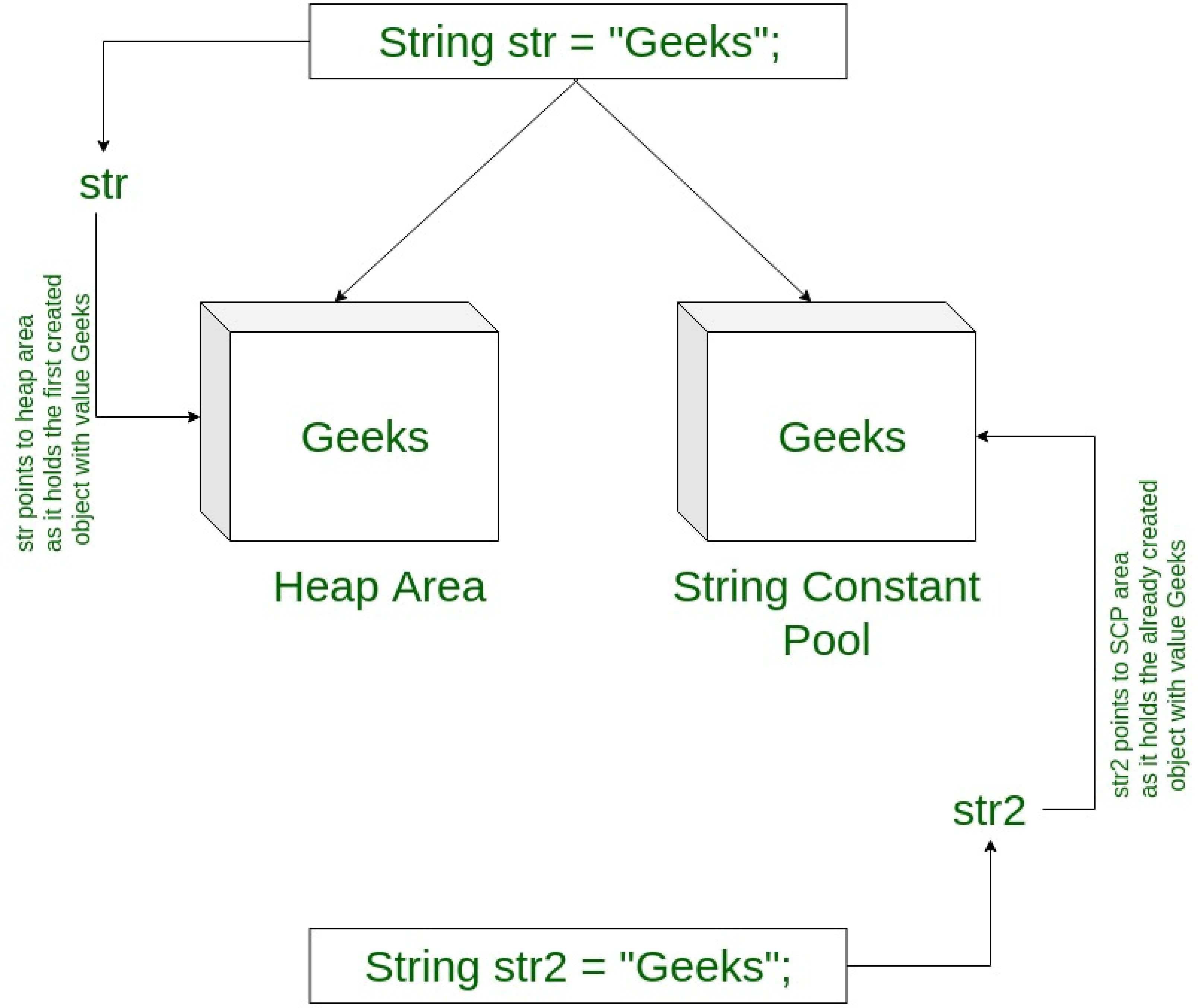


Memory allotment of String

Whenever a String Object is created, two objects are created- one in the Heap Area and one in the String constant pool, and the String object reference always points to the heap area object.

For example:

```
String str = "Geeks";
```


String str = "Geeks";
str
str points to heap area as it holds the first created object with value Geeks
Geeks
Heap Area
Geeks
String Constant Pool
str2 points to SCP area as it holds the already created object with value Geeks
str2
String str2 = "Geeks";

Dash

All

Articles

Videos

Problems

Quiz

Example to illustrate how to declare String:

Java

```
// Java code to illustrate String

import java.io.*;
import java.lang.*;

class Test {
    public static void main(String[] args)
    {
        // Declare String without using new operator
        String s = "GeeksforGeeks";

        // Prints the String.
        System.out.println("String s = " + s);

        // Declare String using new operator
        String s1 = new String("GeeksforGeeks");

        // Prints the String.
        System.out.println("String s1 = " + s1);
    }
}
```

Dash
All
Articles
Videos
Problems
Quiz

Output

```
String s = GeeksforGeeks
String s1 = GeeksforGeeks
```

Interfaces and Classes in Strings in Java

- CharBuffer: This class implements the CharSequence interface. This class is used to allow character buffers to be used in place of CharSequences. An example of such usage is the regular-expression package java.util.regex.
- String: String is a sequence of characters. In java, objects of String are immutable, which means that they are a constant and cannot be changed once created.

Creating a String

- There are two ways to create string in Java:
  - ***String literal***

```
String s = “GeeksforGeeks”;
```

- **Using *new* keyword**

```
String s = new String (“GeeksforGeeks”);
```

- StringBuffer: **StringBuffer** is a peer class of **String** that provides much of the functionality of strings. String represents fixed-length, immutable character sequences while StringBuffer represents growable and writable character sequences.

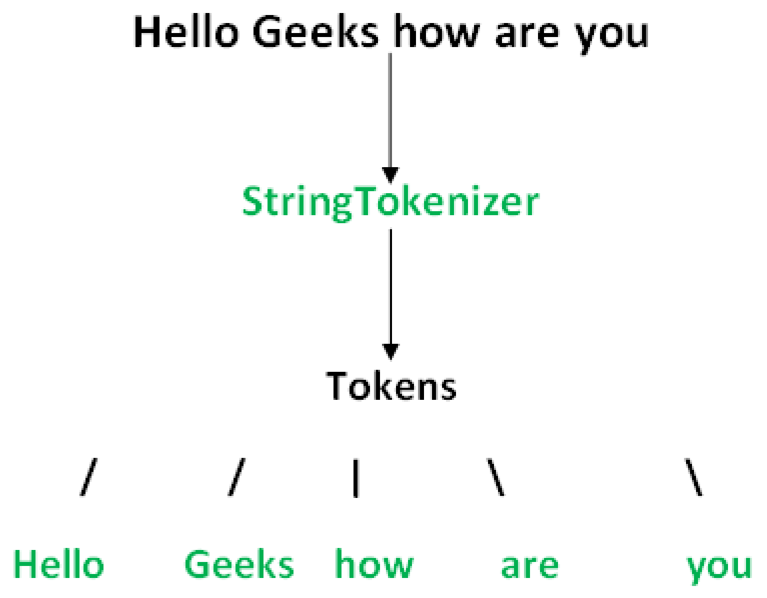
- **Syntax:**

```
StringBuffer s = new StringBuffer("GeeksforGeeks");
```

- StringBuilder: The **StringBuilder** in Java represents a mutable sequence of characters. Since the String Class in Java creates an immutable sequence of characters, the StringBuilder class provides an alternate to String Class as it creates a mutable sequence of characters. **Syntax:**

```
StringBuilder str = new StringBuilder();
str.append("GFG");
```

- StringTokenizer: StringTokenizer class in Java is used to break a string into tokens. **Example:**



A StringTokenizer object internally maintains a current position within the string to be tokenized. Some operations advance this current position past the characters processed. A token is returned by taking a substring of the string that was used to create the StringTokenizer object.

- StringJoiner: StringJoiner is a class in *java.util* package which is used to construct a sequence of characters(strings) separated by a delimiter and optionally starting with a supplied prefix and ending with a supplied suffix. Though this can also be done with the help of the StringBuilder class, StringJoiner provides an easy way to do so without writing much code. **Syntax:**

```
public StringJoiner(CharSequence delimiter)
```

Compare two Strings in Java

String is a sequence of characters. In Java, objects of String are immutable which means they are constant and cannot be changed once created.

Below are 5 ways to compare two Strings in Java:

1. **Using user-defined function :** Define a function to compare values with following conditions :
   1. if (string1 > string2) it returns a **positive value**.
   2. if both the strings are equal lexicographically i.e.(string1 == string2) it returns **0**.
   3. if (string1 < string2) it returns a **negative value**.
2. **Using String.equals():** In Java, string equals() method compares the two given strings based on the data/content of the string. If all the contents of both the strings are the same then it returns true. If all characters do not match, then it returns false. **Syntax:**

```
str1.equals(str2);
```

1. Here str1 and str2 both are the strings that are to be compared. **Examples:**

```
Input 1: GeeksforGeeks
Input 2: Practice
Output: false

Input 1: Geeks
Input 2: Geeks
Output: true

Input 1: geeks
Input 2: Geeks
Output: false
```

1. **Program:**

Java

```
// Java program to Compare two strings
// lexicographically
public class GFG {
    public static void main(String args[])
    {
        String string1 = new String("Geeksforgeeks");
        String string2 = new String("Practice");
        String string3 = new String("Geeks");
        String string4 = new String("Geeks");
        String string5 = new String("geeks");

        // Comparing for String 1 != String 2
        System.out.println("Comparing " + string1 + " and " + string2
                           + " : " + string1.equals(string2));

        // Comparing for String 3 = String 4
        System.out.println("Comparing " + string3 + " and " + string4
                           + " : " + string3.equals(string4));

        // Comparing for String 4 != String 5
        System.out.println("Comparing " + string4 + " and " + string5
                           + " : " + string4.equals(string5));

        // Comparing for String 1 != String 4
        System.out.println("Comparing " + string1 + " and " + string4
                           + " : " + string1.equals(string4));
    }
}
```

Output

```
Comparing Geeksforgeeks and Practice : false
Comparing Geeks and Geeks : true
Comparing Geeks and geeks : false
Comparing Geeksforgeeks and Geeks : false
```

1. **Using String.equalsIgnoreCase() :** The String.equalsIgnoreCase() method compares two strings irrespective of the case (lower or upper) of the string. This method returns true if the argument is not null and the contents of both the Strings are same, ignoring case, else it returns false. **Syntax:**

```
str2.equalsIgnoreCase(str1);
```

1. Here str1 and str2 both are the strings that are to be compared. **Examples:**

```
Input 1: GeeksforGeeks
Input 2: Practice
Output: false
```

Dash
All
Articles
Videos
Problems
Quiz

```
Input 1: Geeks
Input 2: Geeks
Output: true

Input 1: geeks
Input 2: Geeks
Output: true
```

1. **Program:**

Java

```java
// Java program to Compare two strings
// lexicographically
public class GFG {
    public static void main(String args[])
    {
        String string1 = new String("Geeksforgeeks");
        String string2 = new String("Practice");
        String string3 = new String("Geeks");
        String string4 = new String("Geeks");
        String string5 = new String("geeks");

        // Comparing for String 1 != String 2
        System.out.println("Comparing " + string1 + " and " + string2
                           + " : " + string1.equalsIgnoreCase(string2));
        // Comparing for String 3 = String 4
        System.out.println("Comparing " + string3 + " and " + string4
                           + " : " + string3.equalsIgnoreCase(string4));

        // Comparing for String 4 = String 5
        System.out.println("Comparing " + string4 + " and " + string5
                           + " : " + string4.equalsIgnoreCase(string5));

        // Comparing for String 1 != String 4
        System.out.println("Comparing " + string1 + " and " + string4
                           + " : " + string1.equalsIgnoreCase(string4));
    }
}
```

Output

```
Comparing Geeksforgeeks and Practice : false
Comparing Geeks and Geeks : true
Comparing Geeks and geeks : true
Comparing Geeksforgeeks and Geeks : false
```

1. **Using Objects.equals() :** Object.equals(Object a, Object b) method returns true if the arguments are equal to each other and false otherwise. Consequently, if both arguments are null, true is returned and if exactly one argument is null, false is returned. Otherwise, equality is determined by using the equals() method of the first argument. **Syntax:**

```
public static boolean equals(Object a, Object b)
```

1. Here a and b both are the string objects which are to be compared. **Examples:**

```
Input 1: GeeksforGeeks
Input 2: Practice
Output: false

Input 1: Geeks
Input 2: Geeks
Output: true

Input 1: null
Input 2: null
Output: true
```

1. **Program:**

Java

```java
// Java program to Compare two strings
// lexicographically

import java.util.*;

public class GFG {
    public static void main(String args[])
    {
        String string1 = new String("Geeksforgeeks");
```

Dash
All
Articles
Videos
Problems
Quiz

```
        String string2 = new String("Geeks");
        String string3 = new String("Geeks");
        String string4 = null;
        String string5 = null;

        // Comparing for String 1 != String 2
        System.out.println("Comparing " + string1 + " and " + string2
                           + " : " + Objects.equals(string1, string2));

        // Comparing for String 2 = String 3
        System.out.println("Comparing " + string2 + " and " + string3
                           + " : " + Objects.equals(string2, string3));

        // Comparing for String 1 != String 4
        System.out.println("Comparing " + string1 + " and " + string4
                           + " : " + Objects.equals(string1, string4));

        // Comparing for String 4 = String 5
        System.out.println("Comparing " + string4 + " and " + string5
                           + " : " + Objects.equals(string4, string5));
    }
}
```

Output

```
Comparing Geeksforgeeks and Geeks : false
Comparing Geeks and Geeks : true
```

Dash
All
Articles
Videos
Problems
Quiz

```
Comparing Geeksforgeeks and null : false
Comparing null and null : true
```

1. **Using String.compareTo() : Syntax:**

```
int str1.compareTo(String str2)
```

1. **Working:** It compares and returns the following values as follows:
    1. if (string1 > string2), it returns a **positive value**.
    2. if both the strings are equal lexicographically i.e.(string1 == string2), it returns **0**.
    3. if (string1 < string2), it returns a **negative value**.

Why not to use == for comparison of Strings?

In general both **equals()** and "==" operator in Java are used to compare objects to check equality but here are some of the differences between the two:

- Main difference between .equals() method and == operator is that one is a method and other is a operator.
- One can use == operators for reference comparison **(address comparison)** and .equals() method for **content comparison**.
    - Both s1 and s2 refers to different objects.
    - When one uses == operator for s1 and s2 comparison then the result is false as both have different addresses in memory.
    - Using equals, the result is true because it's only comparing the values given in s1 and s2.

Marked as Read

Dash

All

Articles

Videos

Problems

Quiz

«

»

Report An Issue

If you are facing any issue on this page. Please let us know.