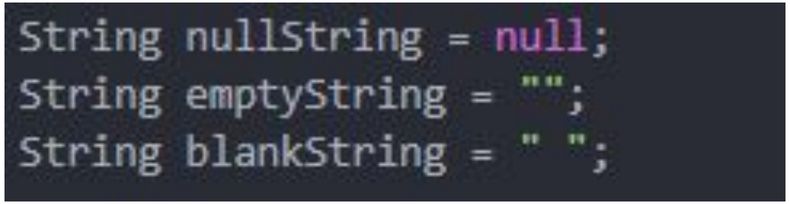
String Manipulations

-Prof. Anamika Dhawan

Handle Null and Empty Strings

In Java, there is a distinct difference between null, empty, and blank Strings.

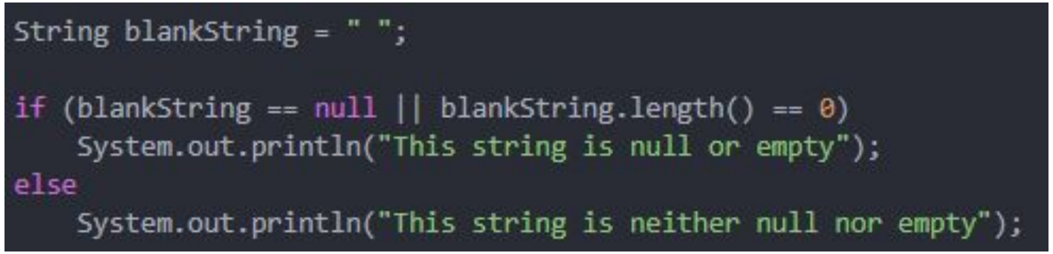
- An empty string is a String object with an assigned value, but its length is equal to zero.
- A null string has no value at all.
- A blank String contains only whitespaces, are is neither empty nor null, since it does have an assigned value, and isn't of 0 length.



```java
String nullString = null;
String emptyString = "";
String blankString = " ";
```

Using the Length of the String

As mentioned before, a string is empty if its length is equal to zero. We will be using the `length()` method, which returns the total number of characters in our string.



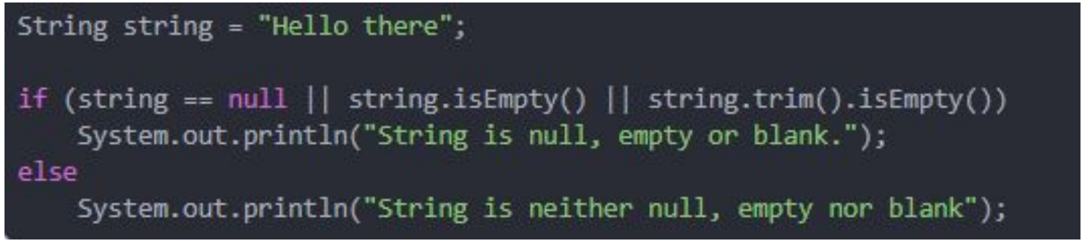
```
String blankString = " ";

if (blankString == null || blankString.length() == 0)
    System.out.println("This string is null or empty");
else
    System.out.println("This string is neither null nor empty");
```

The String is *blank*, so it's obviously neither `null` nor empty. Now, based *just on the length*, we can't really differentiate between Strings that only contain whitespaces or any other character, since a whitespace is a `Character`.

Note: It's important to do the `null`-check *first*, since the short-circuit OR operator `||` will break immediately on the first `true` condition. If the string, in fact, is `null`, all other conditions *before it* will throw a `NullPointerException`

Using the isEmpty() Method

The `isEmpty()` method returns `true` or `false` depending on whether or not our string contains any text. It's easily chainable with a `string == null` check, and can even differentiate between *blank* and *empty* strings:

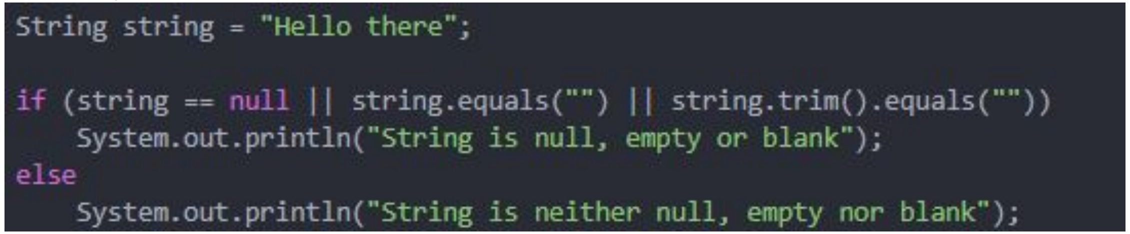


```java
String string = "Hello there";

if (string == null || string.isEmpty() || string.trim().isEmpty())
    System.out.println("String is null, empty or blank.");
else
    System.out.println("String is neither null, empty nor blank");
```

The `trim()` method removes all whitespaces to the left and right of a String, and returns the new sequence. If the String is blank, after removing all whitespaces, it'll be *empty*, so `isEmpty()` will return `true`.

Using the *equals()* Method

The `equals()` method compares the two given strings based on their content and returns `true` if they're equal or `false` if they are not:



```
String string = "Hello there";

if (string == null || string.equals("") || string.trim().equals(""))
    System.out.println("String is null, empty or blank");
else
    System.out.println("String is neither null, empty nor blank");
```

In much the same fashion as the before, if the trimmed string is `""`, it was either empty from the get-go, or was a blank string with `0..n` whitespaces

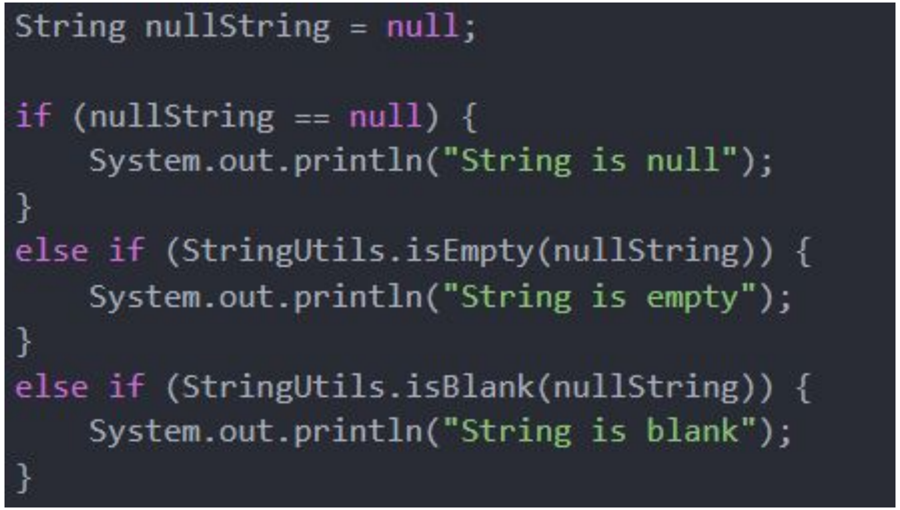
Using the *StringUtils* Class

The Apache Commons is a popular Java library that provides further functionality. `StringUtils` is one of the classes that Apache Commons offers. This class contains methods used to work with `Strings`, similar to the `java.lang.String`.

Dependency required:

```
<dependency>
   <groupId>org.apache.commons</groupId>
   <artifactId>commons-lang3</artifactId>
   <version>3.11</version>
</dependency>
```

One of the key differences between `StingUtils` and `String` methods is that all methods from the `StringUtils` class are null-safe. It additionally provides a few methods that we can leverage for this, including `StringUtils.isEmpty()` and `StringUtils.isBlank()`



```
String nullString = null;

if (nullString == null) {
    System.out.println("String is null");
}
else if (StringUtils.isEmpty(nullString)) {
    System.out.println("String is empty");
}
else if (StringUtils.isBlank(nullString)) {
    System.out.println("String is blank");
}
```

String Comparison

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.

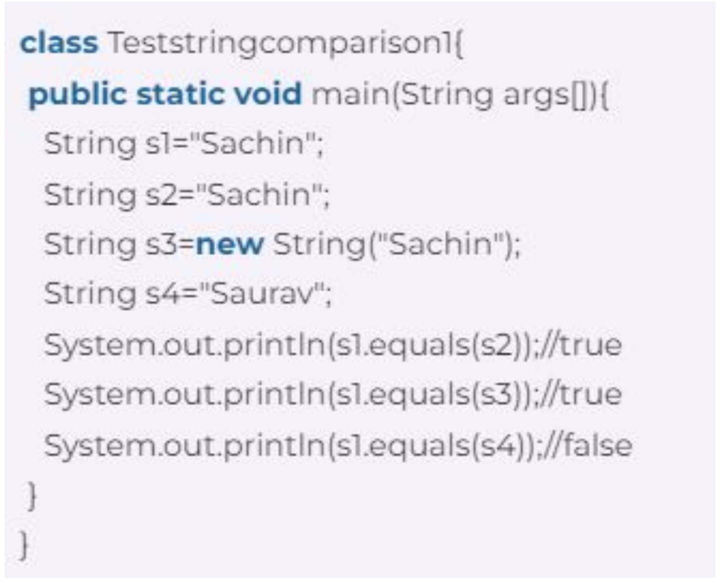
There are three ways to compare String in Java:

1. By Using equals() Method
2. By Using == Operator
3. By compareTo() Method

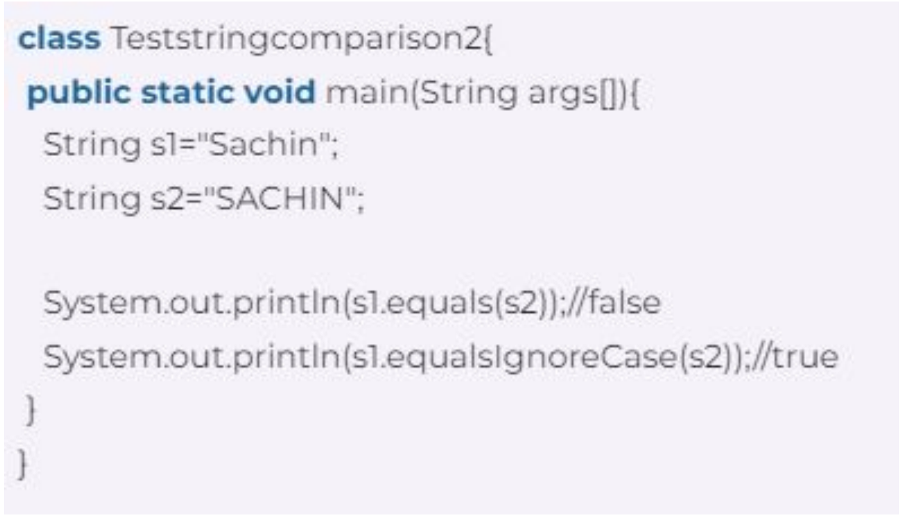
1) By Using equals() Method

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.



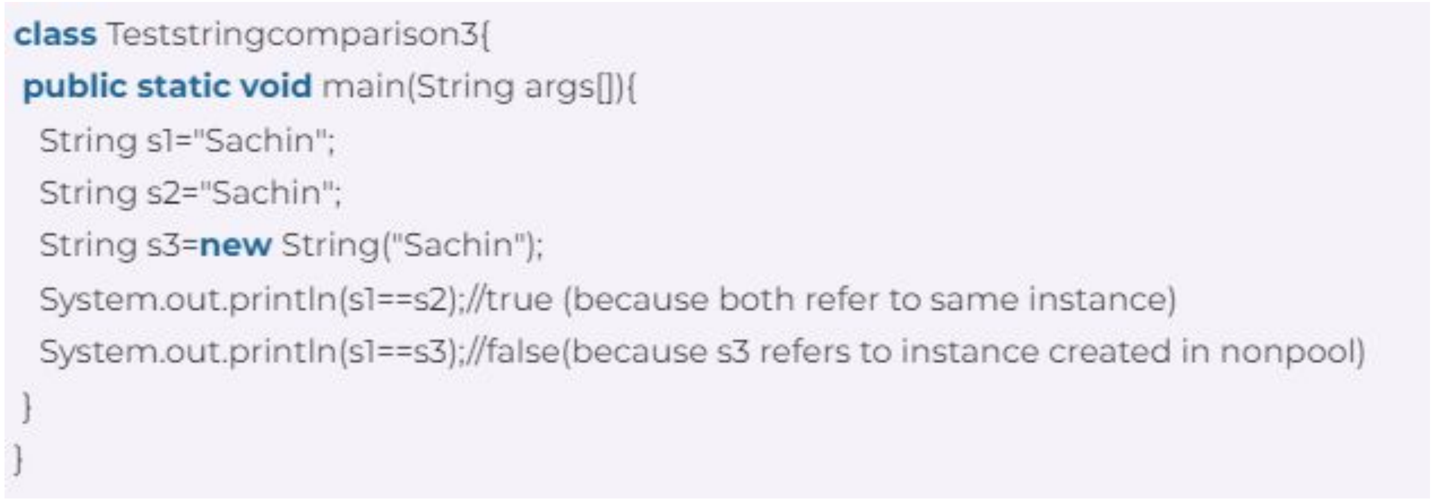
```
class Teststringcomparison1{
 public static void main(String args[]){
   String s1="Sachin";
   String s2="Sachin";
   String s3=new String("Sachin");
   String s4="Saurav";
   System.out.println(s1.equals(s2));//true
   System.out.println(s1.equals(s3));//true
   System.out.println(s1.equals(s4));//false
 }
}
```



```
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
 }
}
```

2) By Using == operator

The == operator compares references not values.



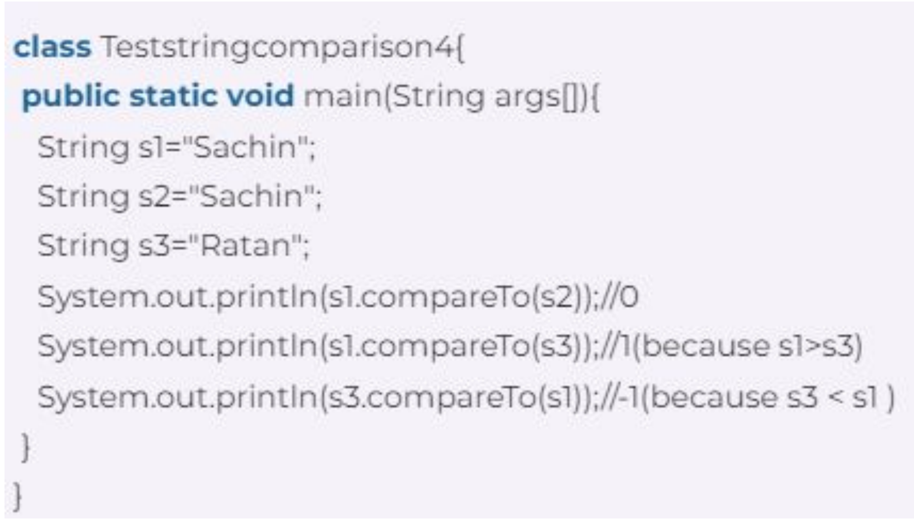
```
class Teststringcomparison3{
 public static void main(String args[]){
   String s1="Sachin";
   String s2="Sachin";
   String s3=new String("Sachin");
   System.out.println(s1==s2);//true (because both refer to same instance)
   System.out.println(s1==s3);//false(because s3 refers to instance created in nonpool)
 }
}
```

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:

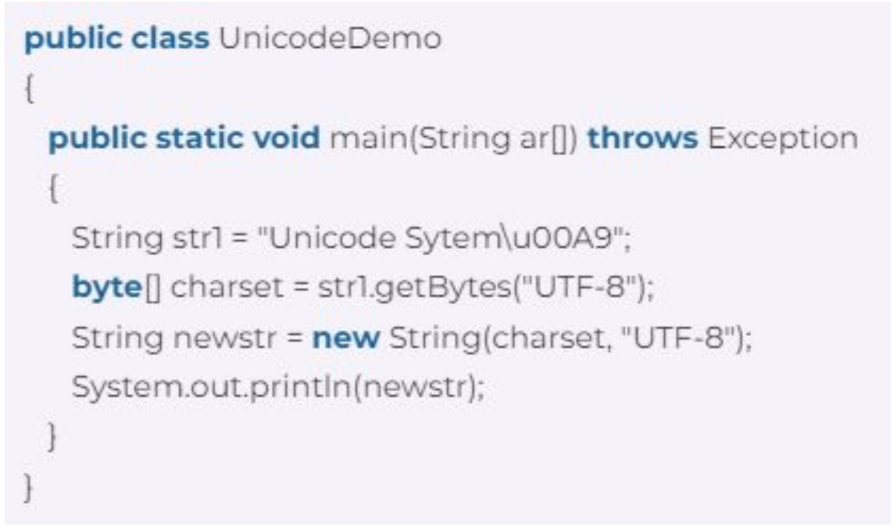
- s1 == s2 : The method returns 0.
- s1 > s2 : The method returns a positive value.
- s1 < s2 : The method returns a negative value.



```
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 )
 }
}
```

Unicode and Character Encoding

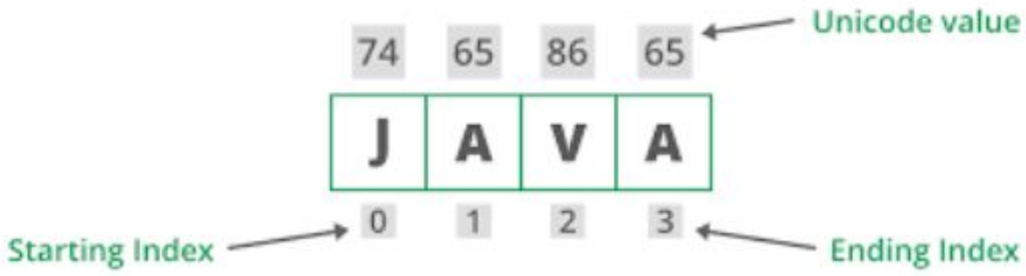
- Unicode system is an international character encoding technique that can represent most of the languages around the world.
- Unicode System is established by Unicode Consortium.
- Hexadecimal values are used to represent Unicode characters.
- There are multiple Unicode Transformation Formats:
    - UTF-8: It represents 8-bits (1 byte) long character encoding.
    - UTF-16: It represents 16-bits (2 bytes) long character encoding
    - UTF-32: It represents 32-bits (4 bytes) long character encoding.
- To access a Unicode character the format starts with an escape sequence \u followed by 4 digits hexadecimal value.
- A Unicode character has a range of possible values starting from \u0000 to \uFFFF.
- Some of the Unicode characters are
  \u00A9 represent the copyright symbol - ©
  \u0394 represent the capital Greek letter delta - Δ
  \u0022 represent a double quote - "



```java
public class UnicodeDemo
{
  public static void main(String ar[]) throws Exception
  {
    String str1 = "Unicode Sytem\u00A9";
    byte[] charset = str1.getBytes("UTF-8");
    String newstr = new String(charset, "UTF-8");
    System.out.println(newstr);
  }
}
```

Unicode in java is known as the 16-bit character programming standard and can easily represent each and all characters in the world of recognized languages.

The main purpose of Unicode is always to integrate different language programming patterns. So that it does not create any confusion in computer systems using fewer programming methods like EBCDIC, ASCII, etc.



ASCII in JAVA

ASCII (American Standard Code for Information Interchange) is a character encoding system used to represent text in computers and other devices. It assigns a unique numerical value to each character, including letters, numbers, punctuation marks, and other symbols. In the ASCII system, each character is represented by a 7-bit binary code.

It can only represent a maximum of 128 characters because it is a seven-bit code**. There are 95 printable characters now defined by it, including 26 upper cases (A to Z), 26 lower cases, 10 numbers (0 to 9), and 33 special characters such as mathematical symbols, punctuation marks, and space characters.

Split Strings

The string split() method breaks a given string around matches of the given regular expression. After splitting against the given regular expression, this method returns a string array.
Following are the two variants of the split() method in Java:

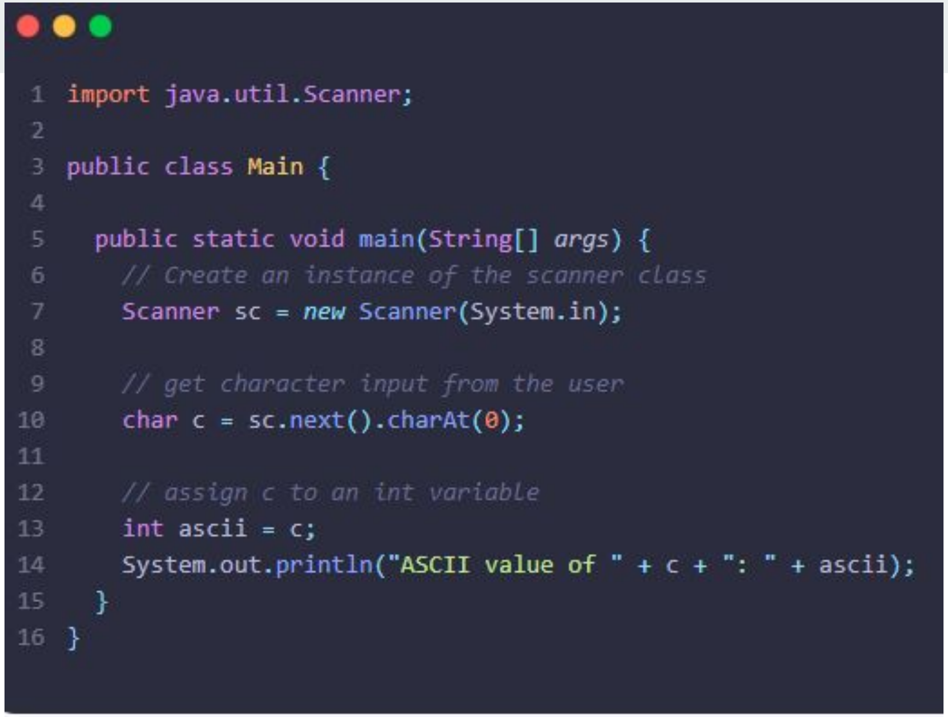
1. Public String [] split (String regex, int limit)
Parameters
regex – a delimiting regular expression
Limit (Optional) – the resulting threshold
Returns
An array of strings is computed by splitting the given string.

Exception Thrown
PatternSyntaxException – if the provided regular expression's syntax is invalid.

ASCII (American Standard Code for Information Interchange) is a system that assigns unique numerical values to characters so that computers can store and manipulate them. The assigned numerical values are represented in binary form for use by electrical equipment, since it is not possible to use or store the original character form.

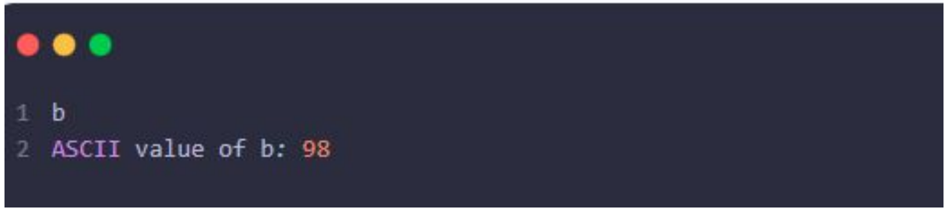
There are four methods available in Java to print the ASCII value. Each method is briefly explained below, along with an example of how to implement it:

- Brute force technique
- Type-casting method
- Format specifier method
- Byte class method [Most Optimal]

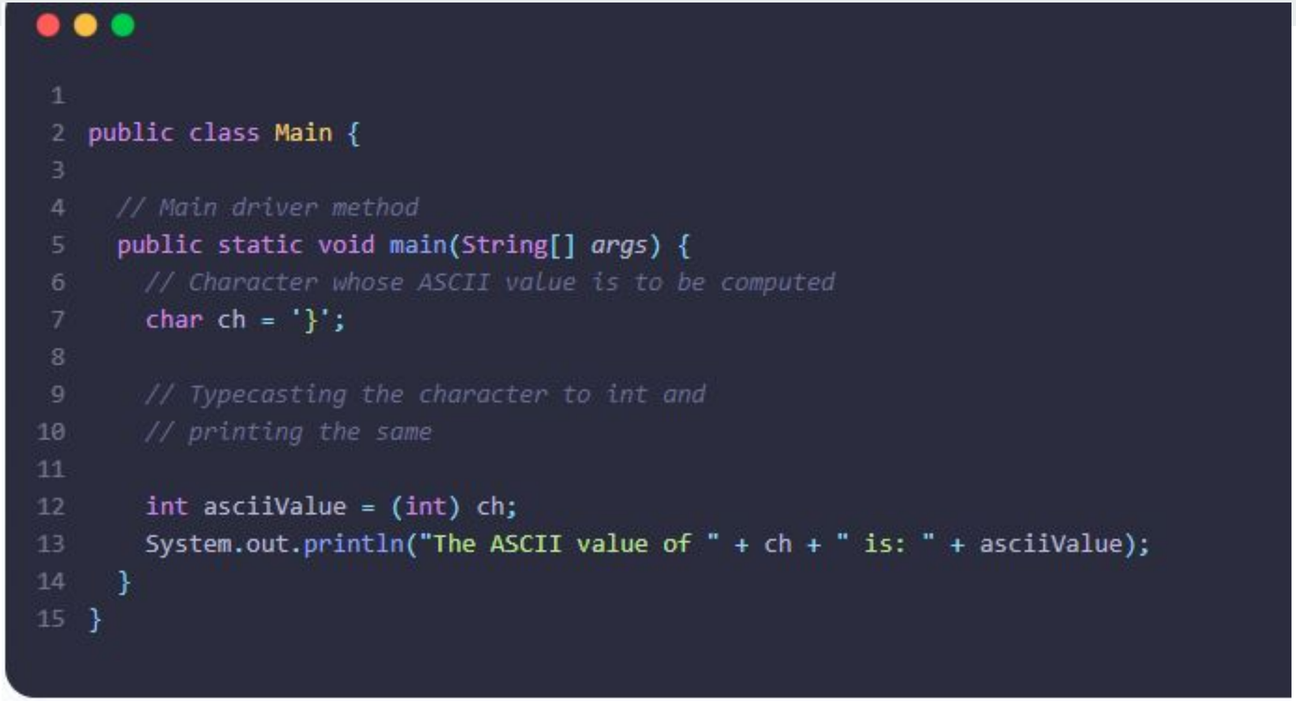


```java
import java.util.Scanner;

public class Main {

  public static void main(String[] args) {
    // Create an instance of the scanner class
    Scanner sc = new Scanner(System.in);

    // get character input from the user
    char c = sc.next().charAt(0);

    // assign c to an int variable
    int ascii = c;
    System.out.println("ASCII value of " + c + ": " + ascii);
  }
}
```

utput —

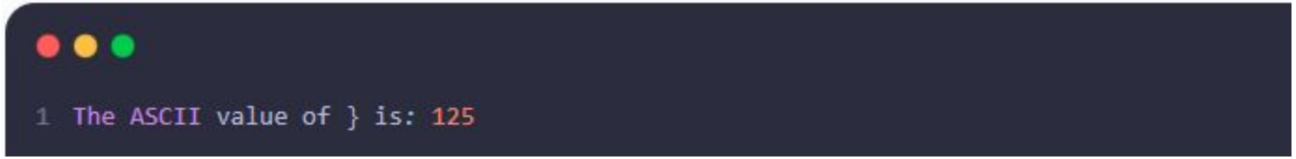


```
b
ASCII value of b: 98
```

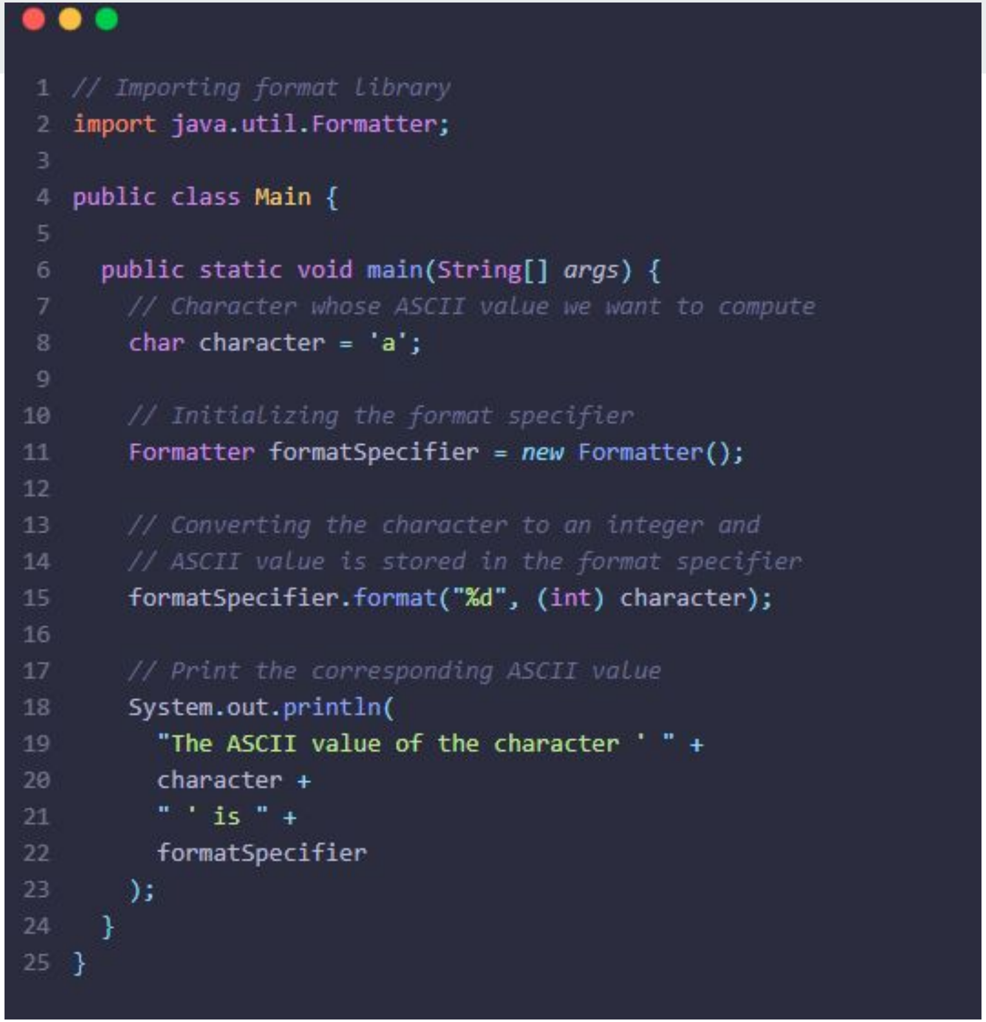


```java

public class Main {

  // Main driver method
  public static void main(String[] args) {
    // Character whose ASCII value is to be computed
    char ch = '}';

    // Typecasting the character to int and
    // printing the same

    int asciiValue = (int) ch;
    System.out.println("The ASCII value of " + ch + " is: " + asciiValue);
  }
}
```

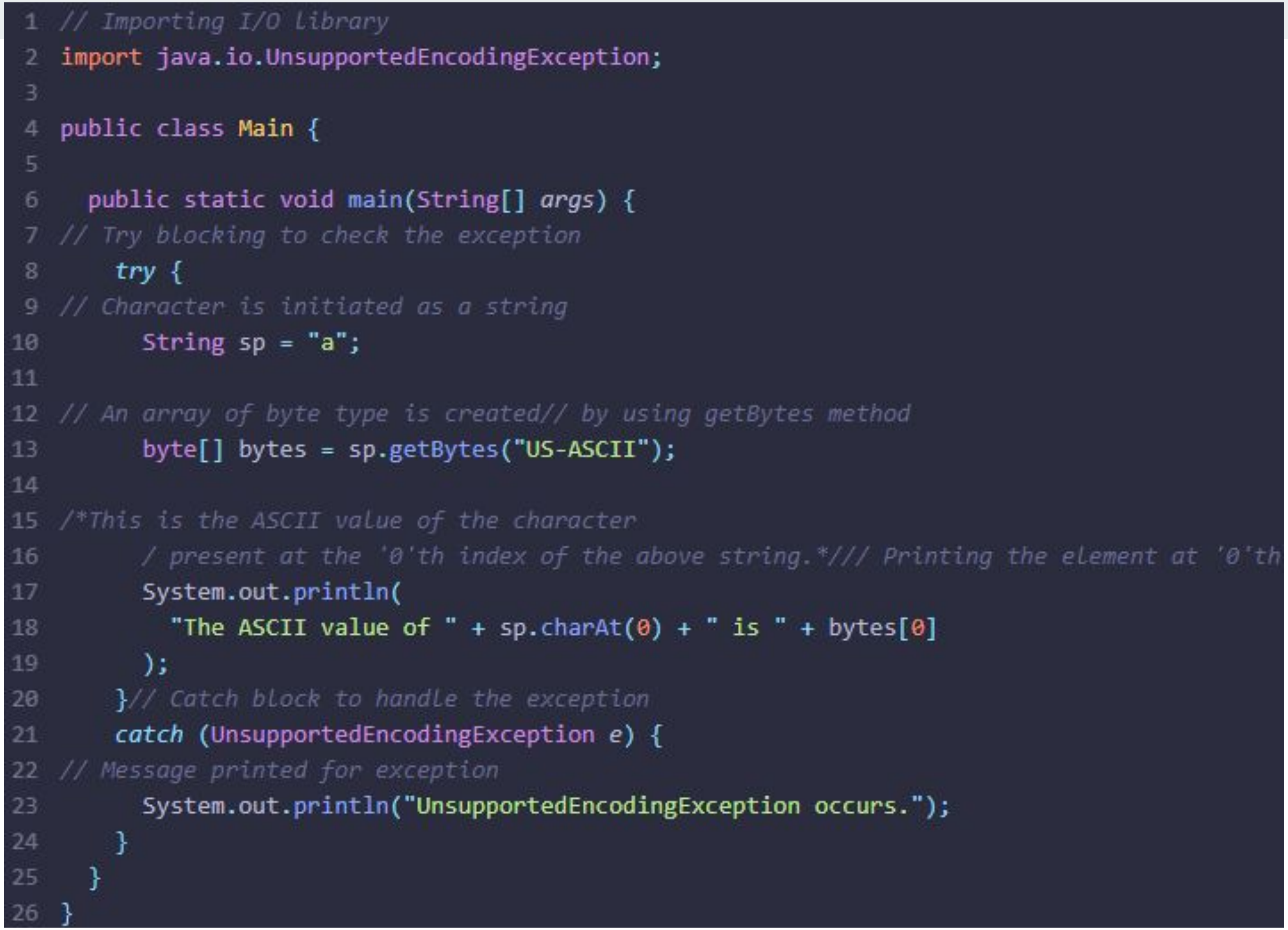
Output —



```
The ASCII value of } is: 125
```

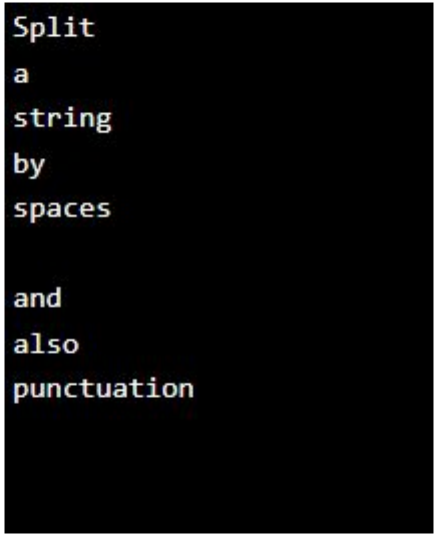


```java
// Importing format library
import java.util.Formatter;

public class Main {

  public static void main(String[] args) {
    // Character whose ASCII value we want to compute
    char character = 'a';

    // Initializing the format specifier
    Formatter formatSpecifier = new Formatter();

    // Converting the character to an integer and
    // ASCII value is stored in the format specifier
    formatSpecifier.format("%d", (int) character);

    // Print the corresponding ASCII value
    System.out.println(
      "The ASCII value of the character ' " +
      character +
      " ' is " +
      formatSpecifier
    );
  }
}
```



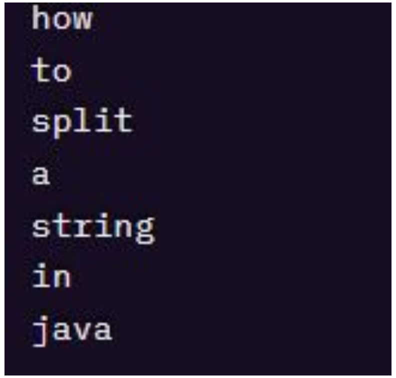
```java
// Importing I/O library
import java.io.UnsupportedEncodingException;

public class Main {

  public static void main(String[] args) {
// Try blocking to check the exception
    try {
// Character is initiated as a string
      String sp = "a";

// An array of byte type is created// by using getBytes method
      byte[] bytes = sp.getBytes("US-ASCII");

/*This is the ASCII value of the character
      / present at the '0'th index of the above string.*/// Printing the element at '0'th
      System.out.println(
        "The ASCII value of " + sp.charAt(0) + " is " + bytes[0]
      );
    }// Catch block to handle the exception
    catch (UnsupportedEncodingException e) {
// Message printed for exception
      System.out.println("UnsupportedEncodingException occurs.");
    }
  }
}
```

```
public class Main {
 public static void main(String[] args) {
  String myStr = "Split a string by spaces, and also punctuation.";
  String regex = "[,\\.\\s]";
  String[] myArray = myStr.split(regex);
  for (String s : myArray) {
		System.out.println(s);
  }
 }
}
```



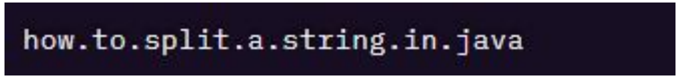
```
Split
a
string
by
spaces

and
also
punctuation
```

```
public class Main {

    public static void main(String[] arg) {
        String str = "how:to:split:a:string:in:java";
        String[] arrOfStr = str.split(":");
        for (String a : arrOfStr) {
            System.out.println(a);
        }
    }
}
```



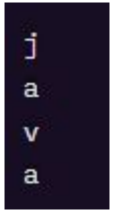
```
how
to
split
a
string
in
java
```

```
public class Main {

  public static void main(String[] arg) {
    String str = "how.to.split.a.string.in.java";
    String[] arrOfStr = str.split("z");
    for (String a : arrOfStr) {
      System.out.println(a);
    }
  }
}
```



```
how.to.split.a.string.in.java
```

```
public class Main {

   public static void main(String[] arg) {
      String str = "java";
      String[] arrOfStr = str.split("");
      for (String a : arrOfStr) {
         System.out.println(a);
      }
   }
}
```



```
j
a
v
a
```

Let the string that is to be split is –
helloss@for@helloss

Regex	Limit	Result
@	2	{“helloss”, ”for@helloss”}
@	5	{“helloss”, ”for”, ”helloss”}
@	-2	{“helloss”, ”for”, ”helloss”}
s	5	{“hello”, ””, “@for@hello”, “”, “”}
s	-2	{“hello”, ” “, ” “, “@for@hello”, “”, “”}
s	0	{“hello”, ””, ”@for@hello”}

```
// Java program to demonstrate working of split()
// using regular expressions

public class GFG {

    public static void main(String args[])
    {
        String str = "word1, word2 word3@word4?word5.word6";
        String[] arrOfStr = str.split("[, ?.@]+");

        for (String a : arrOfStr)
            System.out.println(a);
    }
}
```

Convert Other Data Types to Strings

The Object class is the superclass of all other classes in Java. It implies that a reference variable of Object type can refer to an instance of any other class. Every class in Java inherits methods of the Object class implicitly.

In Java, you can convert an object to a string using the toString() method, which is a method of the Object class. Similarly, there are other methods in the Object class that can be used for converting an instance to a string as listedin the table:

Methods	Description
String toString()	Returns the string representation that describes the object.
String valueOf()	This method belongs to a String class and is a static method.
+ operator	It is used for concatenating any type of value with a string and returns the output as a string.
join()	It allows you to an array of objects into a single string. This method is part of the String class and it takes two parameters: a delimiter and an array of objects.

Java provides several methods for converting an object to a string, including the toString() method, String.valueOf() method, + operator, and join() method.

The toString() and String.valueOf() methods are part of the Object and String classes, respectively, and they return a string that represents the object.

The + operator is used for concatenating any type of value with a string and returns the output as a string.

Finally, the join() method allows you to join an array of objects into a single string.

When converting an object to a string, it is essential to consider the time complexity of the method used. The + operator has a time complexity of O(1), while the join() method has a time complexity of O(n*k).

String

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

Creating a String
There are two ways to create string in Java:

1. String literal
String s = “GeeksforGeeks”;

2. Using new keyword
String s = new String (“GeeksforGeeks”);

StringBuilder

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 alternative to String Class, as it creates a mutable sequence of characters.

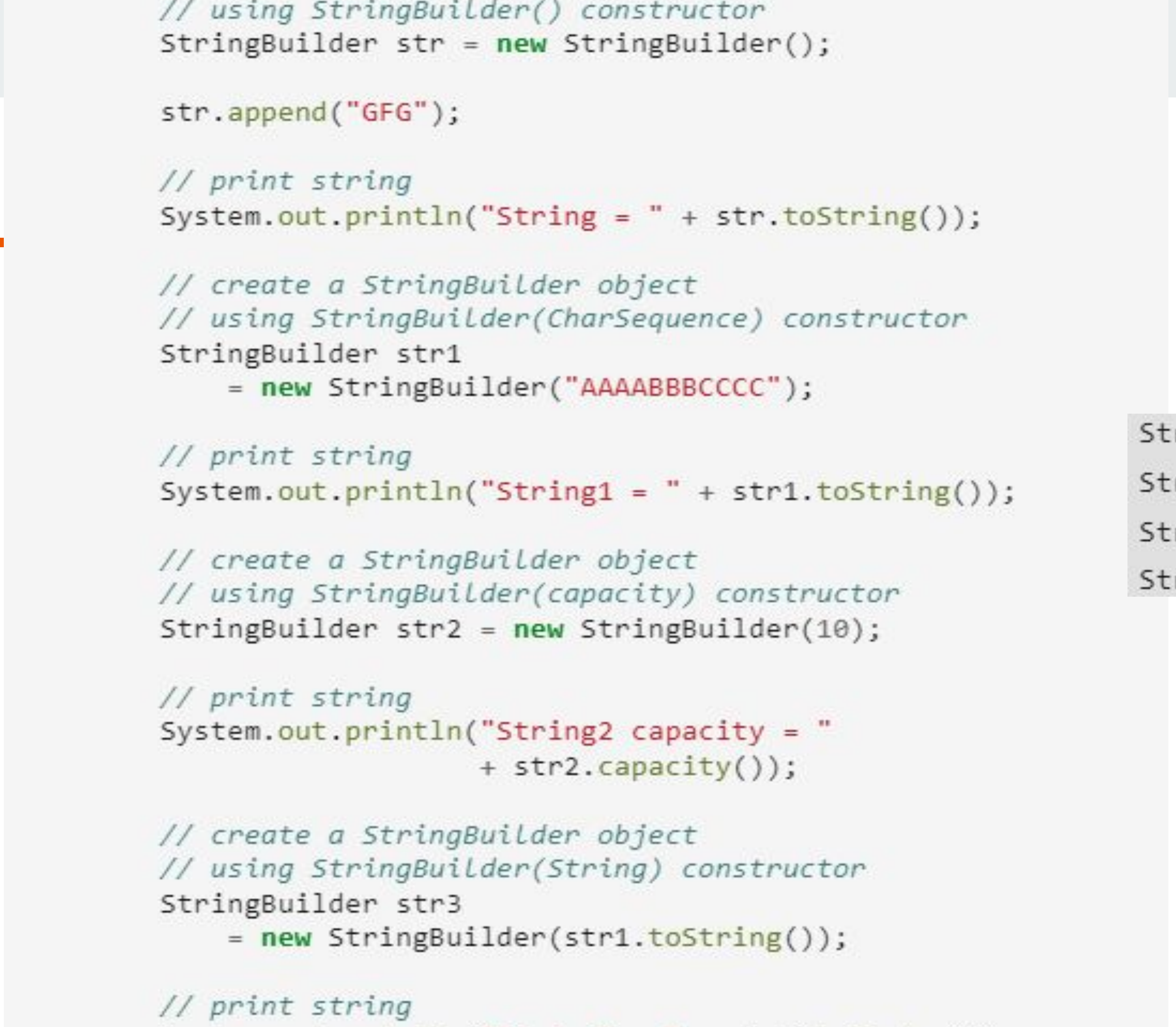
StringBuilder

StringBuilder(): Constructs a string builder with no characters in it and an initial capacity of 16 characters.

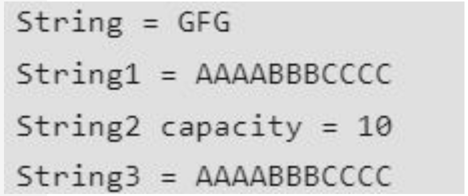
StringBuilder(int capacity): Constructs a string builder with no characters in it and an initial capacity specified by the capacity argument.

StringBuilder(CharSequence seq): Constructs a string builder that contains the same characters as the specified CharSequence.

StringBuilder(String str): Constructs a string builder initialized to the contents of the specified string.



```java
// using StringBuilder() constructor
StringBuilder str = new StringBuilder();

str.append("GFG");

// print string
System.out.println("String = " + str.toString());

// create a StringBuilder object
// using StringBuilder(CharSequence) constructor
StringBuilder str1
    = new StringBuilder("AAAABBBCCCC");

// print string
System.out.println("String1 = " + str1.toString());

// create a StringBuilder object
// using StringBuilder(capacity) constructor
StringBuilder str2 = new StringBuilder(10);

// print string
System.out.println("String2 capacity = "
                   + str2.capacity());

// create a StringBuilder object
// using StringBuilder(String) constructor
StringBuilder str3
    = new StringBuilder(str1.toString());

// print string
```



```
String = GFG
String1 = AAAABBBCCCC
String2 capacity = 10
String3 = AAAABBBCCCC
```

StringBuffer

StringBuffer is a class in Java that represents a mutable sequence of characters.

It provides an alternative to the immutable String class, allowing you to modify the contents of a string without creating a new object every time.

StringBuffer(): creates an empty string buffer with an 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.

StringBuffer objects are mutable, meaning that you can change the contents of the buffer without creating a new object.

The initial capacity of a StringBuffer can be specified when it is created, or it can be set later with the ensureCapacity() method.

The append() method is used to add characters, strings, or other objects to the end of the buffer.

The insert() method is used to insert characters, strings, or other objects at a specified position in the buffer.

The delete() method is used to remove characters from the buffer.

The reverse() method is used to reverse the order of the characters in the buffer.

CharSequence

A CharSequence is a readable sequence of char values.

This interface provides uniform, read-only access to many different kinds of char sequences.

A Charsequence is an important concept in Java that refers to an array of characters, or a sequence of characters, usually referred to as a “string”.

It is similar to a traditional string in terms of the data type it holds, but the underlying implementation is different.

A Charsequence is usually represented by the CharSequence interface in Java. It allows for the manipulation of strings in a more efficient way than a traditional string.

Modifier and Type	Method	Description
`char`	`charAt(int index)`	Returns the `char` value at the specified index.
`default IntStream`	`codePoints()`	Returns a stream of code point values from this sequence.
`static int`	`compare(CharSequence cs1, CharSequence cs2)`	Compares two `CharSequence` instances lexicographically.
`int`	`length()`	Returns the length of this character sequence.
`CharSequence`	`subSequence(int start, int end)`	Returns a `CharSequence` that is a subsequence of this sequence.
`String`	`toString()`	Returns a string containing the characters in this sequence in the same order as this sequence.

Pattern Matcher

The **matcher(CharSequence)** method of the **Pattern** class used to generate a matcher that will helpful to match the given input as parameter to method against this pattern.

The Pattern.matcher() method is very helpful when we need to check a pattern against a text a single time, and the default settings of the Pattern class are appropriate.

```
public Matcher matcher(CharSequence input)
```

Parameters: This method accepts a single parameter **input** which represents the character sequence to be matched.

Return Value: This method returns a new matcher for this pattern.

Below programs illustrate the matcher(CharSequence) method:

```
package demo;

//Java program to demonstrate
//Pattern.matcher(CharSequence) method

import java.util.regex.Pattern;

public class Matcher {
        public static void main(String[] args)
        {
                // create a REGEX String
                String REGEX = "oo";

                // create the string
                // in which you want to search
                String actualString
                        = "lpsforloops";
// create a Pattern
            Pattern pattern = Pattern.compile(REGEX);

            // get a matcher object
            java.util.regex.Matcher matcher = pattern.matcher(actualString);

            // print values if match found
            boolean matchfound = false;

            while (matcher.find()) {
                System.out.println("found the Regex in text:"
      + matcher.group()
                                         + " starting index:" + matcher.start()
                                         + " and ending index:"
                                         + matcher.end());

                matchfound = true;
            }
            if (!matchfound) {
                System.out.println("No match found for Regex.");
            }
      }
}
```

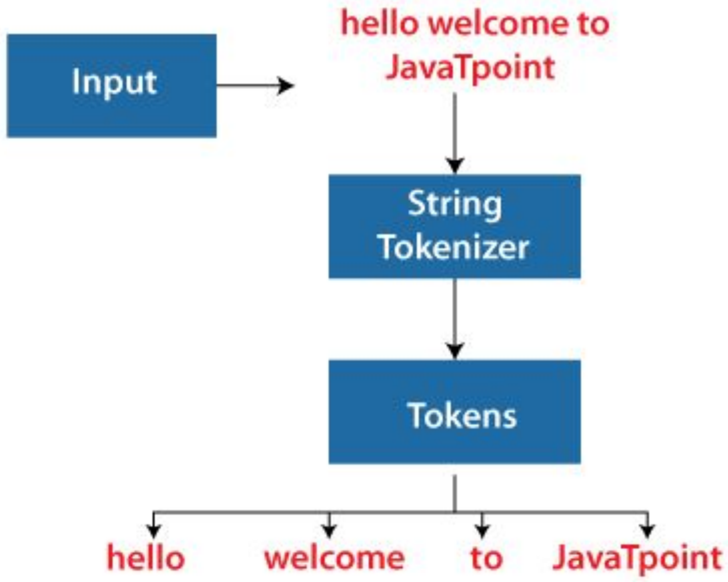
StringTokenizer

The java.util.StringTokenizer class allows you to break a String into tokens. It is simple way to break a String. It is a legacy class of Java.

It doesn't provide the facility to differentiate numbers, quoted strings, identifiers etc. like StreamTokenizer class. We will discuss about the StreamTokenizer class in I/O chapter.

In the StringTokenizer class, the delimiters can be provided at the time of creation or one by one to the tokens

Example of String Tokenizer class in Java



Constructor	Description
StringTokenizer(String str)	It creates StringTokenizer with specified string.
StringTokenizer(String str, String delim)	It creates StringTokenizer with specified string and delimiter.
StringTokenizer(String str, String delim, boolean returnValue)	It creates StringTokenizer with specified string, delimiter 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	**Description**
boolean hasMoreTokens()	It checks if there is more tokens available.
String nextToken()	It returns the next token from the StringTokenizer object.
String nextToken(String delim)	It returns the next token based on the delimiter.
boolean hasMoreElements()	It is the same as hasMoreTokens() method.
Object nextElement()	It is the same as nextToken() but its return type is Object.
int countTokens()	It returns the total number of tokens.

Character

Java provides a wrapper class Character in java.lang package. An object of type Character contains a single field, whose type is char. The Character class offers a number of useful class (i.e., static) methods for manipulating characters.

```
Character ch = new Character('a');
```

If we pass a primitive char into a method that expects an object, the compiler automatically converts the char to a Character class object. This feature is called Autoboxing and Unboxing.

Note: The Character class is immutable like String class i.e once it’s object is created, it cannot be changed.

CharArrayReader

java.io.CharArrayReader class creates a character buffer using a character array.

```
public class CharArrayReader
      extends Reader
```

Constructor :

CharArrayReader(char[] char_array) : Creates a CharArrayReader from a specified character array.

CharArrayReader(char[] char_array, int offset, int maxlen) : Creates a CharArrayReader from a specified part of character array.

CharArrayWriter

java.io.CharArrayWriter class creates a character buffer that can be used as a writer. The buffer automatically grows when data is written to the stream. The data can be retrieved using toCharArray() and toString().

```
public class CharArrayWriter

   extends Writer
```

Constructor :

CharArrayWriter() : Creating a CharArrayWriter from a specified character array.

CharArrayWriter(int size) : Creating a CharArrayWriter with the specified initial size.

StringJoiner

StringJoiner is a class in java.util package is used to construct a sequence of characters(strings) separated by a delimiter and optionally starting with a supplied prefix and ending with a given suffix.

Though this can also be done with the help of the StringBuilder class to append delimiter after each string, StringJoiner provides an easy way to do that without much code to write.

StringJoiner constructor

1. StringJoiner(CharSequence delimiter): It constructs a StringJoiner with no characters, no prefix or suffix, and a copy of the supplied delimiter.

2. StringJoiner(CharSequence delimiter, CharSequence prefix, CharSequence suffix): It constructs a StringJoiner with no characters using copies of the supplied prefix, delimiter, and suffix. If no characters are added to the StringJoiner and methods accessing the string value are invoked, it will return the prefix + suffix (or properties thereof) in the result unless *setEmptyValue* has first been called.

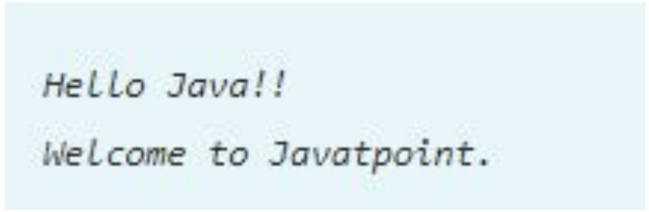
StringReader

Java StringReader class is a character stream with string as a source. It takes an input string and changes it into character stream. It inherits Reader class.

In StringReader class, system resources like network sockets and files are not used, therefore closing the StringReader is not necessary.

int read()	It is used to read a single character.
int read(char[] cbuf, int off, int len)	It is used to read a character into a portion of an array.
boolean ready()	It is used to tell whether the stream is ready to be read.
boolean markSupported()	It is used to tell whether the stream support mark() operation.
long skip(long ns)	It is used to skip the specified number of character in a stream
void mark(int readAheadLimit)	It is used to mark the mark the present position in a stream.
void reset()	It is used to reset the stream.
void close()	It is used to close the stream.

```
import java.io.StringReader;
public class StringReaderExample {
    public static void main(String[] args) throws Exception {
        String srg = "Hello Java!! \nWelcome to Javatpoint.";
        StringReader reader = new StringReader(srg);
        int k=0;
            while((k=reader.read())!=-1){
                System.out.print((char)k);
            }
        }
}
```



```
Hello Java!!
Welcome to Javatpoint.
```

StringWriter

Java StringWriter class is a character stream that collects output from string buffer, which can be used to construct a string. The StringWriter class inherits the Writer class.

In StringWriter class, system resources like network sockets and files are not used, therefore closing the StringWriter is not necessary.

void write(int c)	It is used to write the single character.
void write(String str)	It is used to write the string.
void write(String str, int off, int len)	It is used to write the portion of a string.
void write(char[] cbuf, int off, int len)	It is used to write the portion of an array of characters.
String toString()	It is used to return the buffer current value as a string.
StringBuffer getBuffer()	It is used t return the string buffer.
StringWriter append(char c)	It is used to append the specified character to the writer.
StringWriter append(CharSequence csq)	It is used to append the specified character sequence to the writer.
StringWriter append(CharSequence csq, int start, int end)	It is used to append the subsequence of specified character sequence to the writer.
void flush()	It is used to flush the stream.
void close()	It is used to close the stream.

```
import java.io.*;
public class StringWriterExample {
    public static void main(String[] args) throws IOException {
        char[] ary = new char[512];
        StringWriter writer = new StringWriter();
        FileInputStream input = null;
        BufferedReader buffer = null;
        input = new FileInputStream("D://testout.txt");
        buffer = new BufferedReader(new InputStreamReader(input, "UTF-8"));
        int x;
        while ((x = buffer.read(ary)) != -1) {
                    writer.write(ary, 0, x);
        }
        System.out.println(writer.toString());
        writer.close();
        buffer.close();
```

Formatter

The Java Formatter class provides support for layout justification and alignment, common formats for numeric, string, and date/time data, and locale-specific output.

```
import java.util.Formatter;
import java.util.Locale;

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

    // create a new formatter
    StringBuffer buffer = new StringBuffer();
    Formatter formatter = new Formatter(buffer, Locale.US);

    // format a new string
    String name = "World";
    formatter.format("Hello %s !", name);

    // print the formatted string
    System.out.println(formatter);

    formatter.close();
  }
}
```