**Java ASCII**

**ASCII** stands for **American Standard Code for Information Interchange**. ASCII is a standard data-transmission code that is used by the computer for representing both the textual data and control characters.

**ASCII** is a 7-bit character set having 128 characters, i.e., from 0 to 127. ASCII represents a numeric value for each character, such as 65 is a value of **A.** In our Java program, we need to manipulate characters that are stored in ASCII.

public class AsciiValue {

public static void main(String[] args) {

char ch = 'a';

int ascii = ch;

// You can also cast char to int

int castAscii = (int) ch;

System.out.println("The ASCII value of " + ch + " is: " + ascii);

System.out.println("The ASCII value of " + ch + " is: " + castAscii);

}

}

**JAVA Strings**

Strings, which are widely used in Java programming, are nothing but a sequence of characters.

**Creating Strings :**

String s = "Hello World";

Strings can be created as shown above, here String is the data type and s is the reference which refers to the data "Hello World"

String s = new String("Hello World");

char c[] = {'j', 'a', 'v', 'a'};  
String s = new String(c);

When we declare a string variable and assign some value, it means that we are creating a string object which cannot be changed. Whenever we modify the String a new String object is created with the modifications, and the original string is left as it is.  **String objects are immutable**.

class A{

public static void main(String args[]){

String s="Rohit";

s.concat(" Shetty");//concat() method appends the string at the end

System.out.println(s);//will print Rohit because strings are immutable objects

}

}

Diagram

Description automatically generated

If we change it to s = s. concat(" Shetty");// then it will refer to Rohit Shetty object

**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.

**Concatenation**

String a = “a” + “b” + “c”;

**Java length() Method**

Methods used to obtain information about an object are known as accessor methods. One such accessor method is length() method, which returns the number of characters contained in the String object.

class StringLengthTest  
{  
    public static void main(String arg[])  
    {  
        String place = "Hyderabad";  
        int length = place.length(); // LINE A  
        System.out.println("String Length of " + place + " is : " + length);  
          
        String welcome = "Welcome to Merit Campus";  
        for (int i = 0; i < welcome.length(); i++) // LINE B  
        {  
            System.out.print(welcome.charAt(i));  
        }      
    }  
}

**Storage of String literals and String object**

String literals are stored in String-pool(Collection of Strings) and the String objects are stored in the heap memory.  
Each time when you create a String literal JVM first checks whether the String is available in String-pool if the same String exists in String-pool then no new String object is created just the reference is changed. But it is not the case with String objects each time when you create Sting object using new key word each time a new object of type String is created.

String literals :

String s = "Tuesday";

String object :

String s1 = new String("Tuesday");

String s1 = **new** String("Tuesday");

String s2 = **new** String("Tuesday");

String s3 = "Tuesday";

String s4 = "Tuesday";

Comparing strings using == and equals

**equals and ==**

== is a relational operator that returns true when both the string reference variables under comparison are referring to the same object. == simply looks at the bits in the variable, and they're either identical or they're not.  
  
equals is a method defined in the Object class. The default implementation of the method provided in this class is similar to "==" equality operator and returns true if we are comparing two references of the same object. But this method is overridden by String class in such a way that when equals() of the String class is invoked, the contents of the String objects rather than the references are compared and a boolean value is returned accordingly i.e. if both the String objects contain the same sequence of characters, true is returned else false.

**String Concatenation**

String concatenation is the operation of joining two or more string objects end to end.

Use concat method and + operator

  NOTE: concat is not applicable to any other data type other than String.

A string can be concatenated with various data types. They include int, float, double, boolean, references etc.,

Different example using the “+” operator

“Java + 1 + 2”

**String Conversion**

import java.io.\*;  
  
class StringConversiontoStringTest  
{  
    public static void main(String arg[])  
    {  
        int a = 10;  
        float b = 20.5f;  
        char c = 'd';  
        double d = 19.5;  
        System.out.println(a + b + c + d);  
          
        String s1 = Integer.toString(a);  
        String s2 = Float.toString(b);  
        String s3 = Character.toString(c);  
        String s4 = Double.toString(d);  
        System.out.println(s1 + " " + s2+ " " + s3 + " " + s4);  
          
        double cake = 245.75;  
        String rate = Double.toString(cake);  
        String[] r = rate.split("\\.");  
        String r1 = r[0];  
        String r2 = r[1];  
        System.out.println("The Price of cake is " + r1 + " Rupees " + r2 + " Paise.");      
    }  
}

**charAt() method**

class CharAt  
{  
    public static void main(String arg[])  
    {  
        String s1 = "Tuesday";  
        char c1 = s1.charAt(0); //LINE A  
        System.out.println(c1);  
        char c2 = s1.charAt(4);  //LINE B  
        System.out.println(c2);      
    }  
}

see exceptions being thrown

**getChars**()

The getChars() method copies characters from the string into the destination character array.

Syntax of getChars() method is:  
public void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)  
This method copies characters from index srcBegin to (srcEnd - 1) to the subarray of dst starting at the index dstBegin. The number of characters that are copied to the character array is given by (srcEnd - srcBegin).

String source = "Happy Birthday";

**char**[] dest = **new** **char**[20];

dest[5] = 'N';

dest[6] = 'E';

dest[7] = 'W';

dest[8] = 'Y';

dest[9] = 'E';

dest[10] = 'A';

dest[11] = 'R';

**try** {

System.***out***.println("dest array contents:" + String.*valueOf*(dest) );

source.getChars( 0, 5, dest, 0 ); // LINE A

System.***out***.println( dest );

} **catch** ( Exception ex ) {

System.***out***.println( "An exception occured!" );

}

**toCharArray()**

The method toCharArray() is used to convert any String into array of characters.

class ToCharDemo  
{  
    public static void main(String arg[])  
    {  
        String one = "Hyderabad";  
        char[] char\_one = one.toCharArray();  
        System.out.println(one);  
        for (int i = 0; i < char\_one.length; i++)   
        {  
            System.out.print(char\_one[i] + " ");  
        }      
    }  
}

**Equals and Equals Ignore case**

The equals and equalsIgnoreCase are used to compare two Strings and states whether they are equal or not.

The equals is represented as:

*boolean equals(String str)*

The equals will compare String strwith the invoking String. It returns true if both the strings have same characters in same sequence else it returns false.  
  
The equalsIgnoreCase is represented as:

*boolean equalsIgnoreCase(String str)*

The equalsIgnoreCase() will compare String str with the invoking String Object. It returns true if both the strings have same characters in same sequence but it ignores the case differences i.e it considers [A-Z]=[a-z], A is same as a and Y is same as y.

class EqualsDemo  
{  
    public static void main(String arg[])  
    {  
        String one = "Hyderabad";  
        String two = "HYDERABAD";  
        String three = "Hyderabad";  
        String four = "Kakinada";  
        System.out.println(one + " equals " + three + " -- " + one.equals(three));  
        System.out.println(one + " equals " + two + " -- " + one.equals(two));  
        System.out.println(one + " equals IgnoreCase " + two + " -- "   
        + one.equalsIgnoreCase(two));  
        System.out.println(one + " equals " + four + " -- " + one.equals(four));      
    }  
}

**regionMatches()**

The regionMatches method compares a specific region in a String with a specific region in another String. It returns trueif both the regions in both the strings are having same sequence of characters, else it returns false.

The regionMatches has two forms. The general form is

*boolean regionMatches(int startIndex, String str, int strStartIndex, int numChars)*

Here startIndex specifies starting index of region of the invoked string. str is the String to be compared. strStartIndex is the starting index of the region of the String str. numChars is the length of the substring being compared.  
The second form of regionMatches is the overloaded form. It includes option for ignoring the case while comparing.

*boolean regionMatches(boolean ignoreCase, int startIndex, String str, int strStartIndex, int numChars)*

Here if ignoreCase is true, then the comparison is not case-sensitive else if ignoreCase is false the comparison is case-sensitive.

class RegionCompare  
{  
    public static void main(String arg[])  
    {  
        String fruit = "apple";  
        boolean result = fruit.regionMatches(0, "pineapple", 4, 5);  
        System.out.println("apple regionMatches pineapple -- " + result);  
        boolean result1 = fruit.regionMatches(0, "pineapple", 0, 4);  
        System.out.println("apple regionMatches pineapple -- " + result1);  
        boolean result2 = fruit.regionMatches(0, "pineAPPLE", 4, 5);  
        System.out.println("apple regionMatches pineAPPLE -- " + result2);  
        boolean result3 = fruit.regionMatches(false, 0, "pineAPPLE", 4, 5);  
        System.out.println("apple regionMatches pineAPPLE -- " + result3);  
        boolean result4 = fruit.regionMatches(true, 0, "pineAPPLE", 4, 5);  
        System.out.println("apple regionMatches pineAPPLE -- " + result4);  
      
    }  
}

**startWith() and EndsWith()**

**Searching strings – java indexOf and lastIndexOf**

The String class provides 2 methods for searching a string. They are :

* indexOf() : Searches for the first occurrence of a character or substring.
* lastIndexOf() : Searches for the last occurrence of a character or substring.

These methods return the starting index of character or a substring on a successful search else they return -1.

class StringSearchDemo  
{  
    public static void main(String arg[])  
    {  
        String str = "The Sun rises in the east and sets in the west.";  
        System.out.println("Length of str : " + str.length());  
        System.out.println("indexOf(i) : " + str.indexOf('i')); // LINE A  
        System.out.println("lastIndexOf(i) : " + str.lastIndexOf('i')); // LINE B  
        System.out.println("indexOf(st) : " + str.indexOf("st")); // LINE C  
        System.out.println("lastIndexOf(st) : " + str.lastIndexOf("st")); // LINE D  
        System.out.println("indexOf(e, 2) : " + str.indexOf('e', 2)); // LINE E  
        System.out.println("lastIndexOf(e, 46) : " + str.lastIndexOf('e', 46)); // LINE F  
        System.out.println("indexOf(st, 2) : " + str.indexOf("st", 2)); // LINE G  
        System.out.println("lastIndexOf(st, 46) : " + str.lastIndexOf("st", 46)); // LINE H      
    }  
}