# **String**

• **String is a built-in class present inside java.lang package.**

**• String is final class, so you can't define the subclass for String class.**

• **String class implements the following interfaces:-**

* **java.io.Serializable**
* **java.lang.Comparable**
* **java.lang.Char[]Sequence**

• **String class has following variable to hold data:-**

* **prívate final char value[);**

**Following is the String class definition by Java Vendor:-**

* **public final class String implements Serializable, Comparable, CharSequence(**

**private final char value[];**

**….**

**}**

• **String objects are immutable objects. It means once the object is created then the content**

**data of the object can't be modified.**

• **When you try to modify the contents of object then new object will be created as a result.**

• **You can create the object of String in two ways.**

* **Without using new operator**
* **Using new operator.**

# **Creating String object without new operator**

1) **JVM** allocates the memory for the **String reference variable.**

2) **JVM** verifies the **String literal** in the **String Constant Pool.**

**3) If String literal is not available in the String Constant Pool then it creates new String object inside the String Constant pool and newly created String object address will be assigned to the String reference variable.**

**4) If String literal is available then existing String object address will be assigned to String reference variable.**

**Ex:-**

**package** com.String;

**public** **class** P1 {

**public** **static** **void** main(String[] args) {

String s="Raushan";

String s1="Raushan";

String s2="Raushan";

System.***out***.println(s);//Raushan

System.***out***.println(s1);//Raushan

System.***out***.println(s2);//Raushan

System.***out***.println(s==s1);//true

System.***out***.println(s==s2);//true

}

}

**“Raushan”**

P1@100

S

P1@100

S2

P1@100

S3

# **Creating String object with new operator**

**1) JVM allocates the memory for the String reference variable.**

**2) JVM verifies the String literal in the String Constant Pool.**

**3) If String literal is not available in the String Constant Pool then it creates new String object**

**Inside the String Constant pool.**

**4) If String literal is available in the String Constant Pool then ignores that.**

**5) At creates another new String object outside the constant pool and assigns address of the**

**Newly created String object outside the pool to String reference variable.**

**Ex:-**

**package** com.String;

**public** **class** P2 {

**public** **static** **void** main(String[] args) {

String s1="Raushan";

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

String s3=**new** String("Raushan");

System.***out***.println(s1);

System.***out***.println(s2);

System.***out***.println(s3);

System.***out***.println(s1==s2);

System.***out***.println(s1==s3);

System.***out***.println(s2==s3);

}

}

**P2@100**

“Raushan”

**P2@100**

S1

**P2@200 P2@200**

“Raushan”

S2

**P2@300 P2@300**

“Raushan”

S3

# **String Class API**

**Important Constructors from String Class**

* **public String()**

**Create a new String with the empty content**

* **public String(String str)**

**Create a new String with the specified content.**

* **public String(char[] arr)**

**Create a new String with the specified array of characters.**

* **public String(char[], int startIndex, int length)**

**Create a new String with the specified part of array of characters.**

* **public String(byte[], int startIndex, int length)**

**Create a new String with the specified part of array of bytes**

* **public String(byte[])**

**Create a new String with the specified array of bytes.**

* **public String(StringBuffer)**

**Create a new String with the specified StringBuffer.**

* **public String(StringBuilder)**

**Create a new String with the specified StringBuilder.**

**Important Methods from String Class:-**

* **public native String intern()**

**Returns the reference of string object from Constant pool.**

* **public int length()**

**Returns the length of the String object.**

* **public boolean isEmpty()**

**Returns true if, and only if, length () is 0.**

* **public String concat(String)**

**Concatenates the specified string to the end of current string.**

* **public String toLowerCase()**

**Converts all of the characters in the given String to lower case.**

* **public String toUpperCase()**

**Converts all of the characters in the given String to upper case.**

* **public String trim()**

**Trim whitespace from the beginning and end of a string.**

* **public String toString()**

**Returns the current object content.**

* **public static String valueOf(X val)** X can be Object, int , boolean , double etc.

**Returns the string representation of the Specified argument.**

* **public char charAt(int Index)**

**Returns the char value at the specified index.**

* **public char [] toCharArray()**

**Converts the string to a new character array.**

* **public void getChars(int srcBegin, int srcEnd,char [] dst , int dstBegin)**

**Copies characters from the string into the destination character array.**

* **public byte [] get Bytes()**

**Converts the String to a new byte array using ASCII.**

* **public void getBytes(int srcBegin, int srcEnd, byte [] dst, int dstBegin)**

**Copies ASCII of the characters from the string into the destination byte array.**

* **public boolean equals(Object)**

**Compares content of this string to the specified object.**

**Case of the character also will be verified.**

* **public boolean equalsIgnoreCase(String)**

**Compares content of the string to the specified object.**

**Case of the character will be ignored.**

* **public boolean contentEquals(StringBuffer)**

**Compares content of the string to the specified StringBuffer object.**

**Case of the character also will be verified.**

* **public boolean contentEquals(CharSequence)**

**Compares content of the string to the specified CharSequence object.**

**Case of the character also will be verified.**

* **public int compareTo(String)**

**Compares content of the string to the specified object.**

**Case of the character also will be verified.**

* **public int compareTolgnoreCase(String)**

**Compares content of the string to the specified object.**

**Case of the character will be ignored.**

* **public boolean startsWith(String prefix)**

**Checks whether string starts with the specified prefix.**

* **public boolean startsWith(String P, int fromlndex)**

**Checks whether string starts with the specified prefix from specified starting index.**

* **public boolean endsWith(String suffix)**

**Checks whether string starts with the specified suffix.**

* **public int indexOf(int ch)**

**Returns the index of the first occurrence of the specified character.**

* **public int indexOf(int ch, int fromlndex)**

**Returns the index of the first occurrence of the specified character.**

**It starts search from the specified index.**

* **public int indexOf(String st)**

**Returns the index of the first occurrence of the specified String.**

* **public int indexOf(String st, int fromlndex)**

**Returns the index of the first occurrence of the specified String.**

**It starts search from the specified index.**

* **public int lastIndexOf(int ch)**

**Returns the index of the last occurrence of the specified character.**

* **public int lastIndexOf(int ch, int fromlndex)**

**Returns the index of the last occurrence of the specified character.**

**It starts search backward from the Specified index.**

* **public int lastindex0f(String st)**

**Returns the index of the last occurrence of the specified String.**

* **public int lastindex0f(String st, int fromlndex)**

**Returns the index of the last occurrence of the specified String.**

**It starts search backward from the specified index.**

* **public String substring(int beginlndx)**

**Returns the part of the String.**

**Substring begins with the character at the specified index and up to the end of this string.**

* **public String substring (int beginlndx, int endindx)**

**Returns the part of the String.**

**Substring begins with the character at the specified index and up to the (endindx- 1).**

* **public CharSequence subSequence(int beginlndx, int endindx)**

**Returns the part of the String as CharSequence. Result SubSequence begins with the character at the specified index and up to the (endindx - 1).**

* **public String, replace(char oChar, char nChar)**

**Replaces all occurrences of oChar with nChar.**

* **public String replaceFirst(String regEx, String newString)**

**Replaces first occurrences of matching String with newString.**

* **public String replaceAll(String regEx, String newString)**

**Replaces all occurrences of matching String with newString.**

* **public String replace(CharSequence target, CharSequence newCharSeq)**

**Replaces all occurrences of matching CharSequence with new CharSequence.**

* **public boolean matches(String regEx)**

**Checks whether the string matches with the given regular expression.**

* **public boolean contains(CharSequence)**

**Compares content of the string to the specified CharSequence.**

* **public boolean regionMatches(int fromlndex, String otherStr ,int fromlndexlnOtherStr, int length)**

**Compares parts or regions of two strings. Case of the character also will be verified.**

* **public boolean regionMatches(Boolean ignoreCase, int fromlndex, String otherStr, int fromlndexlnOtherStr, int length)**

**Compares parts or regions of two strings.**

**When ignoreCase is true then case of the character also will be ignored.**

* **public String[] split(String regEx)**

**Splits the string when the given regular expression matches.**

* **public String[I split(String regEx, int limit)**

**Splits the string when the given regular expression matches.**

**Limit controls the number of times the pattern is applied and**

**It affects the length of the resulting array.**

* **public static String format(String, Object...)**

**Returns a formatted string using the specified format string and arguments.**

* **public int hashCode()**

**Returns the hashCode of String object.**

**Uses contents of String generate the hashCode.**