String

- string is an object that represents sequence of char values.
- Java string is an immutable class.
- The java.lang.String class implements *Serializable*, *Comparable* and *CharSequence* interfaces.
- There are two ways to create String object:
  1. By string literal
  2. By new keyword
- **Using String Literal:**
  - 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 string doesn't exist in the pool, a new string instance is created and placed in the pool.

    ```
    String s1="Welcome";
    String s2="Welcome";//will not create new instance
    ```

String…

- **Using new keyword**
  - 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.
  - String s=new String("Welcome");//creates two objects and one r eference variable
- **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);
}}
```

String …

- **string objects are immutable**.
- Example

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

Output:Sachin

String …

- String Comparison
  - There are three ways to compare string in java:
    1. By equals() method
    2. By = = operator
    3. By compareTo() method
  - String equals() method compares the original content of the string.
  - It compares values of string for equality.
  - String class provides two methods:
    - **public boolean equals(Object another)**
    - **public boolean equalsIgnoreCase(String another)**

```
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
```

String …

- **String compare by == operator**
  - The = = operator compares references not values.
  - Example:

```
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)
 }
}
```

String …

- **compareTo() method**
  - The String compareTo() method compares values lexicographically and returns an integer value.
  - Suppose s1 and s2 are two string variables. If:
    - **s1 == s2** :0
    - **s1 > s2** :positive value
    - **s1 < s2** :negative value
  - Example:

```
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 )
 }
}
```

String …

- String Concatenation
  - String Concatenation by + (string concatenation) operator

    Ex 1.

    String s="Sachin"+" Tendulkar";

    System.out.println(s);//Sachin Tendulkar

    Ex2.

    String s=50+30+"Sachin"+40+40;

    System.out.println(s);//80Sachin4040
  - String Concatenation by concat() method
    - The String concat() method concatenates the specified string to the end of current string.
    - Ex1. String s1="Sachin ";

      String s2="Tendulkar";

      String s3=s1.concat(s2);

      System.out.println(s3);//Sachin Tendulkar

String …

- **Substring**
  - A part of string is called **substring**.
  - There are two methods to get substring from string object.
    - public String substring(int startIndex)
    - public String substring(int startIndex, int endIndex)

  Example:

```
String s="SachinTendulkar";
System.out.println(s.substring(6));//Tendulkar
System.out.println(s.substring(0,6));//Sachin
```

- **toUpperCase() :** this method converts this string into uppercase letter
- **toLowerCase():** this method converts this string into lowercase letter
- **trim():** method eliminates white spaces before and after string.

String …

- **startsWith():** This method check whether the string starts with some prefix or not.
- **endsWith():** This method check whether the string ends with some suffix or not.
- **charAt():** This method returns a character at specified index.
- **length():** This method returns length of the string.
- **valueOf():** This method coverts given type such as int, long, float, double, boolean, char and char array into string.
- **replace():** This method replaces all occurrence of first sequence of character with second sequence of character.
- **contains(CharSequence sequence):** It returns *true* if sequence of char values are found in this string otherwise returns *false*.

String …

- **format():** this method returns the formatted string by given locale, format and arguments.
  - Example: String sf3=String.format("value is %32.12f",32.33434);
- **getBytes():** this method returns the byte array of the string. In other words, it returns sequence of bytes.
- **getChars()**: this method copies the content of this string into specified char array.

  public void getChars(int srcBeginIndex, int srcEndIndex, char[] destination, int dstBeginIndex)
- **indexOf():** this method returns index of given character value or substring. If it is not found, it returns -1.
- **intern():** this method returns the interned string. It can be used to return string from pool memory, if it is created by new keyword.
  - String s1=new String("hello"); String s2="hello"; String s3=s1.intern(); System.out.println(s1==s2);//false System.out.println(s2==s3);//true

String …

- **isEmpty()**: this method checks if this string is empty. It returns *true*, if length of string is 0 otherwise *false*.
- **join()** : this method returns a string joined with given delimiter.

  String joinString1=String.join("-","welcome","to",“CGEC");

  Output: welcome-to-CGEC
- **split():** This method splits this string against given regular expression and returns a char array.

  String s1="java string split method by javatpoint";
  String[] words=s1.split("\\s");//splits the string based on whitespace
- **toCharArray()**: this method converts this string into character array.

StringBuffer class

- StringBuffer class is used to create mutable (modifiable) string.
- Java StringBuffer class is thread-safe
- **Constructors of StringBuffer class**
  - StringBuffer()
  - StringBuffer(String str)
  - StringBuffer(int capacity)
- **methods of StringBuffer class**
  - **append(String s):** it is used to append the specified string with this string.
  - **insert(int offset, String s):**it is used to insert the specified string with this string at the specified position.
  - **replace(int startIndex, int endIndex, String str):** it is used to replace the string from specified startIndex and endIndex.
  - **delete(int startIndex, int endIndex):** it is used to delete the string from specified startIndex and endIndex.

StringBuffer class…

- **reverse():** it is used to reverse the string.
- **capacity():** it is used to return the current capacity.
- **ensureCapacity(int minimumCapacity):** it is used to ensure the capacity at least equal to the given minimum.
- **charAt(int index):** it is used to return the character at the specified position.
- **length():** it is used to return the length of the string i.e. total number of characters.
- **substring(int beginIndex):** it is used to return the substring from the specified beginIndex.
- **substring(int beginIndex, int endIndex):** it is used to return the substring from the specified beginIndex and endIndex.

StringBuilder class

- StringBuilder class is used to create mutable (modifiable) string.
- StringBuilder class is same as StringBuffer class except that it is non-synchronized.
- **Constructors of StringBuilder class**
  - StringBuilder()
  - StringBuilder(String str)
  - StringBuilder(int length)
- **methods of StringBuilder class(**Same as String Buffer but all methods non-synchronized**)**
  - **append(String s):** it is used to append the specified string with this string.
  - **insert(int offset, String s):**it is used to insert the specified string with this string at the specified position.
  - **replace(int startIndex, int endIndex, String str):** it is used to replace the string from specified startIndex and endIndex.
  - **delete(int startIndex, int endIndex):** it is used to delete the string from specified startIndex and endIndex.

StringBuilder class…

- **reverse():** it is used to reverse the string.
- **capacity():** it is used to return the current capacity.
- **ensureCapacity(int minimumCapacity):** it is used to ensure the capacity at least equal to the given minimum.
- **charAt(int index):** it is used to return the character at the specified position.
- **length():** it is used to return the length of the string i.e. total number of characters.
- **substring(int beginIndex):** it is used to return the substring from the specified beginIndex.
- **substring(int beginIndex, int endIndex):** it is used to return the substring from the specified beginIndex and endIndex.

Difference between String and StringBuffer

String	StringBuffer
String class is immutable.	StringBuffer class is mutable.
String is slow and consumes more memory when you concat too many strings because every time it creates new instance.	StringBuffer is fast and consumes less memory when you cancat strings.
String class overrides the equals() method of Object class. So you can compare the contents of two strings by equals() method.	StringBuffer class doesn't override the equals() method of Object class.

toString() method

- The toString() method returns the string representation of the object.
- If you print any object, java compiler internally invokes the toString() method on the object.
- So overriding the toString() method, returns the desired output.
- Example:

```
class Student{
 int rollno;
 String name;
 String city;
 public String toString(){//overriding the toString() method
   return rollno+" "+name+" "+city;
  }
}
```

StringTokenizer

- The **java.util.StringTokenizer** class allows you to break a string into tokens.
- **Constructors of StringTokenizer class**
  - StringTokenizer(String str)
  - StringTokenizer(String str, String delim)
  - StringTokenizer(String str, String delim, boolean returnValue)
- **Methods of StringTokenizer class**
  - boolean hasMoreTokens(): checks if there is more tokens available.
  - String nextToken(): returns the next token from the StringTokenizer object.
  - String nextToken(String delim): returns the next token based on the delimeter.
  - boolean hasMoreElements(): same as hasMoreTokens() method.
  - Object nextElement(): same as nextToken() but its return type is Object.
  - int countTokens(): returns the total number of tokens.

Regex

- **Regular Expressions** or **Regex** (in short) is an API for defining String patterns that can be used for searching, manipulating and editing a text.
- It is widely used to define constraint on strings such as password and email validation.
- **Regular Expressions** are provided under **java**.util.**regex** package.
- **java.util.regex package contains 1 interface and 3 classes**
  1. MatchResult interface
  2. Matcher class
  3. Pattern class
  4. PatternSyntaxException class

Regex: Matcher class

- It implements **MatchResult** interface. It is a *regex engine.*

Method	Description
boolean matches()	test whether the regular expression matches the pattern.
boolean find()	finds the next expression that matches the pattern.
boolean find(int start)	finds the next expression that matches the pattern from the given start number.
String group()	returns the matched subsequence.
int start()	returns the starting index of the matched subsequence.
int end()	returns the ending index of the matched subsequence.
int groupCount()	returns the total number of the matched subsequence.

Regex: Pattern class

- It is the *compiled version of a regular expression*.
- It is used to define a pattern for the regex engine.

Method	Description
static Pattern compile(String regex)	compiles the given regex and return the instance of pattern.
Matcher matcher(CharSequence input)	creates a matcher that matches the given input with pattern.
static boolean matches(String regex, CharSequence input)	It works as the combination of compile and matcher methods. It compiles the regular expression and matches the given input with the pattern.
String[] split(CharSequence input)	splits the given input string around matches of given pattern.
String pattern()	returns the regex pattern.

Regex...

- How to use regex?
  - 1st way

    Pattern p = Pattern.compile(".s");
    Matcher m = p.matcher("as");
    boolean b = m.matches();
  - **2nd way**

    boolean b2=Pattern.compile(".s").matcher("as").matches();
  - **3rd Way**
    - boolean b3 = Pattern.matches(".s", "as");

Regex...

- **Regex Character classes**

Character Class	Description
[abc]	a, b, or c (simple class)
[^abc]	Any character except a, b, or c (negation)
[a-zA-Z]	a through z or A through Z, inclusive (range)
[a-d[m-p]]	a through d, or m through p: [a-dm-p] (union)
[a-z&&[def]]	d, e, or f (intersection)
[a-z&&[^bc]]	a through z, except for b and c: [ad-z] (subtraction)
[a-z&&[^m-p]]	a through z, and not m through p: [a-lq-z](subtraction)

Regex…

- **Regex Quantifiers**

Regex	Description
X?	X occurs once or not at all
X+	X occurs once or more times
X*	X occurs zero or more times
X{n}	X occurs n times only
X{n,}	X occurs n or more times
X{y,z}	X occurs at least y times but less than z times

Regex...

- **Regex Metacharacters**

Regex	Description
.	Any character (may or may not match terminator)
\d	Any digits, short of [0-9]
\D	Any non-digit, short for [^0-9]
\s	Any whitespace character, short for [\t\n\x0B\f\r]
\S	Any non-whitespace character, short for [^\s]
\w	Any word character, short for [a-zA-Z_0-9]
\W	Any non-word character, short for [^\w]
\b	A word boundary
\B	A non word boundary