UNIT-III

* Strings in Java:

→ Strings are type of objects that can store the character of values & in Java, every character is stored in 16 bits i.e. using UTF 16 bit encoding.

→ Sequence of characters.

→ Immutable => constant & cannot be changed once created.

→ 2 ways to create a string:

① String Literal

② Using new keyword

① **String Literal:** To make java more memory efficient. ↓ because no new objects are created if it exists already in the string constant pool.

Eg: String demoString = "GeeksforGeeks";

② **Using new keyword:**

* String s = new String("Welcome");
* In such case, JVM will create a new string object in normal (non pool) heap memory & the literal "Welcome" will be placed in the string constant pool.
* The variable s will refer to the object in the heap (non-pool)

Eg: String demoString = new String("GeeksforGeeks");

* Immutable String :

→ String objects are immutable
↳ (unmodifiable / unchangeable)

→ once a string is ~~changed~~ created it can't be changed but a new string object is created.

Example :

```
class TestImmutableString1
{
public static void main (String args[])
{
String s = "sachin";
s = s.concat ("Tendulkar");
System.out.println (s);
}
}
```

* why string objects are immutable in java ?

→ It uses the concept of string literal.
Suppose there are 5 reference variables, all refer to one object "sachin". If one reference variable changes the value of the object, it will be affected by all the reference variables. That is why immutable.

→ Features of String to make it immutable :

① Class loader : A class loader in java uses a string object as an argument. To avoid the misinterpretation of string objects & classes, the class loader is used.

② Thread Safe: As the string object is immutable, we don't have to take care of the synchronization that is required while sharing an object across multiple threads.

③ Security: This makes the appl'cation program more secure.

Eg: Banking software → the username & password cannot be modified by any intruder because string objects are immutable.

④ Heap Space: helps to minimize the usage in the heap memory. Thus, whenever we try to declare a new string object, the JVM checks whether the value already exists in the string pool or not, If it exists, the same value is assigned to the new object.

* String Comparison:

→ 3 ways to compare string:

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

①. By using equals() Method:

It compares the original content of the string. It compares values of string for equality.

→ It uses 2 methods -

① public boolean equals(object another) compares the string to the specified object.

② public boolean equalsIgnoreCase(String another) compares this string to another string, ignoring case.

② By using == operator :

The == operator compares references not values.

③ By using compareTo() method :

It compares the values & returns an integer value that describes if first string is less than, equals to or greater than second string.

Suppose ;

S1 == S2 : The method returns 0.

S1 > S2 : +ve value

S1 < S2 : -ve value

* String Concatenation :

* String concatenatn forms a new string that is combinatn of multiple strings.

* 2 ways to concatenate :

① By + (string concatenatn) operator

② By concat() method.

① String concatenation by + operator :

```
class TestStringConcatenation
{
public static void main (String args[]).
{
String s = "Sachin" + "Tendulkar";
System.out.Println (s);
}
}
```

② String concatenation by concat() method :
The string concat() method concatenates the specified string to the end of current string.

```
class TestStringConcatenation3
{
  public static void main (String args[])
{
  String s1 = "Sachin";
  String s2 = "Tendulkar";
  String s3 = s1.concat(s2);
  System.out.Println(s3);
  }
}
```

③ String concatenation using StringBuilder class :
→ string builder provides append() method to perform concatenation.
→ fastest way to concatenate.
→ it is a mutable class.

```
public class StrBuilder.
{
  public static void main (String args[])
{
  StringBuilder s1 = new StringBuilder ("Hello");
  StringBuilder s2 = new StringBuilder ("World");
  StringBuilder s = s1.append(s2);
System.out.Println(s.toString());
  }
}
```

* Substring:

→ a part of string is called a substring.
→ substring is a subset of another string.
→ it extracts a substring from the given string by using the index values passed as an argument.
→ startIndex is inclusive
endIndex is exclusive

* we can get substring from given string by 2 methods-

① public string substring (int startIndex):

→ this method returns new string object containing the substring of the given string from specified startIndex.
→ this method throws an IndexOutOfBoundException when the startIndex is larger than the length of the string or less than zero.

② public string substring (int startIndex, int endIndex):

→ this method throws an IndexOutOfBoundException when the startIndex < 0 or startIndex > endIndex or endIndex > length of string.

Example:

```
public class ~~substring~~ TestSubstring
{
public static void main (String args[])
{
String s = "SachinTendulkar";
System.out.println ("Original String: " + s);
System.out.println ("substring starting from index 6:"
                    + s.substring (6));
System.out.println ("substring starting from index 0 to 6:"
                    + s.substring (0,6));
}
}
```

* Methods of String class:

By the help of these methods, we can perform operations on string objects such as trimming, concatenating, converting, comparing, replacing strings, etc.

①. Java String toUpperCase() & toLowerCase() method:

toUpperCase() method → uppercase letters
toLowerCase() method → lowercase letters

Eg:

```
public class Stringoperation1
{
  public static void main(String ar[])
  {
    String s = "sachin";
    System.out.Println(s.toUpperCase());
    System.out.Println(s.toLowerCase());
    System.out.Println(s);
  }
}
```

②. Java String Trim() method:

trim() method eleminates white spaces before & after the string.

Eg:

```
public class Stringoperation2
{
  public static void main(String ar[])
  {
    String s = "sachin";
    System.out.Println(s);
    System.out.Println(s.trim());
  }
}
```

③. Java String startsWith() & endsWith() method:

startsWith() → checks whether the string starts with the letters passed as arguments.

endsWith() → checks whether the string ends with the letters passed as arguments.

Eg:
```
public class StringOperation 3
{
 public static void main (String ar[])
 {
  String s = "Sachin";
  System.out.Println (s.startsWith ("Sa"));
  System.out.Println (s.endsWith ("n"));
 }
}
```

④. Java String charAt() method:

charAt() method returns a character at specified index.

Eg:
```
public class StringOperation 4
{
 public static void main (String ar[])
 {
  String s = "Sachin";
  System.out.Println (s.charAt(0));
  System.out.Println (s.charAt(3));
 }
}
```

⑤. Java String length() method:

length() → returns length of the specified string.

Eg:
```
public class StringOperation 5
{
 public static void main (String ar[])
 {
  String s = "Sachin";
  System.out.Println (s.length());
 }
}
```

⑥. Java String intern() method :

when the intern() method is invoked, if the pool already contains the string equal to this String object as determined by the equals(Object) method, then the string from the pool is returned. Otherwise, the String object is added to the pool & a reference to this string object is returned.

Eg :

```
public class stringoperation 6
{
  public static void main (string ar[])
  {
    string s = new string ("sachin");
    String s2 = s.intern();
    System.out.Printen (s2);
  }
}
```

⑦. Java String valueOf() method :

valueOf() method converts given type such as int, long, float, double & char array into string.

Eg :

```
public class stringoperation 7
{
  public static void main (string ar[])
  {
    int a=10;
    string s = string.valueOf (a);
    System.out.Printen (s+10);
  }
}
```

⑧. Java string indexof() method:

returns the index within the string of the 1st occurance of the specified string.

```
String s = "learn" share" learn";
int output = s.indexof("share"); //return 6;
```

* Java String Buffer class:

→ used to create mutable string objects.
→ String buffer class is same as the string class except that it is mutable.

* Constructors of string buffer class:

①. StringBuffer(): it creates an empty string buffer with the initial capacity of 16.

②. StringBuffer(String str): it creates a string buffer with the specified string.

③. String Buffer(int capacity): it creates an empty string buffer with the special capacity as length.

* Methods of string buffer class:

①. append(String s):

→ concatenates the given argument with string.

Eg:

```
class stringBufferExample
{ public static void main (String args[])
{ StringBuffer sb = new StringBuffer("Hello");
  sb.append("Java");
  System.out.Println(sb);
}
}
```

② insert() method :
insert the string with the given string at the given position.

Eg:
```
class StringBufferExample2
{
public static void main (String args[])
{
StringBuffer sb = new StringBuffer ("Hello");
sb.insert (1,"Java");
System.out.Println (sb);
}
}
```

③. replace() method :
replaces the given string from the specified given beginIndex & endIndex.

```
class StringBufferExample 3
{
public static void main (String args[])
{
StringBuffer sb = new StringBuffer ("Hello");
Sb.replace (1,3,"Java");
System.out.Println (Sb);
}
}
```

④. delete() method :
delete the string from specified beginIndex to the endIndex.

Eg:
```
class StringBufferExample 4
{
public static void main (String args[])
{
StringBuffer sb = new StringBuffer ("Hello");
sb.delete(1,3);
System.out.Println(sb);
}
}
```

⑤. reverse() method:

reverses the current string

Eg:

```
class StringBufferExample 5
{
public static void main (String args[])
{
StringBuffer sb = new StringBuffer ("Hello");
sb.reverse();
System.out.Println(sb);
}
}
```

⑥. capacity() method:

returns the current capacity of the buffer.

Eg:

```
class StringBufferExample 6
{
public static void main (String args[])
{
StringBuffer sb = new StringBuffer();
System.out.Println(sb.capacity());
sb.append("Hello");
System.out.Println(sb.capacity());
sb.append("java is my favourite language");
System.out.Println(sb.capacity());
}
}
```

* difference between String & StringBuffer :

String	StringBuffer
* Immutable	* Mutable
* String is slow & consumes more memory when we concatenate too many strings because every time it creates new instance.	* String buffer is fast & consumes less memory when we concatenate the strings.
* String class overrides the equals() method of object class.	* It doesn't override the equals() method of object class.
* slower while performing concatenatn operation.	* faster while performing concatenatn operation.
* String class uses string constant Pool.	* Uses heap memory.

* Advantage of using StringBuffer over String :

1. Mutable : easily modify.
2. Efficient : as they are mutable
3. Thread safe : which means multiple threads can access it simultaneously. In contrast string objects are not thread safe, which means that you need to use synchronization if you want to access a string object from Multiple Thread.

* String and StringBuffer :

```
public class concatTest
{
 public static void main String concatWithString()
 {
   String t = "Java";
  for (int i=0; i<10000; i++)
 {
   t = t + "Tpoint";
 }
 return t;
}
 public static String concatWithStringBuffer()
 {
   StringBuffer sb = new StringBuffer ("Java");
  for (int i=0; i<10000 ; i++)
 {
   sb.append ("Tpoint");
 }
  return sb.toString();
}
 public static void main (String args[])
 {
  pa long startTime = System.currentTimeMillis();
   concatWithString();
System.out.Println("Time taken by concating with StringBuffer:"
                    + (System.currentTimeMillis()- startTime
                                + ms");
  startTime = System.currentTimeMillis();
   concatWithStringBuffer();
System.out.println("Time taken by concating with StringBuffer:"+
          (System.currentTimeMillis()- startTime)+ms");
  }
}
```

* Difference between StringBuffer and StringBuilder :

StringBuffer	StringBuilder
* Synchronized, i.e. thread safe	* non synchronized, i.e. not thread safe.
* less efficient	* more efficient
* was introduced in Java 1.0	* was introduced in Java 1.5

* Example of StringBuffer :

```
public class BufferTest
{
  public static void main (String args[])
  {
    StringBuffer buffer = new StringBuffer ("hello");
    buffer.append ("java");
    System.out.Println (buffer);
  }
}
```

* Example of StringBuilder :

```
public class BuilderTest
{
  public static void main (String args[])
  {
    StringBuilder builder = new StringBuilder ("hello");
    builder.append ("java");
    System.out.Println (builder);
  }
}
```

* How to create immutable class ?

ImmutableDemo.java

```
public final class Employee
{
 final String pancardNumber;
 public Employee (String pancardNumber)
{
 this.pancardNumber = pancardNumber;
 }
 public String getPancardNumber()
{
 return pancardNumber;
 }
}
public class ImmutableDemo
{
 public static void main (String args[])
{
 Employee e = new Employee ("ABC123");
String s1 = e.getpancardNumber();
System.out.Println ("Pancard Number :" + s1);
 }
}
```

* The class is immutable because :

1. We cannot change the value of class after creating an object.
2. class is final so we cannot create the subclass.
3. We have no option to change the value of the instance variable.

* Rules to create immutable class :

1. Don't allow other classes to override the method.
2. Don't allow to create subclass.
3. All fields should be private & final.
4. Use parameterized constructor to initialize fields.
5. Don't allow direct access to mutable instance variable.
6. Don't provide method to update field.