> How to understand topic:

```
A) need of concept

B) how to use it properly

C) implement it in real time
```

> List of coverage :

Basic Concepts:

1. **What is a String:** Explain that a string is a sequence of characters in Java, represented by the `String` class.
2. **String Literal vs. String Object:** Discuss the difference between creating strings using string literals (e.g., `"Hello"`) and creating `String` objects using the `new` keyword.
3. **Immutability:** Emphasize that strings are immutable in Java, meaning their values cannot be changed after creation.
4. **Concatenation:** Describe string concatenation using the `+` operator and `concat()` method.
5. **String Length:** Explain how to get the length of a string using the `length()` method.
6. **Indexing and Character Access:** Discuss how to access individual characters in a string using index positions and the `charAt()` method.

Intermediate Concepts:

7. **String Comparison:** Explain how to compare strings using the `equals()` method for content-based comparison and `compareTo()` for lexicographical comparison.
8. **String Manipulation:** Introduce common string manipulation methods like `substring()`, `toUpperCase()`, `toLowerCase()`, and `trim()`.
9. **String Searching:** Discuss methods like `indexOf()`, `lastIndexOf()`, and `contains()` for searching substrings within a string.
10. **String Splitting:** Describe how to split a string into an array of substrings using `split()`.
11. **String Formatting:** Introduce string formatting options using `String.format()` or `printf()`-style formatting.

Advanced Concepts:

12. **StringBuilder and StringBuffer:** Discuss the `StringBuilder` (non-thread-safe) and `StringBuffer` (thread-safe) classes for efficient string manipulation when concatenating or modifying strings

repeatedly.

13. **Regular Expressions:** Explain the use of regular expressions (`Pattern` and `Matcher` classes) for advanced string searching and manipulation.
14. **String Interning:** Describe string interning, where string literals with the same content share the same memory location, and how to explicitly intern strings using the `intern()` method.
15. **Unicode and Character Encoding:** Discuss Unicode representation in Java, the `char` data type, and character encoding using classes like `Charset` and `StringEncoder`.
16. **String Pool:** Explain the concept of the string pool, where string literals are stored to improve memory efficiency and reduce duplicate strings.
17. **String Performance:** Discuss performance considerations when working with strings, including the impact of immutability and the use of `StringBuilder` in performance-critical scenarios.
18. **String Deduplication:** Mention Java 8's string deduplication feature, which reduces memory usage by automatically sharing string instances with identical content using the `G1 Garbage Collector`.
19. **Compact Strings (Java 9+):** Discuss Java 9's introduction of compact strings, a memory optimization to represent strings more efficiently when they contain only single-byte characters.
20. **Text Blocks (Java 13+):** Introduce text blocks, a feature introduced in Java 13 for more readable and multiline string literals.
21. **String Methods (Java 11+):** Mention new string methods introduced in Java 11, such as `isBlank()`, `strip()`, `stripLeading()`, and `stripTrailing()` for improved string handling.
22. **Record and Text Blocks (Java 16+):** Explain how Java 16 introduced text blocks as a preview feature and how they can be used in combination with record types for concise code.

Intro.

- Other language - string in character arrays,In java String is Type of object.
- why : to make string handling convinient
- some features it makes easy such as
  - compare two strings
  - concatenate string
  - find substring
  - make uppercase or lowercase
- java.lang has three classes
  - String
  - StringBuffer
  - StringBuilder
- thus accessible to all programs automatically

- Among them String is immutable other two can be changeable
- All three implements "CharSequence" interface -All are declared final, which means that none of these classes may be subclassed
- **Important to note** :
  - the strings within objects of type String are unchangeable means that ***the contents of the String instance cannot be changed after it has been created.***
  - However,a ***variable declared as a String reference can be changed to point at some other String object at any time***.

Important summary

- Use String class : when you need to work with an immutable sequence of characters.
- Use StringBuffer : when you need to work with mutable sequence of characters *in Multithreaded Environment (synchronized)*
- Use StringBuilder : when need to work with a mutable sequence of characters in a single-threaded environment.

```
/*
Constructors in java
*/

String str =  new String(); // Instance of String class with no characters

// String(char[])
Char[] chars= {'T','E','J','A','S'};
String str1 = new String(chars); // TEJAS

// String(Char char[],int startIndex,int numberOfChars)
 String strWithSubrange =  new String(chars,2,3);
 // Output = "JAS"
```

```
/*
Constructors in java
*/

// String (String strObj);
char[] s = {'A','S','T'}
String str1 = new String(s)
String str2 = new String(str1);
```

---

constructors with ascii i.e array of byte params

- String(byte asciiChars[ ])
- String(byte asciiChars[ ], int startIndex, int numChars)
- example

```
// Construct string from subset of char array.
class SubStringCons {
public static void main(String args[]) {
byte ascii[] = {65, 66, 67, 68, 69, 70 };
String s1 = new String(ascii);
System.out.println(s1);
String s2 = new String(ascii, 2, 3);
System.out.println(s2);
}
}

/*
output :
ABCDEF
CDE
*/
```

constructor with stringbuffer params

- String(StringBuffer strBufObj)
- same can be done with StringBuilder object param
- String(StringBuilder strBuildObj)

constructor with unicode code points

- **note** : Read more on internet

```
String(int codePoints[],int startIndex,int numChars)
```

- The codePoints array contains the Unicode code points to decode, and the startIndex and numChars parameters allow us to specify which subset of the array to decode. The resulting string contains all of the characters represented by the specified code points, including any supplementary characters.

Common APIs used in String Library

API 1 : length()

---

length()

use of literals

```
// example lines
String s2 = "abc";

System.out.println("abc".length());
```

use of + operator to concat

```
// Using concatenation to prevent long lines.
class ConCat {
public static void main(String args[]) {
String longStr = "This could have been " +
"a very long line that would have " +
"wrapped around. But string concatenation " +
"prevents this.";
System.out.println(longStr);
}
}
```

- follow precedence table while arithmetic and concat operation

```
String s = "four: " + 2 + 2; // four:22
String s = "four: " + (2 + 2); // four:4
```

API 2 : toString() and valueOf()

- toString Gives String representaion of objects of classess
- default is seldom suffice .must overide for meaningful string representation
- valueOf gives primitive to String representaion as well as when object to String it calls toString method.
- String toString();

```


// Override toString() for Box class.
class Box {
double width;
double height;
double depth;
Box(double w, double h, double d) {
width = w;
height = h;
depth = d;
}
```

~~~
public String toString() {
return "Dimensions are " + width + " by " +
depth + " by " + height + ".";
}
}

```java
class toStringDemo {
public static void main(String args[]) {
Box b = new Box(10, 12, 14);
String s = "Box b: " + b; // concatenate Box object
System.out.println(b); // convert Box to string
System.out.println(s);
}
}
/*
The output of this program is shown here:
Dimensions are 10.0 by 14.0 by 12.0
Box b: Dimensions are 10.0 by 14.0 by 12.0
*/
~~~

API 3 : character extraction

- charAt()
- getChars()

```java
class CharacterExtraction {
    public static void main(String[] args) {

    }

    public static void main1(String[] args) {
        // charAt()
        System.out.println("Print 1st (i.e n-1 index)letter of string tejas using 
charAt  ---> " + "tejas".charAt(0));
        System.out.println("Print 2nd letter of string tejas using charAt  ---> "
+ "tejas".charAt(1));

        System.out.println("Print 3rd letter of string tejas using charAt  ---> "
+ "tejas".charAt(2));
        System.out.println("Print 4th letter of string tejas using charAt  ---> "
+ "tejas".charAt(3));

        // getChars()
        String str = "Desire is a contract that you make with yourself.";
        int start1 =10;
        int end1 = 25;
        char buf1[] = new char[end1 - start1];
        str.getChars(start1, end1, buf1, 0);
        System.out.println(buf1);
```

```
    }
}
```

- getBytes()

```
 public static void main1(String[] args) {
        // charAt()
        System.out.println("Print 1st (i.e n-1 index)letter of string tejas using charAt  ---> " + "tejas".charAt(0));
        System.out.println("Print 2nd letter of string tejas using charAt  ---> " + "tejas".charAt(1));

        System.out.println("Print 3rd letter of string tejas using charAt  ---> " + "tejas".charAt(2));
        System.out.println("Print 4th letter of string tejas using charAt  ---> " + "tejas".charAt(3));

        // getChars()
        String str = "Desire is a contract that you make with yourself.";
        int start1 =10;
        int end1 = 25;
        char buf1[] = new char[end1 - start1];
        str.getChars(start1, end1, buf1, 0);
        System.out.println(buf1);
    }
```

- toCharArray()

API 4 : character comparison

- equals() and equalsIgnoreCase()
- equals() vs ==
- regionMatches()
- startsWith() and endsWith()
- compareTo() and compareToIgnoreCase() ***

API 5 : Searching String

- indexOf()
  - int indexOf(int ch)
  - int lastIndexOf(int ch)
  - int indexOf(Strign str)
  - int lastIndexOf(String str)
  - int indexOf(int ch,int startIndex)
  - int lastIndexOf(int ch,int startIndex)

API 6 : modifying String

- substring()

- String substring(int startIndex); // gives sub string from starting index to end of invoking string
- String substring(int startIndex, int endIndex) // gives sub string from starting index to endIndex not including end index
- **CODE TO UNDERSTAND**

```
public class SubStringDemo {
    public static void main(String[] args) {
        String s = "This is a test. This is, too"; // Original string
        String search = "is"; // String to search for
        String replace = "was"; // String to replace the search string with
        String result = ""; // Variable to store the resulting string

        // Demonstrating substring extraction
        System.out.println("Tejashree".substring(4)); // Extract substring
from the 4th index (5th letter) onwards
        System.out.println("Tejashree".substring(4, 8)); // Extract
substring from index 4 to 7
        System.out.println("Tejashree".substring(0, 2)); // Extract
substring from index 0 to 1

        System.out.println("------ ### ------");

        int i;
        // Search for occurrences of the search string within the original
string
        do {
            System.out.println(s);

            i = s.indexOf(search); // Find the index of the first occurrence
of the search string

/*
* If a match is found (i.e., i != -1), the code performs the replacement by
constructing a new string result. The substring before the matched string is
extracted using substring(0, i), and the replace string is appended. The
remaining portion of the original string after the matched string is
concatenated using substring(i + search.length()).
*/
            if (i != -1) {
                result = s.substring(0, i); // Extract the substring before
the matched string

                result = result + replace; // Append the replacement string

                result = result + s.substring(i + search.length()); //
Append the remaining portion of the original string

                s = result; // Update the current string with the result
                System.out.println("RESULT END: " + result);
            }
        } while (i != -1);
    }
}
```

- Output :

```
shree
shre
Te
------ ### ------
This is a test. This is, too
i : 2
RESULT 1 : Th
RESULT 2 : Thwas
RESULT 3: Thwas is a test. This is, too
RESULT END : Thwas is a test. This is, too
Thwas is a test. This is, too
i : 6
RESULT 1 : Thwas
RESULT 2 : Thwas was
RESULT 3: Thwas was a test. This is, too
RESULT END : Thwas was a test. This is, too
Thwas was a test. This is, too
i : 20
RESULT 1 : Thwas was a test. Th
RESULT 2 : Thwas was a test. Thwas
RESULT 3: Thwas was a test. Thwas is, too
RESULT END : Thwas was a test. Thwas is, too
Thwas was a test. Thwas is, too
i : 24
RESULT 1 : Thwas was a test. Thwas
RESULT 2 : Thwas was a test. Thwas was
RESULT 3: Thwas was a test. Thwas was, too
RESULT END : Thwas was a test. Thwas was, too
Thwas was a test. Thwas was, too
i : -1
```

- concat()
  - String concat(String str)
  - String s = "Teja"; s.concat("Bhai"); // TejaBhai
- replace()
  - String replace(char original,char replacement);
    - "Hello".replace('l','z'); Hezzo
  - String replace(CharSequence original,CharSequence replacement);
- trim()

  - String.trim();
  - removes leading and trailing spaces

  ```
  Strings t = "  Tejas Space De ";
  t.trim(); // removes spaces
  //output -->  Tejas Space De
  ```

- strip()

  - jdk11 introduced
  - String strip();
  - String stripLeading();
  - String stripTrailing();
  - removes all white spaces within string

```
    Strings t = "  Tejas Space De ";
    t.strip(); // removes spaces
    //output -->  TejasSpaceDe
```

valueOf()

- converts data from its internal representation into a human readable form

- It is static method overloaded within String for all Java's built in types so that ***each type can be converted properly in to string.***

- constructors

  - static String valueOf(double num);
  - static String valueOf(long num);
  - static String valueOf(char chars[]);
  - static String valueOf(Object ob);

- when we pass Object as argument then in result Object's toString() method called and returns result, you could just called toString directly

API 7 : changing the case

- String toUpperCase()
- String toLoweCase()

**API 8 : Joininng Strings

- static String join(CharSequence delimeter CharSequence.... strs);
- example demo of join()

```
package com.tejas.kk.string;

public class StringJoinDemo {

  public static void main(String args[]) {
    // Using String.join() to concatenate strings with a delimiter
    String res = String.join(" : ", "absfd", "33333", "area", "curve");

    // Printing the joined string
    System.out.println("String after join " );
    System.out.println(res );
  }
}


// output
String after join
absfd : 33333 : area : curve
```

> Other used API

- split()
- matches()
- replaceAll() vs replaceFirst()

> difference between equals and contentEquals()

- equals() is used to compare two String objects directly, while contentEquals() is used to compare a String object with any other object that implements the CharSequence interface.
- example :

```

String str1 = "Hello";
String str2 = "Hello";
String str3 = "World";

boolean result1 = str1.equals(str2); // true
boolean result2 = str1.equals(str3); // false

//////////

String str1 = "Hello";
StringBuffer buffer1 = new StringBuffer("Hello");
StringBuffer buffer2 = new StringBuffer("World");

boolean result1 = str1.contentEquals(buffer1); // true
boolean result2 = str1.contentEquals(buffer2); // false

```

About StringBuffer class

- It supports modifiable string
- represents Growable and writable charSequence
-

constructors

- StringBuffer( )
- StringBuffer(int size)
- StringBuffer(String str)
- StringBuffer(CharSequence chars)

API

- length()
- capacity()
- ensureCapacity()
- setLength()
- charAt()
- ***setCharAt()***
- deleteCharAt()
- getChars()
- ***append()***
- insert()
    - Constructors
        - StringBuffer insert(int index, String str)
        - StringBuffer insert(int index, char ch)
        - StringBuffer insert(int index, Object obj)
- reverse()
- replace()
    - ex.

```java
public static void main(String args[]) {

    StringBuffer sbuff =  new StringBuffer("I Java !");
    sbuff.insert(2, "respect__");
    System.out.println(sbuff);
    System.out.println(sbuff.reverse());
    System.out.println(sbuff.reverse().replace(2, 9,"Like"));

    }
output :
I respect__Java !
! avaJ__tcepser I
I Like__Java !
```

- indexOf() & lastIndexOf()
- subString()
- trimToSize()

SpringBuilder

- similar to SpringBuffer Except one difference, IT IS NOT SYNCHRONISED ,which means not thread safe
- when mutable string accessed by multiple threads give preference to StringBuffer only !!
- advantage: faster performance

UNCODE related

> About code point and its method

- A code point is a value that represents a single character in the Unicode standard
- allows representation of a wide range of characters from different scripts and languages.
- Deals while internationalization and text manupulation
  - Method 1 :int codePoint(int i)
    - returns unicode code point at location specified by i
    - similar line method ***codePointBefore***() gives unicode code point location that precedes.
    - ***codePointCount***() gives number of code points within invoking string portion start and end-1
  - Method 3 : offsetByCodePoints()

    ```
    String str = "Hello, 世界!";
    int startIndex = 0; // Starting position
    int offset = 7; // Offset by 7 code points

    int newIndex = str.offsetByCodePoints(startIndex, offset);
    System.out.println("New index: " + newIndex); // Output: New index: 10
    ```

    - while dealing with text processing ,need to extract a substring based on code point offsets. By using offsetByCodePoints(), you can accurately determine the start and end indices of the desired substring.