String

- It is a final class declared in java.lang package.
- It is a sub class of java.lang.Object class which implements following interfaces:
    1. CharSequence
    2. Comparable
    3. Serializable
- Serializable is a marker/tagging interface i.e it doesn't contain any method.
- "public int compareTo(String other)" is a method of Comparable interface. It makes String sortable.
- CharSequence I/F provide methods to read char/chars from string. Following are methods of CharSequence I/F:
    1. int length()
    2. char charAt(int index)
    3. CharSequence subSequence(int start, int end)
- String is a class in Java hence it is considered as non primitive / reference type.
- Even though it it non primitive type, we can create its instance using new as well as without new operator.
- Consider following code snippet:

```
class Program{
    public static void main(String[] args) {
        String s1 = new String("Sandeep");  //OK
        String s2 = "Sandeep";  //OK
    }
}
```

- For the simplicity, let us call "Sandeep" as a String literal and "new String("Sandeep")" as a String instance.
- String literals are designed to share and it gets space on String literal pool / String pool. String instance get space on heap section.
- Let us see, how "Sandeep" is interpreted by JVM:

```
class Program{
    public static void main(String[] args) {
        String str = "Sandeep";

        //is equivalent to:

        char[] data = { 'S','a','n','d','e','e','p' };
        String str = new String( data );
    }
}
```

- char is a primitive type whereas Character is a Wrapper class (non primitive type).

- Character information is based on the Unicode Standard.
- For more information of unicode please visit: http://www.unicode.org.
- String is collection of character object which do not ends with '/0' i.e null character.

```
class Program{
    public static void main(String[] args) {
        String str = "SunBeam";
        char ch = str.charAt( 0 );  //S
        ch = str.charAt( str.length() - 1 );  //m
        ch = str.charAt( str.length() );
//StringIndexOutOfBoundsException
    }
}
```

- Using illegal index, if we try to access character from String then string method throws StringIndexOutOfBoundsException.
- Let us see, how to concatnate data to the String.

```
class Program{
    public static void main(String[] args) {
        int code = 46;
        String s1 = "SunBeam";
        String s2 = "Pune";
        String s3 = s1.concat( s2 );    //OK
        String s4 = s2.concat( code );    //Not OK
        String s5 = s2 + code;    //OK
    }
}
```

- Using concat() method, we can concat String to another String but using + operator we can concat state of any primitive / non primitive instance to the String.
- The Java language provides special support for the string concatenation operator (+), and for conversion of other objects to strings.

```
class Program{
    public static void main(String[] args) {
        String s1 = "SunBeam" + "Pune";    //OK
        String s2 = "Pune-"+46; //OK
        String s3 = "System Date : "+new Date();//OK
    }
}
```

- String objects are immutable. In other words, Strings are constant; their values cannot be changed after they are created.

```java
class Program{
    public static void main(String[] args) {
        String str = "SunBeam"; //Line 1
        str = str + "Pune"; //Line 2
    }
}
```

- In the above code snippet, at Line 1, str contains reference of String whose state is "SunBeam". At Line 2, str contains reference of new String instance whose state is "SunBeamPune".

```java
class Program{
    public static void main(String[] args) {
        String s1 = "SunBeam";
        s1.concat("Pune");
        System.out.println(s1); //SunBeam

        String s2 = s1.concat("Pune");
        System.out.println(s2); //SunBeamPune
    }
}
```

- Summary of methods of String class:

1. public int compareToIgnoreCase(String str)
2. public String concat(String str)
3. public boolean startsWith(String prefix)
4. public boolean endsWith(String suffix)
5. public static String format(String format, Object... args)
6. public byte[] getBytes()
7. public int indexOf(int ch)//Overloaded
8. public int lastIndexOf(int ch)//Overloaded
9. public String intern()
10. public boolean isEmpty()
11. public boolean matches(String regex)
12. public String[] split(String regex)
13. public String substring(int beginIndex)//Overloaded
14. public String toLowerCase()
15. public String toUpperCase()
16. public char[] toCharArray()
17. public String trim()
18. public static String valueOf(char c) //Overloaded

- Constructors:

1. public String()

```
String str = new String();
```

2. public String(String original)

```
String str = new String("Java");
```

3. public String(char[] value)

```
char[] data = { 'A', 'B', 'C' };
String str = new String( data );
```

4. public String(byte[] bytes)

```
byte[] bs = {65,66,67};
String str = new String(bs);
```

5. public String(StringBuffer buffer)

```
StringBuffer sb = new StringBuffer("Sandeep");
String str = new String(sb);
```

6. public String(StringBuilder builder)

```
StringBuilder sb = new StringBuilder("Sandeep");
String str = new String(sb);
```

String twisters

- Example 1

```
class Program{
    public static void main(String[] args) {
        String s1 = new String("CDAC");
        String s2 = new String("CDAC");
        if( s1 == s2 )
            System.out.println("Equal");
        else
            System.out.println("Not Equal");
        //Output : Not Equal
    }
}
```

- Example 2

```
class Program{
    public static void main(String[] args) {
        String s1 = new String("CDAC");
        String s2 = new String("CDAC");
        if( s1.equals(s2) )
            System.out.println("Equal");
        else
            System.out.println("Not Equal");
        //Output : Equal
    }
}
```

- Example 3

```
class Program{
    public static void main(String[] args) {
        String s1 = "SunBeam";
        String s2 = "SunBeam";
        if( s1 == s2 )
            System.out.println("Equal");
        else
            System.out.println("Not Equal");
        //Output : Equal
    }
}
```

- Example 4

```
class Program{
    public static void main(String[] args) {
        String s1 = "SunBeam";
        String s2 = "SunBeam";
        if( s1.equals(s2) )
            System.out.println("Equal");
        else
            System.out.println("Not Equal");
        //Output : Equal
    }
}
```

- Example 5

```
class Program{
    public static void main(String[] args) {
        String s1 = "Sandeep";
        String s2 = new String("Sandeep");
        if( s1 == s2 )
            System.out.println("Equal");
        else
            System.out.println("Not Equal");
        //Output : Not Equal
    }
}
```

- Example 6

```
class Program{
    public static void main(String[] args) {
        String s1 = "Sandeep";
        String s2 = new String("Sandeep");
        if( s1.equals(s2) )
            System.out.println("Equal");
        else
            System.out.println("Not Equal");
        //Output : Equal
    }
}
```

- Example 7

```
class Program{
    public static void main(String[] args) {
        String s1 = "San"+"deep";
        String s2 = "Sandeep";
        if( s1 == s2 )
            System.out.println("Equal");
        else
            System.out.println("Not Equal");
        //Output : Equal
    }
}
```

- Constant expressions get evaluated at compile time where as non constant expressions gets evaluated at runtime.

```
int a = 10;
int b = 20;
int c = 10 + 20;    //at compile time it is 30
int d = a + b;    //at runtime time it is 30
```

- Example 8

```
class Program{
    public static void main(String[] args) {
        String str = "San";
        String s1 = str+"deep";
        String s2 = "Sandeep";
        if( s1 == s2 )
            System.out.println("Equal");
        else
            System.out.println("Not Equal");
        //Output : Not Equal
    }
}
```

- Example 9

```
class Program{
    public static void main(String[] args) {
        String str = "San";
        String s1 = (str+"deep").intern();
        String s2 = "Sandeep";
        if( s1 == s2 )
            System.out.println("Equal");
        else
            System.out.println("Not Equal");
        //Output : Equal
    }
}
```

- intern() method returns reference from String pool.

```
//File : A.java
package p1;
public class A{
    public static String s1 = "SunBeam";
}
```

```
//File : Program.java
package p2;
import p1.A;
class B{
    public static String s2 = "SunBeam";
}
class Program{
    public static String s3 = "SunBeam";
    public static void main(String[] args) {
        String s4 = "SunBeam";
        System.out.println(A.s1 == B.s2);           //true
        System.out.println(A.s1 == Program.s3);     //true
        System.out.println(A.s1 ==s4);              //true
        System.out.println(B.s2 == Program.s3);     //true
        System.out.println(B.s2 ==s4);              //true
        System.out.println(A.s1 == Program.s3);     //true
        System.out.println(Program.s3 ==s4);        //true
    }
}
```

A Strategy for Defining Immutable Objects

```
1. Don't provide "setter" methods — methods that modify fields or objects
referred to by fields.
2. Make all fields final and private.
3. Don't allow subclasses to override methods. The simplest way to do this
is to declare the class as final. A more sophisticated approach is to make
the constructor private and construct instances in factory methods.
4. If the instance fields include references to mutable objects, don't
allow those objects to be changed:
    - Don't provide methods that modify the mutable objects.
    - Don't share references to the mutable objects.
```

Benefits of programming with immutable objects.

```
1. Immutable objects are thread-safe so you will not have any
synchronization issues.
2. Immutable objects are good Map keys and Set elements, since these
typically do not change once created.
3. Immutability makes it easier to write, use and reason about the code.
4. Immutability makes it easier to parallelize your program as there are
no conflicts among objects.
5. The internal state of your program will be consistent even if you have
exceptions.
6. References to immutable objects can be cached as they are not going to
change.
```

StringBuffer and StringBuilder

Similarity

- Both are final classes declared in java.lang package.
- Both are sub classes of Object class and implements Serializable, CharSequence interface
- In both the classes equals and hashCode method is not overriden.
- Both are used to create mutable String objects.

Difference

- Implementation of StringBuffer is thread-safe whereas Implementation of StringBuilder is not thread-safe.
- StringBuffer is introduced in jdk 1.0 and StringBuilder in jdk1.5

Methods of StringBuffer

1. public StringBuffer append(String str) //Overloaded
2. public int capacity()
3. public char charAt(int index)
4. public int indexOf(String str)
5. public int lastIndexOf(String str)
6. public int length()
7. public StringBuffer reverse()
8. public String substring(int start, int end)

Program to reverse number

```
class Program{
    public static Scanner sc = new Scanner(System.in);
    public static void main(String[] args) {
        System.out.println("Number  :   ");
        int num1 = sc.nextInt();
        String str = String.valueOf( num1 );
        StringBuilder sb = new StringBuilder( str );
        sb.reverse();
        str = sb.toString();
        int num2 = Integer.parseInt( str );
        System.out.println("Number  :   "+num2);
    }
}
```

StringTokenizer

How to split string?

- Example 1

```
class Program{
    public static void main(String[] args) {
        String str = "SunBeam Infotech Pune";
        String regex = " ";
        String[] words = str.split( regex );
        for( String word : words ){
            System.out.println( word );
        }
    }
}
```

- Example 2

```
class Program{
    public static void main(String[] args) {
        String str = "www.sunbeaminfo.com";
        String regex = "\\.";
        String[] words = str.split( regex );
        for( String word : words ){
            System.out.println( word );
        }
    }
}
```

- StringTokenizer is a sub class of Object class and it implements Enumeration interface.
- The string tokenizer class allows an application to break a string into tokens.
- Methods of Enumeration I/F
    1. boolean hasMoreElements()
    2. E nextElement()
- Methods of StringTokenizer
    1. public int countTokens()
    2. public boolean hasMoreTokens()
    3. public String nextToken()
- Constructors
    1. public StringTokenizer(String str)

```
String str = "SunBeam Infotech Pune";
StringTokenizer stk = new StringTokenizer( str );
```

```
2. public StringTokenizer(String str, String delim)
```

```
String str = "www.yahoo.com";
String delim = ".";
StringTokenizer stk = new StringTokenizer( str,delim );
```

```
3. public StringTokenizer(String str, String delim, boolean returnDelims)
```

```
String str = "www.yahoo.com";
String delim = ".";
boolean returnDelims = true;
StringTokenizer stk = new StringTokenizer( str,delim, returnDelims );
```

- Example 3

```
class Program{
    public static void main(String[] args) {
        String str = "SunBeam Infotech Pune";
        StringTokenizer stk = new StringTokenizer( str );
        String token;
        while( stk.hasMoreTokens()){
            token = stk.nextToken();
            System.out.println( token );
        }
    }
}
```

- Example 4

```
class Program{
    public static void main(String[] args) {
        String str = "www.sunbeaminfo.com";
        String delim = ".";
        StringTokenizer stk = new StringTokenizer( str, delim );
        String token;
        while( stk.hasMoreTokens()){
            token = stk.nextToken();
            System.out.println( token );
        }
    }
}
```

Pattern and Matcher

- These are final classes declared in java.util.regex package.
- If we want to process string then we should use regular expression.
- Instance of pattern class represent compiled regular expression.
- A Matcher instance is an engine that performs match operations on a string by interpreting a Pattern.
- Example

```
String regex = "a*b", input = "aaaaab";
Pattern p = Pattern.compile(regex);
Matcher m = p.matcher( input );
boolean b = m.matches();
```

- Example

```
String regex = "a*b", input = "aaaaab";
 boolean b = Pattern.matches(regex, input);
```

- Example

```
String regex = "a*b", input = "aaaaab";
boolean b = input.matches(regex);
```