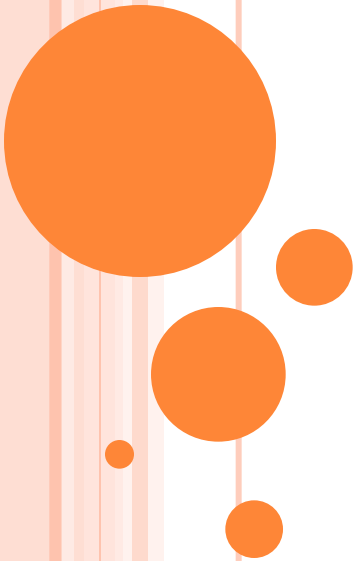

String Class

Tanjina Helaly

String

- Is a final class in java.lang pacakage
- Example 1:

  String s = new String();
- Example 2:

  char chars[] = { 'a', 'b', 'c' };

  String s = new String(chars);

  This constructor initializes **s with the string "abc".**
- Example 3:

  String s = new String(“Hello World”);
- Example 4: Simplified literal version

  String s = “Hello World”;

Note: Before Java 7 - Can’t use String in Switch

String Methods

Modifier and Type	Method and Description
char	charAt(int index) Returns the char value at the specified index.
int	compareTo(String anotherString) Compares two strings lexicographically.
int	compareToIgnoreCase(String str) Compares two strings lexicographically, ignoring case differences.
String	concat(String str) Concatenates the specified string to the end of this string.
boolean	endsWith(String suffix) Tests if this string ends with the specified suffix.
boolean	equals(Object anObject) Compares this string to the specified object.
boolean	equalsIgnoreCase(String anotherString) Compares this String to another String, ignoring case considerations.
byte[]	getBytes() Encodes this String into a sequence of bytes using the platform's default charset, storing the result into a new byte array.
int	indexOf(String str) Returns the index within this string of the first occurrence of the specified substring.
boolean	isEmpty() Returns true if, and only if, length() is 0.
int	lastIndexOf(String str) Returns the index within this string of the last occurrence of the specified substring.

String Methods

Modifier and Type	Method and Description
int	length() Returns the length of this string.
String	replaceAll(String regex, String replacement) Replaces each substring of this string that matches the given regular expression with the given replacement.
String[]	split(String regex) Splits this string around matches of the given regular expression.
boolean	startsWith(String prefix) Tests if this string starts with the specified prefix.
String	substring(int beginIndex) Returns a new string that is a substring of this string.
String	substring(int beginIndex, int endIndex) Returns a new string that is a substring of this string.
String	toLowerCase() Converts all of the characters in this String to lower case using the rules of the default locale.
String	toUpperCase() Converts all of the characters in this String to upper case using the rules of the default locale.
String	trim() Returns a copy of the string, with leading and trailing whitespace omitted.
static String	valueOf(double d) Returns the string representation of the double argument.

String Concatenation

- String concatenation is the process of joining two or more small String to create a big String.
- 4 ways to do concatenation
  - Concatenation operator (+)
  - StringBuffer class
  - StringBuilder class
  - String.concat() function

- **For example,**
  - The following fragment concatenates three strings and produce output “He is 9 years old”.

```
String age = "9";
String s = "He is " + age + " years old.";
System.out.println(s);
```

String Concatenation

```
public class StringConcat{
public static void main(String args[]){
    String firstname = “Tareq”, lastname = “Mahmud";

    // 1st way - Use + operator to concatenate String
    String name = firstname + " " + lastname;
    System.out.println(name);

    // 2nd way – by concat() method
    name = firstname.concat(lastname);
    System.out.println(name);

    // 3rd way - by using StringBuilder
    StringBuilder sb = new StringBuilder(14);
    sb.append(firstname).append(" ").append(lastname);
    System.out.println(sb.toString());

    // 4th way - by using StringBuffer
    StringBuffer sBuffer = new StringBuffer(15);
    sBuffer.append(firstname).append(" ").append(lastname);
    System.out.println(sBuffer.toString());
    }
}
```

Output:
Tareq Mahmud
Tareq Mahmud
Tareq Mahmud
Tareq Mahmud

Performance:
StringBuilder
StringBuffer
String concat
Concat operator

+ OPERATOR

- Works from left to right
- + is overloaded operator. For example, for numeric value it will add the number
- Any data after the first String will automatically converted to String

Concatenation example	Output
“My age” + 2 + 2	
2 + 2 + “ years old”	
“My age” + (2+2)	

String Comparison

- equals(), equalsIgnoreCase() - Return boolean
- startsWith( ) and endsWith( ) -Return boolean
- compareTo( ) - Return int
  - int compareTo(String *str*)

Value	Meaning
Less than zero	The invoking string is less than *str*.
Greater than zero	The invoking string is greater than *str*.
Zero	The two strings are equal.

  - **Example:**
    - *"A".compareTo("a"); // will return -32*
    - *"a".compareTo("A"); // will return 32*
    - *"A".compareTo("A"); // will return 0*
    - *"A".compareTo("B"); // will return -1*

EQUALS( ) VERSUS ==

- equals( ) method compares the characters inside a String object.
- The == operator compares two object references to see whether they refer to the same instance.

EQUALS( ) VERSUS ==

```
// equals() vs ==
class EqualsNotEqualTo {
      public static void main(String args[]) {
            String s1 = “Hello”;
            String s2 = new String(s1);
            System.out.println(s1 + " equals " + s2 + " -> " +
            s1.equals(s2));
            System.out.println(s1 + " == " + s2 + " -> " + (s1 == s2));
      }
}
```

- the contents of s1 and s2 are identical but they are 2 distinct objects,
  - therefore, not ==, as is shown here by the output of the preceding example:
- Output

  Hello equals Hello -> true

  Hello == Hello -> false

EQUALS( ) VERSUS ==

```
public class StringTest {
    public static void main( String[] args ) {
        String me = "Roger";
        if ( me == "Roger" )
            System.out.println( "Yes, I am me" );
        else
            System.out.println( "No, I am not me?" );
        String shortName = me.substring( 0, 3 );
        System.out.println( shortName );
        if ( shortName == "Rog" )
            System.out.println( "Very Good" );
        else
            System.out.println( "Trouble here" ); //How is this possible?
        if ( shortName.equals( "Rog" ) )
            System.out.println( "Do it this way" );
    }
}
```

- **Output**

  Yes, I am me
  Rog
  Trouble here
  Do it this way

EQUALS( ) VERSUS ==

```
if ( me == "Roger" )
   System.out.println( "Yes, I am me" );
else
   System.out.println( "No, I am not me?" );
```

- Compilers are allowed, but not required to store equal strings in the same memory location.
- Since **me was initialized with a string literal the compiler** was smart enough to store the two strings in the same location.
- In the comparison:

  if (shortName == "Rog")

  - shortName was not initialized with a literal, so the compiler did not know to store in the same location as "Rog". Thus this **comparison is false !**

StringBuffer, StringBuilder

- String is *immutable* (once created can not be changed)object .
  - String once assigned can not be changed.
- Every immutable object in Java is **thread safe** ,that implies String is also thread safe .
- String can not be used by two threads simultaneously.

String demo = " hello " ; // The “hello” object is stored in constant string pool and its value can not be modified.

demo="Bye" ; //new "Bye" string is created in constant pool and referenced by the demo variable
// "hello" string still exists in string constant pool and its value is not overrided but we lost reference to the "hello"string

StringBuffer, StringBuilder

- StringBuffer is mutable means one can change the value of the object .
  - StringBuffer has the same methods as the StringBuilder ,
  - but **each method in StringBuffer is synchronized** that is **StringBuffer is thread safe**.
- StringBuilder is same as the StringBuffer , that is it stores the object in heap and it can also be modified .
  - The main difference between the StringBuffer and StringBuilder is that **StringBuilder is also not thread safe.**
  - StringBuilder is fast as it is not thread safe .

Split method- Example

```
import java.util.StringTokenizer;
public class Simple{
 public static void main(String args[]){
        String st = new String("Welcome to OOP");
        String[] words = st.split(" ");
        for(String s: words)
                System.out.println(s);
   }
}
```

- Output:

  Welcome
  to
  OOP

StringTokenizer

- The **java.util.StringTokenizer** class allows you to break a string into tokens. It is simple way to break string.
- The default delimiter set,
  - " \t\n\r\f":
  - space character, tab character, newline character, carriage-return character, form-feed character.

Constructor and Description
StringTokenizer(String str) Constructs a string tokenizer for the specified string.
StringTokenizer(String str, String delim) Constructs a string tokenizer for the specified string.
StringTokenizer(String str, String delim, boolean returnDelims) Constructs a string tokenizer for the specified string.

StringTokenizer - Methods

Modifier and Type	Method and Description
int	**countTokens**() Calculates the number of times that this tokenizer's nextToken method can be called before it generates an exception.
boolean	**hasMoreElements**() Returns the same value as the hasMoreTokens method.
boolean	**hasMoreTokens**() Tests if there are more tokens available from this tokenizer's string.
Object	**nextElement**() Returns the same value as the nextToken method, except that its declared return value is Object rather than String.
String	**nextToken**() Returns the next token from this string tokenizer.
String	**nextToken**(String delim) Returns the next token in this string tokenizer's string.

StringTokenizer - Example

```
import java.util.StringTokenizer;
public class Simple{
 public static void main(String args[]){
        StringTokenizer st = new StringTokenizer("Welcome to OOP"," ");
        while(st.hasMoreTokens())
                System.out.println(st.nextToken());
   }
}
```

- Output:

  Welcome
  to
  OOP