JAVA @17

String API

Objective

After completing this session you will be able to understand,

- Introduction to String Class
- StringBuffer and StringBuilder classes
- StringTokenizer class
- Define equals() and hashCode() methods.

String class

Strings, which are widely used in Java programming, they *are a sequence of characters*.

- Java provides ***String*** class to create and process strings.
- Strings are ***objects***.

How to create a String?

Option 1: String *greeting* = "Hello world!";

// Create a string literal and assign it to a String reference.

(OR)

Option 2: String *greeting* = new String("Hello world!“);

// Using the String constructor.

About String class

String class is available from ***java.lang*** package.

What does String class contains?

String class contains the APIs used for creating and processing strings.

Example:

- Comparing Strings
- Searching Strings
- Extracting the sub string
- Also constructors for creating String in different ways.

Example – String constructor

Lets develop a program to explore the various String constructors to build a String

```
public class StringConstructDemo {

    public static void main(String[] args) {
        String s1 = new String();
        char chars[] = { 'h', 'e', 'l', 'l', 'o' };
        String s2 = new String(chars);
        byte bytes[] = { 'w', 'o', 'r', 'l', 'd' };
        String s3 = new String(bytes);
        String s4 = new String(chars, 1, 3);
        String s5 = new String(s2);
        String s6 = s2;
        System.out.println(s1);
        System.out.println(s2);
        System.out.println(s3);
        System.out.println(s4);
        System.out.println(s5);
        System.out.println(s6);
    }
}
```

Creating string objects from a character array.

From byte array.

From other char objects.

From other string objects.

String class APIs

Some commonly used APIs inside the String class

Return type	Method	Description
char	**charAt(int index)**	Returns the character at the specified index.
int	**compareTo(String obj)**	Compares two strings lexicographically.
boolean	**equalsIgnoreCase(String str)**	Compares this String to another String, ignoring case considerations.
Boolean	**equals(Object another)**	checks the equality of string with object
int	**indexOf(int ch)**	Returns the index within this string of the first occurrence of the specified character.
int	**length()**	Returns the length of this string.

String class APIs

Return type	Method	Description
String	**concat(String str)**	Concatenates the specified string to the end of this string.
String	**replace(char oldChar, char newChar)**	Returns a new string resulting from replacing all occurrences of oldChar in this string with newChar.
String	**substring(int beginIndex, int endIndex)**	Returns a new string that is a substring of this string.
String	**trim()**	Returns a copy of the string, with leading and trailing whitespace omitted.
String[]	**split(String regex)**	returns splitted string matching regex

Key points

Key points on Strings:

- Strings are immutable i.e. once it is created a String object cannot be changed.
- If you assign a String reference to a new String, the old String will be lost.

Example:

```
String str1="Hello";
str1="World";
Now, str1 contains "World"
```

- All string operations (concatenate, trim, replace, substring) construct and return new strings.
- String class is final.

Example – String APIs

Lets develop a program to explore the few API's of String class, namely,
print a specific character of a string, compare, print length of string, replace a string etc.

```
public class StringImplDemo {
    public static void main (String[] args) {
        String str = "Eyeopen";
        String str1 = "Eyeopen";
        String str2 = "eyeopen";
        String str3 = "         Eyeopen Technologies       ";
        System.out.println("4th Position is "+str.charAt(4));
        System.out.println("Size of string is "+str.length());
        System.out.println("To compare str & str1:: "+str.equals(str1));
        System.out.println("To compare str & str2:: "+str.equals(str2));
        System.out.println("To compare str & str2:: "+str.equalsIgnoreCase(str2));
        System.out.println("To upper case:: "+str2.toUpperCase());
        System.out.println("To lower case:: "+str.toLowerCase());
        System.out.println(str3.trim());
        System.out.println(str3.substring(8));
        System.out.println(str3.substring(2, 5));
        String str4="India,Pakistan,Australia,South Africa,Sri lanka,England";
        String[] country = str4.split(",");
        for(String name:country) {
            System.out.println(name);
        }
    }
}
```

StringBuffer

StringBuffer objects unlike Strings are ***mutable*** i.e. string value can be changed. So string buffer is like a String, that can be ***modified***.

• Using the API's in the StringBuffer the content and the length of the string can be changed without creating a new object.

• The API's of StringBuffer are ***synchronized***.

• This class is preferred when modification of character strings are needed since it is efficient in memory utilization.

Examples:

- Appending String
- Inserting characters in string.
- Deleting characters from a string

StringBuffer class

StringBuffer is a final class.

StringBuffer objects can be created empty,

```
StringBuffer strBuf = new StringBuffer();
```

StringBuffer can be created from a String.

```
StringBuffer strBuf = new StringBuffer(“Bob”);
```

StringBuffer can be created with a capacity.

```
StringBuffer strBuf = new StringBuffer(100);
```

StringBuffer class

String buffers are used by developers to concatenate String rather than using concatenation operator "+" on string objects.

StringBuffer is efficient than "+" concatenation.

Example:

```
String str="Stanford";
str = str+ "University";
```

Can be developed as

```
str = new StringBuffer().append("Standford").append("University").toString();
```

Step 2: Appends the string representation of each operand to the string buffer in turn.

Step 1: Compiler creates a new string buffer initially empty.

Step 3: Converts the contents of the string buffer to a string as ‘str’ is a string object.

StringBuffer APIs

Return type	Method	Description
void	**setCharAt(int index, char ch)**	The character at the specified index of this string buffer is set to the character ch.
StringBuffer	**insert(int offset, String str)**	Inserts the string argument into this string buffer.
StringBuffer	**delete(int start, int end)**	Removes the characters in a substring of this StringBuffer.
StringBuffer	**replace(int start, int end, String str)**	Replaces the characters in a substring of this StringBuffer with characters in the specified String.

StringBuffer APIs

Return type	Method	Description
StringBuffer	**reverse()**	The character sequence contained in this string buffer is replaced by the reverse of the sequence.
StringBuffer	**append(String str)**	Appends the string to this string buffer.
void	**setLength(int newLength)**	Sets the length of this String buffer.
String	**substring(int start, int end)**	Returns a new String that contains a subsequence of characters currently contained in this StringBuffer.

Example – StringBuffer APIs

Lets develop a program to explore the few API's of String Buffer class.

We will solve the following problem,

1. Append two Strings "Hello" & "World" . Output: "Hello World"
2. Insert a string "_Java" in the String after "Hello". Output: "Hello_Java World"
3. Replace _ with space. Output: "Hello Java World".
4. Print the character at the 6'th position . Output: J
5. Delete the character in the third position. Output: "Helo World"
6. Print the capacity of the buffer.
7. Reverse the string and print the string. Output: "dlroW avaJ oleH"

Solution – StringBuffer APIs

```
public class StringBufferDemo {
    public static void main(String[] args) {
    StringBuffer strbuf = new StringBuffer("Hello");
    System.out.print(strbuf.length());
    strbuf.append("World");
    System.out.println(strbuf);
    strbuf.insert(5,"_Java ");
    System.out.println(strbuf);
    strbuf.setCharAt(5,' ');
    System.out.println(strbuf);
    System.out.print("Character at 6th position : ");
    System.out.println(strbuf.charAt(6));
    strbuf.deleteCharAt(3);
    System.out.println(strbuf);
    System.out.print("Capacity of StringBuffer object : ");
    System.out.println(strbuf.capacity());
    strbuf.reverse();
    System.out.print("Reversed string : ");
    System.out.println(strbuf);
    }
}
```

This returns 5.

This returns “HelloWorld” .

This returns “Hello_Java World”.

This returns “Hello Java World”.

This returns ‘J ‘ .

This returns “Helo Java World”.

This returns ‘21 ‘.

This returns “dlroW avaJ oleH“ .

StringBuilder

StringBuilder class, which is a drop-in replacement for StringBuffer.

- StringBuilder is not synchronized which means it is not thread-safe.
- Use StringBuilder class where thread safety is not an issue.
- It offers faster performance than StringBuffer.
- All the methods available on StringBuffer are also available on StringBuilder, so it really is a drop-in replacement.

Example – StringBuilder APIs

Lets develop a program to explore the few API's of String Builder class.

We will solve the following problem,

1. Append two Strings "Hello" & "World" . Output: "HelloWorld"
2. Insert a string "_Java" in the String after "Hello". Output: "Hello_Java World"
3. Replace _ with space. Output: "Hello Java World".

Example – StringBuilder APIs

```
public class StringBufferDemo {
    public static void main(String[] args) {
    StringBuilder strbuf = new StringBuilder("Hello");
    System.out.println(strbuf.length());
    strbuf.append("World");
    System.out.println(strbuf);
    strbuf.insert(5,"_Java ");
    System.out.println(strbuf);
    strbuf.setCharAt(5,' ');
    System.out.println(strbuf);
    }
}
```

This returns 5.

This returns “HelloWorld”.

This returns “Hello_Java World”.

This returns “Hello Java World” .

StringTokenizer class

The ***StringTokenizer*** class is used to break a string into tokens base don some delimiters.

Example: India, USA, UK, Russia – This string can be split based on the delimiter “,”

- It is available in java.util package.
- ***StringTokenizer*** implements the Enumeration interface.
- The given string can be enumerated, you can enumerate the individual tokens contained in it using ***StringTokenizer***.

Tokens from the String:

Token 1- India ***Token 2-*** USA ***Token 3-*** UK ***Token 4-*** Russia

StringTokenizer APIs

The default delimiters are whitespace characters. space, tab, newline, and carriage return.

Return type	Method	Description
boolean	**hasMoreTokens()**	Tests if there are more tokens available from this tokenizer's string.
String	**nextToken()**	Returns the next token in this string tokenizer's string.

Example - StringTokenizer

Lets develop a program to explore the ***StringTokenizer***

```
public class TokenizerDemo {
    public static void main(String[] args) {
        String str = "Eyeopen Technologies, 18th Main Road, Anna Nagar, Chennai.";
        StringTokenizer sToken = new StringTokenizer(str,",");
        while(sToken.hasMoreTokens()) {
            System.out.println(sToken.nextToken());
        }
    }
}
```

Run the program and check the output.

Now execute the same program without using delimiter and see the output.

StringTokenizer sToken = new StringTokenizer(str);

equals() hashCode() method

java.lang.Object has methods called *hashCode() and equals()*.

These methods can be overridden and implemented with the object specific logic,

What is a Hash code?

Hash code is an unique id number allocated to an object by JVM. This number is maintain through the lifecycle of the Object.

hashCode()

- This method by default returns the hash code value of the object on which this method is invoked.
- This method returns the hash code value as an integer.
- You can develop your own logic of generating hash code.

equals() hashCode() method

What is a Equals method used for?

This particular method is used for comparing two objects for equality.

equals()

equals() refers to equivalence relation **i.e.** you say that two objects are equivalent they satisfy the “equals()” condition.

Override the ***equals()*** with a logic which needs to be used for comparing for equivalence.

Example:

Assume we have a object Employee with the following instance variables, EmployeeId, EmployeeName.

The developer can override the equals method and compare the employee Id for checking equivalence.

Example – equals and hashCode

Lets develop a program to explore how equals and hash code works.

Create a *Employee* object with instance variable age and name. Override the hash code and equals method as mentioned below,

- ***Equals*** – The method overridden to compare the age of the employees if same they should return a true else return false.
- ***hashCode*** – Should return the age as hash code.

Solution – equals and hashCode

```
public class Emp {
    private int age ;

    public Emp( int age )
    {
    super();
    this.age = age;
    }

    public int hashCode()
    {
    return age;
    }

    public boolean equals( Object obj )
    {
    boolean flag = false;
    Emp emp = ( Emp )obj;
    if( emp.age == age )
    flag = true;
    return flag;
    }
}
```

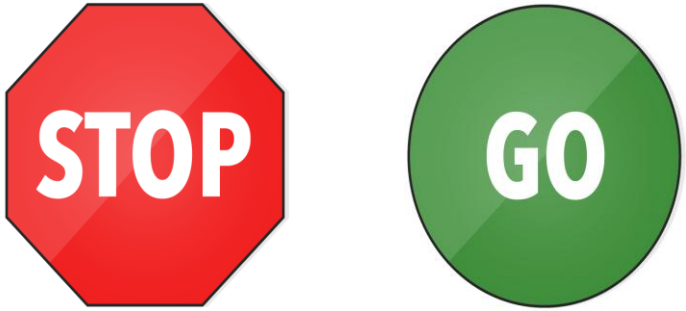
```
public class TestEmp {
    public static void main(String[] args) {
        Emp emp1 = new Emp(23);
        Emp emp2 = new Emp(23);
        System.out.println("emp1.equals(emp2)--->>>"+emp1.equals(emp2));
    }
}
```

Overriding hashcode ().

This returns true.

Overriding equals().

Time To Reflect



Trainees to reflect the following topics before proceeding.

- What makes implementing Runnable interface better than extending Thread class for Thread Creation?
- How can a Thread wait with out finishing for another Thread to get completed?

Thank you

You have successfully completed

String API