UNIT-II

Building Blocks of Language

Abstract

- Introduction of String
- Creation of String
- String Length
- String Concatenation
- Changing case
- Character Extraction
- String Comparison
- StringBuffer

Introduction of String

- Java string is a sequence of characters. They are objects of type String.
- Once a String object is created it cannot be changed. Strings are **Immutable**.
- To get **changeable strings** use the class called **StringBuffer.**
- String and StringBuffer classes are declared final, so there cannot be subclasses of these classes.

Creation of String

- Two ways to create strings:
  - Using String
    - Ex: String s1="Welcome";
  - Using new keyword
    - Ex: String s1=new String("Welcome");

- `String str = "abc";` is equivalent to:

```
char data[] = {'a', 'b', 'c'};
String str = new String(data);
```

- The java String is immutable i.e. it cannot be changed. Whenever we change any string, a new instance is created.

- For mutable string, you can use StringBuffer and StringBuilder classes.

- **Java String** class provides a lot of methods to perform operations on string such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.

- The **java.lang.String** class is used for String.

String Length

- The **java string length()** method used for length of the string. It returns count of total number of characters.

- The **length()** method returns the length of the string.

```
public class LengthExample
{
    public static void main(String args[])
    {
    String s1="Hello";
    System.out.println("string length is: "+ s1.length());
    //5 is the length of Hello string
    }
}
```

String Concatenation

- The **java string concat()** method combines specified string at the end of this string. It returns combined string. It is like appending another string.

- **public** String concat(String anotherString)

- The + operator is used to concatenate two or more strings.

  Ex: String name = “ROHAN”
  String str = “My name is” + name+ “.”;

- For string concatenation the Java compiler converts an operand to a String whenever the other operand of the + is a String object.

```
public class ConcatExample
{
    public static void main(String args[])
    {
    String s1="java string";
    s1.concat("is immutable");
    System.out.println(s1);
    }
}
```

Changing case of String

- **toLowerCase():** Converts all of the characters in a String to lower case.
- **toUpperCase():** Converts all of the characters in this String to upper case.

```
public String toLowerCase()
public String toUpperCase()

Ex: "HELLO THERE".toLowerCase();
   "hello there".toUpperCase();
```

String s1="HELLO! HOW are You";
String s1lower=s1.toLowerCase();

Character Extraction

- Substring
- charAt()
- getChars()

getChar()

- **getChars()** - Copies more than one characters from string into the destination character array.

- public void **getChars**(int srcBeginIndex,
  int srcEndIndex,
  char[] destination,
  int dstBeginIndex)

```
public class StringGetCharsExample
{
    public static void main(String args[])
    {
    String str = new String(" string getchar example is given");
    char[] ch = new char[10];
    try
    {
    str.getChars(6, 16, ch, 0);
    System.out.println(ch);
}
    catch(Exception ex)
    {
    System.out.println(ex);}
}
}
```

charAt()

- **Used to extract Characters at a specific index of string.**

- The **java string charAt()** method returns *a char value at the given index number*. The index number starts from 0.

- It returns **StringIndexOutOfBoundsException** if given index number is greater than this string or negative index number.

public char **charAt**(int index)

– Returns the character at the specified index. An index ranges from 0 to length() - 1. The first character of the sequence is at index 0, the next at index 1, and so on, as for array indexing.

```
char ch;
ch = "abc".charAt(1); // ch = "b"
```

Substring

- The **java string substring()** method returns a part of the string.

- We pass begin index and end index number position in the java substring method where start index is inclusive and end index is exclusive. In other words, start index starts from 0 whereas end index starts from 1.

- There are two types of substring methods in java string.

- **public** String substring(**int** startIndex)

- **public** String substring(**int** startIndex,
  int endIndex)

```
public class SubstringExample
{
    public static void main(String args[])
    {
    String s1=“Warehouse";
    System.out.println(s1.substring(2,4));
    System.out.println(s1.substring(2));
    }
}
```

String Comparison

- We can compare string in java on the basis of content and reference.

- It is used in
  - **authentication** (by equals() method),
  - **sorting** (by compareTo() method),
  - **reference matching** (by == operator) etc.

- There are three ways to compare string in java:
  - **By equals() method**
  - **By = = operator**
  - **By compareTo() method**

Equals() method

- The **String equals()** method compares the original content of the string. It compares values of string for equality. String class provides two methods:

- **public boolean equals(Object another)** compares this string to the specified object.

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

```
class Stringcomp
{
    public static void main(String args[])
    {
    String s1=“Rohini";
    String s2=“Meera";
    String s3=new String(“Rohini");
    String s4=“ROHINI";
    System.out.println(s1.equals(s2));
    System.out.println(s1.equals(s3));
    System.out.println(s1.equals(s4));
    System.out.println(s1.equalsIgnoreCase(s4));
    }
}
```

Using == Operator

- The = = operator compares references not values.

- Ex:

```
String s1=“Hello";
String s2=“Hello";
String s3=new String(“Hello");
System.out.println(s1==s2);
//true (because both refer to same instance)
System.out.println(s1==s3);
//false(because s3 refers to instance created in nonpool)
```

compareTo() method

- The **String compareTo()** method **compares values and returns an integer value** that describes **if first string is less than, equal to or greater than second string.**

- Suppose s1 and s2 are two string variables. If:
  - **s1 == s2** : 0
  - **s1 > s2** : positive value
  - **s1 < s2** : negative value

```
class Stringcomp
{
    public static void main(String args[])
    {
    String s1=“Japan";
    String s2=“Japan";
    String s3=“Pakistan";
    System.out.println(s1.compareTo(s2));
    System.out.println(s1.compareTo(s3));
    System.out.println(s3.compareTo(s1));
    }
}
```