Object Oriented Programming

Topics to be covered today

String Handling continued

Character Extraction

String class provides a number of ways in which characters can be extracted from a String object.

String index begin at zero.

These extraction methods are:

- charAt() -- char
- getChars() -- void
- getBytes() – Byte[]
- toCharArray() – char[]
- **codePointAt**(int index) -- int

charAt() and codePointAt()

To extract a single character from a String.

General form:

- char charAt(int where)

where is the index of the character you want to obtain. The value of *where* must be nonnegative and specify a location within the string.

Example:

- char ch; int a;
- ch = “abc”.charAt(1); //Assigns a value of “b” to ch.
- a=“abc”.codePointAt(1); Assigns a value of 98 to b.

getChars()

Used to extract more than one character at a time.

General form:

void getChars(int sourceStart, int sourceEnd, char[] target, int targetStart)

sourceStart – specifies the index of the beginning of the substring

sourceEnd – specifies an index that is one past the end of the desired subString

target – is the array that will receive the characters

targetStart – is the index within target at which the subString will be copied is passed in this parameter

getBytes()

Alternative to *getChars()* that stores the characters in an array of bytes. It uses the default character-to-byte conversions provided by the platform.

General form:

byte[] getBytes()

Usage:

- Most useful when you are exporting a String value into an environment that does not support 16-bit Unicode characters.
- For example, most internet protocols and text file formats use 8-bit ASCII for all text interchange.

toCharArray()

To convert all the characters in a String object into character array.

It returns an array of characters for the entire string.

General form:

```
char[] toCharArray()
```

It is provided as a convenience, since it is possible to use getChars() to achieve the same result.

Example

```
String s= "Hello World!";
char b=s.charAt(1);       System.out.println(b);

int a=s.codePointAt(1);       System.out.println(a);

char result1[]=s.toCharArray();       System.out.println(result1);

byte result2[]=s.getBytes();       System.out.println(result2);

char result3[]=new char[5]; // extracting World
s.getChars(6,11,result3,0);       System.out.println(result3);
```

OUTPUT:

e
101
Hello World!
[B@7344699f
World

String Comparison

The String class includes several methods that compare strings or substrings within strings.

They are:

- equals() and equalsIgnoreCase()
- regionMatches()
- startWith() and endsWith()
- equals() Versus ==
- comapreTo()

equals()

To compare two Strings for equality, use equals()

General form:

boolean equals(Object str)

str is the String object being compared with the invoking String object.

It returns true if the string contain the same character in the same order, and false otherwise.

The comparison is case-sensitive.

equalsIgnoreCase()

To perform operations that ignores case differences.

When it compares two strings, it considers A-Z as the same as a-z.

General form:

boolean equalsIgnoreCase(Object str)

str is the String object being compared with the invoking String object.

It returns true if the string contain the same character in the same order, and false otherwise.

The comparison is case-sensitive.

Example:equals() and equalsIgnoreCase()

```
class equalsDemo {
  public static void main(String args[]) {
    String s1 = "Hello";
    String s2 = "Hello";
    String s3 = "Good-bye";
    String s4 = "HELLO";
    System.out.println(s1 + " equals " + s2 + " -> " +
                                         s1.equals(s2));
    System.out.println(s1 + " equals " + s3 + " -> " +
                                         s1.equals(s3));
    System.out.println(s1 + " equals " + s4 + " -> " +
                                      s1.equals(s4));
   System.out.println(s1 + " equalsIgnoreCase " + s4 +
                    " -> " + s1.equalsIgnoreCase(s4));
  }
}
```

regionMatches()

Compares a specific region inside a string with another specific region in another string.

There is an overloaded form that allows you to ignore case in such comparison.

General form:

```
boolean regionMatches(int startindex, String str2,
            int str2StartIndex, int numChars)


boolean regionMatches(boolean ignoreCase, int
      startindex, String str2, int str2StartIndex,
                                    int numChars)
```

Example

```
public class RegionMatchesExample{
   public static void main(String args[]){
      String str1 = new String("Hello, How are you");
      String str2 = new String("How");
      String str3 = new String("HOW");

      System.out.print("Result of Test1: " );
      System.out.println(str1.regionMatches(7, str2, 0, 3));

      System.out.print("Result of Test2: " );
      System.out.println(str1.regionMatches(7, str3, 0, 3));

      System.out.print("Result of Test3: " );
      System.out.println(str1.regionMatches(true, 7, str3, 0, 3));
   }
}
```

OUTPUT:

```
Result of Test1: true
Result of Test2: false
Result of Test3: true
```

startsWith() and endsWith()

String defines two routines that are more or less the specialised forms of *regionMatches().*

The *startsWith()* method determines whether a given string begins with a specified string.

Conversely, *endsWith()* method determines whether the string in question ends with a specified string.

General form:

- boolean startsWith(String str)
- boolean endsWith(String str)

str is the String being tested. If the string matches, true is returned otherwise false is returned.

startsWith() and endsWith()

Example:

“Foobar”.endsWith(“bar”);

and

“Foobar”.startsWith(“Foo”);

are both true.

startsWith() and endsWith()

A second form of *startsWith(),* let you specify a starting point.

General form:

boolean startsWith(String str, int startIndex)

Where *startIndex* specifies the index into the invoking string at which point the search will begin.

Example:

“Foobar”.startsWith(“bar”, 3);

returns true.

Equals() versus ==

It is important to understand that the two method performs different functions.

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

Example: Equals() versus ==

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

compareTo()

It is not enough to know that two Strings are identical. You need to know which is *less than, equal to, or greater than* the next.

A string is *less than* the another if it comes before the other in the dictionary order.

A string is *greater than* the another if it comes after the other in the dictionary order.

The String method *compareTo()* serves this purpose.

compareTo()

General form:

- int compareTo(String str)

str is the string that is being compared with the invoking String. The result of the comparison is returned and is interpreted as shown here:

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

```
public class CompareToExample{

public static void main(String args[]){

String s1="hello";

String s2="hello";

String s3="meklo";

String s4="hemlo";

String s5="flag";

System.out.println(s1.compareTo(s2));//0 because both are equal

System.out.println(s1.compareTo(s3));//negative value

System.out.println(s1.compareTo(s4));//negative value

System.out.println(s1.compareTo(s5));//positive value

}}
```

Searching String

String class provides two methods that allows you search a string for a specified character or substring:

- indexOf() – Searches for the first occurrence of a character or substring.
- lastIndexOf() – Searches for the last occurrence of a charater or substring.

These two methods are overloaded in several different ways. In all cases, the methods return the index at which the character or substring was found, or -1 on failure.

Searching String

To search for the first occurrence of a character, use

int indexOf(int ch)

To search for the last occurrence of a character, use

int lastIndexOf(int ch)

To search for the first and the last occurence of a substring, use

int indexOf(String str)

int lastIndexOf(String str)

Here str specifies the substring.

Searching String

You can specify a starting point for the serach using these forms:

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

startIndex – specifies the index at which point the search begins.

For indexOf(), the search runs from startIndex to the end of the string.

For lastIndexOf(), the search runs from startIndex to zero.

Example: Searching String

```
class indexOfDemo {
  public static void main(String args[]) {
    String s = "Now is the time for all good men " +
               "to come to the aid of their country.";
    System.out.println(s);
    System.out.println("indexOf(t) = " +
                       s.indexOf('t'));
    System.out.println("lastIndexOf(t) = " +
                       s.lastIndexOf('t'));
    System.out.println("indexOf(the) = " +
                       s.indexOf("the"));
    System.out.println("lastIndexOf(the) = " +
                       s.lastIndexOf("the"));
```

Example: Searching String

```
    System.out.println("indexOf(t, 10) = " +
                                        s.indexOf('t', 10));
    System.out.println("lastIndexOf(t, 60) = " +
                                      s.lastIndexOf('t', 60));
    System.out.println("indexOf(the, 10) = " +
                                     s.indexOf("the", 10));
    System.out.println("lastIndexOf(the, 60) = " +
                                     s.lastIndexOf("the", 60));
  }
}
```

Modifying a String

String object are immutable.

Whenever you want to modify a String, you must either copy it into a *StringBuffer* or use the following String methods,, which will construct a new copy of the string with your modification complete.

They are:

- subString()
- concat()
- replace()
- trim()

subString()

You can extract a substring using *subString().*

It has two forms:

- String substring(int startIndex)

startIndex specifies the index at which the substring will begin. This form returns a copy of the substring that begins at *startIndex* and runs to the end of the invoking string.

subString()

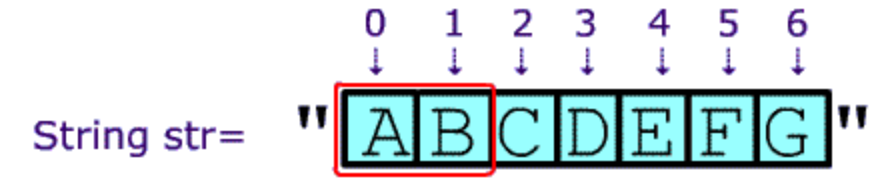
The second form allows you to specify both the beginning and ending index of the substring.

- String substring(int startIndex, int ensIndex)

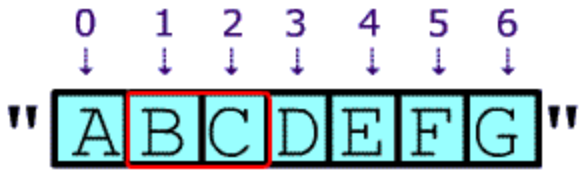
startIndex specifies the index beginning index, and

endIndex specifies the stopping point.

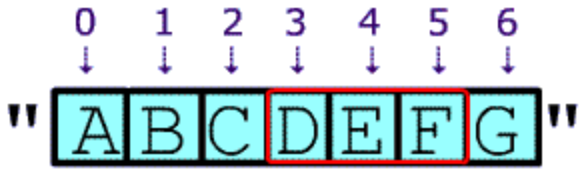
The string returned contains all the characters from the beginning index, up to, but not including, the ending index.



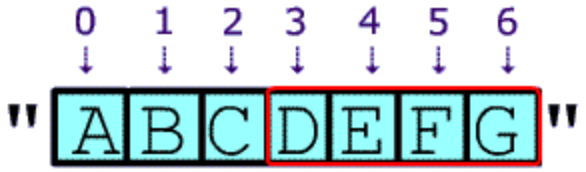

```
str.substring(0,2)  ⟶  "AB"
```




```
str.substring(1,3)  ⟶  "BC"
```




```
str.substring(3,6)  ⟶  "DEF"
```




```
str.substring(3,7)  ⟶  "DEFG"
```

Concat()

You can concatenate two string using concat()

General form:

- String concat(String str)

This method creates a new object that contains the invoking string with the contents of *str* appended to the end.

concat() performs the same function as +.

Example:

- String s1 =“one”;
- String s2 = s1.concat(“two”); Or
- String s2 = s1 + “two”;

Replace()

Replaces **all occurences** of one character in the invoking string with another character.

General form:

- String replace(char original, char replacement)

original – specifies the character to be replaced by the character specified by *replacemen*t. The resulting string is returned.

Example:

- String s = “Hello”.replace(‘l’,’w’);

Puts the string “Hewwo” into *s.*

Trim()

Returns a copy of the involving string from which any leading and trailing whitespace has been removed.

General form:

- String trim();

Example:

- String s = “ Hello world “.trim();

This puts the string “Hello world” into *s*.

It is quite useful when you process user commands.

Case of Characters

The method toLowerCase() converts all the characters in a string from uppercase to lowercase.

The toUpperCase() method converts all the characters in a string from lowercase to uppercase.

Non-alphabetical characters, such as digits are unaffected.

General form:

```
String toLowercase()

String toUppercase()
```

Example: Case of Characters

```
class ChangeCase {
  public static void main(String args[]) {
    String s = "This is a test.";
    System.out.println("Original: " + s);
    String upper = s.toUpperCase();
    String lower = s.toLowerCase();
    System.out.println("Uppercase: " + upper);
    System.out.println("Lowercase: " + lower);
  }
}
```

Java String join() method

The **java string join()** method returns a string joined with given delimiter. In string join method, delimiter is copied for each elements.

```
public class StringJoinExample{

public static void main(String args[])
{
String joinString1=String.join("-", "welcome","to","javaClass");
System.out.println(joinString1);
}
}
```

welcome-to-javaClass

Java String join() method

```
public class StringJoinExample2 {
    public static void main(String[] args) {
        String date = String.join("/","25","06","2018");
        System.out.print(date);

        String time = String.join(":", "12","10","10");
        System.out.println(" "+time);
    }
}
```

25/06/2018 12:10:10

Java String contains() method

The **java string contains()** method searches the sequence of characters in this string. It returns *true* if sequence of char values are found in this string otherwise returns *false*.

```
class ContainsExample{
public static void main(String args[]){
String name="what do you know about me";

System.out.println(name.contains("do you know"));
System.out.println(name.contains("about"));
System.out.println(name.contains("hello"));  }}
```

Java String contains() method

```
public class ContainsExample3 {
    public static void main(String[] args) {
        String str = "To learn Java visit Java.com";
        if(str.contains("Java.com")) {
            System.out.println("This string contains java.com");
        }else
            System.out.println("Result not found");
    }
}
```

Java String isEmpty() method

The **java string isEmpty()** method checks if this string is empty or not. It returns *true*, if length of string is 0 otherwise *false*. In other words, true is returned if string is empty otherwise it returns false.

```
public class IsEmptyExample{
public static void main(String args[]){
String s1="";
String s2="java";

System.out.println(s1.isEmpty());
System.out.println(s2.isEmpty());
}}
```

Java String “split” method

```
// Java program to demonstrate working of split()
public class GFG {
        public static void main(String args[])
        {
        String str = "GeeksforGeeks:A Computer Science Portal";
        String[] arrOfStr = str.split(":");

        for (String a : arrOfStr)
                System.out.println(a);
        }
}
```

OUTPUT:
GeeksforGeeks
A Computer Science Portal

Java String “split” method

```
// Java program to demonstrate working of split()
public class GFG {
        public static void main(String args[])
        {
                String str = "GeeksforforGeeksfor ";
                String[] arrOfStr = str.split("for");

                for (String a : arrOfStr)
                        System.out.println(a);
        }
}
```

OUTPUT:
Geeks

Geeks

Questions

How many objects will be created in the following code?

```
String s1="java";
String s2="java";
```

Questions

What is the difference between equals() method and == operator?

Questions?