String and StringBuffer

Agenda

1 String and StringBuffer

Objectives

At the end of this module, you will be able to:

- Implement String and StringBuffer class methods

String and StringBuffer

String

- String is a group 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 as final, so there cannot be subclasses of these classes.
- The default constructor creates an empty string.

```
String s = new String();
```

Creating Strings

- To Create a String in JAVA is

```
String str = "abc";
```

  is equivalent to:

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

- If data array in the above example is modified after the string object str is created, then str remains unchanged.

- Construct a string object by passing another string object.

```
String str2 = new String(str);
```

String class Methods

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

```
Eg: System.out.println("Varun".length()); // prints 5
```

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

```
Eg: String myName = "Varun";
String s = "My name is" + myName+ ".";
```

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

String class Methods (Contd.).

Characters in a string can be retrieved in a number of ways

- `public char charAt(int index)`

  Method returns the character at the specified index. An index ranges from 0 to length() – 1

- `char c;`

```
c = "abc".charAt(1); // c = “b”
```

String class Methods (Contd.).

- **equals() Method-** This method is used to compare the invoking String to the object specified. It will return true, if the argument is not null and it is String object which contains the same sequence of characters as the invoking String.

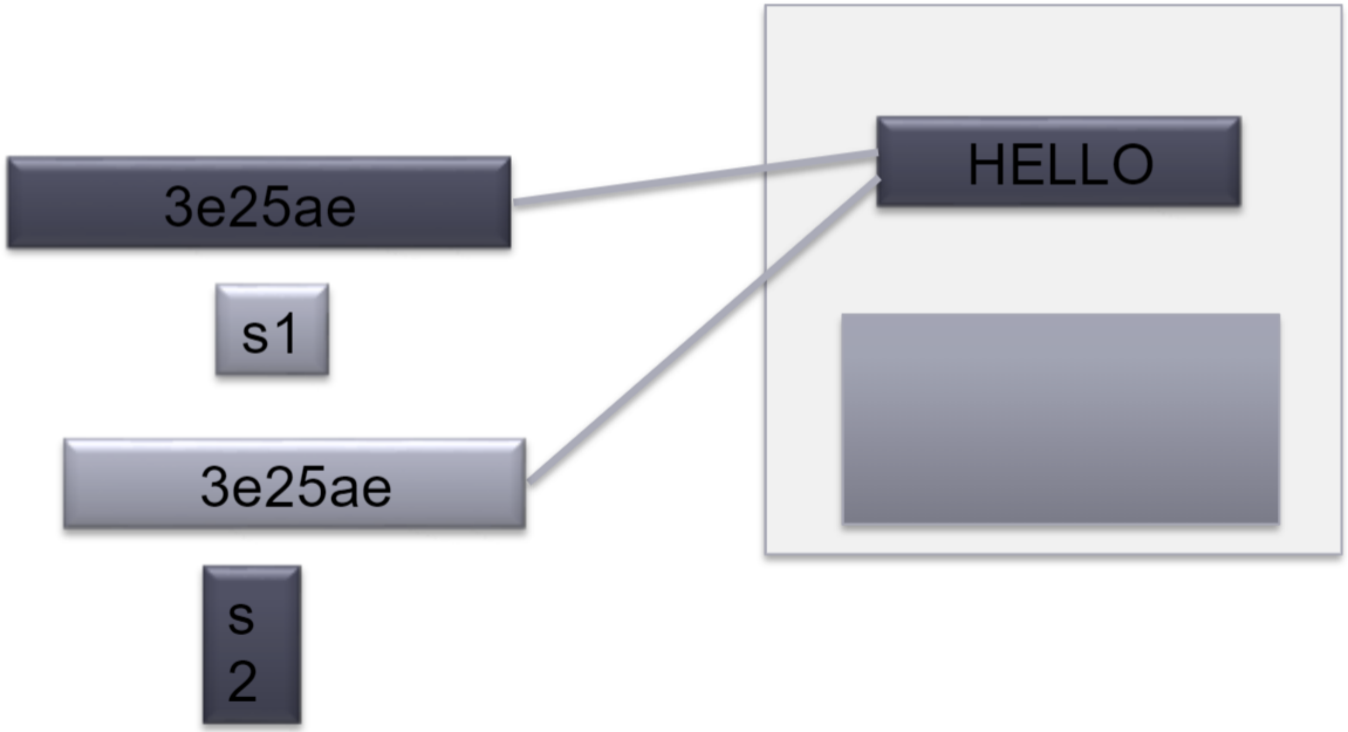
```
public boolean equals(Object anObject)
```

- **equalsIgnoreCase() Method-** Compares this String to another String, ignoring case considerations. Two strings are considered equal ignoring case if they are of the same length, and corresponding characters in the two strings are equal ignoring case.

```
public boolean equalsIgnoreCase(String anotherString)
```

Strings created using assignment operator

- String s1 = “HELLO”;
- String s2 = “HELLO”;



Comparing Strings using == operator

What is the output ?

```
public class StringTest{
  public static void main(String[] args){
    String s1="Hello";
    String s2="Hello";
    if(s1==s2)
        System.out.println("String objects referenced are same");
    else
        System.out.println("String objects referenced are not same");
  }
}
```

Output: String objects referenced are same

Comparing Strings using equals method

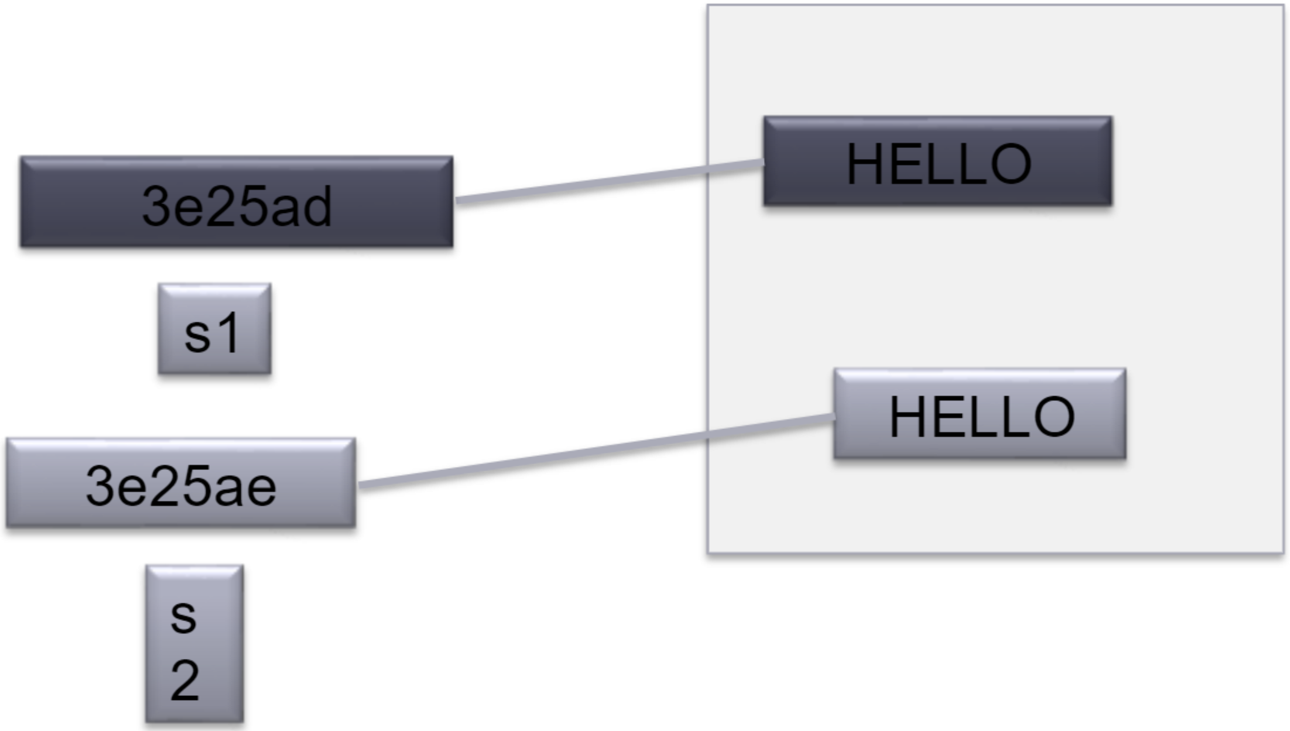
What is the output ?

```
public class StringTest{
  public static void main(String[] args){
    String s1="Hello";
    String s2="Hello";
    if(s1.equals(s2))
        System.out.println("Strings are equal");
    else
        System.out.println("Strings are not equal");
  }
}
```

Output: Strings are equal

Strings created using new keyword

- `String s1 = new String(“HELLO”);`
- `String s2 = new String(“HELLO”);`



Comparing Strings using == operator

What is the output ?

```
public class StringTest{
   public static void main(String[] args){
      String s1= new String("Hello");
      String s2= new String("Hello");
      if(s1==s2)
            System.out.println("String objects referenced are same ");
      else
            System.out.println("String objects referenced are not same");
   }
}
```

Output: String objects referenced are not same

Comparing Strings using equals method

What is the output ?

```
public class StringTest{
  public static void main(String[] args){
    String s1= new  String("Hello");
    String s2= new String("Hello");
    if(s1.equals(s2))
        System.out.println("Strings are equal");
    else
        System.out.println("Strings are not equal");
  }
}
```

Output: Strings are equal

String class Methods

startsWith() – Tests if this string starts with the specified prefix.

```
public boolean startsWith(String prefix)
"January".startsWith("Jan"); // true
```

endsWith() - Tests if this string ends with the specified suffix.

```
public boolean endsWith(String suffix)
"January".endsWith("ry"); // true
```

String class Methods (Contd.).

compareTo() - Compares two strings and to know which string is bigger or smaller

- We will get a negative integer, if this String object is less than the argument string
- We will get a positive integer if this String object is greater than the argument string.
- We will get a return value 0(zero), if these strings are equal.

```
public int compareTo(String anotherString)
public int compareToIgnoreCase(String str)
```

This method is similar to compareTo() method but this does not take the case of strings into consideration.

String class Methods (Contd.).

- **indexOf** – Searches for the first occurrence of a character or substring. Returns -1 if the character does not occur

- **public int indexOf(int ch)-** It searches for the character represented by ch within this string and returns the index of first occurrence of this character

- **public int indexOf(String str) -** It searches for the substring specified by str within this string and returns the index of first occurrence of this substring

```
String str = “How was your day today?”;
str.indexOf(‘t’);
str.indexOf(“was”);
```

String class Methods (Contd.).

- **public int indexOf(int ch, int fromIndex)-** It searches for the character represented by ch within this string and returns the index of first occurrence of this character starting from the position specified by fromIndex

- **public int indexOf(String str, int fromIndex)** - Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.

```
String str = “How was your day today?”;
str.indexOf(‘a’, 6);
 str.indexOf(“was”, 2);
```

String class Methods (Contd.).

- **lastIndexOf()** –It searches for the last occurrence of a particular character or substring

- **substring() -** This method returns a new string which is actually a substring of this string. It extracts characters starting from the specified index all the way till the end of the string

```
public String substring(int beginIndex)
    Eg: "unhappy".substring(2) returns "happy"

public String substring(int beginIndex, int endIndex)
    Eg: "smiles".substring(1, 5) returns "mile“
```