**java.lang package – Part-06**

* **Constructors of String class:**

String s = new String();

Creates an empty string object.

String s = new String(String literal);

Creates a String object on the heap for the given String literal.

String s = new String(StringBuffer sb);

Creates an equivalent String object for the given StringBuffer.

String s = new String(char[] ch);

Creates an equivalent String object for the given char[]

Example:

char[] ch = {‘a’, ‘b’, ‘c’, ‘d’};

String s = new String(ch);

System.out.println(s);

Output: abcd

String s= new String(byte[] b);

Creates an equivalent String object for the given byte[]

Example:

byte[] b = {100, 101, 102, 103};

String s = new String(b);

System.out.println(s);

Output: defg

* **Important methods of String class:**

public char charAt(int index);

Returns the character located at specified index.

Example:

String s = “durga”;

System.out.println(s.charAt(3)); //g

System.out.println(s.chartAt(30));

RE: StringIndexOutOfBoundsException

public String concat(String s);

The overloaded + and += operators also meant for concatenation purpose only.

Example:

String s = “durga”;

s = s.concat(“software”);

s = s+”software”;

s += “software”;

System.out.println(s); // durgasofware

public boolean equals(Object o);

To perform content comparison where case is important. This overriding version of Object class equals method.

public boolean equalsIgnoreCase(Object o);

To perform where content comparison for case is not important. This method is available only on String class.

Example:

String s = “java”;

System.out.println(s.equals(“JAVA”)); // false

System.out.println(s.equalsIgnoreCase(“java”));

Note:

In general we can use equalsIgnoreCase() method to validate usernames. Where case is not important, whereas we can use equals method to validate password where is important.

public String substring(int begin);

Returns substring from begin index to end of the String.

public String substring(int begin, int end);

Returns substring from begin index to end-1 index.

Example:

String s = “abcdefg”;

System.out.println(s.substring(3)); // defg

System.out.println(s.substring(3, 6)); // cdef

public int length();

Returns number of characters present in the String.

Example:

String s = “durga”;

System.out.println(s.length);

CE: cannot find symbol

symbol:variable length

location: java.lang.String

System.out.println(s.length()); //5

Note:

length variable applicable for Arrays but not for String class. Whereas length() method applicable for String objects but not for Arrays.

public String replace(char oldChar, char newCh);

Example:

String s = “ababa”;

System.out.println(s.replace(‘a’, ‘b’)); // bbbbb

public String toLowerCase();

public String toUpperCase();

public String trim();

To remove blank spaces present at beginning and end of the String but not middle blank spaces.

public int indexOf(char ch);

Returns index of first occurrence of specified character.

public int lastIndexOf(char ch);

Example:

String s = “ababa”;

System.out.println(s.indexOf(‘b’)); //1

System.out.println(s.lastIndexOf(‘b’)); // 4

* **Note \*\*\*\***

Because of runtime operation if there is a change in the content, then with those changes a new object will be created on the heap.

If there is no changes in the content, then existing object will be reused and new object won’t be created.

Whether the object presents in heap or SCP, the rule is same.

Example:

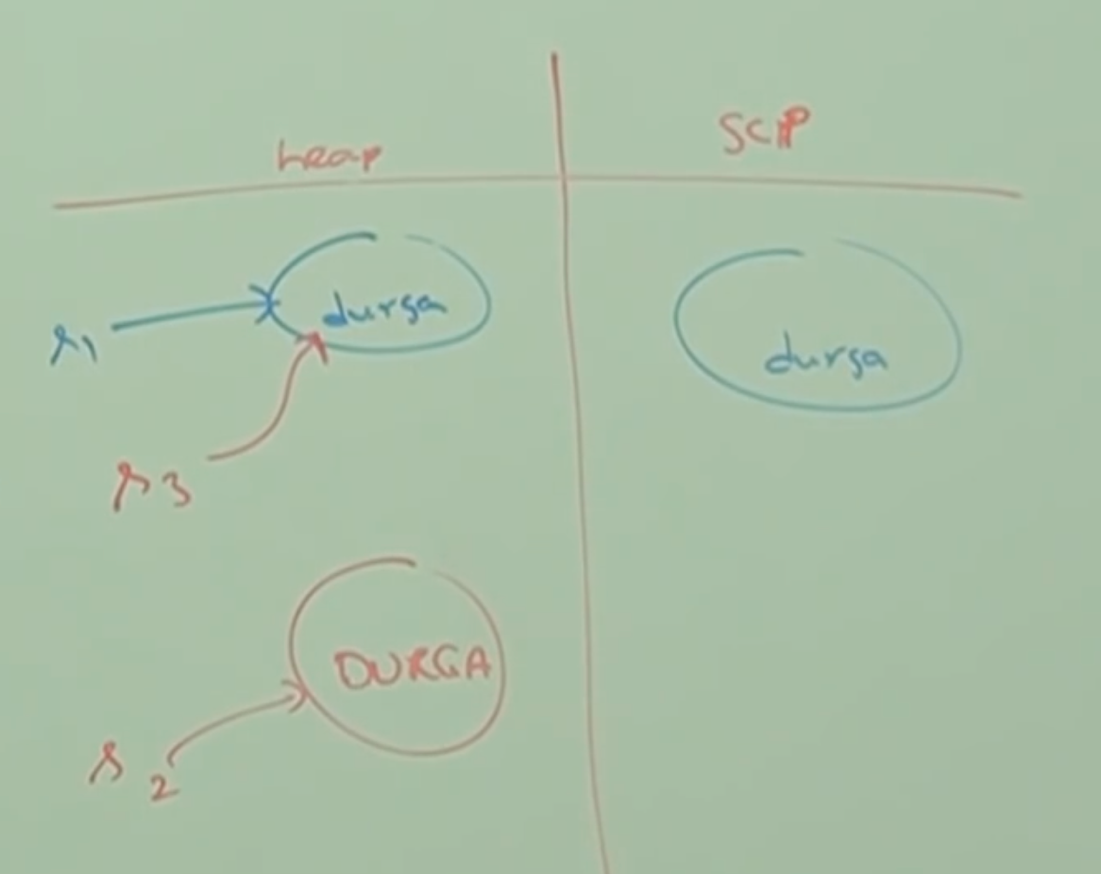
String s1 = new String(“durga”);

String s2 = s1.toUpperCase();

String s3 = s1.toLowerCase();

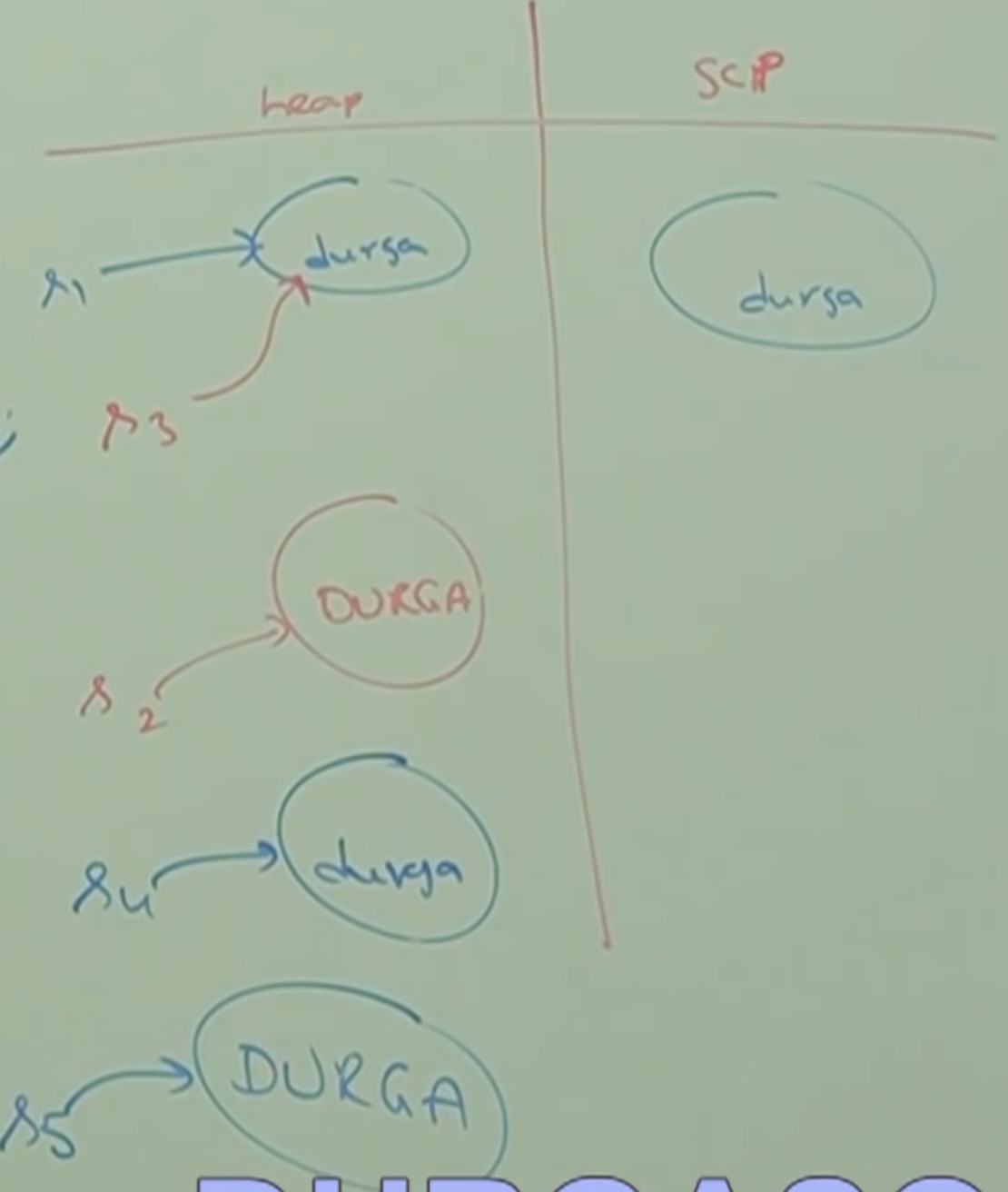
System.out.println(s1 == s2); // false

System.out.println(s1 == s3); // true



String s4 = s2.toLowerCase();

String s5 = s4.toUpperCase();



Note:

If there is a change on content because of the method, then new object will be created. It won’t check whether the object is already there or not.

String s1 = “durga”;

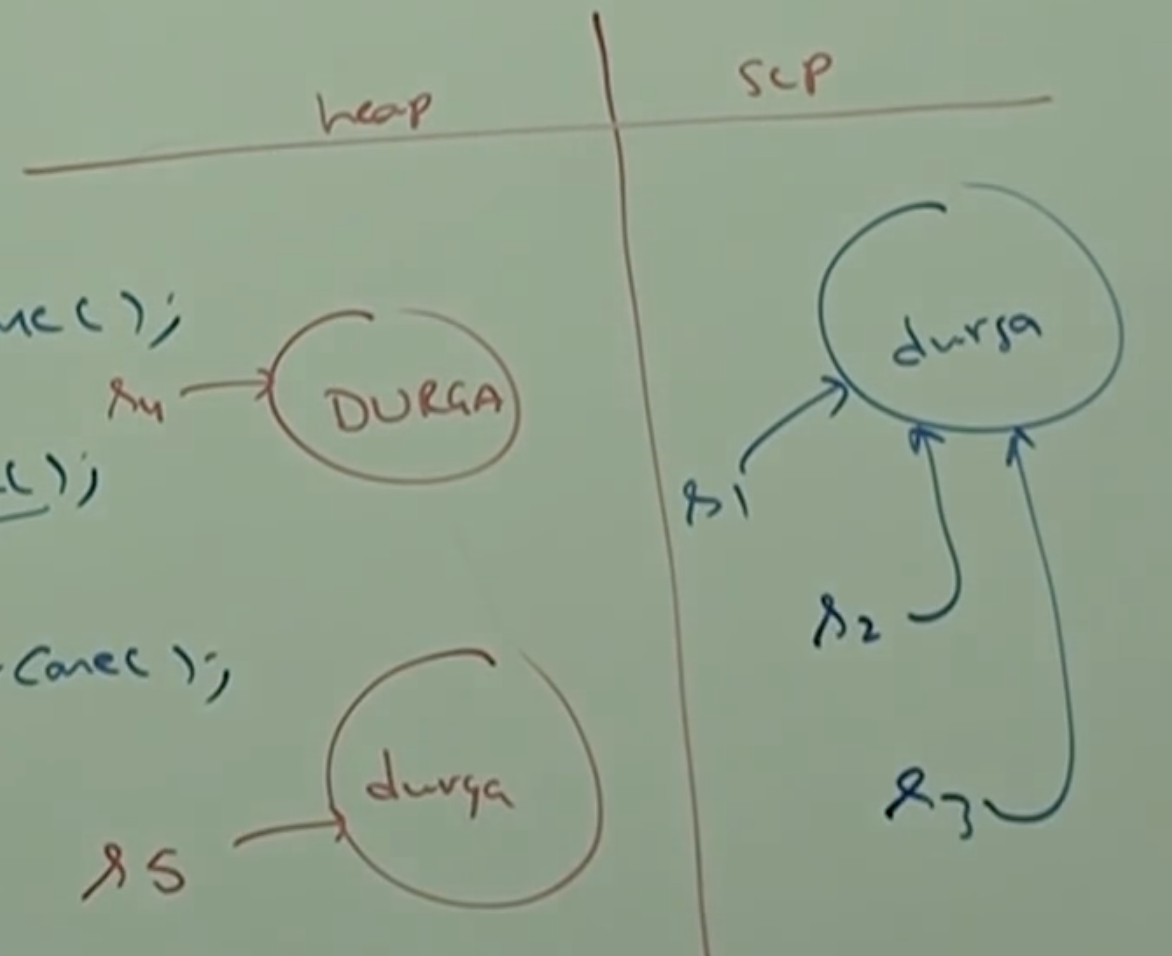
String s2 = s1.toString();

System.out.println(s1 == s2); // true

String s3 = s1.toLowerCase();

String s4 = s1.toUpperCase();

String s5 = s4.toLowerCase();



* **How to create our own immutable class:**

Once we create an object, we can’t perform any changes in that object. If we are trying to perform any change and if there is a change in the content, then with those changes a new object will be created.

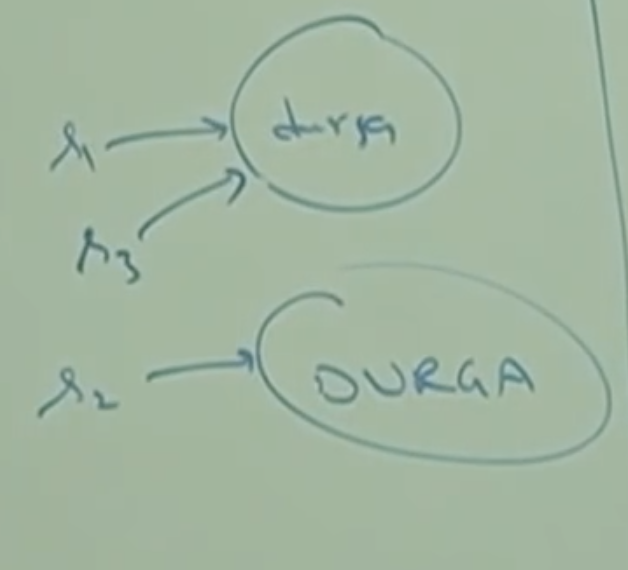
If there is no change in the content, then existing object will be reused.

This behavior is nothing but immutability.

String s1 = new String(“durga”);

String s2 = s1.toUpperCase();

String s3 = s1.toLowerCase();



We can create our own immutable class.

final class Test{

private int i;

Test(int i){

this.i = i;

}

public Test modify(int i){

if(this.i == i){

return this;

} else{

return (new Test(i));

}

}

}

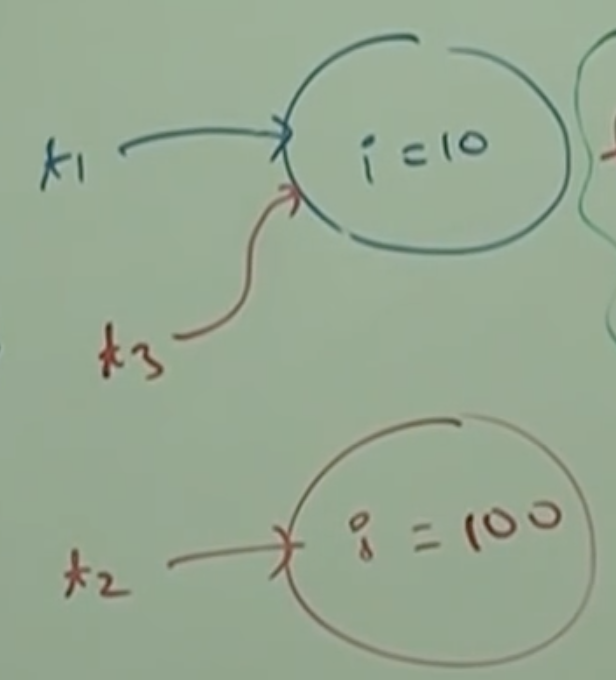
Test t1 = new Test(10);

Test t2 = t1.modify(100);

Test t3 = t1.modify(10);

System.out.println(t1 == t2); // false

System.out.println(t1 == t3); // true



Note:

Once we create a test object, we can’t perform any change in the existing object. If we are trying to perform any change and if there is a change in the content then with those changes a new object will be created. And, if there is no change in the content, then existing object will be reused.

* **final VS immutability**

final applicable for variables but not for objects. Whereas immutability applicable for objects but not for variables.

By declaring a reference variable as final we won’t get any immutability nature. Even though reference variable is the final, we can perform any type of change in the corresponding object. But, we can’t perform re-assignment for that variable.

Hence final and immutable both are different concepts.

Example:

final StrinBuffer sb = new StringBuffer(“durga”);

sb.append(“software”);

System.out.println(sb); // durgasoftware

sb = new StringBuffer(“solutions”);

Last line we will get compile time error:

Cannot assign a value to final variable sb.

Which of the following are meaningful?

final variable // Valid

immutable variable // Invalid

final object // Invalid

immutable object // valid