

Superclass Object

- Superclass of all Java™ classes.
- Common methods to be redefined by derived classes.

String literals

Stack

Heap

```
String s1 = "Kate";
```

```
String s2 = "Kate";
```

String literals

Stack

Heap

```
String s1 = "Kate";
```

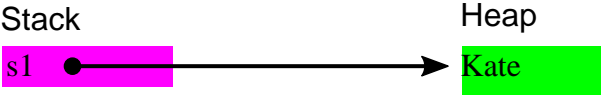
Kate

```
String s2 = "Kate";
```

String literals

```
String s1 = "Kate";
```

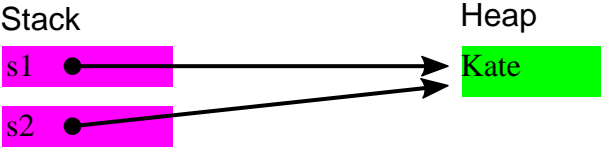
```
String s2 = "Kate";
```



String literals

```
String s1 = "Kate";
```

```
String s2 = "Kate";
```



OpenJDK String implementation

Implementation of java.lang.String:

```
public final class String ... {  
    private final char value[];  
    private int hash;  
    private static final long serialVersionUID = -6849794470754667710L;  
    ...  
}
```

String copy constructor

Code	Output
<pre>final String s = "Eve"; ❶ final String sCopy = new String(s); ❷ System.out.println("sCopy == s: " + (sCopy == s)); ❸ System.out.println("sCopy.equals(s): " + sCopy.equals(s)); ❹</pre>	<pre>sCopy == s: false ❸ sCopy.equals(s): true ❹</pre>

Copy constructor and heap

Stack

Heap

```
String s1 =  
new String("Kate");
```

```
String s2 =  
new String("Kate");
```


Copy constructor and heap

Stack

```
String s1 =  
new String("Kate");
```

```
String s2 =  
new String("Kate");
```

Heap

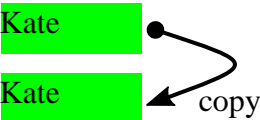
```
Kate
```

Copy constructor and heap

Stack

```
String s1 =  
new String("Kate");  
  
String s2 =  
new String("Kate");
```

Heap



Copy constructor and heap

```
String s1 =  
new String("Kate");
```

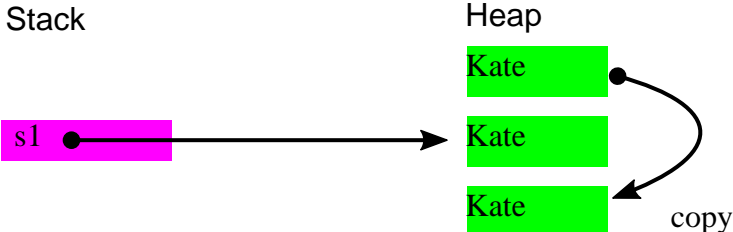
```
String s2 =  
new String("Kate");
```



Copy constructor and heap

```
String s1 =  
new String("Kate");
```

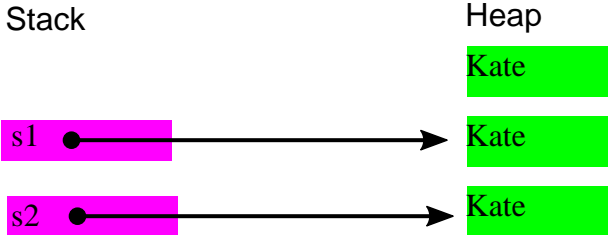
```
String s2 =  
new String("Kate");
```



Copy constructor and heap

```
String s1 =  
new String("Kate");
```

```
String s2 =  
new String("Kate");
```



Operator == and equals ()

Primitive type

```
// equal values
```

```
int a = 12, b = 12;
```

```
System.out.println(  
    "==: " + (a == b));
```

```
// No equals(...) method  
// for primitive types
```

```
==: true
```

Object

String

```
s1 = new String("Kate"),  
s2 = new String("Kate");
```

```
System.out.println(  
    "==: " + (s1 == s2));
```

```
System.out.println(  
    "equals: " + s1.equals(s2));
```

```
==: false
```

```
equals: true
```

Remarks == vs. equals ()

- The == operator acting on primitive types compares expression values.
- The == operator acting on objects compares for equality of reference values and thus for object identity.
- The == operator acting on objects does **not** check whether two objects carry semantically equal values.
- The equals () method defines the equality two objects.

Operator == and equals () implications

- Each object is equal by value to itself:

```
object1 == object2 ⇒ object1.equals(object2)
```

- The converse is not true. Two different objects may be of common value:

Code

```
String s = "Hello", copy = new String(s);
```

```
System.out.println("equals: " + s.equals(copy));
```

```
System.out.println("    ==: " + (s == copy));
```

Result

```
equals: true
```

```
    ==: false
```


equals () is being defined within respective class!

Implementation at <https://github.com/openjdk/.../String.java> :

```
public final class String ... {
public boolean equals(Object anObject) {
    if (this == anObject) {
        return true;
    }
    return (anObject instanceof String aString)
        && (!COMPACT_STRINGS || this.coder == aString.coder)
        && StringLatin1.equals(value, aString.value);
}
```

Hashing principle



Price List

	ice cream cone	3p
	ice cream with flake	4p
	banana split	18p
	knickerbocker glory	15p
	tub	12p
	lollipop	2p

“I want the 12p one”

Quickly identify by “simple” value

- Where is the blond haired guy?
- I take the pink flower.
- The 334.50\$ cellular phone.

Hashing in Java and equals ()

Method hashCode(): Instance 0 # o.hashCode(), of type int.

- Same value on repeated invocation
- Objects with identical value with respect to equals () must have identical hash values:
`true == a.equals(b) # a.hashCode() == b.hashCode()`.
- Conversely: Two instances differing with respect to equals () may have identical hash values.

Consequence: equals () and hashCode () must be **redefined simultaneously!**

Rectangle equals(...) and hashCode()

```
public class Rectangle {
    int width, height;
    @Override public boolean equals(Object o) {
        if (o instanceof Rectangle r) {
            return width == r.width
                && height == r.height;
        } else {
            return false;
        }
    }
    @Override public int hashCode() {
        return width + height;
    }
}
```

Rectangle hash values

<pre>public class Rectangle { int width, height; ... @Override public int hashCode() { return width + height; } }</pre>	width	height	hash value
	1	3	4
	2	2	4
	5	5	10
	2	7	9
	4	9	13

Better hashCode() method

<pre>public class Rectangle { int width, height; ... @Override public int hashCode() { return width + 13 * height; } }</pre>	width	height	hash value
	1	3	40
	2	2	28
	5	5	70
	2	7	93
	4	9	121

Related exercises

Exercise 133: Choosing a “good” hashCode () method

Exercise 134: String and good hashCode() implementations.

Math.sin(double x)

Code	Result	Math notation
<pre>final double x = 90; final double y = Math.sin(x); System.out.println(y + " == <i>sin</i>(" + x + ")");</pre>	0.8939966636005579 == sin(90.0)	

Related exercises

Exercise 135: Common pitfall using trigonometric functions

Exercise 136: Using constants from `java.lang.Math`.

Exercise 137: Strings on CodingBat

Exercise 138: Masking strings

Exercise 139: Analyzing strings

Exercise 140: Pitfalls using `==`: Equality of `String` instances

Exercise 141: Weird, weirder, weirdest!

Exercise 142: Analyzing file pathnames