Java – Strings

Presented By Sagar Java Consultant

String class

- An object of the String class represents a string of characters.
- The String class belongs to the java.lang package, which does not require an import statement.
- Like other classes, String has constructors and methods.
- Unlike other classes, String has two operators, + and += (used for concatenation).

String Literals

- are anonymous objects of the String class
- are defined by enclosing text in double quotes. “Welcome to Java World!”
- don’t have to be constructed.
- can be assigned to String variables.
- can be passed to methods and constructors as parameters.

String Literals - Examples

```
//assign a literal to a String variable
String name = “Robert”;

//calling a method on a literal String
char firstInitial = “Robert”.charAt(0);

//calling a method on a String variable
char firstInitial = name.charAt(0);
```

Immutability

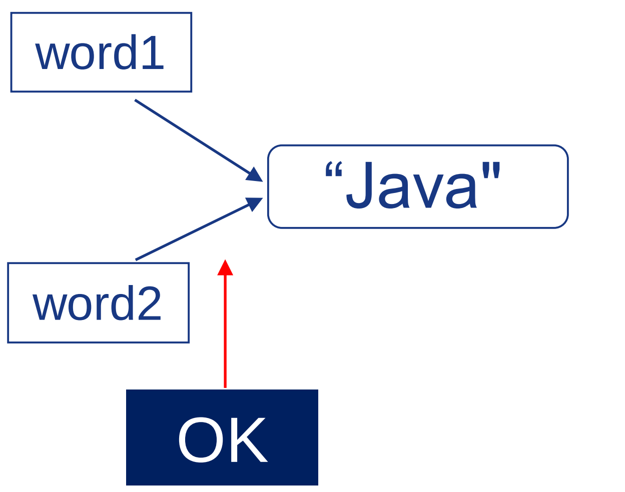
- Once created, a string cannot be changed: none of its methods changes the string.
- Such objects are called *immutable*.
- Immutable objects are convenient because several references can point to the same object safely: there is no danger of changing an object through one reference without the others being aware of the change.

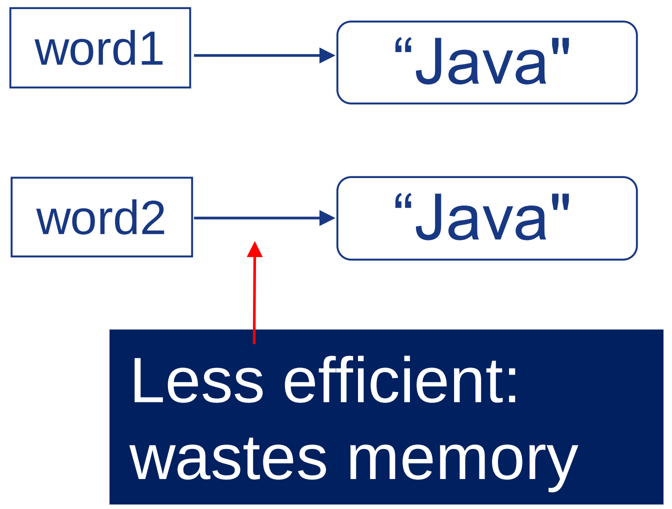
Advantages Of Immutability

Uses less memory.

```
String word1 = "Java";
String word2 = word1;
```

```
String word1 = “Java";
String word2 = new String(word1);
```

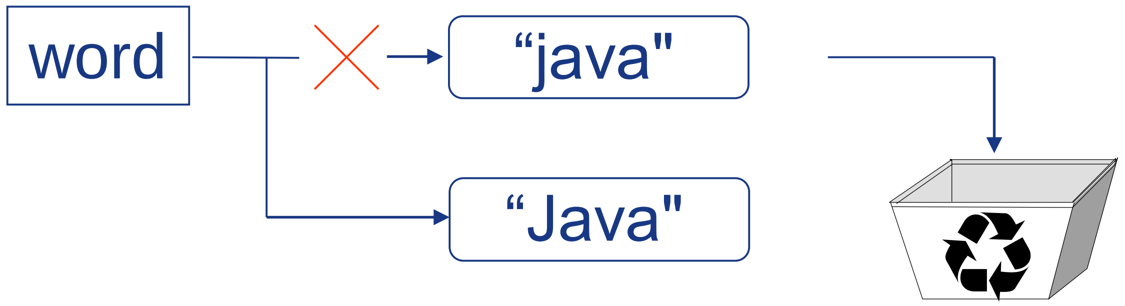




Disadvantages of Immutability

Less efficient — you need to create a new string and throw away the old one even for small changes.

```
String word = “Java";
char ch = Character.toUpperCase(word.charAt (0));
word =  ch + word.substring (1);
```



Empty Strings

- An empty String has no characters. It’s length is 0.

```
String word1 = "";
String word2 = new String();
```

Empty strings

- Not the same as an uninitialized String.

```
private String errorMsg;
```

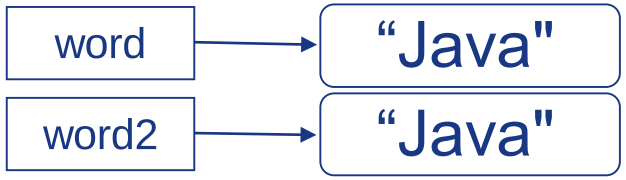
errorMsg is **null**

Copy Constructors

- Copy constructor creates a copy of an existing String. Also rarely used.
- Not the same as an assignment.

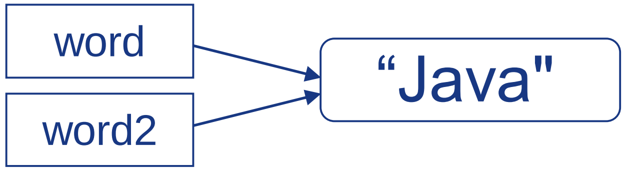
Copy Constructor: Each variable points to a different copy of the String.

```
String word = new String(“Java”);
String word2 = new String(word);
```



Assignment: Both variables point to the same String.

```
String word = “Java”;
String word2 = word;
```



Other Constructors

Most other constructors take an array as a parameter to create a String.

```
char[] letters = {‘J’, ‘a’, ‘v’, ‘a’};
String word = new String(letters);//”Java”
```

Methods — length(), charAt()

int length();

- Returns the number of characters in the string

char charAt(i);

- Returns the char at position i.

Character positions in strings are numbered starting from 0 – just like arrays.

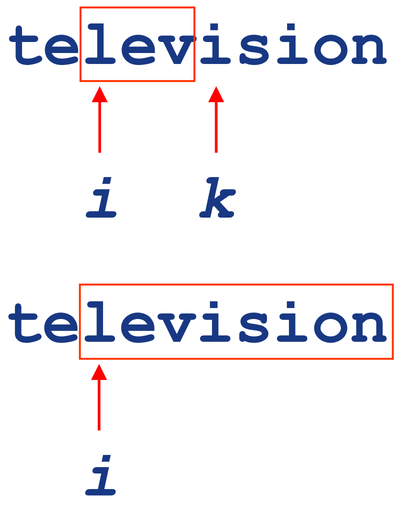
Returns:

”welcomem".length();	7
”Window".charAt (2);	’n'

Methods — substring()

Returns a new String by copying characters from an existing String.

- String subs = word.**substring** (i, k);
  - returns the substring of chars in positions from **i** to **k-1**
- String subs = word.**substring** (i);
  - returns the substring from the **i**-th char to the end



	Returns:
”television".substring (2,5);	“lev"
“immutable".substring (2);	“mutable"
“bob".substring (9);	"" (empty string)

Methods — concat()

String word1 = “re”, word2 = “think”; word3 = “ing”;

int num = 2;

- String result = word1 **+** word2;

  //concatenates word1 and word2   “rethink“
- String result = word1.**concat** (word2);

  //the same as word1 + word2 “rethink“
- **result +=** word3;

  //concatenates word3 to result “rethinking”
- **result** += num; //converts num to String

  //and concatenates it to result “rethinking2”

Methods — indexOf()

```
                  0  2     6    10       15
String name =“President George Washington";
```

```
                                      Returns:
date.indexOf (‘P');                      0
date.indexOf (‘e');                      2
date.indexOf (“George");                10
date.indexOf (‘e', 3);                   6      (starts searching
                                                 at position 3)

date.indexOf (“Bob");                   –1      (not found)
date.lastIndexOf (‘e');                 15
```

Methods — equals(), equalsIgnoreCase()

boolean b = word1.**equals**(word2);

returns **true** if the string **word1** is equal to **word2**

boolean b = word1.**equalsIgnoreCase**(word2);

returns **true** if the string **word1** matches **word2**, case-blind

```
b = “Raiders”.equals(“Raiders”);//true
b = “Raiders”.equals(“raiders”);//false
b = “Raiders”.equalsIgnoreCase(“raiders”);//true
```

```
if(team.equalsIgnoreCase(“raiders”))
        System.out.println(“Go You “ + team);
```

Methods — Comparisons

String word1 = “Welcome”;

String word2 = “welcome”

int diff = word1.**compareTo**(word2);

returns the “difference” **word1 – word2**

int diff = word1.**compareToIgnoreCase**(word2);

returns the “difference” **word1 – word2**, case-blind

```
if(word1.compareTo(word2) > 0){
        //word1 comes after word2…
}
```

Comparison Examples

```
//negative differences
diff = “apple”.compareTo(“berry”);//a before b
diff = “Zebra”.compareTo(“apple”);//Z before a
diff = “dig”.compareTo(“dug”);//i before u
diff = “dig”.compareTo(“digs”);//dig is shorter
```

```
//zero differences
diff = “apple”.compareTo(“apple”);//equal
diff = “dig”.compareToIgnoreCase(“DIG”);//equal
```

```
//positive differences
diff = “berry”.compareTo(“apple”);//b after a
diff = “apple”.compareTo(“Apple”);//a after A
diff = “BIT”.compareTo(“BIG”);//T after G
diff = “huge”.compareTo(“hug”);//huge is longer
```

Methods — trim

String word2 = **word1.trim** ();

returns a new string formed from **word1** by removing white space at both ends **does not** affect whites space in the middle

```
String word1 = “ Hi Bob “;
String word2 = word1.trim();
//word2 is “Hi Bob” – no spaces on either end
//word1 is still “ Hi Bob “ – with spaces
```

Methods — replace

String word2 = word1.**replace**(oldCh, newCh);

returns a new string formed from **word1** by replacing all occurrences of **oldCh** with **newCh**

```
String word1 = “rare“;
String word2 = “rare“.replace(‘r’, ‘d’);
//word2 is “dade”, but word1 is still “rare“
```

Methods — Changing Case

String word2 = word1.**toUpperCase**();
String word3 = word1.**toLowerCase**();

returns a new string formed from **word1** by converting its characters to upper (lower) case

```
String word1 = “HeLLo“;
String word2 = word1.toUpperCase();//”HELLO”
String word3 = word1.toLowerCase();//”hello”

//word1 is still “HeLLo“
```

Replacements

- Example: to “convert” word1 to upper case, replace the reference with a new reference.

```
word1 = word1.toUpperCase();
```

- A common bug:

```
word1.toUpperCase();
```

word1 remains unchanged

StringBuffer object

StringBuffer object

StringBuffer object

StringBuffer object

enter
return
THANK YOU
shutterstock
alt
control