innovate achieve lead

BITS Pilani
Pilani Campus

Object Oriented Programming

The String Class

Objectives

- Learn about literal strings
- Learn about String constructors
- Learn about commonly used methods
- Understand immutability of strings
- Learn to format numbers into strings

String class facts

- 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).

Literal Strings

- are anonymous objects of the String class
- are defined by enclosing text in double quotes. “This is a literal String”
- don’t have to be constructed.
- can be assigned to String variables.
- can be passed to methods and constructors as parameters.
- have methods you can call.

Literal String 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);
```

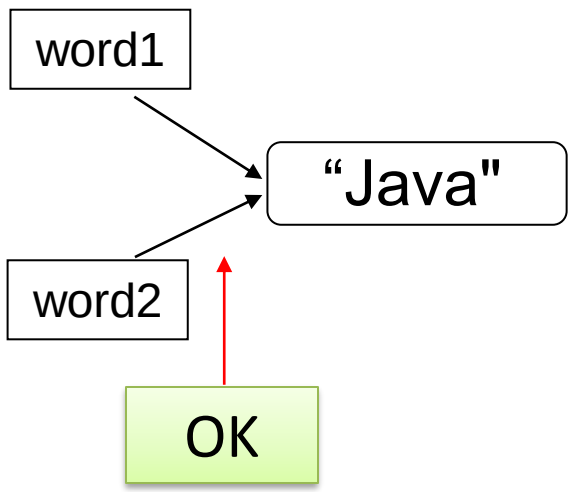
Immutability

- 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.

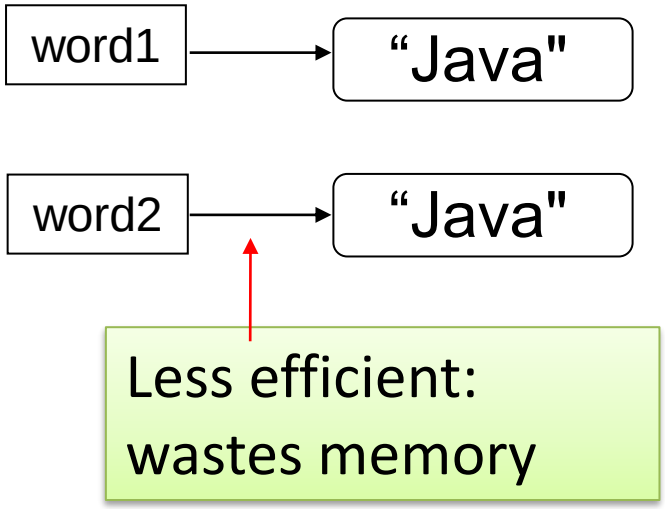
Advantages Of Immutability

Uses less memory.

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



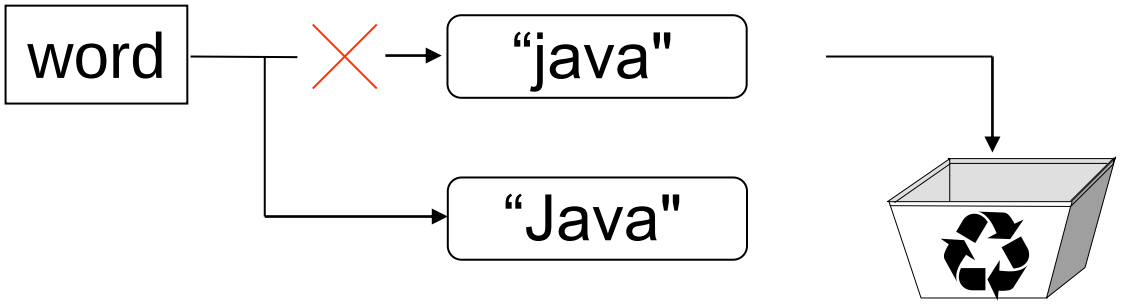

```
String word1 = "Java";
String word2 = new String(word1);
```



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**

No Argument Constructors

- No-argument constructor creates an empty String. Rarely used.

```
String empty = new String();
```

- A more common approach is to reassign the variable to an empty literal String. (Often done to reinitialize a variable used to store input.)

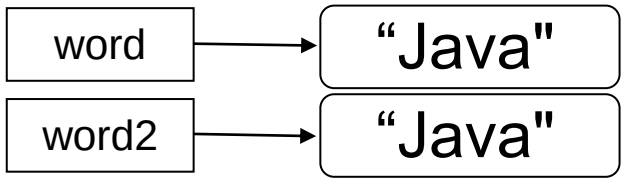
```
String empty = “”;//nothing between quotes
```

Copy Constructors

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

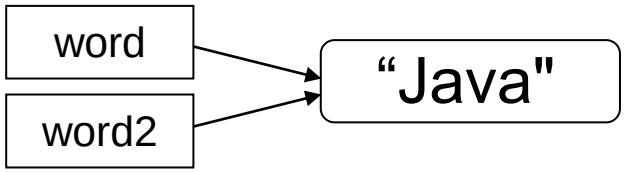
Copy Constructor: Each variable points to a different copy of the String.

```
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.

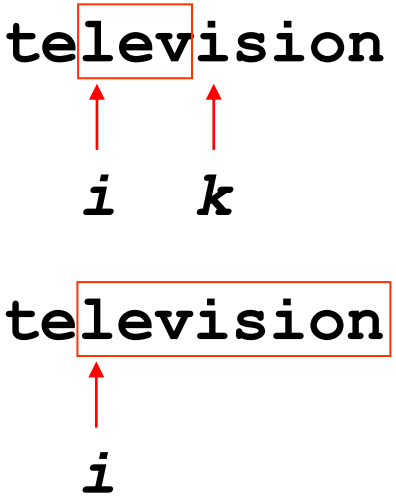
Character positions in strings are numbered starting from 0 – just like arrays.

	Returns:
”Problem".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);	Error

Methods — Concatenation

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 — Find (indexOf)

0 2 6 10 15

String name =“President George Washington";

	Returns:
name.indexOf ('P');	0
name.indexOf ('e');	2
name.indexOf ("George");	10
name.indexOf ('e', 3);	6
name.indexOf ("Bob");	−1
name.lastIndexOf ('e');	15

(starts searching at position 3)

(not found)

Methods — Equality

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

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

returns the "difference" **word1 – word2**

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

returns the "difference" **word1 – word2**, case-blind

Usually programmers don't care what the numerical "difference" of **word1 - word2** is, just whether the difference is negative (word1 comes before word2), zero (word1 and word2 are equal) or positive (word1 comes after word2). Often used in conditional statements.

```
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

Numbers to Strings

Three ways to convert a number into a string:

1. String s = "" + num;

```
s = “” + 123;//”123”
```

2. String s = Integer.toString (i);

   String s = Double.toString (d);

```
s = Integer.toString(123);//”123”
s = Double.toString(3.14); //”3.14”
```

3. String s = String.valueOf (num);

```
s = String.valueOf(123);//”123”
```

Integer and **Double** are “wrapper” classes from **java.lang** that represent numbers as objects. They also provide useful static methods.

Review Questions:

1. The String class is part of what package?
2. What does the String class have that other classes do not have?
3. “Text enclosed in quotes is called ?”
4. What is the returned value for “Rumplestiltskin”.length()?
5. Define immutable objects.

Review (cont'd):

6. How does immutability of Strings make Java more efficient?
7. How does immutability of Strings make Java less efficient?
8. How do you declare an empty string?
9. Why are String constructors not used very often?
10. "Bob" + " " + "Smith" is called ____ ?

Review (cont'd):

11. String city = "Bloomington";
    What is returned by city.charAt (2)?
12. By city.substring(2, 4)?
13. By city.lastIndexOf('o')?
14. By city.indexOf(3)?
15. What does the trim method do?

Review (cont'd):

16. "sam".equals("Sam") returns ?
17. What kind of value does "sam".compareTo("Sam") return?
18. What will be stored in s?
    s = "mint".replace('t', 'e');
19. What does s.toUpperCase() do to s?
20. Name a simple way to convert a number into a string.

THANK YOU