



The Class String



Outline

- The String Class
- Explicit String Objects
- String variables are References
- String Methods
- Escape Sequences



- The String Class

- There is no primitive type for strings in Java
- The class (type) **String** is a predefined class in Java that is used to store and process strings
- A **String** object is a *sequence of characters* that is treated as a single value.
 - When any " " (quote) appears, a **String** object is created automatically.

```
System.out.println("Hello, how are you?");
```

- A variable of type **String** can be given the value of a **String** object

```
String sentence = "Java is Fun!";
```



- Explicit String Objects

- A declaration and object creation are needed for instances of the String class. For example,

```
String name1;  
name1 = new String("Salam");
```

- We normally use shorthand notation (only for Strings):

```
String name1;  
name1 = "Salam";
```

These two statements are equivalent.

- String variables are References...

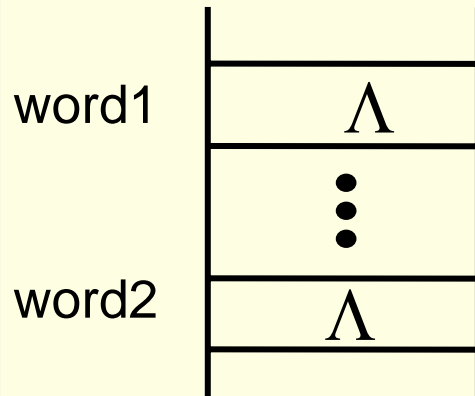
Code




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

Both word1 and word2 are allocated memory (to store references), but the objects themselves are not yet created, so they both contain null.

State of Memory



After  is executed

-...String variables are References...

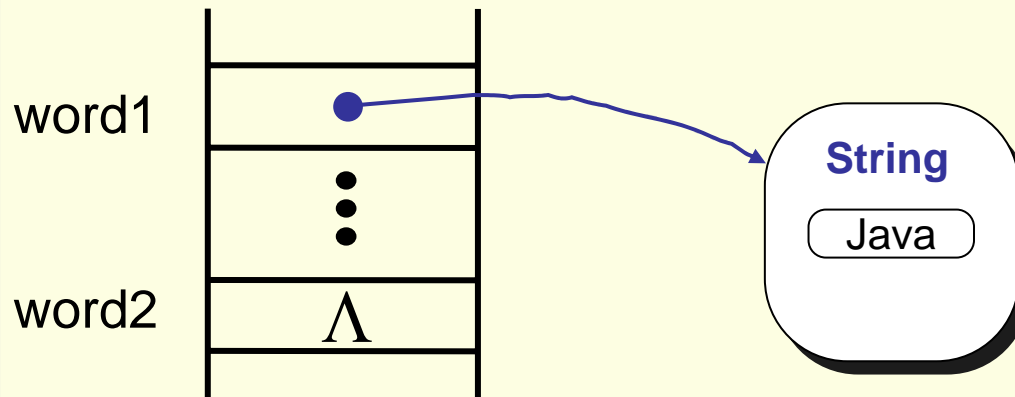
Code

B

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

One String object is created and assigned to word1, so word1 contains the address of this object.

State of Memory



After **B** is executed

-...String variables are References.

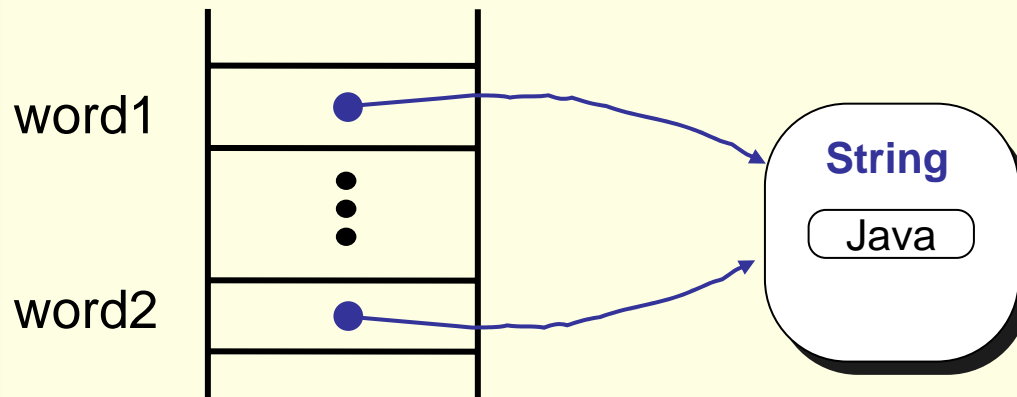
Code




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

Content of word1, which is an address, is assigned to word2, making word2 refer to the same object.

State of Memory



After  is executed



- String Methods

- String Concatenation
- String Length
- String Starting Position
- Other Useful String Operators



-- String Concatenation

- *Concatenation*: Using the `+` operator on two (or more) strings in order to connect them to form one longer string

```
String greeting = "Hello ";  
String course   = "ICS102";  
System.out.println(greeting + course);
```

```
Hello ICS102
```

- A new **String** object is created operands are not affected.
- When a string is combined with almost any other type of item, the result is a string

```
int x = 6, y = 10;  
String s = ".Hi.";  
System.out.println(s + x);  
System.out.println(x + y + s);  
System.out.println(s + (x + y));  
System.out.println(s + x + y);
```

```
.Hi.6  
16.Hi.  
.Hi.16  
.Hi.610
```



-- String Length

- We determine the number of characters in a String with the **length** method.

```
String name = "Espresso!";  
String str2 = "";  
String str3;
```

```
name.length();
```

→ 9

```
str2.length();
```

→ 0

```
str3.length();
```

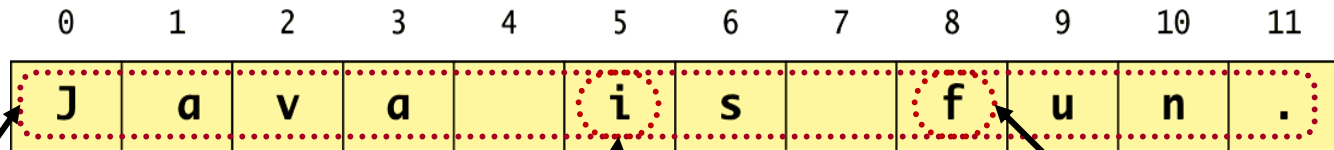
→ **Error!**

Error because no object is created for **str3**, so it is a **null**.

- String Starting Position

- Individual characters in a String can be accessed with the `charAt` method.
- Position of a character or String can be found with `indexOf` method.

```
String text = "Java is fun.";
```



`text`

This variable refers to the whole string.

`text.indexOf("is")`

The method returns the position of the string "is".

`text.charAt(8)`

The method returns the character at position # 8.



-- Other Useful String Operators

Method	Meaning
equals	Checks if two strings are equal. (Use equalsIgnoreCase for case insensitive) <code>str1.equals(str2)</code>
compareTo	Compares the two strings. <code>str1.compareTo(str2)</code>
substring	Extracts the a substring from a string. <code>str1.substring(1, 4)</code> <code>str1.substring(5)</code>
trim	Removes the leading and trailing spaces. <code>str1.trim()</code>
toUpperCase	Converts a string to all caps string. (Use toLowerCase for all small) <code>str1.toUpperCase()</code>



- Escape Sequences ...

- A backslash (\) immediately preceding a character (i.e., without any space) denotes an *escape sequence* or an *escape character*
 - The character following the backslash does not have its usual meaning
 - Although it is formed using two symbols, it is regarded as a single character



... - Escape Sequences

Display 1.6 Escape Sequences

`\"` Double quote.

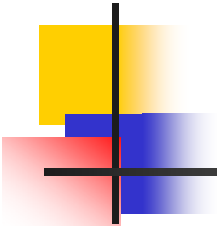
`\'` Single quote.

`\\` Backslash.

`\n` New line. Go to the beginning of the next line.

`\r` Carriage return. Go to the beginning of the current line.

`\t` Tab. White space up to the next tab stop.



THE END