BIT115: Introduction to Programming

Lecture 20 (Part 2)

The String Class

Instructor: Craig Duckett

Topics

- ***Strings***
- ***String Arrays***
- ***Difference between String Literal and String Object***
  - ***Difference between == and .equals() method***

The *String* Class

- Java has no primitive data type that holds a series of characters.
- The **String** class from the *Java Standard Library* is used for this purpose.
- In order to be useful, the a variable must be created to reference a **String** object. `String number;`
- Notice the **S** in **String** is upper case.
- By convention, class names should *always* begin with an upper case character.

Primitive vs. Reference Variables

- Primitive variables actually contain the value that they have been assigned.

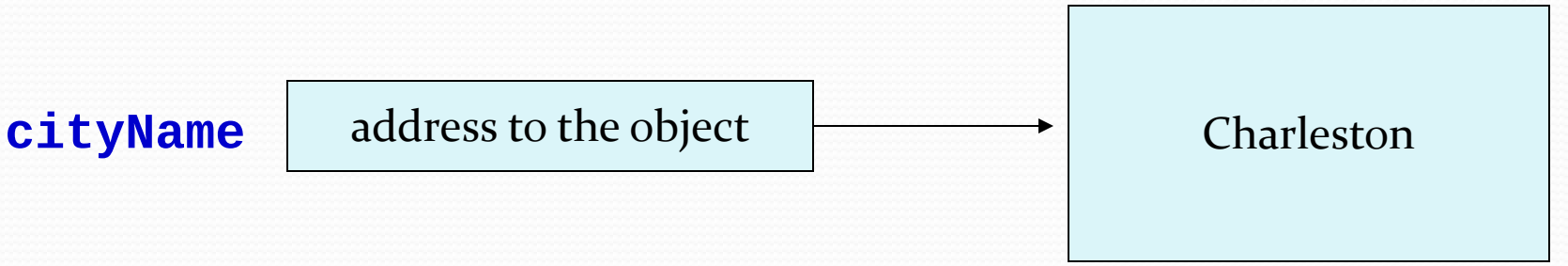
```
int number = 25;
```

- The value 25 will be stored in the memory location associated with the variable **`number`**.
- Objects are not stored in variables, however. Objects are *referenced* by variables.

Primitive vs. Reference Variables

- When a variable references an object, it contains the memory address of the object's location.
- Then it is said that the variable *references* the object.

```
String cityName = "Charleston";
```



The object that contains the character string "Charleston"

`String` Objects

- A variable can be assigned a string **literal**.

```
String value = "Hello,World!";
```

- **String** objects are the only objects that can be created in this way.
- A string **variable** can be created using the ***new*** keyword.

```
String value = new String("Hello, World!");
```

Here **`value`** string can be changed into a different string by Interacting somehow with the code in the programs

```
value = “Goodbye, World!“;
```

This is the method that all other objects must use when they are created.

See example: StringDemo.java

http://www.javaranch.com/journal/200409/ScjpTipLine-StringsLiterally.html

The `String` Methods

- Since **String** is a class, objects that are *instances* of it have methods.
- One of those methods is the **length** method.

```
stringSize = name.length();
```

- This statement calls the length method on the object pointed to by the **value** variable.

See example: StringLength.java

`String` Methods

- The **String** class contains many methods that help with the manipulation of **String** objects. Here are just a few:

`length()`	`toUpperCase()`	`startsWith()`
`charAt()`	`toLowerCase()`	`endsWith()`

- **String** objects are ***immutable***, meaning that they cannot be changed. They can be copied but they cannot be changed. Why is this a good thing? String Pool, Hash Code, Security

- Many of the methods of a **String** object can create new versions of the object.

For the complete list of **String** methods,
see: http://docs.oracle.com/javase/1.4.2/docs/api/java/lang/String.html

Here is another web page with a great String class tutorials:

http://www.tutorialspoint.com/java/java_strings.htm

See example: StringMethods.java

`String` Input Methods

The **String** class contains two **input** methods for getting input from the user

```
next()
nextLine()
```

The **next()** method gets a **single word** or until it encounters a **space**. For example, if the input was "Rex Winkus" it would only capture "Rex" as the input.

The **nextLine()** method will get the entire line of string, including all characters and spaces, until the **\n** character is encountered, or until it reaches end-of-line.

See examples:

- BasicStringInputExample.java
- StringInputExample.java
- StringInputExample2.java

String Arrays

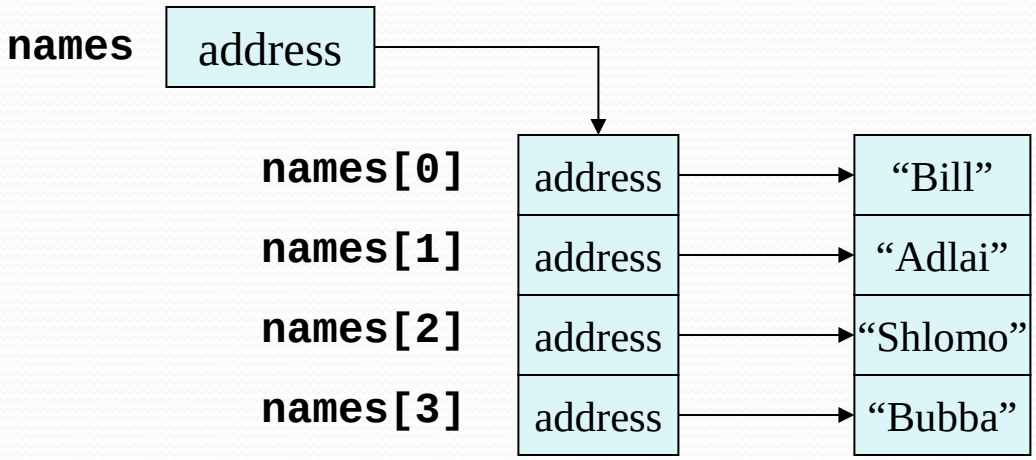
Java allows us to create arrays of String objects. Here is a statement that creates a String array initialized with values:

```
String[] names = {"Bill", "Adlai", "Shlomo", "Bubba"};
```

In memory, an array of String objects is arranged differently than an array of a primitive data type. To use a String object, you must have a reference to the string object. So, an array of String objects is really an array of references to String objects.

The **`names`** variable holds the address to the array.

A **`String`** array is an array of references to **`String`** objects.



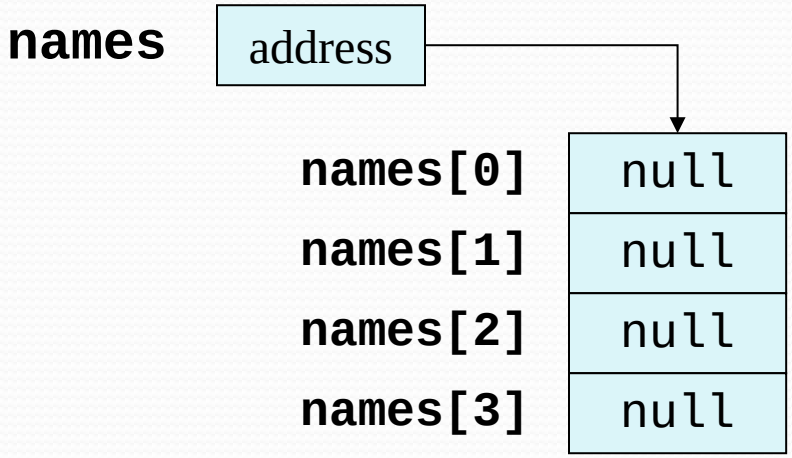
Example:
MonthDays.java

String Arrays

If an initialization list is not provided, the new keyword must be used to create the array:

```
String[] names = new String[4];
```

The **names** variable holds the address to the array.

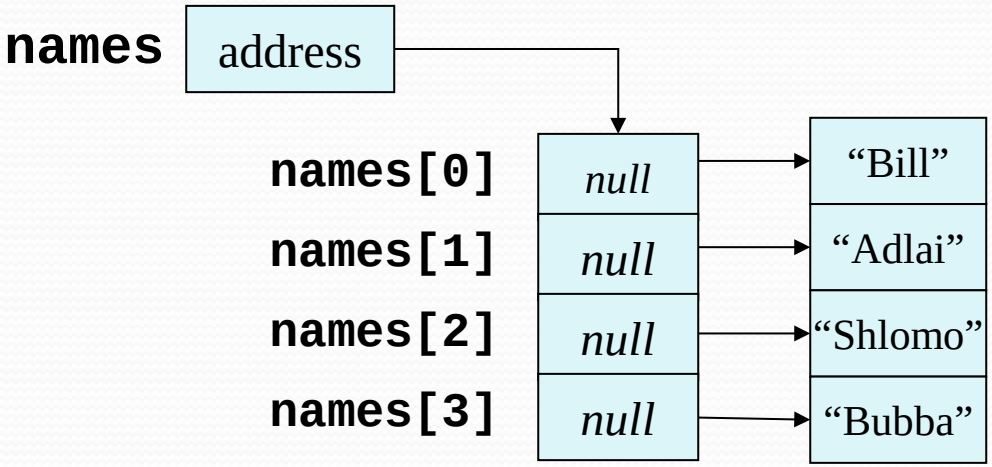


String Arrays

When an array is created in this manner, each element of the array must be initialized:

```
String[] names = new String[4];

names[0] = "Bill";
names[1] = "Adlai";
names[2] = "Shlomo";
names[3] = "Bubba";
```



Calling `String` Methods On Array Elements

- **`String`** objects have several methods, including:
  - **`toUpperCase`**
  - **`compareTo`**
  - **`equals`**
  - **`charAt`**
- Each element of a **`String`** array is a **`String`** object.
- Methods can be used by using the *array name* and *index* as before.

```
System.out.println(names[0].toUpperCase());
char letter = names[3].charAt(0);
```

See Example: StringArrayMethods.java

Whats the Difference Between == and equals() ?

IMPORTANT!

The **equals( )** function is a method of Object class which should be overridden by programmer.
String class overrides it to check if two strings are equal i.e. in **content** and not **reference**.
== operator checks if the references of both the objects are the same. Consider the programs:

```
String abc = "Awesome" ;
String xyz = abc;

if(abc == xyz)
System.out.println("Refers to same string");
```

Here the **abc** and **xyz**, both refer to same string **"Awesome"**. Hence the expression **(abc == xyz)** is **true**.

```
String def= "Hello World";
String stu= "Hello World";

if(def == stu)
  System.out.println("Refers to same string");
else
  System.out.println("Refers to different strings");
if(def.equals(stu))
  System.out.prinln("Contents of both strings are same");
else
  System.out.prinln("Contents of strings are different");
```

Here **abc** and **xyz** are two different strings with the same content **"Hello World"**.
Hence the expression **(abc == xyz)** is **false** where as **(abc.equals(xyz))** is **true**.

Check It! http://www.programmerinterview.com/index.php/java-questions/java-whats-the-difference-between-equals-and/
Check This Too! http://mindbugzz.blogspot.com/2012/05/what-is-difference-between-strings-and.html

See Example: StringLiteral.java

Extra Credit 01: String Class

Let's Have a Look at the Extra Credit

DUE LECTURE 23 Wednesday, December 14th