CS251L

REVIEW

2010.8.30 | Derek Trumbo | UNM

Arrays

- Example of array thought process in Eclipse

Arrays

- Multi-dimensional arrays are also supported by most PL's
- 2-dimensional arrays are just like a matrix (monthly accident counts for past decade, 0 == 2000):

```
int[][] accCount = new int[10][12];
accCount[0][0] = 3;
System.out.println(accCount[3][4]);
```

Arrays

- We can also provide the contents of an array when we declare it – but don't provide size information

```
int[][] multi = new int[][]
    {{1, 2, 3}, {4, 5, 6}};
```

Row			
0	1	2	3
1	4	5	6*

```
int asteriskCell = multi[1][2];
```

Strings

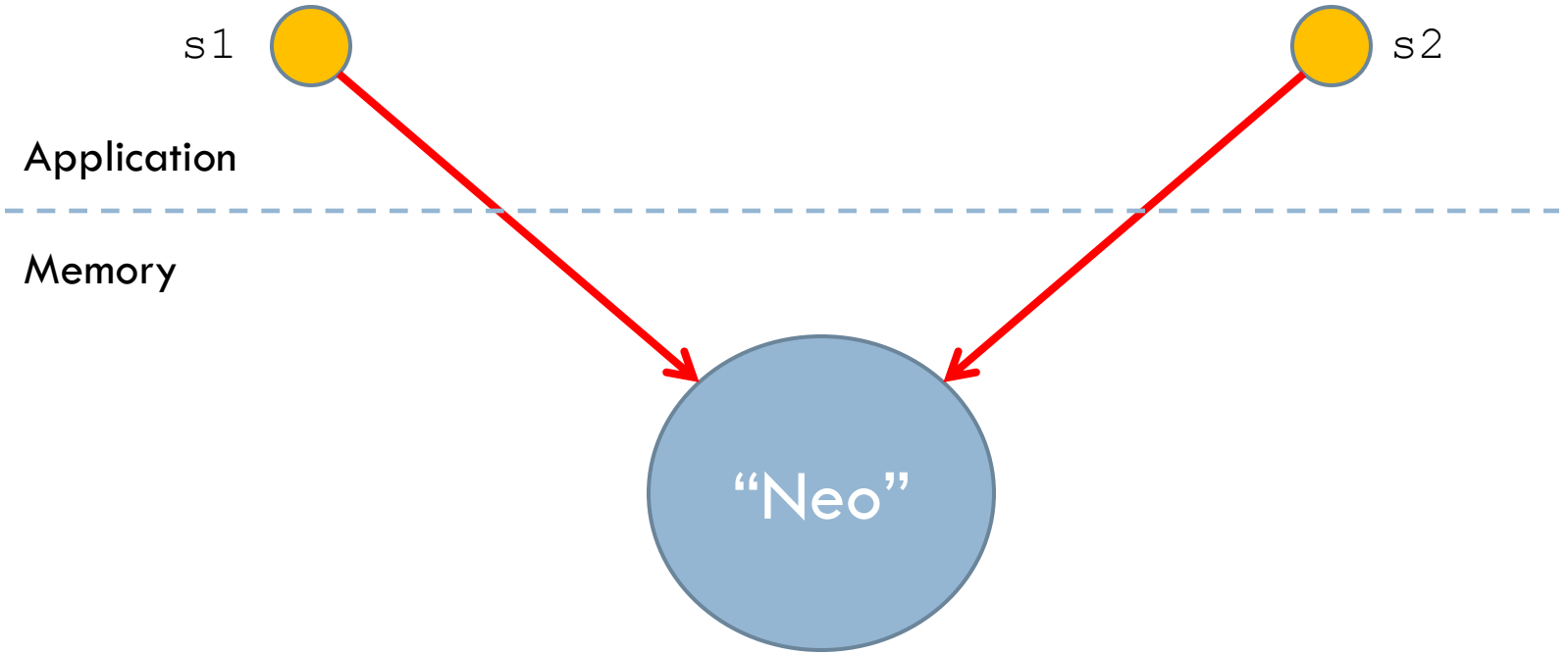
- Strings in Java are not primitives
- Strings are fully-fledged objects that encapsulate an array of characters
- Java uses “string pooling” to minimize duplication of String objects in memory
- String objects are *immutable*
  - Immutable objects are those objects whose internal state does not change after that object is initially created

Strings

- Just as == identifies if two primitive data types have the same value, the == when applied to two object references identifies if the references point to the *exact same object*
- However, when comparing strings in Java, we rarely care if two strings are the "same object" but rather care more if the two strings contain the *same characters*

Strings

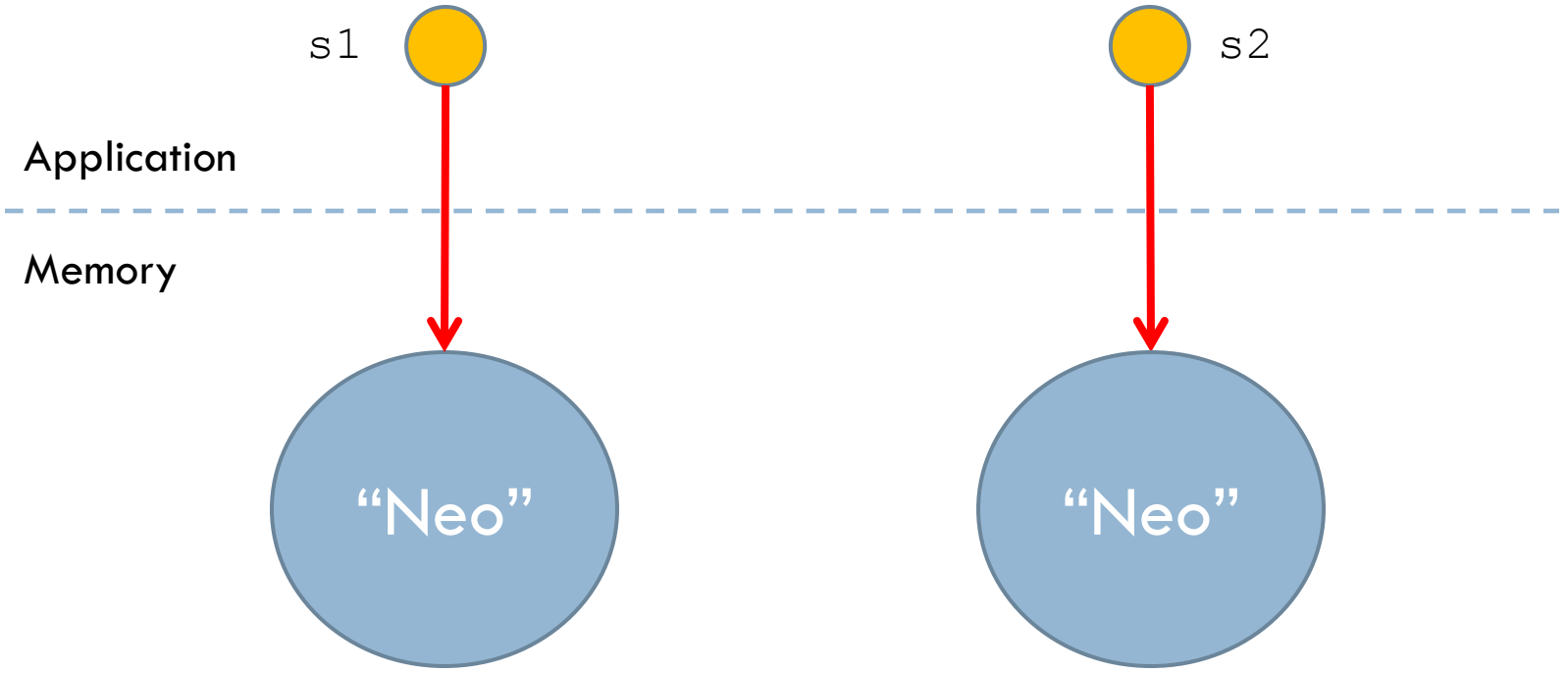
- `String s1 = "Neo"; String s2 = s1;`



- `s1 == s2 is true, s1.equals(s2) is true`

Strings

- `String s1 = "Neo"; String s2 = new String("Neo");`



- `s1 == s2 is false, s1.equals(s2) is true`

Strings

- A method, equals(), in the String class performs this comparison for you

```
String s1 = "ABC";
String s2 = "abc";
boolean same = s1.equals(s2);
boolean sameIC = s1.equalsIgnoreCase(s2);
```

Strings

- Become familiar with the Java API for the String class:
  - http://download.oracle.com/javase/6/docs/api/java/lang/String.html
- Some useful methods are:
  - charAt, endsWith, equals, equalsIgnoreCase, indexOf, lastIndexOf, length, startsWith, substring, toLowerCase, toUpperCase, trim

Strings

- Strings in Java get some royal treatment – the concatenation operator ("+") made special for them
- At the same level of precedence as the regular arithmetic + operator, this operator combines a String object and another primitive data type or object into a larger String object

```
String day = "Monday";
int date = 3;
double temp = 89.4;
String msg = day + "/" + date + ": T=" + temp;
System.out.println(msg);
```

Methods

- The code in useful programs can always be divided into areas of responsibility
- Most generic term for these are “procedures”
- In C/C++ these areas are called “functions” when they don’t belong to a class, and those within classes are called “member functions” (since they are members of the class)
- In Java, we call these areas of responsibility “methods” – and they always exist within some class

Methods

- Methods exist to decompose the problem into smaller units of work for many reasons:
  - Readability
  - Stability (debug-ability)
  - Code reuse (maintainability)

Methods

- Every method in Java exists within some class
- Methods have these parts:
  - Access modifier (public, protected, private, *package*)
  - Optional static keyword
  - Return type (any primitive data type or class reference)
  - Method name (in lower-camel case, e.g. “setTheThing”)
  - Parameter list (zero or more, comma-delimited)
  - Exception list (if the method declares that it throws ex.)
  - Method body (one or more lines of Java in {})

Methods

- Before you learn about object oriented programming, many methods you write in your apps may be “static”.
- This allows the method to execute without first having an instance of the class that’s in existence (more will be covered on this later on in course)

Methods

```
public class MyClass {
    public static void main(String[] args) {
        aaa();
    }
    public static void aaa() {
        bbb();
    }
    public static void bbb() {
        System.out.println("Hello World");
    }
}
```

Methods

```
public class MyClass {
    public static void main(String[] args) {
        aaa();
    }
    public static void aaa() {
        int result = bbb();
        System.out.println("bbb = " + result);
    }
    public static int bbb() {
        return 42;
    }
}
```

Methods

```
public class MyClass {
    public static void main(String[] args) {
        aaa(5);
    }
    public static void aaa(int param) {
        bbb("Saturn", param * 2.5);
    }
    public static void bbb(String a, double b) {
        System.out.println(a + b);
    }
}
```

Variables & Literals

- Variables are storage locations you can declare in code, whose values can change over time, and which are referenced by a name of your choosing
- Literals are those constant values of any primitive data type (and strings) that are placed directly into the code

Variables & Literals

```
public class MyClass {
   public static void main(String[] args) {
      double value = 40.05 * 2003.44 *
         2003.44 / 78;
      String msg = "Value is " + value + "!";
      System.out.println(msg);
   }
}
```

Variables & Literals

```
public class MyClass {
    public static void main(String[] args) {
        double value = 40.05 * 2003.44 *
            2003.44 / 78;
        String msg = "Value is " + value + "!";
        System.out.println(msg);
    }
}
```

Variables – both declarations and uses

Variables & Literals

```
public class MyClass {
    public static void main(String[] args) {
        double value = 40.05 * 2003.44 *
            2003.44 / 78;
        String msg = "Value is " + value + "!";
        System.out.println(msg);
    }
}
```

Literals – primitives and strings

Variables & Literals

```
public class MyClass {
    public static void main(String[] args) {
        double mass = 40.05;
        double velocity = 2003.44;
        int radius = 78;
        double centripetalForce = mass *
            Math.pow(velocity, 2) / radius;
        String msg = "Centripetal Force: " +
           centripetalForce;
        System.out.println(msg);
    }
}
```

Variables & Literals

```
public class MyClass {
    public static void main(String[] args) {
        double mass = 40.05;
        double velocity = 2003.44;
        int radius = 78;
        double centripetalForce = mass *
            Math.pow(velocity, 2) / radius;
        String msg = "Centripetal Force: " +
            centripetalForce;
        System.out.println(msg);
    }
}
```

Variables – both declarations and uses

Variables & Literals

```
public class MyClass {
    public static void main(String[] args) {
        double mass = 40.05;
        double velocity = 2003.44;
        int radius = 78;
        double centripetalForce = mass *
            Math.pow(velocity, 2) / radius;
        String msg = "Centripetal Force: " +
            centripetalForce;
        System.out.println(msg);
    }
}
```

Literals – primitives and strings