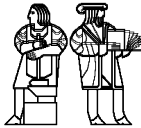


6.170 Lecture 2 Java semantics: Objects



Michael Ernst
MIT EECS

Outline

Declarations, assignments, method calls
Java primitives vs. objects
Variables vs. objects; null references
Mutable and immutable objects
Sharing
Equality testing
Intro to classes (more next time)

Michael Ernst/John Guttag

Spring 2001

Slide 2

Java compared to Scheme

Relax! Java is much like Scheme
garbage-collected, call-by-value, safe (runtime-checked)

Scheme's strengths:
easy to extend
read-eval-print loop

Java's strengths
statically typed (discover errors earlier)
built-in support for object-oriented programming
wider acceptance

You will need to learn
syntax, library routine names, OO programming

Michael Ernst/John Guttag

Spring 2001

Slide 3

Declarations, variables, method calls

Declaration creates a new variable
`String a;`

Assignment binds a variable to a value
`a = "mit";`
Evaluate an assignment by evaluating each side in turn; "mit" is a literal that evaluates to a String object

Method call invokes a procedure
`System.out.println(a);`
`String b = a.toUpperCase();`
`System.out.println(b);`
`int i = a.indexOf('i');`
`int j = b.indexOf('i');`
System.out is of type PrintStream
toUpperCase takes a String, returns a String
1 (indexing starts at 0)
-1 (by the Java spec)

Methods are invoked on objects (the "receiver"; `this`)

Michael Ernst/John Guttag

Spring 2001

Slide 4

Two types of values

Primitives: `int`, `float`, `char`, `boolean`, ...

`int x = y + 3;`
no user-serviceable parts within
created by writing literals
operations built into the Java language

Objects: `String`, `PrintStream`, `Vector`, ...
composed of other values
created by calling a constructor (or via a String literal)
`new Vector();` // empty sequence
`new Date(101, 2, 7);` // Feb 7, 2001
operations are associated with the type, can be extended

Michael Ernst/John Guttag

Spring 2001

Slide 5

Variables, references, and assignment

A variable is declared to hold:
a primitive value, or
a reference to an object

Uninitialized variables
`String a;`
`System.out.println(a);`
`String b = a.toUpperCase();`
a holds no reference
prints "null" (because println checks)
throws NullPointerException:
receiver must be an object

Michael Ernst/John Guttag

Spring 2001

Slide 6

Mutable and Immutable Objects

Immutable objects: can never be changed after creation

```
String tute = "mit";
tute.toUpperCase();
tute = tute.toUpperCase();
```

has no effect; result is discarded

Mutable objects: can be changed

```
Date now = new Date();
System.out.println(now);
now.setMonth(1);
System.out.println(now);
```

Wed Feb 07 2001
Sun Jan 07 2001

Assignment changes the variable binding

Mutation changes the object

Michael Ernst/John Guttag Spring 2001 Slide 7

Sharing (aliasing)

Multiple variables may refer to the same object

```
Vector v1 = new Vector();
Vector v2 = v1;
v1.addElement("mit");
v2.addElement("MIT");
System.out.println(v1.firstElement());
System.out.println(v2.lastElement());
```

```
Vector v3 = new Vector();
v3.addElement(v1.elementAt(0));
v3.addElement(v1.elementAt(1));
```

Michael Ernst/John Guttag Spring 2001 Slide 8

Equality

Object (reference) equality: == (like Scheme eq)

Value equality: equals(Object) (like Scheme equal)

```
if (v1 == v2) System.out.println("same object");
if (v1.equals(v2)) System.out.println("same value");
```

Should (x == y) imply x.equals(y)? Yes: same object => same value

Aliasing is only significant in the presence of mutability

Why do languages have (im)mutable types? immutable types simplify reasoning; mutable types can increase efficiency

What does this do?

```
Vector v4 = new Vector();
v4.addElement(v4);
Vector v5 = new Vector();
v5.addElement(v5);
System.out.println(v4.equals(v5));
System.out.println(v4);
System.out.println(v4.equals(v4));
```

infinite loop
infinite loop
true

Michael Ernst/John Guttag Spring 2001 Slide 9

New types

Declaring a new type of Object:

```
class Professor {
    String name;
    boolean tenured;
}
```

Creating a new object (instance) of the new type

```
Professor p1 = new Professor();
p1.name = "Michael Ernst";
p1.tenured = false;
System.out.println(p1);
```

Professor@47e553

```
// better: new Professor("John Guttag", true);
Professor p2 = new Professor();
p2.name = "John Guttag";
p2.tenured = true;
```

Michael Ernst/John Guttag Spring 2001 Slide 10

Adding methods: toString and outranks

```
class Professor {
    String name;
    boolean tenured;
    String toString() {
        if (tenured)
            return "Herr Dr. Prof. " + name;
        else
            return name;
    }
    boolean outranks(Professor p) {
        return this.tenured && (! p.tenured);
    }
}
```

toString is used by println; previously, println used a default for Professor; toString really needs a "public" modifier

"tenured" and "name" refer to fields in the receiver object

"this.tenured" is equivalent to "tenured"

```
System.out.println(p2);
System.out.println(p2.outranks(p1));
```

Herr Dr. Prof. John Guttag
true

Michael Ernst/John Guttag Spring 2001 Slide 11