

---

**CSE 142**

**Expressions and Statements**

---

1/9/2002 (c) University of Washington 2001-2 D1-1

---

**Overview**

---

- **Quick Review:**
  - Creating and using objects
  - Naming objects
  - Sending messages
- **New**
  - Reinforce the above concepts
  - Get a little bit more formal
- **Reading**
  - Dugan notes: first part of Ch. 5
  - Niño & Hosch: sec. 5.2.2

---

1/9/2002 (c) University of Washington 2001-2 D1-2

---

**Expressions**

---

- **Look at this statement:**

```
Rectangle rect = new Rectangle(10,20,30,40);
```
- **Remember our pattern for naming:**

```
<the type of thing> <the name> = <the thing we're naming>;
```
- **What is the stuff on the right of the '='? We call it an *expression*.**
- **We *evaluate* an expression to compute a *value*.**
- **The name is then *bound* to the *value* of the expression.**

---

1/9/2002 (c) University of Washington 2001-2 D1-3

---

**Legal Expressions**

---

- **What are legal expressions?**
  - a literal representation of a value
  - the creation of a new object
  - a name of an object (also called an *identifier* or *variable name*)
  - the result of sending a message to an object
  - combinations of the above (we'll see how to combine them later)
- **Examples**

```
1  
"hello"  
aSquare  
aSquare.getX()  
new Rectangle(10, 20, 30, 40)
```

---

1/9/2002 (c) University of Washington 2001-2 D1-4

---

**Statements**

---

- **Most programs need to do a *sequence* of things. In Java, we do this by writing a *sequence* of *statements*:**

```
int side = 20;  
Rectangle aSquare = new Rectangle(side, side, 100, 200);  
aSquare.moveBy(35, 10);
```
- **A semicolon *terminates* a statement. Semicolons are like the "." (period or full stop) in written English.**
- **The machine *evaluates* one statement at a time.**
- **Unlike expressions, a statement is evaluated for *effect*, not *value*.**

---

1/9/2002 (c) University of Washington 2001-2 D1-5

---

**Arithmetic Operators**

---

- **Java provides *arithmetic operators* so we can build mathematical expressions:**

| Symbol | Meaning   | Example | Value (if y=11) |
|--------|-----------|---------|-----------------|
| +      | add       | y + 5   | 16              |
| -      | subtract  | y - 5   | 6               |
| *      | multiply  | y * 5   | 55              |
| /      | divide    | y / 5   | 2               |
| %      | remainder | y % 5   | 1               |

---

1/9/2002 (c) University of Washington 2001-2 D1-6

### Binary and Unary Expressions

- We call the above *binary operators*, because they operate upon two *subexpressions*:

```
<argument expression> <binary operator> <argument expression>
```

- The "-" symbol can be used to negate values as well:

```
int negX = -x;
```

- It's a *unary operator*, because it operates upon only one *subexpression*:

```
<unary operator> <argument expression>
```

1/9/2002

(c) University of Washington 2001-2

D1-7

### Precedence

- Precedence follows normal math rules. What are they?

- If you're unsure, or wish to override, use parentheses.

```
int x = 2;  
int y = 4;  
int m = x + y * 8;  
int n = (x + y) * 8;
```

1/9/2002

(c) University of Washington 2001-2

D1-8

### Division

- Division seems a little strange in Java.
- What is 5 divided by 2?

```
int x = 5;  
int y = x / 2;
```

- Division between integers is *integer division* (no fractions)!
- If you want the remainder, use the % operator.
- If you want to represent a fractional amount, use a different kind of number (like a double)

1/9/2002

(c) University of Washington 2001-2

D1-9