CSE 142 Summer 2001

Objects, Expressions, Types, Messages

Introduction

- Quick Review:
  - Creating and using objects
  - Naming objects
  - Sending messages
- In this lesson:
  - Reinforce the above concepts
  - Get a little bit more formal

Expressions

- Look at this statement

```
Rectangle rect = new Rectangle(10,20,30,40);
```

- Remember our pattern for naming:

```
<The kind 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 *bound* to the *value* of the expression.

Expressions 2

- What are legal expression?
  - a literal representation of an value
  - the creation of a new object
  - a name of another object
  - 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.width( )
new Rectangle(10, 20, 30, 40)
```


Arithmetic Operators

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

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

Arithmetic Operators (2)

- We call the above *binary operators*, because the operate upon two *subexpressions*.
- The "-" symbol can be used to negate values as well:

```
int x = 5 * - y;
```

- Precedence works like normal math rules. If you're unsure, or wish to override, use parenthesis:

```
int x = 2;
int y = 4;
int m = x + y * 8;          // What's m ? (this is a "comment")
int n = (x + y) * 8;        // What's n ?
```

Operator Patterns

- All binary operators are used in the following pattern:

```
<expression> <operator> <expression>
```

- Where <operator> is one of +, -, *, /, or %

Division

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

```
int x = 5;
int y = x / 2;      // What's y?
```

- 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)

Sequence of Statements

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

```
int width = 20;
Rectangle square = new Rectangle(width, width, 100, 200);
square.move(35, 10);
```

- The above example has three *statements*. A semicolon *terminates* a statement.
- Semicolons are like the "." (full stop) in written English.
- The machine evaluates one statement at a time (for effect, not value). Different than expressions!

Types

- Some more expressions:

```
"hello"
aSquare
aSquare.width( )
aSquare.length( ) * 24
```

- What kind of thing does each expression evaluate to?
- Java calls "kinds of things" *types*.
- When we create a new name, we must always provide a type:

```
String greeting = "hello";
```

- We can now write our naming pattern more formally:

```
<Type> <name> = <expression>;
```


Type Mismatches

- What's wrong with this sentence: "Her age is green."
- Java is extremely picky about these kinds of errors, which we call *type mismatches*. For example:

```
int myAge = "green";
String greeting = 23;
```

- Why do you think that Java is picky about this kind of thing?

Kinds of Errors

- A couple more Bushisms:
  - Is our children learning?
  - We ought to make the pie higher.
- Both fail to convey meaning, but why?
  - The first has a *syntax error* (a grammar mistake)
  - The second has a *semantic error* (a misuse of types): we think of pies getting bigger and smaller, not higher
- Both of these kinds of errors are illegal in Java.

Syntax & Semantic Errors

- What is wrong with each of these statements? Syntax or semantics?

```
int x 7;

String myName = 7;

aSquare.move("hello", 40);

int velocity = distance / ;

String greeting = "hello" 56;

int area = length * "width";
```


Method Parameters

- Many messages require some information:

```
aRectangle.move(10, 20);
```

- We call the items we send with a message *parameters* (or sometimes *arguments*)
- Each parameter may be any expression, as long as the type and number of parameters matches what the method expects:

```
aRectangle.move("hello", 20);
aRectangle.move(30);
```


Method Interface

- How do we know the right way to send a message?
- To really know (and how Java knows): look at the *interface*.
- An interface defines: method name, number, type of parameters, and type of the resulting value. It often also has commentary about the behavior of the method.

Method Interface: Examples

- Here's an example for the move method for Shapes:

```
// Change the position of the shape by the given deltas.
// Parameters:
//   deltaX: the distance in the X direction to move
//   deltaY: the distance in the Y direction to move
void move(int deltaX, int deltaY);
```

- What does void mean?
- Here's an example for the length method for Strings:

```
// Answer the length of the string
int length( );
```