Expressions

CSE 142, Summer 2002

Computer Programming 1

http://www.cs.washington.edu/education/courses/142/02su/

Readings and References

- Reading
  - Chapter 5, *An Introduction to Programming and Object Oriented Design using Java*, by Niño and Hosch
  - Chapter 5, *Introduction to Programming in Java*, Dugan
- Other References
  - http://java.sun.com/docs/books/tutorial/java/nutsandbolts/opsummary.html
  - http://java.sun.com/j2se/1.4/docs/tooldocs/windows/javadoc.html

Statements

```
public class Dog  {
 public Dog(int rate) {
     consumptionRate=rate;
     weight = 20;
 }
 public void bark() { . . . }
 public int getRate() { . . . }
 public void eat(int pounds) { . . . }

 int consumptionRate;
 int weight;
}
```

these are the "statements" that make up the body of a constructor or method

dog.java

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 in Java are like the "." (period or full stop) in written English.
- The machine evaluates one statement at a time.
- Statements can be grouped into blocks using curly braces
  - { … }

Expressions

- Expression
  - An expression describes how to compute a particular value
  - Evaluation of an expression produces a value
- An assignment statement takes a value produced by an expression and assigns the value to a declared variable in a program

```
Double area = PI * radius * radius;

int index = base + increment;

greeting = "hello " + userName;
```

Some Valid Expressions

- a literal representation of a value
  `7, boolean, "hello"`
- the creation of a new object
  ```
  new AlarmClock("ringin.wav")
  new Dog(4)
  ```
- a name of an object (also called an *identifier* or *variable name*)
  `base, increment`
- the result of sending a message to an object
  ```
  bowser.getRate()
  aSquare.getX( )
  ```
- combinations of expressions are created using operators
  `PI*radius*radius`

MaxValues.java

Arithmetic Operators

- Java provides *arithmetic operators* so we can build mathematical expressions:
  - assume y is equal to 11 when the expression is evaluated

Symbol	Meaning	Example	Value (for y=11)
+	add	y + 5	16
-	subtract	y - 5	6
*	multiply	y * 5	55
/	divide	y / 5	2.2 or 2
%	remainder	y % 5	1

http://java.sun.com/docs/books/tutorial/java/nutsandbolts/opsummary.html

Division

- Most of the arithmetic operators work as you would expect
  - add, subtract, and multiply
- You have to be a little more careful with division
  - `double` values will act as you expect them to
  - 5.0 / 2.0 is equal to 2.5
  - But remember that `int` values are integers and cannot hold any fractional part
- So what is integer 5 divided by integer 2?

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

y will have the value 2 at this point, not 2.5

`1.0 + ( 7 / 8 )` is equal to what?

Remainder

- Sometimes you want to know what was left over after an integer division
  - Recall this: value = quotient * divisor + remainder
- Say that you want to know the remainder, not the quotient
- For example

```
int x = 7;
int y = x / 2;
```

  - y will have the value 3 at this point, but we want to know the remainder
- The remainder operator is %

```
int rem = 7 % 2;
```

`rem` will have the value 1 at this point since 7-(3*2) is equal to 1

Binary and Unary Expressions

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

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

```
5 * 3
(a+b)*(c/d)
```

- Most operators are binary operators
- A *unary* operator operates upon only one subexpression:

  <unary operator> <argument expression>

- For example, the "-" symbol can be used as a unary operator to negate values:

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

Precedence

- How does this expression get evaluated?

```
(a+b)*(c/d)
```

  - First (a+b) is evaluated, then (c/d) is evaluated, then the two values are multiplied together

- How does this expression get evaluated?

```
a+b*c/d
```

  - First b*c is evaluated, then that value is divided by d, then the result is added to a

Precedence

- Java evaluates *, /, and % before + and -
  - ie, multiply, divide, and remainder before add and subtract
  - these are well defined rules and so the compiler will always know exactly what to do
  - but you or another programmer may not read it exactly the same way as the compiler does
- Use parentheses unless it's *really really super duper obvious* what the evaluation order is
  - They don't cost a thing, they don't slow down the program, and if they save one programmer from misunderstanding the code they have done a great service!

Document your source code!

- If you write comments as you go along, then the documents are done when the code is done!
  - This will earn you the eternal gratitude of your boss at work …
- Java provides a very nice tool called javadoc that reads your code and produces html web pages describing it
  - some of the information is from your comments
  - some of the information is from the structure of the code itself
- BlueJ invokes the javadoc tool when you use the “interface” pull down menu item in the source code editor
- http://java.sun.com/j2se/1.4/docs/tooldocs/windows/javadoc.html

Documenting Source Code

- Java provides several ways to indicate that you are writing a comment instead of source code
- // - single line comment
  - everything after the // is ignored
- /* multiple line comment */
  - everything between the /* and */ is ignored, no matter how many lines it takes
- /** javadoc style comment */
  - javadoc expects to find information that it can build a description from
  - These comments can be very fancy, but simple comments provide 80 to 90 % of all the information needed

Javadoc Tags

- A javadoc comment applies to the element of the class that follows the comment
- The comment should provide basic information necessary to use the class, the field, or the method
- The javadoc utility supports several “tag” fields in javadoc comments
  - @param -- passed parameter description
  - @return -- returned value description
  - @author -- author
  - @throws -- possible error conditions

Dog.java

```
/**
 * Sample class for demonstrating class structure.
 */
public class Dog {
    /**
    * Create a new Dog.  This constructor supplies a default weight of 20 pounds.
    * @param rate the rate at which this dog eats, specified in pounds/fortnight.
    */
    public Dog(int rate) {
        consumptionRate = rate;
        weight = 20;
    }
    /**
    * Provide this dog with a voice.
    */
    public void bark() {
        System.out.println("Woof! Woof!");
    }
    /**
    * Provide this dog with a way to rest his bones.
    */
    public void sleep() {
        System.out.println("Snrf ... woof ... snrf ...");
    }
```

Initial comment describes the overall purpose of the class.

This comment describes this particular constructor.

This comment describes this particular method.

Dog.java

```
/**
 * Eat some goodies.  There is some weight gain after eating.
 * @param pounds the number of pounds of food provided.
 */
public void eat(int pounds) {
    double coverage = (double)pounds/(double)consumptionRate;
    String foodUnits = (pounds == 1) ? "pound" : "pounds";
    String timeUnits = (coverage == 1) ? "fortnight" : "fortnights";
    System.out.println(
        Integer.toString(pounds)+" "+foodUnits+
            " lasted "+coverage+" "+timeUnits+".");
    weight += (double)pounds/2;
    System.out.println("Weight is now "+weight+" pounds.");
}
/**
 * Retrieve the rate value for this Dog.
 * @return the consumption rate specified for this Dog.
 */
public int getRate() {
    return consumptionRate;
}
```

Note the use of the @param tag to describe “pounds”

Note the use of the @return tag to describe the value that this method returns to its caller

Dog.java

```
/**
 * Run this guy through his day.  This is a simple test harness for this class.
 * @param args ignored
 */
public static void main(String[] args) {
    Dog rover = new Dog(28);
    rover.sleep();
    rover.bark();
    rover.eat(2);
    rover.bark();
    rover.eat(14);
}
//-------------------------------------------------------------------
/** the consumption rate specified for this dog.  Given in
 * pounds per fortnight.  A fortnight is two weeks or 14 days.
 */
int consumptionRate;

/** the weight of this dog.  Assumed to be in pounds. */
int weight;
}
```

You can describe data fields (variables) also by using javadoc comments.