Week 2 - Lecture Notes

Codegain

March 11, 2013

1 More Operators

Unary Operators are operations you perform on 1 variable.

Increment Operator (++): adds 1 to the value of a variable

Decrement Operator (−−): subtracts 1 from the value of the variable

Example:

```
int x = 1;
int y = 5;
x++;
System.out.println("x is now " + x);
y--;
System.out.println("y is now " + y);
```

2 More on Types

Division performs operations differently on integers and doubles.

Example:

```
public class MathPractice2 {
    public static void main ( String arguments[] ) {
        double y;
        int x = 6 / 3;
        System.out.println("x = " + x);
        x = 6 / 4;
        System.out.println("x = " + x);
        x = 1 / 2;
        System.out.println("x = " + x);
```

```
        y = 6 / 3;
        System.out.println("y = " + y);
        y = 6 / 4;
        System.out.println("y = " + y);
        y = 1 / 2;
        System.out.println("y = " + y);
    }
}
```

Integer division discards everything after the decimal point in the number.

- **Note:** It is not rounding the number. Example: $10/3 = 3$ (not 3.3333...) and $11/3 = 3$ (not 3.6666...)
- Modulo (%) can be used to extract the remainder when performing integer division Example:

```
System.out.println("10 divided by 3 is " + (14 / 3));
System.out.println("with a remainder of " + (14 % 3));
```

Dividing with whole number constants can cause an incorrect output that can go unnoticed

Example:

```
float totalMins = secs / 60.0;
```

This works because we have added the decimal point in the constant number 60.0. However, if we were to write:

```
float totalMins = secs / 60;
```

we would get an incorrect assignment because integer division would be performed and the numbers after the decimal point would be discarded.
Remember, Java performs type checking. The types must be compatible.

```
String five = 5;        // error
boolean isOpen = "true";  // error
```

3 Type Casting

We can type cast a variable to temporarily change its type to suit our needs.

Example:

```
int x = 9;
int y = 2;
double result = x / (double) y;
```

Without the type cast, the value of result would be 4. However, by temporarily widening the type for y from int to double, we get the correct value of 4.5.

- Type casting does not change the value of the variable itself.
- Smaller type -¿ larger type = Widening
- Larger type -¿ smaller type = narrowing

Java performs automatic type casts called type coercions to widen types

- Explicit type cast otherwise in order to narrow types

```
double d = 5;           // converts 5 to 5.0 and stores that as the value of d
int i = 5.5;            // illegal
int i = (int)5.5;       // legal, casts 5.5 to int making it 5
double d = 3/4          // d = 0.0
double d = (double)3/4  // d = 0.75
```

4 Method Structure

Java is all about Object-Oriented programming meaning we create objects to store data and we can perform actions with them.

- Classes are different categories of objects
- Methods are the actions these objects can perform

Example of Method Use:

```
String name = "Billy";                          // name is a String class object
int nameLength = name.length();        // length() is the method we can call with this objec
```

Main Method Syntax

```
public static void main(String[] arguments) {
    System.out.println("hello world");
}
```

Method Syntax

```
public static void METHOD_NAME () {
    // Write your statements here
}
```

We can add our own methods as well:

```
public class HelloTwice {
    public static void sayHello() {
        System.out.println(Hello);
    }
    public static void sayHelloTwice() {
        sayHello();
        sayHello();
    }
    public static void main (String[] arguments) {
        System.out.println("Say hello twice please:");
        sayHelloTwice();
    }
}
```

5 Parameters

Methods can also have required inputs to be used for execution. These inputs are called parameters.

- The value that is used as input for the parameter is called an argument

We've used parameters before like in:

```
System.out.println("Hello world");
```

The String "Hello world" is the argument we're using as input.

Structure of Method With Parameter

```
public static void METHOD_NAME (TYPE NAME) {
    // Write your statements here
}
```

Example:

```
public SquareNumber {
    public static void printSquare(int x) {
        System.out.println(x * x);
    }
    public static void main (String[] arguments) {
        int number = 4;
        printSquare(number);
        printSquare(3);
        printSquare(hello);
        printSquare(5.5);
    }
}
public SquareNumber2 {
    public static void printSquare(double x) {
        System.out.println(x * x);
    }
    public static void main (String[] arguments) {
        printSquare(5);
    }
}
```

Structure of Method with Multiple Parameters

```
public static void METHOD_NAME (TYPE_1 NAME_1, TYPE_2 NAME_2, ...) {
    // Write your statements here
}
class Add {
    public static void printSum (double a, double b) {
        System.out.println(a+b);
    }
    public static void main (String[] arguments) {
        printSum(2, 4);
        printSum(50, 30);
    }
}
```

6 Return Values

Methods can also return values as well

- The type of the data returned by the method must be specified in the method signature
- If the method returns no value, then the keyword void is used in place of a type.

- Calling a method that returns a value can be used as an expression as long as the type returned is valid.

Structure of Method with Return Value

```
public TYPE_RETURNED METHOD_NAME() {
    // Write your statements here
    return EXPRESSION;
}
```

Structure of Method with Void Return Value

```
public void METHOD_NAME() {
    // Write your statements here
}
```

Example (assume methods are in a class definition):

```
public int getAge() {
    int age = 10;
    return age;
}
public String getName() {
    return "Billy";
}
public void printGoodbye() {
    System.out.println("Goodbye!");
}
int myAge = getAge();
String myName = getName();
System.out.println(myName + " is " + myAge + " years old.");
printGoodbye();
```

Another Example:

```
class Square {
    public static double getSquare(double a) {
        return a * a;
    }
    public static void main(String[] arguments) {
        double a = 3.5;
        double squaredA = getSquare(a);
        System.out.println(a + " squared equals " + squaredA);
    }
}
```

7 Variable Scope

Variables that are declared in a block () are confined in that block (scope)

- These variables are called local variables
- Method parameters are an example of defining local variables in a method
- If two methods have a local variable with the same name, they are still two different variables which just happen to have the same name

Example:

```
class SquareNumber {
    public static void printSquare(int x) {
        System.out.println("printSquare x = " + x);
        x = x * x;
        System.out.println("printSquare x = " + x);
    }
    public static void main(String[] arguments) {
        int x = 5;
        System.out.println("main x = " + x);
        printSquare(x);
        System.out.println("main x = " + x);
    }
}
```

8 If Statement

In order to handle the flow of control within a program, Java employs branches and loops.

- Java branching and looping are controlled by boolean expressions (true or false values)
- If-statements are a way to branch the flow of a program one way or another.

Syntax:

```
if (CONDITION) {
    // STATEMENTS
}
```

A conditional statement is used to determine whether a program enters an if-block.

- Conditional statements are statements that use comparison operators which return boolean values

$x > y$: x is greater than y $x < y$: x is less than y $x >= y$: x is greater than or equal to y $x <= y$: x is less than or equal to y $x == y$: x equals y $x! = y$: x is not equal to y Example:

```
x = 5;
if (x == 5) {
    System.out.println("x equals 5");
}
if (x < 5) {
    System.out.println("x is less than 5");
}
if (x <= 5) {
    System.out.println("x is less than or equal to 5");
}
```

Note: To compare strings, use the equals() method instead of ==.

- Although == can be used to compare primitive types, it behaves differently when used with Objects. All strings are a part of the class String, so == doesn't check for the actual contents of the string but whether they are referencing the same object.

Example of String Comparisons:

```
String a = "hello";
String b = "Hello";
if (a.equals(b)) {
        System.out.println("The strings are equal");
}
if (a.equalsIgnoreCase(b)) {
    System.out.println("The strings are equal");
}
```

9 Combining Boolean Expressions

We can also combine several Boolean expressions to create more elaborate Boolean expressions:

- && is used as the logical AND operator When using &&, both Boolean expressions must be true for the entire expression to be true. Otherwise, it is false
- || is used as the logical OR operator.
- When using ||, only one of the Boolean expressions needs to be true for the entire expression to be true. If both of the smaller Boolean expressions are false, then the entire expression is false.

a	b	result
true	true	true
true	false	false
false	true	false
false	false	false

Table 1: AND truth table

a	b	result
true	true	true
true	false	true
false	true	true
false	false	false

Table 2: OR truth table

a	result
true	false
false	true

Table 3: NOT truth table

Example:

```
int x = 5;
if (x < 10 && x > 0) {
    System.out.println(x);
}
if (x != 5 || x > 0) {
    System.out.println(x);
}
```

NOTE: It is syntactically incorrect to write a series of inequalities. For instance, if you had rewritten the above example as $0 < x < 10$, you would have received a compiler error.

10 Else Statements

Else Statements are an optional block of code that come after an if-statement block

- Only when the preceding if-statement is false will the code within the else block be executed.

Syntax:

```
if (CONDITION) {
    STATEMENTS
}
else {
    STATEMENTS
}
```

Example:

```
String name = "Bruce";
if (name.equals("Alfred") {
    System.out.println("Hi Alfred");
} else {
    System.out.println("Hi " + name);
}
```

Note: Variables that are initialized inside of if-else blocks are local to that block only. Therefore, the variable's scope is limited to its if-else block. Example:

```
int x = 5;
if (x < 10) {
    int y = 10;
    System.out.println(x);
}
System.out.println(x);
System.out.println(y);
```