CMSC 131: Chapter 3 (Supplement)
Variables, Operators, and Control Flow

Variables

What is a variable?
- The name of some location of memory used to hold a data value.
- Different types of data require different amounts of memory.

Example:

```java
int width = 3;
int height = 4;
int area = width * height;

width = 6;
area = width * height;
```

Primitive Data Types

Java's basic data types:

**Integer Types:**
- `int` The most common integer type
- `byte, short` For small values
- `long` For huge values.

**Floating-Point Types:**
- `float` Roughly 7 digits of precision
- `double` Roughly 15 digits of precision

**Other types:**
- `boolean` (true, false)
- `char` A single (Unicode) character

**String (String):** is not a primitive type.
Data Types and Variables

**Strong Type Checking:** Java checks that all expressions involve compatible types.

```java
int x, y;  // x and y are integer variables
double d; // d is a double variable
String s; // s is a string variable
boolean b; // b is a boolean variable
char c;   // c is a character variable

x = 7;    // legal (assigns the value 7 to x)
b = true; // legal (assigns the value true to b)
c = '#';  // legal (assigns character # to c)
s = "cat" + "bert"; // legal (assigns the value "catbert" to s)
d = x - 3; // legal (assigns the integer value 7 - 3 = 4 to double d)

b = 5;    // illegal (cannot assign int to boolean)
y = x * b; // illegal (cannot add int and boolean)
c = x;    // illegal (cannot assign int to char)
```

Common Numeric Operators

**Arithmetic Operators:**
- Unary negation: \(-x\)
- Multiplication/Division: \(x \times y\), \(x / y\)
  - Division between integer types **truncates** to integer: \(23 / 4 \rightarrow 5\)
  - \(x \times y\) returns the **remainder** of \(x\) divided by \(y\): \(23 \times 4 \rightarrow 3\)
- Addition/Subtraction: \(x + y\), \(x - y\)

**Comparison Operators:**
- Equality/Inequality: \(x = y\), \(x \neq y\)
- Less than/Greater than: \(x < y\), \(x > y\)
- Less than or equal/Greater than or equal: \(x \leq y\), \(x \geq y\)

Common String Operators

**String Concatenation:** The `+` operator **concatenates** (joins) two strings.

- When a string is concatenated with another type, the other type is first evaluated and **converted** into its string representation.

**String Comparison:** Let \(s\) and \(t\) be strings.

- \(s.equals(t)\): returns true if \(s\) equals \(t\).
- \(s.compareTo(t)\): compares strings **lexicographically** (dictionary order)

```java
result < 0  // if s is less than t
result == 0  // if s is equal to t
result > 0  // if s is greater than t
```
Converting (Parsing) Strings to Numbers

Parsing: Convert a string to a numeric type.

String → int:
    int year = Integer.parseInt("2004");    // year = 2004

String → float:
    float weight = Float.parseFloat("175.35");    // weight = 175.35

String → double:
    double pi = Double.parseDouble("3.1415926");    // pi = 3.1415926

Example: Enter height from JOptionPane and convert to a float.

    String heightString = JOptionPane.showInputDialog(null, "Enter height");
    float height = Float.parseFloat(heightString);

Control Flow and Conditionals

Control flow:
    - Conditionals:
    - Loops:

The if statement:

    if ( inchesOfSnow > 7 )
        System.out.println("I'm staying home");

The if-else statement:

    if ( inchesOfSnow > 7 )
        System.out.println("I'm staying home");    // if snow > 7
    else
        System.out.println("I'm staying home anyway");    // if snow <= 7
More on Conditionals

Basic Structure:

if ( (conditional expression) ) (executed if condition is true)

or

if ( (conditional expression) ) (executed if condition is true)
else (executed if condition is false)

Logical Operators:

Logical "and": &&
if ( temp >= 97 && temp <= 99 )
System.out.println( "Patient is healthy" );

Logical "or": ||
if ( months >= 3 || miles >= 3000 )
System.out.println( "Change your oil" );

Logical "not": !
if ( ! phone.equals( "301-555-1212" ) )
System.out.println( "Sorry, wrong number" );

More on Conditionals

Block statement:

if ( totalHours > 40 ) {
    // worked overtime?
    stdHours = 40;
    overtimeHours = totalHours - 40;
} else {
    // no overtime
    stdHours = totalHours;
    overtimeHours = 0;
}

pay = (stdHours * rate) + (overtimeHours * (1.5 * rate));

Nested Conditionals:

/* computes the minimum of a, b, and c */
if ( a < b ) {
    // b is not the minimum
    if ( a < c ) minimum = a;
    else minimum = c;
} else {
    // a is not the minimum
    if ( b < c ) minimum = b;
    else minimum = c;
}
Example of Conditionals

```java
/**
 * An example using JOptionPane and conditionals
 */
import javax.swing.*;

public class JOptionPaneExample {
    public static void main(String[] args) {
        int answer = JOptionPane.showConfirmDialog(null, "Isn't Java great?");
        if (answer == JOptionPane.YES_OPTION)
            JOptionPane.showMessageDialog(null, "Wise choice.");
        else
            JOptionPane.showMessageDialog(null, "Wrong answer.");
        System.exit(0);
    }
}
```

Another Example of Conditionals

```java
/**
 * A simple intelligence test
 */
import javax.swing.*;

public class SimpleTest {
    public static void main(String[] args) {
        String choice = JOptionPane.showInputDialog("What is the world's greatest university? (hint: UMC?)");
        if (choice.equals("UMC")) // correct response
            JOptionPane.showMessageDialog(null, "Wise choice.");
        else // incorrect response
            JOptionPane.showMessageDialog(null, "Sorry, you blew it.");
        System.exit(0); // terminate the program
    }
}
**while and do-while Loops**

while and do-while loops are used to perform repetitive, or iterative, operations.

**while-loop**: The condition is tested at the top of the loop.
while ( <conditional expression> )
  (executed as long as the condition is true)

**do-while -loop**: The condition is tested at the bottom of the loop.
do
  (executed as long as the condition is true)
while ( <conditional expression> );

**while Loop Example**

**Task**: Print the statement "x bottles of beer on the wall" for x running from 10 down to 0.

```java
int x = 10; // initialize x
while ( x >= 0 ) { // check that x is greater or equal to 0
    System.out.println( x + " bottles of beer on the wall" );
    x = x - 1; // decrease x by 1
}
System.out.println( "Done" );
```

**Output:**

10 bottles of beer on the wall
9 bottles of beer on the wall
8 bottles of beer on the wall
... (and so on)
0 bottles of beer on the wall
Done

**do-while Loop Example**

**Task**: Input commands from JOptionPane until seeing "quit".

```java
String command;
do {
    command = JOptionPane.showInputDialog( "Enter a command" );
    // ... add statements to process the command
} while ( !command.equals( "quit" ) ); // exit when "quit" seen
```
Which loop should I use?

while or do-while?
- Use a while loop when there is some chance that the loop body might not be executed. A while loop is executed zero times if the condition is initially false. A do-while loop is always executed at least once.
- Use do-while loops when the condition for loop termination is based solely on things that occur within the loop body (not before).

Example of Loops and Conditionals

```java
//
/* Another intelligence test (don't give up until we get the right answer)
*/
import java.awt.*;
public class SimpleTest2 {
    public static void main(String[] args) {
        boolean isCorrect;
        do {
            String choice = JOptionPane.showInputDialog(
                "What is the world's greatest university? (hint: UMOP) ");
            if (choice.equals("UMOP") ) { // correct response
                isCorrect = true;
            } else { // incorrect response
                isCorrect = false;
                JOptionPane.showMessageDialog(null, "You blew it. Try again.");
            }
        } while (!isCorrect); // keep trying until correct
        JOptionPane.showMessageDialog(null, "Wise choice.");
        System.exit(0); // terminate program
    }
}
```