CSC 2014 Java Bootcamp

Lecture 2
Variables, Expressions, I/O & Control

VARIABLES & EXPRESSIONS

Variables
- A variable is a name for a location in memory
- A variable must be declared by specifying the variable's name and the type of information that it will hold

```
data type variable name
int total;
int count, temp, result;
```
- Multiple variables can be created in one declaration

Variable Initialization
- A variable can be given an initial value in the declaration

```
int sum = 0;
int base = 32, max = 149;
```
- When a variable is referenced in a program, its current value is used

Primitive Data
- There are eight primitive data types in Java
- Four of them represent integers:
  - byte, short, int, long
- Two of them represent floating point numbers:
  - float, double
- One of them represents characters:
  - char
- And one of them represents boolean values:
  - boolean

Numeric Primitive Data
- The difference between the various numeric primitive types is their size, and therefore the values they can store:

<table>
<thead>
<tr>
<th>Type</th>
<th>Storage</th>
<th>Min Value</th>
<th>Max Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>8 bits</td>
<td>-128</td>
<td>127</td>
</tr>
<tr>
<td>short</td>
<td>16 bits</td>
<td>-32,768</td>
<td>32,767</td>
</tr>
<tr>
<td>int</td>
<td>32 bits</td>
<td>-2,147,483,648</td>
<td>2,147,483,647</td>
</tr>
<tr>
<td>long</td>
<td>64 bits</td>
<td>&lt; -9 x 10^18</td>
<td>&gt; 9 x 10^18</td>
</tr>
<tr>
<td>float</td>
<td>32 bits</td>
<td>+/- 3.4 x 10^38 with 7 significant digits</td>
<td></td>
</tr>
<tr>
<td>double</td>
<td>64 bits</td>
<td>+/- 1.7 x 10^308 with 15 significant digits</td>
<td></td>
</tr>
</tbody>
</table>
Characters

- A char variable stores a single character
- Character literals are delimited by single quotes:
  'a'  'x'  '?'  '
'  ','
- Example declarations:
  ```java
  char topGrade = 'A';
  char terminator = ';', separator = ' ';
  ```
- Note the distinction between a primitive character variable, which holds only one character, and a String object, which can hold multiple characters.

Boolean

- A boolean value represents a true or false condition
- The reserved words true and false are the only valid values for a boolean type
  ```java
  boolean done = false;
  ```
- A boolean variable can also be used to represent any two states, such as a light bulb being on or off.

Constants

- A constant is an identifier that is similar to a variable except that it holds the same value during its entire existence
- As the name implies, it is constant, not variable
- The compiler will issue an error if you try to change the value of a constant
- In Java, we use the final modifier to declare a constant
  ```java
  final int MIN_HEIGHT = 69;
  ```

Assignment

- An assignment statement changes the value of a variable
- The assignment operator is the = sign
  ```java
  total = 55;
  ```
- The expression on the right is evaluated and the result is stored in the variable on the left
- The value that was in total is overwritten
- You can only assign a value to a variable that is consistent with the variable’s declared type

Expressions

- An expression is a combination of one or more operators and operands
- Arithmetic expressions compute numeric results and make use of the arithmetic operators:
  ```java
  Addition +
  Subtraction -
  Multiplication *
  Division /
  Remainder %
  ```
- If either or both operands used by an arithmetic operator are floating point, then the result is a floating point

Division and Remainder

- If both operands to the division operator (/) are integers, the result is an integer (the fractional part is discarded)
  ```java
  14 / 3     equals    4
  8 / 12     equals    0
  ```
- The remainder operator (%) returns the remainder after dividing the second operand into the first
  ```java
  14 % 3     equals    2
  8 % 12     equals    8
  ```
Operator Precedence

- Operators can be combined into complex expressions
  \[\text{result} = \text{total} + \text{count} / \text{max} - \text{offset}\]
- Operators have a well-defined precedence which determines the order in which they are evaluated
- Multiplication, division, and remainder are evaluated prior to addition, subtraction, and string concatenation
- Arithmetic operators with the same precedence are evaluated from left to right, but parentheses can be used to force the evaluation order

Operator Precedence

- What is the order of evaluation in the following expressions?
  \[a + b + c + d + e\]
  \[a + b + c - d / e\]
  \[a / (b + c) - d \% e\]
  \[a / (b * (c + (d - e)))\]

Increment and Decrement

- The increment and decrement operators use only one operand
- The increment operator (++) adds one to its operand
- The decrement operator (--) subtracts one from its operand
- The statement \(\text{count}++;\) is functionally equivalent to \(\text{count} = \text{count} + 1;\)

Increment and Decrement

- The increment and decrement operators can be applied in postfix form: \(\text{count}++\)
- or prefix form: \(++\text{count}\)
- When used as part of a larger expression, the two forms can have different effects
- Because of their subtleties, the increment and decrement operators should be used with care

Shortcut Assignment Operators

- Often we perform an operation on a variable, and then store the result back into that variable
- Java provides assignment operators to simplify that process
- For example, the statement \(\text{num} += \text{count};\) is equivalent to \(\text{num} = \text{num} + \text{count};\)

Shortcut Assignment Operators

- There are many assignment operators in Java, including the following:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Example</th>
<th>Equivalent To</th>
</tr>
</thead>
<tbody>
<tr>
<td>+=</td>
<td>x += y</td>
<td>x = x + y</td>
</tr>
<tr>
<td>-=</td>
<td>x -= y</td>
<td>x = x - y</td>
</tr>
<tr>
<td>*=</td>
<td>x *= y</td>
<td>x = x * y</td>
</tr>
<tr>
<td>/=</td>
<td>x /= y</td>
<td>x = x / y</td>
</tr>
<tr>
<td>%=</td>
<td>x %= y</td>
<td>x = x % y</td>
</tr>
</tbody>
</table>
**Shortcut Assignment Operators**
- The right hand side of an assignment operator can be a complex expression
- The entire right-hand expression is evaluated first, then the result is combined with the original variable
- Therefore
  \[ \text{result} /= (\text{total} - \text{MIN}) \% \text{num}; \]
  is equivalent to
  \[ \text{result} = \text{result} / ((\text{total} - \text{MIN}) \% \text{num}); \]

**Interactive Programs**
- Programs generally need input on which to operate
- The `Scanner` class provides convenient methods for reading input values of various types
- A `Scanner` object can be set up to read input from various sources, including the user typing values on the keyboard
- Keyboard input is represented by the `System.in` object

**Reading Input**
- The following line creates a `Scanner` object that reads from the keyboard:
  \[ \text{Scanner scan} = \text{new Scanner} (\text{System.in}); \]
- The `new` operator creates the `Scanner` object
- Once created, the `Scanner` object can be used to invoke various input methods, such as:
  \[ \text{answer} = \text{scan.nextLine()}; \]

**Input Tokens**
- Unless specified otherwise, *white space* is used to separate the elements (called *tokens*) of the input
- White space includes space characters, tabs, new line characters
- The `next` method of the `Scanner` class reads the next input token and returns it as a string
- Methods such as `nextInt` and `nextDouble` read data of particular types
Example: The High-Low Game

```java
import java.util.Scanner;
public class highlow
{
    public static void main(String[] args)
    {
        Scanner scanner = new Scanner(System.in);
        HandGenerator hand = new HandGenerator();
        hand.generate();
        int target = generator.nextInt(100);
        System.out.println(“What is your guess?”);
        int guess = scanner.nextInt();
        System.out.println(“Initial value out of range”);
        System.out.println(“Guess again”);
        while (guess != target)
        {
            System.out.println(“What is your guess?”);
            guess = scanner.nextInt();
            if (guess < target)
            {
                System.out.println(“Too low”);
            }
            else if (guess > target)
            {
                System.out.println(“Too high”);
            }
            else
            {
                System.out.println(“Congratulations!”);
                break;
            }
        }
    }
}
```

The printf Method

- The System.out.printf method can be used to print formatted output to the console window
- System.out is a PrintStream object

```java
double height = 24.56832;
System.out.printf(“The height is %.2f%n”, height);
The height is 24.57
```

The printf Method

- The printf method takes a format string followed by series of values that are “plugged” into the format string
- The format string may contain format specifiers (such as %.2f and %n)

```
System.out.printf(“The height is %.2f%n”, height);
```

- %.2f - print this floating-point value to two decimal places
- %n - new line (move to the next line of output)

The printf Method

- A format specifier starts with the % sign and ends with a conversion code

<table>
<thead>
<tr>
<th>Format Specifier</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>%d</td>
<td>decimal integer</td>
</tr>
<tr>
<td>%f</td>
<td>floating point</td>
</tr>
<tr>
<td>%e</td>
<td>scientific notation</td>
</tr>
<tr>
<td>%c</td>
<td>character</td>
</tr>
<tr>
<td>%b</td>
<td>boolean</td>
</tr>
<tr>
<td>%s</td>
<td>character string</td>
</tr>
<tr>
<td>%%</td>
<td>percent sign</td>
</tr>
<tr>
<td>%n</td>
<td>newline</td>
</tr>
</tbody>
</table>

The printf Method

- The printf method accepts any number of parameters
- The values must match up to the format specifiers in the format string

```java
int a = 32;
double b = 123.45;
System.out.printf(“The sum of %d and %.3f is %.3f%n”, a, b, a + b);
The sum of 32 and 123.450 is 155.450
```

CONTROL STRUCTURES
Flow of Control
- Unless specified otherwise, the order of statement execution through a method is linear: one statement after another in sequence.
- Some programming statements allow us to:
  - decide whether or not to execute a particular statement
  - execute a statement over and over, repetitively
- These decisions are based on boolean expressions (or conditions) that evaluate to true or false.
- The order of statement execution is called the flow of control.

Conditional Statements
- A conditional statement lets us choose which statement will be executed next.
- Therefore they are sometimes called selection statements.
- Conditional statements give us the power to make basic decisions.
- The Java conditional statements are the:
  - if statement
  - if-else statement
  - switch statement

The if Statement
- The if statement has the following syntax:

  ```java
  if (condition) statement;
  ```

  - if is a Java reserved word.
  - The condition must be a boolean expression. It must evaluate to either true or false.
  - If the condition is true, the statement is executed.
  - If it is false, the statement is skipped.

Logic of an if statement
- An example of an if statement:

  ```java
  if (sum > MAX) 
  delta = sum - MAX;
  System.out.println(“The sum is “ + sum);
  ```

  - First the condition is evaluated -- the value of sum is either greater than the value of MAX, or it is not.
  - If the condition is true, the assignment statement is executed -- if it isn’t, it is skipped.
  - Either way, the call to println is executed next.

Boolean Expressions
- A condition often uses one of Java’s equality operators or relational operators, which all return boolean results:

  ```plaintext
  == equal to
  != not equal to
  < less than
  > greater than
  <= less than or equal to
  >= greater than or equal to
  ```

- Note the difference between the equality operator (==) and the assignment operator (=).
The if-else Statement

- An else clause can be added to an if statement to make an if-else statement

```java
if ( condition )
    statement1;
else
    statement2;
```

- If the condition is true, statement1 is executed;
- If the condition is false, statement2 is executed

- One or the other will be executed, but not both
- See Wages.java (page 211)

Logic of an if-else statement

The switch Statement

- The switch statement provides another way to decide which statement to execute next

- The switch statement evaluates an expression, then attempts to match the result to one of several possible cases

- Each case contains a value and a list of statements

- The flow of control transfers to statement associated with the first case value that matches

- Often a break statement is used as the last statement in each case's statement list

- A break statement causes control to transfer to the end of the switch statement

- If a break statement is not used, the flow of control will continue into the next case

- Sometimes this may be appropriate, but often we want to execute only the statements associated with one case

An example of a switch statement:

```java
switch (option)
{
    case 'A':
        aCount++;
        break;
    case 'B':
        bCount++;
        break;
    case 'C':
        cCount++;
        break;
}
```
The switch Statement

- A switch statement can have an optional default case.
- The default case has no associated value and simply uses the reserved word default.
- If the default case is present, control will transfer to it if no other case value matches.
- If there is no default case, and no other value matches, control falls through to the statement after the switch.

The switch Statement

- The expression of a switch statement must result in an integral type, meaning an integer (byte, short, int, long) or a char.
- It cannot be a boolean value or a floating point value (float or double).
- The implicit boolean condition in a switch statement is equality.
- You cannot perform relational checks with a switch statement.

The while Statement

- A while statement has the following syntax:

  ```java
  while (condition)
  statement;
  ```

  - If the condition is true, the statement is executed.
  - Then the condition is evaluated again, and if it is still true, the statement is executed again.
  - The statement is executed repeatedly until the condition becomes false.

Logic of a while Loop

- An example of a while statement:

  ```java
  int count = 1;
  while (count <= 5)
  {
      System.out.println (count);
      count++;
  }
  ```

  - If the condition of a while loop is false initially, the statement is never executed.
  - Therefore, the body of a while loop will execute zero or more times.

The for Statement

- A for statement has the following syntax:

  ```java
  for (initialization ; condition ; increment )
  statement;
  ```

  - The initialization is executed once before the loop begins.
  - The statement is executed until the condition becomes false.
  - The increment portion is executed at the end of each iteration.
Logic of a for loop

- **Initialization**: The variable is declared and initialized.
- **Condition**: The condition is evaluated.
- **Statement**: The body of the loop is executed.
- **Increment**: The counter is incremented.

The for Statement

- An example of a for loop:
  ```java
  for (int count=1; count <= 5; count++)
      System.out.println (count);
  ```
- The initialization section can be used to declare a variable.
- Like a while loop, the condition of a for loop is tested prior to executing the loop body.
- Therefore, the body of a for loop will execute zero or more times.

Pseudocode

- a notation resembling
- a simplified programming language,
- used in program design.

Rules for Pseudocode

- Write only one statement per line
- Capitalize initial keyword
- Indent to show hierarchy
- End multiline structures
- Keep statements language independent

The Selection Structure

- IF amount < 100
  - InterestRate = .06
- ELSE
  - InterestRate = .10
- ENDIF

Pseudocode

- IF amount < 100
  - InterestRate = .06
- ELSE
  - InterestRate = .10
- ENDIF
WHILE / ENDWHILE

count = 0
WHILE count < 10
ADD 1 to count
WRITE count
ENDWHILE
WRITE "The End"

Start

count = 0
WHILE count < 10
ADD 1 to count
WRITE count
ENDWHILE
WRITE "The End"

Mainline
count = 0
WHILE count < 10
DO Process
ENDWHILE
WRITE "The End"

Process
ADD 1 to count
WRITE count

Stop

REPEAT / UNTIL

count = 0
REPEAT
ADD 1 to count
WRITE count
UNTIL count >= 10
WRITE "The End"

Mainline
count = 0
REPEAT
ADD 1 to count
UNTIL count >= 10
WRITE "The End"

Process
ADD 1 to count
WRITE count