CSC220 (CSI)
Computational Problem Solving

Control Structures

The College of New Jersey

Please turn off your cell phone!

Sequencing

- Refers to sequential execution of a program's statements

```
do this;
then do this;
and then do this;
etc.
```

- Unless specified otherwise, the order of statement execution through a method is linear: one after another
- The order of statement execution is called the flow of control

Make Decisions

- Some programming statements allow us to make decisions – conditional/selection statements.
- These decisions are based on boolean expressions (also called conditions) that evaluate to true or false
- The Java conditional statements are the:
  - if and if-else statement
  - switch statement

Boolean Expressions

op	meaning	a *op* b	true	false
==	equal to	a is equal to b	2 == 2	3 == 2
!=	not equal to	a is not equal to b	3 != 2	2 != 2
<	less than	a is less than b	2 < 13	2 < 2
<=	less than or equal to	a is less than or equal to b	2 <= 2	3 <= 2
>	greater than	a is greater than b	13 > 2	2 > 13
>=	greater than or equal to	a is greater than or equal to b	3 >= 2	2 >= 3

Sample Run

```
Enter your age: 47
You entered: 47
Age is a state of mind.
```

Another Sample Run

```
Enter your age: 12
You entered: 12
Youth is a wonderful thing. Enjoy.
Age is a state of mind.
```

```java
import java.util.Scanner;
public class Age
{
   //-----------------------------------------------------------------
   //  Reads the user's age and prints comments accordingly.
   //-----------------------------------------------------------------
   public static void main(String[] args)
   {
      final int MINOR = 21;
      Scanner scan = new Scanner(System.in);
      System.out.print("Enter your age: ");
      int age = scan.nextInt();
      System.out.println("You entered: " + age);

      if (age < MINOR)
         System.out.println("Youth is a wonderful thing. Enjoy.");

      System.out.println("Age is a state of mind.");
   }
}
```

Logical Operators

- and: a && b is true if both a and b are true, and false otherwise.
- or: a || b is true if either a or b is true (or both are true), and false otherwise
- not: !a is true if a is false, and false otherwise.

a	b	a && b	a || b
false	false	false	false
false	true	false	true
true	false	false	true
true	true	true	true

Boolean Expressions

- Specific expressions can be evaluated using truth tables

total < MAX	found	!found	total < MAX && !found
false	false	true	false
false	true	false	false
true	false	true	true
true	true	false	false

Short-Circuited Operators

- The processing of `&&` and `||` is "short-circuited"
- If the left operand is sufficient to determine the result, the right operand is not evaluated

```
if (count != 0 && total/count > MAX)
   System.out.println("Testing.");
```

- This type of processing should be used carefully

Selection: If Statement

```
if ( <condition> ) {
   do this
}
```

`if` is a Java reserved word

```
if ( <condition> ) {
   do this
}
else {
   do that
}
```

The *`condition`* must be a boolean expression. It must evaluate to either true or false.

```
if ( <condition> ) {
   do this
}
else if ( <condition> ) {
   do that
}
else if (…) {
…
}
else {
   whatever it is you wanna do
}
```

At most ONE block is selected and executed.

Quick Check

- What do the following statements do?

```
if (total != stock + warehouse)
   inventoryError = true;
```

Sets the boolean variable to true if the value of `total` is not equal to the sum of `stock` and `warehouse`

```
if (found || !done)
   System.out.println("Ok");
```

Prints "Ok" if `found` is true or `done` is false

Comparing Data

- Comparing floating point values for equality
- Comparing characters
- Comparing strings (alphabetical order)

Comparing Float Values

- You should rarely use the equality operator (==) when comparing two floating point values (`float` or `double`)
- Two floating point values are equal only if their underlying binary representations match exactly

```
if (Math.abs(f1 - f2) < TOLERANCE)
    System.out.println("Essentially equal");
```

- The tolerance could be set to any appropriate level, such as 0.000001

Comparing Characters

- Java character data is based on the Unicode character set
- Unicode establishes a particular numeric value for each character, and therefore an ordering
- Appendix C provides an overview of Unicode

Characters	Unicode Values
0 – 9	48 through 57
A – Z	65 through 90
a – z	97 through 122

Comparing Strings

- Remember that in Java a character string is an object
- The `equals` method can be called with strings to determine if two strings contain exactly the same characters in the same order
- The `equals` method returns a boolean result

```
if (name1.equals(name2))
    System.out.println("Same name");
```

Comparing Strings

- We cannot use == to compare strings
- The `String` class contains the `compareTo` method for determining if one string comes before another (lexicographic ordering)
- A call to `name1.compareTo(name2)`
  - returns zero if `name1` and `name2` are equal (contain the same characters)
  - returns a negative value if `name1` is less than `name2`
  - returns a positive value if `name1` is greater than `name2`

```
int result = name1.comareTo(name2);
if (result < 0)
   System.out.println(name1 + "comes first");
else
   if (result == 0)
      System.out.println("Same name");
   else
      System.out.println(name2 + "comes first");
```

switch Statement

```
if ( <condition> ) {
   do this
}
else if ( <condition> ) {
   do that
}
else if (…) {
…
}
else {
   whatever it is you wanna do
}
```

```
switch (expression){
        case value1 :
                //Statements
                break; //optional
        case value2 :
                //Statements
                break; //optional
        //You can have any number of
        //case statements.
        default : //Optional
                //Statements
}
```

If ***expression*** matches ***value2***, control jumps to here

The Conditional Operator

- The conditional operator evaluates to one of two expressions based on a boolean condition
- Its syntax is:

```
condition ? expression1 : expression2
```

- If the ***condition*** is true, ***expression1*** is evaluated; if it is false, ***expression2*** is evaluated
- The value of the entire conditional operator is the value of the selected expression

The Conditional Operator

- For example:

```
larger = ((num1 > num2) ? num1 : num2);
```

- Another example:

```
System.out.println("Your change is " + count +
    ((count == 1) ? "Dime" : "Dimes"));
```

Quick Check

Express the following logic in a succinct manner using the conditional operator.

```
if (val <= 10)
   System.out.println("It is not greater than 10.");
else
   System.out.println("It is greater than 10.");
```

```
System.out.println("It is" +
   ((val <= 10) ? " not" : "") +
   " greater than 10.");
```

Loops: Controlled Repetition

- **While Loop**

```
while (<condition>) {
    stuff to repeat
}
```

- **Do-While Loop**

```
do {
    stuff to repeat
} while (<condition>)
```

- **For Loop**

```
for (<init>; <condition>; <update>) {
    stuff to repeat
}
```

All of these repeat
the stuff in the block

The block
{…}
is called the Loop's Body

The while Statement

```
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

Infinite Loops

- An example of an infinite loop:

```
int count = 1;
while (count <= 25)
{
    System.out.println(count);
    count = count - 1;
}
```

- This loop will continue executing until interrupted (Control-C) or until an underflow error occurs

Quick Check

How many times will the string "Here" be printed?

```
count1 = 1;
while (count1 <= 10)
{
    count2 = 1;
    while (count2 < 20)
    {
        System.out.println("Here");
        count2++;
    }
    count1++;
}
```

10 * 19 = 190

The do Statement

- A *do statement* has the following syntax:

```
do
{
    statement-list;
}
while (condition);
```

- The **`statement-list`** is executed once initially, and then the **`condition`** is evaluated
- The statement is executed repeatedly until the condition becomes false
- The body of a `do` loop executes at least once

The for Statement

- A *for statement* has the following syntax:

The *initialization* is executed once before the loop begins

The *statement* is executed until the *condition* becomes false

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

The *increment* portion is executed at the end of each iteration

```
initialization;
while ( condition )
{
    statement;
    increment;
}
```

Loops: Critical Components

- **Loop initialization**

  Things to do to set up the repetition

- **Loop Termination Condition**

  When to terminate the loop

- **Loop Body**

  The stuff to be repeated

- **Loop update**

  For the next repetition/iteration