Java Programming Week 11
Iteration

Orange Coast College
Computer Science 170
Spring 2004

The Plan For Week 11
- Introducing Loops: loops and a loop-building strategy
- Counting Loops: fixed repetition with for and while
- Indefinite Loops: while, do-while, and sentinel loops
- Java’s Jumps: using break and continue with loops
- Applet Parameters: passing information to your applets
- Homework: add a colorful “Ticker” to your page
- Reading: Ch 8 (335-341 loops), Ch 12 (334-336)

What Is Iteration?
- A CS term that means repeating a set of actions
- Also called repetition or looping
- Statements used in iteration are called loops
- Loops are “flow-of-control” statements similar to selection
  - Like the if statement, a loop evaluates a condition
  - If the condition is true, a group of actions are performed
  - If the condition is false, the actions are skipped
Iteration and Selection

- Iteration is **not the same** as selection, however
- A loop returns to its test after performing its actions
- If the test is still true, then the actions are performed again
- A selection statement only performs its actions once
- You cannot replace a loop with a selection statement

Java and Iteration

- Java has three loops: **while**, **do-while**, and **for**
  - Each varies in way best used and place best employed
- One difference is **where** the test is made

  - Test at the bottom
  ```java
  { 
    // Loop Body
  }
  Loop Test
  ```

  - Test at the top
  ```java
  Loop Test 
  { 
    // Loop Body
  }
  ```

Loop Classification

- Where tested is **not the best** method of classification
- Doesn't provide any empirical guidance
- Better to ask how each loop is controlled
- Two basic methods of loop control
  - Simplest method relies on a counter
    - These are called **counting** or **counted** loops
    - "Repeat these actions 10 times"
  - More complex loops test the occurrence of a particular event, or combination of events
    - These are called **indefinite** or **indeterminate** loops
Indefinite Loops

- There are also three common kinds of indefinite loops
  - Data loops
    - Keep going until there is no more data to process
    - Used when reading files or sending Web pages
  - Sentinel loops
    - Keep going until you see a particular value
    - Used when searching or sorting
  - Limit loops
    - Process until the answer is "close enough"
    - Often used in scientific and graphics programming

How to Write Loops

- Loops are a common source of errors in code
- You must have a loop-building strategy
  - Gives you some place to start
  - Step-by-step approach helps avoid errors
- Strategy designed by Doug Cooper
  - Professor at Berkeley who wrote "Oh Pascal"
  - Will help you write correct loops first time

The Four Sides of a Loop

- Each loop has four “places” where actions occur
  - Precondition: actions that occur before the loop
  - Bounds: the loop test
  - Operation: the actions that the loop performs
  - Postcondition: actions taken after the loop is complete
Identifying the Bounds and Goal

Before we can write a loop, we have to be able to describe, understand, and separate the **bounds** and **goal** of loop

- The **bounds** are the actions that make the loop “work”
  - Think of them as the “loop mechanics”
  - These are the actions that start and stop the loop
  - “Continue working from 8:00 to 5:00”

- The **goal** is what we want the loop to accomplish
  - These are the actions that calculate, add, or count
  - “Complete each of the 1040A forms in your inbox”

A Bounds and Goal Example

Use this problem statement as an example:

**Count the characters in a String until a period is encountered. Assume the String contains a period.**

- What is the **goal** of this loop?
  - Count the characters that precede a period

- What is the **bounds** of the loop?
  - A period was encountered

- Can use same bounds, but have different goal
  - **Print characters in a String until a period is encountered**

Step 1: Determine the Bounds

- What’s the first thing you need to know when using a new appliance or software package?

  **How to turn it off !!!**

- Same thing is true for loops. Begin by asking:

  "**How do I make this loop quit?**"

  - Correctly stopping your loop is the single most important step in writing a correct loop
  - Write condition using pseudo-code
Step 1: Determining the Bounds

// Before the loop
While C is not a period
{
    // Inside the loop
}
// After the loop

Step 2: The Preconditions

- Next, ask yourself
  "How do I get into the loop?"
- Precondition statements usually involve:
  - Creating variables needed by the bounds
  - Initializing them to some reasonable value
  - In our example we have two variables
    - C which is a character and S which is a String
  - Must make sure that C holds a character from S

Step 2: The Preconditions

S is a string passed as an argument
C is a character
Assign the first character from S to C
While C is not a period
{
    // Inside the loop
}
Step 3: Advance the Loop

- Add statements that move you closer to the bounds
- Suppose we didn't add any statements to loop
  - If S began with a period, loop would not be entered and program would perform correctly
  - Otherwise, once the loop was entered, nothing would change the value of C, so the loop would repeat over and over again
  - This is called an **endless loop**
- Solution: store next character from S in C

---

Step 3: Advance the Loop

**S** is a string passed as an argument
**C** is a character
Assign the first character from S to C

While C is not a period
{
  Assign next character in S to C
}

---

Loop-Building Part I

- Ensure the *loop works* before **putting it to work**
- **Step 1**: Write the loop bounds
  - Check for unavoidable and impossible conditions
- **Step 2**: Write the loop preconditions
  - Add and initialize any variables used in the loop's conditional test
- **Step 3**: Advance the loop
  - Write statements in the loop body that move you toward the loop bounds
Loop Building Part II: The Goal

Step 4: Start with the goal preconditions
- Usually involves creating and initializing variables
  - Counters: used to count something
  - Accumulators: used to add up something
- In our program we need:
  - A counter to hold the number of characters
  - The counter should be initialized to 0

Step 5: Perform necessary operations in body
- Usually involves updating variables or making tests
- In our case, we need to update our counter each time we go through the loop

Step 6: Make postcondition tests
- Loop exits because bound is reached, not because the goal is necessarily accomplished
- In our case [counter] no test is necessary
- If averaging, for instance, would need to make sure that the loop had actually been entered

The Finished Loop Plan

Variable S is a String, passed as an argument
Variable C is a character
Variable counter is an integer, initialized to 0

Assign the first character in S to C
While C is not a period
{ Add 1 to Counter
  Assign the next character in S to C
}
Counter holds number of characters before period in S
Loop Syntax

- Now that you have a "plan", let's look at syntax
- You'll learn how to:
  - Write counted loops with the for statement
  - Write counted loops with the while statement
  - Write indefinite loops using the while statement
  - Write loops that use the do-while statement
  - How to use the break and continue statements
- Complete Exercise 11A

Counted Loops

- A counted loop is simplest kind of loop
  - The loop bounds is controlled by a counter
  - Use for "perform this action 10 times"
- Java's for loop specializes in this

```
for (int count = 0; count < 10; count++)
{
    System.out.println("count="+count);
}
```

The for Loop Syntax

- The for loop has both a header and body
- Header has 3 expressions: initialize, test, and update
The Initialization Expression
- The initialization expression
  - Executed once, before the loop is entered
  - Always executed, regardless of the test
  - Usually used to declare your counter variable

```java
for (int counter = 0; ... ; ... )
{
    // Can use counter here
}
// Cannot use counter here
```

The Test Expression
- Works just like the test in an if statement
  - If true, loop body is entered. If false, it is not.
  - Evaluated once for every iteration of the loop
  - Ends with a semicolon

```java
for (int counter = 0; counter < 10; ... )
{
    // Can use counter here
}
// Cannot use counter here
```

The Update Expression
- Executed every time the loop is entered
  - Executed after the body of the loop is finished
  - Always executed one fewer times than test
  - Usually used to update your counter

```java
for (int counter = 0; counter < 10; counter++)
{
    // Can use counter here
}
// Cannot use counter here
```
A Real Example

- SumOfCodes.java [run the applet]
  - The Goal
    - Add up the Unicode values of the characters in a String
    - Use the charAt() method to process each character
  - The Bounds
    - All the characters in the String have been processed
    - Can use the String.length() method to find how many
  - Variables needed
    - A String and a TextField to get the user input
    - A counter to hold the number of characters in String
    - An accumulator to hold the sum of the characters

Counting Down

- The for loop is a general purpose loop
  - Don't have to use a simple, incrementing counter
  - You don't have to count up:
    - for (int i = 10; i > 0; i--) { ... }
    - Works like Pascal's DOWNTO command
  - Example: use this to reverse the characters in a String
    - Examine the example ReverseString.java [run it]
    - Note that Strings are numbered starting with 0

Counting by Steps

- You don't have to count by one:
  - for (int i = 0; i < 50; i += 5) { ... }
    - Works like BASIC's STEP command
    - Not restricted to a constant step, like BASIC
  - for (int i = 2; i < 10000; i *= 2 )
    { System.out.println(i); }

Initialization Variations

- The initialization and update expressions can initialize and update more than one variable
- Just separate each expression with a comma
- Only place this is allowed
- Multiple initializations:

```
for (int i = 0, j=10; // notice, 1 int
     (i+j) < 100;
     i++, j++) {...}
```

Introducing the while Loop

- Counted loops with `for` are easy
- All of the pieces are in one place
  - The initializer for your counter
  - The test expression
  - The update expression
- The simpler, general-purpose `while` loop can also be used to create counted loops
  - It's just not quite as convenient as `for`

The while Loop Syntax

- Parts of a `while`
  - Keyword `while`, a boolean test, loop body

```java
while ( condition )
{
    statement;
    statement;
}
```
Counted while Loops

- To use the while statement to build a counted loop, you must do three things:
  1. Create a counter variable, and initialize it before the loop is encountered.
  2. Test the counter variable inside the while loop's boolean condition expression
  3. Update your counter at the end of the loop body. This takes the place of the for loop's update expression.

A Counted while Skeleton

Here is a skeleton you can use for building counted while loops:

```java
int counter = 0;
while (counter < 10 ) {
    // Loop body statements
    counter++; // Update counter at end of loop body
}
```

Counted while Loop Errors

- Several problems common to counted while loops

Problem 1: Using an "expired" counter
  - Define and initialize counter in for initializer
  - Looks empty if you forget initialization
  - Allows you to use the same variable name in each loop

- The while requires counter initialization before loop
  - Scope of counter is "rest of program"
  - Cannot create a new counter for each while loop

- You must remember to initialize your counter just before you enter the loop
Common while Loop Errors II

- **Problem 2**: Endless Loops
  - Not unique to `while` loops
  - Counted `while` loops are especially susceptible
    - Counter update is often far removed from test
    - Makes it easy to forget the update
  - Solution?
    - Use loop-building strategy that starts with bounds
    - Get mechanics working before attacking the goal
    - Use the `for` loop for counted loops

Common while Loop Errors III

- **Problem 3**: The phantom semicolon
  - If you put a semicolon after the test condition of a `while` loop, its body becomes the null statement
    - ```java
        int counter = 0;
        while (counter < 10); 
        { 
            // This is unreachable
            counter++; 
        }
    ```
  - Complete Exercise 11B

Indefinite Loops

- How many times will this loop execute?
  - ```java
      someChar = getRandomChar();
      while (someChar != 'Q') 
      { 
          output.setText("" + ++ count);
          someChar = getRandomChar();
      }
  ```
  - You can't tell. That's why it's called an **indefinite** loop
  - This is where the `while` loop really shines
An Example of an Indefinite Loop

- How Indefinite.java works [run it]
  - Character variable randomChar is assigned a value between 0 and 127, using the Math.random() method
  - Variable randomChar is compared to 'Q' in while test
    - If it is not equal to 'Q' then the loop body is entered
    - In the body, number of repetitions is incremented, displayed
    - A new value for randomChar is then generated

Sentinel Loops

- This kind of indefinite loop is called a Sentinel Loop
  - A specific character ['Q'] is searched for
  - Sentinel doesn't have to be a single value, but may include a range of values
  - Here's a problem that requires a sentinel range

  "Generate random integer numbers until a negative number is generated. Display the sum, count, and average of the numbers entered"

- Use your loop-building strategy to create a loop that meets these requirements

Sentinel Loops II

- Here are the questions you should ask
  - 1. What is the loop's bounds?
  - 2. What are the necessary preconditions?
  - 3. What actions are required to advance the loop?
  - 4. What is the loop's goal?
  - 5. What are the goal preconditions?
  - 6. What actions are necessary in the body?
  - 7. What postconditions are necessary to meet the goal?
- Answer these on a sheet of paper
Sentinel Loop III

1. What is the loop's bounds?
   - Answer: A negative number was generated
   - The Java loop condition should look like this:
     ```java
     while (num >= 0)
     ```

2. What are the necessary preconditions?
   - Look at the test condition: num must have a value
     ```java
     int num = getRandomNumber();
     ```

Sentinel Loop IV

3. What actions advance the loop?
   - Must change something in the test expression
     - Should generate another random number
   - At this point, the “mechanics” are finished
     - You can test the loop to see that it works

4. What is the loop's goal?
   - Display the `count`, `sum` and `average` of positive integers generated

Sentinel Loop V

5. What are the goal preconditions?
   - Must have variables for the count and sum
     - Sum should be a `double` because of overflow
     - `count` and `sum` must both be initialized
     ```java
     int count = 0; double sum = 0.0;
     ```

6. What actions are necessary to advance goal?
   - Add `num` to `sum`, incr `count`, before next num
     ```java
     count++; sum += num;
     ```
Sentinel Loop VI

7. What post-condition actions are required?
   - Must check for a `count` of zero.
     - Means no numbers were entered
     - Can't compute the average if there are no numbers
   - If `count` is > 0 then
     - Compute and display the sum and the average
   - Examine `PositiveAverage.java` [run the applet]

Primed and Inline Tests I

- So far, all our indefinite loops have been primed
  - Initialize the test condition before the loop
    - Usually this involves reading a value
    - This is called priming the loop
  - Test the value in the loop condition
  - Process the value in the body of the loop
  - Initialize the test condition at bottom of loop body
    - Preparing for the next repetition of the loop
    - Means "preparation" statements appear in two places

Primed and Inline Tests II

- Assignment operator allows concise inline test
  - Assign and test inside the loop condition
    ```java
    while ((num = getRandom()) >= 0)
    {
        // loop statements here
    }
    ```
  - Only requires a single "read"
  - Location of parentheses is critical
### Intentional Bounds
- Suppose you search a String for the letter "F"
  - A sentinel bounds is the obvious choice

```java
int i = 0;
while (s.charAt(i) != 'F')
    i++;
// i contains the position of the 'F'
```

- This is called the intentional bounds

### Necessary Bounds
- What happens if the letter 'F' is not in the String s?
  - Your program "crashes" when i becomes too big
- To fix this, you need to supply an additional bounds to use in case the value you are looking for is not found

```java
int i = 0;
while (s.charAt(i++) != 'F' && i < s.length())
    ; // do nothing
// if i == s.length() then 'F' not in String s
```

- This additional bounds is called a necessary bounds

### Test at the Bottom
- One more loop in addition to for and while
  - The do-while loop is an exit-condition loop
  - That means the test is located after the body

[Diagram of do-while loop]
When To Use Do-While?

- When you want to execute the body before making the test.
- Examples? Do you want to make another transaction?
- Complete Exercise 11C

Jump Statements

- Jumps are "low-level" flow of control statements.
  - In contrast to "high-level" statements like if and loops.
- A jump statement is sometimes called a goto.
  - Although goto is a reserved word in Java, it is not used.
- Java does have two "controlled" jump statements:
  - break: used to exit a switch or a loop (but not an if).
  - continue: used to prematurely "restart" a loop.
- Java also has "labeled" versions of each of these.
  - Used to exit or restart a particular nested loop.

The break Statement

- The break jumps out of a switch or loop (but not an if).
- In a loop, break jumps to the first statement following the loop body as shown here.
- Can make your code clearer when used to construct a "loop-and-a-half."
A "Loop-and-a-half"

- A loop-and-a-half is available in some languages like Ada
- In Java, use an if along with break to do the same thing

```
for (int i = 0; i < str.length(); i++) {
    char ch = str.charAt(i);
    if (ch == '.') break;
    ans += ch;
}
```

- Especially useful for "primed" loops with complex setup
- Example: BreakFor.java [run the applet]

The continue Statement

- The continue statement only works inside loops
- Instead of leaving the loop, it starts the next iteration
- In while and do-while continue jumps to the loop test expression
- In for, it jumps to the update expression

Using continue

- Suppose you want to convert 1,234,456 to an int
- You can’t use Integer.parseInt() because of the commas
- You can write a loop to convert the number manually
  - Use the String’s length as the loop bounds
  - Extract each character from the String
  - Use Character.isDigit() to see if it is a number
  - If not, then restart the loop using continue
  - If it is, then convert the digit into a binary number
  - Add it to the running total
- Example: ContinueFor.java [run the applet]
Nested Loops

- Loops can be nested, just like if statements
- We call the nested loop the inner loop
- We call the enclosing loop the outer loop

```java
for (int outer = 0; outer < 5; outer++)
{
    for (int inner = 0; inner < 10; inner++)
    {
        // inner body
    }
}
```

- Example: Times.java [run the applet]

Labeled break and continue

- `break` and `continue` only work on the "current" loop
- To exit an outer loop, Java has a labeled `break`
- Add a label to "name" the loop you're interested in
- The label must appear immediately before the loop
- A label is an identifier, followed by a colon

```java
outerLoop:
    for (int outer = 0; outer < 5; outer++) ...
```

- Use the label's name when using `break` or `continue`

Example: How Many Repetitions?

```java
int count = 0;
outerLoop:
    for (int outer = 1; outer < 1000; outer++)
    for (int inner = 1; inner < 1000; inner++)
    {
        count++;
        if (inner % 2 == 0) continue;
        if (inner % 201 == 0) break;
        if (outer % 25 == 0) continue outerLoop;
        if (outer % 51 == 0) break outerLoop;
    }
```

- Complete Exercise 11D
**Applets and Arguments I**

- How do you pass arguments to your applets?
- Use the HTML tag named `<param>`
- Placed between the `<applet>` and `</applet>` tags

```
<applet code=SomeApplet.class
       width=300 height=100>
   <param ...
   <param ...
   <param ...
</applet>
```

- The tag has two attributes, `name` and `value`

**Param Tag Attributes**

- How do you write the `<param>` attributes?
- For each argument create a name-value pair

```
<param name=msg value = "Hi Mom">
<param name=font-size value = 72>
```

- Use quotes around the value if it contains spaces
- Quotes are required in xhtml
- Attribute names are not case sensitive
- Attribute names do not have to follow identifier rules

**Retrieving Param Values**

- Inside your applet, retrieve values like this:

```
String msgString = getParameter("msg");
```

- Pass the HTML param name as the argument
- The case of the argument is ignored when searching
- If multiple param name matches, last one is used
- The `getParameter()` method returns a `String`
  - If the param name is found, `value` is returned
  - The value returned has leading and trailing spaces removed
Handling the null Value
- If `getParameter()` does not find a match it returns `null`.
- Your code **must be prepared** for this situation.
- Here's an example (inside the `init()` method):

```java
String message = getParameter("msg");
if (message == null)
    message = "No Message Passed";
```

Numeric Arguments
- Numeric arguments are sent as `Strings` too.
- You'll have to convert them before use.
- Here's an example that looks for name “font-size”:

```java
String fs = getParameter("font-size");
if (fs == null) fs = "72";
int fsize = Integer.parseInt(fs);
Font f = new Font("Serif", Font.BOLD, fsize);
setFont(f);
```

Finish Up
- Assignments due on May 4
- Quiz 11
- **Homework 10** [Guess'0'Matic]
  - Remember to test each branch
- **Homework 11** [ColorTicker] due May 11
  - Test your program with hard-coded values
  - Add applet parameters for font-size, speed, etc.
  - Create font, set background **after** retrieving parameters