Loops

Repeat after me …

Loops

• A loop is a control structure in which a statement or set of statements execute repeatedly
• How many times the statements repeat is determined by the value of a control variable, which is tested at the beginning of each loop iteration

Loop types

• Repetition statements control a block of code to be executed for a fixed number of times or until a certain condition is met.
• Count-controlled repetitions terminate the execution of the block after it is executed for a fixed number of times.
• Sentinel-controlled repetitions terminate the execution of the block after one of the designated values called a sentinel is encountered.
Count control vs. event control

- Count control uses automatic increment (or decrement) of control variable to update its value and stop loop
- Event control requires intervention (input) to change control variable’s value and stop loop

Java syntax for a while loop

```java
// initialize control variable - e.g. int x = 0;
while (control variable not equal to final value)
{
    // statements that repeat a process
    // statement or statements that update control variable
}

NOTE: Loop body can be a single statement, a null statement, or a block.
```

Example

```java
int x = 0;  // control variable initialization
while (x < 100)  // control variable test
{
    System.out.printf("x=%d\n", x);
    x ++;  // control variable updated
}
```
While Statement

while ( Expression )
{
    statement(s);
}

- Expression is test for *terminating condition*
- Loop *exit* occurs when test succeeds (tests false)
- Loop *entry* is moment flow of control reaches 1st statement in loop body
- One *iteration* means one pass through the loop
- Even though actual value being tested changes inside loop body, exit does not occur until next time value is tested

Count-controlled loop example

```java
public class countdown {
    public static void main(String [] args) {
        int x = 10;
        while (x > 0) {
            System.out.println("%4dn", x);
            x--;
        }
        System.out.println("Lift off!");
    }
}
```

<table>
<thead>
<tr>
<th>Value of <code>x</code></th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0 (loop ends)</td>
<td>Lift off!</td>
</tr>
</tbody>
</table>

Example

```java
int sum = 0, number = 1;

while ( sum <= 1000000 ) {
    sum = sum + number;
    number = number + 1;
}
```
Example

int product = 1, number = 1,
count = 20, lastNumber;

lastNumber = 2 * count - 1;

while (number <= lastNumber) {
    product = product * number;
    number = number + 2;
}

Example: finding sum and average

100 numeric values need to be added
together and averaged

Using a while loop:
• read the 100 values
• find their total
• find their average

Scanner kb = new Scanner(System.in);
int thisNum;
int total = 0;     // initialize sum
int count = 0;     // initialize loop control
while ( count < 100 ) // test expression
{
    System.out.print("Enter value: ");
    thisNum = kb.nextInt();  // read 1 value
    total = total + thisNum ; // add value to sum
    count++ ;         // update loop control
}
System.out.println("The sum of the values is: "+total);
System.out.println("The average is: "+
(double)total/count);
Event-controlled Loops: examples of events

- Sentinel value trigger
  keep processing data until a special value
  which is not a possible data value is entered
  to indicate that processing should stop

- End-of-input trigger
  keep processing data as long as there is more
  data to be read

- Flag value trigger
  keep processing data until the value of a flag
  changes in the loop body because of
  abnormal data

Examples of Kinds of Loops

<table>
<thead>
<tr>
<th>Count controlled loop</th>
<th>Read exactly 100 values.</th>
</tr>
</thead>
<tbody>
<tr>
<td>End-of-input controlled loop</td>
<td>Read all the values no matter how many are there; check with user for end of data.</td>
</tr>
</tbody>
</table>

Examples of Kinds of Loops

<table>
<thead>
<tr>
<th>Sentinel controlled loop</th>
<th>Read values until a special value (like -1) selected by you is read.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flag controlled loop</td>
<td>Read values until a value outside the expected range (say, 200 or more) is read.</td>
</tr>
</tbody>
</table>
A Sentinel-controlled Loop

- requires a “priming read”

- “priming read” means you read one set of data before the while

Sentinel-controlled loop

- Priming read occurs before loop body
- The value being processed in the loop body is the value that was read in the previous iteration
- The last action in the loop body is to read the next value
- When the sentinel value is read, the loop test will fail and the loop will stop

// Sentinel controlled loop
Scanner kb = new Scanner(System.in);
int total = 0;
int thisNum;

// priming read
System.out.print("Enter a positive number (-1 to stop): ");
thisNum = kb.nextInt();

while (thisNum != -1) { // while last value read is not sentinel
    total = total + thisNum;
    System.out.print("Enter a positive number (-1 to stop): ");
    thisNum = kb.nextInt();
}
System.out.println("Total sum is: "+total);
// End-of-input controlled loop
Scanner kb = new Scanner(System.in);
String reply = "y";
int thisNum, total = 0;
System.out.println("Enter a positive number: ");
thisNum = kb.nextInt();
while (reply.equals("y") || reply.equals("Y")) {
    total = total + thisNum;
    System.out.print("Enter y to continue, n to stop: ");
    reply = kb.next();
    if (!reply.equals("n") && !reply.equals("N")) {
        System.out.print("Enter a positive number: ");
        thisNum = kb.nextInt();
    }
}
System.out.println("Total of values entered is: "+total);

Loop applications

• We have already seen some common applications of loops, including:
  – reading and performing calculations (such as finding a sum) on a set of data values
  – processing a set of data to find a concluding value (finding an average, finding the greatest common divisor)
• Other applications include:
  – data validation
  – keeping track of occurrences of particular values in a data set

Example: Data Validation
Scanner kb = new Scanner (System.in);
int age;
System.out.print("Enter your age (between 0 and 130): ");
age = kb.nextInt();
while (age < 0 || age > 130) {
    System.out.println("An invalid age was entered. Please try again.");
    System.out.print("Enter your age (between 0 and 130): ");
age = kb.nextInt();
}
**Example: counting occurrences**

Scanner kb = new Scanner(System.in);
int num, // random number
    factor, // factor to find multiples of
    howMany, // number of random values to generate
    loopCnt = 0, // loop counter
    count = 0; // occurrence counter
Random rg = new Random();
System.out.print("How many tirs? ");
howMany = kb.nextInt();
System.out.print("Enter numeric factor to look for: ");
factor = kb.nextInt();
while (count < howMany) {
    num = Math.abs(rg.nextInt());
    if (num % factor == 0)
        count++;
}
System.out.println("There were "+ count + " multiples of " + factor + " in this set of " + howMany + " random numbers.");

**Watch Out for Pitfalls**

1. Watch out for the off-by-one error (OBOE).
2. Make sure the loop body contains a statement that will eventually cause the loop to terminate.
3. Make sure the loop repeats exactly the correct number of times.
4. If you want to execute the loop body N times, then initialize the counter to 0 and use the test condition counter < N or initialize the counter to 1 and use the test condition counter <= N.

**Example**

```java
int product = 0;

while (product < 500000) {
    product = product * 5;
}
```
Example

```java
int count = 1;

while ( count != 10 ) {
    count = count + 2;
}
```

Overflow

- An **overflow error** occurs when you attempt to assign a value larger than the maximum value the variable can hold.
- In Java, an overflow does not cause program termination:
  - With types **float** and **double**, a value that represents infinity is assigned to the variable
  - With type **int**, the value “wraps around” and becomes a negative value

Example

- The two loops below are almost identical; the only difference is that loop 1’s terminal value has one less digit than loop 2
- Loop 2 will terminate; loop 1 won’t

```java
// Loop 1:
float count = 0.0f;
while ( count != 1.0f ) {
    count = count + 0.3333333f;
} // seven 3s

// Loop 2:
float count = 0.0f;
while ( count != 1.0f ) {
    count = count + 0.3333333f;
} // eight 3s
```
Example

- Again, two almost identical loops produce different results
- In this case, both will terminate, but they will print different values

```java
int result = 0; double cnt = 1.0; int result = 0; double cnt = 0.0;
while (cnt <= 10.0) {
    cnt += 1.0;
    result++;
}
System.out.println(result); // prints 10
```

```java
while (cnt <= 1.0) {
    cnt += 0.1;
    result++;
}
System.out.println(result); // prints 11
```

Loop Design - considerations

- What process is repeated?
- How is process initialized & updated?
- What condition ends loop?
- How is condition initialized?
- How is condition updated?
- What is state of program upon loop exit?

Loop ending condition -- designing flow of control

- Can usually determine condition by looking at the problem statement
- Initialization:
  - assignment (e.g. for count control)
  - priming read (for event)
- Update
  - autoincrement vs. input
Designing loop process

• Decide what a single iteration will do
• May need to initialize variables before loop body and update their values within loop body
• Examples
  – accumulating a sum
  – keeping a running tally

Program state on loop exit

• All variables involved in the loop will have values at exit
• Need to initialize variables with care to avoid off-by-one errors

Loop design example

• Write a program that reads in a line of text and reports the number of characters and the number of capital letters read
• What type of loop control?
• What is stop condition?
• What variables are needed & how should they be initialized?