Class #14: Repetition with Loops

Software Design I (CS 120): D. Mathias, 25 Sept 19

Write a program that...

- Reads an integer value from the user and prints a message, including the value, to the screen.
  - That’s easy – we’ve done very similar things already.
- Reads 10 integer values from the user and prints their mean to the screen.
  - Hmm, that shouldn’t be too hard.
  - But it’s repetitive.
  - And boring.
- Reads 1000 integer values from the user and prints their mean to the screen.
  - That’s it, I’m changing my major.

Repetition with Loops

- Often, we want to repeat some operation in a program more than once
- Loops allow us to do this without needing to repeat the same code over and over ourselves
  - Precise starting and stopping conditions
  - Control exactly how many times we do something if needed
- When combined with conditional execution, looping allows us to create far more complex and interesting programs, behaving in a variety of sophisticated ways

General Types of Loops

- Loops can be broken down into two types:
  1. **Definite**: when we write these loops, we **know exactly** how many times we want to run them
     - Add up the integers from 1 to 1,000,000
     - Print out the first 100 prime numbers
  2. **Indefinite**: when we write these loops we **do not** always know exactly how many times they will run
     - Read integers from user until they enter -1
     - Divide input by 3 until it is less than 2
Looping with `while`

- The `while` loop can be used for either definite or indefinite cases (most common kind for indefinite)
- Simple syntax:

```java
while (condition) {
    instructions;
}
```

The `while` loop has similar structure, but more complex behavior:

```java
while (condition) {
    instructions;
} // rest of program
```

### A Definite Example

- Every loop has four parts:
  - **Initialization**: establish the state before looping
  - **Condition**: boolean to control when loop stops
  - **Main work**: code to be repeatedly executed
  - **Make progress**: code to move the loop toward termination by eventually changing the condition

```java
int sum = 0;
int count = 1;
while (count < 6) {
    sum = sum + count;
    count++;
}
```

Runs exactly 5 times, every time. (Why?)

### Comparing `if` and `while`

- The `if` conditional has a simple structure:

```java
if (condition) {
    instructions;
} // rest of program
```

(a) **boolean expression** is evaluated

(b) if false, skip the instructions and move on to rest of program

(c) if true, do the instructions, then move on to rest of program

- `while` loop has similar structure, but more complex behavior:

```java
while (condition) {
    instructions;
} // rest of program
```

(a) **boolean expression** is evaluated

(b) if false, skip the instructions and move on to rest of program

(c) if true, do the instructions, then repeat the process all over again, starting again at step (a)
An Indefinite Example

Every loop has four parts:

- **Initialization**: establish the state before looping
- **Condition**: boolean to control when loop stops
- **Main work**: code to be repeatedly executed
- **Update**: code to move the loop toward termination by eventually changing the condition

System.out.print("Enter a value: ");
Scanner scan = new Scanner( System.in );
it = scan.nextInt();
int i = 0;
while ( i > 1 ) {
    log++;
    i = i / 2;
} System.out.println( "log_2 = " + log );

Will run differently depending upon input given.
If input i > 1, runs log_2(i) times.
If input i <= 1, then never runs at all.

Tracing Loop Operations

<table>
<thead>
<tr>
<th>Iteration</th>
<th>count &lt; 6?</th>
<th>sum</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>initialize</td>
<td>N/A</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>true</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>true</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>true</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>true</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>true</td>
<td>15</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>false</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Controlling Repetition

- **Original loop:**
  sum == 1+2+3+4+5

- **Change it to generate:**
  sum == 1+2+3+...+25

- **sum == 3+4+5+...+25**

- **sum == 1+3+5+...+11**
How Many Times will a Loop Run?

```java
int sum = 0;
int count = 1;
while ( count <= 100 ) {
    sum = sum + count;
    count = count + 2;
}
Runs 50 times:
1, 3, 5, ..., 99
```

```java
int sum = 0;
int count = 99;
while ( count != 0 ) {
    sum = sum + count;
    count = count - 2;
}
Runs Many, Many Times!
```

```
int sum = 0;
int count = 99;
while ( count > 0 ) {
    sum = sum + count;
    count = count - 2;
}
Runs 50 times:
99, 97, 95, ..., 1
```

```
int sum = 0;
int count = 0;
while ( count > 0 ) {
    sum = sum + count;
    count = count - 2;
}
Never runs at all.
```

Using Loops

- Sam earns $100 per day with a daily raise of $100.
- Sue earns $0.01 per day with a salary that doubles every day.

**How many days pass before Sue’s total income actually exceeds Sam’s?**

```java
double sam = 100.0;
double sue = 0.01;
// day counter
int count = 0;
// loop until sue earns more
while ( sue < sam ) {
    // count one day at time
    count++;
    // increase salaries
    sam = sam + 100.0;
    sue = sue * 2;
}
// print out final results
System.out.println( … );
```

Note: this code sample actually calculates the day on which Sue’s daily pay exceeds Sam’s. We need to keep track of running total as well (see demo code)