Introduction to Programming
with Java, for Beginners

Algorithm Thinking
Debugging
More on Loops

Recap While vs. For

<table>
<thead>
<tr>
<th>Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>int x = 1;</td>
<td>An example of a while loop that has this pattern</td>
</tr>
<tr>
<td>while (x &lt;= 10){</td>
<td></td>
</tr>
<tr>
<td>System.out.println(x);</td>
<td></td>
</tr>
<tr>
<td>x = x + 1;</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
<tr>
<td>for (int x = 1; x &lt;= 10; x = x + 1){</td>
<td>A for loop that does the same thing</td>
</tr>
<tr>
<td>System.out.println(x);</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
</tbody>
</table>

Note: In a for loop a “missing” end-test or re-initialization condition evaluates to true (infinite loop)

Common semantic (runtime) errors w/ loops

- No re-initialization condition that will eventually halt the loop
  - For preferred over while due the nature of syntax

- Condition starting out to be false initially

- Wrong initialization value or end-test condition
  - Start condition: x = 0 but loops tests x >= 0
  - x = 1 and we want the loop to go 10 times
    - x< 10 , will execute the loop 9 times
    - So be careful of < vs. <=

Formulating a Solution

- First Think *Algorithmically*
  - Well defined step by step procedure
  - Use *psuedocode* to write out your steps
    - English like code
    - It allows the designer to focus on the logic of the algorithm without being distracted by details of language syntax

- Then *Translate* the solution into programming language
  - Put together the components we have so far
    - declarations, assignments, control structures
Example: Fibonacci sequences

- A Fibonacci sequence is an infinite list of integers
- The first two numbers are given
  - Usually (but not necessarily) these are 1 and 1
- Each subsequent number is the sum of the two preceding numbers:
  1  1  2  3  5  8  13  21  34  55  89  144 …
- Let’s write a program to compute sequence when we get to a number that’s 1000 or bigger

Starting the Fibonacci sequence

- We need to initialize two numbers in the sequence
  - Set first to 1
  - Set second to 1
- We need to print these out:
  - Print first and second
- We need to compute and print the next number:
  - Set next to sum of first & second;
  - print next

Taking the next step

- Now what?
  - Need to add second and next
    - set nextnext to sum of second & next
  - What if the sequence is to long
    - I do want to make 100s of storage to hold each item
- The sequence so far is: first second next
- Do I see a pattern emerging?

Preparing to make many steps

- We need to make these moves:
  - set first to second
  - set second to next

- We can do it like this:
  - Set first to second
  - Set second to next
- We can put these statements in a loop and do them as many times as we please
**Psuedocode Rules**

- Can use words such as while, if else-if
  - E.g. for 1 to n
- Do not specify data declarations or types
- Use Words that specify an action such as set, reset, increment, compute, calculate, add, sum, multiply, print, getinput
- Use indentation for block of code i.e. {}
Syntax Error

- Use the Dr Java tool to your advantage
  - Keywords turn blue
  - Comments turn green
  - {} matching

- Reading compiler Errors
  - Turn on line numbers (In DrJava (Edit Preferences -> Display)
  - Learn common syntax errors
    - Missing Semicolon at end of a statement (line no. indicated)
    - For loop as three statements (each ending in semi-colon)
    - Using variable that are not assigned (sometimes also error when not initialized)
    - Wrong assignment of types

Reading compiler Errors

- Turn on line numbers (In DrJava (Edit Preferences -> Display)
- Learn common syntax errors
  - Missing Semicolon at end of a statement (line no. indicated)
  - For loop as three statements (each ending in semi-colon)
  - Using variable that are not assigned (sometimes also error when not initialized)
  - Wrong assignment of types

Debugging with System.out.println

//sum positive odd numbers upto n
// assume n = 3 then should expect 4 as outcome
int sum = 0;
for (int i = 1; i <= n; i = i + 1){
    sum = sum + i;
    //Debugging print statement
    System.out.println(i + ":" + sum);
}
System.out.println(sum);

Result of print
1 :1
2 :3
3 :6
6

Remember to comment out print statement when you are done testing

Scope of variables declared in loop

- **Scope** means the area of code in which an entity is known (or alive)

- For now we are concerned with variable(s)
  - declared inside a loop are accessible from point of declaration to end of closest enclosing loop
  - declared inside main are accessible with from the point of declaration to end of main i.e. closing curly brace }

Example

```java
public static void main(String [] args){
    int limit = 1000;
    int first = 1;
    int second = 1;
    while (first < limit) {
        System.out.println(first + " ");
        int next = first + second;
        first = second;
        second = next;
    }
    System.out.println( );
}
```
Break and Continue Statements

- `break` and `continue` are Java statements
- Are also “flow control” statements
- A break “breaks you out” of the closest enclosing loop
- A continue is a shortcut to the next iteration of the loop
- A loop may have
  - Zero or more break statements
  - Zero or more continue statements

while-loop with break, continue

```java
while (condition1) {
    if (condition2)
        continue; // go up and re-evaluate condition1
    if (condition3)
        break; // exit the loop
    . . .
    . . .
// after a break statement, execution resumes here
```

for-loop with break, continue

```java
for (expr1; condition1; expr2) {
    if (condition2)
        continue; // evaluate expr2, then condition1
    if (condition3)
        break; // exit the loop
    . . .
    . . .
// after a break statement, execution resumes here
```

Nested Loops

- A nested loop is a loop within a loop, an inner loop within the body of an outer one
- Just like nested if-statements
- Usually good for working problems that have rows and columns
**Example: Multiplication Table**

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>6</td>
<td>9</td>
<td>12</td>
<td>15</td>
<td>18</td>
<td>21</td>
<td>24</td>
<td>27</td>
<td>30</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>8</td>
<td>12</td>
<td>16</td>
<td>20</td>
<td>24</td>
<td>28</td>
<td>32</td>
<td>36</td>
<td>40</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>10</td>
<td>15</td>
<td>20</td>
<td>25</td>
<td>30</td>
<td>35</td>
<td>40</td>
<td>45</td>
<td>50</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>12</td>
<td>18</td>
<td>24</td>
<td>30</td>
<td>36</td>
<td>42</td>
<td>48</td>
<td>54</td>
<td>60</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>14</td>
<td>21</td>
<td>28</td>
<td>35</td>
<td>42</td>
<td>49</td>
<td>56</td>
<td>63</td>
<td>70</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>16</td>
<td>24</td>
<td>32</td>
<td>40</td>
<td>48</td>
<td>56</td>
<td>64</td>
<td>72</td>
<td>80</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>18</td>
<td>27</td>
<td>36</td>
<td>45</td>
<td>54</td>
<td>63</td>
<td>72</td>
<td>81</td>
<td>90</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>20</td>
<td>30</td>
<td>40</td>
<td>50</td>
<td>60</td>
<td>70</td>
<td>80</td>
<td>90</td>
<td>100</td>
</tr>
</tbody>
</table>

**Example: Multiplication Table**

```java
int product = 0;
for (int i = 1; i <= 10; i++) {
    for (int j = 1; j <= 10; j++) {  
        System.out.print("t" + i * j);
    }
    System.out.println();
}
```

Note: \t – tab character, allows for aligned spacing

---

**for loop variable scope**

- You can declare variable in the `for` statement
- The scope of the variable is the entire `for` loop

```java
for (int x = 1; x <= 10; x = x + 1){
    System.out.println(x);
}
```

```java
for (int x = 1; x <= 10; x = x + 1)  
    System.out.println(x);
```

---

**Another loop: Do-While**

```java
do {
    statement(s)  
} while (condition);  
```

- Do the statement/block at least once
- Evaluate the `condition`. If it is
  - true: re-execute `statement(s)`; repeat step 2
  - false: we’re done with the loop
- int x = 0;
- do {
    x = x + 1;
} while (x < 3);

```java
```

---