CITS1001 week 5
Repetition

Arran Stewart

March 27, 2018
Outline

- topics:
  - repetition
  - for loops
  - while loops
- Reading: Chapter 4 (section 4.10 to 4.16) of Objects First with Java - A Practical Introduction using BlueJ, © David J. Barnes, Michael Kölling
Review

- We often want to repeat some actions over and over
  - e.g. “do this action for each student in the university”
    - typically we’ll handle this with a “for-each” loop
  - e.g. “do this action seventeen times”
    - This is typically done using a for loop
  - e.g. “do this action until this condition is true”
    - The third paradigm is done using a while loop
for loops
Pseudocode for for loop

for (initialization; condition; post-body action) {
  statements to be repeated
}

General form of the for loop
The header of the loop is the information in the round brackets

for (<initialization>; <boolean-expression>; <post-body update>)
{
<statement-1>
<statement-2>
…
<statement-n>
}

The body of the loop is the collection of statements in the curly brackets
The initialization part consists of any Java statement.

- It is performed once only, when execution first reaches the for loop.
- It is normally used to initialize a counter variable (also known as “the index variable”)

Header – the initialization part
The boolean expression controls whether or not the body of the loop is executed.

The expression is evaluated immediately after initialization, and at the start of every subsequent iteration.

If its value is `true`, then the statements in the body of the loop are executed;
if its value is `false`, then the loop has finished and the statements in the body are NOT executed.

When the loop finishes, execution continues at the first statement after the `for` loop.
The post-body update is a Java statement that is executed once each time the body of the for loop is executed. It is executed immediately after the last statement of the body has been executed. It is usually used to update the counter variable.
for loop flowchart

Is the boolean expression true?

- Before loop
  - Initialization
  - Is the boolean expression true?
    - No
    - After loop
    - Yes
      - Body
      - Post-body update
The for loop idiom

- for loops are quite general, but one common use is when we want to do something a specific number of times.

```java
for (int i=0; i<5; i=i+1) {
    System.out.println(i);
}
Output:
0
1
2
3
4
```
The for loop idiom

- for loops are quite general, but one common use is when we want to do something a specific number of times.

- A typical idiom for doing that is code like the following:

```java
for (int i=0; i<5; i=i+1) {
    System.out.println(i);
}
```
The for loop idiom

- for loops are quite general, but one common use is when we want to do something a specific number of times.

- A typical idiom for doing that is code like the following:

```java
for (int i=0; i<5; i=i+1) {
    System.out.println(i);
}
```

- Output:

  0
  1
  2
  3
  4
The for loop idiom – steps

How did this work?

First iteration
- Initialization creates $i$ and sets it to 0
- Check if $i < 5$, yes
- Print out $i$ – causes 0 to appear on terminal window
- Update $i$ from 0 to 1
- Check if $i < 5$, yes
- Print out $i$ – causes 1 to appear on terminal window
- Update $i$ from 1 to 2
- Check if $i < 5$, yes
- Print out $i$ – causes 2 to appear on terminal window
- Update $i$ from 2 to 3
- Check if $i < 5$, yes
- Print out $i$ – causes 3 to appear on terminal window
- Update $i$ from 3 to 4
- Check if $i < 5$, yes
- Print out $i$ – causes 4 to appear on terminal window
- Update $i$ from 4 to 5
- Check if $i < 5$, NO

Second iteration
The increment operator

- Something you may see in existing Java code
- The post-body update often consists of just:
  
i = i+1;

- Because it’s so often used, there is a short-hand notation for this operation –
  
  The statement i=i+1 may be replaced simply by i++ (often pronounced “increment i”)

```
for (int i=0; i<5; i++) {
    System.out.println(i);
}
```

**NB:** Use either i=i+1 or i++, but don’t try and mix the two
If the body consists of only one statement, then you can leave out the braces . . .
However, it is better style to always include them
Serious security bugs have been caused by programmers omitting them
for (int i=0; i<5; i++) {
    System.out.println(i);
}

is the same as

for (int i=0; i<5; i++)
    System.out.println(i);
What output do we expect to get from the following code?

```java
for (int i=0; i<5; i++); // Syntax error
{
    System.out.println(i*i);
}
```
Writing for loops

What output do we expect to get from the following code?

```java
for (int i=0; i<5; i++);
{
    System.out.println(i*i);
}
```

Perhaps ...?

0
1
4
9
16
What output do we expect to get from the following code?

```java
for (int i=0; i<5; i++) {
    System.out.println(i*i);
}
```

Perhaps ...?

0
1
4
9
16

In fact, the output is just

100
A common mistake when writing for loops is accidentally including a surplus semicolon.

(this mistake can be very hard to track down)

The problem with the previous code was a problem with the loop body

We might have thought the body was

```
System.out.println(i*i);
```

but in fact it was

```
;
```
A common mistake

int i;
for (i=0; i<10; i++) ;
{
    System.out.println(i*i);
}

int i;
for (i=0; i<10; i++)
{
    System.out.println(i*i);
}

The first loop has an empty body (just a single semicolon!), while the second shows the “desired” body.
Another use for for loops – making tables

- Another common use of for loops is to produce tables
- Suppose you are asked to produce a temperature conversion table listing the Fahrenheit equivalents of Celsius temperatures from 0–100°C, going up in increments of 5°C

<table>
<thead>
<tr>
<th>Celsius</th>
<th>Fahrenheit</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>32</td>
</tr>
<tr>
<td>5</td>
<td>41</td>
</tr>
<tr>
<td>10</td>
<td>50</td>
</tr>
<tr>
<td>15</td>
<td>59</td>
</tr>
<tr>
<td>20</td>
<td>68</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
Making tables (cont'd)

A for loop is the solution:

```java
int celsius;
int fahrenheit;
for (celsius=0; celsius <= 100; celsius = celsius + 5) {
    fahrenheit = 32 + celsius*9/5;
    System.out.print(celsius);
    System.out.print(" ");
    System.out.println(fahrenheit);
}
```

NB: we use `System.out.print()` instead of `System.out.println()` when we want to print something without starting a new line afterward
A numerical example – approximating $\pi$

- A formula for the value of $\pi$ is:

$$\pi = 4 \times \left( \frac{1}{1} - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \ldots \right)$$

- Suppose we wish to approximate $\pi$ using this formula; here are two approaches we might take:

  - Approximate $\pi$ using a given number of terms from the above formula (say, the first 10 terms)
  - Approximate $\pi$ to within a given accuracy (e.g., to within 0.01)

- The first way is best done using a for loop
Approximating $\pi$ using a given number of terms

**Formula:**

$$\pi = 4 \times \left( \frac{1}{1} - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \ldots \right)$$

**Code:**

```java
public double pi(int n) {
    double approx_pi = 0;
    double mult    = 4; // mult. each term
                        // by +4 or -4
    for (int i=0; i<n; i++) {
        approx_pi = approx_pi + mult/(2*i+1); // "flip" multiplier
        mult    = -mult;
    }
    return approx_pi;
}
```
Approximating $\pi$ – variable values at top of loop

<table>
<thead>
<tr>
<th>i</th>
<th>mult</th>
<th>2*i+1</th>
<th>approx</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4.0</td>
<td>1</td>
<td>0.00</td>
</tr>
<tr>
<td>1</td>
<td>-4.0</td>
<td>3</td>
<td>4.00</td>
</tr>
<tr>
<td>2</td>
<td>4.0</td>
<td>5</td>
<td>2.67</td>
</tr>
<tr>
<td>3</td>
<td>-4.0</td>
<td>7</td>
<td>3.47</td>
</tr>
<tr>
<td>4</td>
<td>4.0</td>
<td>9</td>
<td>2.90</td>
</tr>
<tr>
<td>5</td>
<td>-4.0</td>
<td>11</td>
<td>3.34</td>
</tr>
<tr>
<td>6</td>
<td>4.0</td>
<td>13</td>
<td>2.98</td>
</tr>
</tbody>
</table>

Loop stops when $i$ reaches the requested value
for loops with bigger steps

// Print multiples of 3 up to 40
for(int num = 3; num < 40; num = num + 3) {
    System.out.println(num);
}

Output:
3
6
9
12
15
18
21
24
27
30
33
36
39
Review of for loops

Use them when:

- the number of repetitions is known in advance
- an index variable is required
- there is a regular step-size

But note:

- “For-each” loops have less scope for error than for loops
- So: use a for-each loop unless you need access to indices or step-size
while loops
Use of *while* loops

- The repetition pattern embodied in *while* loops is: "do this action until this condition is true"
  - We don’t know in advance how many iterations there will be
- In Java, this is done with a *while* loop
- We use a boolean condition to decide whether or not to keep going
while loop pseudocode

General form of a while loop

whilekeyword

while(loop condition) {
    loop body
}

boolean test

Statements to be repeated

while we wish to continue, do the things in the loop body

Pseudo-code expression of the actions of a while loop

while we wish to continue, do the things in the loop body
while loop flowchart

Before loop

Is the boolean expression true?

Yes

Body

No

After loop
while (the keys are missing) {
    look in the next place
}

or equivalently:

while (not (the keys have been found)) {
    look in the next place
}
Looking for your keys – Java code

```java
boolean stillSearching = true;
Location place = firstPlace;
while( stillSearching) {
    if( /* the keys are in place... */ ) {
        stillSearching = false;
    } else {
        place = next(place);
    }
}```
π to within a given accuracy

Another strategy for approximating π:

- approximate it to within a given accuracy (say, to within 0.001)
public double pi(double accuracy) {
    double approx = 0;
    double mult = 4;
    double denom = 1;
    while (Math.abs(mult/denom) > accuracy) { // use Math class
        approx = approx + mult/denom;
        mult = -mult;
        denom = denom + 2;
    }
    return approx;
}
Approximating $\pi$ – values of variables at top of loop

<table>
<thead>
<tr>
<th>Step</th>
<th>denom</th>
<th>mult/denom</th>
<th>approx</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.0</td>
<td>4.00</td>
<td>0.00</td>
</tr>
<tr>
<td>1</td>
<td>3.0</td>
<td>-1.33</td>
<td>4.00</td>
</tr>
<tr>
<td>2</td>
<td>5.0</td>
<td>0.80</td>
<td>2.67</td>
</tr>
<tr>
<td>3</td>
<td>7.0</td>
<td>-0.57</td>
<td>3.47</td>
</tr>
<tr>
<td>4</td>
<td>9.0</td>
<td>0.44</td>
<td>2.90</td>
</tr>
<tr>
<td>5</td>
<td>11.0</td>
<td>-0.36</td>
<td>3.34</td>
</tr>
<tr>
<td>6</td>
<td>13.0</td>
<td>0.31</td>
<td>2.98</td>
</tr>
</tbody>
</table>

Loop stops when the next term is smaller than accuracy
Features to note

- We have effectively declared an index variable
  - (In this case, `denom`)
- The index variable must be incremented explicitly
  - It’s not updated automatically in the header, as with the `for` loop
- The condition must be expressed correctly
- We must know that the loop will end
Exercise – searching a collection with \texttt{while}

- Recall the "book journal" class from previous lectures

- Let us write another taken on a search method: one which searches for the first title containing a search string, and returns the index of that item (or -1 if no such item is found)
Searching a collection (cont’d)

pseudocode:

index = 0
stillSearching = true

while stillSearching && index < bookTitles.size():
    bookTitle = bookTitles.get(index)
    if bookTitle contains searchString:
        stillSearching = false // stop searching.
    else:
        index++

if stillSearching:
    return -1 // We didn't find it.
else:
    return index // Return where it was found.
Searching a collection (cont’d)

Questions

- The loop’s condition repeatedly asks the bookTitles collection how many titles it is storing.
Questions

- The loop’s condition repeatedly asks the bookTitles collection how many titles it is storing.
  - Does the value returned by `size()` vary from one check to the next?
The loop’s condition repeatedly asks the bookTitles collection how many titles it is storing.

- Does the value returned by `size()` vary from one check to the next?
- If not – rewrite the method so the number of titles is stored once in a variable, before the loop starts. Then use that variable, rather than calling `size()`.
Questions

- The loop’s condition repeatedly asks the bookTitles collection how many titles it is storing.
  - Does the value returned by `size()` vary from one check to the next?
  - If not – rewrite the method so the number of titles is stored *once* in a variable, before the loop starts. Then use that variable, rather than calling `size()`.

- Can `findFirst` be implemented using the “search and return” pattern we’ve seen previously? How do the two implementations compare?
Questions (cont’d)

- Does the code in `findFirst` work if the collection is empty?
What are some of the advantages and drawbacks of using a “for-each” loop, as opposed to a while loop?
for-each versus while (cont’d)

- **for-each:**
  - Easier to write
  - Safer: it is guaranteed to stop

- **while:**
  - Easy to stop processing part-way through a collection
  - Doesn’t even have to be used with a collection
  - Take care: could be an infinite loop
  - Handy when you don’t know how many times a loop will be repeated
Exercises

- Write a while loop that prints out multiples of 5 between 10 and 95
- Write a while loop to add up the values 1 to 10 and print the sum once the loop has finished