CSE 1710

Lecture 17

Iteration, Friendly Validation

Today

- Basic Iteration
- Friendly Validation

Let's talk about two forms of iteration…

- one form: built upon a boolean condition
- another form: built around a *collection*

The "Collection" Form of Iteration

- a **collection** is simply a bunch of elements, possibly in a particular order, but not necessarily
- the **elements** must have a type (e.g., `int`, `Pixel`, etc)
- a **set** is a collection in which duplicates are not permitted
- a **list** is a collection in which the elements are ordered
- an **array** is a specific kind of list

collection, set, list	:	abstractions, not specific to Java
array	:	a Java programming element

The “Collection” Form of Iteration

```
for ( Type-of-Element  e : Identifier-of-Collection ) {
     // here is the body of the loop…
     }
}
```

FOR EXAMPLE:

```
Pixel[] thePixels = myPict.getPixels();
```
// here we obtain an array

The “Collection” Form of Iteration

```
Pixel[] thePixels = myPict.getPixels();

for (Pixel p : thePixels) {
     // here is the body of the loop…
}
```


The “Collection” Form of Iteration

…various in-class exercises…

The “Condition” Form of Iteration

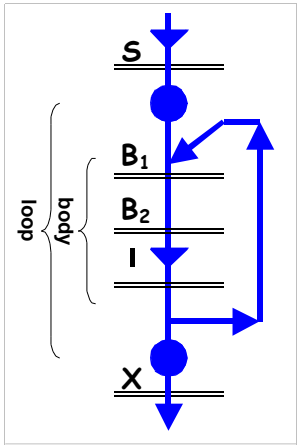
```
for (; boolean expression ;) {
     // here is the body of the loop…
}
```

The "Condition" Form of Iteration

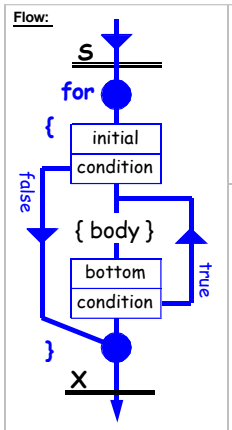
```
for ( initial ; boolean expression ; bottom ) {
      // here is the body of the loop…
}
```


5.2.1 Flow of Control

Iteration



5.2.2 The `for` statement



Syntax:

```
Statement -S
 for (initial; condition; b ottom)
 {
   body;
 }
Statement -X
```

Algorithm:

1. Start the `for` scope
2. Execute `initial`
3. If `condition` is `false` go to `9`
4. Start the body scope `{`
5. Execute the body
6. End the body scope `}`
7. Execute `bottom`
8. If `condition` is `true` go to `4`
9. End the `for` scope

Example

```
final int MAX = 10;
final double SQUARE_ROOT = 0.5;
for (int i = 0; i < MAX; i = i + 1)
{
    double sqrt = Math.pow(i, SQUARE_ROOT);
    output.print(i);
    output.print("\t"); // tab
    output.println(sqrt);
}
```

for (initial; condition; bottom)

```
for (int i = 0; i < MAX; i = i + 1)
{
    ...
}
```

```
int i;
for (; i < MAX; i = i + 1)
{
    ...
}
```


for (initial; condition; bottom)

- Can it be omitted?
- Can it be set to the literal `true`?
- What if it were false at the beginning?
- Is it monitored throughout the body?

for (initial; condition; bottom)

- Can it be any statement?
- Will the loop be infinite if it is omitted?

Example

Write a fragment to output the exponents of all powers of 2 that are smaller than a million.

Correct output:

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
```

Example

Write a fragment to output the exponents of all powers of 2 that are smaller than a million.

```
final int MILLION = 1000000;
for (int expo = 0; Math.pow(2, expo) < MILLION; expo++)
{
   output.print(expo);
   output.print(" ");
}
output.println();
```

As a second example, rewrite the fragment so it only outputs the exponent of the greatest power of 2 that is smaller than a million.

Example

Rewrite the fragment so that it only outputs the exponent of the greatest power of 2 that is smaller than a million.

```
int expo = 0;
for (; Math.pow(2, expo) < MILLION; expo++)
{
}
output.println(expo - 1);
```

```
int expo = 0;
for (; Math.pow(2, expo) < MILLION; expo++);
output.println(expo - 1);
```


5.2.3 Building the Loop

- **Sentinel-based example**
  Write a program that reads integers with a -1 sentinel and outputs their arithmetic mean.
- **Number statistics examples**
  Read numbers and determine their largest, smallest, second-largest, …

Sentinel-Based Looping

Write a prog that reads integers with a -1 sentinel and outputs their arithmetic mean.

Pseudo-code:

```
for (?; not sentinel; ?)
{
    process the int
    read an int
}
```

Sentinel-Based Looping

Write a prog that reads integers with a -1 sentinel and outputs their arithmetic mean.

Pseudo-code:

```
for (?; not sentinel; ?)
{
    process the int
    read an int
}
```

Priming needed

Sentinel-Based Looping

```
read an int
for (?; not sentinel; ?)
{
    process the int
    read an int
}
```

```
for (read an int; not sentinel; ?)
{
    process the int
    read an int
}
```


Sentinel-Based Looping

```
for (read an int; not sentinel; ?)
{
    process the int
    read an int
}
```

```
for (read an int; not sentinel; read an int)
{
    process the int
}
```


Sentinel-Based Looping

```
for (int n=input.nextInt(); not sentinel; n=input.nextInt())
{
   process the int
}
```

- How do you count the entries?
- How do you compute the mean?
- Is a cast needed?

Number Statistics

- Finding the max entry
- Using and challenging a candidate
- Seeding the candidate
- A multi-statement primer

5.2.4 Nested Loops

- Disjoint or fully nested
- Nested structures imply nested scopes

```
for (int i = 0; i < max; i++)
{
   for (int j = 0; j < max; j++)
   {
      display i and j
   }
}
```


5.3.1 Input Validation

Three ways for handling bad input:

- **Crash**
  Primitive (but better than no validation)
- **Print a message then end**
  Better. Requires an `else` statement to skip the rest of the program
- **Print a message and allow retries**
  Best. Requires an `if` statement inside a loop

5.3.1 Input Validation

Three ways for handling bad input:

- **Crash**
  Primitive (but better than no validation)
- **Print a message then end**
  Better. Requires an `else` statement to skip the rest of the program
- **Print a message and allow retries**
  Best. Requires an `if` statement inside a loop