CIS2571 – Intro to Java

Chapter 4 → Loops

repeat an action multiple times

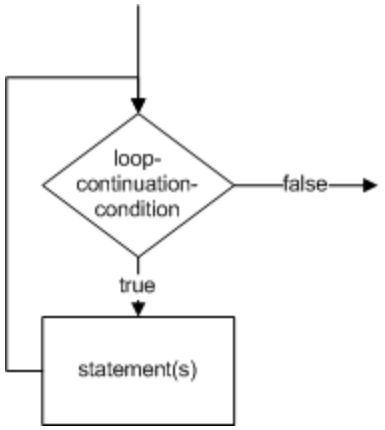
Information consolidated from a variety of textbook and online resources

Topic Objectives

- Practice the use of loops
  - while
  - do-while
  - for
- Understand the difference between
  - break
  - continue
- Understand how sentinel values control looping
- Know when to use which loop
- Know how to nest loops
- Be aware of some common problems when using loops
- Know how to use input and output redirection

The while Loop

```
while (loop-continuation-condition) {
    statements;
}
```



- loop body → part of loop containing statements to be repeated
- iteration → an execution of the loop body
- loop-continuation-condition → Boolean expression that controls execution of loop body
  - evaluated each time **prior** to execution of loop body
    - if **true**, execute loop body
    - if **false**, do not execute loop body
- block braces can be omitted if they enclose a single statement
- common programming error involves infinite loops; condition never becomes false

while loop Example

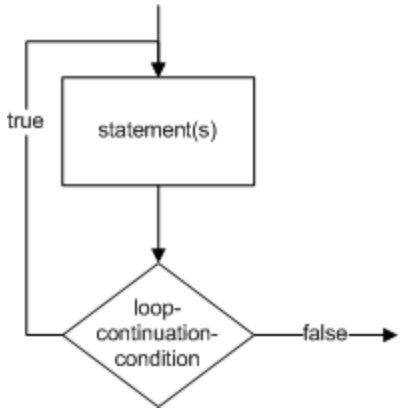
- Sum of numbers 0 to 4

```
int sum = 0, i = 1;
while (i < 5) {
   sum = sum + i;
   i++;
}
System.out.println("sum is " + sum); // sum is 10
```

loop iteration	condition	end-of-loop i	sum
		1	0
1	true	2	1
2	true	3	3
3	true	4	6
4	true	5	10
5	false		

The do-while Loop

```
do {
    statement(s);
} while(loop-continuation-condition);
```



- loop body → part of loop containing statements to be repeated
- iteration → an execution of the loop body
- loop-continuation-condition → Boolean expression that controls execution of loop body
  - evaluated each time **after** execution of loop body
    - if **true**, execute loop body
    - if **false**, do not execute loop body
- block braces can be omitted if they enclose a single statement
- common programming error involves infinite loops; condition never becomes false

do-while loop Example

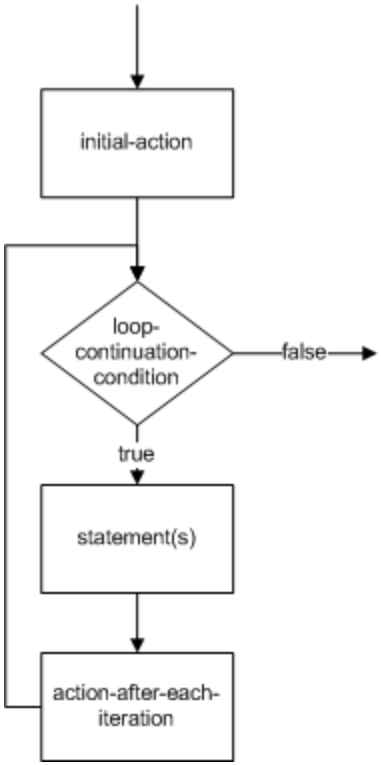
- Sum of numbers 0 to 4

```
int sum = 0, i = 1;
do {
   sum = sum + i;
   i++;
} while (i < 5);
System.out.println("sum is " + sum); // sum is 10
```

loop iteration	condition	end-of-loop i	sum
		1	0
1	true	2	1
2	true	3	3
3	true	4	6
4	false	5	10

The for Loop

```
for (initial-action;
    loop-continuation-condition;
       action-after-each-iteration) {
    statement(s);
}
```



- iteration → an execution of the loop body
- loop body → part of loop containing statements to be repeated
- initial-action → executes once
- loop-continuation-condition → Boolean expression that controls execution of loop body
  - evaluated each time **before** execution of loop body
    - if **true**, execute loop body
    - if **false**, do not execute loop body

The for Loop

```
for (initial-action;
    loop-continuation-condition;
       action-after-each-iteration) {
    statement(s);
}
```

- action-after-each-iteration → executed after iteration of loop body, before evaluation of loop-continuation-condition
- block braces can be omitted if they enclose a single statement
- use of floating point numbers in loop-continuation-condition may cause numeric errors

```
float sum = 0;
// Add 0.01, 0.02, ..., 0.99, 1 to sum
for (float i = 0.01f; i <= 1.0f; i = i + 0.01f)
    sum += i;
System.out.println("The sum is " + sum); // sum is 50.50?
```

```
The sum is 50.499985
```

for loop Example

- Sum of numbers 0 to 4

```
int sum, i;
for (sum = 0, i = 1; i < 5; i++){
   sum = sum + i;
}
System.out.println("sum is " + sum); // sum is 10
```

loop iteration	condition	end-of-loop i	sum
		1	0
1	true	2	1
2	true	3	3
3	true	4	6
4	true	5	10
5	false		

break and continue statements

- **break** was used in switch statement to transfer execution out of **switch** statement
- **break** can also be used to terminate execution of current iteration and transfer execution out of **loop**
  - prior to full loop iteration
- **continue** used to terminate execution of current iteration and transfer execution to next iteration of loop
  - **while** → check pretest loop-continuation-condition
  - **do-while** → check posttest loop-continuation-condition
  - **for** → execute action-after-each-iteration then check pretest loop-continuation-condition

break Example

- Sum of numbers 0 to 2

```
int sum, i;
for (sum = 0, i = 1; i < 5; i++){
   sum = sum + i;
   if (i == 2)
      break;
}
System.out.println("sum is " + sum); // sum is 3
```

loop iteration	condition	end-of-loop i	end-of-loop sum
		1	0
1	true	2	1
2	true	2*	3

**early loop termination*

continue Example

- Sum of even numbers 0 to 4

```
int sum, i;
for (sum = 0, i = 1; i < 5; i++){
   if (i % 2 == 1)
        continue;
   sum = sum + i;
}
System.out.println("sum is " + sum); // sum is 6
```

loop iteration	condition	end-of-loop i	sum
		1	0
1*	true	2	0
2	true	3	2
3*	true	4	2
4	true	5	6
5	false		

*loop continuation to next iteration

Controlling Loop with Sentinel Value

- Special value to signify the end of input
  - Invalid value
  - Listing 4.5 SentinelValue.java
    - *Calculates sum of integers; 0 indicates end of integer input*

```
Scanner input = new Scanner(System.in);
System.out.print(
    "Enter an integer (the input ends of it is 0):");
int data = input.nextInt();
int sum = 0;
while (data != 0){
    sum += data;
    System.out.print(
        "Enter an integer (the input ends of it is 0):");
    data = input.nextInt();
}
System.out.println("The sum is " + sum);
```

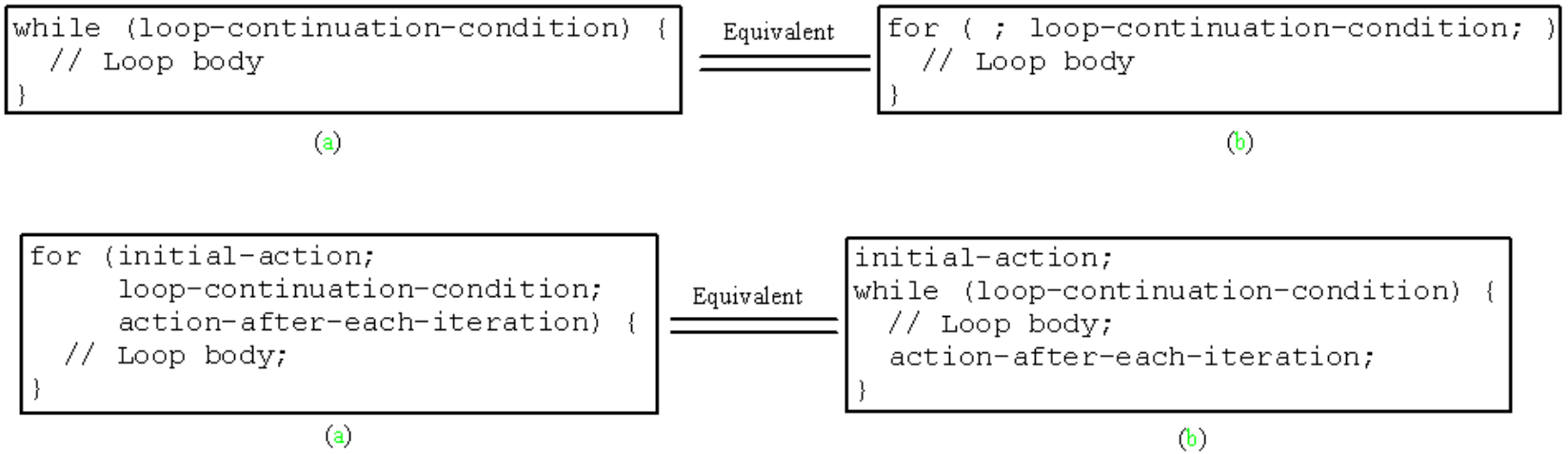
- Can be implemented using a confirmation dialog box
  - JOptionPane.YES_OPTION

Which loop to use when?

- Loop design strategies
  - Think before coding
    - Identify statements to be repeated
    - Wrap statements in a loop
  - Code incrementally
- **pretest** loop → checks condition before body executes
  - body may never execute
  - types:
    - while
    - for
- **posttest** loop → checks condition after body executes
  - body must execute at least once
  - type:
    - do-while

Which loop to use when?

- Some loops can be converted into another type
- Use one that is most intuitive and easy to understand



Nested Loops

- Outer loop and one or more inner loops
  - Each iteration of outer causes complete iteration of inner
- Example: Listing 4.7 Multiplication Table.java
  - *Prints multiplication table for integers 1 to 9*

```
for (int i = 1; i <= 5; i++) {
  System.out.print(i + " | ");
  for (int j = 1; j <= 5; j++) {
    System.out.printf("%4d", i * j);
  }
  System.out.println();
}
```

```
 i     j     i * j
----  ----   -----
 1     1      1
 2     1      2
 3     1      3
 4     1      4
 5     1      5
 1     2      2
 2     2      4
 3     2      6
 4     2      8
 5     2     10
 1     3      3
 2     3      6
 3     3      9
 4     3     12
 5     3     15
 1     4      4
 2     4      8
 3     4     12
 4     4     16
 5     4     20
 1     5      5
 2     5     10
 3     5     15
 4     5     20
 5     5     25
```

Caution: Common Errors

- Infinite loops
  - Loops that have a condition that is always true and will never cause the loop to terminate
  - **break** statements can be used to transfer control out of the loop
  - Sometimes used for keyboard interrupt

```
for ( ; ; ) {
  // Do something
}
```

(a)

Equivalent

```
while (true) {
  // Do something
}
```

(b)

Caution: Common Errors

- Misplaced semicolon at end of condition

```
int sum = 0, i = 1;
while (i < 5);
{
   sum = sum + i;
   i++;
}
System.out.println("sum is " + sum); // sum is 10
```

```
int sum, i;
for (sum = 0, i = 1; i < 5; i++);
{
   sum = sum + i;
}
System.out.println("sum is " + sum); // sum is 10
```

Input and Output Redirection

- Preferred method for large number of data
- Input from file instead of keyboard
- Output to file instead of screen
- Data separated by whitespaces in a text file
- Run bytecode from command line
  - For input:

```
java SentinelValue < input.txt
```

  - For output

```
java SentinelValue > output.txt
```

  - For input and output

```
java SentinelValue < input.txt > output.txt
```

See 4.5 SentinelValue.java