CS 106A, Lecture 5
Java Control Statements and Logic

reading:

Art & Science of Java, 3.4 - 3.6, 4.2 - 4.6

Interactive programs

- **interactive program**: Reads input from the console.
  - The program pauses, waiting for the user to type a value.
  - The value typed by the user is stored in a variable.

- `ConsoleProgram` methods for reading user input:

Method	Description
`readInt(`***msg***`)`	reads an `int` value from the user
`readDouble(`***msg***`)`	reads a `double` value from the user
`readLine(`***msg***`)`	reads a one-line `String` from the user
`readBoolean(`***msg***`)`	reads a `boolean` value from the user

```
int ssn = readInt("Type your social security #: ");
```

User input example

```
public class UserInputExample extends ConsoleProgram {
    public void run() {
        int age = readInt("How old are you? ");
        int years = 65 - age;
        println(years + " years until retirement!");
    }
}
```

age	29
years	36

- Console (user input underlined):

```
How old are you? 29
36 years until retirement!
```

Exercise: Receipt2

- Modify the `Receipt` program from Monday's lecture so it prompts the user to type the subtotal cost of the meal and computes the total based on that.

```
What was the meal cost? 50
Tax: 4.0
Tip: 7.5
Total: 61.5

What was the meal cost? 125
Tax: 10.0
Tip: 18.75
Total: 153.75
```

Receipt2 solution

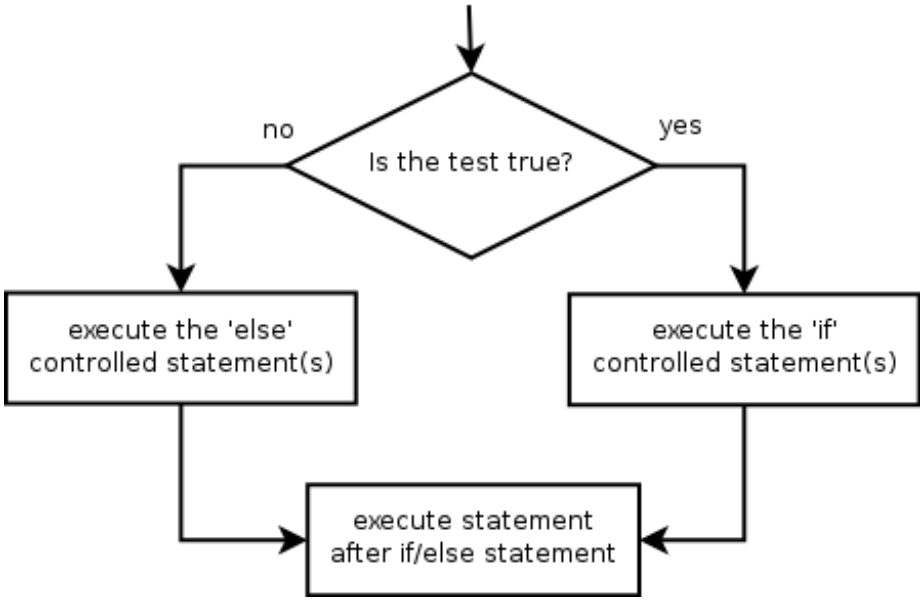
```
import acm.program.*;

public class Receipt2 extends ConsoleProgram {
    public void run() {
        // Calculate total owed, assuming 8% tax / 15% tip
        int subtotal = readInt("What was the meal cost? $");
        double tax = subtotal * .08;
        double tip = subtotal * .15;
        double total = subtotal + tax + tip;

        println("Tax: $" + tax);
        println("Tip: $" + tip);
        println("Total: $" + total);
    }
}
```

Revisiting if/else

Revisiting if/else

Executes one group if a test is true, another if false

```
if (test) {
    statements;
} else {
    statements;
}
```



- Example:

```
int age = readInt("Your age? ");
if (age < 40) {
    println("young");
} else {
    println("young at heart");
}
```

Relational operators

Operator	Meaning	Example	Value
`==`	equals	`1 + 1 == 2`	`true`
`!=`	does not equal	`3.2 != 2.5`	`true`
`<`	less than	`10 < 5`	`false`
`>`	greater than	`10 > 5`	`true`
`<=`	less than or equal to	`126 <= 100`	`false`
`>=`	greater than or equal to	`5.0 >= 5.0`	`true`

- `if` statements and loops use logical tests.

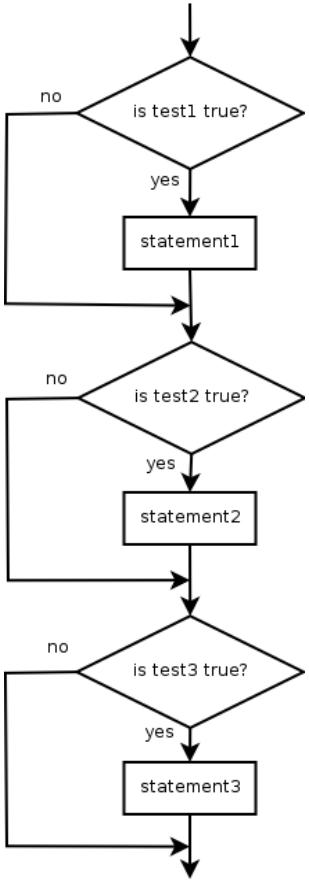
```
for (int i = 0; i < 10; i++) { ...
if (age >= 40) { ...
```

  – These are `boolean` expressions. Boolean is a logical data type.

Misuse of if

- What's wrong with the following code?

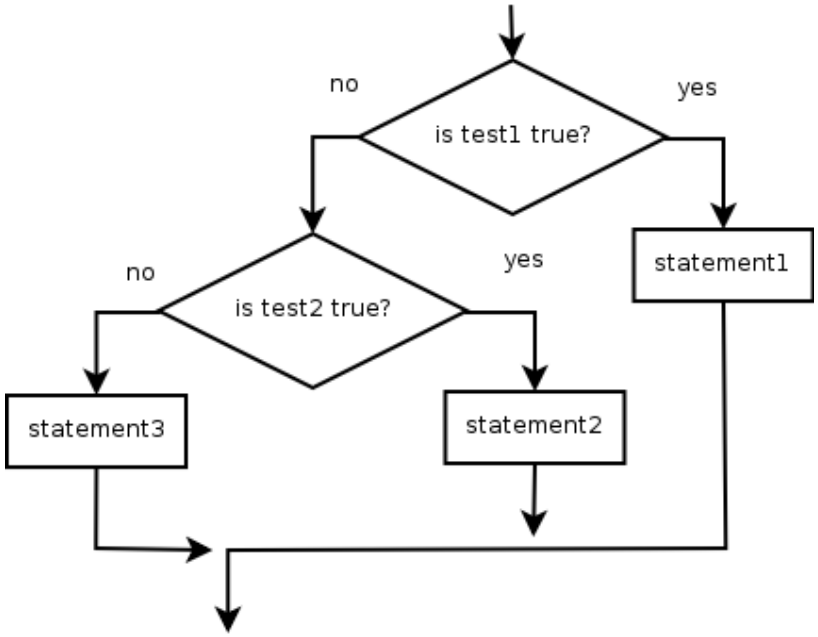
```
int percent = readInt("What percentage did you earn? ");
if (percent >= 90) {
    println("You got an A!");
}
if (percent >= 80) {
    println("You got a B!");
}
if (percent >= 70) {
    println("You got a C!");
}
if (percent >= 60) {
    println("You got a D!");
}
if (percent < 60) {
    println("You got an NP!");
}
...
```



Nested if/else

Chooses between outcomes using many tests

```
if (test) {
    statements;
} else if (test) {
    statements;
} else {
    statements;
}
```



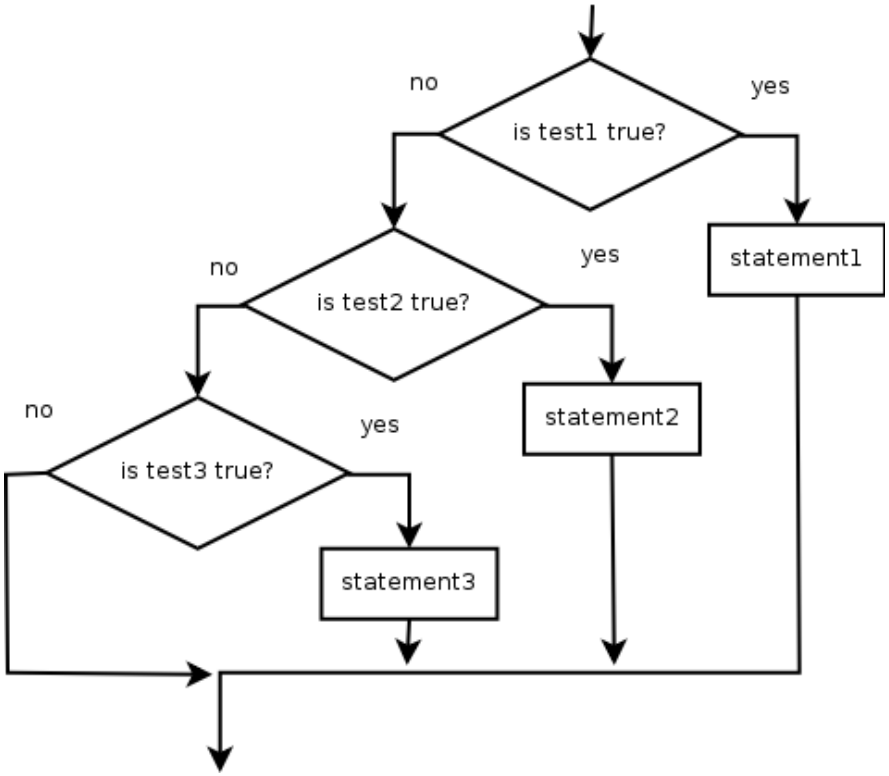
- Example:

```
if (x > 0) {
    println("Positive");
} else if (x < 0) {
    println("Negative");
} else {
    println("Zero");
}
```

Nested if/else/if

Ends with `else`: exactly 1 path taken. *Ends with `if`: 0-1 paths taken.*

```
if (test) {
    statements;
} else if (test) {
    statements;
} else if (test) {
    statements;
}
```



- Example:

```
if (place == 1) {
    println("Gold medal!!");
} else if (place == 2) {
    println("Silver medal!");
} else if (place == 3) {
    println("Bronze medal.");
}
```

Unnecessary if

- The following code is unnecessarily verbose and redundant:

```
if (x < 0) {
    println("x is negative");
} else if (x >= 0) {
    println("x is non-negative");
}
```

- The second test is unnecessary and can be removed:

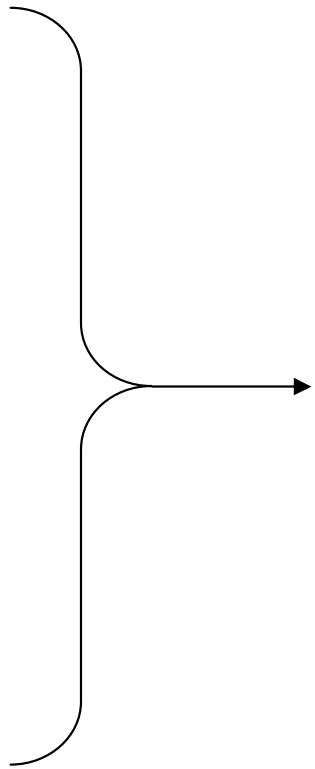
```
if (x < 0) {
    println("x is negative");
} else {
    println("x is non-negative");
}
```

Factoring if/else

- **factoring**: Extracting common/redundant code.
  - Can reduce or eliminate redundancy from `if/else` code.

- Example:

```
if (a == 1) {
    println(a);
    x = 3;
    b = b + x;
} else if (a == 2) {
    println(a);
    x = 6;
    y = y + 10;
    b = b + x;
} else {  // a == 3
    println(a);
    x = 9;
    b = b + x;
}
```



```
println(a);
x = 3 * a;
if (a == 2) {
    y = y + 10;
}
b = b + x;
```

Logical operators

- Tests can be combined using *logical operators*:

Operator	Description	Example	Result
&&	and	`(2 == 3) && (-1 < 5)`	`false`
||	or	`(2 == 3) || (-1 < 5)`	`true`
!	not	`!(2 == 3)`	`true`

- "Truth tables" for each, used with logical tests *p* and *q*:

p	q	p && q	p || q
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

p	!p
true	false
false	true

Evaluating logic exprs

- Precedence: arithmetic > relational > logical

```
5 * 7 >= 3 + 5 * (7 - 1) && 7 <= 11
5 * 7 >= 3 + 5 * 6 && 7 <= 11
35    >= 3 + 30 && 7 <= 11
35    >= 33 && 7 <= 11
true && true
true
```

- Cannot "chain" tests as in algebra; use && or || instead

```
// assume x is 15
2 <= x <= 10
true   <= 10
Error!
```

```
// correct version
2 <= x && x <= 10
true   && false
false
```

  – *Exercise:* Write a program that prompts for information about a potential dating partner and decides whether or not to date them.

Revisiting for

Revisiting for loops

```
for (initialization; test; update) {      } header
    statements;                           } body
}
```

- Perform ***initialization*** once.
- Repeat the following:
  - Check if the ***test*** is true. If not, stop.
  - Execute the ***statements***.
  - Perform the ***update***.

- If any variables are declared in the ***initialization***, they can be used in the body of the loop.

Increment/decrement

shortcuts to increase or decrease a variable's value by 1

Shorthand	Equivalent longer version
***variable*++;**	***variable* = *variable* + 1;**
***variable*--;**	***variable* = *variable* - 1;**

```
int x = 2;
x++;                    // x = x + 1;
                        // x now stores 3

double gpa = 2.5;
gpa--;                  // gpa = gpa - 1;
                        // gpa now stores 1.5
```

Using the loop variable

```
println("Let's count!");
for (int i = 0; i < 5; i++) {
    println(i + "...");
}
println("Time's up!");
```

- Output:

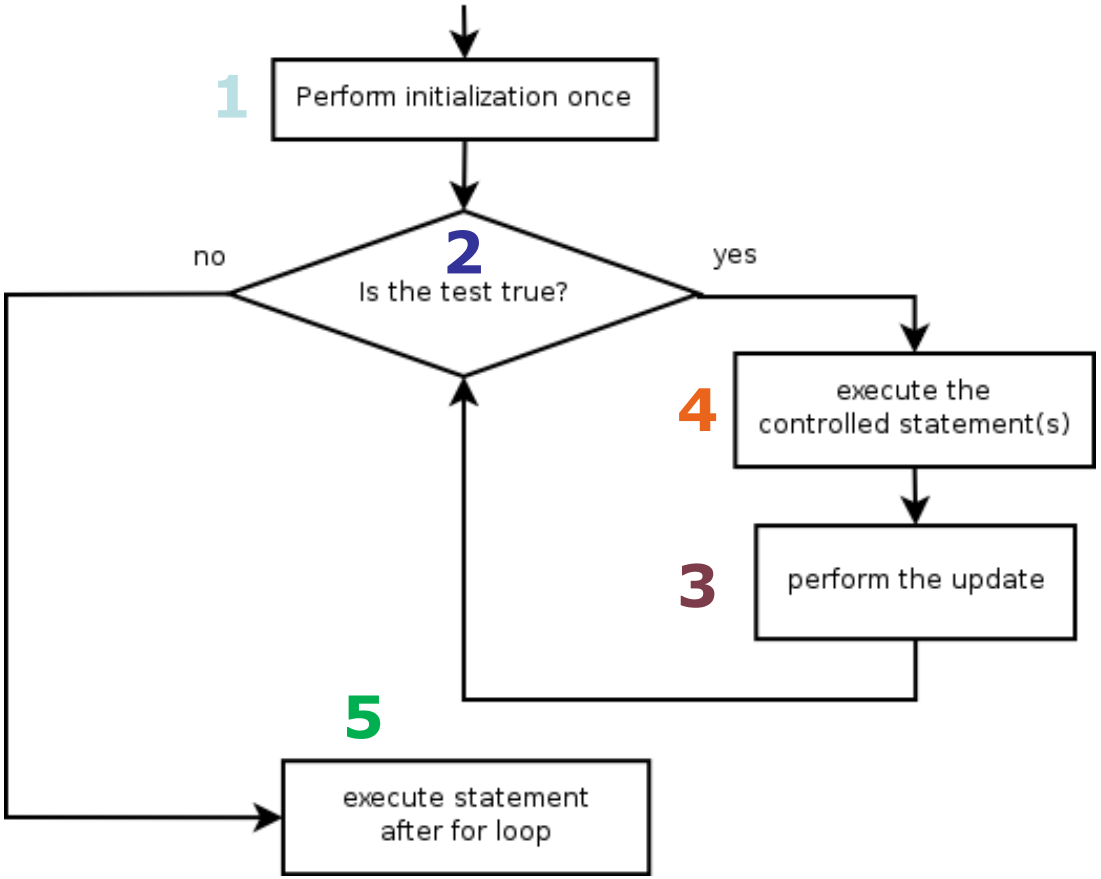
```
Let's count!
0...
1...
2...
3...
4...
Time's up!
```

Loop walkthrough

```
                1        2      3
  for (int i = 1; i <= 4; i++) {
    4 println(i + " squared = " + (i * i));
  }
5 println("Whoo!");
```

Output:

```
1 squared = 1
2 squared = 4
3 squared = 9
4 squared = 16
Whoo!
```



Modify-and-assign

shortcuts to modify a variable's value

Shorthand	Equivalent longer version
`variable += value;`	`variable = variable + value;`
`variable -= value;`	`variable = variable - value;`
`variable *= value;`	`variable = variable * value;`
`variable /= value;`	`variable = variable / value;`
`variable %= value;`	`variable = variable % value;`

```
x += 3;                    // x = x + 3;
gpa -= 0.5;                // gpa = gpa - 0.5;
number *= 2;               // number = number * 2;
```

Decrementing loops

- The ***update*** can use `--` to make the loop count down.
  - The ***test*** must say > instead of <
  - The `print` method displays output without going to the next line.

```
print("T-minus ");
for (int i = 10; i >= 1; i--) {
     print(i + " ");
}
println("blastoff!");
println("The end.");
```

  - Output:

```
T-minus 10 9 8 7 6 5 4 3 2 1 blastoff!
The end.
```

Cumulative loops

```
int sum = 0;
for (int i = 1; i <= 1000; i++) {
    sum = sum + i;
}
println("The sum is " + sum);
```

- **cumulative sum**: A variable that keeps a sum in progress and is updated repeatedly until summing is finished.
  - The `sum` in the above code is a cumulative sum.
  - Cumulative sum variables must be declared *outside* the loops that update them, so that they will still exist after the loop.

Nested loops

- **nested loop**: A loop placed inside another loop.

```
for (int i = 1; i <= 5; i++) {
    for (int j = 1; j <= 10; j++) {
        print("*");
    }
    println();   // to end the line
}
```

- Output:

```
**********
**********
**********
**********
**********
```

- The outer loop repeats 5 times; the inner one 10 times.

Nested loop question

- **Q:** What output is produced by the following code?

```
for (int i = 1; i <= 5; i++) {
    for (int j = 1; j <= i; j++) {
        print("*");
    }
    println();
}
```

A.
```
*****
*****
*****
*****
*****
```

B.
```
*****
****
***
**
*
```

C.
```
*
**
***
****
*****
```

D.
```
1
22
333
4444
55555
```

E.
```
12345
```

(How would you modify the code to produce each output above?)

Nested loop question 2

- How would we produce the following output?

```
....1
...22
..333
.4444
55555
```

- Answer:

```
for (int i = 1; i <= 5; i++) {
    for (int j = 1; j <= 5 - i; j++) {
        print(".");
    }
    for (int j = 1; j <= i; j++) {
        print(i);
    }
    println();
}
```

Nested loop question 3

- How would we produce the following output?

```
....1
...2.
..3..
.4...
5....
```

- Answer:

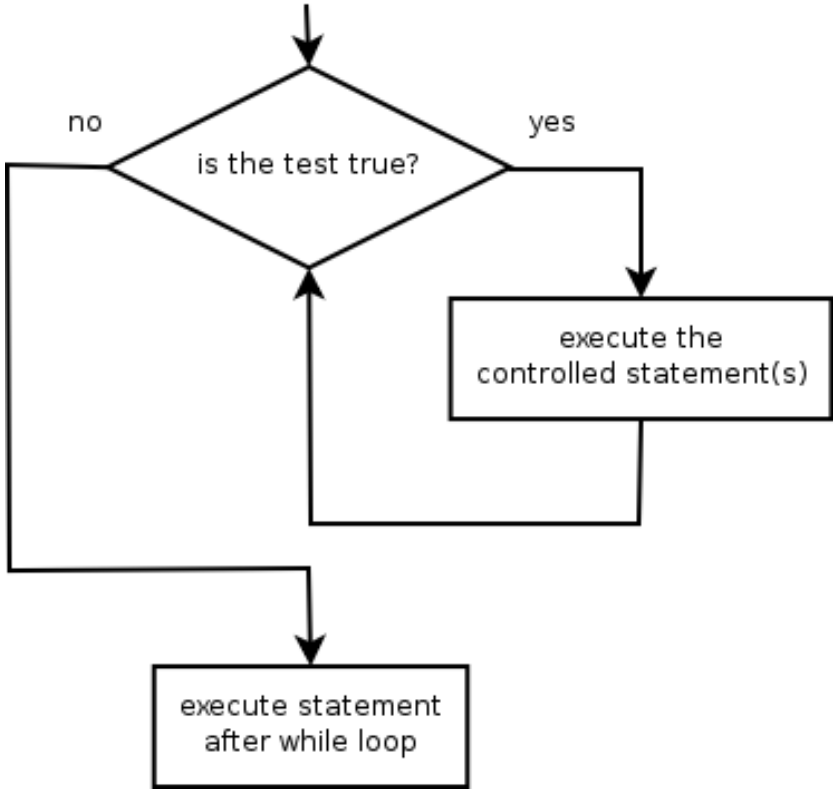
```
for (int i = 1; i <= 5; i++) {
    for (int j = 1; j <= 5 - i; j++) {
        print(".");
    }
    print(i);
    for (int j = 1; j <= i - 1; j++) {
        print(".");
    }
    println();
}
```

Revisiting while

Revisiting while

Repeatedly executes its body as long as a logical test is true

```
while (test) {
    statements;
}
```



- Example:

```
int x = 0;
while (x < 10) {
    println(x);
    x++;
}
```

  – The above is similar to a `for` loop from 0-10.

Sentinel loops

- **sentinel**: A value that signals the end of user input.
  - **sentinel loop**: Repeats until a sentinel value is seen.

- Example: Write a program that prompts the user for numbers until the user types 0, then output the sum of the numbers.
  - In this case, 0 is the sentinel value.

```
Type a number: 10
Type a number: 20
Type a number: 30
Type a number: 0
Sum is 60
```

Sentinel solution?

- This solution *seems to* work just fine ...

```
int sum = 0;
int n = 1;          // "dummy" value, anything but 0
while (n != 0) {
    n = readInt("Type a number: ");
    sum += n;
}
println("Sum is " + sum);
```

- Example output:

```
Type a number: 10
Type a number: 20
Type a number: 30
Type a number: 0
Sum is 60
```

Incorrect solution

- Change the sentinel to -1. The solution now fails. Why?

```
int sum = 0;
int n = 1;          // "dummy" value, anything but -1
while (n != -1) {
    n = readInt("Type a number: ");
    sum += n;
}
println("Sum is " + sum);
```

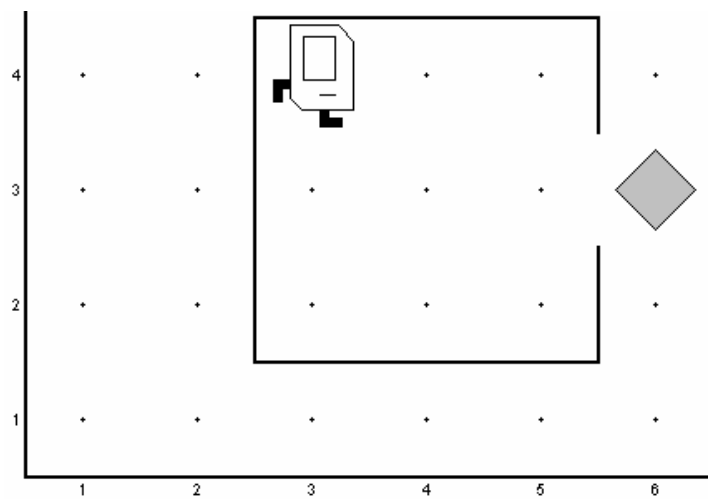
- Example output:

```
Type a number: 10
Type a number: 20
Type a number: 30
Type a number: -1
Sum is 59
```

Semantics of while

- A loop always executes its body completely, never partially.
  - If the loop's test becomes false, the loop stops at the end of the body.

```
// Karel example: move 3 squares at a time (bad)
while (frontIsClear()) {
    move();
    move();
    move();
}
```



Sentinel fix #1

- Prompt for the first number outside the `while` loop.
  - This is really a fencepost problem. Move a "post" (prompt) out.
  - Reverse the order of the two statements in the `while` loop.

```
int sum = 0;
int n = readInt("Type a number: ");
while (n != -1) {
    sum += n;
    n = readInt("Type a number: ");
}
println("Sum is " + sum);
```

Sentinel fix #2

- For this particular problem, simply initializing the sum to 0 will work because the 0 gets added to the sum and doesn't affect it.
  - Would not work for some other problems, e.g. finding max/min value

```
int sum = 0;
int n = 0;   // must be 0 to avoid corrupting sum
while (n != -1) {
    sum += n;
    n = readInt("Type a number: ");
}
println("Sum is " + sum);
```

Sentinel fix #3

- A `while (true)` loop continues until it is manually stopped using a command called `break`.
  - Sometimes called an *infinite loop*, *forever loop*, or *loop-and-a-half*

```
int sum = 0;
while (true) {
    int n = readInt("Type a number: ");
    if (n == -1) {
        break;        // exit the loop
    }
    sum += n;
}
println("Sum is " + sum);
```

Overflow (extra) slides

Exercise: digit sum

- Write a program that prompts for an integer and adds the digits of that integer.

```
Type an integer: 827104
Digit sum is 22
```

Digit sum solution

```
import acm.program.*;

public class DigitSum extends ConsoleProgram {
    public void run() {
        int n = readInt("Type an integer: ");
        int sum = 0;
        while (n > 0) {
            int lastDigit = n % 10;
            sum += lastDigit;
            n = n / 10;
        }
        println("Digit sum is " + sum);
    }
}
```