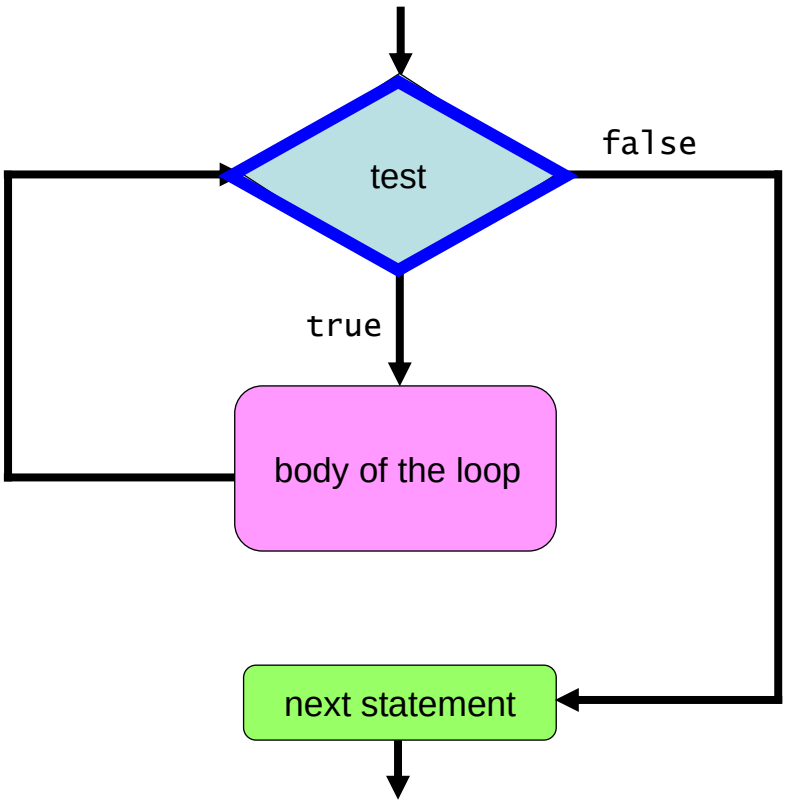
From Python to Java: Loops

Computer Science 112
Boston University

Christine Papadakis

The while loop

```
while ( continuation test ) {
    body
}
```



- In a `while`, the test is evaluated first
  - thus, the body may not be executed if the first test fails!

while Loops

Python

```python
def fac(n):
    result = 1
    while n > 1:
        result = result * n
        n = n - 1
    return result
```

Java

```java
public static int fac(int n) {
    int result = 1;
    while (n > 1) {
        result = result * n;
        n = n - 1;
    }
    return result;
}
```

- Here again, the Java version needs:
  - parenthesis around the condition (no colon)
  - curly braces around the block (the *body* of the loop)

`while` Loops

Python

```
def fac(n):
    result = 1
    while n > 1:
        result = result * n
        n = n - 1
    return result
```

Java

```
public static int fac(int n) {
    int result = 1;
    while (n > 1) {
        result = result * n;
        n = n - 1;
    }
    return result;
}
```

- Here again, the Java version needs:
  - parenthesis around the condition (no colon)
  - curly braces around the block (the *body* of the loop)
- The `int` type in Java uses 4 bytes per integer.
  - gives an approx. domain of [-2 billion, +2 billion]

`while` Loops

Python

```python
def fac(n):
    result = 1
    while n > 1:
        result = result * n
        n = n - 1
    return result
```

Java

```java
public static long fac(int n) {
    long result = 1;
    while (n > 1) {
        result = result * n;
        n = n - 1;
    }
    return result;
}
```

- Here again, the Java version needs:
  - parenthesis around the condition (no colon)
  - curly braces around the block (the *body* of the loop)
- The `int` type in Java uses 4 bytes per integer.
  - gives an approx. domain of [-2 billion, +2 billion]
- For larger integers, we can use the `long` type instead.
  - allocates 8 bytes per integer

`while` Loops

Python

```
def fac(n):
    result = 1
    while n > 1:
        result = result * n
        n = n - 1
    return result
```

Java

```
public static unsigned long fac(int n)
    unsigned long result = 1;
    while (n > 1) {
        result = result * n;
        n = n - 1;
    }
    return result;
}
```

- Here again, the Java version needs:
  - parenthesis around the condition (no colon)
  - curly braces around the block (the *body* of the loop)
- The `int` type in Java uses 4 bytes per integer.
  - gives an approx. domain of [-2 billion, +2 billion]
- For larger integers, we can use the `long` type instead.
  - allocates 8 bytes per integer
  - only positive integers

Shortcut Operators

Python

```
def fac(n):
    result = 1
    while n > 1:
        result *= n
        n -= 1
    return result
```

Java

```
public static long fac(int n) {
    long result = 1;
    while (n > 1) {
        result *= n;
        n -= 1;
    }
    return result;
}
```

- Python has operators that combine arithmetic with assignment:

  `+=`
  `-=`
  `*=`
  etc.

- Java also has these operators.

- In addition, it has two special ones for adding/subtracting 1:

  `x++` *is the same as* `x = x + 1`

  `x--` *is the same as* `x = x - 1`

Shortcut Operators

Python

```
def fac(n):
    result = 1
    while n > 1:
        result *= n
        n -= 1
    return result
```

Java

```
public static long fac(int n) {
    long result = 1;
    while (n > 1) {
        result *= n;
        n--;
    }
    return result;
}
```

- Python has operators that combine arithmetic with assignment:

  `+=`
  `-=`
  `*=`
  etc.

- Java also has these operators.

- In addition, it has two special ones for adding/subtracting 1:

  `x++` *is the same as* `x = x + 1`

  `x--` *is the same as* `x = x - 1`

Shortcut Operators

CAREFULL!!!

Java allows for **pre** and **post** increment and decrement operators

x++;

++x;

x--;

--x;

As independent statements they are identical, *however*, as part of an expression they behave differently!

Shortcut Operators

```
def
```

```
int k = 5;
int j = k++;
```

```
int k = 5;
int j = ++k;
```

post increment

Increment *after* assignment

pre increment

Increment *before* the assignment

- 1

- 1

etc.

Shortcut Operators

```
int k = 5;
int j = k++;
```

```
int k = 5;
int j = ++k;
```

```
System.out.println(k + " " + j);
```

6 5

Shortcut Operators

```
int k = 5;
int j = k++;
```

```
int k = 5;
int j = ++k;
```

```
System.out.println(k + " " + j);
```

6 6

Counter controlled while() loop

Design a while loop to execute some *body of code* 10 times.

```
int counter = 1;          // declare and initialize a variable
                          // to control the execution of the loop

while (counter <= 10) {       // base condition
                              // on this control variable

    Code to execute

    counter++;            // increment the control variable
                          // with each iteration of the loop
}
```

Counter controlled while() loop

Design a while loop to execute some *body of code* 10 times.

```
int counter = 1;          // declare a
                          //

while (counter <= 10) {   //

    counter++;            // increment the control
                          // with each iteration of the loop
}
```

Code to execute

Note the three keys pieces of this loop:

1. declare and initialize the control variable.

Counter controlled while() loop

Design a while loop to execute some *body of code* 10 times.

```
int counter = 1;          // declare a
                          // to con

while (counter <= 10) {   //

    counter++;            // increment the control  e
                          // with each iteration of the loop
}
```

Code to execute

Note the three keys pieces of this loop:

1. declare and initialize the control variable.
2. base the conditional expression on the control variable.

Recall:
Counter controlled while() loop

Design a while loop to execute some *body of code* 10 times.

```
int counter = 1;          // declare a
                          // to con

while (counter <= 10) {   /

    counter++;            // increment the control  e
                          // with each iteration of the loop
}
```

Code to execute

Note the three keys pieces of this loop:

1. declare and initialize the control variable.
2. base the conditional expression on the control variable.
3. adjust the control variable.

`for` Loops in Java

- Syntax:

```
for (initialization; continuation-test; update) {
    body
}
```

- In our example:

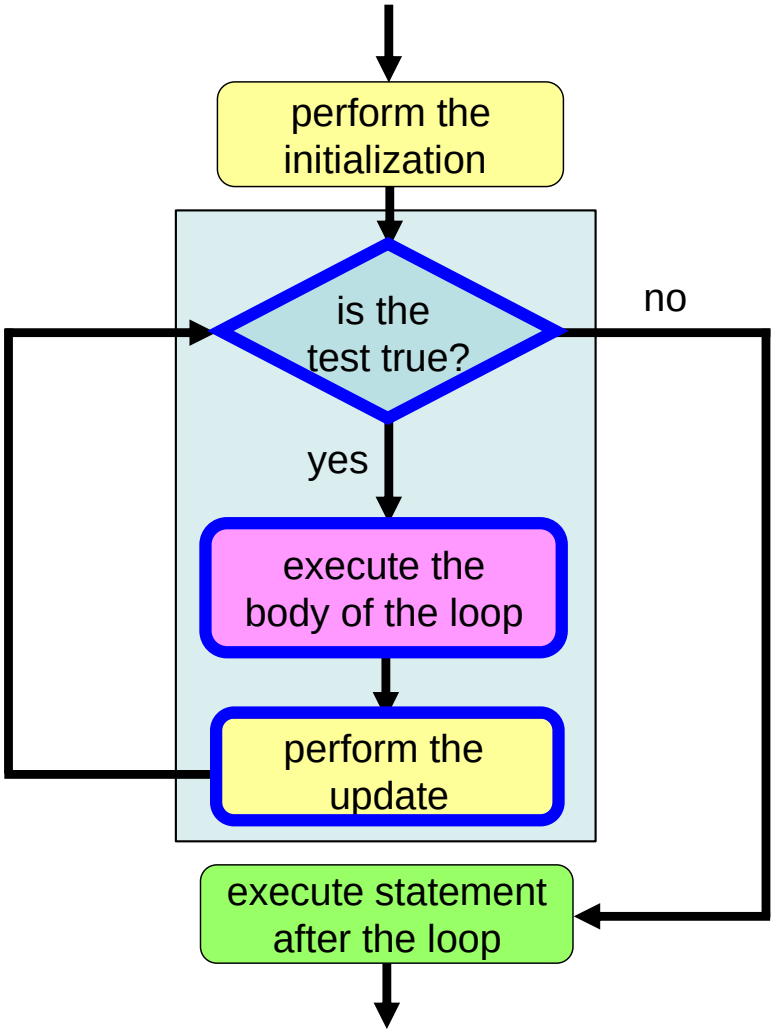
initialization *continuation test*

```
for (int x = 2; x <= n; x++) {
    result = result * x;
}
```

update

Executing a `for` Loop

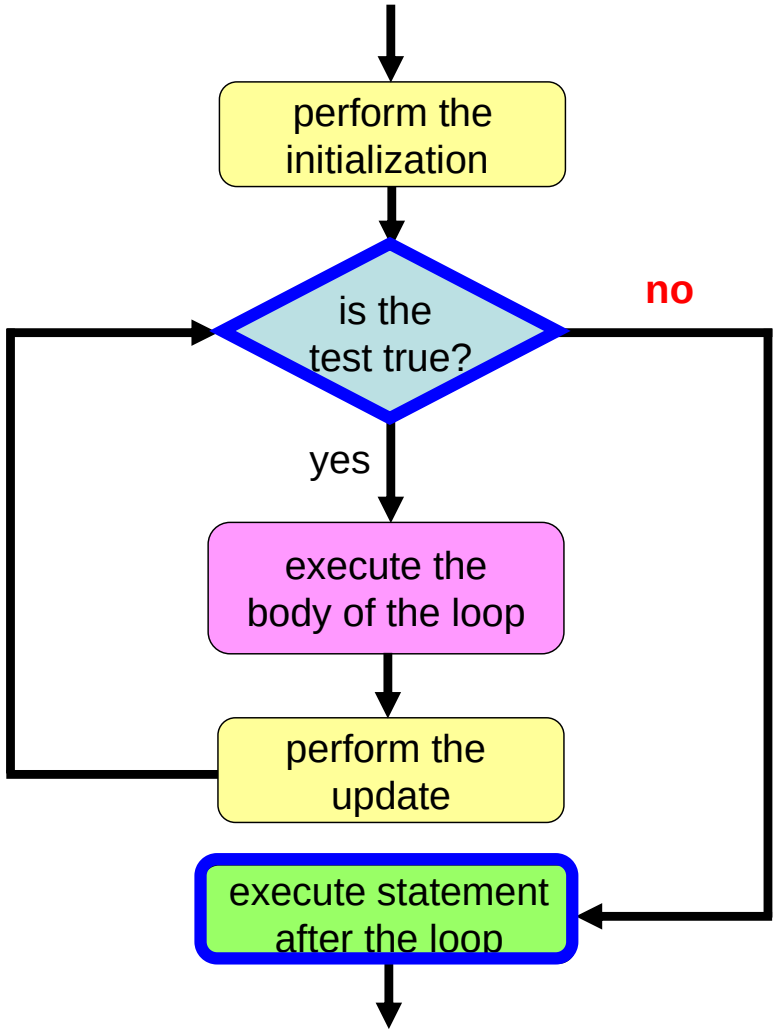
```
for (initialization; continuation-test; update) {
    body
}
```



Repeat these steps until the conditional expression evaluates to false!

Executing a `for` Loop

```
for (initialization; continuation-test; update) {
    body
}
```

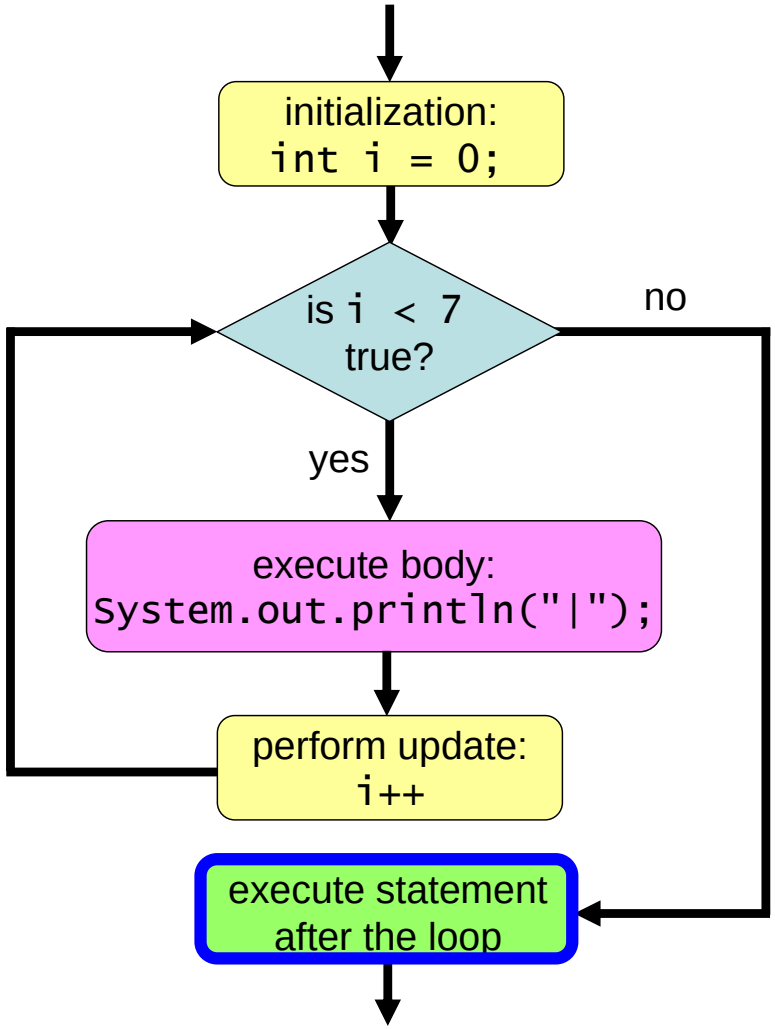


Notes:

- When the conditional expression evaluates to false, execute the statement directly following the for loop.

Tracing a `for` Loop

```
for (int i = 0; i < 7; i++) {
    System.out.println("|");
}
```



i	i < 7	action
0	true	print 1st "|"
1	true	print 2nd "|"
2	true	print 3rd "|"
3	true	print 4th "|"
4	true	print 5th "|"
5	true	print 6th "|"
6	true	print 7th "|"
7	false	execute stmt. after the loop

To Get *N* Repetitions

Python **Java**

- templates:

```
for i in range(N):
    body of loop
```

```
for (int i = 0; i < N; i++) {
    body of loop
}
```

- example: what do these print?

```
for i in range(5):
    print('Hip, hip!')
    print('Hooray!')
```

```
for (int i = 0; i < 5; i++) {
    System.out.println("Hip, hip!");
    System.out.println("Hooray!");
}
```

- to get 100 repetitions instead:

```
for i in range(100):
    print('Hip, hip!')
    print('Hooray!')
```

```
for (int i = 0; i < 100; i++) {
    System.out.println("Hip, hip!");
    System.out.println("Hooray!");
}
```

Tracing a `for` Loop

- Trace this loop by filling in the table::

```
for (int i = 2; i <= 10; i += 2) {
    System.out.println(i * 10);
}
```

i	i <= 10	value printed
2	true	20
4	true	40
6	true	60
8	true	80
10	true	100
12	false, so we exit the loop	

Tracing a `for` Loop

- Trace this loop by filling in the ta

```
for (int i = 2; i <= 1
    System.out.println
}
```

What would happen if we added this statement after the for loop?

```
System.out.println("value of is " + i);
```

i	i <= 10	value printed
2	true	20
4	true	40
6	true	60
8	true	80
10	true	100
12	false, so we exit the loop	

Tracing a `for` Loop

- Trace this loop by filling in the ta

```
for (int i = 2; i <= 1
    System.out.println
}
```

```
System.out.println("value of is " + i);
```

We would get an error because variable **i** exists only in the scope of the `for` loop!

i	i <= 10	value printed
2	true	20
4	true	40
6	true	60
8	true	80
10	true	100
12	false, so we exit the loop	

Common Mistake

- You should <u>not</u> put a semi-colon after the for-loop header:

```
for (int i = 0; i < 7; i++); {
    System.out.println("|");
}
```

What would happen if we added a semi colon at the end of the for loop header.

Common Mistake

- You should <u>not</u> put a semi-colon after the for-loop header:

```
for (int i = 0; i < 7; i++); {
    System.out.println("|");
}
```

- The semi-colon ends the for statement and thus, the loop ends and nothing is repeated.

- **T**he *println* is treated as an independent statement (following the for statement) and executed once.

Which option(s) work?

- Fill in the blanks below to print the integers from 1 to 5:

```
for (____________; ____________; ____________) {
    System.out.println(i);
}
```

A. `int i = 1; i < 5; i = i + 1`

B. `int i = 1; i < 6; i += 1`

C. `int i = 1; i <= 5; i++`

D. two of the above (B and C)

E. A, B, and C all work

Practice

- Fill in the blanks below to print the integers from 1 to 10:

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

- Fill in the blanks below to print the integers from 10 to 20:

```
for (int i = 10; i <= 20; i++) {
    System.out.println(i);
}
```

- Fill in the blanks below to print the integers from 10 down to 1:

```
for (int i = 10; i >= 1; i--) {
    System.out.println(i);
}
```

Practice

- How about a loop to print every 3rd number from 100 to 20?

```
for (int x = 100; x >= 20; x-=3) {
        System.out.println(x);
}
```

Practice

- How about a loop to print every 3rd number from 100 to 20?

Note the conditional operator used!

```
for (int x = 100; x >= 20; x-=3) {
        System.out.println(x);
}
```

Practice

- What is the output when this loop executes?

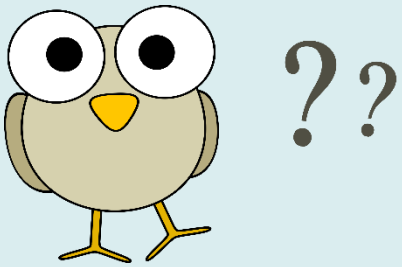
```
for (int x = 1; x < 10; x+=2)
   System.out.print(x + “-”);
   System.out.println( “Hello” );
```

Practice

- What is the output when this loop executes?

```
for (int x = 1; x < 10; x+=2)
   System.out.print(x + "-");
   System.out.println( "Hello" );

Output: 1-3-5-7-9-Hello
```



Practice

- What is the output when this loop executes?

What is the body of code associated with this loop?

```
for (int x = 1; x < 10; x+=2)
   System.out.print(x + “-”);
   System.out.println( “Hello” );
```

Practice

- What is the output when this loop executes?

The single statement directly following the for statement!

```
for (int x = 1; x < 10; x+=2)
   System.out.print(x + "-");
   System.out.println( "Hello" );
```

Practice

- What is the output when this loop executes?

Note the use of *print* and not *println* here!

```
for (int x = 1; x < 10; x+=2)
   System.out.print(x + "-");
   System.out.println( "Hello" );
```

Practice

- What is the output when this loop executes?

This code executes following this logical structure:

```
for (int x = 1; x < 10; x+=2)
   System.out.print(x + "-");

System.out.println( "Hello" );
```

Practice

- What is the output when this loop executes?

In Java for both of these statements to be executed with each iteration of the loop, we must use { }:

```
for (int x = 1; x < 10; x+=2) {
   System.out.print(x + "-");
   System.out.println( "Hello" );

}
```

Practice

- What is the output when this loop executes?

What is the value of x when the loop concludes?

```
for (int x = 1; x < 10; x+=2) {
   System.out.print(x + "-");
   System.out.println( "Hello" );

}
```

Practice

- What is the output when this loop executes?

What is the value of x when the loop concludes?
11

```
for (int x = 1; x < 10; x+=2) {
   System.out.print(x + "-");
   System.out.println( "Hello" );

}
```

Practice

Let’s go Loopy with Loops!!!

Practice

- Write three different loop variations, each to execute 20 times:

```
for (____________; ____________; ____________) {
    System.out.println(i);
}
```

- Fill in the blanks below to print the integers from 10 to 20:

```
for (____________; ____________; ____________) {
    System.out.println(i);
}
```

- Fill in the blanks below to print the integers from 10 down to 1:

```
for (____________; ____________; ____________) {
    System.out.println(i);
}
```

Nested loops: **Rows** vs. **columns**

For these loops, let **side=5**:

Design nested loops (in Java) to display each of the shapes to the right!!!

columns

rows

```
0 0 0 0 0
1 1 1 1 1
2 2 2 2 3
3 3 3 3 3
4 4 4 4 4
```

```
  0 1 2 3 4
0 0 0 0 0 0
1 1 1 1 1
2 2 2 2
3 3 3
4 4
```

```
  0 1 2 3 4
0 0 0 0 0 0
1 1 1 1 1
2 2 2 2
3 3 3 3
4 4 4 4 4 4
```

Extra! how about *this hourglass*?

```
0 0 0 0 0
  1 1 1
    2
  3 3 3
4 4 4 4 4
```

Indefinite Loops in Java,

a variation

- We typically use a `while` loop when we need an *indefinite* loop.
  - when we don't know how many repetitions we need

- Example: getting a positive integer from the user:

```
Scanner scan = new Scanner(System.in);
System.out.print("Enter a positive integer: ");
int num = console.nextInt();
while (num <= 0) {
    System.out.print("Enter a positive integer: ");
    num = scan.nextInt();
}
```

Indefinite Loops in Java,

a variation

- We typically use a `while` loop when we need an *indefinite* loop.
  - when we don't know how many repetitions we need

- Example: getting a positive integer from the user:

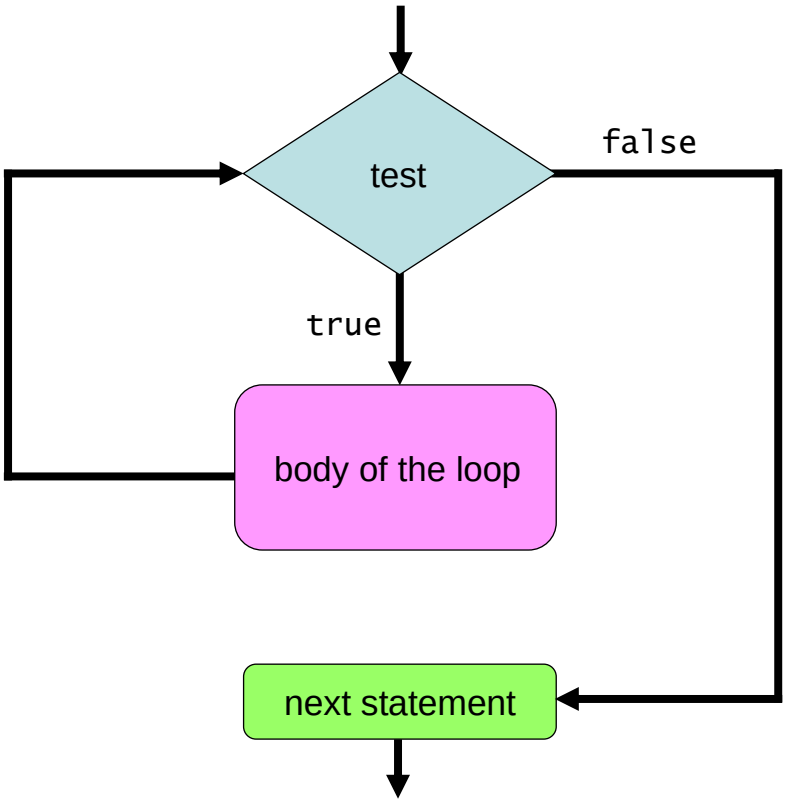
```
Scanner scan = new Scanner(System.in);
System.out.print("Enter a positive integer: ");
int num = console.nextInt();
while (num <= 0) {
    System.out.print("Enter a positive integer: ");
    num = scan.nextInt();
}
```

- Java gives us another option!

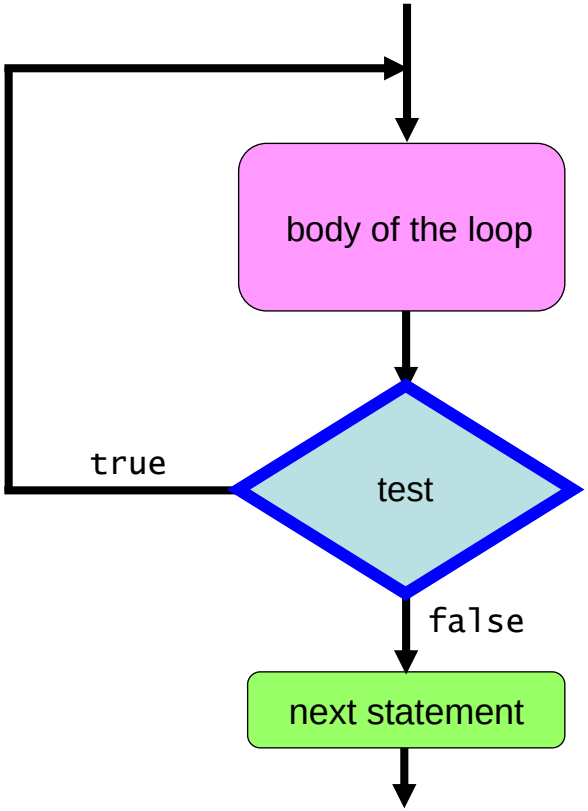
```
Scanner scan = new Scanner(System.in);
int num;
do {
    System.out.print("Enter a positive integer: ");
    num = scan.nextInt();
} while (num <= 0);
```

Comparing `while` and `do-while`

while loop



do-while loop



- In a `do-while`, the first test comes *after* executing the body.
  - thus, the body is always executed *at least once*