Introduction to Programming

Week 3

Magnus Madsen

Week 3: Conditionals and Loops

Monday

- Conditionals
- Loops

Thursday

- For-Loops
- Live programming

Prologue

Quote of the Week

"Typing is no substitute for thinking."

— Richard W. Hamming

Epigram of the Week

"Software and cathedrals are much the same – first we build them, then we pray."

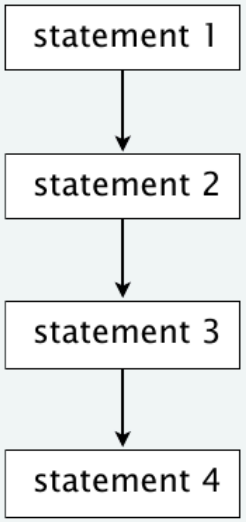
— Sam Redwine

Conditionals

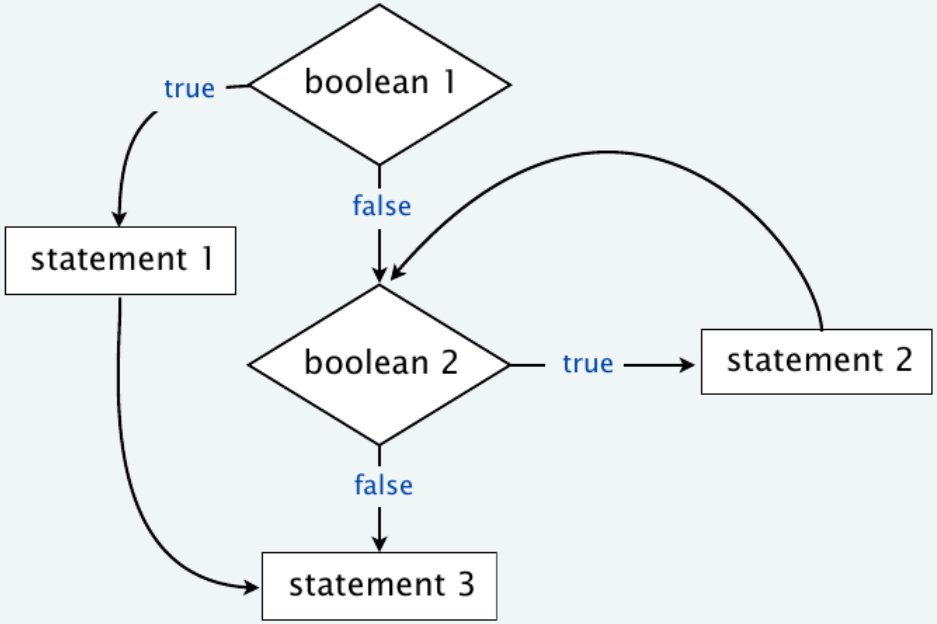
Conditionals and Loops

Control flow

- The sequence of statements that are actually executed in a program.
- Conditionals and loops enable us to choreograph control flow.



straight-line control flow
[previous lecture]



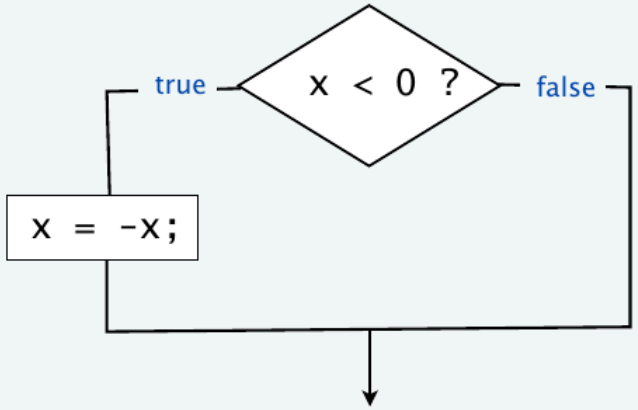
control flow with conditionals and a loop
[this lecture]

The `if` statement

Execute certain statements depending on the values of certain variables.

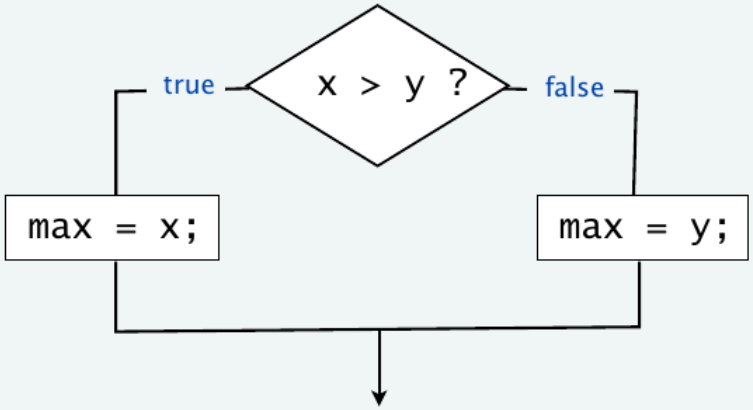
- Evaluate a `boolean` expression.
- If `true`, execute a statement.
- The `else` option: If `false`, execute a different statement.

Example: `if (x < 0) x = -x;`



Replaces x with the absolute value of x

Example:
```
if (x > y) max = x;
else       max = y;
```



Computes the maximum of x and y

Example of if statement use: simulate a coin flip

```
public class Flip
{
   public static void main(String[] args)
   {
      if (Math.random() < 0.5)
           System.out.println("Heads");
      else
           System.out.println("Tails");
   }
}
```

```
% java Flip
Heads

% java Flip
Heads

% java Flip
Tails

% java Flip
Heads
```

Example: Guessing Game (revisited)

We can **reimplement** the Guessing Game from last week:

```
public class GuessRevised {
    public static void main(String[] args) {
        int answer = (int) (Math.random() * 10);
        int guess = Integer.parseInt(args[0]);
        System.out.println("The correct answer was: " + answer);
        if (answer == guess) {
            System.out.println("You won!");
        } else {
            System.out.println("You lost!");
        }
    }
}
```

Example: Drinking age

We can use if-then-else statements to implement an age checker:

```
public class AgeControl {
    public static void main(String[] args) {
        int age = Integer.parseInt(args[0]);
        if (age < 16) {
            System.out.println("You may buy soda.");
        } else if (age < 18) {
            System.out.println("You may buy beer.");
        } else {
            System.out.println("You may buy liquor.");
        }
    }
}
```

Age limits	
Beer	16 years
Liquor	18 years

Example: Tax brackets

We can use if-else-statements to compute taxes at different rates:

```java
public class TaxCalculator {
    public static void main(String[] args) {
        int income = Integer.parseInt(args[0]);
        if (income < 100) {
            System.out.println("Tax: " + income * 0.10);
        } else if (income < 400) {
            System.out.println("Tax: " + income * 0.20);
        } else {
            System.out.println("Tax: " + income * 0.50);
        }
    }
}
```

Tax brackets	
0–99	10%
100–399	20%
400+	50%

Note: This is not how tax brackets are actually calculated.

Example: Greeting Dracula

We can use if-then-else statements to guard against 🧛:

```
public class FrontDoor {
    public static void main(String[] args) {
        String guest = args[0];
        if (!guest.equals("Dracula")) {
            System.out.println("Please come in " + guest + "!");
        } else {
            System.out.println("You are not welcome here.");
        }
    }
}
```

Note: The negation can be avoided by swapping the branches.

Example: Greeting Dracula

Example: Division by zero check

We can use an if-then-else statement to check for division by zero.

```
if (den == 0) {
    System.out.println("Division by zero");
} else {
    System.out.println("Quotient = " + num / den);
}
```

If the denominator is zero, print an error. Otherwise print the quotient.

Example: Negative root check

We can use an if-then-else statement to check for square-root of a negative value.

```
double discriminant = b * b - 4.0 * c;
if (discriminant < 0.0) {
    System.out.println("No real roots");
} else {
    System.out.println((-b + Math.sqrt(discriminant)) / 2.0);
    System.out.println((-b - Math.sqrt(discriminant)) / 2.0);
}
```

If the discriminant is negative, print an error. Otherwise print the solutions.

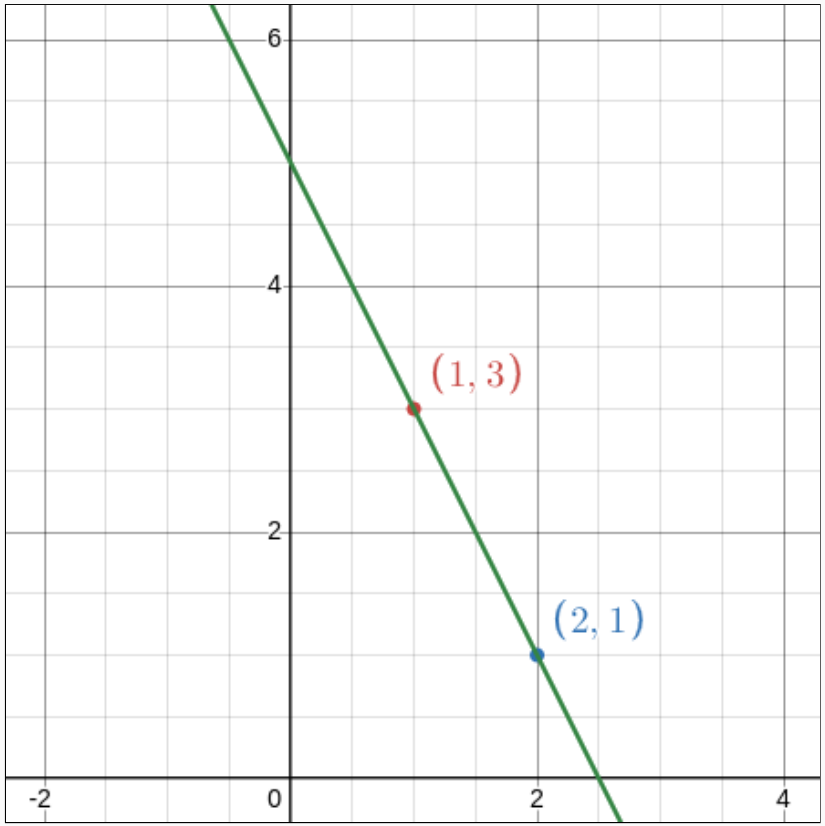
Example: Infinite slope

We often use if-statements to handle **special cases**.

Here we rule out vertical slopes, which would result in division by zero in the formula.

```
public class Slope {
    public static void main(String[] args) {
        double x1 = Double.parseDouble(args[0]);
        double y1 = Double.parseDouble(args[1]);
        double x2 = Double.parseDouble(args[2]);
        double y2 = Double.parseDouble(args[3]);

        if (x1 == x2) {
            System.out.println("vertical");
        } else {
            double slope = (y2 - y1) / (x2 - x1);
            System.out.println(slope);
        }
    }
}
```



If and if-else

We can either use `if` alone, or we can pair it with an `else`.

If

```
if (a == 0) {
    System.out.println("zero");
}
```

If-Else

```
if (a == 0) {
    System.out.println("zero");
} else {
    System.out.println("nonzero");
}
```

Program equivalence

An interesting phenomenon is that two programs can have *different* source code, yet have the same *meaning* – i.e. have the same runtime behavior.

We say that the two programs are **equivalent**.

Compare:

Checks whether a is zero.

```
if (a == 0) {
    System.out.println("zero");
} else {
    System.out.println("nonzero");
}
```

Checks whether a is non-zero.

```
if (a != 0) {
    System.out.println("nonzero");
} else {
    System.out.println("zero");
}
```

Quiz

Are these two programs equivalent?

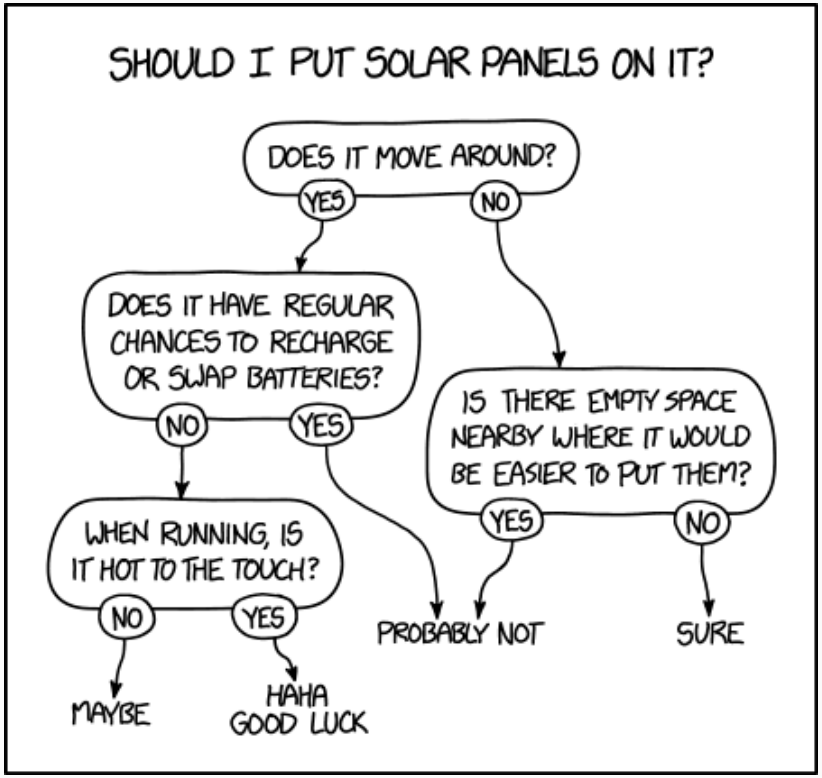
```java
public class ComputeMax1 {
    public static void main(String[] args) {
        int x = Integer.parseInt(args[0]);
        int y = Integer.parseInt(args[1]);
        if (x > y) {
            int z = x;
            System.out.println("max is " + z);
        } else {
            int z = y;
            System.out.println("max is " + z);
        }
    }
}
```

```java
public class ComputeMax2 {
    public static void main(String[] args) {
        int x = Integer.parseInt(args[1]);
        int y = Integer.parseInt(args[0]);
        int max = 42;
        if (x >= y) {
            max = x;
        } else {
            max = y;
        }
        System.out.println("max is " + max);
    }
}
```

Nested ifs

By nesting `if`s, we can create a decision tree.

```
if (movesAround) {
    if (chanceToRecharge) {
        System.out.println("probably not");
    } else {
        if (hotWhenRunning) {
            System.out.println("haha good luck");
        } else {
            System.out.println("maybe");
        }
    }
} else {
    if (emptySpace) {
        System.out.println("probably not");
    } else {
        System.out.println("sure");
    }
}
```



Unnesting ifs

Sometimes nested-ifs are redundant and can be expressed more simply with `&&`.

```
if (pinaColadas && caughtInRain) {
    if (!yoga) {
        if (brain >= 0.5) {
            if (loveAtMidnight) {
                System.out.println("Write to me and escape.");
            }
        }
    }
}
```

is the same as

```
if (pinaColadas && caughtInRain && !yoga && brain >= 0.5 && loveAtMidnight) {
    System.out.println("Write to me and escape.");
}
```

Q:

What does this program print?

```
boolean a = true;
boolean b = false;
boolean c = true;
if (a) {
    if (b) {
        System.out.println("X");
    } else {
        if (c) {
            System.out.println("Y");
        }
    }
    System.out.println("Z");
}
```

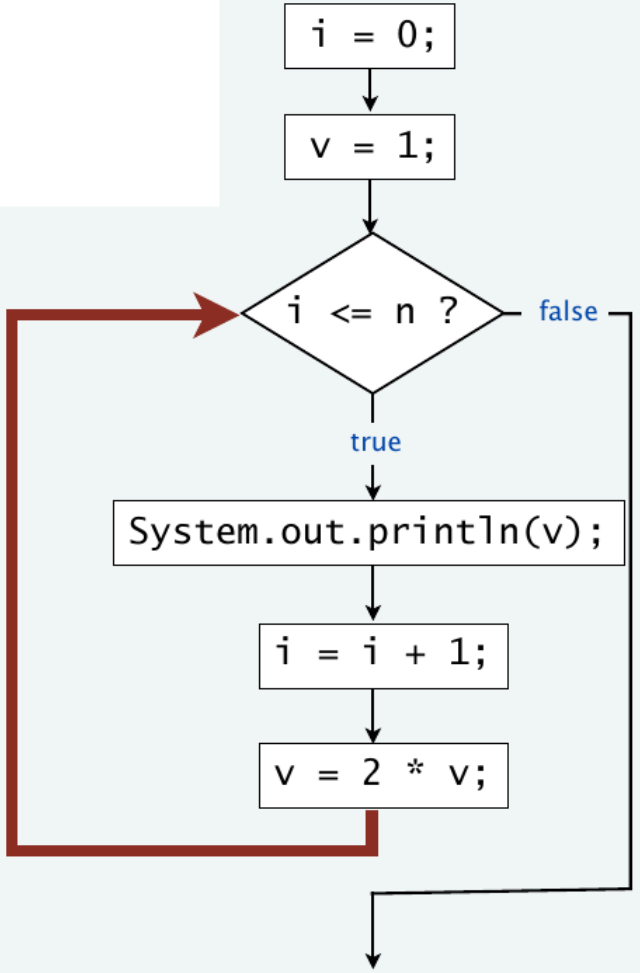
Q: What happens when you evaluate `1.0 / 0`?

Loops

The `while` loop

Execute certain statements repeatedly until certain conditions are met.

- Evaluate a `boolean` expression.
- If `true`, execute a sequence of statements.
- Repeat.



Example:

```
int i = 0;
int v = 1;
while (i <= n)
{
   System.out.println(v);
   i = i + 1;
   v = 2 * v;
}
```

Prints the powers of two from 2^0 to 2^n .

[stay tuned for a trace]

Don’t repeat yourself (DRY)

This program does the same thing many times, printing “Nth Hello”.

```
public class TenHellosLong {
    public static void main(String[] args) {
        System.out.println("1st Hello");
        System.out.println("2nd Hello");
        System.out.println("3rd Hello");
        System.out.println("4th Hello");
        System.out.println("5th Hello");
        System.out.println("6th Hello");
        System.out.println("7th Hello");
        System.out.println("8th Hello");
        System.out.println("9th Hello");
        System.out.println("10th Hello");
    }
}
```

```
$ java TenHellosLong
```

```
1st Hello
2nd Hello
3rd Hello
4th Hello
5th Hello
6th Hello
7th Hello
8th Hello
9th Hello
10th Hello
```

Ten hellos

We can avoid a lot of repetition by using a loop instead.

```
public class TenHellos {
    public static void main(String[] args) {
        System.out.println("1st Hello");
        System.out.println("2nd Hello");
        System.out.println("3rd Hello");
        int i = 4;
        while (i <= 10) {
            System.out.println(i + "th Hello");
            i = i + 1;
        }
    }
}
```

```
$ java TenHellos
```

```
1st Hello
2nd Hello
3rd Hello
4th Hello
5th Hello
6th Hello
7th Hello
8th Hello
9th Hello
10th Hello
```

Example: A While and If

What pattern does this program print?

```
int size = 5;
int row = 0;
while (row < size) {
    int col = 0;
    while (col < size) {
        if ((row + col) % 2 == 0) {
            System.out.print("* ");
        } else {
            System.out.print("  ");
        }
        col++;
    }
    System.out.println();
    row++;
}
```

Example: Answer

And the answer is:

```
int size = 5;
int row = 0;
while (row < size) {
    int col = 0;
    while (col < size) {
        if ((row + col) % 2 == 0) {
            System.out.print("* ");
        } else {
            System.out.print("  ");
        }
        col++;
    }
    System.out.println();
    row++;
}
```

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

Control flow

✏️ **Control flow** is “the order in which individual statements, instructions, or function calls of an imperative program are executed or evaluated.”

via Wiktionary

Recall: Expressions and statements

Remember these two fundamental concepts:

- An **expression** evaluates to a *value*.
- A **statement** executes an *action*.

Expression	Statement
`x + 5`	`int x = 5;`
`Math.random()`	`System.out.println("Hello");`

If, if-else, while

We can describe the **grammar** of if-then-else and while statements as:

If	**if (**<*expression*>**) {** <*statements*> **}**
If-else	**if (**<*expression*>**) {** <*statements*> **} else {** <*statements*> **}**
While	**while (**<*expression*>**) {** <*statements*> **}**

How they work:

- **If**: Execute statements **once** if condition is true.
- **If-else**: Execute **one branch** based on condition (true ⟶ first, false ⟶ second).
- **While**: Execute statements **repeatedly** while condition stays true.

While loop: Terminating

This while loop counts from 0 to 4, then terminates:

```java
public class CountToFive {
    public static void main(String[] args) {
        int i = 0;
        while (i < 5) {
            System.out.println("Count: " + i);
            i++;
        }
        System.out.println("Done!");
    }
}
```

```
$ java CountToFive
```

```
Count: 0
Count: 1
Count: 2
Count: 3
Count: 4
Done!
```

While loop: Non-terminating

This while loop **runs forever** because `i` never changes:

```java
public class CountForever {
    public static void main(String[] args) {
        int i = 0;
        while (i < 5) {
            System.out.println("Count: " + i);
            // Missing: i++;
        }
        System.out.println("Done!");
    }
}
```

```
$ java CountForever
```

```
Count: 0
Count: 0
Count: 0
Count: 0
Count: 0
...
(continues forever)
```

While loop: Immediate termination

This while loop **never executes** because the condition is initially false:

```java
public class NeverRuns {
    public static void main(String[] args) {
        int i = 10;
        while (i < 5) {
            System.out.println("Count: " + i);
            i++;
        }
        System.out.println("Done!");
    }
}
```

```
$ java NeverRuns
```

```
Done!
```

While true

We commonly use `while(true)` to specify an **infinite loop**.

```
public class Forever {
    public static void main(String[] args) {
        int n = 0;
        while (true) {
            System.out.println(n);
            n++;
        }
        System.out.println("Done!");
    }
}
```

```
$ java Forever
```

```
0
1
2
...
1594367
1594368
...
```

Q: Why would we ever want to *deliberately* write an infinite loop?

An ode to an infinite loop

What does this print?

```
while (true) {
    System.out.println("This is the program that never ends.");
    System.out.println("Yes, it goes on and on, my friends.");
    System.out.println("A student started running it, not knowing what it was,");
    System.out.println("and it's still running to this very day because:");
}
```

Escaping infinite loops

We can stop an infinite-looping program *from the outside*.

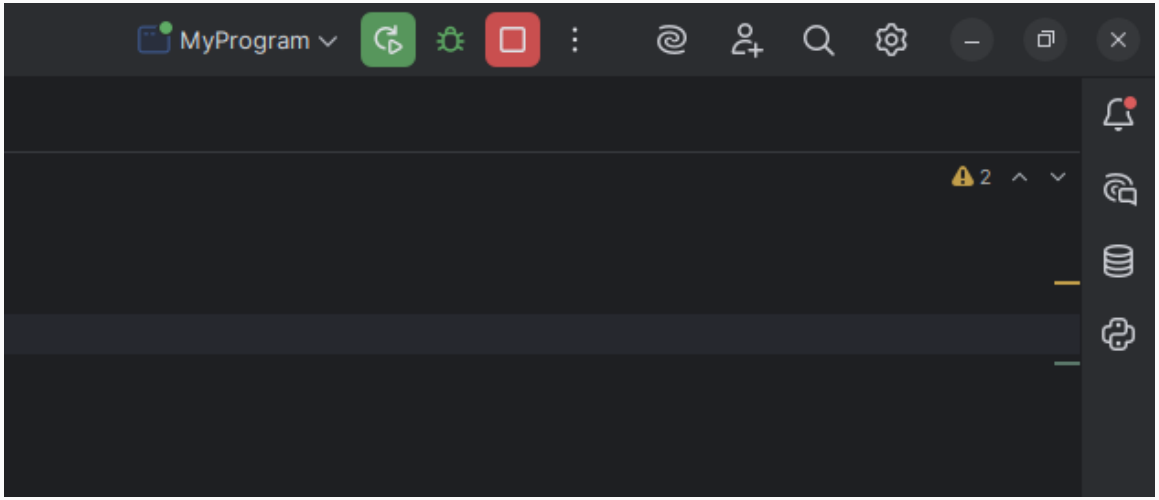
In the terminal,

Pressing `Ctrl-c` can exit most programs.

```
This is the program that never ends.
Yes it goes on and on, my friends.
A student started running it, not knowing what it was,
and it's still running to this very day because:
This is the program that never ends.
Yes it goes on and on, my friends.
A student started running it, not knowing what it was,
and it's still running to this very day because:
This is the program that never ends.
Yes it goes on and on, my friends.
A student started running it, not knowing what it was,
and it's still running to this very day because:
^C
```

In IntelliJ,

The stop button (🟧) ends a program.



The pause button (⏸) triggers the debugger!

Program trace

✏️ A **program trace** is a record of the values of variables and paths taken in a program when it runs.

```
public class Trace {
    public static void main(String[] args) {
        int count = 0;
        double total = 0;
        System.out.println("start loop");          // trace
        while (total < 10) {
            total += Math.random();
            count += 1;
            System.out.println("total: " + total); // trace
            System.out.println("count: " + count); // trace
        }
        System.out.println("end loop");            // trace
        System.out.println(count);
    }
}
```

This program counts how many random numbers (0.0 - 1.0) are needed to add to 10.

We create a trace of the program by adding `println` calls.

Q: Adding print statements to a program to understand its execution is called *print-debugging*.

Is there a better way?

For-loops

What is a `for`-loop?

A **for-loop** is a control-flow construct that repeats a block of statements a **specific number of times**.

- Unlike while-loops, for-loops are designed for **counting-based iteration**.
- They combine **initialization**, **condition checking**, and **increment** in one line.
- Useful when you know **how many times** to repeat something.

A **quality of life** feature.

The for loop

An alternative repetition structure. ← Why? Can provide code that is more compact and understandable.

- Evaluate an *initialization statement*.
- Evaluate a *boolean expression*.
- If true, execute a *sequence of statements*, then execute an *increment statement*.
- Repeat.

Example:

```
int v = 1;
for ( int i = 0; i <= n; i++ )
{
   System.out.println( i + " " + v );
   v = 2*v;
}
```

initialization statement

boolean expression

increment statement

Prints the powers of two from 2^0 to 2^n

Every for loop has an equivalent while loop:

```
int v = 1;
int i = 0;
while ( i <= n; )
{
   System.out.println( i + " " + v );
   v = 2*v;
   i++;
}
```

Why use a `for`-loop?

For-loops *can* make code more **clean** and **readable**, especially when counting:

While Loop

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

For Loop

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

Benefits of for-loops:

- **Shorter code** - all loop control in one line
- **Fewer bugs** - harder to forget the increment
- **Clear intent** - obviously counting-based
- **Local scope** - loop variable `i` is contained within the loop

Downsides of for-loops:

- **Less flexible** - awkward for complex patterns
- **Less natural** for non-counting tasks

Q:

What does this program print?

```
public class Main {
    public static void main(String[] args) {
        for (int i = 1; i <= 3; i++) {
            for (int j = 1; j <= 4; j++) {
                System.out.print(i * j + " ");
            }
            System.out.println();
        }
    }
}
```

The many faces of `for` (1/2)

The many faces of `for` (2/2)

For-loops can take many different forms by varying their three parts:

Standard counting:

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

The many faces of `for` (2/2)

For-loops can take many different forms by varying their three parts:

Standard counting:

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

Counting backwards:

```
for (int i = 10; i > 0; i--) {
    System.out.println(i);
}
```

The many faces of `for` (2/2)

For-loops can take many different forms by varying their three parts:

Standard counting:

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

Counting backwards:

```
for (int i = 10; i > 0; i--) {
    System.out.println(i);
}
```

Skip counting:

```
for (int i = 0; i < 20; i += 2) {
    System.out.println(i);
}
```

The many faces of `for` (2/2)

For-loops can take many different forms by varying their three parts:

Standard counting:

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

Multiple variables:

```
for (int i = 0, j = 10; i < j; i++, j--) {
    System.out.println(i + " " + j);
}
```

Counting backwards:

```
for (int i = 10; i > 0; i--) {
    System.out.println(i);
}
```

Skip counting:

```
for (int i = 0; i < 20; i += 2) {
    System.out.println(i);
}
```

The many faces of `for` (2/2)

For-loops can take many different forms by varying their three parts:

Standard counting:

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

Counting backwards:

```
for (int i = 10; i > 0; i--) {
    System.out.println(i);
}
```

Skip counting:

```
for (int i = 0; i < 20; i += 2) {
    System.out.println(i);
}
```

Multiple variables:

```
for (int i = 0, j = 10; i < j; i++, j--) {
    System.out.println(i + " " + j);
}
```

Using existing variables:

```
int x = 5;
for (; x > 0; x--) {
    System.out.println(x);
}
```

The many faces of `for` (2/2)

For-loops can take many different forms by varying their three parts:

Standard counting:

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

Counting backwards:

```
for (int i = 10; i > 0; i--) {
    System.out.println(i);
}
```

Skip counting:

```
for (int i = 0; i < 20; i += 2) {
    System.out.println(i);
}
```

Multiple variables:

```
for (int i = 0, j = 10; i < j; i++, j--) {
    System.out.println(i + " " + j);
}
```

Using existing variables:

```
int x = 5;
for (; x > 0; x--) {
    System.out.println(x);
}
```

Empty parts (infinite loop):

```
for (;;) {
    System.out.println("Hi!");
}
```

When not to use `for`-loop

Here is a `while`-loop where it would be inelegant to use a `for`-loop:

Using while-loop (elegant):

```
int flips = 0;
while (Math.random() < 0.5) {
    System.out.println("Heads!");
    flips++;
}
System.out.println("Tails!");
System.out.println("Took " + flips + " flips");
```

Using for-loop (awkward):

```
int flips;
for (flips = 0; Math.random() < 0.5; flips++) {
    System.out.println("Heads!");
}
System.out.println("Tails!");
System.out.println("Took " + flips + " flips");
```

The while-loop is more natural because we need to access `flips` **after** the loop ends.

Increment, decrement, and re-assignment

Java has several operators for modifying variables:

Operator	Example	Equivalent to
`++`	`i++`	`i = i + 1`
`--`	`i--`	`i = i - 1`
`+=`	`i += 2`	`i = i + 2`
`-=`	`i -= 3`	`i = i - 3`
`*=`	`i *= 4`	`i = i * 4`
`/=`	`i /= 5`	`i = i / 5`

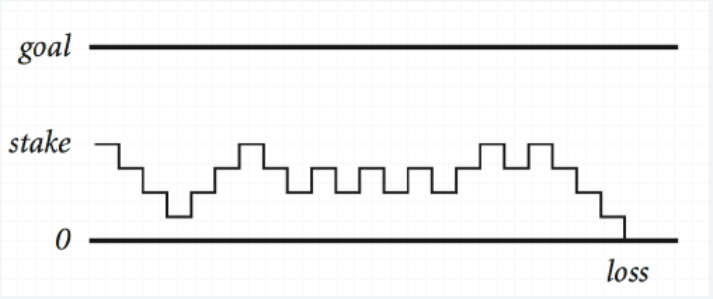
Whether these operators make code easier to read is debatable.

Q:

What does this program print?

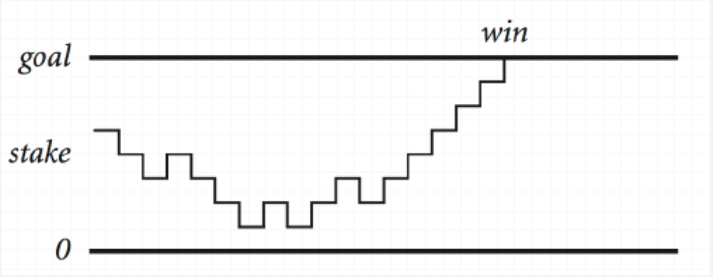
```
public class Main {
    public static void main(String[] args) {
        int x = 5;
        int i = 0;
        for (; i < 3; i++) {
            x += i;
            System.out.print(x + " ");
        }
        x *= 2;
        System.out.println(x);
    }
}
```

Gambler's ruin problem



A gambler starts with $*stake* and places $1 fair bets.

- Outcome 1 (loss): Gambler goes broke with $0.
- Outcome 2 (win): Gambler reaches $*goal*.



Q. What are the chances of winning?

Q. How many bets until win or loss?

One approach: Monte Carlo simulation.

- Use a *simulated coin flip*.
- Repeat and compute statistics.

Example of nesting conditionals and loops: Simulate gamber's ruin

Gambler's ruin simulation

- Get command-line arguments.
- Run all the experiments.
  - Run one experiment.
    - Make one bet.
  - If goal met, count the win.
- Print #wins and # trials.

```
public class Gambler
{
   public static void main(String[] args)
   {
      int stake  = Integer.parseInt(args[0]);
      int goal   = Integer.parseInt(args[1]);
      int trials = Integer.parseInt(args[2]);

      int wins   = 0;
      for (int t = 0; t < trials; t++)
      {
         int cash = stake;
         while (cash > 0 && cash < goal)
         {
            if (Math.random() < 0.5) cash++;
            else                     cash--;
         }
         if (t == goal) wins++;
      }
      System.out.println(wins + " wins of " + trials);
   }
}
```

for loop

while loop within a for loop

if statement within a while loop within a for loop

```
% java Gambler 5 25 1000
191 wins of 1000
```

Digression: simulation and analysis

Facts (known via mathematical analysis for centuries)

- Probability of winning = stake ÷ goal.
- Expected number of bets = stake × desired gain.

Christiaan Huygens
1629-1695

Early scientists were fascinated by the study of games of chance.

Example

- 20% chance of turning $500 into $2500.
- Expect to make 1 *million* $1 bets.

500/2500 = 20%

500*(2500 - 500) = 1,000,000

uses about 1 *billion* coin flips

stake goal trials

```
% java Gambler   5 25 1000
191 wins of 1000

% java Gambler   5 25 1000
203 wins of 1000

% java Gambler 500 2500 1000
197 wins of 1000
```

Remarks

- Computer simulation can help validate mathematical analysis.
- For this problem, mathematical analysis is simpler (if you know the math).
- For more complicated variants, computer simulation may be the *best* plan of attack.

What does this program print?

Q:

```
int height = 6;
for (int i = 1; i <= height; i++) {
    for (int j = 1; j <= height - i; j++) {
        System.out.print(" ");
    }
    for (int j = 1; j <= (2 * i - 1); j++) {
        System.out.print("*");
    }
    System.out.println();
}

for (int i = 1; i <= 2; i++) {
    for (int j = 1; j <= height - 2; j++) {
        System.out.print(" ");
    }
    System.out.println("|||");
}
```

And the answer is:

```
int height = 6;
for (int i = 1; i <= height; i++) {
    for (int j = 1; j <= height - i; j++) {
        System.out.print(" ");
    }
    for (int j = 1; j <= (2 * i - 1); j++) {
        System.out.print("*");
    }
    System.out.println();
}

for (int i = 1; i <= 2; i++) {
    for (int j = 1; j <= height - 2; j++) {
        System.out.print(" ");
    }
    System.out.println("|||");
}
```

```
     *
    ***
   *****
  *******
 *********
***********
    |||
    |||
```

For vs. While

For vs. while (1/2)

These two programs both perform a countdown to rocket liftoff:

While Loop:

```
public class CountdownWhile {
    public static void main(String[] args) {
        int count = 10;
        while (count > 0) {
            System.out.print(count + "...");
            count--;
        }
        System.out.println("Liftoff! 🚀");
    }
}
```

For Loop:

```
public class CountdownFor {
    public static void main(String[] args) {
        for (int count = 10; count > 0; count--) {
            System.out.print(count + "...");
        }
        System.out.println("Liftoff! 🚀");
    }
}
```

Both programs output:

```
10...  9...  8...  (...)  2...  1...  Liftoff! 🚀
```

For vs. while (2/2)

For-loops and while-loops are *equivalent*.

Every for-loop can be rewritten as a while-loop and vice-versa.

Program	**For-loop**	**While-loop**

Print 0 - 9	`for (int i = 0; i < 10; i++) {`
	`  System.out.println(i);`
	`}`
		`int i = 0;`
		`while (i < 10) {`
		`  System.out.println(i);`
		`  i++;`
		`}`

Count flips until tails	`int c = 0;`
	`for (double n = 0.0; n < 0.5; n = Math.random()) {`
	`  c++;`
	`}`
	`System.out.println(c);`
		`double n = 0.0;`
		`int c = 0;`
		`while (n < 0.5) {`
		`  n = Math.random();`
		`  c++;`
		`}`
		`System.out.println(c);`

Rewrite the program below to use for-loops:

Q:

```
public class Main {
    public static void main(String[] args) {
        int i = 1;
        int j = 1;
        while (i <= 3) {
            j = 1;
            while (j <= i) {
                System.out.print("*");
                j++;
                if (j > i) {
                    i++;
                }
            }
            System.out.println();
        }
    }
}
```

Scope

What is scoping?

The **scope** of a variable is the region where it can be accessed.

- Variables are created when they are **declared**.
- Variables are destroyed when their **block ends**.
- A block is delimited by curly braces: `{ }`.
- Variables declared inside a block are **not accessible** outside of the block.

Example: Scope (1/2)

We can use variables in scope, but not variables out of scope:

```
int x = 1;
if (true) {
    int y = 3;
}
System.out.println(x);
```

OK: `x` is in scope

```
int x = 1;
if (true) {
    int y = 3;
    System.out.println(y);
}
```

OK: `y` is in scope

```
int x = 1;
if (true) {
    int y = 3;
}
System.out.println(y);
```

ERROR: `y` is out of scope

Example: Scope (2/2)

Variables have different scopes in different blocks:

```
public class Main {
    public static void main(String[] args) {
        int a = 1;                    // Scope: entire main method
        if (a > 0) {
            int b = 2;                // Scope: if block only
            System.out.println(b);
        }
        // System.out.println(b);     // ERROR: b is out of scope
        for (int i = 0, j = 10; i < j; i++, j--) { // i and j scope: for block only
            int sum = i + j;          // sum scope: for block only
            System.out.println(sum);  // OK: sum in scope
        }
        // System.out.println(i);     // ERROR: i is out of scope
        // System.out.println(j);     // ERROR: j is out of scope
        System.out.println(a);        // OK: a is still in scope
    }
}
```

Prologue

Compiler error of the week

What is the problem here?

```
int sum = 0;
int i = 0;
while (i < 10){
    int tmp = 0;
    if (i % 2 == 0){
    for (int j = 0; j < 10; j += 2){
    tmp -= j;}} else {
    for (int j = 0; j < 10; j += 2){
    tmp += j; }}
    sum += tmp * j; i++; }
```

Compiler error of the week

What is the problem here?

```
int sum = 0;
int i = 0;
while (i < 10){
    int tmp = 0;
    if (i % 2 == 0){
    for (int j = 0; j < 10; j += 2){
    tmp -= j;}} else {
    for (int j = 0; j < 10; j += 2){
    tmp += j; }}
    sum += tmp * j; i++; }
```

```
java: cannot find symbol
  symbol:   variable j
```

Live Programming

Live Programming

- An equivalent while and for-loop.
- Using a loop as an if-statement.
- Flattening nested-ifs.
- Flattening nested-loops.
- Debugging infinite loops.
- Simulate Gambler's Ruin

Sources for slides and images

- https://introcs.cs.princeton.edu/java/lectures/
- https://xkcd.com/1924/
- https://commons.wikimedia.org/wiki/File:Moai_Rano_raraku.jpg