Week01

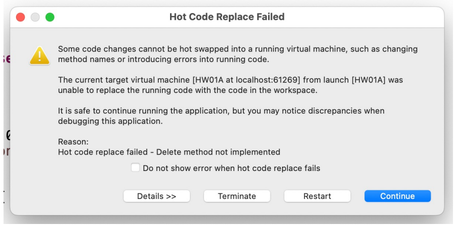
- tips and tricks
- operators
- control flow
- Cow.Java (graphics, UI)

NOTE: squack!

tips and tricks

"hot code replace failed" Eclipse message

if you've been getting this error a bunch...



Hot Code Replace Failed

Some code changes cannot be hot swapped into a running virtual machine, such as changing method names or introducing errors into running code.

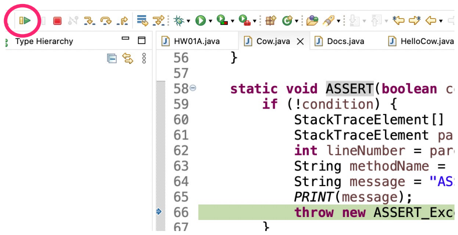
The current target virtual machine [HW01A at localhost:61269] from launch [HW01A] was unable to replace the running code with the code in the workspace.

It is safe to continue running the application, but you may notice discrepancies when debugging this application.

Reason:
Hot code replace failed - Delete method not implemented

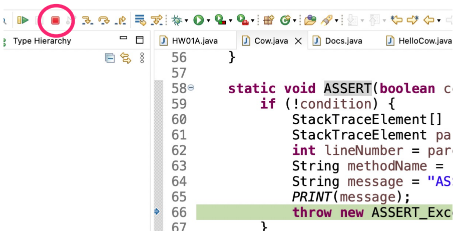
□ Do not show error when hot code replace fails

make sure you are pressing **this button** after every crash / failed assert (otherwise Eclipse keeps your program running, and will try to "hot swap" your changes into it)



```
    }

    static void ASSERT(boolean c
        if (!condition) {
            StackTraceElement[]
            StackTraceElement pa
            int lineNumber = par
            String methodName =
            String message = "AS
            PRINT(message);
            throw new ASSERT_Exc
        }
```

you can also press **this button**
(which, for us, does basically the same thing)



```
    }

    static void ASSERT(boolean c
        if (!condition) {
            StackTraceElement[]
            StackTraceElement pa
            int lineNumber = par
            String methodName =
            String message = "AS
            PRINT(message);
            throw new ASSERT_Exc
        }
```

you can also just close Eclipse and reopen it if anything really weird happens
🤷

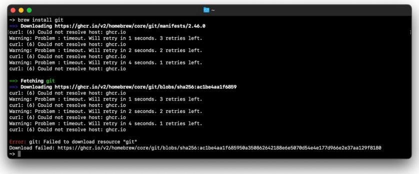
how to email jim

first, how to email not-jim

- **Subject Line:** Question Regarding The Reading
- **Email Body:**
  - Dear Prof. Dr. Professorson Ph.D Exquire,
  - I hope you are having a wonderful afternoon; that the morning sun is shining upon you this fine evening, and that you do not feel the quiet whisper of the all-consuming Void chilling you to your very core.
  - I am writing because I am unclear on the assigned reading for tomorrow. Your very, very helpful syllabus says to read pages 300 – 500 of Beowulf, however the book is, it seems, much fewer than 500 pages long.
  - Thank you so much for your help with this matter. I appreciate the time you take in supporting me in my studies—I could not do this without you.
  - Sincerely,
  - James Bern
  - Caltech '15

how to email jim

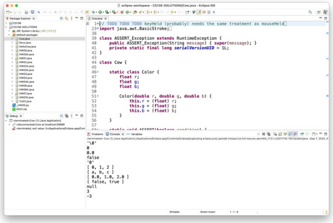
- **Reply to Q + A Thread**
- **Email Body:** brew won't install 😡



```
~> brew install git
==> Downloading https://ghcr.io/v2/homebrew/core/git/manifests/2.46.0
curl: (6) Could not resolve host: ghcr.io
Warning: Problem : timeout. Will retry in 1 seconds. 3 retries left.
curl: (6) Could not resolve host: ghcr.io
Warning: Problem : timeout. Will retry in 2 seconds. 2 retries left.
curl: (6) Could not resolve host: ghcr.io
Warning: Problem : timeout. Will retry in 4 seconds. 1 retries left.
curl: (6) Could not resolve host: ghcr.io

==> Fetching git
==> Downloading https://ghcr.io/v2/homebrew/core/git/blobs/sha256:ac1be4aa1f6859
curl: (6) Could not resolve host: ghcr.io
Warning: Problem : timeout. Will retry in 1 seconds. 3 retries left.
curl: (6) Could not resolve host: ghcr.io
Warning: Problem : timeout. Will retry in 2 seconds. 2 retries left.
curl: (6) Could not resolve host: ghcr.io
Warning: Problem : timeout. Will retry in 4 seconds. 1 retries left.
curl: (6) Could not resolve host: ghcr.io

Error: git: Failed to download resource "git"
Download failed: https://ghcr.io/v2/homebrew/core/git/blobs/sha256:ac1be4aa1f685950a350862642188e6e5070d54e4e177d966e2e37aa129f8180
~>
```

how to email jim

- **Reply to Q + A Thread**
- **Email Body:** can't run tic tac toe 😢



```java
// TODO TODO TODO keyHeld (probably) needs the same treatment as mouseHeld
import java.awt.BasicStroke;

class ASSERT_Exception extends RuntimeException {
    public ASSERT_Exception(String message) { super(message); }
    private static final long serialVersionUID = 1L;
}

class Cow {

    static class Color {
        float r;
        float g;
        float b;

        Color(double r, double g, double b) {
            this.r = (float) r;
            this.g = (float) g;
            this.b = (float) b;
        }
    }

    static void ASSERT(boolean condition) {
```

```
<terminated> Cow (1) [Java Application] /Applications/Eclipse.app/Contents/Eclipse/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.macosx.aarch64_17.0.1.v20211116-1657/jre/bin/java (Sep 7, 2024, 4
'\0'
0
0.0
false
'0'
[ 0, 1, 2 ]
[ a, b, c ]
[ 0.0, 1.0, 2.0 ]
[ false, true ]
null
3
-3
```

this is the fastest way to get your question answered

i will not be offended i promise

how to do HW

implement one thing at a time

get it working perfectly

then, try the next thing

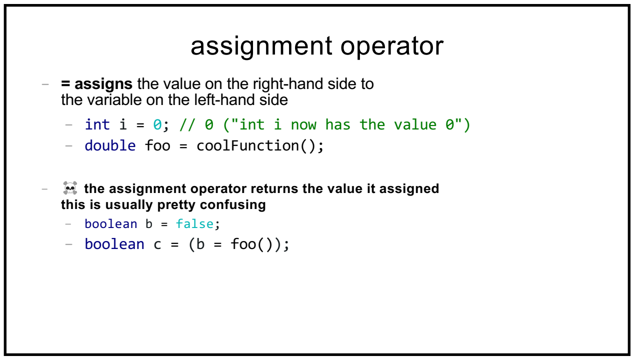
code up the entire homework without ever trying to run it

compile early; compile often

operators

(except for bitwise operators, which we'll do later maybe)

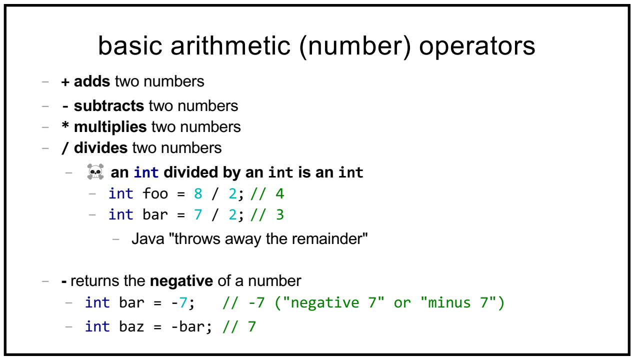
assignment operator



assignment operator

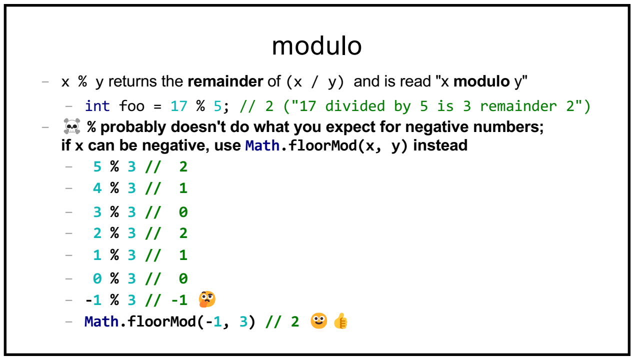
- **= assigns** the value on the right-hand side to the variable on the left-hand side
  - `int i = 0; // 0 ("int i now has the value 0")`
  - `double foo = coolFunction();`
- ☠️ **the assignment operator returns the value it assigned**
  this is usually pretty confusing
  - `boolean b = false;`
  - `boolean c = (b = foo());`

arithmetic operators



basic arithmetic (number) operators

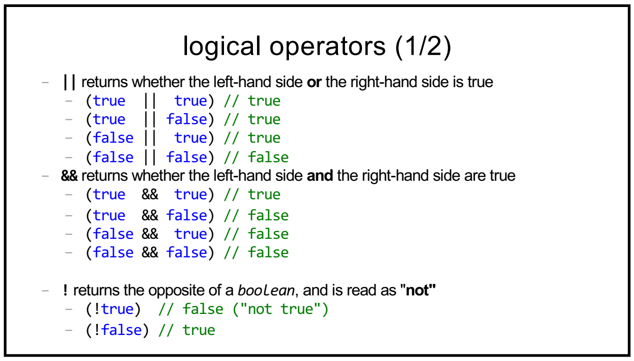
- **+ adds** two numbers
- **- subtracts** two numbers
- *** multiplies** two numbers
- **/ divides** two numbers
  - ☠️ **an `int` divided by an `int` is an `int`**
    - `int foo = 8 / 2; // 4`
    - `int bar = 7 / 2; // 3`
      - Java "throws away the remainder"
- **-** returns the **negative** of a number
  - `int bar = -7;   // -7 ("negative 7" or "minus 7")`
  - `int baz = -bar; // 7`



modulo

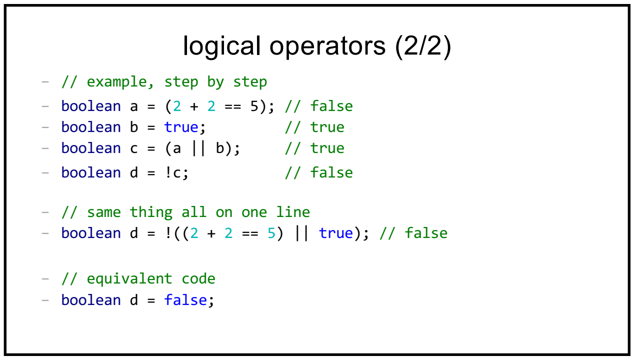
- `x % y` returns the **remainder** of `(x / y)` and is read "x **modulo** y"
  - `int foo = 17 % 5; // 2 ("17 divided by 5 is 3 remainder 2")`
- ☠️ **% probably doesn't do what you expect for negative numbers;**
  if x can be negative, use `Math.floorMod(x, y)` instead
  - ` 5 % 3 //  2`
  - ` 4 % 3 //  1`
  - ` 3 % 3 //  0`
  - ` 2 % 3 //  2`
  - ` 1 % 3 //  1`
  - ` 0 % 3 //  0`
  - `-1 % 3 // -1` 🤔
  - `Math.floorMod(-1, 3) // 2` 🙂👍

logical operators



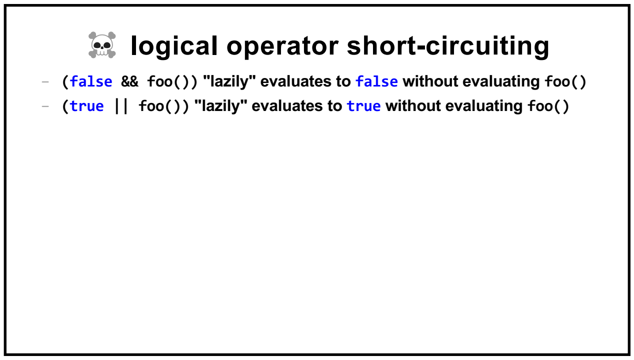
logical operators (1/2)

- **||** returns whether the left-hand side **or** the right-hand side is true
  - `(true  ||  true) // true`
  - `(true  || false) // true`
  - `(false ||  true) // true`
  - `(false || false) // false`
- **&&** returns whether the left-hand side **and** the right-hand side are true
  - `(true  &&  true) // true`
  - `(true  && false) // false`
  - `(false &&  true) // false`
  - `(false && false) // false`
- **!** returns the opposite of a *boolean*, and is read as **"not"**
  - `(!true)  // false ("not true")`
  - `(!false) // true`



logical operators (2/2)

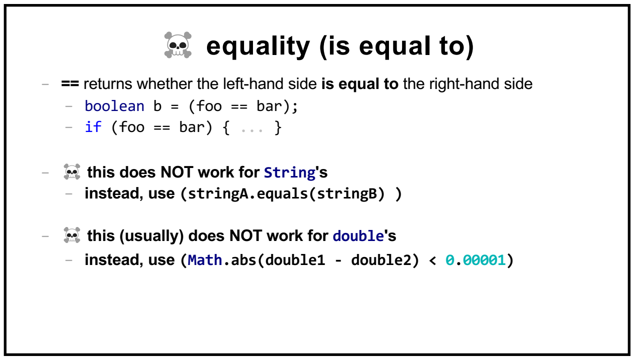
```
// example, step by step
boolean a = (2 + 2 == 5); // false
boolean b = true;         // true
boolean c = (a || b);     // true
boolean d = !c;           // false

// same thing all on one line
boolean d = !((2 + 2 == 5) || true); // false

// equivalent code
boolean d = false;
```



☠️ logical operator short-circuiting

- **(`false` && `foo()`) "lazily" evaluates to `false` without evaluating `foo()`**
- **(`true` || `foo()`) "lazily" evaluates to `true` without evaluating `foo()`**

comparison operators

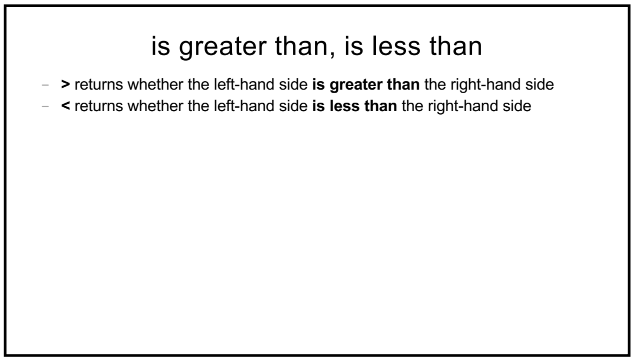


☠️ equality (is equal to)

- **==** returns whether the left-hand side **is equal to** the right-hand side
  - `boolean b = (foo == bar);`
  - `if (foo == bar) { ... }`

- ☠️ **this does NOT work for `String`'s**
  - **instead, use (`stringA.equals(stringB)`)**

- ☠️ **this (usually) does NOT work for `double`'s**
  - **instead, use (`Math.abs(double1 - double2) < 0.00001`)**



is greater than, is less than

- **>** returns whether the left-hand side **is greater than** the right-hand side
- **<** returns whether the left-hand side **is less than** the right-hand side

convenient operators

(feel free to ignore these for now)

inequality

- **!=** returns whether the left-hand side **is not equal to** the right-hand side
  - `(left != right)` is exactly the same as `(!(left == right))`

greater than or equal to, less than or equal to

- **>=** returns whether the left-hand side **is greater than or equal to** the right-hand side
  - `(left >= right)` is basically the same as

```
((left > right) || (left == right))
 greater-than   or      equal
```

- **<=** returns whether the left-hand side **is less than or equal to** the right-hand side

arithmetic assignment operators

```
– a += b; // a = a + b;
– a -= b; // a = a - b;
– a *= b; // a = a * b;
– a /= b; // a = a / b;
```

String concatenation

- + concatenates two `String`'s

```
– // String foo = "Hello".concat("World");
     String foo = "Hello" + "World"; // "HelloWorld"
```

- it also does some conversions for you
  (very convenient, kind of confusing)

```
– // String foo = "Hello".concat(Integer.toString(2));
     String foo = "Hello" + 2; // "Hello2"
```

increment operator

- to "**increment**" means to increase the value of a number by one

```
– i = i + 1;
– i += 1;
```

- the **pre-increment** `++i` increments `i` and returns the new value of `i`

```
– j = ++i; // i = i + 1;
–          // j = i;
```

- the **post-increment** `i++` increments `i` and returns the old value of `i`

```
– j = i++; // j = i;
–          // i = i + 1;
```

decrement operator

- to "**decrement**" means to decrease the value of a number by one

```
– i = i - 1;
– i -= 1;
```

- the **pre-decrement** `--i` decrements `i` and returns the new value of `i`

```
– j = --i; // i = i - 1;
–          // j = i;
```

- the **post-decrement** `i--` decrements `i` and returns the old value of `i`

```
– j = i--; // j = i;
–          // i = i - 1;
```

control flow

the execution of a Java program starts at the top of `main(...)` and flows down down down

let's step through this program in our minds, and then in a debugger

```
class Main {
    public static void main(String[] arguments) {
        double a = 3.0;
        double b = 4.0;

        double result = Math.sqrt(a * a + b * b);
        System.out.println(result);
    }
}
```

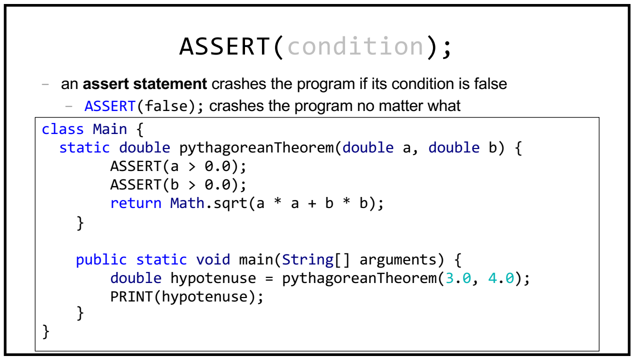
functions (preview)

functions (preview)

- when a **function** is called, control flow jumps to the top of the function, flows down through it, and then jumps back to right after the function call

```
class Main {
    static double pythagoreanTheorem(double a, double b) {
        return Math.sqrt(a * a + b * b);
    }

    public static void main(String[] arguments) {
        double hypotenuse = pythagoreanTheorem(3.0, 4.0);
        System.out.println(hypotenuse);
    }
}
```

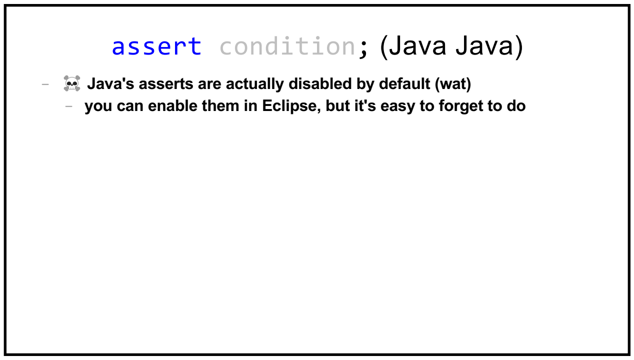
ASSERT



ASSERT(condition);

- an **assert statement** crashes the program if its condition is false
  - `ASSERT(false);` crashes the program no matter what

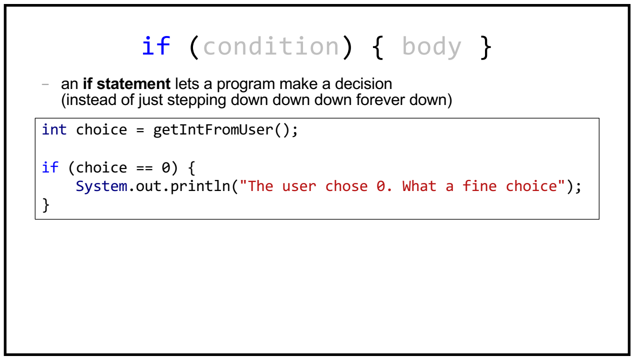
```
class Main {
  static double pythagoreanTheorem(double a, double b) {
        ASSERT(a > 0.0);
        ASSERT(b > 0.0);
        return Math.sqrt(a * a + b * b);
    }

    public static void main(String[] arguments) {
        double hypotenuse = pythagoreanTheorem(3.0, 4.0);
        PRINT(hypotenuse);
    }
}
```



assert condition; (Java Java)

- ☠ **Java's asserts are actually disabled by default (wat)**
  - **you can enable them in Eclipse, but it's easy to forget to do**

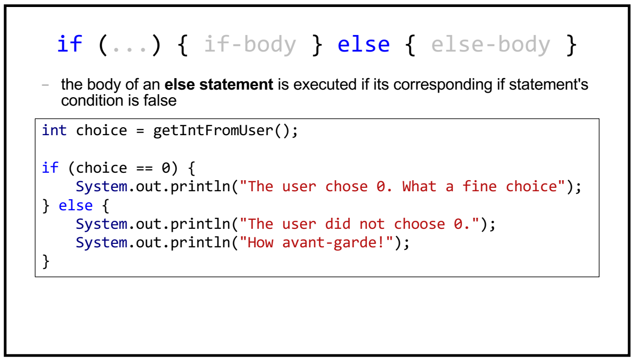
if and else



if (condition) { body }

- an **if statement** lets a program make a decision
  (instead of just stepping down down down forever down)

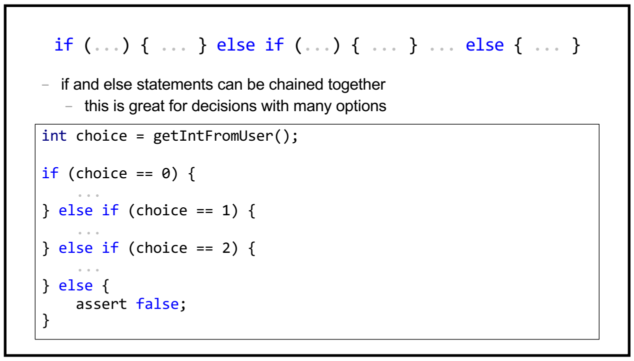
```
int choice = getIntFromUser();

if (choice == 0) {
    System.out.println("The user chose 0. What a fine choice");
}
```



if (...) { if-body } else { else-body }

- the body of an **else statement** is executed if its corresponding if statement's condition is false

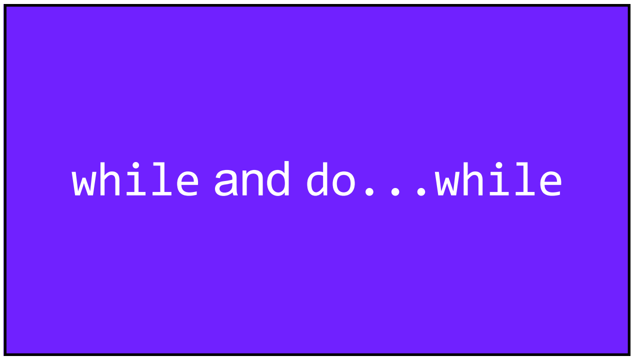
```
int choice = getIntFromUser();

if (choice == 0) {
    System.out.println("The user chose 0. What a fine choice");
} else {
    System.out.println("The user did not choose 0.");
    System.out.println("How avant-garde!");
}
```



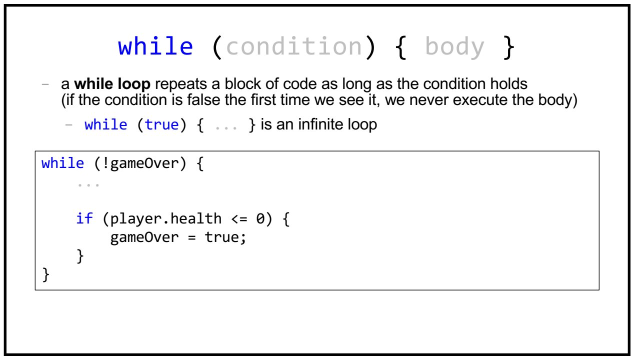
if (...) { ... } else if (...) { ... } ... else { ... }

- if and else statements can be chained together
  - this is great for decisions with many options

```
int choice = getIntFromUser();

if (choice == 0) {
    ...
} else if (choice == 1) {
    ...
} else if (choice == 2) {
    ...
} else {
    assert false;
}
```



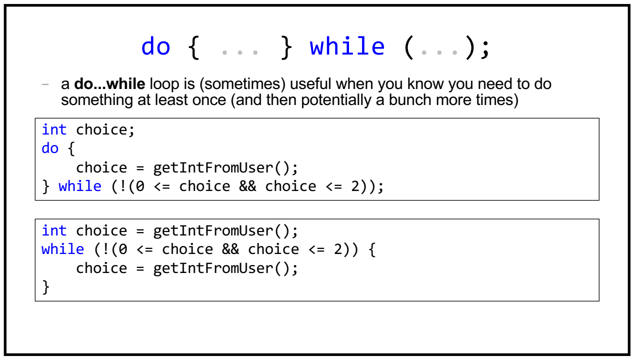
while and do...while



while (condition) { body }

- a **while loop** repeats a block of code as long as the condition holds (if the condition is false the first time we see it, we never execute the body)
  - while (true) { ... } is an infinite loop

```
while (!gameOver) {
    ...

    if (player.health <= 0) {
        gameOver = true;
    }
}
```

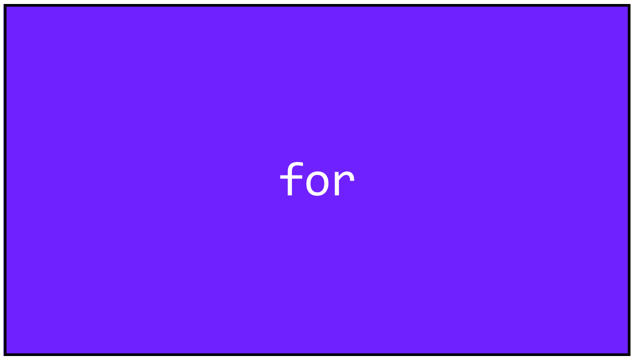


do { ... } while (...);

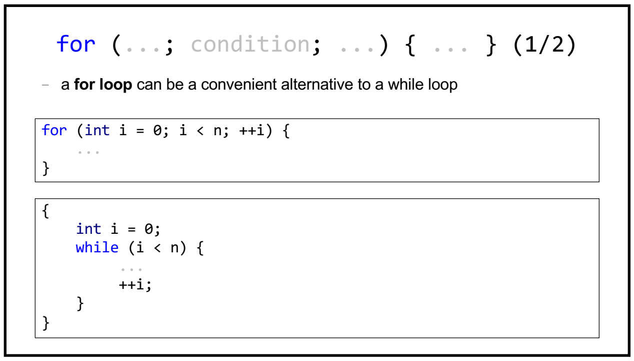
- a **do...while** loop is (sometimes) useful when you know you need to do something at least once (and then potentially a bunch more times)

```
int choice;
do {
    choice = getIntFromUser();
} while (!(0 <= choice && choice <= 2));
```

```
int choice = getIntFromUser();
while (!(0 <= choice && choice <= 2)) {
    choice = getIntFromUser();
}
```



for

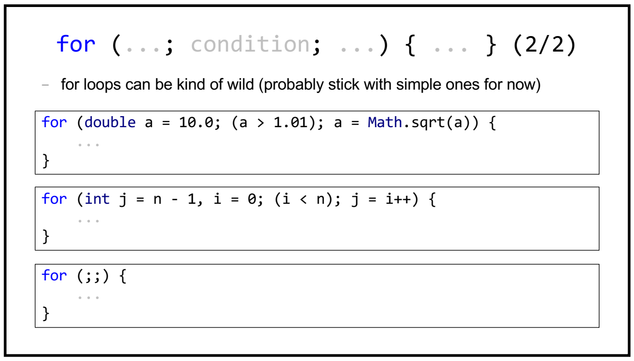


for (...; condition; ...) { ... } (1/2)

- a **for loop** can be a convenient alternative to a while loop

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

```
{
    int i = 0;
    while (i < n) {
        ...
        ++i;
    }
}
```



for (...; condition; ...) { ... } (2/2)

- for loops can be kind of wild (probably stick with simple ones for now)

```
for (double a = 10.0; (a > 1.01); a = Math.sqrt(a)) {
    ...
}
```

```
for (int j = n - 1, i = 0; (i < n); j = i++) {
    ...
}
```

```
for (;;) {
    ...
}
```

break and continue

break

- **break** breaks out of a loop

```
while (beginFrame()) {
    ...

    if (player.health <= 0) {
        break;
    }
}
```

continue

- **continue** continues to the next iteration of a loop

```
for (int i = 0; i < slots.length; ++i) {
    if (!slots[i].isOccupied) {
        continue;
    }

    ... // do something with whatever is in the i-th slot
}
```

exceptions to the rule that "scope is the same as curly braces"

for (...; ...; ...) { ... }

- variables declared inside the parentheses of a **for loop** are not available outside of the for loop (this is probably the behavior you already expected)

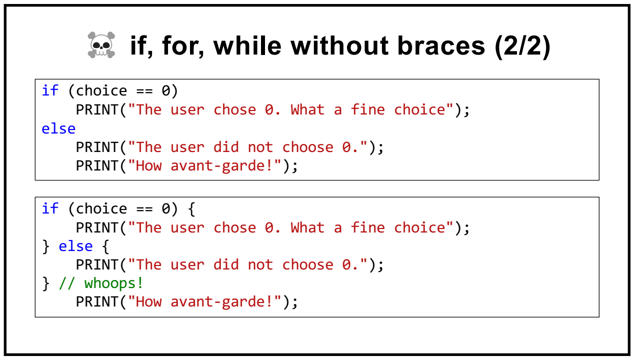
```
Compile Error: cannot find symbol i

class Main {
    public static void main(String[] arguments) {
        for (int i = 0; i < 10; ++i) {
            ...
        }
        PRINT(i);
    }
}
```

☠ if, else, for, while without braces (1/2)

- **☠ if you (intentionally or unintentionally) forget your curly braces, then Java will assume you wanted them go around the first statement after the `if (...)`, else, `for (...)`, or `while (...)`**
  - **in this class, i highly recommend always using curly braces**

```
if (choice == 0)
    System.out.println("The user chose 0. What a fine choice");
```

```
if (choice == 0) {
    System.out.println("The user chose 0. What a fine choice");
}
```



☠ if, for, while without braces (2/2)

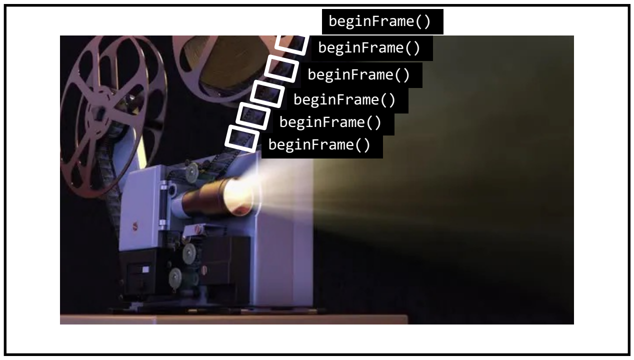
```
if (choice == 0)
    PRINT("The user chose 0. What a fine choice");
else
    PRINT("The user did not choose 0.");
    PRINT("How avant-garde!");
```

```
if (choice == 0) {
    PRINT("The user chose 0. What a fine choice");
} else {
    PRINT("The user did not choose 0.");
} // whoops!
    PRINT("How avant-garde!");
```

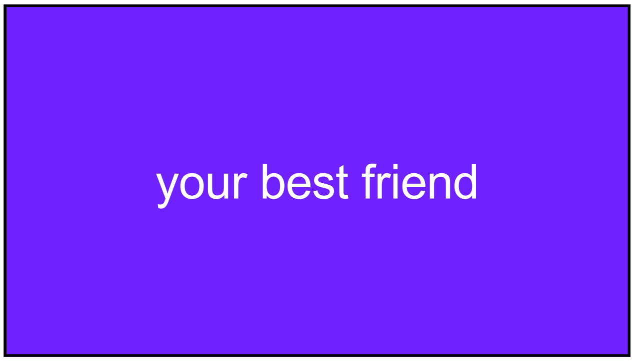
Cow.Java

overview

during a **frame** (like a frame of a movie)
we **update** and **draw** the world



```
beginFrame()
beginFrame()
beginFrame()
beginFrame()
beginFrame()
beginFrame()
```



your best friend