Week13a

-OOP

WARMUP

what is object-oriented programming?

are *you* an object-oriented programmer?

do you know *anything* about hackers?

can you jam with the console cowboys in cyberspace?!

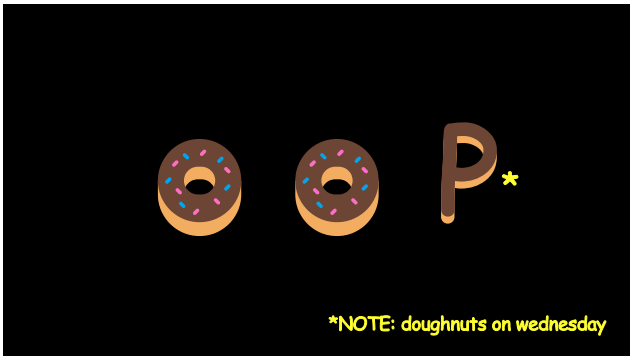
ANNOUNCEMENTS

doughnuts on wednesday

1

PBS's Ghostwriter. Season 2 Episode 5

2



3

indiana, let it go

4

5

[record lecture]

6

object
(instance of a **class**)

7

anatomy of a class (1/2)

```
class ClassName {
    VariableOneType variableOne;
    ...

    FunctionOneReturnType functionOneName(...) { ... }
    ...
}
```

- a **class** is (a blueprint for) a lil chunk of data that you can make elsewhere
  - a class may have any number of **variables** (fields)
    - `int foo; // objects of this class have an int called foo`
  - a class may have any number of **functions** (methods)
    - `int bar() { ... } // objects of class have function bar`

8

anatomy of a class (2/2)

```
class Point {
    // instance variables
    double x;
    double y;

    // constructor
    Point(double x, double y) { ... }

    // instance methods
    double distanceTo(Point otherPoint) { ... }
    ...
}
```

9

an **object** is **an instance of a class**

```
// p is a reference to an instance of the Point class
// p is an instance of the Point class
// p is a Point object
// "p is a Point"
Point p = new Point();
```

10

object-oriented programming (OOP)

note: jim maybe has opinions but who's to sayyy

11

object-oriented programming (OOP)

- **object-oriented programming** means *thinking* in terms of nouns
  - "how can i break down this problem into classes/objects?"

12

object-oriented programming (OOP)

- **object-oriented programming** is NOT just *having* classes/objects
  - recall, a **class** is just (a blueprint for) a lil chunk of data
  - rather, OOP means my problem-solving is *oriented* around objects
    - ...instead of, for example, data (**data-oriented design**)
    - ...functions (**functional programming**)

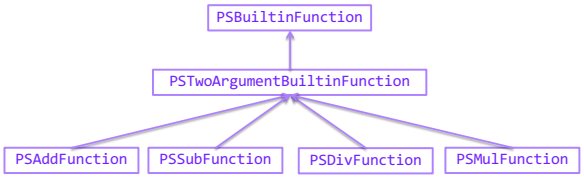
13

object-oriented programming

- **example:** to implement an Object-Oriented PostScript interpreter...
  - `class PSInterpreter`
  - `class PSProgram`
  - `class PSStack`
  - `class PSMap`
  - `class PSBuiltinFunction`
  - `class PSTwoArgumentBuiltinFunction extends PSBuiltinFunction`
  - `class PSAddFunction extends PSTwoArgumentBuiltinFunction`
  - `class PSSubFunction extends PSTwoArgumentBuiltinFunction`
  - `class PSMulFunction extends PSTwoArgumentBuiltinFunction`
  - `class PSDivFunction extends PSTwoArgumentBuiltinFunction`
  - `...`

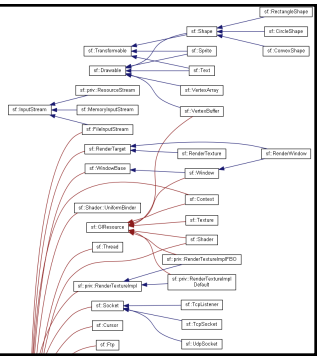
14

unified modeling language (UML) diagram



15

real-world example:
SMFL



16

note that we still haven't written any actual code

we've made a *plan* for how to break the problem into objects

17

note: it can be hard to break problems into objects

Consider a very basic question: should a Message send itself?
'Sending' is a key thing I wish to do with Messages,
so surely Message objects should have a 'send' method, right?
If Messages don't send themselves, then some other object
will have todo the sending, like perhaps some not-yet-created Sender object.
Or wait, every sent Message needs a Recipient, so maybe instead Recipient
objects should have a 'receive' method.
This is the conundrum at the heart of object decomposition.
Every behavior can be re-contextualized by swapping around
the subject, verb, and objects.
Senders can send messages to Recipients;
Messages can send themselves to Recipients;
and Recipients can receive messages.
--Brian Will

18

so

19

it is very hard to break a problem into objects

20

but

21

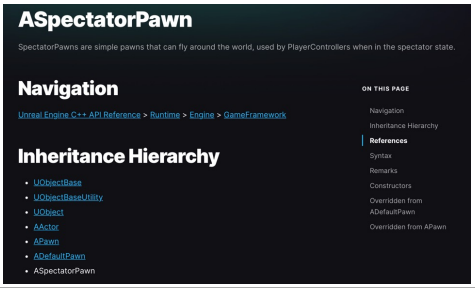
it is also very *popular* to break a problem into objects

22

so let's learn some OOP 🙂 👍

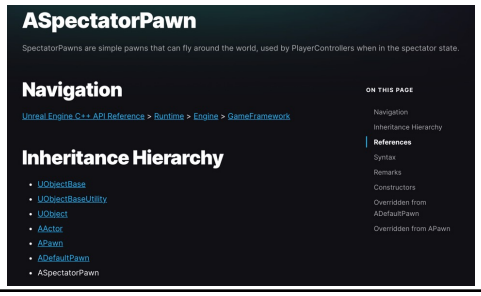
23

example: Unreal Engine API



24

example: PyTorch



ASpectatorPawn

SpectatorPawns are simple pawns that can fly around the world, used by PlayerControllers when in the spectator state.

Navigation

Unreal Engine C++ API Reference > Runtime > Engine > GameFramework

Inheritance Hierarchy

- UObjectBase
- UObjectBaseUtility
- UObject
- AActor
- APawn
- ADefaultPawn
- ASpectatorPawn

ON THIS PAGE

Navigation
Inheritance Hierarchy
References
Syntax
Remarks
Constructors
Overridden from ADefaultPawn
Overridden from APawn

25

inheritance

26

one class can **inherit** from another

- a **child class** (**derived class**, **subclass**) inherits from its **parent class** (**base class**, **superclass**)
  - a child **inherits** (gets, has) its parents' variables and functions

```
// Inheritance: HW13 is a Cow (app)
class HW13 extends Cow {
    public static void main(...) {
        while (beginFrame()) {
            ...
        }
    }
}
```

```
class Cow {
    static boolean beginFrame(...);

    static double mouseX;
    static double mouseY;
    static boolean keyPressed(...);
    ...
}
```

27

inheritance is convenient, but NOT fundamental

(except sometimes in Java)

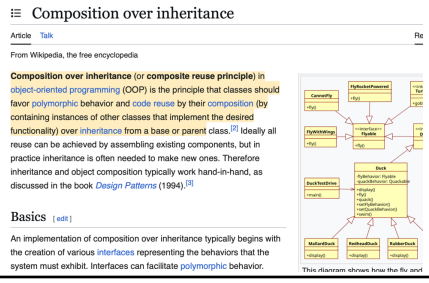
- instead of extending a class, we can store a reference to an instance of it
  - this is called "**composition**"
  - we will have to use the dot operator (a lot) more, but c'est la vie; they're fundamentally the same thing*

```
// Inheritance: HW13 is a Cow
class HW13 extends Cow {
    public static void main(...) {
        while (beginFrame()) {
            ...
        }
    }
}
```

```
// Composition: HW13 has a Cow
class HW13 {
    static Cow cow;
    public static void main(...) {
        while (cow.beginFrame()) {
            ...
        }
    }
}
```

28

***sidenote:** some people don't think agree with me that composition and inheritance are the same thing



Composition over inheritance

Article Talk

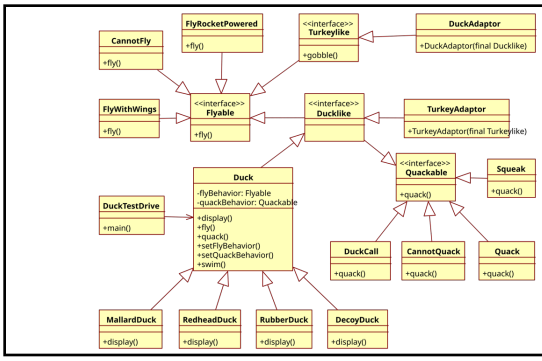
From Wikipedia, the free encyclopedia

Composition over inheritance (or **composite reuse principle**) in object-oriented programming (OOP) is the principle that classes should favor polymorphic behavior and code reuse by their composition (by containing instances of other classes that implement the desired functionality) over inheritance from a base or parent class.[2] Ideally all reuse can be achieved by assembling existing components, but in practice inheritance is often needed to make new ones. Therefore inheritance and object composition typically work hand-in-hand, as discussed in the book *Design Patterns* (1994).[3]

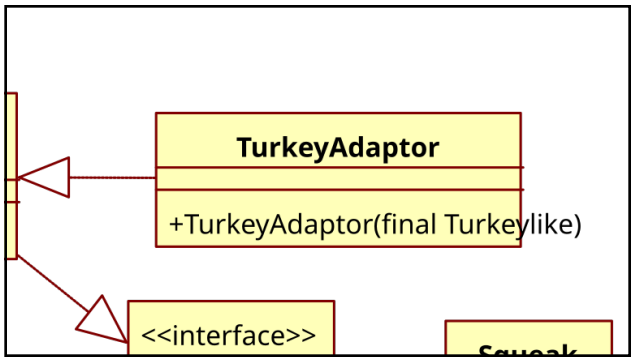
Basics [edit]

An implementation of composition over inheritance typically begins with the creation of various interfaces representing the behaviors that the system must exhibit. Interfaces can facilitate polymorphic behavior.

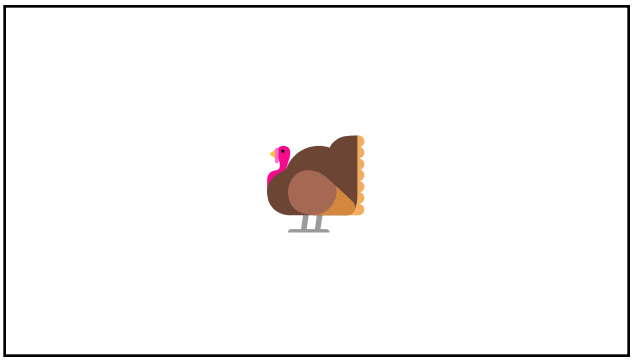
29



30



31



32

gobble gobble

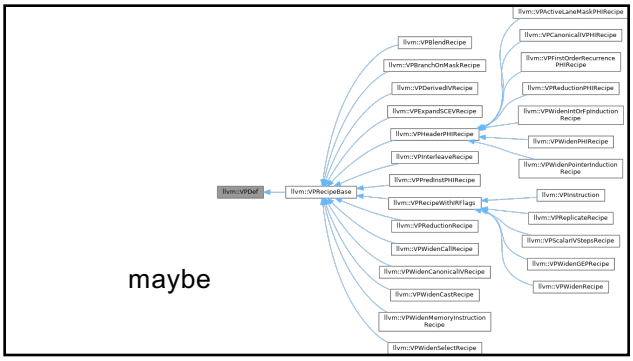
33

anyway.

34

the fundamental point is that
maybe it's nice to reuse code

35



36

final note: inheritance doesn't simplify a problem so much as i t s p r e a d s i t o u t (and spreads it *around*?)

37

down the rabbit hole...

```
public class ArrayList<E>
```

38

down the rabbit hole...

```
public class ArrayList<E> extends AbstractList<E>
                          implements List<E>,
                                     RandomAccess,
                                     Cloneable,
                                     java.io.Serializable
```

39

down the rabbit hole...

```
public abstract class AbstractList<E> extends AbstractCollection<E>
                                      implements List<E> {
```

40

down the rabbit hole...

```
public abstract class AbstractCollection<E> implements Collection<E> {
```

41

down the rabbit hole...

```
public interface Collection<E> extends Iterable<E> {
```

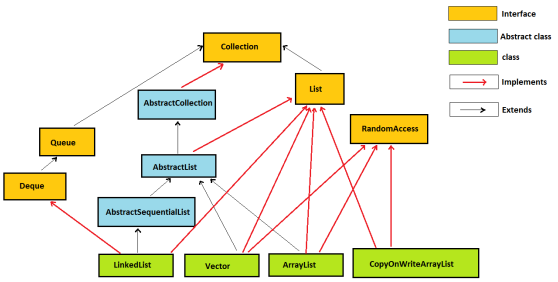
42

down the rabbit hole...

```
public interface Iterable<T> {


    /**
     * Returns an iterator over a set of elements of type T.
     *
     * @return an Iterator.
     */
    Iterator<T> iterator();
}
```

43

down the rabbit hole...



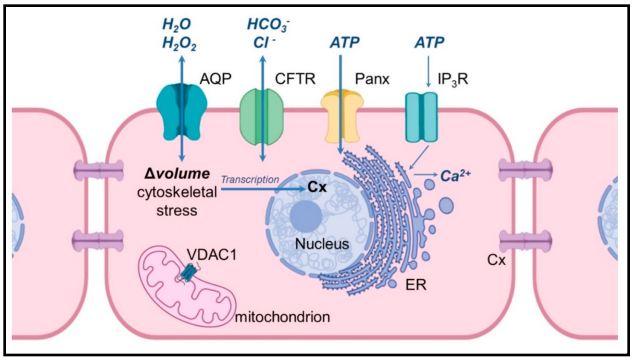
44

gobble gobble

45

encapsulation

46



47

encapsulation

- **encapsulation** is the idea that a class should be like a "capsule"
  - the variables inside the capsule should be **`private`**
    - users of the class CANNOT touch them
  - for its users, the class should expose safe, **`public`** functions

48

```
void put(KeyType key, ValueType value) { ... }
ValueType get(KeyType key) { ... }
int size() { return this._size; }
```

```
ArrayList<KeyValuePair>[] buckets;
int _size;
```

encapsulated hashmap

49

note: this probably makes sense

the typical user of a hashmap shouldn't be messing with the **`private`** array

(and perhaps the exceptional user should write their own hashmap)

50

big idea: encapsulation can hide away messy, dangerous details

note: this is just a metaphor;
do NOT play with fire

51

however

encapsulation can maybe be taken too far

```
bullet.age++;
```

```
bullet.setAge(bullet.getAge() + 1);
```

```
bullet.ageUp(); // ?
```

52

OOP considered maybe mildly frictious to prototyping

53

final note: encapsulation doesn't *add* functionality
encapsulation *removes* functionality

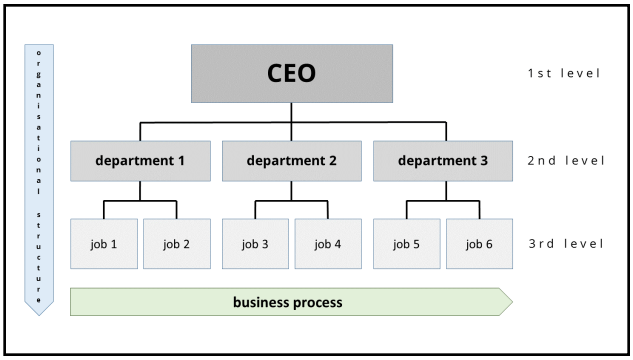
54

special topic:
conway's law

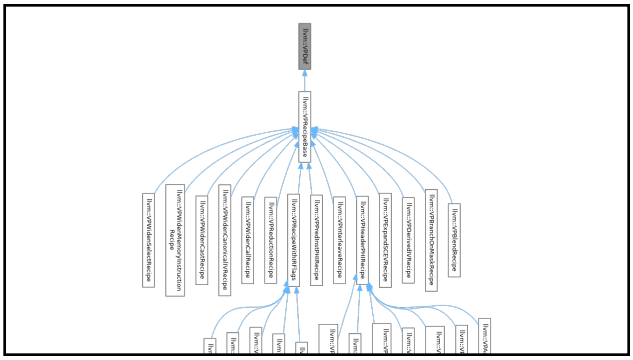
55

[O]rganizations which design systems (in the broad sense used here) are constrained to produce designs which are copies of the communication structures of these organizations.
—*Melvin E. Conway, How Do Committees Invent?*

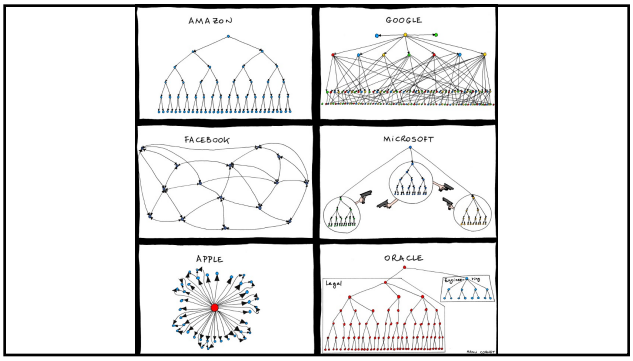
56



57



58



59

W E E K 1 3 b

T O D A Y
is Wednesday

W A R M U P
what is a computer program?
what is programming?
what is a data structure?

62

Week13b

TODAY
final exam logistics
Fun Kahoot Wednesday Final Review
also doughnuts

WARMUP
what is a computer?
what is a computer program?
what is programming?
what is data?
what is a data structure?

63

final exam logistics

64

final exam logistics

65

final exam

- the 2 hour **final exam** will be Sun, Dec 15, 1:30 PM in TCL 123 (Wege)
  - if you have a conflict (exam hardship, international travel), and talk to me by the end of day today, then you take the final exactly 24 hours early (same location, same time)
- the final exam covers data structures and (A-, A) HW from **after** the midterm
  - from PostScript to Recursion + Dynamic Programming
  - **note:** you will still need to know some stuff from the first half of the course
    - Java syntax
    - what is an array
    - etc.

66

what has this course been about?

67

data structures & big O runtime

as i review...
use the big O runtimes to remind yourself how the data structures work 😊👍

68

pop quiz:
what is the most important data structure?

69

quadtree

70

just kidding

71

it's the arrayyyyy

72

array

73

(also trees, but really mostly arrays.)

74

data structures

75

array, string

76

an **array** is a sequence of contiguous elements

- the length of an array is fixed
- ⏱ creating an array takes O(n) time, where n is the length of the array
- because the elements are all the same size and all right next to each other, **accessing** an array is FAST
  - ⏱ getting the value of the i-th element of an array takes O(1) time
  - ⏱ setting the value of the i-th element of an array takes O(1) time

77

a **string** is internally an array of characters

- ⏱ adding (concatenating) two strings of length n is an O(n) operation
  - we have to make a new string of length 2n, which is O(n)

78

array list

79

we used an array to implement an **array list**, which automatically resizes itself

- ⏱ when the internal private array is not full, adding (to the end) is O(1)
- ⏱ when the internal private array is full, adding is O(n)
  - have to make a new array (of, for example, twice the length) and copy

80

stack, queue, hash map

81

we used an array list to implement a **stack** and a **queue**

- stacks and queues restrict the way we access a sequence of elements
  - a **stack** can **push** an element to the top & **pop** the top element
    - 🍽️
  - a **queue** can **add** an element to the back & **remove** the front element
    - 🚶🚶🚶🚶🚶🚶🚶

82

we can use an array of array lists to implement a **hash map**

- ⏱ putting (adding) a key-value pair into a hash map is (amoritzed) O(1)
  - it's a "slow O(1)," which involves **hashing** the key
    - in **separate chaining** (like we did in class), this takes us to a **bucket**
      - if the key was already in the bucket, we overwrite its value
      - otherwise we add the key-value pair to the bucket
- ⏱ get key's value from a hash map is O(1)

83

node, linked list, tree, binary tree, binary search tree, heap

84

a **node** is a little teeny class

- a node can store **data** (like a **value**) as well as **references** to other nodes

85

a chain of nodes is a **linked list**

- a linked list has the same **interface** as an array list, but acts very differently
- ⏱ adding to the end of a singly linked list is O(n)
- ⏱ getting an element by index is O(n) ☹️

86

a branching chain of nodes (with no cycles) is a **tree**

- each node has references to its **children**
- a node's children & its children's children & its children's children's children, ... are called its **descendants**
- the node with no parents is called the **root**

87

a tree where each node can only have up to two children is a **binary tree**

- a node in a binary tree can have a **left child** and a **right child**
- the height of a **balanced** binary tree is O(log n)

88

binary search trees and heaps are special kinds of binary trees

- in a **binary search tree**, each node's value is greater than the values of all of its left descendants and less than the values of all of its right descendants
  - ⏱ **searching** a **balanced** binary search tree is O(log n)
- in a **max binary heap**, each node's value is greater than the values of both of its children
  - ⏱ **adding** an element to a max binary heap is O(log n)
    - 🌸 add to the first empty slot and **swim up**
  - ⏱ **removing** the max element (root) from a max binary heap is O(log n)
    - ⚓ swap the **last element** with the root and **sink down**

89

90

91

92

what does it all mean?

93

what does it all mean?

94

here's my attempt to summarize the course in a few sentences

95

136

- **data** means numbers (letters are numbers, colors are numbers, ...)
- there's no one perfect way to organize your data
  - **it depends on the problem**
- to find a good data structure for a particular problem, we can...
  - ...just try an array first; if that works, great, we're done 😊👍
  - ...compare different data structures using math (**big O** notation)
    - ...in terms of speed
    - ...in terms of space
  - ...compare different data structure's **usage code**
    - ...an array might be *fast*, but is it *pleasant*?

96

136 was us moving from finding *a* solution...

...to finding a good solution

97

and starting to build the tools **and confidence** to solve bigger problems

💪🥲

98

for example, the final project 😀 👍

99

homework

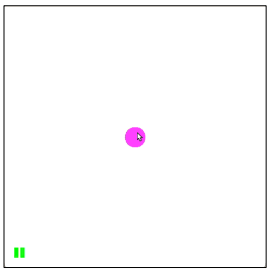
✨ **how do the post-midterm homeworks *work?***
can you "run them by hand?"

100

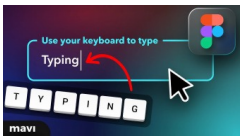

```python
def pythag(a, b):
    global D

    result = (a * a + b * b) ** 0.5
    D += result
    return result

A = pythag(3.0, 4.0)
print(A) # 5.0

A = pythag(A, A)
print(A) # 7.0710...

B = (A < 10.0)
if B:
    C = 0
    for i in range(5):
        C += i
    print(C) # 10 (NOT 10.0)
    D += C

assert (D > 20.0)
```

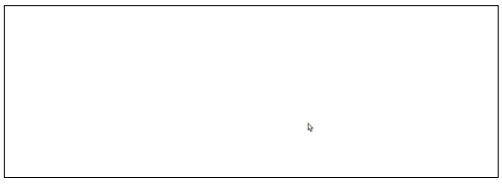
101

EMMA WATSON TOM HANKS
THE CIRCLE
KNOWING IS GOOD.
KNOWING Cow.java IS BETTER.

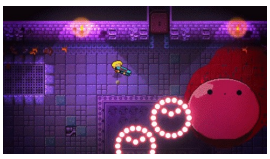


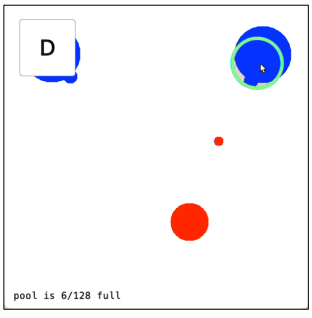
102



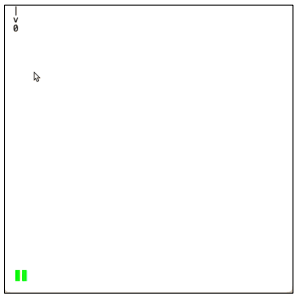


103





104



105

Adobe PostScript

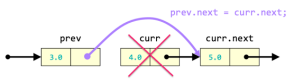
how do you interpret PostScript?

106

how do you generate text?

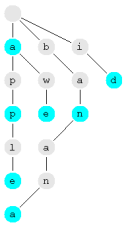
107

remove (delete) a node



how do you create & change a linked list?

108

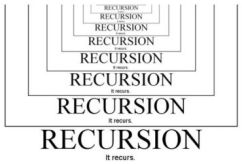


how do you store a vocabulary in a tree (trie)?

109

how can i search and sort?

110



what is the big O runtime of a piece of code?

111

questions

112

what does it all mean?

113

114

what does it all meme?

115

116

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;`

117

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;`

118

story time

119

About the security content of iOS 7.0.6

This document describes the security content of iOS 7.0.6.

For the protection of our customers, Apple does not disclose, discuss, or confirm security issues until a full investigation has occurred and any necessary patches or releases are available. To learn more about Apple Product Security, see the Apple Product Security website.

For information about the Apple Product Security PGP Key, see "How to use the Apple Product Security PGP Key."

Where possible, CVE IDs are used to reference the vulnerabilities for further information.

To learn about other Security Updates, see "Apple Security Updates".

iOS 7.0.6

- **Data Security**

  Available for: iPhone 4 and later, iPod touch (5th generation), iPad 2 and later

  Impact: An attacker with a privileged network position may capture or modify data in sessions protected by SSL/TLS

  Description: Secure Transport failed to validate the authenticity of the connection. This issue was addressed by restoring missing validation steps.

  CVE-ID

  CVE-2014-1266

120

```
static OSStatus
SSLVerifySignedServerKeyExchange(SSLContext *ctx, bool isRsa, SSLBuffer signedParams,
                                 uint8_t *signature, UInt16 signatureLen)
{
        OSStatus        err;
        ...

        if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
                goto fail;
        if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
                goto fail;
                goto fail;
        if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
                goto fail;
        ...

fail:
        SSLFreeBuffer(&signedHashes);
        SSLFreeBuffer(&hashCtx);
        return err;
}
```

121

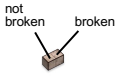
```
static OSStatus
SSLVerifySignedServerKeyExchange(SSLContext *ctx, bool isRsa, SSLBuffer signedParams,
                                 uint8_t *signature, UInt16 signatureLen)
{
        OSStatus        err;
        ...

        if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
                goto fail;
        if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0) {
                goto fail; }
        goto fail;
        if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
                goto fail;
        ...

fail:
        SSLFreeBuffer(&signedHashes);
        SSLFreeBuffer(&hashCtx);
        return err;
}
```

122

Caaaaaaaaaaarl

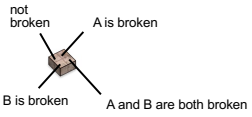
- **e.g.,** Imagine a classroom with n students. I want to figure out if any students are named Carl.
  - I need an ✨Algorithm✨ -- `boolean isAnyoneNamedCarl(Student[] students);`
  - What is the big O of the following algorithms?
    - **Algorithm 1:** Ask each student, one at a time, "Are you named Carl?"
    - **Algorithm 2:** Pass a paper around the room, and have each student write their name on it. Then take the paper, and read through it.
    - **Algorithm 3:** The students draw straws. The student who draws the short straw must leave. On their way out of the room, ask them whether their name is Carl. Repeat this procedure until the room is empty.
    - **Algorithm 4:** Play Kahoot. The winner legally changes their name to Carl.

123



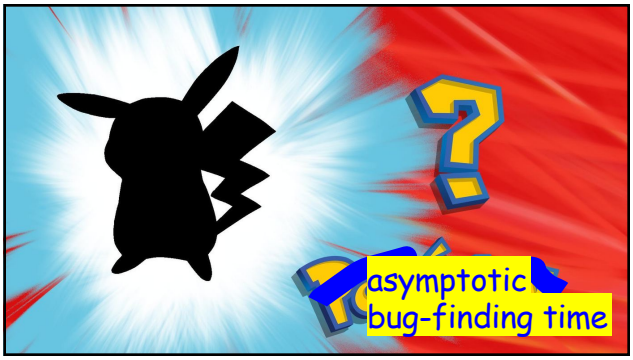
developing one
thing at a time

124

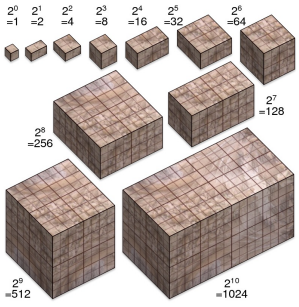


developing two
things at a time

125



126

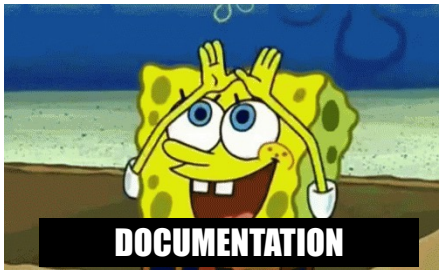


$\mathcal{O}(2^n)$

gl;hf 😳 👍

127

128



129

Tungsten break

did you know the density of Tungsten is 19.25 g/cm^3?
Gold is 19.32 g/cm^3

did you know Tungsten is really hard?
Tungsten alloys can be as hard as Sapphire! that's really hard!

Tungsten is expensive (that cube is $60) but not like...that expensive

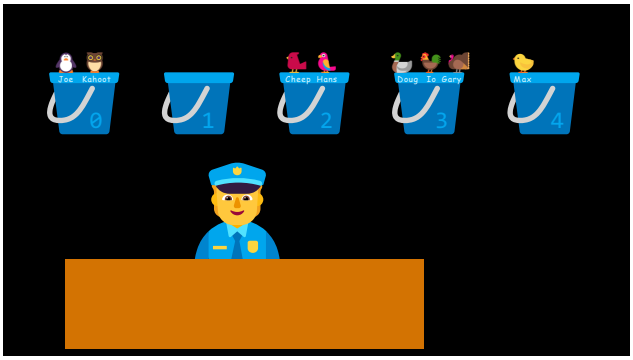
what could YOU do with Tungsten?

130

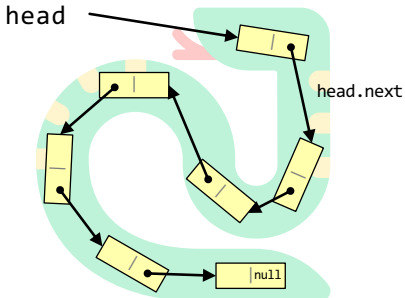
```
if (token.type == 2) {
    ...
}
```

```
if (token.type == Token.TYPE_STRING) {
    ...
}
```

131



132



133

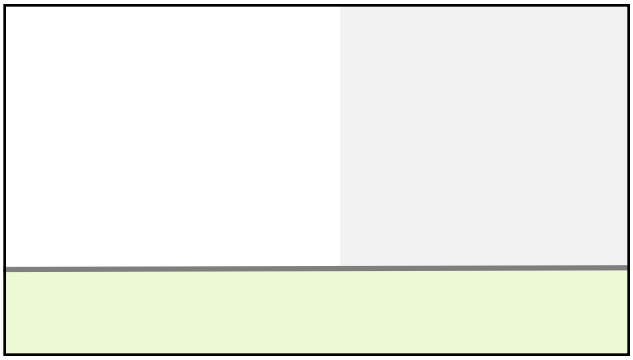
"the stack"

local variable primitives & **references** to Objects
undefined *by default (will NOT compile if used)*

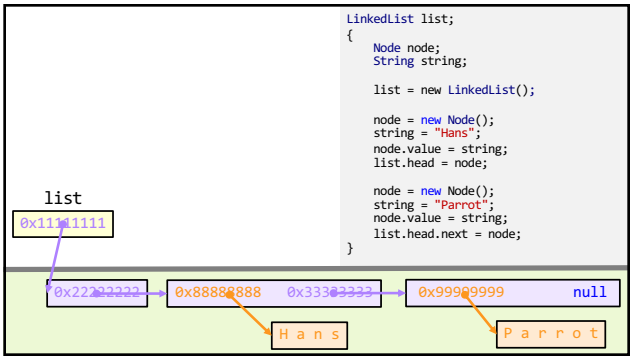
"the heap"

the actual **Objects** (arrays and String's count as Objects)
cleared to 0 or null by default

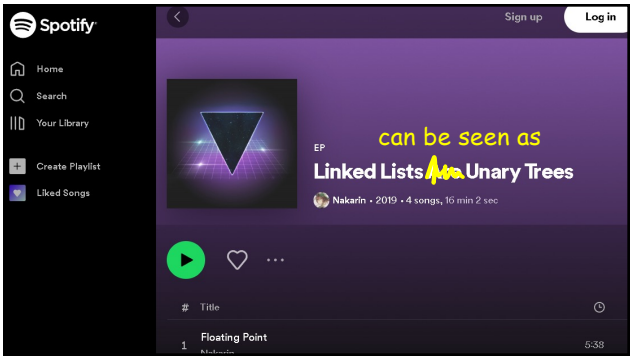
134



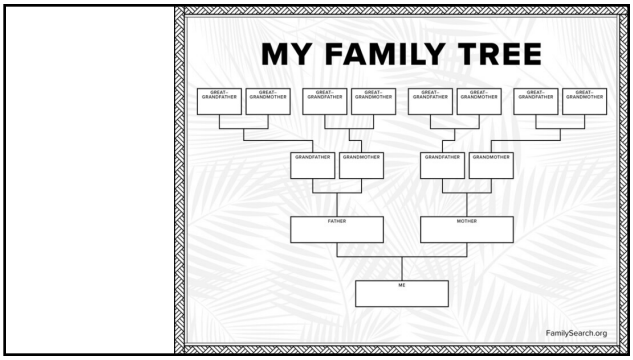
135



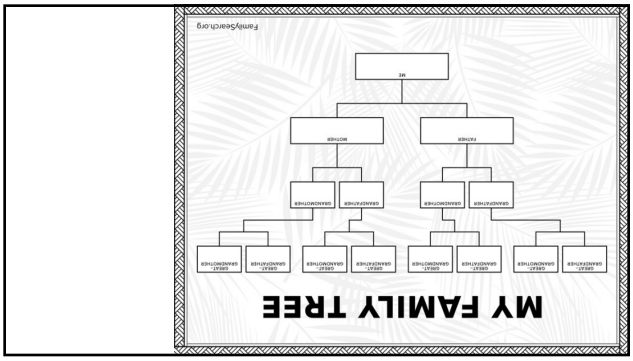
136



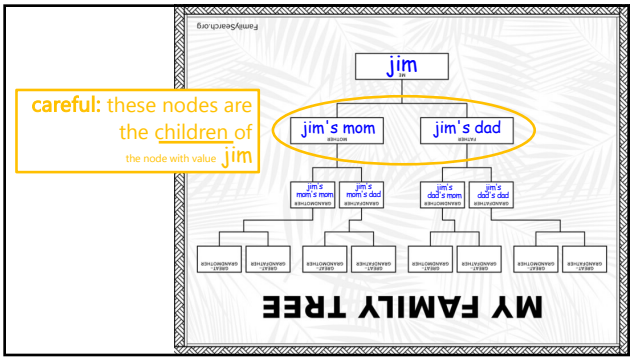
137



138

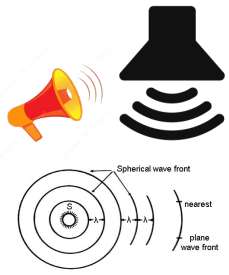


139

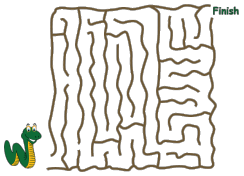


140

breadth-first vs. (pre-order) **depth-first**



"a persistent, optimistic, very thorough worm" –Mark Hopkins

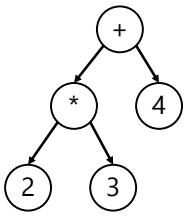


141

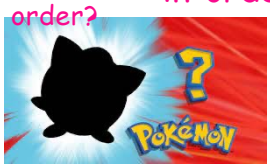
quadtree rabbit hole

142

2 * 3 + 4

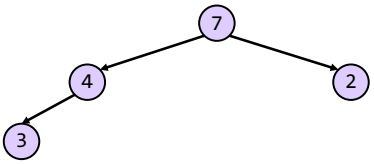


who's that traversal order? in-order!



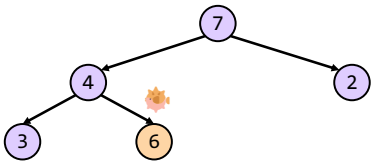
143

[7 4 2 3 6 1 8 5 9]



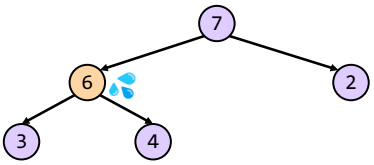
144

[7 4 2 3 6 1 8 5 9]



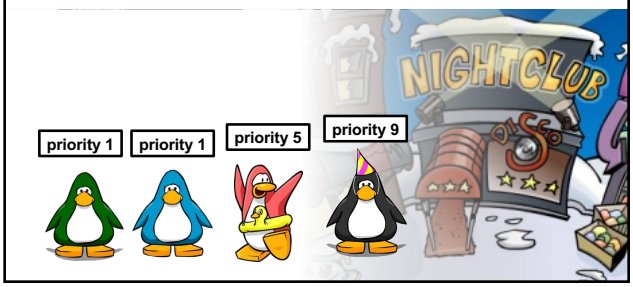
145

[7 6 2 3 4 1 8 5 9]



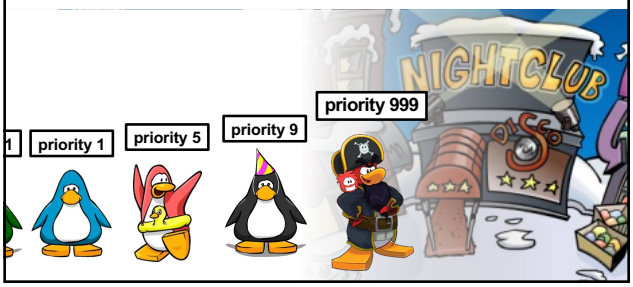
146

extension: priority queue



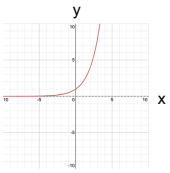
147

extension: priority queue



148

$y = 2^x$

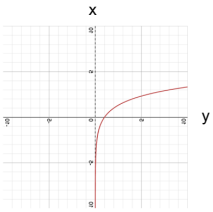


$x = \log_2 y$

149

$y = 2^x$

$x = \log_2 y$



150

😳👍

151

😳👍

152


questions