Inheritance and Interfaces

- Implements - an interface, signs contract and must define those methods
- Extends - "is a" marked as subclass of parent, inherits stuff except private (?)
- Interfaces can be used as static containers - that's the point. Pet, Machine, etc.

Static vs Dynamic

- Local variaable - only within method
- Class variable - shared by all instances. can be accessed by Class.variable
- Instance variable - this.variable
- ONLY instance methods accessed by dynamic type
- static methods use static type and class variables

Scope

1. asking for field or static method => use static type
2. asking for non-static method => use dynamic type
3. "this" within a [particular class] => static type becomes [particular class], dynamic type same
4. a method of dynamic type D and static type S is calling a non-static [method], then the [method] has to exist in S (not necessarily in D)
5. static methods must be overridden by static methods, non-static must be overridden by non-static
6.
7. Look up static methods and fields of any time in the class of the static type.
8. Look up instance methods in the class of the dynamic type.
9. Additionally, whenever a call is made to an instance method, keep in mind that the this pointer also has a static and dynamic type, so when you call a method like f() or this.f(), you also have to use the same rules to determine which method actually gets called.

Emacs and GJDB Emacs:

- C-x c-s - save
- C-x c-f - load file
- M-x compile - execute command (make)
- M-x gjdb - debug.

GJDB

- C-c C-s - step (step into it(?))
- C-c C-n - next (next breakpoint/line)
- C-c < - up (to caller)
- C-c > - down (into call (?))
- C-c SPACE - set break point
- javac -g Main.java to debug, then do gjdb Main
- run - starts program
- where - prints stack trace
- up - climbs to parent frame
- down - undo an "up"
- print E - prints E
- quit - duh
- break - make a breakpoint
- cont - continue
- step - executes line, stops at start of next line

Javadoc
Lists

- add(?index?, elem), clear(), get(elem), remove(), size()
- contains(elem), indexOf(elem), isEmpty(elem), iterator()

Arrays

- use bracket notation
- .length to get size

Converting numbers

- Note - notation is 0xFF for hex and 0b1010 for binary
- binary to decimal - split into sums of powers of 2.
- decimal to binary - convert to powers of two and add
- decimal to hex - convert to binary, then group by 4 bits, and find corresponding value. A = 10 (1010), B = 11, C = 12, D = 13, E = 14, F = 15 (1111).
- binary to hex - see above.
- hex to decimal - either add powers of 16, or convert to binary then decimal
- hex to binary - for each element, find 4-bit sequence, and concatenate.

Running Time

- O - bounded above, worst case scenario. can be overly pessimistic
- Ω - bounded below, can be overly optimistic
- Θ - bounded adove and below, tight bond. "Family"

General tips for finding running time

- loops are a variable with length = number of times loop runs
- nested loops cause squaring. even smart nested loops just cut time in half, so we ignore the constant
- If we cut search space in half, we get lg time
- if we make recursive calls, say 2, we get 2^n time, so exponential (think bacteria dividing)
- Take limit of fraction $\frac{f(x)}{g(x)}$. if ∞ then f(x) bigger, if 0 g(x) bigger, if constant then $f(x) \in O(g(x))$
- $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$, $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$
- Order of times:
  $1 < log(n) < n < nlog(n) < n^2 < 2^n < O(n!)$

Bit shifting

- >> - arithmetic right (shift and fill with sign). ie, $1001 >> 2 = 1110$. the 01 tail is dropped, and leading one is cloned and shifted in. Divides by 2^N and rounds down.
- << - arithmetic left. ie, $1001 >> 2 = 0100$. the 10 lead is dropped, and 2 zeros shifted in on the right. Basically multiplying by 2^N, with N = shift
- >>> - logical right . ie, $10101 >>> 3 = 00101$. Drop tail, shift, and fill in front with zeros.
- ~ (NOT) - flip all bits.
- ^ (XOR) - 0, 1, 1, 0 - (even bits problem, flip back and forth)
- & (AND) - 0,0,0,1
- | (OR) - 0,1,1,1
- masking - use hex number, shift it to desired position and AND to target. Ie, 0x15 is 4 ones, so 0x15 << 2 would extract bits 6,5,4,3, and nuke bits 2 and 1
- Inverse masking $x\&\sim M$ extracts all BUT M
- two's complement - leading 1 means negative, so flip bits and add 1

Note - 2's complement - if number starts with a 1 it's negative, so flip all bits and add 1, and that number is the negative number. so $1010 \rightarrow 0101 + 1 \rightarrow 0110 = -6$

Powers

- $2 = 1_0, 2_1, 4_2, 8_3, 16_4, 32_5, 64_6, 128_7$ $256_8, 512_9, 1024_10, 2048_11, 4096_{12}$
- $16 = 1_0, 16_1, 256_2, 4096_3, 655364$

Access modifiers

- private - only this class can use it - no subclasses or anything
- package private - only class, package, and subclass IN SAME package can access
- protected - class, package, subclass can access
- default - class and package can access, not subclass or world
- public - world accessible

Functions as Objects OOP slide 3 of review session lecture 19 minute 40:00

```
public abstract class IntFunction {
    public abstract int apply(int x);
    public final static IntFunction ID = new IDFunction();
    public IntFunction compose(IntFunction g){
        return new Composition(this, g);
    }
    private static class IDFunction extends IntFunction {
        public int apply(int x) {return x;}
    }
    private static class Composition extends IntFunction {
        Composition(IntFunction f, IntFunction g) { _ f = f; _
g = g; }
        public int apply(int x) {return f.apply(g.apply(x)); }
    }
}
```

Random Notes

- Destructive = no new operators, use reassignment (intlist tails, etc)
- non-destructive - be sure to use NEW operators
- Functions as objects - some internal state (for adder), apply does the actual function - adder.apply$_n + arg$

Exceptions

- Unchecked - programmer errores - bad arg, null pointer, etc. can be thrown ANYWHERE by JVM
- Checked - crazy circumstances, forseen but not programmer error. ie, file DNE, connection dropped, etc
- Checked should use try catch statement. add "Throws" to method signature

- Relax!
- This is the last push.
- You know this, and you will do well.
- Channel!