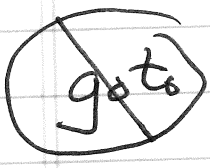
Object Oriented Programming OOP (1)

began: Simula 67
Norwegian Computing Center (Oslo)
Ole-Johan Dahl
& Kristen Nygaard
⇒ influenced Bjarne Stroustrup (C++)

Structured Programming
Edsger W. Dijkstra
"Goto statement considered Harmful"



- most languages incorporated "structured" constructs
  if/else, while, for
  break, continue, return, throw/catch

Motivation - Reuse & Independence
- reuse components
- easy modification of behavior
- independence of components

modules
- extension of data & operations
- first class user def types
- redefining existing operations
- app. frameworks
  ex: C++ containers (STL)
  Java collections & containers

Design Issues

OOP 2

- dynamic vs static
  - dynamic → runtime
  - static → compile time
- performance penalty for
  - dynamic dispatch (compiled lang)
  - less import interp lang

C++ "you don't pay for what you don't use"

- runtime environment
- compiler optimizations

Classes vs types

1. exclude from type checking
   objects all dynamically checked @runtime

Objective-C = C + Smalltalk
(Smalltk too slow:
in 1980 needed ≥ 512 KB RAM, 2 MHz)
used @Apple

ex:
```
size = [anarray count];
[map insert: thing withkey: foo];
```

2. classes as type constructors
   - part of type system (C++ and Java)
   C++: class ≡ struct — POD
   but may have functions

~~So if Ta, Ub, then~~
So if U extends T and — in Java

T
|
U

Ta; Ub then a=b is OK but
b=a is error
in C++ a=b is SLICED.
but OK ŵ ptrs

3. classes ARE type system (Smalltalk)
   - all other types excluded.
   Java exception: primitives & arrays

Classes vs Modules

- classes specify interfaces
- information hiding (private, protected, public, friend, pkg)

module = namespace (C++) or package (Java)
modules orthogonal to ~~class~~ class system
C++ has friend concept
Java protected can access all package

Polymorphism

OOP (4)

- universal
  - parametric
  - inclusion
- ad hoc
  - overloading
  - coercion

Parametric polymorphism

Java generic; C++ template; ML boxed

class stack<T>

template <typename T> class stack

'a stack

C++ recompiles class ∀ each Type param
- works for primitives

Java compiles Type param as Object
- only objects allowed; not primitives

consequence: all Java instance dispatch dynamic
C++ instance dispatch fast as C

```
class stack<T> {
    void push (T x){...}
    T pop (){...}
    ...
}
```

```
stack<Integer> s;
s.push(i)
i = s.pop();
```

```
class stack {
    void push (Object x) {...}
    Object pop() {...}
}
```

```
stack s;
s.push (new Integer(i));
i = (Integer) s.pop().
        intValue
```

NO TYPE CHECKING

C++ - no dyn dispatch
- allows primitives.

Inclusion polymorphism

- subtyping
- inheritance
- overriding

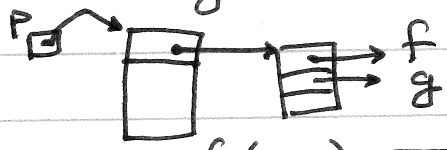
dynamic

superclass | subclass

base class | derived class

virtual fns in C++
non-static fns in Java
each object has pointer to class table
virtual fn table

T *p;



$\therefore p \to f(x,y) \Longrightarrow (p \to vft \to f)(p, x, y)$

- single dispatch (left opnd only)
- multimethod dispatch possible
- fn defined in subclass overrides fn in superclass

Overloading polymorphism

static

fns or operators defined multiple times with diff # of params or diff types

ex:
```
void print(int);
void print(string);
```

overloading common for
builtin operators. e.g. int + int
double + double
C++ also for user opers & fns

Coercion polymphs

- automatic conversion
- ex: widening int → double
narrowing double → int

static
static_cast in C++
using constructor
eg. "_" char[] → string

dynamic_cast: subclass → superclass

Smalltalk

- Alan Kay @ Xerox PARC
1970's Dynabook
- WIMP interface
windows, icons, mouse, pointing
derived from Simula & Lisp
garbage collection

PURE OO dynamic types
(interpreted)

messages sent to receivers.
message has selectors (+ argument?)
returns new object

~~m~~utators vs accessors
~~class~~ class messg vs instance messg

reference semantics not value semantics
var refers to value var contains value

Java hybrid : primitives value
objects reference
C++ objects: as you like it

Java x.clone() → value semantics
C++ copy ctor → value sem.
move ctor → ref semantics with deletion

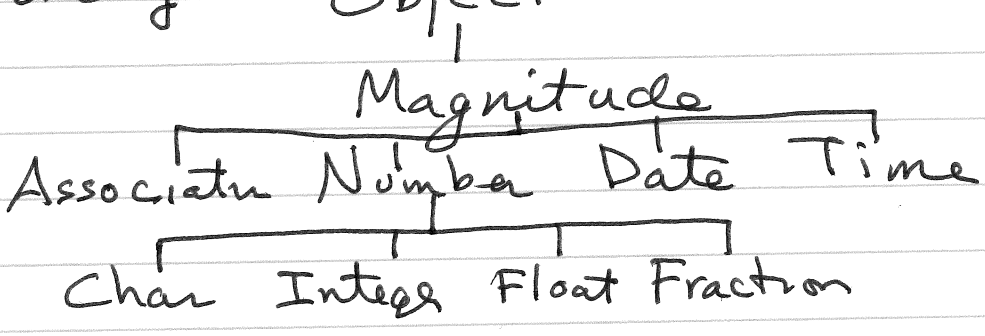
~~Blocks are~~

	Smalltalk	Java	C++ (usually)
equal	a = b	a.equals(b)	a == b
same (identity)	a == b	a == b	&a == &b

ex : to:do:

```
to: num do: block [
    |i|
    i := self
    [i<=num]
        whileTrue : [
            block value: i
            i := i + 1
        ] ]
```

hierarchy



collection hierarchy.
- etc.

Objective-C
- Smalltalk grafted onto C
- C's "other" descendant

Brad Cox answer to Smalltalk because ST was too slow requiring 512KB RAM + 2MHz CPU

ex
```
[super dealloc];
val = [a size];
[a at: i put: v];
```

Apple & NextStep

Java

James Gosling et al @ Sun Msystems

"write once, run anywhere"

- portability, small footprint
- Mwave oven, phone, ...
- machine independent bytecodes
- RPN, JVM
- hybrid non-pure OO
- gcol, references (no ptrs)
- primitives are NOT objects
- low level syntax like C/C++ = familiarity

classes: static (class) fields & methods
non-static (instance) f & m

borrowed from C++, Smalltalk

{ private / protected / public } { package } protection

constructor (def ctor)

constructor chaining
stack C){ this (10) }

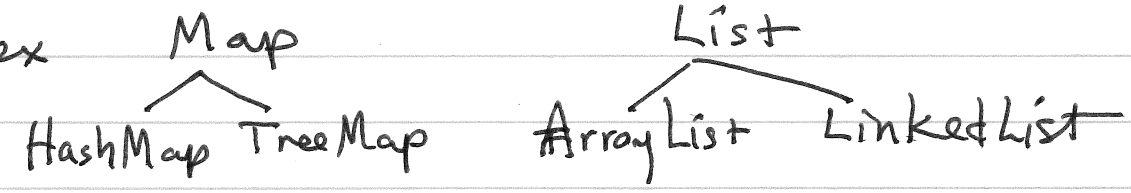
call to super class
arraystack C){ super (foo) }

Java

classes are reference types
prim are value types

single inheritance classes
multiple inherit. interfaces
collections – data structures
containers – GUI

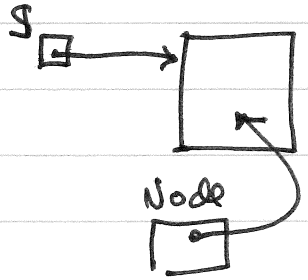
ex



parametric polymorphism (generic)
- implemented as inheritance
- primitives not allowed
- type parameter as Object
- but runtime type checked implicitly

Inner classes

```
class stack{
    class node{
    }
}
```



node has implicit ptr to its outer class, can access outer class fields

private inner class private to clients but not to outer class

```
class stack {
    private class iter{
    }
    Iterator iterator() {
        return new iter
    }
}
```

Binding : static vs dynamic

all ^methods use dynamic binding

non-static

C++ (Bjarne Stroustrup)

latest C++11

in progress C++14 (bug fixes)

originally extension to C

"what you don't use, you don't pay for"

"no lower level language"

private

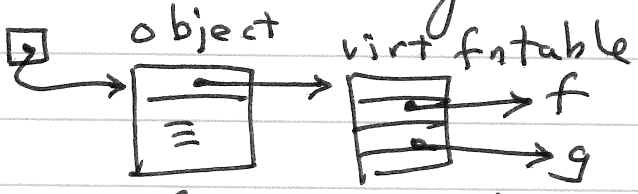
protected friend

public

constructors - used also for conversion
destructors - no GC.
-inline for efficiency

generics as templates
- compile for each arg
- code bloat but runtime efficient

Virtual fns - dynamic binding.



$p \to f(x,y) \Rightarrow (p \to vft \to f)(p, x, y)$

shared_ptr — ref counting
unique_ptr.

~~Implementation~~

Garbage Collection

- must keep RT type id each obj
- can't work with raw ptrs.

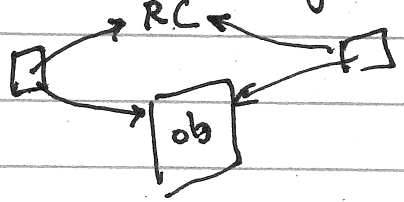
Mark & Sweep:

scan root set for all ptrs to heap
mark those objects
compute closure
= graph marking alg.
scan whole heap
unmark if marked
free if unmarked.

Copying collection:

maintain two heaps
copy all objs pointed by root set
to new heap.
scan new heap copying all old
objs pointed at
then swap heaps

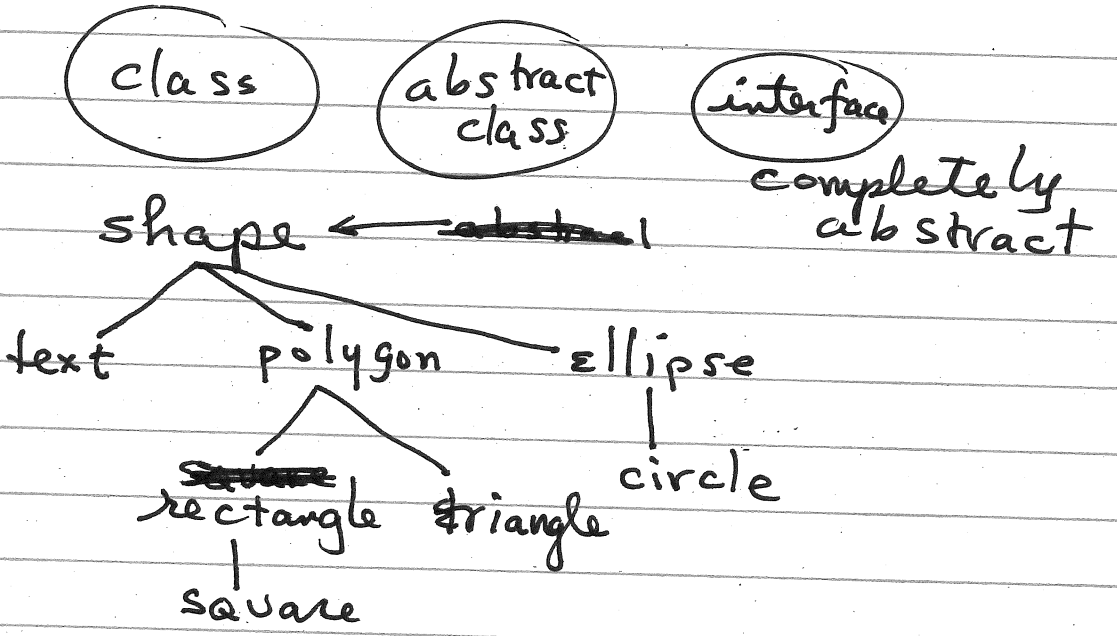
Reference counting



Abstract Classes

OOP (14)

- partially implemented
- contains abstract methods

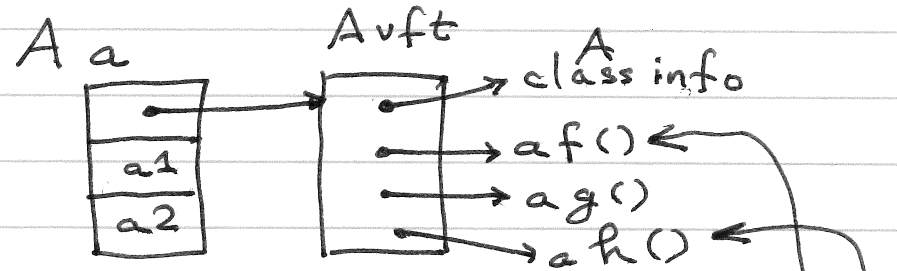


shape.draw is abstract
- can not instantiate (no ctor)
- must be virtual

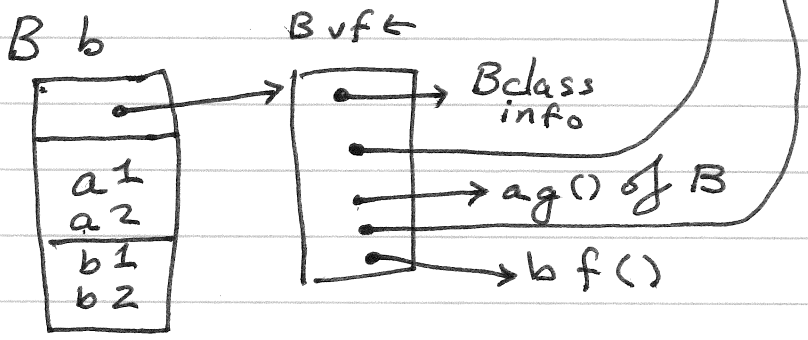
abstract class may have fields & methods
inst. inst.

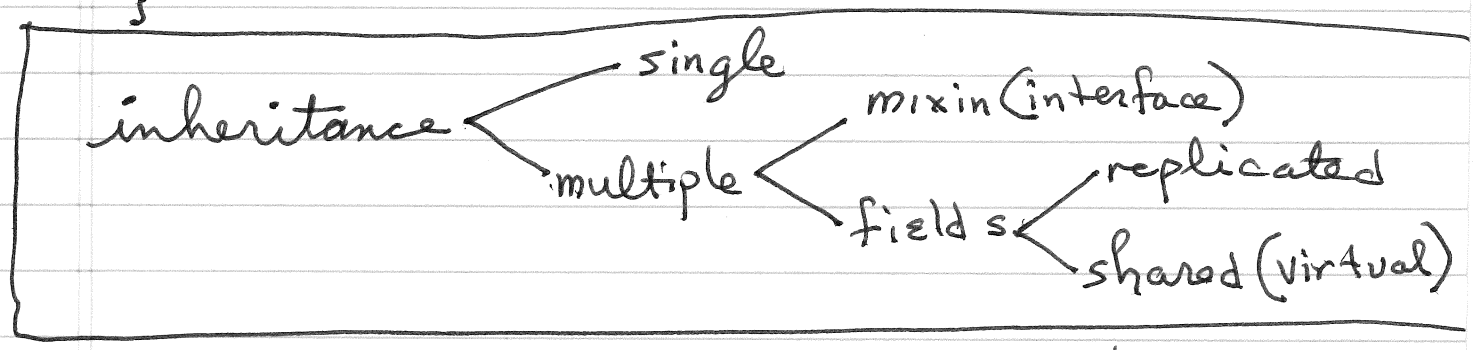
Implementation


```
class A {
    int a1
    int a2
  virtual f()
    g()
    h()
}
```




```
class B ext A {
    int b1
    int b2
    g()
    bf()
}
```





C++ `a=b;` cuts of b1, b2 fields
copies only a1, a2

`b=a; // illegal`

Java
- `A* pa;`
- `B* pb;`

`a=b;` OK copies ptr

`b=a; // illegal`

`b = dynamic_cast<B>(a);`
// OK if *a is really ab
// else nullptr
~~// exn in J.~~
// class cast exn in Java

~~B* pb; pb->~~

A* pa; p->ag(x,y)

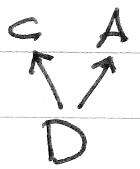
$\Rightarrow$ (p->vft->ag)(p,x,y)

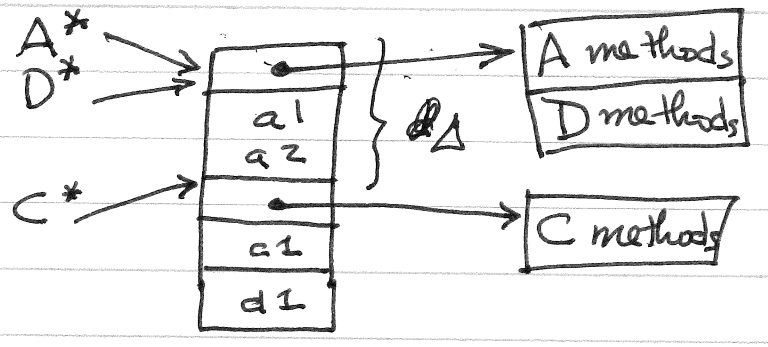
calls A::ag() or B::ag()

depending on which obj.

Non repeated multiple inh

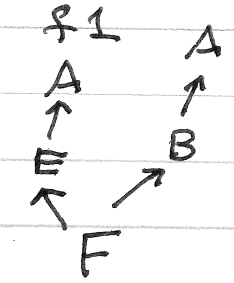
```
class C{
   int c1
   cf()
}
class D : C, A
   int d1
   df()
}
```

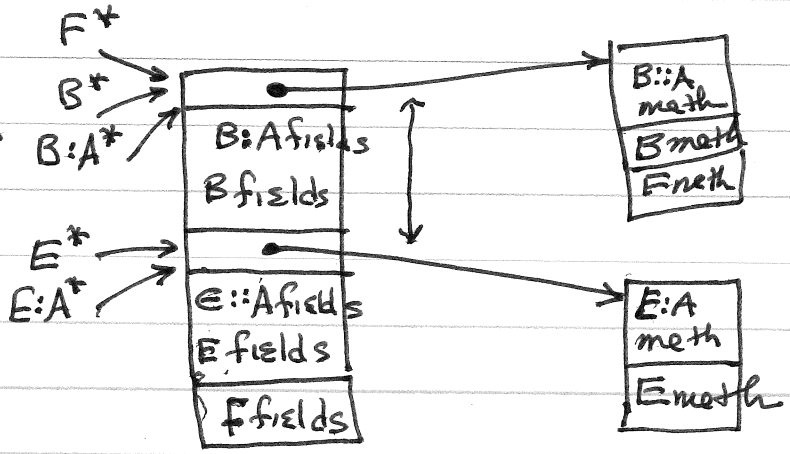




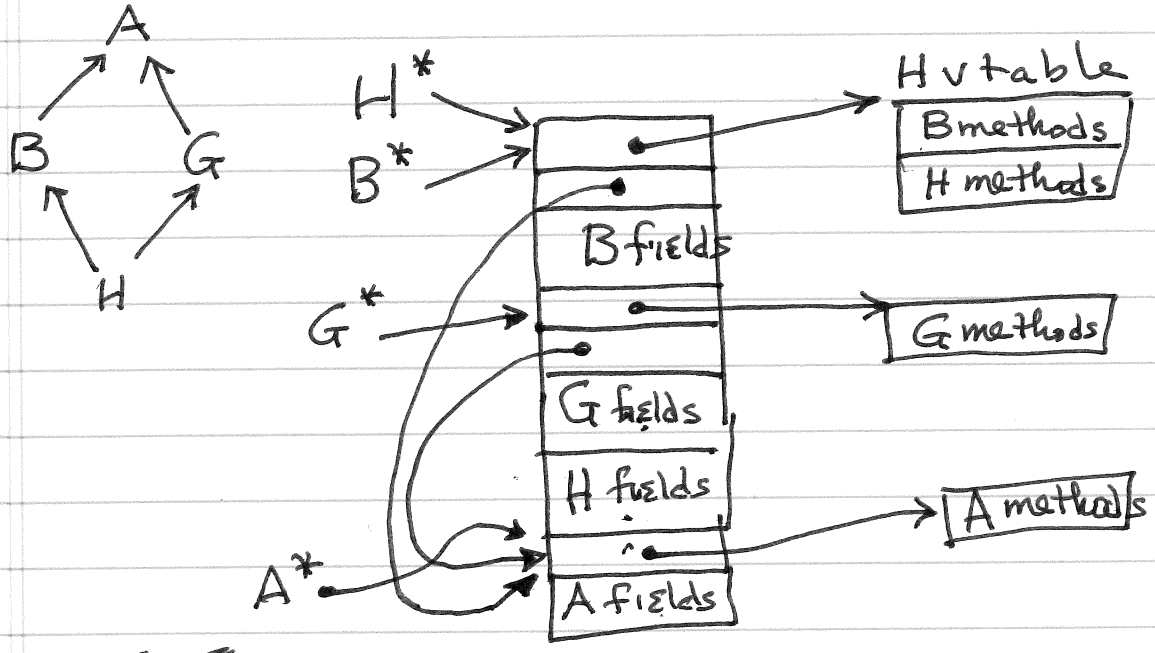
Replicated Inheritance

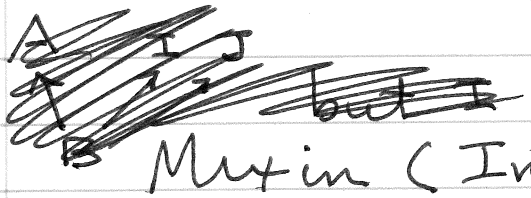
```
class E : A{
   e1
}
class F : E, B{
   f1
}
```



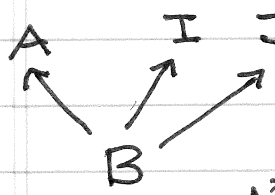


Shared Inheritance (Virtual)





Mix-in (Interface) inheritance



I, J abstract classes
no concrete fns
no fields