

Objects

CS31

Pascal Van Hentenryck



Overview

More data structures

- Object without methods
- methods
- references

Objects

Collection of data with

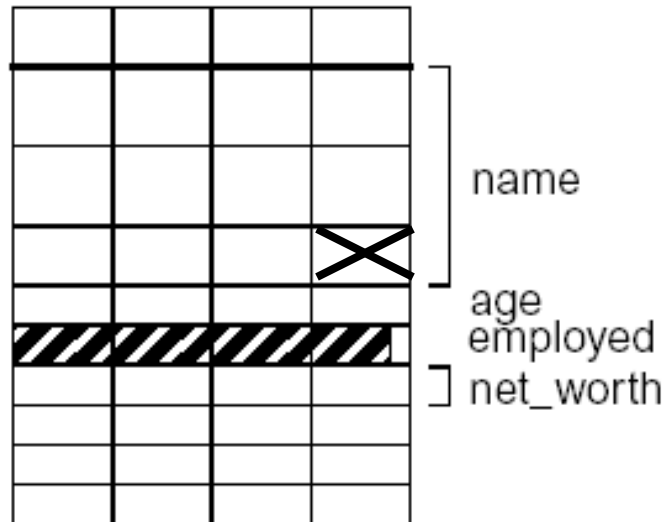
- fixed number of fields
- fields can have different types

In Java, you don't have to worry about how they get allocated in memory.

In assembler, you do.

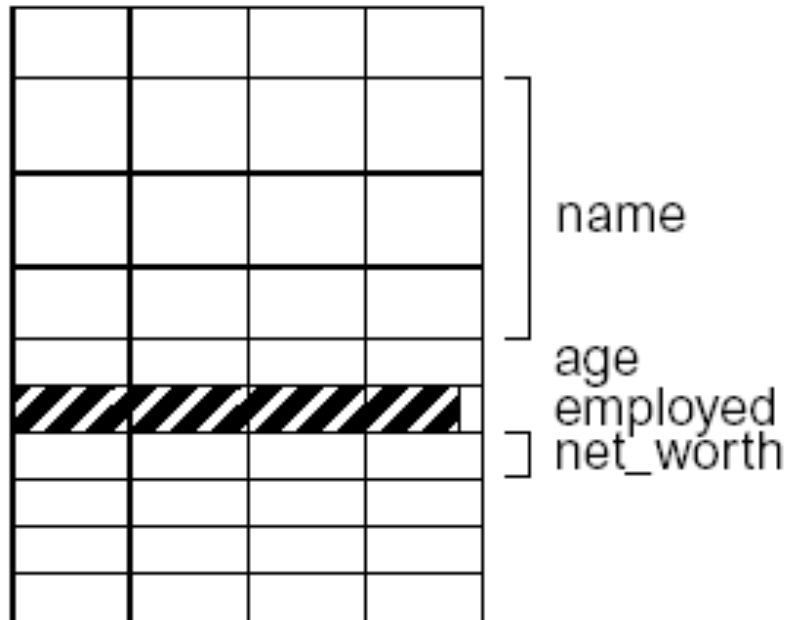
Allocating Objects

```
public class person {  
    public char name[11];  
    public int age;  
    public boolean employed;  
    public int net_worth;  
}
```



- arrange memory to be sure integers and floats are word-aligned
- to pack or not to pack?
(or, time vs space)

Data Declarations



```
                .data
name           =          0
age            =         12
empl           =         16 # take a word
worth          =         20
psize          =         24
pnum           =         10

people:       .word          240
```

Initializing an Object

Assume \$t0 points to the start of a people object in memory.

```
sb    $0, name($t0)    #set first byte of
                        #name to 0
li    $t1, 30          #thirty something
sw    $t1, age($t0)
sw    $0, empl($t0)    #everybody's idle
li    $t2, 1000        #poor
sw    $t2, worth($t0)
```

- Doesn't matter where \$t0's pointing as long as it's supposed to be an object.
- This shows the strength of based addressing; the field offsets are fixed at assembly time.

Array of Objects

`.data`

`_____] field offset declarations`
`_____]`
`_____]`

`psize = 24`

`n = 10`

`people: .byte 240`

`.text`

`#initialize all of the records`

`la $s0,people`

`li $s1,n`

`loop: jal init_person`

`add $s0,$s0,psize`

`sub $s1,$s1,1`

`bnez $s1,loop`

Methods in Objects

Methods are just procedures

- the receiver is simply the first argument
- the rest is just handled at compile-time

What is the compiler doing?

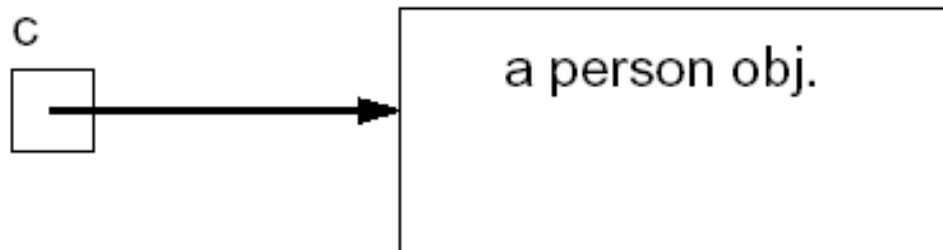
- checking if the method can be applied
- type checking
- replacing this by the first argument
- ...

Memory Management

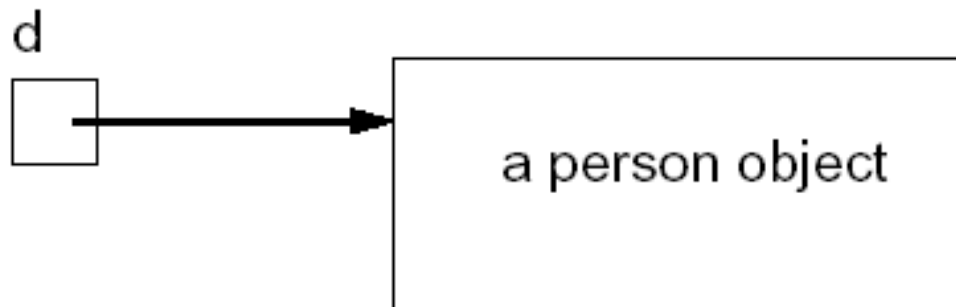
What happens when a new object is created

- Java allocates memory

```
Person c = new Person (...);
```



```
Person d = new Person (...);
```



`c = d`

What your JAVA Teachers Didn't Tell You

and were right not to tell you

- Objects are...
 MEMORY ADDRESSES!
 (no. really.)
- Think of them as synonyms

Compiling Inheritance

```
class Animal {
    void eat();
    void sleep();
}
class Dog extends Animal {
    void eat();
    void bark();
}
class Cat extends Animal {
    void sleep();
    void meow();
}
```

How to compile a method

```
typicalDay(Animal l) {
    l.eat();
    l.sleep();
    l.eat();
    l.sleep();
    ...
}
```

Compiling Inheritance

Each object is associated with a virtual table

- it stores pointers to the code to execute

The table are organized in the same order

- to make sure that inheritance works properly

The cute name for the virtual table is

- VTBL

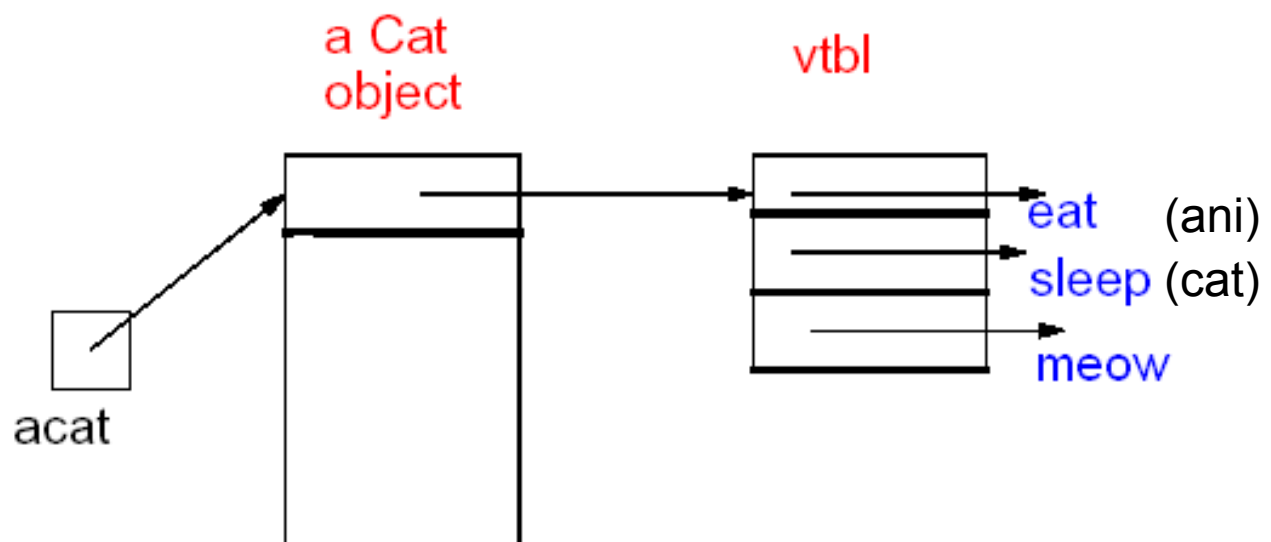
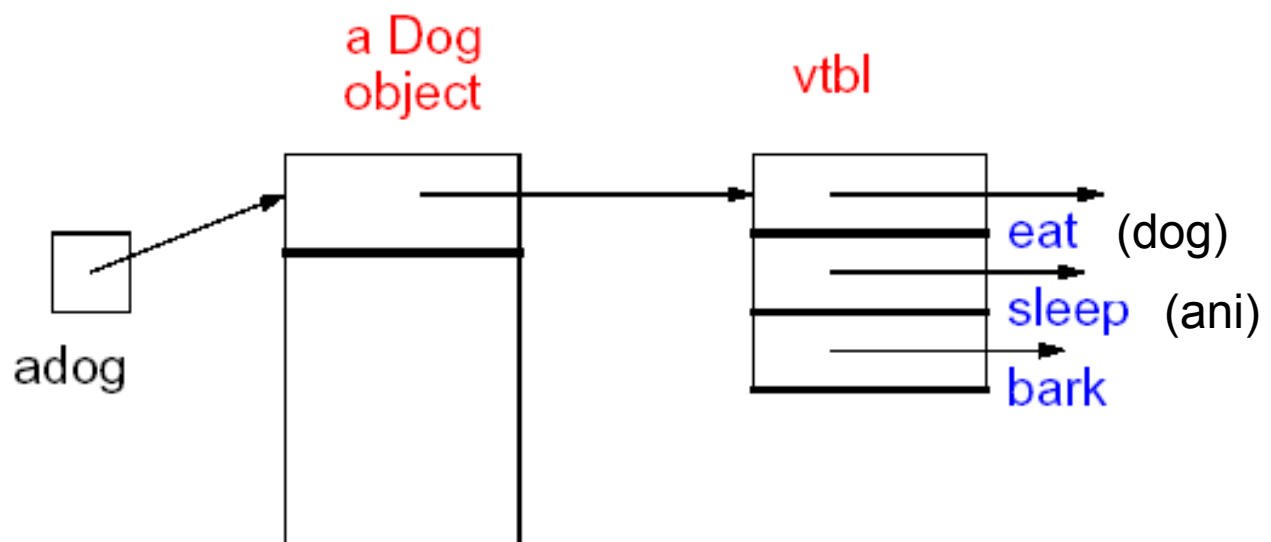
In Java, an object always has

- a pointer to its VTBL
- space for the remaining data

Note that

- The VTBL is the same for all objects

Compiling Inheritance



Compiling Interfaces

```
class Animal {
    void eat();
}
interface Mammal {
    void sleep();
    void eat();
}
class Dog extends Animal
    implements Mammal
{
    void eat();
    void sleep();
    void bark();
}
```

Compiling Interfaces

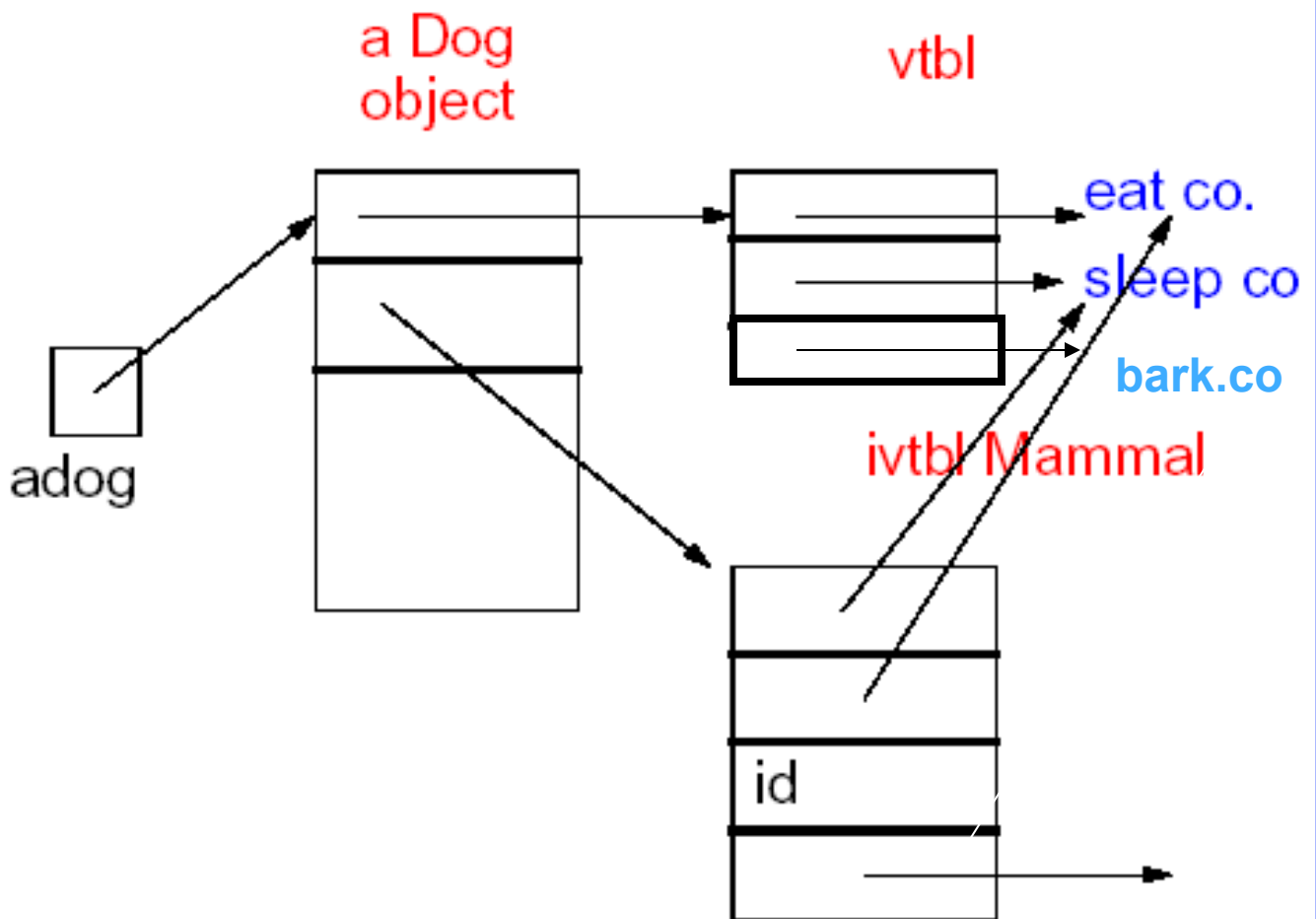
Simplest way

- associate with the object a list of interface tables (ivtbl)
- each ivtbl points to the appropriate code for the object

At call time of an interface (invokeInterface)

- retrieve the IVTBL from the object
- index it to retrieve the proper code

Interfaces



Linked Lists in Assembly

Conventions

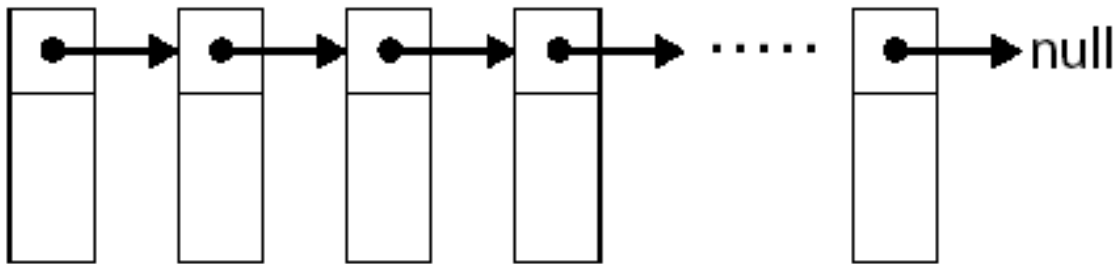
- Probably not what you learned in 15
- More like car/cdr in Lisp

```
class List {  
    private int _key;  
    private List _next;  
    int getKey() { return _key; }  
    List getNext() { return _next; }  
    ...  
}
```

Conventions

- an empty list is represented by Null
- the last element of the list has a Null _next field

Linked Lists (One-Way)

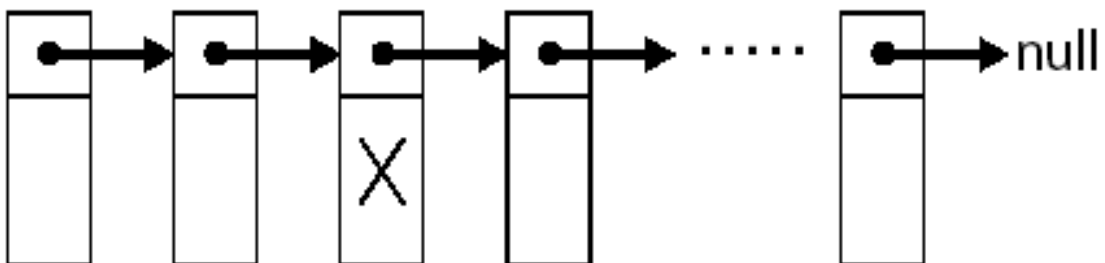


How to represent a linked list in assembler?

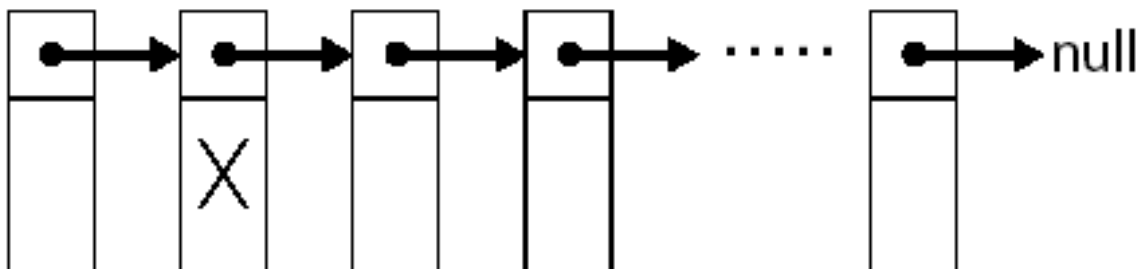
- two fields: a value and a pointer (i.e. an address)
- the object null may be represented by something which is not an address we can use (e.g. 0)

Operations on Lists

- Insert after X



- Delete after X



Search For X

```
// _head is the head of the list;

List search(int x)
{
    List t0 = _head;
    while (t0 != null) {
        if (t0.getKey() == x)
            return t0;
        else
            t0 = t0.getNext();
    }
    return null;
}
```

Search For X

- Assume that X is the key of the element you're looking for

```
# $t0: pointer to current item
# $t1: nil pointer
# $t2: holds key to be searched for
# $t3: key of current item
```

```
        .data
head:   .word          # holds ptr to 1st elt
        next = 0
        key  = 4
        null = 0
```

```
        .text
...
search: lw      $t0, head
        li      $t1, null
        li      $t2, X
loop:   beq     $t1, $t0, failed
        lw      $t3, key($t0)
        beq     $t2, $t3, end
        lw      $t0, next($t0)
        j      loop
failed:
end:
```

- Returns null if X is not found

Insert or Delete an Element

Assume that the address of the element to be inserted is in \$t0, and the address of the element after which we want to insert or delete is in \$t1.

insert:

```
lw $t2,next($t1)      # save where t1 pts
sw $t0,next($t1)      # now t1 pts to t0
sw $t2,next($t0)      # now t0 pts to
jr $ra                # what t1 used to
```

delete:

```
lw $t2,next($t1) # delete this guy
lw $t3,next($t2) # one after delete
sw $t3,next($t1) # now t1 pts to
jr $ra          # one after del
```

What have we left out here?

- memory allocation/deallocation

Insertion

