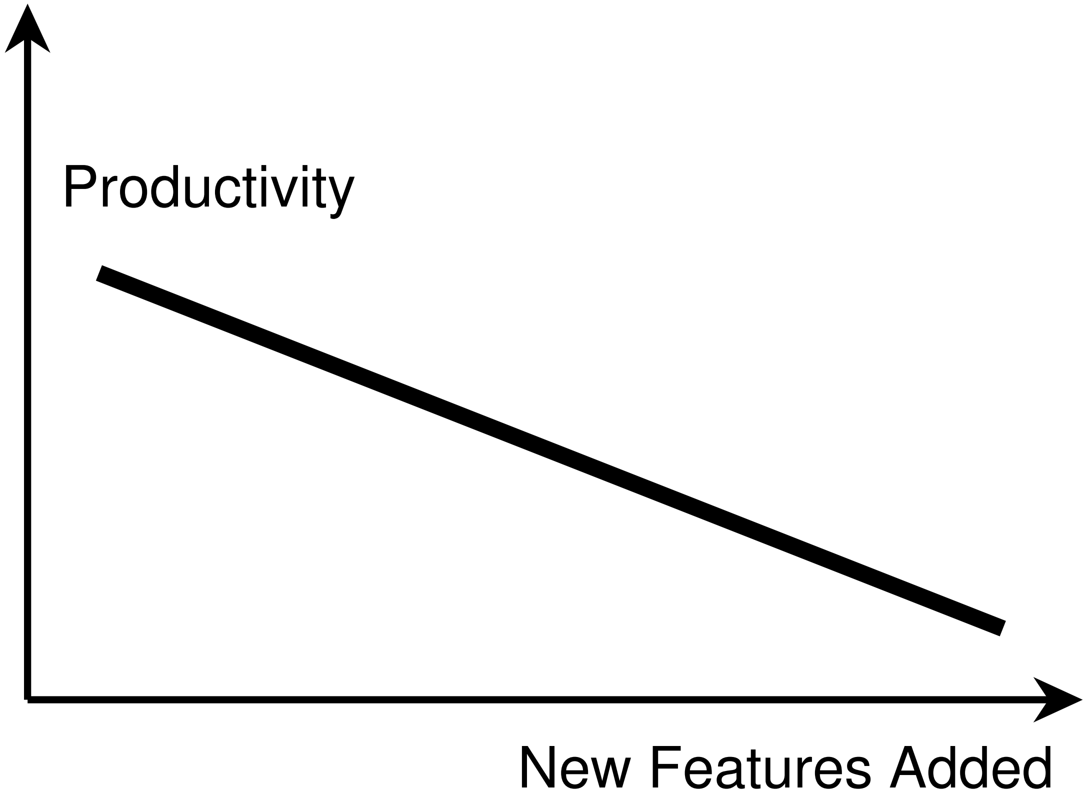
The Software Crisis

If lateness is a disease, the software business is suffering an epidemic.

The most recent generation of programs has grown so massive that it deserves a new name: "bigware".

- Newsweek, April 1989

Falling Productivity



The NeXT Solution

A **seamlessly integrated** software development environment based on object oriented programming (OOP).

A system designed **from the ground up** to take advantage of new software technology in the 1990’s.

Goal:

Create a software development platform which allows developers to create applications using graphical interfaces in about 10% of the time on other systems.

Result:

Application developers have been able to take a multi-year project requiring a large staff of programmers and complete it months with far fewer people.

Applications
Workspace
Interface Builder
Application Kit
Window Server Display Postscript
BSD 4.3/Mach
Hardware

NextStep

Capabilities Seminar

Objectives

Cover the "big picture" of NextStep.

Allow people to understand the **potential** ofthe system.

Teach people the **capabilities** of the system, not the details.

To excite and inspire really great software using the NeXTstep environment.

Seminar Format

First 50 minutes - general audience

Cover basic concepts of Object Oriented Programming and terminology.

break

Second 50 minutes - novice programmer

Syntax of Objective C, sample objective C program. Tour of interface builder.

break

Third 50 minutes - experienced programmer

Detailed disection of a large application.

Discussion and Questions

Advanced Topics.

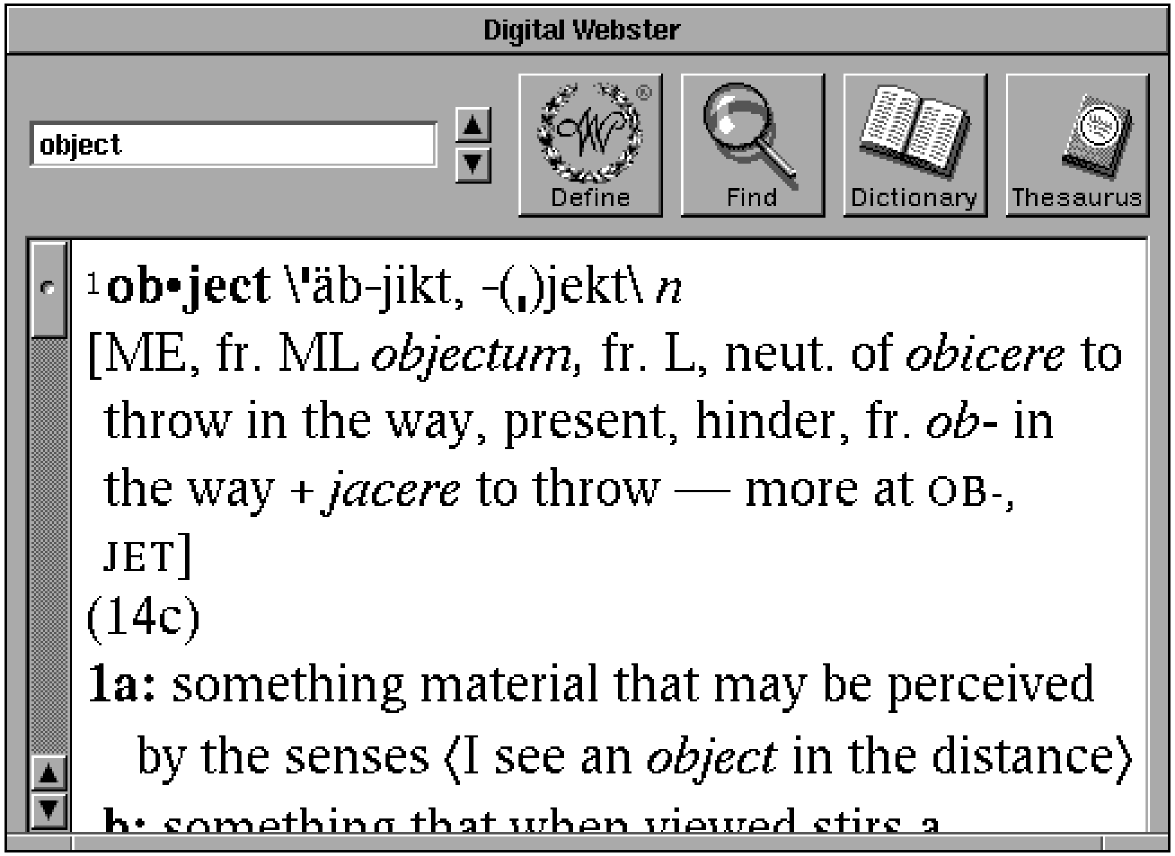
The Big Picture

What is object oriented programming?

What are the advantages of OOP?

What techniques are used?

What is an Object?



What is Object Oriented Programming (OOP)?

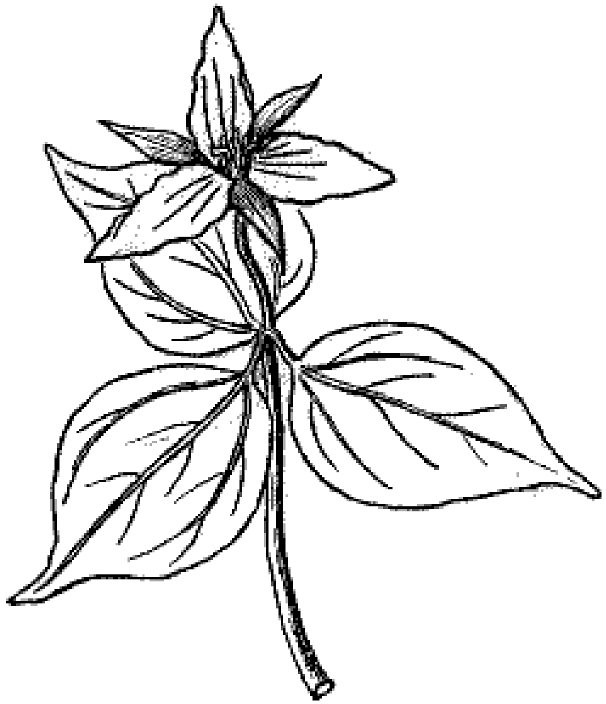
Process of creating computer based objects which are analogs of the real world.

A Top-Down design methodology

A process of continually partitioning abstractions into reusable structures.

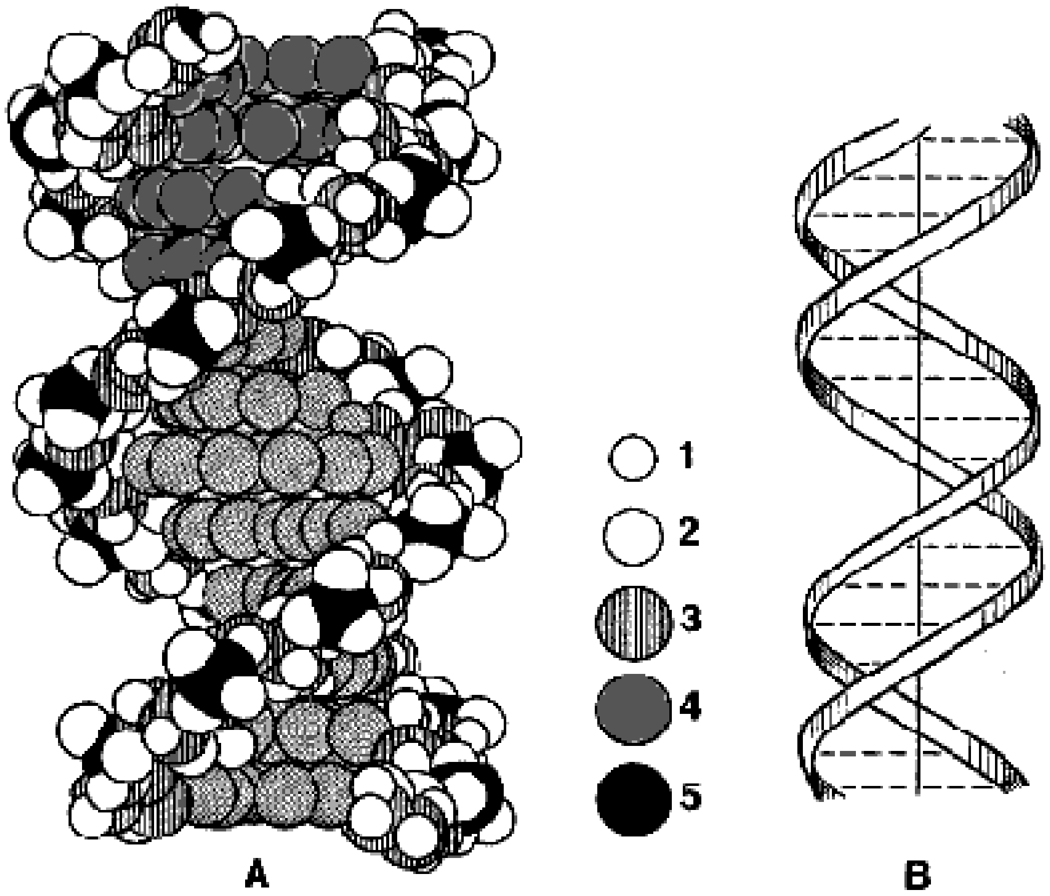
Preserve this decomposition for life of application.

Plant Example

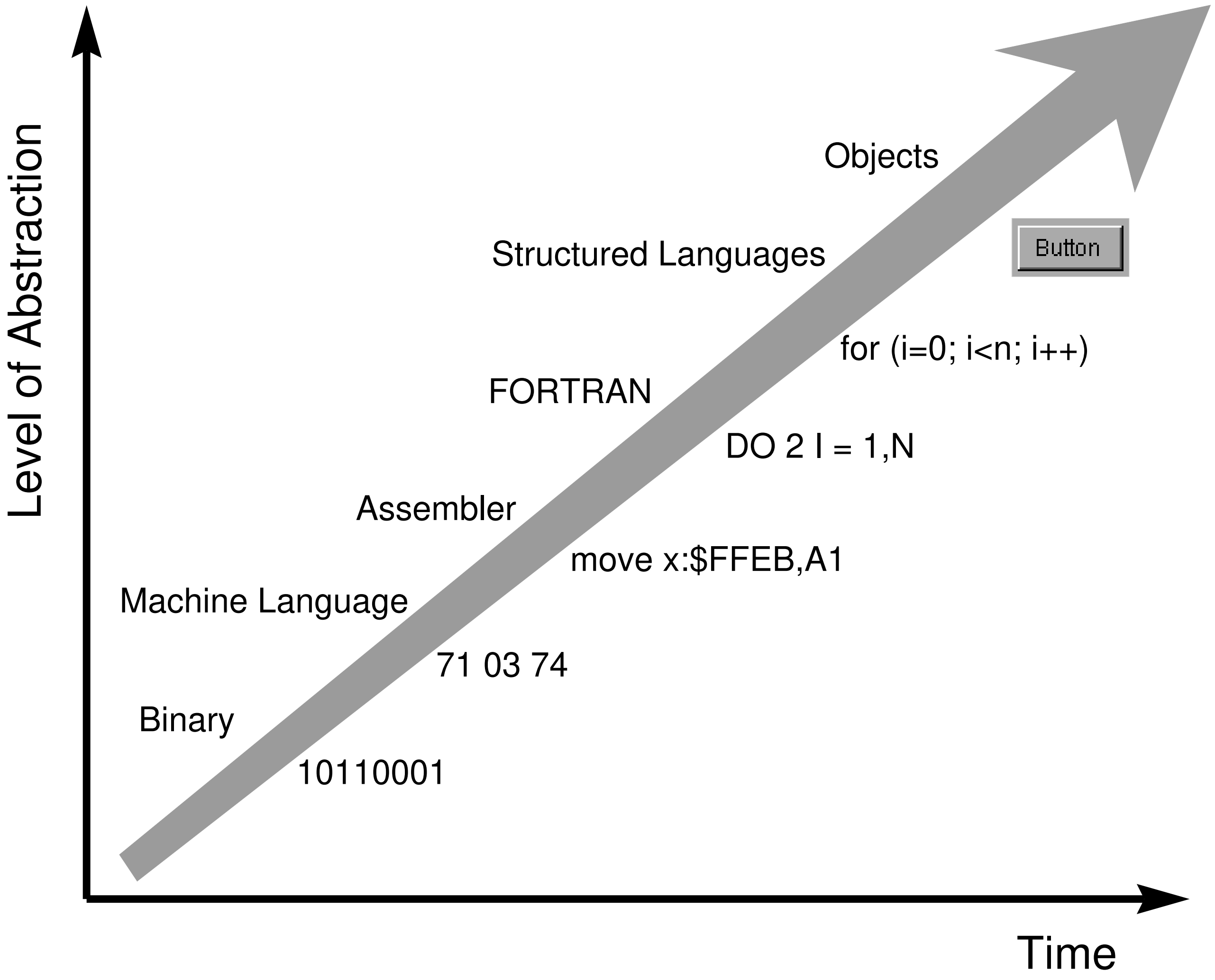


leaves stem flower

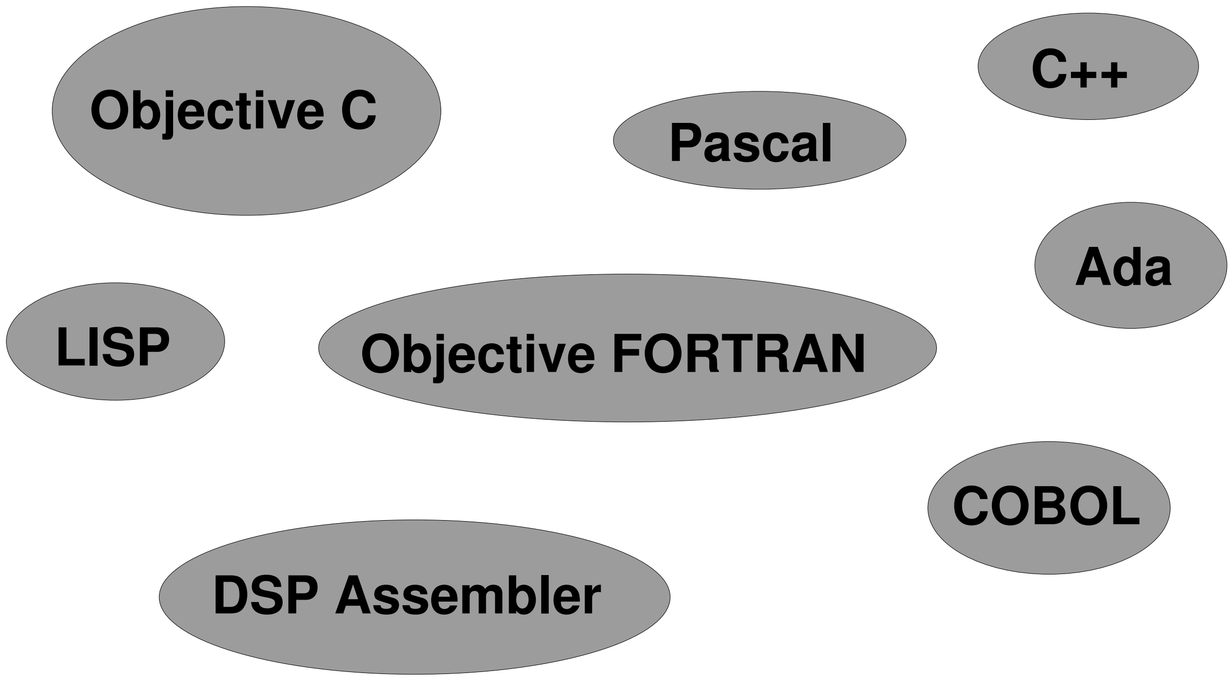
Low Level Data Structures



OOP is Part of the Natural Trend to Use Higher Levels of Abstraction Over Time



Core routines can be multi-lingual



Problems can be expressed the language that suits them best

Objects can be distributed among processors on a board, boards on a system, or workstations on network.

Other OOP Benefits

Encourages code re-use rather than re-invention.

Encourages a seperation of prototyping and code polishing.

Enables you to create software that can be readily comprehended and shared with others.

Encourages separation of user interface and core routines.

Death of Application Based Computing

Monolithic, stand-alone applications are no longer competitive.

Object palettes in the hands of the system integrator provide custom solutions to specific problems.

The Bottem Line:

Object based computing provides superior:

- Personalization
- Integration
- Specialization

Four OOP Techniques

1) Encapsulation

2) Inheritance

3) Messaging

4) Dynamic Binding

Encapsulation

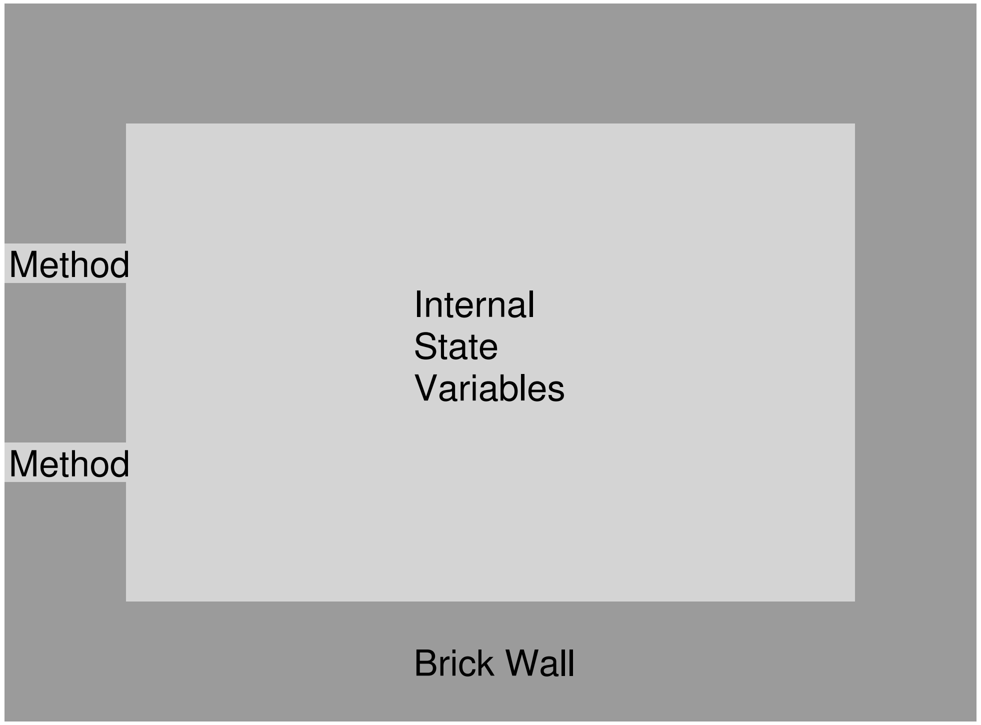
(information hiding, data abstraction)

Way of grouping data **and procedures** to access that data together into a single object.

Users of these objects can **only** access the objects using the procedures the developer provides.

Controlled access.

Object



Internal states are called the *instance variables*.

The set of functions used to access the instance variables are called *methods*.

Benefits of Encapsulation

Insulation and protection of data

Enhanced reliability

Transparent enhancements

Ability to create "views" of objects

Views of Subroutine Libraries

Users see black boxes

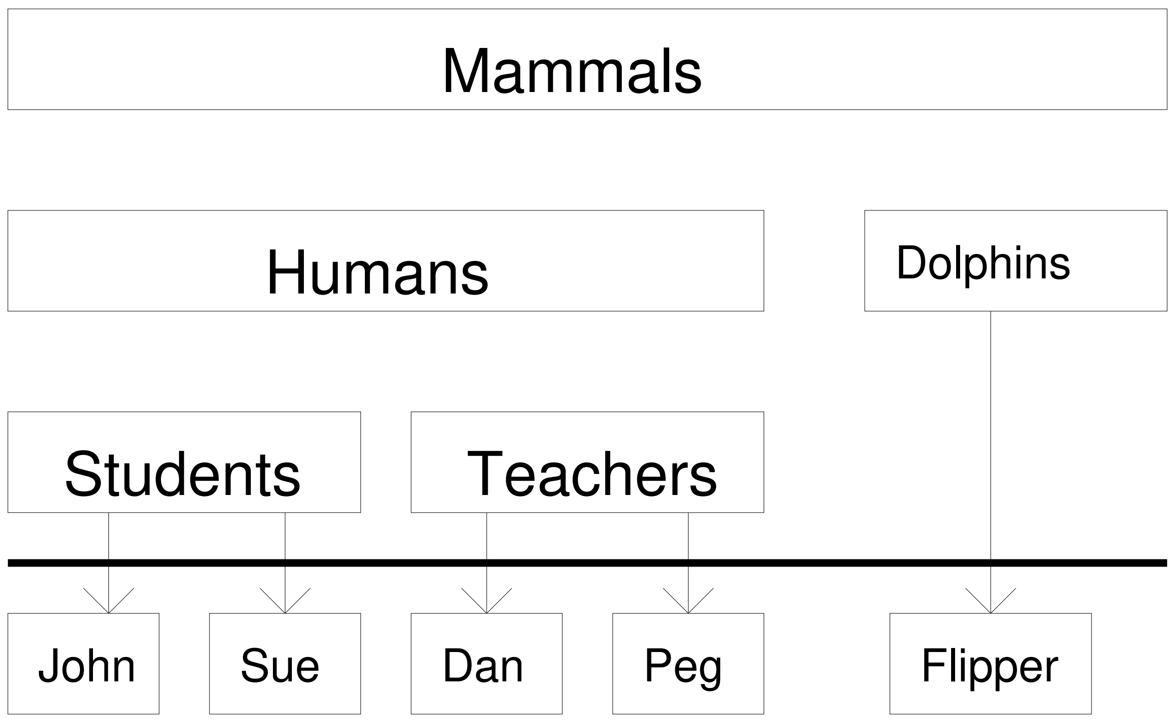
Users can "point and click" to use objects

Moves burden of correctness from
consumers to **producers** of objects.

Accessibility rises dramatically

Inheritance

Objects are always defined relative to some other class, called its *superclass*.



Sample Inheritance Hierarchy

Inheritance (continued)

An object inherits copies of instance variables as well as the methods defined in the superclass.

A new object may add additional instance variables and methods.

A new object may over-ride (re-define) methods defined in a superclass.

Benefits of Inheritance

It is easy to define a new object that is just like an old object except for a few minor differences.

Almost never start from scratch when building a new object.

Principle factor in enhancing the reusability of objects.

Allows users to customize objects to meet application specific needs.

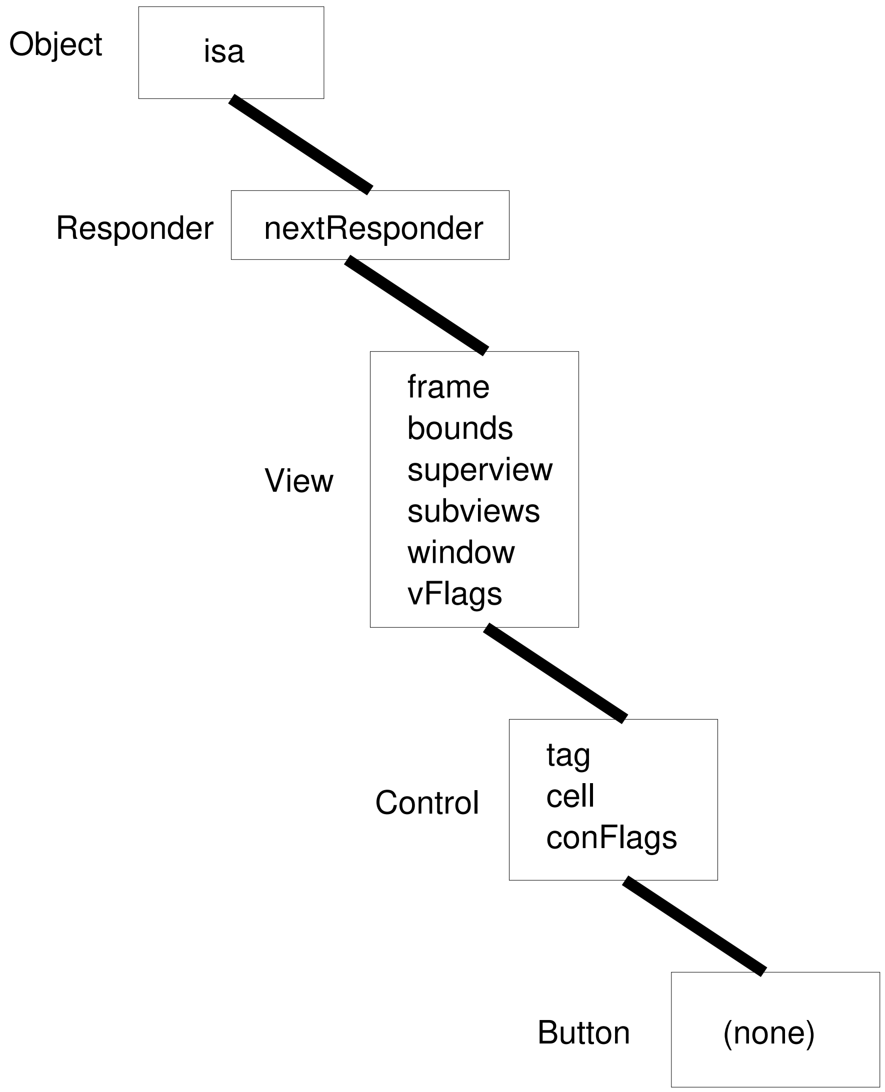
See /NextLibrary/Documentation/NeXT/SysRefMan/21ClassSpecs/Appkit/Button.wn

Button

INHERITS FROM	Control : View : Responder : Object

INSTANCE VARIABLES

Inherited from Object	struct _SHARED	*isa;
Inherited from Responder	id	nextResponder;
Inherited from View	NXRect	frame;
	NXRect	bounds;
	id	superview;
	id	subviews;
	id	window;
	struct __vFlags	vFlags;
Inherited from Control	int	tag;
	id	cell;
	struct _conFlags	conFlags;
Declared in Button	(none)	



Button Inheritance Hierarchy

Messaging

An object can ask another object to perform one of its methods or actions via a message.

A message statement contains:

1) A reference to the object which is to be called (the receiver).

2) The name of the method to be executed

3) Any arguments if required.

Messaging (continued)

Example:

```
[Webster defineWord:myWord]
```

target object: `Webster`

message name (note colons): `defineWord:`

argument: `myWord`

Benefits of Messaging

Different objects can respond differently to the same message. (*Polymorphism)*

Example: message of "draw".

Main program tells all objects to draw themself.

"Line" object does one thing.

"Circle" object does another.

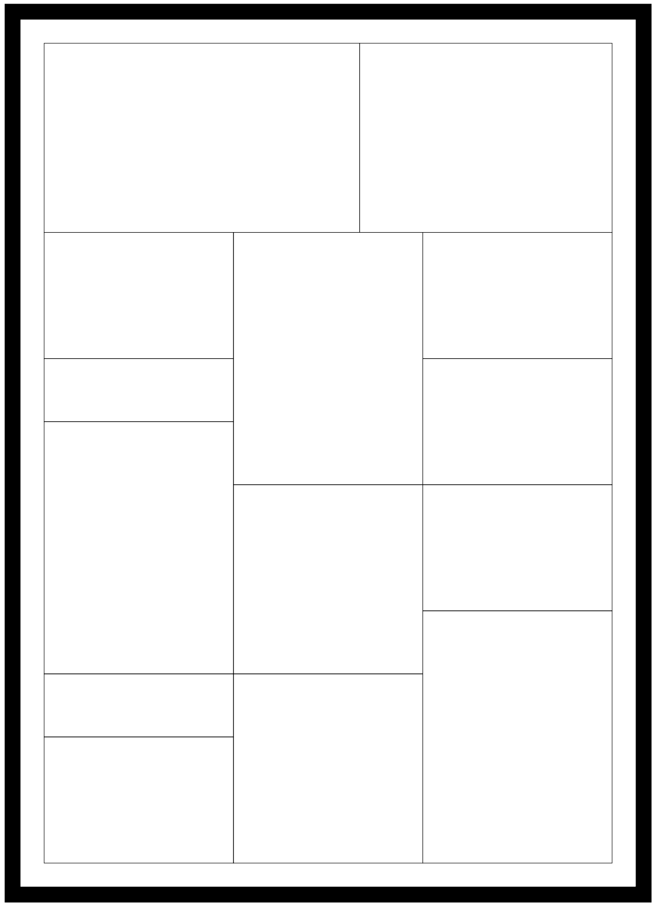
Adding a new object is easy: you provide the "draw" method with the new object and you don’t change the original main program code!

Dynamic Binding

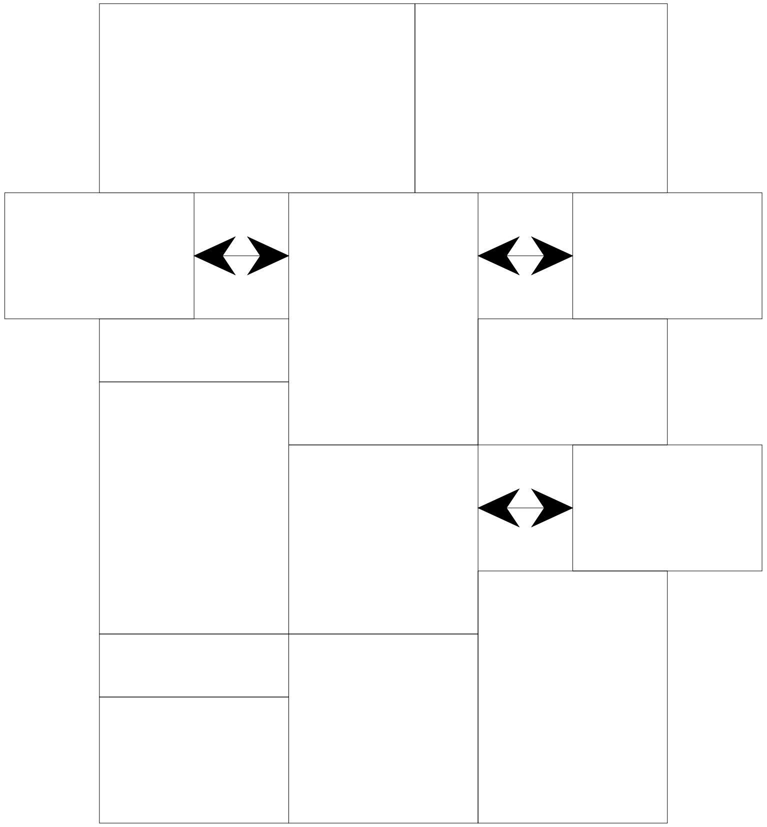
(late binding, run-time binding)

Method of making decisions about what methods to call after compile time.

Binding of a message to a target is done at runtime.



Statically Bound Subroutines



Dynamically Bound Subroutines

Constraints Introduced by Compilers

Benefits of Dynamic Binding

Allows developers of programs to do rapid prototypes and not have to anticipate every future type of object.

Allows users the ability to customize pre-compiled applications at run time.

Allows users to change programs without needing access to source code.

Allows software developers to give end users flexibility without revealing internal algorithms.

Promotes tool kits rather then large, monolithic, standalone programs.

Summary

Encapsulation

Makes data secure

Inheritance

Makes it easy to share and extend objects

Messaging

Flexible way to communicate between objects

Runtime Binding

Allows flexibility at runtime