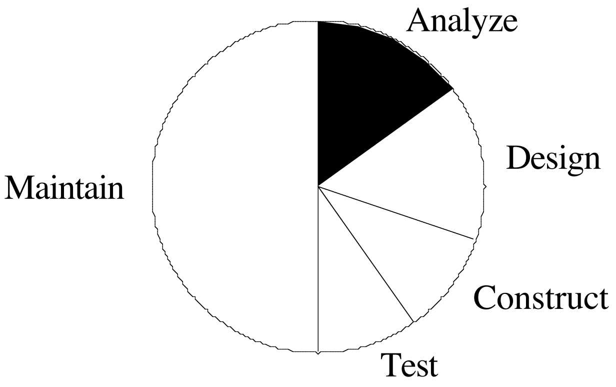
OOP?

- What is OOP?
- Why?
- OOP in a nutshell

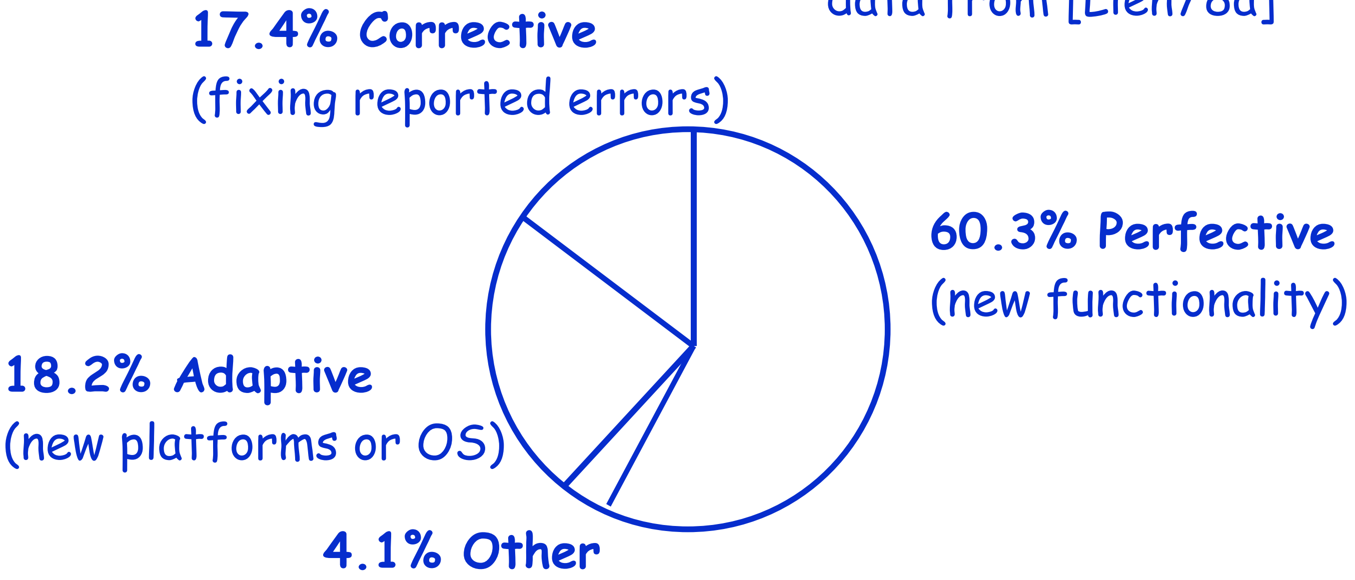
Reality on Software Development



What is important?

- maintainability
- extensibility
- understandability

Maintenance = Evolution



data from [Lien78a]

17.4% Corrective
(fixing reported errors)

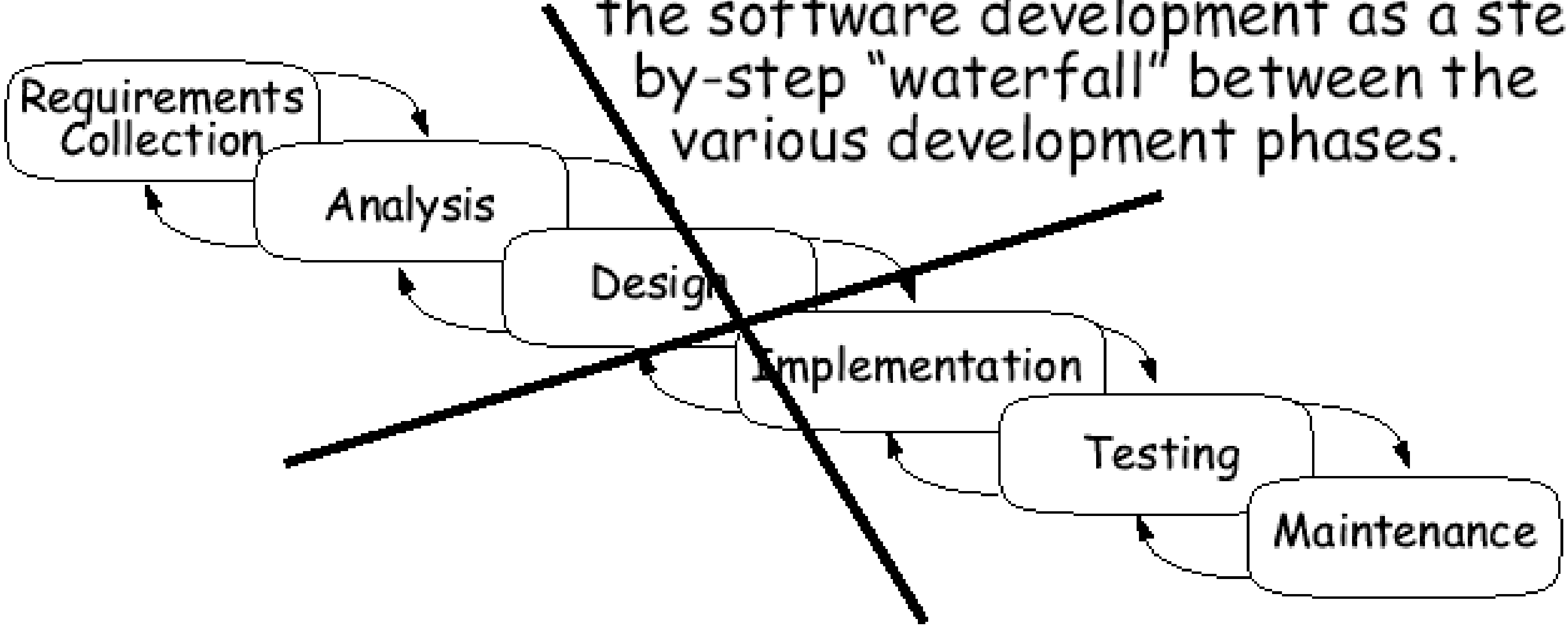
60.3% Perfective
(new functionality)

18.2% Adaptive
(new platforms or OS)

4.1% Other

The bulk of the maintenance cost is due to *new functionality*
=> even with better requirements, it is hard to predict new functions

The Waterfall Model is dead



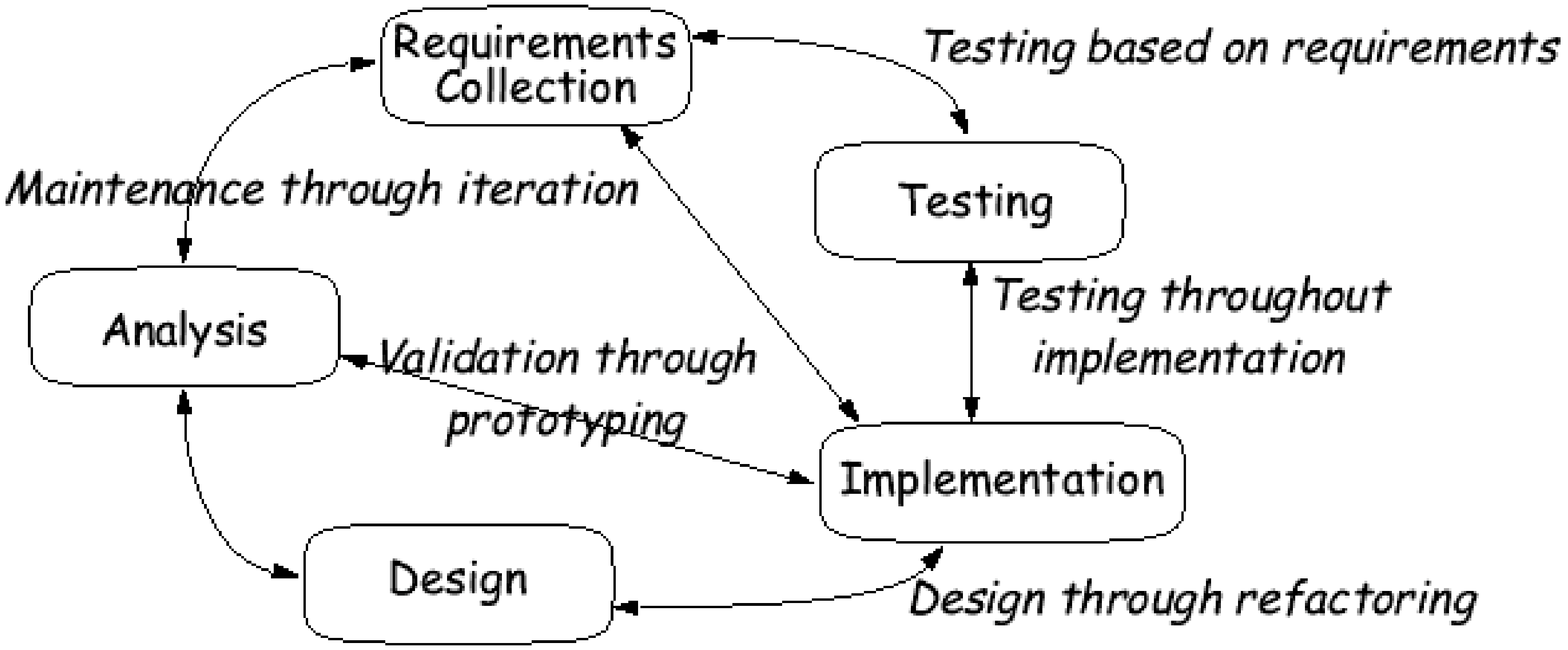
The classical software lifecycle models the software development as a step-by-step "waterfall" between the various development phases.

The waterfall model is unrealistic for many reasons, especially:

- ❑ requirements must be "frozen" too early in the life-cycle
- ❑ requirements are validated too late

Iterative Development

In practice, development is always iterative, and *all* software phases progress in parallel.


Requirements Collection
Testing based on requirements
Testing
Maintenance through iteration
Analysis
Validation through prototyping
Testing throughout implementation
Implementation
Design
Design through refactoring

Lehman's Laws of Software Evolution

Continous Change: "A program that is used in a real-world environment must change, or become progressively less useful in that environment."

Software Entropy: "As a program evolves, it becomes more complex, and extra resources are needed to preserve and simplify its structure."

The Old Way

Computer system consists of data and programs.

Programs manipulate data.

Programs organized by

- functional decomposition
- dataflow
- modules

New Paradigm

- Computer system consists of a set of objects.
- Objects are responsible for knowing and doing certain things.
- Objects collaborate to carry out their responsibilities.
- Programs organized by classes, inheritance hierarchies and subsystems

Accidental vs. Essential Complexity

- Assembly is perfect to write 8k programs!
- But we need abstraction tools to model the complexity of the world
- Object-oriented programming in only one way
  - Reactive languages,
  - Relational languages,
  - Logic Languages, … are others
- OOP helps reducing the accidental complexity not the essential
- Bad OO programs are also difficult to understand, extend, and maintain

What is an object, anyway?

Mystical view

Computing systems are made up of objects that communicate only by sending messages between each other. All computation is message sending.

What is an object, anyway?

Scandinavian view

A program is a simulation. Each entity in the system being simulated is represented by an entity in the program.

What is an object, anyway?

Programming language view

An object-oriented system is characterized by

- data abstraction
- inheritance
- polymorphism by late-binding of procedure calls

Modeling

All phases of software life-cycle are modeling

- analysis - modeling of problem
- design - modeling of solution
- implementation - making model run on a computer
- maintenance - fixing/extending your model

Modeling

Claim: people model the world with "objects"

- objects
- classes
- relationships between objects
- relationships between classes

Modeling

Advantages of object-oriented software development

- more natural - matches the way people think
- single notation - makes it easy to move between software phases

Objects and Relationships

John is Mary's father.

Mary is John's daughter.

Bob is Mary's dog.

Mary is Bob's owner.

Ann is John's employer.

John is Ann's employee.

Objects and Attributes

John's name is "John Patrick O'Brian".

John's age is 27.

John's address is 987 N. Oak St, Champaign IL 61820

What about John's employer?   John's wife?

What is an attribute, and what is a relationship?

Objects and Behavior

John goes on a trip.

John makes reservations.

John buys tickets.

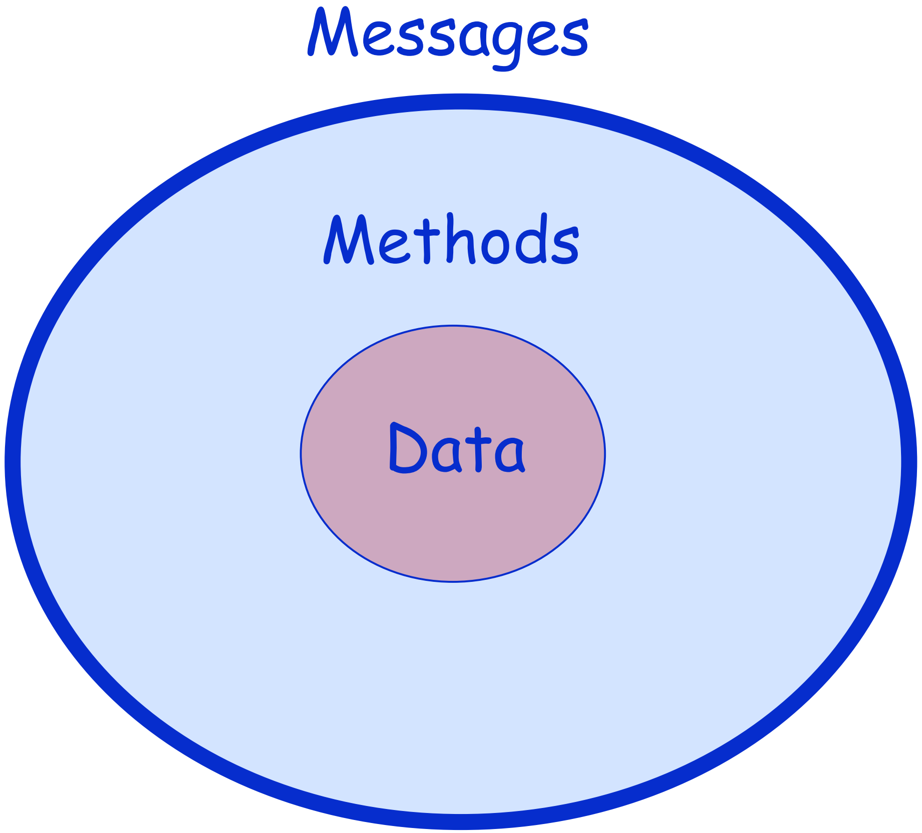
John travels by airplane.

John checks into hotel.

What is really an object?

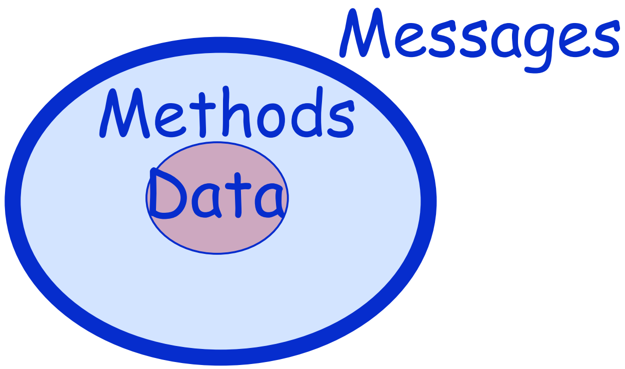
- Anything we can talk about can be an object, including relationships ("the husband of the first party", "first-born son").
- What are we trying to model?
- Models should be as simple as possible, but no simpler.
- Models are dictacted by domains

Object



Object: Behavior + State + Control

- What: Messages
  - Specify what behavior objects are to perform
  - Details of how are left up to the receiver
  - State information only accessed via messages
- How: Methods
  - Specify how operation is to be performed
  - Must have access to (contain or be passed) data
  - Need detailed knowledge of data
  - Can manipulate data directly



Classification

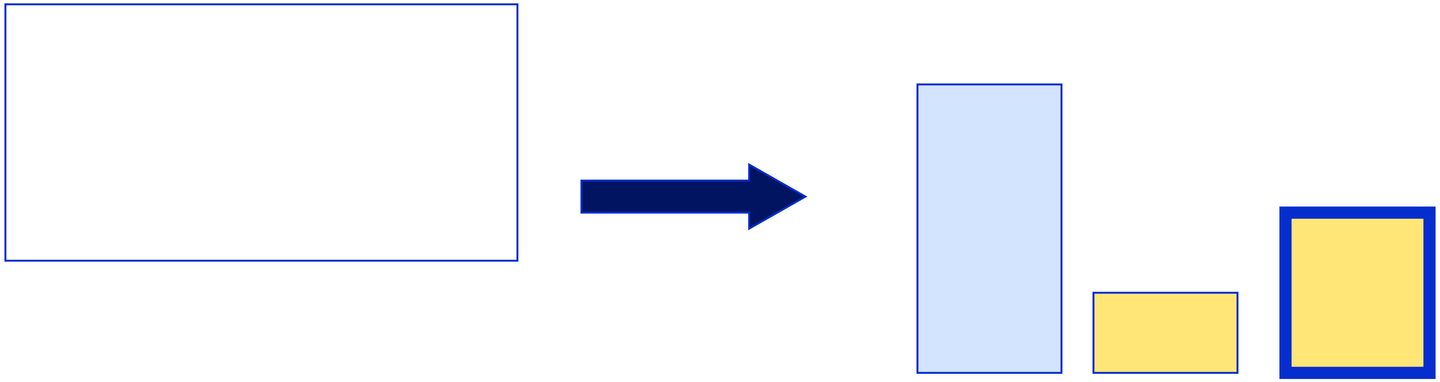
We naturally put objects into classes that have similar characteristics.

- John is a man.
- Mary is a woman.
- Bob is a dog.
- All women are people.
- All people are mammals.

Classes: Factory of Objects

- Reuse behavior

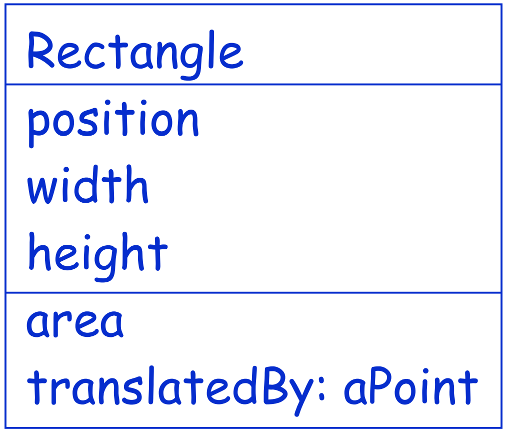
  => Factor into class
- Class: “Factory” object for creating new objects of the same kind
- Template for objects that share common characteristics



Class: Mold of Objects

- **Describe** state but not value of all the instances of the class
  - Position, width and height for rectangles
- **Define** behavior of all instances of the class

```
area
    ^ width * height
```



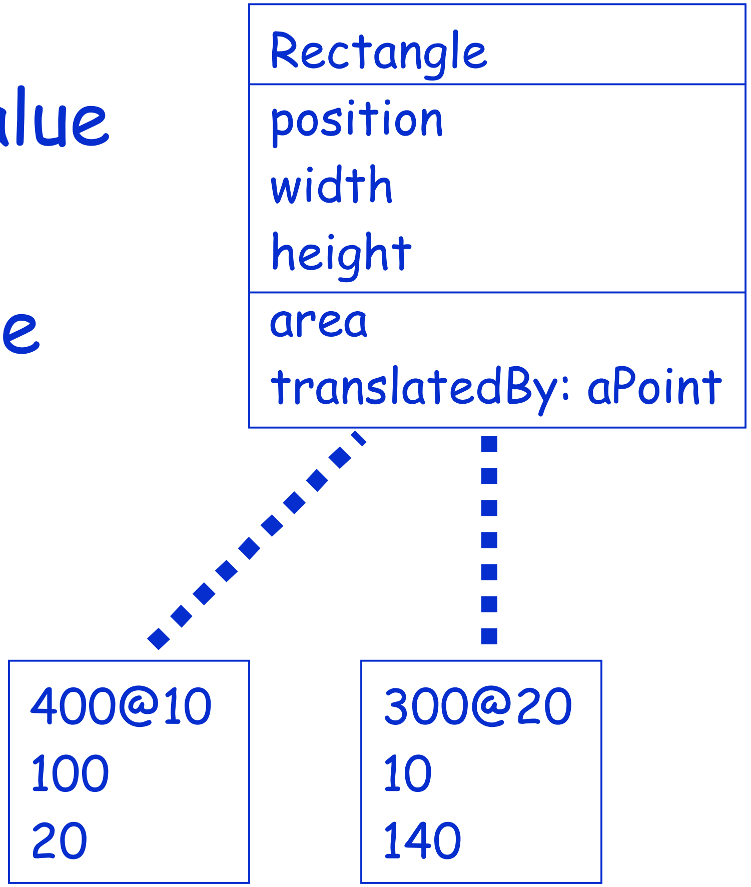
Rectangle

position
width
height

area
translatedBy: aPoint

Instances

- A particular occurrence of an object defined by a class
- Each instance has its own value for the instance variables
- All instances of a class share the same methods



How to Share Specification?

- Do not want to rewrite everything!
- Often times want small changes
- Class hierarchies for sharing of definitions
- Each class defines or refines the definition of its ancestors
- => inheritance

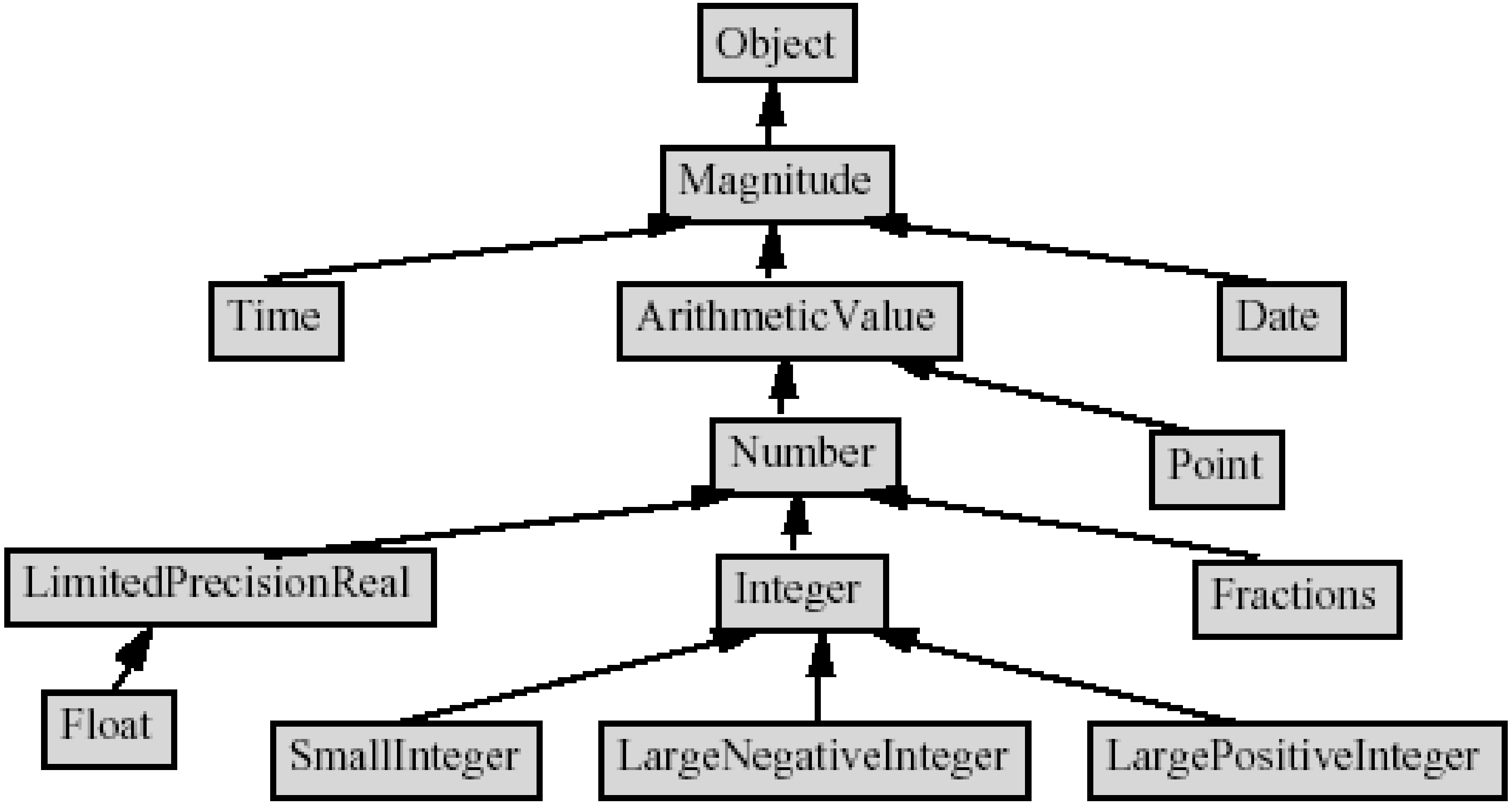
Inheritance

- New classes
  - Can add state and behavior
  - Can specialize ancestor behavior
  - Can use ancestor's behavior and state
  - Can hide ancestor's behavior

  To existing ones

- Direct ancestor = superclass
- Direct descendant = subclass

Comparable Quantity Hierarchy



Summary

Objects

- have identity
- have attributes
- have behavior
- have relationships with other objects

Summary

Classes

- **Describes** the attributes, and relationships of a set of objects
- Define the behavior of a set of objects
- Reuse, extend, specialize behavior from other classes
- Subclasses/superclasses form graph of generalizations