Object-Oriented Programming

Stéphane Ducasse
Stephane.Ducasse@univ-savoie.fr
http://www.iam.unibe.ch/~ducasse/

Outline

- Context: Software ***MUST*** evolve
- OOP Modeling
- Objects
- Classes
- Inheritance

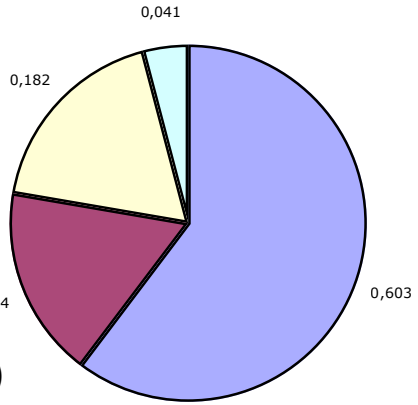
Continuous Development

Relative Maintenance Effort

Between 50% and 75% of global effort is spent on "maintenance" !

4.1% Other

18.2% Adaptive
(new platforms or OS)



data from [Lien78a]

17.4% Corrective
(fixing reported errors)

60.3% Perfective
(new functionality)

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

Lehman's Laws

A classic study by Lehman and Belady [Lehm85a] identified several "laws" of system change.

Continuing change

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

Increasing complexity

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

Those laws are still applicable to brand ***new object-oriented applications***

Software is living…

- Early decisions may have been good at that time
- But the context changes
- Customers change
- Technology changes
- People change

- Successful software ***MUST*** evolve

The Old Way

- Computer system consists of data and programs.
- Programs manipulate data.
- Programs organized by
  - functional decomposition
  - dataflow
  - modules

OOP

- 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

Outline

- Context: Software ***MUST*** evolve
- OOP Modeling
- Objects
- Classes
- Inheritance

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 and classes
- relationships between objects
- relationships between classes

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 dictated by domains

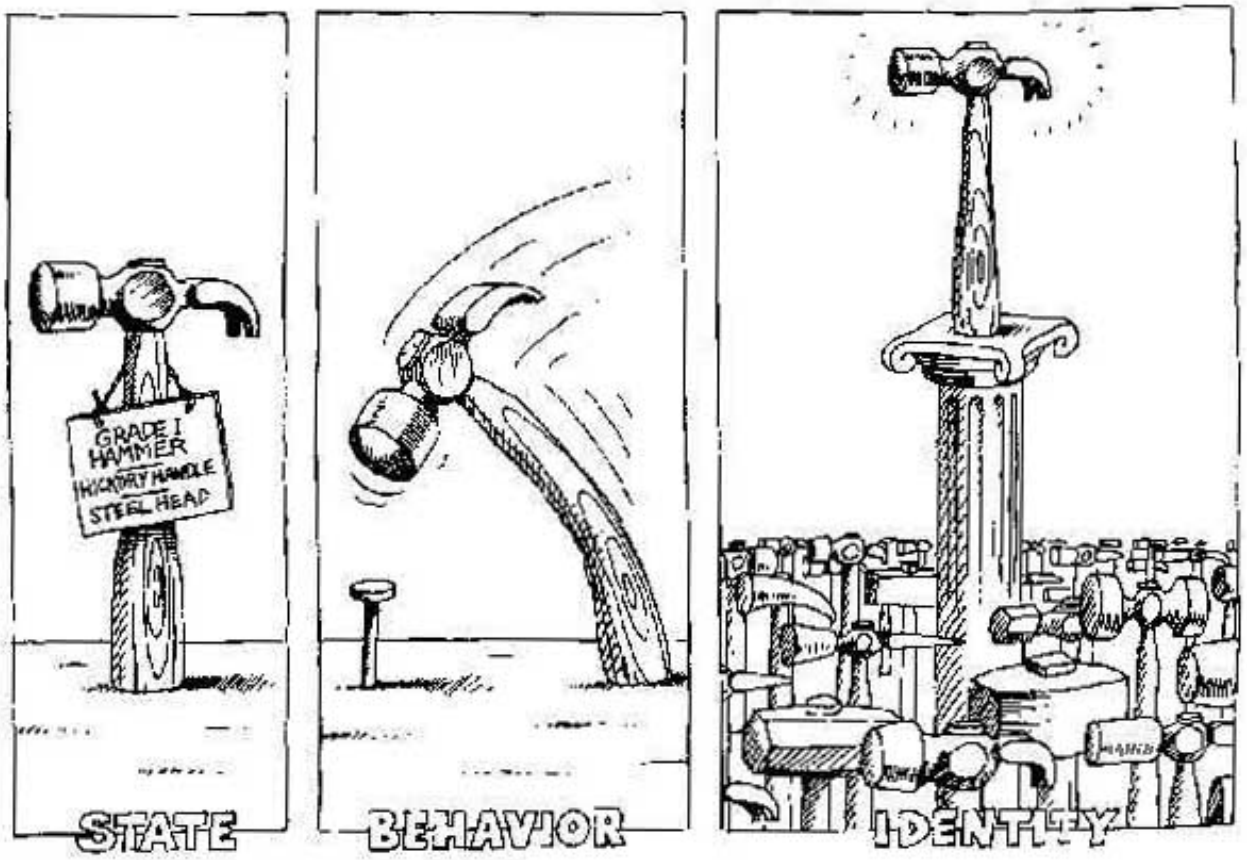
Some Examples

- Things that we can manipulate or see: Book, BD
- Things that we can conceptually see: Process, Mortgage, InsuranceContract, Socket,
- Rôles: Mother, Teacher,
- Drivers, Algorithms
- Events and transactions
- Library elements: Color, Window, Text, Dictionary, Date, Boolean....
- Organizations, processes

RoadMap

- Context: Software ***MUST*** evolve
- OOP Modeling
- Objects
- Classes
- Inheritance

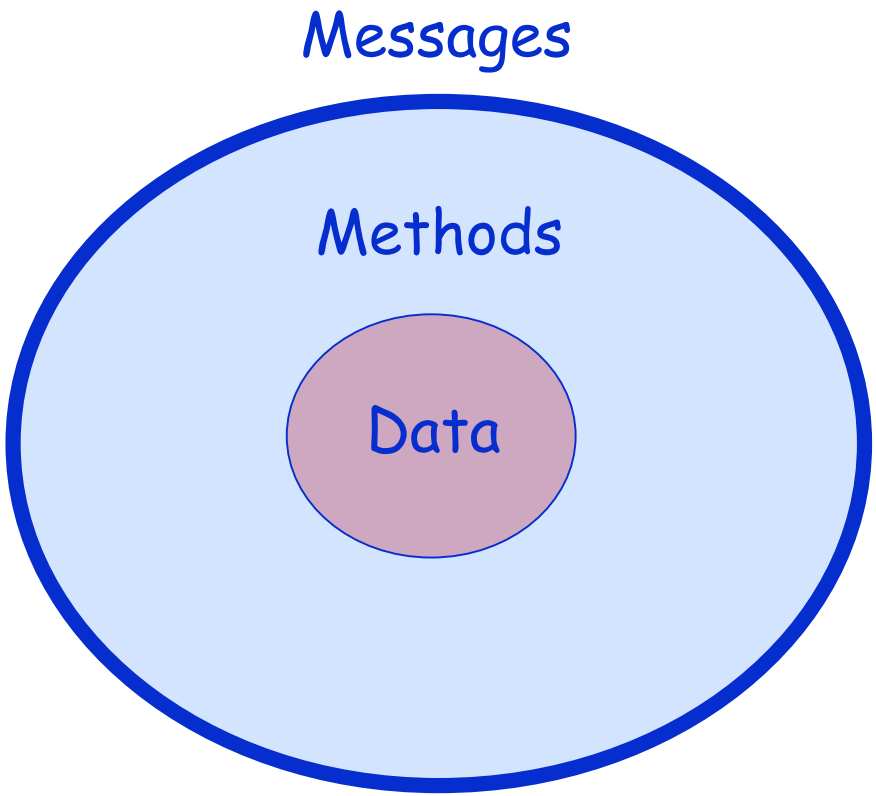
State + Behavior + Identity



State + Behavior + Identity

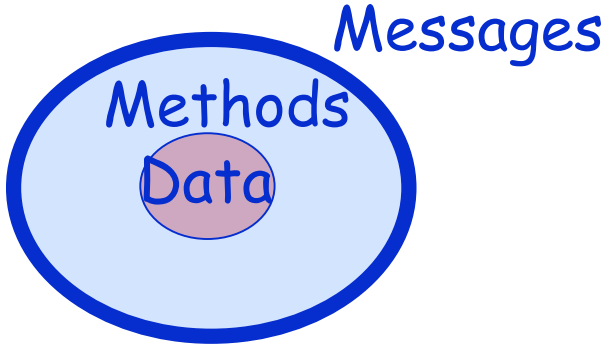
- State: Objects it contains or refers to
  - Ex: point location
- Behavior: an object understands a given set of messages
- Identity: an object can be the same (of the same class) than another one but it has still a different identity (location in memory)

Object



Behavior + State + Control

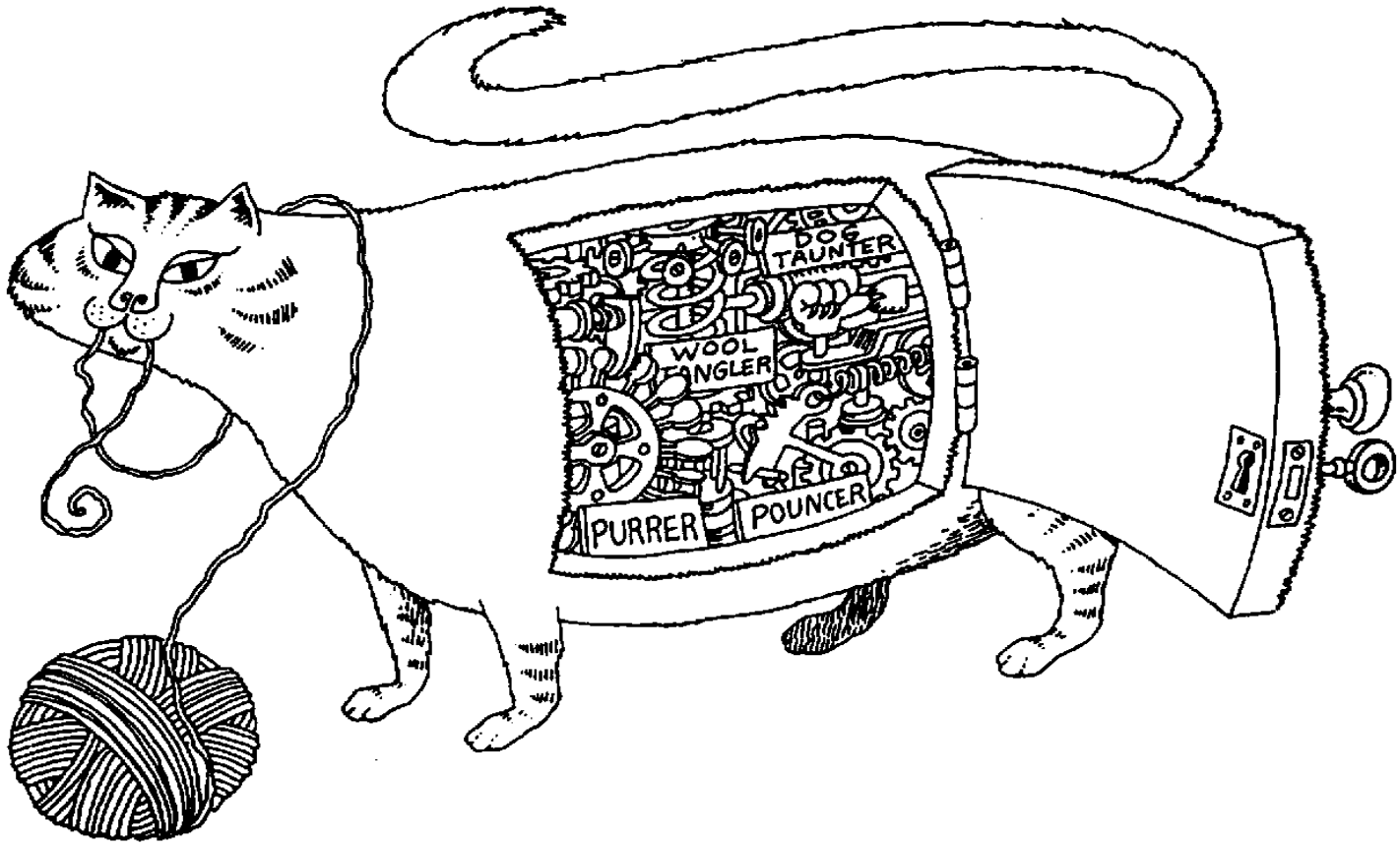
- What: Messages
  - Specify what behavior objects are to perform
  - Details of how are left up to the receiver
  - State information only accessed via messages
- How: Methods
  - Specify how operation is to be performed
  - Must have access to (contain or be passed) data
  - Need detailed knowledge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



Equality and Identity

- I want to eat the pizza that you are eating
- Equality: I want to eat the "same" kind of pizza
- Identity: I eat your pizza

Encapsulation



Encapsulation

- Object protects its data
  - We cannot access *private* data
- Object protects its implementation choice
  - Clients use a public *interface*
  - Object can change its *implementation*

Object Summary

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

Objects
- protect their data
- offer services, protect their implementation

Roadmap

- Context: Software ***MUST*** evolve
- OOP Modeling
- Objects
- ***Classes***
- Inheritance

Classification

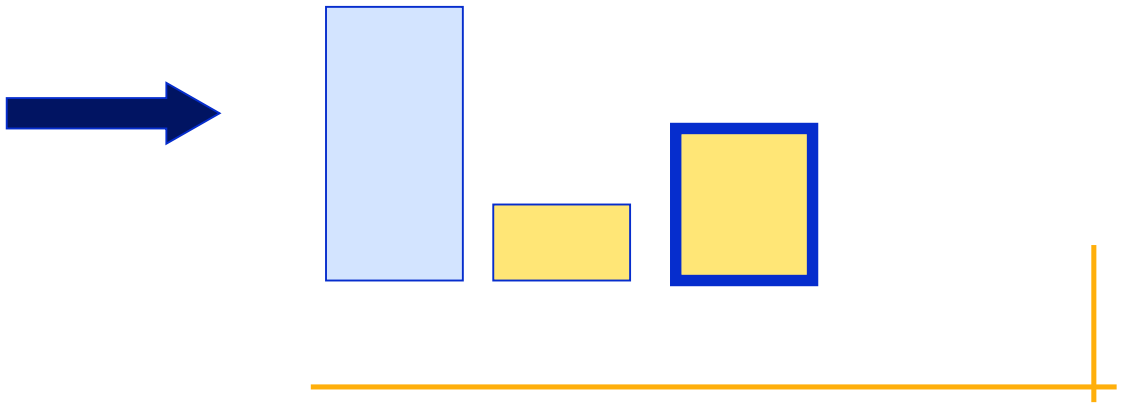
- We naturally put objects into classes that have similar characteristics.
  - John is a man.
  - Mary is a woman.
  - Bob is a dog.
  - All women are people.
  - All people are mammals.

A Class generates Instances

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

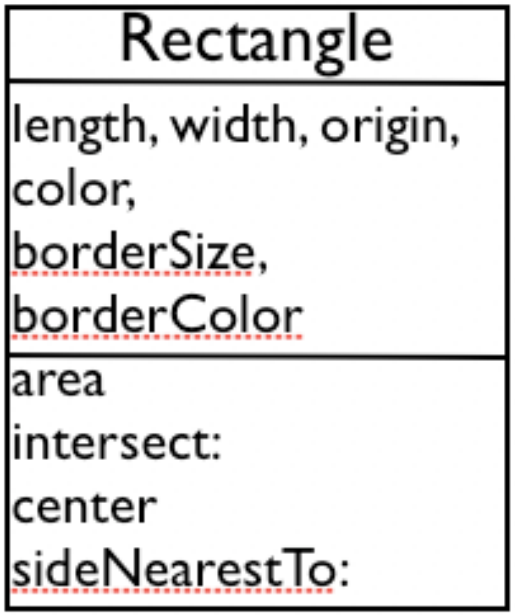
Rectangle
length, width, origin, color, borderSize, borderColor
area intersect: center sideNearestTo:



Class: Mold of Objects

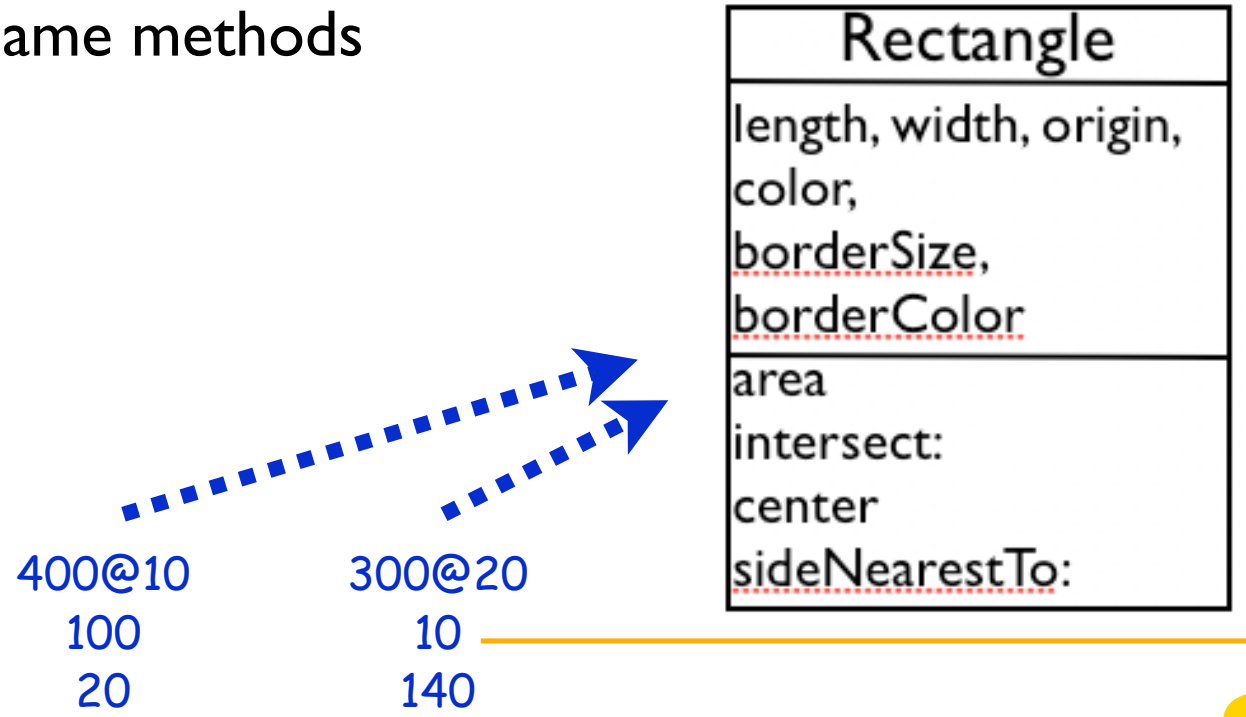
- **Describe** state but not ***effective*** values of all the instances of the class
  - Position, width and length for rectangles
- **Define** behavior of all instances of the class

```
Rectangle>>area
    ^ width * height
```



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

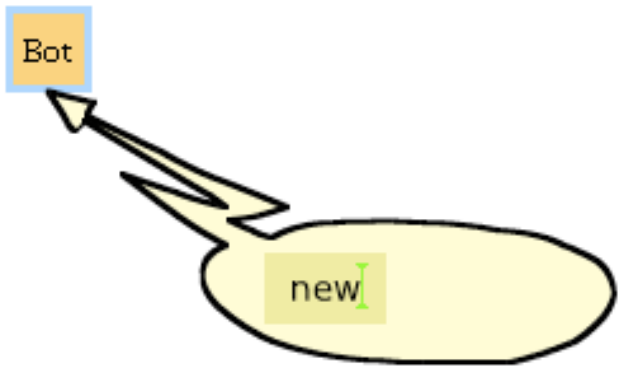


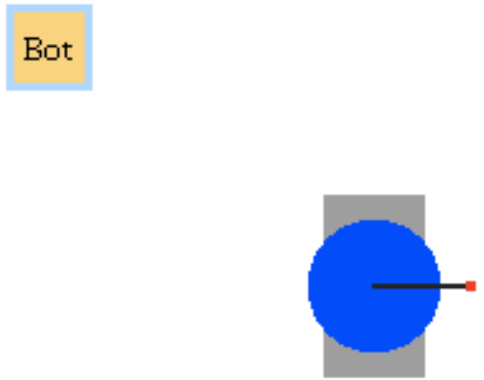
An example

- Let's control robots....
- http://smallwiki.unibe.ch/botsinc/

Instances Creation

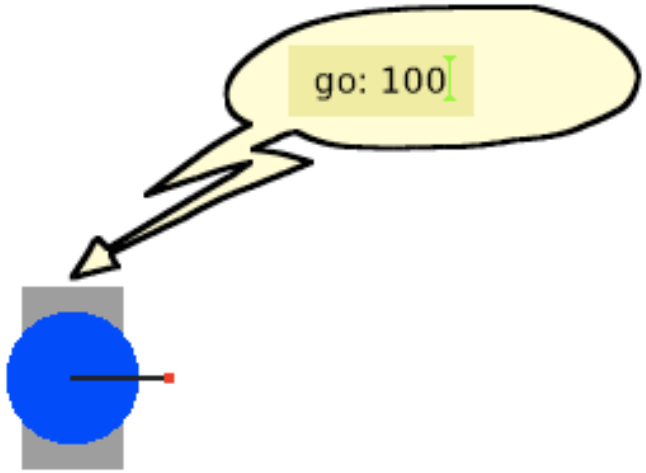
- Asking a class to create an instance
- The Bot **factory** creates a **new** robot

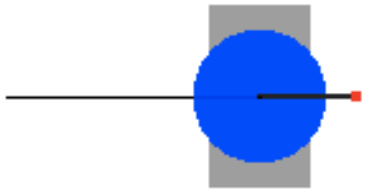




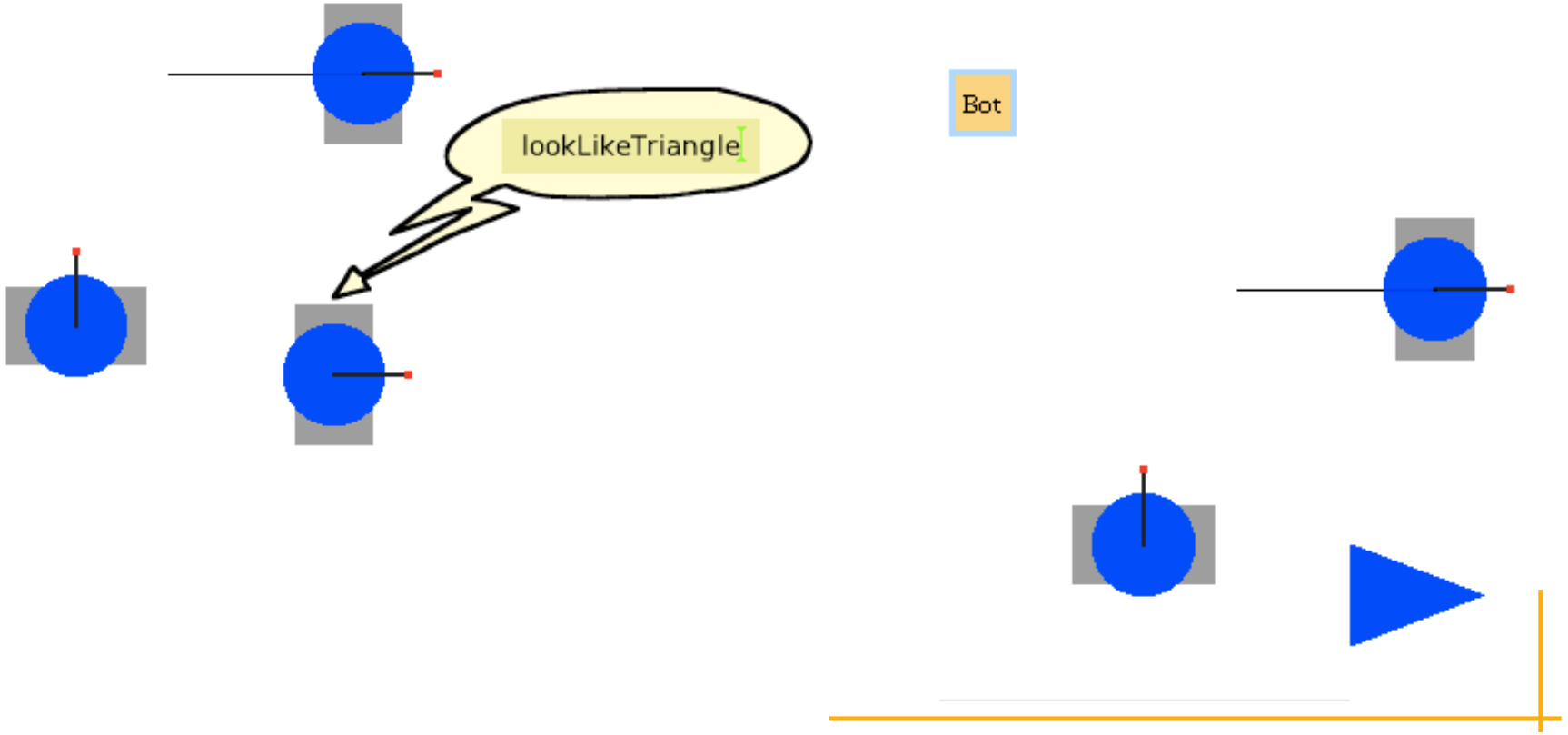
Instance Interaction

- We send messages to individual robots too





Instances are autonomous...



Instances vs. Classes

- The class Bot does **not** understand **go: 100**
- The class Bot understand new to create new robots

- aBot does not understand **new**
- aBot understands robot messages such as turn:, go:, jump:, turnLeft:, color, lookLikeTriangle

Roadmap

- Context: Software ***MUST*** evolve
- OOP Modeling
- Objects
- Classes
- ***Inheritance***

How to Share Specification?

- Do not want to rewrite everything!
- Often times want small changes
- Man are a specialization of People

- OOP language offers inheritance to reuse or specialize classes

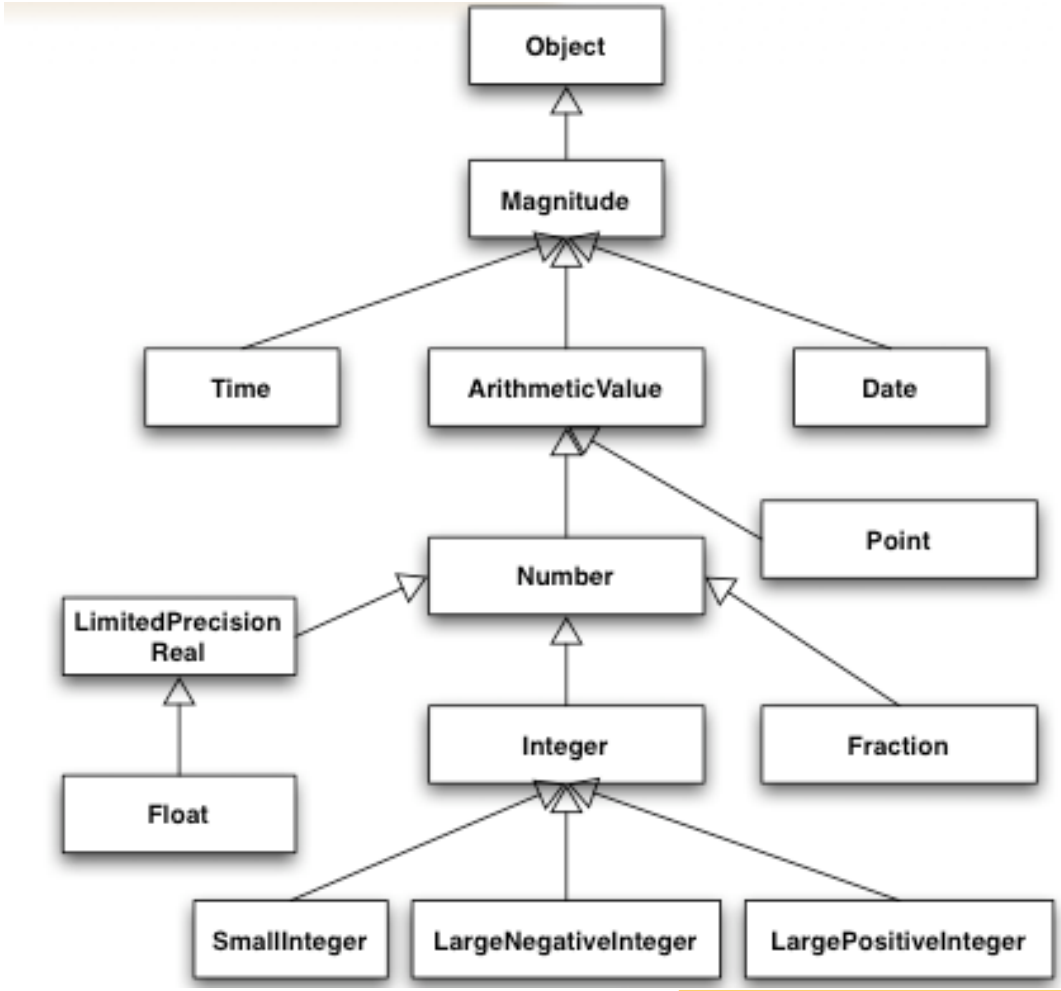
- Class hierarchies for sharing of definitions
- Each class defines or refines the definition of its ancestors

Inheritance

- New classes
  - Can add state and behavior (ColoredRectangle)
  - Can specialize ancestor behavior (GoldenRectangle)
  - Can use ancestor's behavior and state
  - Can hide ancestor's behavior

- Direct ancestor = superclass
- Direct descendant = subclass

Comparable Quantity Hierarchy



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

Summary

An object has a state, a behavior and a unique identity.
The structure and behavior of similar objects is shared in their ***class***
Instances of a class and objects are terms that cn be exchanged
Classes as organized in generalisation/specialisation hierarchy
What is an instance?
What is the difference between instantiation and inheritance?