Adding Object-Oriented Capabilities to Mathematica

Hilarie Nickerson

Fall 2011

Roadmap

About Mathematica

- Environment
- Language features
- Programming paradigms

What might object-oriented Mathematica be like?

- Onging efforts to add capabilities
- The *Objectica* add-on

Discussion of object orientation in Mathematica

- Emergence
- Best uses

Resources

Mathematica

What is it?

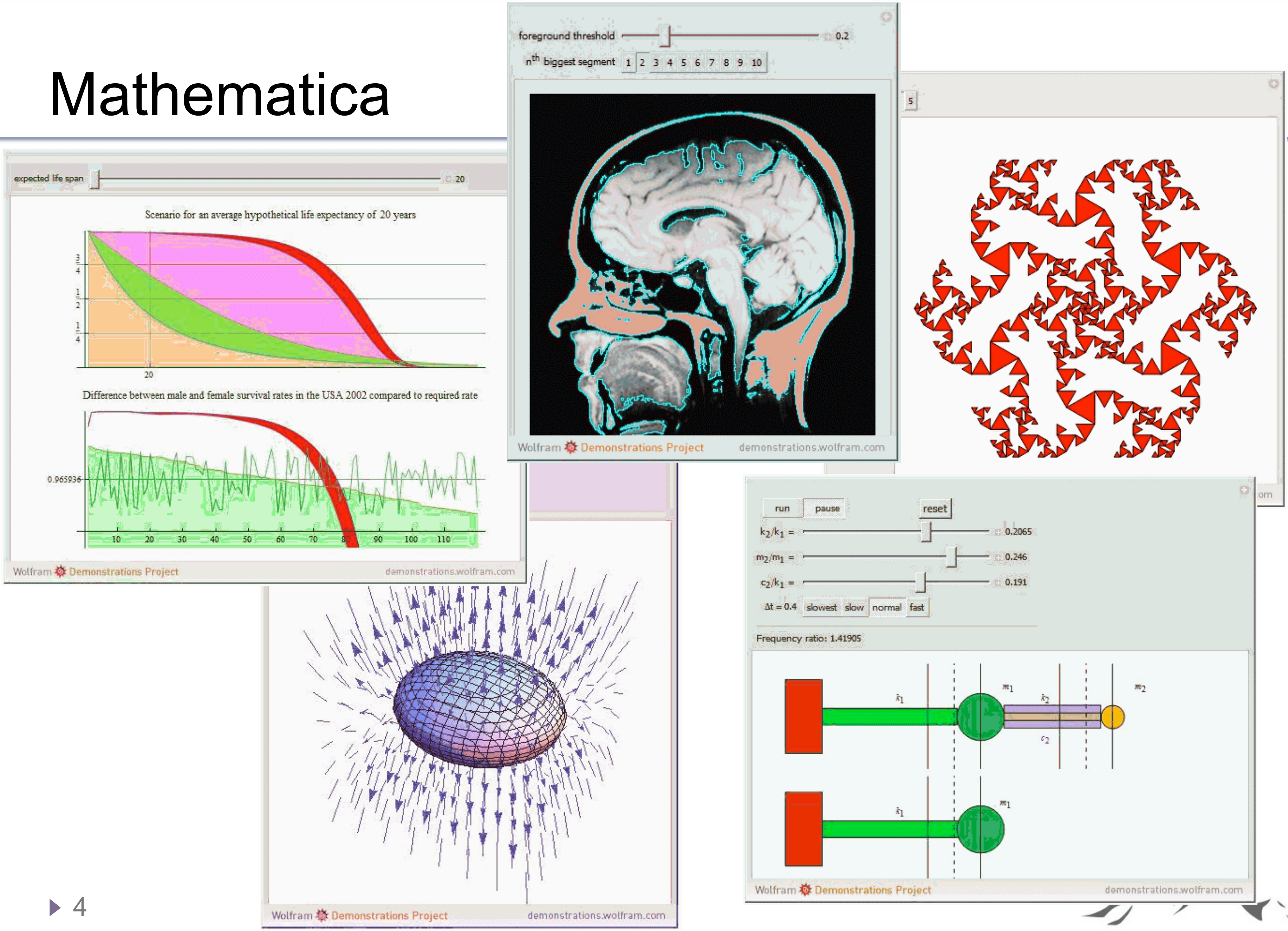
- Software for making computations and visualizing results
  - Interactive exploration is a key feature
- Originally developed and released by Stephen Wolfram in 1988; now sold by Wolfram Research

Who uses it?

- Millions of users
  - STEM / Medicine
  - Business
  - Social sciences
  - Education
  - Arts

"Mathematica has become a standard in a great many organizations, and it is used today in all of the Fortune 50 companies, all of the 15 major departments of the U.S. government, and all of the world's 50 largest universities."

Mathematica



Mathematica: Environment

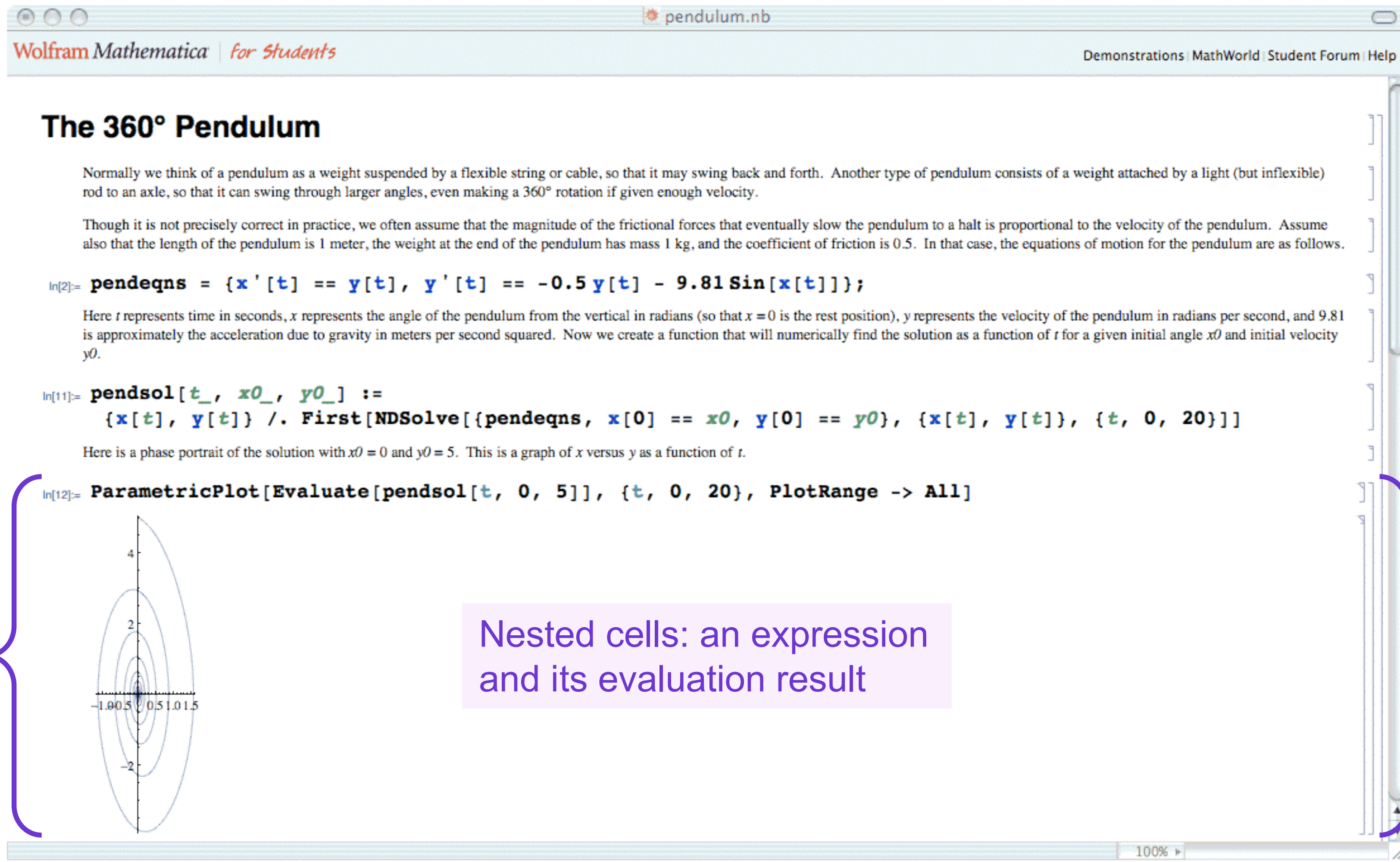
Interactive user interfaces

- Notebooks
  - Evaluate expression in any *cell*, see results immediately
  - May include explanatory text
  - Mathematica help files are also notebooks
- Workbench (Eclipse IDE)
- Web-based player for local and remote content
  - Replaced desktop-based player

Computation engine

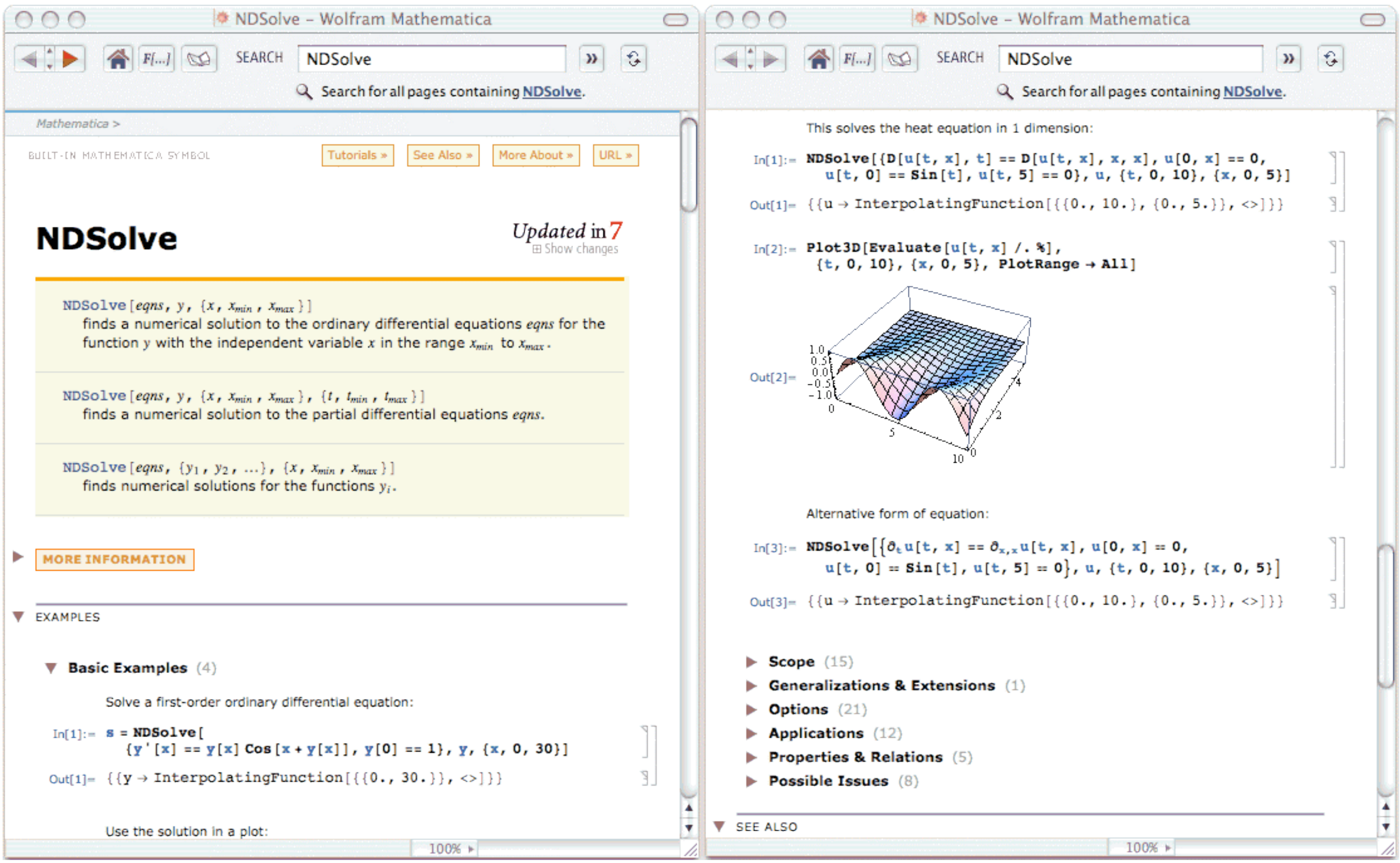
- Kernel accessible from above interfaces and as a service to other programs

Mathematica: Environment



Nested cells: an expression and its evaluation result

Mathematica: Environment



Mathematica: Environment

Interactive user interfaces

- Notebooks
  - Evaluate expressions in *cells*, see results immediately
  - May include explanatory text
  - Mathematica help files are also notebooks
- Workbench (Eclipse IDE)
- Web-based player for local and remote content
  - Replaced desktop-based player

Computation engine

- Kernel accessible from above interfaces and as a service to other programs

Mathematica: Language Features

Numerous built-in functions and libraries

- Chooses best algorithm

Scoping

- Modules (lexical scoping)
- Blocks (dynamic scoping, less commonly used)

Exception handling

String / list manipulation

Rules and pattern matching

Symbolic computation

- Here is a typical numerical computation.

```
In[1]:=  3 + 62 - 1

Out[1]=  64
```

This is a symbolic computation.

```
In[2]:=  3 x - x + 2

Out[2]=  2 + 2 x
```

Mathematica: Programming Paradigms

Major paradigms

- Procedural
- Functional
- **Object-oriented** *(or so they say…)*

Additional paradigms

- List-based
- Rule-based
- String-based

Object-Oriented

Mathematica

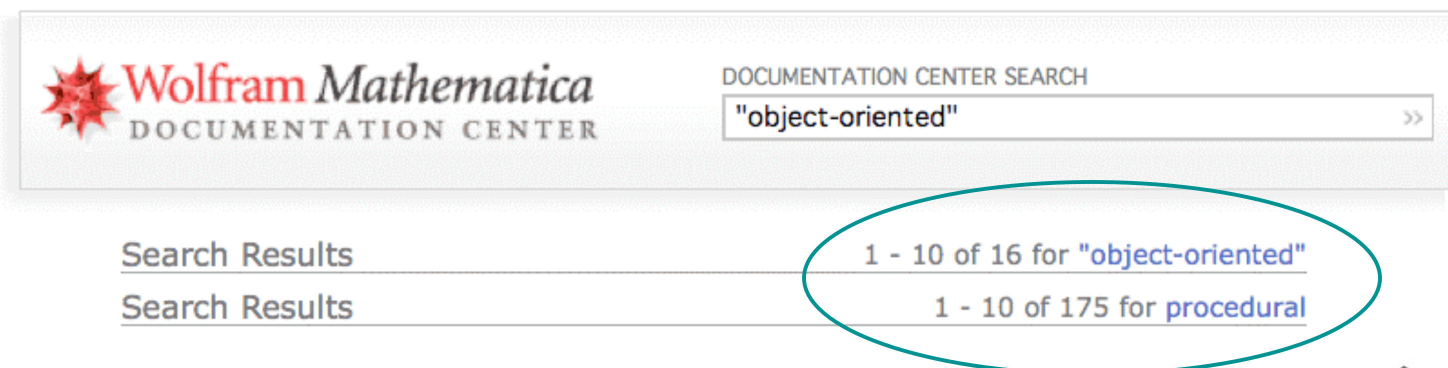
Wolfram's early claims of object-oriented capabilities have faded over time…

- Then

  "It is very easy to do object-oriented programming in Mathematica. The basic idea is to associate Mathematica transformation rules with the objects they act on rather than with the functions they perform."

  — *The Mathematica Book, First Edition*

- Now



Object-Oriented

Mathematica: Built-In Capabilities

Still not giving up…

Object-Oriented Definition

UpSet (^=) — associate a definition with an inner construct

TagSet (/: ... =) — associate a definition with any construct

Functional Programming

Long viewed as an important theoretical idea, functional programming finally became truly convenient and practical with the introduction of *Mathematica*'s symbolic language. Treating expressions like f[x] as both symbolic data and the application of a function f provides a uniquely powerful way to integrate structure and function—and an efficient, elegant representation of many common computations.

Function (&) — specify a pure function (e.g. (# + 1) &)

#, **##** — slots for variables in a pure function

Applying Functions to Lists »

Map (/@) — map across a list: f /@ {x, y, z}⟶{f[x], f[y], f[z]}

Apply (@@, @@@) — apply to a list: f @@ {x, y, z}⟶f[x, y, z]

MapIndexed — map with index information: {f[x, {1}], f[y, {2}], f[z, {3}]}

MapThread ▪ **MapAt** ▪ **MapAll** ▪ **Scan** ▪ ...

Iteratively Applying Functions »

Nest, **NestList** — nest a function: f[f[f[x]]] etc.

Fold, **FoldList** — fold in a list of values: f[f[f[x, 1], 2], 3] etc.

FixedPoint, **FixedPointList** — repeatedly nest until a fixed point

NestWhile ▪ **NestWhileList** ▪ **TakeWhile** ▪ **LengthWhile** ▪ ...

List-Oriented Functions

Select — select from a list according to a function

Array — create an array from a function

Sort, **Split** — sort, split according to a function

Functional Composition Operations

Identity ▪ **Composition** ▪ **Operate** ▪ **Through** ▪ **Distribute**

Procedural Programming

Mathematica stands out from traditional computer languages in supporting many programming paradigms. Procedural programming is the only paradigm available in languages like C and Java, as well as most scripting languages. *Mathematica* supports all standard procedural programming constructs, but often extends them through integration into its more general symbolic programming environment.

x=value (**Set**) — set the value for a variable

expr;expr;expr (**CompoundExpression**) — execute expressions in sequence

Assignments »

= ▪ **+=** ▪ **++** ▪ ***=** ▪ **AppendTo** ▪ ...

Loops »

Do ▪ **While** ▪ **For** ▪ **Table** ▪ **Nest** ▪ ...

Conditionals »

If ▪ **Which** ▪ **Switch** ▪ **And(&&)** ▪ **Equal(==)** ▪ **Less(<)** ▪ ...

Flow Control »

Return ▪ **Throw** ▪ **Catch** ▪ **TimeConstrained** ▪ ...

Scoping Constructs »

Module ▪ **With** ▪ **Block** ▪ ...

Input, Output, Etc. »

Print ▪ **Input** ▪ **Pause** ▪ **Import** ▪ **OpenRead** ▪ ...

Object-Oriented
Mathematica: Built-In Capabilities

Stack example using TagSet (/: … =)

- stackobj /: push[stackobj[stack_, item_]] := Append[stack, item];
  stackobj /: pop[stackobj[stack_]] := Most[stack];
- mystack = {1, 2, 3}
  myitem = 4
- mystack = push[stackobj[mystack, myitem]]
  - {1, 2, 3, 4}
- mystack = pop[stackobj[mystack]]
  - {1, 2, 3}
- mystack = pop[stackobj[mystack]]
  - {1, 2}

In web forums, highly experienced Mathematica programmers suggest that inheritance, etc. is possible; no examples found

Object-Oriented

Mathematica: Ongoing Enhancement Efforts

1988	Mathematica released
1993	Roman Maeder's *Classes.m* package
2002	Hermann Schmitt's *OO System for Mathematica*
2005	Orestis Vantzos' *OOP* package
2008	Stephan Leibbrandt's *Objectica* package
2010	Ross Tang's *MathOO* package

Object-Oriented

Mathematica: Ongoing Enhancement Efforts

Maeder (1993), Leibbrandt (2008) packages most significant

- Sanctioned in some way by Wolfram
- Available as add-ons

Other packages offered by Mathematica enthusiasts

- Some Q&A in user community
- Varying levels of capability, documentation

Syntactic differences (as would be expected)

- Maeder `translateBy[sphere1, {1, 0, 0}]`
- Vantzos `sphere1::translateBy[{1, 0, 0}]`
- Leibbrandt `sphere1.translateBy[{1, 0, 0}]`

Mathematica: Maeder's Classes.m

First serious effort at an add-on

- Originally promising, but…
  - Weakly documented
  - Support later withdrawn

> "Mathematica will surely become the prototyping tool par excellence for object oriented programming."
>
> — *Mastering Mathematica*

Sample code

```
Class[ Account, Object,
        {bal, own},
        {
         {new,  (new[super]; bal = #1; own = #2)&},
         {balance, bal&},
         {deposit, Function[bal += #1]},
         {withdraw, Function[bal -= #1]},
         {owner, own&}
        }
]
```

Object-Oriented

Mathematica: The Objectica Add-On

What is it?

- Package for adding object-oriented features to Mathematica
  - Sales literature emphasizes "abstract data types, inheritance, encapsulation, and polymorphism"
- Developed by Stephan Leibbrandt in 2008; sold by Symbols and Numbers (Germany)

Who uses it?

- Size of user population unclear

"Typical Users
- Software engineers of other object oriented languages for building prototypes
- Developers of big Mathematica projects in order to structure the problem
- Engineers to image real objects"

Object-Oriented

Mathematica: The Objectica Add-On

Abstract data types

- Confusing terminology here; should really say abstract base classes, which are appropriately implemented

Inheritance and polymorphism

- Clear syntax, handled well

Encapsulation

- Some difficulties here with respect to class / subclass relationships (see *Virtual* later on)
- Can hide data and methods with *Private* option

Object-Oriented

Mathematica: The Objectica Add-On

More features

- Interfaces
  - Very much like abstract classes
  - Classes can use multiple interfaces, but can have only one parent class
- Anonymous classes
  - Available, but poorly documented

Overall assessment

- Implements object orientation well, aside from encapsulation (open to programmer error)
- Well-documented, for the most part

Object-Oriented

Mathematica: The Objectica Add-On

Comparison with other languages				
	Ruby	*Java*	*Python*	*Objectica*
Typing	Dynamic	Static	Dynamic	Dynamic
Inheritance	Single with mixins	Single with interfaces	Multiple	Single with interfaces
Method overloading	No	Yes	No	Yes
Class vars & methods	Yes	Yes	No	Yes

Mathematica: Objectica Usage Basics

Package loading options

- Needs["Class`Class`"]
- Get["Class`Class`"]
- Get["<path to Class.m>"]
- Note use of Class continues (name originated by Roman Maeder)

Object-Oriented

Mathematica: Objectica Usage Basics

Class definition

- ```
  Class[base] := {
    Virtual.st = 0,
    f[x_?Positive] := x^2,
    f[x_?Negative] := x^3 + st
    }
  ```
- Default constructor created
- Use *Virtual* to ensure that child's instance of st is accessed when f[x_?Negative] is called
  - Good practice in case children come later

Subclass definition

- ```
  Class[child, base] := {
    st = 20,
    f[x_?Positive] := x,
    g[y_] := Sin[y]
    }
  ```
- More explicit version using *Override*
  ```
  Class[child, base] := {
    Override.Virtual.st = 20,
    Override.f[x_?Positive] := x,
    g[y_] := Sin[y]
    }
  ```

Object-Oriented
Mathematica: Objectica Usage Basics

Creating a new object

- baseObj = New.base[]
  - Note use of dot notation
- Alternatively, New[baseObj].base[]

Redefining class to include constructor

- Class[base] := {
  Virtual.st = 0,
  base[st_] := (This.st = st),
  f[x_?Positive] := x^2,
  f[x_?Negative] := x^3 + st
  }
- baseObj1 = New.base[10]

Object-Oriented

Mathematica: Objectica Usage Basics

Setting an instance variable

- baseObj.st = 100;

Calling a method

- baseObj.f[-5]

Object-Oriented

Mathematica: Objectica Usage Basics

Abstract class with polymorphism

- Class definition (note use of *Abstract*)
  - Class[Room] := { … }
  - SetAttributes[Class[Room], Abstract]
- Alternatively, define in one step
  - Abstract.Class[Room] := { … }
- Subclass definitions (note use of *Super*)
  - Class[Single, Room] := { Single[person_String] := Super[person] }
    Class[Double, Room] := { … }
- Calling Price method polymorphically
  - New.Single["Mr. Smith"].Price[]
    New.Double["Mr. Smith", "Mrs. Smith"].Price[]

Object-Oriented

Mathematica: Objectica Usage Basics

Class with interfaces

```
Interface[one] := {f[x_] := 0};
Interface[two] := {g[x_] := 0};
Class[base2] := {h[x_] := x^2};

Class[child, base2, one, two] := {
  Override.f[x_] := x^3,
  Override.g[x_] := x^4
  };
```

An anonymous class

```
New.base[].{ … }
```

Object-Oriented

Mathematica: Objectica Usage Basics

Public, protected, and private data

- Class[base3] := {
  e[x_] := x^2,
  Public.f[x_] := x^3, ⬅ *any caller*
  Protected.g[x_] := x^4, ⬅ *child classes*
  Private.h[x_] := x^5 ⬅ *this class*
  };
- Alternatively, SetAttributes[Room.Persons, Private]

Class method definition and call

- Class[base4] := {
  Static.z[x_] := x^2
  }
- base4.z[5]

Mathematica: Objectica Usage Example

Stack example, revisited

- ```
  Class[stackobj2] := {
    Virtual.stack = {},
    stackobj2[stack_] := (This.stack = stack),
    push[item_] := stack = Append[stack, item],
    pop[] := stack = Most[stack]
    }
  ```
- `mystack2 = New.stackobj2[{1, 2, 3}]`
- `mystack2.push[myitem]`
  - {1, 2, 3, 4}
- `mystack2.pop[]`
  - {1, 2, 3}
  ```

Object-Oriented

Mathematica: Discussion

Why so slow to emerge?

- Existing programming paradigms are powerful
- Mathematica programmers know workarounds
  - Rules, patterns
  - Interfaces to OO languages such as Java, C++
- Big shift in thinking required (and not desired)
  - E.g., pushback on new OO graph features in Mathematica 8

Best uses

- Large applications with significant complexity
- Real-world applications with hierarchical structure
- User interface programming
- Situations where encapsulation would be helpful

Resources for Further Exploration

Mathematica's built-in object-oriented abilities

- *The Mathematica Book, First Edition*
  section on "object-oriented" programming
  http://reference.wolfram.com/legacy/v1/contents/4.1.6.pdf
- Online documentation of "object-oriented" functionality
  - UpSet
    http://reference.wolfram.com/mathematica/ref/UpSet.html
  - TagSet
    http://reference.wolfram.com/mathematica/ref/TagSet.html

Mathematica linking to object-oriented languages

- http://reference.wolfram.com/mathematica/guide/SystemsInterfacesAndDeploymentOverview.html
- http://reference.wolfram.com/mathematica/tutorial/MathLinkAndExternalProgramCommunicationOverview.html

Resources for Further Exploration

Objectica product pages

- Wolfram
  http://www.wolfram.com/products/applications/objectica/
- Symbols and Numbers
  http://www.objectica.net/

Objectica presentations

- *From Symbols to Objects*
  2010 Wolfram Technology Conference
  http://library.wolfram.com/infocenter/Conferences/7871/
- *Object-Oriented Modeling with Objectica*
  2007 Wolfram Technology Conference
  http://library.wolfram.com/infocenter/Conferences/6923/

Resources for Further Exploration

Other efforts

- Roman Maeder's Classes.m package
  - *The Mathematica Journal* 3:1, pp. 23-31 (1993)
    http://library.wolfram.com/infocenter/Articles/3243/
  - Gray, J., *Mastering Mathematica*, chapter 9
  - Maeder, R. *The Mathematica Programmer*, chapter 4
    http://www.mathconsult.ch/showroom/pubs/MathProg/htmls/1-04.htm
- Hermann Schmitt's OO System for Mathematica
  - An OO System for Mathematica, Version 3
    http://www.schmitther.de/oosys_en/introduction.html

Resources for Further Exploration

Other efforts

- Orestis Vantzos' OOP package
  - *From Symbols to Objects*
    2005 Wolfram Technology Conference
    http://library.wolfram.com/infocenter/Conferences/5773/
- Ross Tang's MathOO package
  - Code repository
    http://code.google.com/p/mathoo-packages/
  - Additional documentation
    http://www.voofie.com/concept/MathOO/
    - Read in date order, not display order

Credits

Slide 4 pictures

- http://demonstrations.wolfram.com/NegligibleSenescenceScenario/
- http://demonstrations.wolfram.com/SegmentingAMedicalImage/
- http://demonstrations.wolfram.com/RecursiveExercisesIIIFirePatterns/
- http://demonstrations.wolfram.com/TunedMassDamper/
- http://demonstrations.wolfram.com/SurfacesAndGradients/

Slide 6 notebook

- http://www.math.umd.edu/undergraduate/schol/primer/Notebooks/pendulum.nb

Slide 7 pictures

- Mathematica documentation

Credits

Slide 9 picture

- http://reference.wolfram.com/mathematica/tutorial/SymbolicComputation.html

Slide 12 pictures

- Mathematica documentation

Slide 16 code

- http://library.wolfram.com/infocenter/Articles/3243/

Slide 20 table

- http://www.schmitther.de/oosys_en/comp_tab.html

Slides 20–27 code

- Objectica documentation