CS 220: Survey of Programming Languages

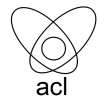
Lecture Slides

Object Oriented Programming Languages

Jan Michael C. Yap

Algorithms and Complexity Laboratory
Department of Computer Science
University of the Philippines, Diliman
jcyap@dcs.upd.edu.ph

Session 5



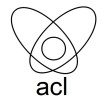
Object Oriented Programming Languages

Object Oriented Programming Languages

Smalltalk

C++

Java



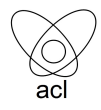
Object Oriented Programming Languages

Object Oriented Programming Languages

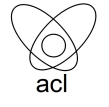
Smalltalk

C++

Java

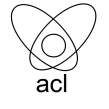


Characteristics of an object-oriented PL



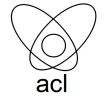
Characteristics of an object-oriented PL

- Classes as objects



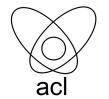
Characteristics of an object-oriented PL

- Classes as objects
  - Variables as properties
  - Procedures/functions as methods



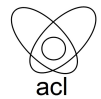
Characteristics of an object-oriented PL

- Classes as objects
  - Variables as properties
  - Procedures/functions as methods
- Access control via abstraction and encapsulation



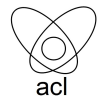
Characteristics of an object-oriented PL

- Classes as objects
  - Variables as properties
  - Procedures/functions as methods
- Access control via abstraction and encapsulation
- Object polymorphism



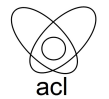
Characteristics of an object-oriented PL

- Classes as objects
  - Variables as properties
  - Procedures/functions as methods
- Access control via abstraction and encapsulation
- Object polymorphism
  - Subclass, superclass, and interfaces
  - Inheritance

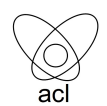


Characteristics of an object-oriented PL

- Classes as objects
  - Variables as properties
  - Procedures/functions as methods
- Access control via abstraction and encapsulation
- Object polymorphism
  - Subclass, superclass, and interfaces
  - Inheritance
- Deeply rooted from imperative PLs



The "birth" of OO according to Alan Kay



http://gagne.homedns.org/∼tgagne/contrib/EarlyHistoryST.html

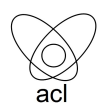
The "birth" of OO according to Alan Kay

- "Central" motivations for object oriented programming

http://gagne.homedns.org/~tgagne/contrib/EarlyHistoryST.html

The "birth" of OO according to Alan Kay

- "Central" motivations for object oriented programming
  - "[F]ind a better module scheme for complex systems involving hiding of details..."



http://gagne.homedns.org/~tgagne/contrib/EarlyHistoryST.html

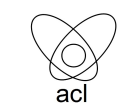
The "birth" of OO according to Alan Kay

- "Central" motivations for object oriented programming
  - "[F]ind a better module scheme for complex systems involving hiding of details..."
  - "... [F]ind a more flexible version of assignment, and then to try to eliminate it altogether."

http://gagne.homedns.org/~tgagne/contrib/EarlyHistoryST.html

The "birth" of OO according to Alan Kay

- "Central" motivations for object oriented programming
  - "[F]ind a better module scheme for complex systems involving hiding of details..."
  - "... [F]ind a more flexible version of assignment, and then to try to eliminate it altogether."
- "The basic principal(sic?) of recursive design is to make the parts have the same power as the whole." - Bob Barton



http://gagne.homedns.org/~tgagne/contrib/EarlyHistoryST.html

The "birth" of OO according to Alan Kay

- "Central" motivations for object oriented programming
  - "[F]ind a better module scheme for complex systems involving hiding of details..."
  - "... [F]ind a more flexible version of assignment, and then to try to eliminate it altogether."
- "The basic principal(sic?) of recursive design is to make the parts have the same power as the whole." - Bob Barton
- "I recalled the monads of Leibniz, the "dividing nature at its joints" discourse of Plato, and other attempts to parse complexity."

http://gagne.homedns.org/~tgagne/contrib/EarlyHistoryST.html

By the way...

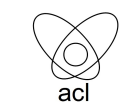


Photo 1: http://www.aes.org/technical/images/Alan_Kay_Photo.jpg
Photo 2: http://famousdude.com/images/allan-k.-06.jpg

By the way...

This is Alan Kay:

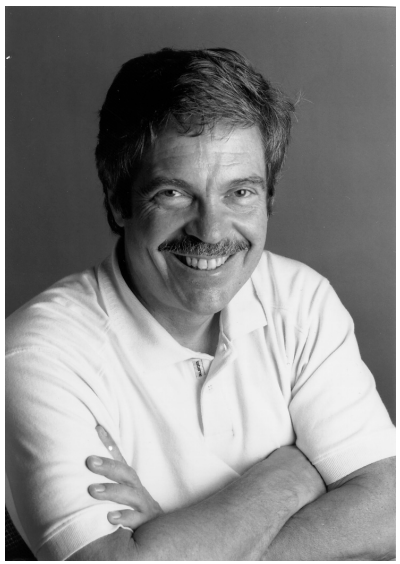
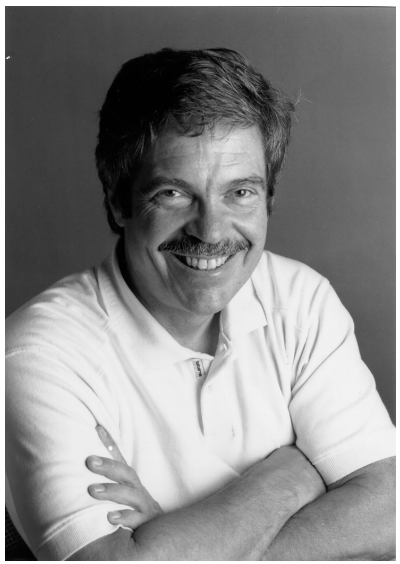


Photo 1: http://www.aes.org/technical/images/Alan_Kay_Photo.jpg
Photo 2: http://famousdude.com/images/allan-k.-06.jpg

By the way...

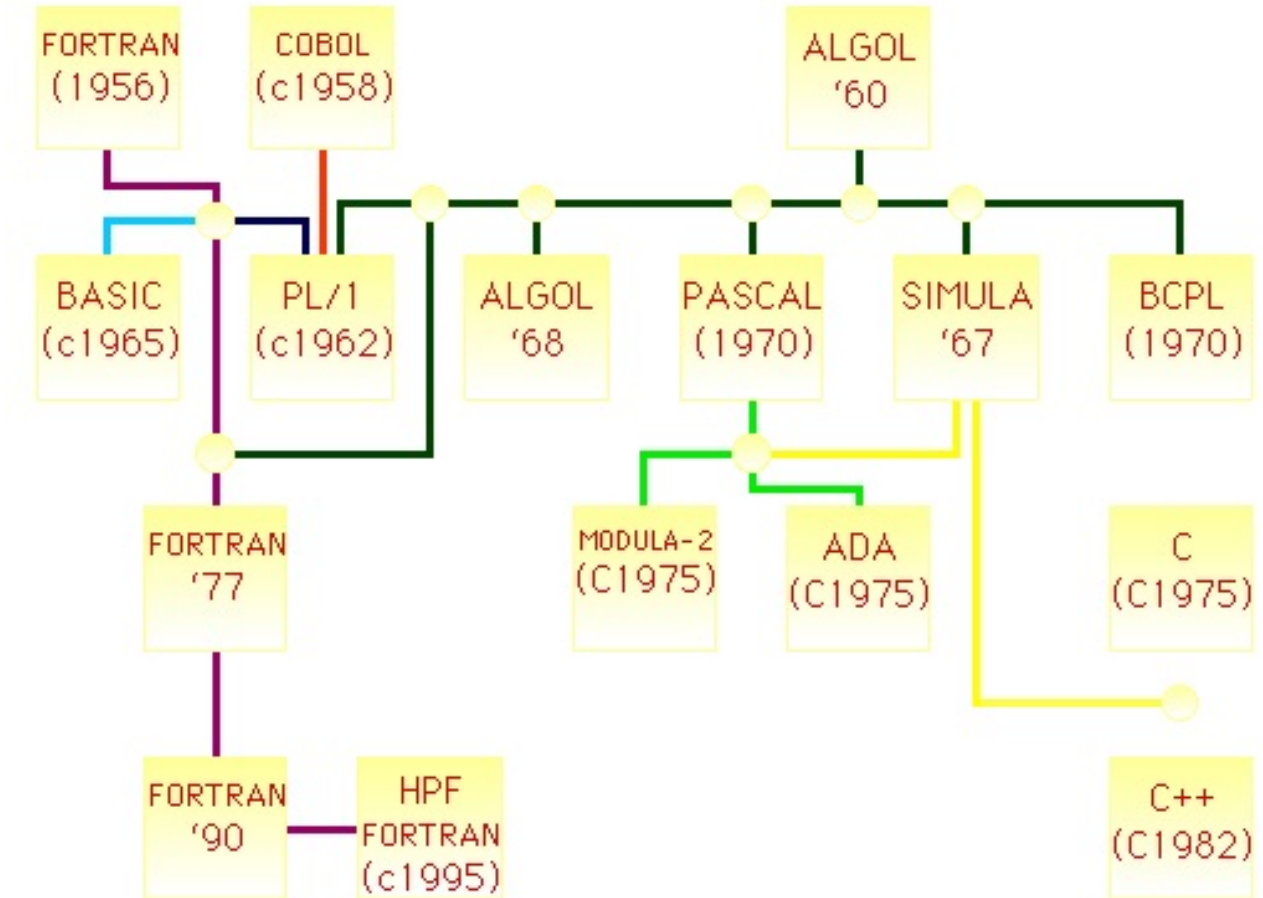
This is Alan Kay:



Not this one:

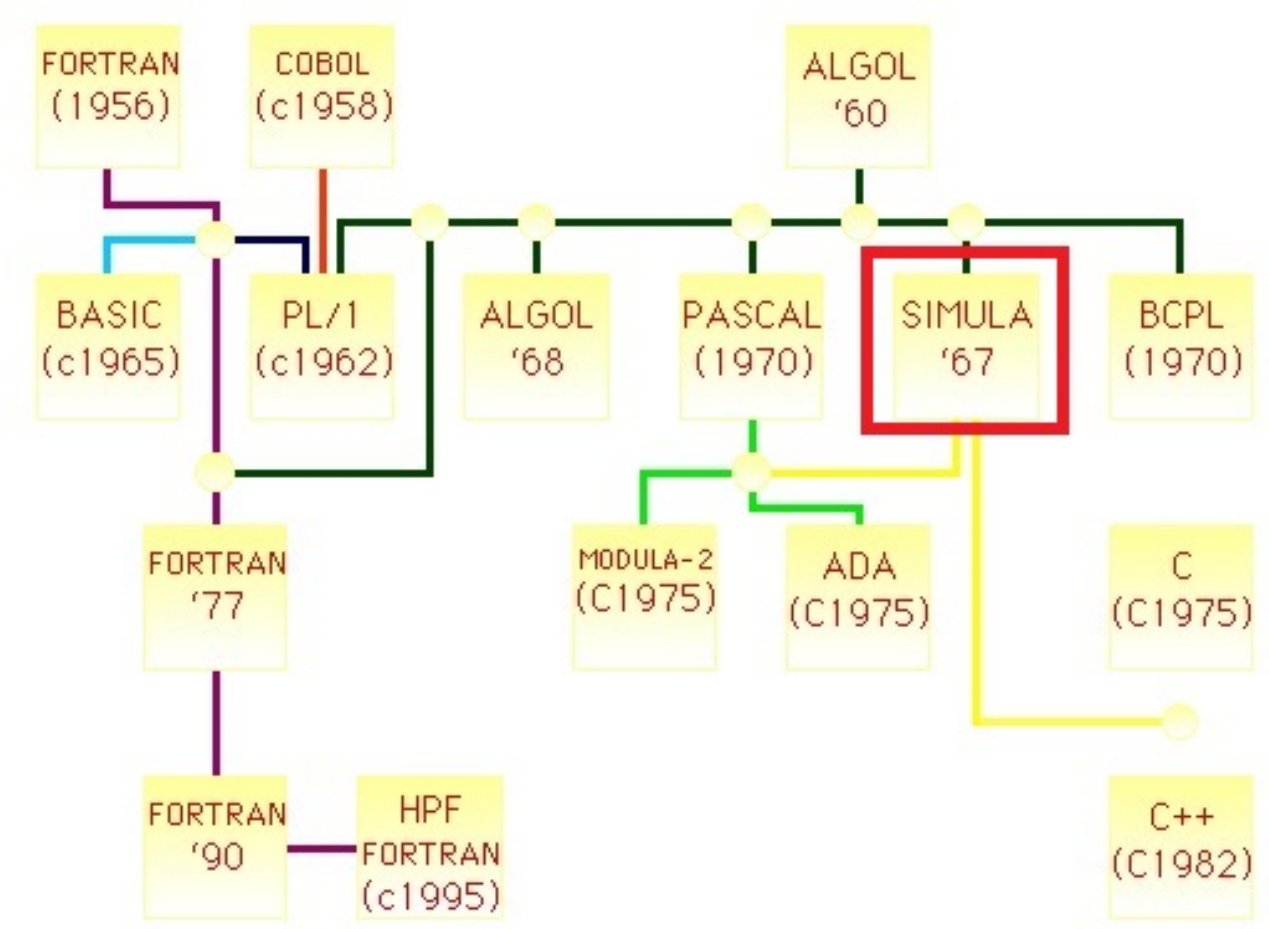
Photo 1: http://www.aes.org/technical/images/Alan_Kay_Photo.jpg
Photo 2: http://famousdude.com/images/allan-k.-06.jpg

Imperative PL"lineage" redux



http://cgi.csc.liv.ac.uk/~grant/Teaching/COMP205/intro.html

Simula as the progenitor of OOPLs



http://cgi.csc.liv.ac.uk/~grant/Teaching/COMP205/intro.html

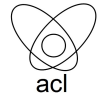
Object Oriented Programming Languages

Object Oriented Programming Languages

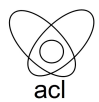
Smalltalk

C++

Java



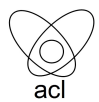
Background



http://gagne.homedns.org/∼tgagne/contrib/EarlyHistoryST.html

Background

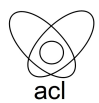
- Name was christened by Alan Kay



http://gagne.homedns.org/∼tgagne/contrib/EarlyHistoryST.html

Background

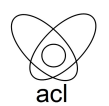
- Name was christened by Alan Kay
  - "[A] language I now called "Smalltalk" - as in "programming should be a matter of ..." and "children should program in... "



http://gagne.homedns.org/~tgagne/contrib/EarlyHistoryST.html

Background

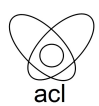
- Name was christened by Alan Kay
  - "[A] language I now called "Smalltalk" - as in "programming should be a matter of ..." and "children should program in... "
- The original Smalltalk was released in 1971, but the first "real" version was not until 1972.



http://gagne.homedns.org/∼tgagne/contrib/EarlyHistoryST.html

Background

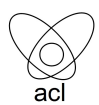
- Name was christened by Alan Kay
  - "[A] language I now called "Smalltalk" - as in "programming should be a matter of ..." and "children should program in... "
- The original Smalltalk was released in 1971, but the first "real" version was not until 1972.
- Main ideas behind Smalltalk's scheme



http://gagne.homedns.org/~tgagne/contrib/EarlyHistoryST.html

Background

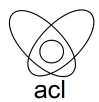
- Name was christened by Alan Kay
  - "[A] language I now called "Smalltalk" - as in "programming should be a matter of ..." and "children should program in... "
- The original Smalltalk was released in 1971, but the first "real" version was not until 1972.
- Main ideas behind Smalltalk's scheme
  - "Everything is an object"



http://gagne.homedns.org/~tgagne/contrib/EarlyHistoryST.html

Background

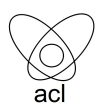
- Name was christened by Alan Kay
  - "[A] language I now called "Smalltalk" - as in "programming should be a matter of ..." and "children should program in... "
- The original Smalltalk was released in 1971, but the first "real" version was not until 1972.
- Main ideas behind Smalltalk's scheme
  - "Everything is an object"
  - "Objects communicate by sending and receiving messages"



http://gagne.homedns.org/~tgagne/contrib/EarlyHistoryST.html

Background

- Name was christened by Alan Kay
  - "[A] language I now called "Smalltalk" - as in "programming should be a matter of ..." and "children should program in... "
- The original Smalltalk was released in 1971, but the first "real" version was not until 1972.
- Main ideas behind Smalltalk's scheme
  - "Everything is an object"
  - "Objects communicate by sending and receiving messages"
  - "Objects have their own memory"



http://gagne.homedns.org/~tgagne/contrib/EarlyHistoryST.html

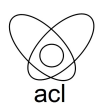
Background

- Name was christened by Alan Kay
  - "[A] language I now called "Smalltalk" - as in "programming should be a matter of ..." and "children should program in... "
- The original Smalltalk was released in 1971, but the first "real" version was not until 1972.
- Main ideas behind Smalltalk's scheme
  - "Everything is an object"
  - "Objects communicate by sending and receiving messages"
  - "Objects have their own memory"
  - "Every object is an instance of a class"

http://gagne.homedns.org/~tgagne/contrib/EarlyHistoryST.html

Background

- Name was christened by Alan Kay
  - "[A] language I now called "Smalltalk" - as in "programming should be a matter of ..." and "children should program in... "
- The original Smalltalk was released in 1971, but the first "real" version was not until 1972.
- Main ideas behind Smalltalk's scheme
  - "Everything is an object"
  - "Objects communicate by sending and receiving messages"
  - "Objects have their own memory"
  - "Every object is an instance of a class"
  - "The class holds the shared behavior for its instances"



http://gagne.homedns.org/~tgagne/contrib/EarlyHistoryST.html

Background

- Name was christened by Alan Kay
  - "[A] language I now called "Smalltalk" - as in "programming should be a matter of ..." and "children should program in... "
- The original Smalltalk was released in 1971, but the first "real" version was not until 1972.
- Main ideas behind Smalltalk's scheme
  - "Everything is an object"
  - "Objects communicate by sending and receiving messages"
  - "Objects have their own memory"
  - "Every object is an instance of a class"
  - "The class holds the shared behavior for its instances"
  - "To eval[uate] a program list, control is passed to the first object and the remainder is treated as its message"

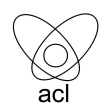
http://gagne.homedns.org/~tgagne/contrib/EarlyHistoryST.html

Evaluation

- Data types[1,2]

[1] http://people.cs.clemson.edu/~turner/courses/cs428/current/webct/content/st/stbasics.html

[2] http://web.cecs.pdx.edu/~harry/musings/SmalltalkOverview.html

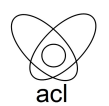


Evaluation

- Data types[1,2]
  - Integers, real (floating point) numbers, strings, logical as primitive types

[1] http://people.cs.clemson.edu/~turner/courses/cs428/current/webct/content/st/stbasics.html

[2] http://web.cecs.pdx.edu/~harry/musings/SmalltalkOverview.html



Evaluation

- Data types[1,2]
  - Integers, real (floating point) numbers, strings, logical as primitive types
    - Primitive types are also classes, and values are objects

[1] http://people.cs.clemson.edu/~turner/courses/cs428/current/webct/content/st/stbasics.html

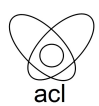
[2] http://web.cecs.pdx.edu/~harry/musings/SmalltalkOverview.html

Evaluation

- Data types[1,2]
  - Integers, real (floating point) numbers, strings, logical as primitive types
    - Primitive types are also classes, and values are objects
  - Support for multidimentional arrays, which can be unbounded

[1] http://people.cs.clemson.edu/~turner/courses/cs428/current/webct/content/st/stbasics.html

[2] http://web.cecs.pdx.edu/~harry/musings/SmalltalkOverview.html

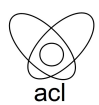


Evaluation

- Data types[1,2]
  - Integers, real (floating point) numbers, strings, logical as primitive types
    - Primitive types are also classes, and values are objects
  - Support for multidimentional arrays, which can be unbounded
  - Pointers are implicit objects

[1] http://people.cs.clemson.edu/~turner/courses/cs428/current/webct/content/st/stbasics.html

[2] http://web.cecs.pdx.edu/~harry/musings/SmalltalkOverview.html



Evaluation

- Syntax design (Redline[3,4])

[3] https://github.com/antlr/grammars-v4/blob/master/smalltalk/Smalltalk.g4
[4] Manually counted
[5] http://www.ianbicking.org/orthogonality-is-pretentious.html

Evaluation

- Syntax design (Redline[3,4])
  - Production rules: 131
  - Number of top alternatives: 51
  - Number of symbols: (unaccounted)
  - Vocabulary
    - Nonterminal symbols: 80
    - Terminal symbols: 109

[3] https://github.com/antlr/grammars-v4/blob/master/smalltalk/Smalltalk.g4
[4] Manually counted
[5] http://www.ianbicking.org/orthogonality-is-pretentious.html

Evaluation

- Syntax design (Redline[3,4])
  - Production rules: 131
  - Number of top alternatives: 51
  - Number of symbols: (unaccounted)
  - Vocabulary
    - Nonterminal symbols: 80
    - Terminal symbols: 109
  - Primitive types are also classes, and values are objects

[3] https://github.com/antlr/grammars-v4/blob/master/smalltalk/Smalltalk.g4
[4] Manually counted
[5] http://www.ianbicking.org/orthogonality-is-pretentious.html

Evaluation

- Syntax design (Redline[3,4])
  - Production rules: 131
  - Number of top alternatives: 51
  - Number of symbols: (unaccounted)
  - Vocabulary
    - Nonterminal symbols: 80
    - Terminal symbols: 109
  - Primitive types are also classes, and values are objects
  - Everything is a class, even primitive data types

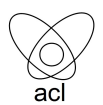
[3] https://github.com/antlr/grammars-v4/blob/master/smalltalk/Smalltalk.g4
[4] Manually counted
[5] http://www.ianbicking.org/orthogonality-is-pretentious.html

Evaluation

- Syntax design (Redline[3,4])
  - Production rules: 131
  - Number of top alternatives: 51
  - Number of symbols: (unaccounted)
  - Vocabulary
    - Nonterminal symbols: 80
    - Terminal symbols: 109
  - Primitive types are also classes, and values are objects
  - Everything is a class, even primitive data types
  - Too orthogonal for an OOPL, that programming tends to create Ravioli Codes[5]

[3] https://github.com/antlr/grammars-v4/blob/master/smalltalk/Smalltalk.g4
[4] Manually counted
[5] http://www.ianbicking.org/orthogonality-is-pretentious.html





Evaluation

- Abstraction

[6] http://pharo.gforge.inria.fr/PBE1/PBE1ch6.html
[7] http://java.ociweb.com/mark/programming/Smalltalk.html
[8] http://www.mimuw.edu.pl/~sl/teaching/00_01/Delfin_EC/Overviews/ExceptionHandling.htm

Evaluation

- Abstraction
  - Methods are public and instance variables are private (to the instances that own it) [6]

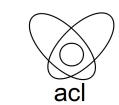
[6] http://pharo.gforge.inria.fr/PBE1/PBE1ch6.html
[7] http://java.ociweb.com/mark/programming/Smalltalk.html
[8] http://www.mimuw.edu.pl/~sl/teaching/00_01/Delfin_EC/Overviews/ExceptionHandling.htm

Evaluation

- Abstraction
  - Methods are public and instance variables are private (to the instances that own it) [6]
  - Classes are not in a namespace, so all class names must be unique.[7]

[6] http://pharo.gforge.inria.fr/PBE1/PBE1ch6.html
[7] http://java.ociweb.com/mark/programming/Smalltalk.html
[8] http://www.mimuw.edu.pl/~sl/teaching/00_01/Delfin_EC/Overviews/ExceptionHandling.htm



Evaluation

- Abstraction
  - Methods are public and instance variables are private (to the instances that own it) [6]
  - Classes are not in a namespace, so all class names must be unique.[7]
- Expressivity: Only apparent drawback is lack of shortcut operators

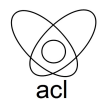
[6] http://pharo.gforge.inria.fr/PBE1/PBE1ch6.html
[7] http://java.ociweb.com/mark/programming/Smalltalk.html
[8] http://www.mimuw.edu.pl/~sl/teaching/00_01/Delfin_EC/Overviews/ExceptionHandling.htm

Evaluation

- Abstraction
  - Methods are public and instance variables are private (to the instances that own it) [6]
  - Classes are not in a namespace, so all class names must be unique.[7]
- Expressivity: Only apparent drawback is lack of shortcut operators
- Exception handling[8]

[6] http://pharo.gforge.inria.fr/PBE1/PBE1ch6.html
[7] http://java.ociweb.com/mark/programming/Smalltalk.html
[8] http://www.mimuw.edu.pl/~sl/teaching/00_01/Delfin_EC/Overviews/ExceptionHandling.htm



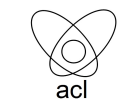


Evaluation

- Abstraction
  - Methods are public and instance variables are private (to the instances that own it) [6]
  - Classes are not in a namespace, so all class names must be unique.[7]
- Expressivity: Only apparent drawback is lack of shortcut operators
- Exception handling[8]
  - Earlier versions did NOT have this mechanism

[6] http://pharo.gforge.inria.fr/PBE1/PBE1ch6.html
[7] http://java.ociweb.com/mark/programming/Smalltalk.html
[8] http://www.mimuw.edu.pl/~sl/teaching/00_01/Delfin_EC/Overviews/ExceptionHandling.htm



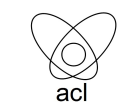


Evaluation

- Abstraction
  - Methods are public and instance variables are private (to the instances that own it) [6]
  - Classes are not in a namespace, so all class names must be unique.[7]
- Expressivity: Only apparent drawback is lack of shortcut operators
- Exception handling[8]
  - Earlier versions did NOT have this mechanism
  - Modern versions
    - Exception classes (e.g. Visual Smalltalk)
    - Signal classes (e.g. VisualWorks)

[6] http://pharo.gforge.inria.fr/PBE1/PBE1ch6.html
[7] http://java.ociweb.com/mark/programming/Smalltalk.html
[8] http://www.mimuw.edu.pl/~sl/teaching/00_01/Delfin_EC/Overviews/ExceptionHandling.htm





Evaluation

- Restricted aliasing

[9] http://web.cecs.pdx.edu/~harry/musings/SmalltalkOverview.html
[10] http://academic.udayton.edu/SaverioPerugini/PL/www/lecture_notes/parampassing.
[11] RA Finkel, Chapter 5.4.5
[12] http://web.cecs.pdx.edu/~harry/musings/SmalltalkOverview.html

Evaluation

- Restricted aliasing
  - Pointers are implicit[9]

[9] http://web.cecs.pdx.edu/~harry/musings/SmalltalkOverview.html
[10] http://academic.udayton.edu/SaverioPerugini/PL/www/lecture_notes/parampassing.
[11] RA Finkel, Chapter 5.4.5
[12] http://web.cecs.pdx.edu/~harry/musings/SmalltalkOverview.html

Evaluation

- Restricted aliasing
  - Pointers are implicit[9]
  - Methods employ pass-by-reference (for non-constants and literals)[10]

[9] http://web.cecs.pdx.edu/~harry/musings/SmalltalkOverview.html
[10] http://academic.udayton.edu/SaverioPerugini/PL/www/lecture_notes/parampassing.
[11] RA Finkel, Chapter 5.4.5
[12] http://web.cecs.pdx.edu/~harry/musings/SmalltalkOverview.html

Evaluation

- Restricted aliasing
  - Pointers are implicit[9]
  - Methods employ pass-by-reference (for non-constants and literals)[10]
- Efficiency [11]

[9] http://web.cecs.pdx.edu/~harry/musings/SmalltalkOverview.html
[10] http://academic.udayton.edu/SaverioPerugini/PL/www/lecture_notes/parampassing.
[11] RA Finkel, Chapter 5.4.5
[12] http://web.cecs.pdx.edu/~harry/musings/SmalltalkOverview.html

Evaluation

- Restricted aliasing
  - Pointers are implicit[9]
  - Methods employ pass-by-reference (for non-constants and literals)[10]
- Efficiency [11]
  - Smalltalk implements dynamic typing [12] in that it type-checks at runtime

[9] http://web.cecs.pdx.edu/~harry/musings/SmalltalkOverview.html
[10] http://academic.udayton.edu/SaverioPerugini/PL/www/lecture_notes/parampassing.
[11] RA Finkel, Chapter 5.4.5
[12] http://web.cecs.pdx.edu/~harry/musings/SmalltalkOverview.html

Evaluation

- Restricted aliasing
  - Pointers are implicit[9]
  - Methods employ pass-by-reference (for non-constants and literals)[10]
- Efficiency [11]
  - Smalltalk implements dynamic typing [12] in that it type-checks at runtime
  - Storage manager is crucial to a lot of mechanisms compiling and running Smalltalk

[9] http://web.cecs.pdx.edu/~harry/musings/SmalltalkOverview.html
[10] http://academic.udayton.edu/SaverioPerugini/PL/www/lecture_notes/parampassing.
[11] RA Finkel, Chapter 5.4.5
[12] http://web.cecs.pdx.edu/~harry/musings/SmalltalkOverview.html



Evaluation

- Restricted aliasing
  - Pointers are implicit[9]
  - Methods employ pass-by-reference (for non-constants and literals)[10]
- Efficiency [11]
  - Smalltalk implements dynamic typing [12] in that it type-checks at runtime
  - Storage manager is crucial to a lot of mechanisms compiling and running Smalltalk
    - Garbage collection

[9] http://web.cecs.pdx.edu/~harry/musings/SmalltalkOverview.html
[10] http://academic.udayton.edu/SaverioPerugini/PL/www/lecture_notes/parampassing.
[11] RA Finkel, Chapter 5.4.5
[12] http://web.cecs.pdx.edu/~harry/musings/SmalltalkOverview.html

Evaluation

- Restricted aliasing
  - Pointers are implicit[9]
  - Methods employ pass-by-reference (for non-constants and literals)[10]
- Efficiency [11]
  - Smalltalk implements dynamic typing [12] in that it type-checks at runtime
  - Storage manager is crucial to a lot of mechanisms compiling and running Smalltalk
    - Garbage collection
    - Dynamic typing

[9] http://web.cecs.pdx.edu/~harry/musings/SmalltalkOverview.html
[10] http://academic.udayton.edu/SaverioPerugini/PL/www/lecture_notes/parampassing.
[11] RA Finkel, Chapter 5.4.5
[12] http://web.cecs.pdx.edu/~harry/musings/SmalltalkOverview.html

Evaluation

- Restricted aliasing
  - Pointers are implicit[9]
  - Methods employ pass-by-reference (for non-constants and literals)[10]
- Efficiency [11]
  - Smalltalk implements dynamic typing [12] in that it type-checks at runtime
  - Storage manager is crucial to a lot of mechanisms compiling and running Smalltalk
    - Garbage collection
    - Dynamic typing
  - Interpreter executes bytecode to evaluate expressions and methods

[9] http://web.cecs.pdx.edu/~harry/musings/SmalltalkOverview.html
[10] http://academic.udayton.edu/SaverioPerugini/PL/www/lecture_notes/parampassing.
[11] RA Finkel, Chapter 5.4.5
[12] http://web.cecs.pdx.edu/~harry/musings/SmalltalkOverview.html

Evaluation

- Restricted aliasing
  - Pointers are implicit[9]
  - Methods employ pass-by-reference (for non-constants and literals)[10]
- Efficiency [11]
  - Smalltalk implements dynamic typing [12] in that it type-checks at runtime
  - Storage manager is crucial to a lot of mechanisms compiling and running Smalltalk
    - Garbage collection
    - Dynamic typing
  - Interpreter executes bytecode to evaluate expressions and methods
  - Primitive methods implemented directly in machine language

[9] http://web.cecs.pdx.edu/~harry/musings/SmalltalkOverview.html
[10] http://academic.udayton.edu/SaverioPerugini/PL/www/lecture_notes/parampassing.
[11] RA Finkel, Chapter 5.4.5
[12] http://web.cecs.pdx.edu/~harry/musings/SmalltalkOverview.html

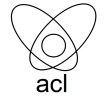
Object Oriented Programming Languages

Object Oriented Programming Languages

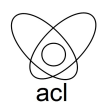
Smalltalk

C++

Java



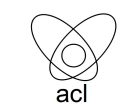
Background



http://groups.engin.umd.umich.edu/CIS/course.des/cis400/cpp/cpp.html

Background

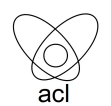
- C++ is an "upgrade" of C to handle object-oriented programming.



http://groups.engin.umd.umich.edu/CIS/course.des/cis400/cpp/cpp.html

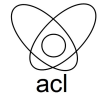
Background

- C++ is an "upgrade" of C to handle object-oriented programming.
- Developed by Bjarne Stroustrup at Bell Laboratories and published initially in 1983



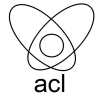
http://groups.engin.umd.umich.edu/CIS/course.des/cis400/cpp/cpp.html

Evaluation



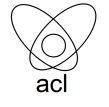
Evaluation

- Data types



Evaluation

- Data types
  - Integers, floating and fixed point real numbers, characters, enumerations as primitive types

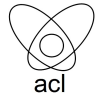


Evaluation

- Data types
  - Integers, floating and fixed point real numbers, characters, enumerations as primitive types
  - Support for multidimentional arrays

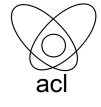
Evaluation

- Data types
  - Integers, floating and fixed point real numbers, characters, enumerations as primitive types
  - Support for multidimentional arrays
  - Explicit pointers



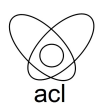
Evaluation

- Data types
  - Integers, floating and fixed point real numbers, characters, enumerations as primitive types
  - Support for multidimentional arrays
  - Explicit pointers
  - Allows for derived structures using `struct` and `typedef` declarations apart from class declarations



Evaluation

- Syntax design (C++ 0x[13])

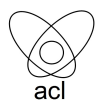


[13] http://slebok.github.io/zoo/cpp/cpp0x/n2723/extracted/index.html

Evaluation

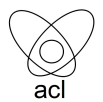
- Syntax design (C++ 0x[13])
  - Production rules: 159
  - Number of top alternatives: 449
  - Number of symbols: 1,835
  - Vocabulary
    - Nonterminal symbols: 162
    - Terminal symbols: 119

[13] http://slebok.github.io/zoo/cpp/cpp0x/n2723/extracted/index.html



Evaluation

- Syntax design (Continuation)



[14] RA Sebesta, Chapter 1.3

Evaluation

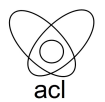
- Syntax design (Continuation)
  - Orthogonality[14]

[14] RA Sebesta, Chapter 1.3

Evaluation

- Syntax design (Continuation)
  - Orthogonality[14]
    - Primitive types and `structs` can be returned from functions, but cannot (technically) be done for arrays.

[14] RA Sebesta, Chapter 1.3



Evaluation

- Syntax design (Continuation)
  - Orthogonality[14]
    - Primitive types and `structs` can be returned from functions, but cannot (technically) be done for arrays.
    - Again, arrays as actual parameters in a function call

[14] RA Sebesta, Chapter 1.3

Evaluation

- Syntax design (Continuation)
  - Orthogonality[14]
    - Primitive types and `structs` can be returned from functions, but cannot (technically) be done for arrays.
    - Again, arrays as actual parameters in a function call
    - `void` pointers and return types

[14] RA Sebesta, Chapter 1.3

Evaluation

- Syntax design (Continuation)
  - Orthogonality[14]
    - Primitive types and `structs` can be returned from functions, but cannot (technically) be done for arrays.
    - Again, arrays as actual parameters in a function call
    - `void` pointers and return types
    - Continued existence of `structs`

[14] RA Sebesta, Chapter 1.3

Evaluation

- Syntax design (Continuation)
  - Orthogonality[14]
    - Primitive types and `structs` can be returned from functions, but cannot (technically) be done for arrays.
    - Again, arrays as actual parameters in a function call
    - `void` pointers and return types
    - Continued existence of `structs`
  - Uniformity

[14] RA Sebesta, Chapter 1.3

Evaluation

- Syntax design (Continuation)
  - Orthogonality[14]
    - Primitive types and `structs` can be returned from functions, but cannot (technically) be done for arrays.
    - Again, arrays as actual parameters in a function call
    - `void` pointers and return types
    - Continued existence of `structs`
  - Uniformity
    - Primitive types are NOT objects/classes

[14] RA Sebesta, Chapter 1.3

Evaluation

- Syntax design (Continuation)
  - Orthogonality[14]
    - Primitive types and `structs` can be returned from functions, but cannot (technically) be done for arrays.
    - Again, arrays as actual parameters in a function call
    - `void` pointers and return types
    - Continued existence of `structs`
  - Uniformity
    - Primitive types are NOT objects/classes
    - Without the explicit use of a (de)referencing operator and excluding pointers, any other type is pass-by-value, EXCEPT arrays, which uses pass-by-reference

[14] RA Sebesta, Chapter 1.3

Evaluation

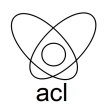
- Syntax design (Continuation)
  - Orthogonality[14]
    - Primitive types and `structs` can be returned from functions, but cannot (technically) be done for arrays.
    - Again, arrays as actual parameters in a function call
    - `void` pointers and return types
    - Continued existence of `structs`
  - Uniformity
    - Primitive types are NOT objects/classes
    - Without the explicit use of a (de)referencing operator and excluding pointers, any other type is pass-by-value, EXCEPT arrays, which uses pass-by-reference
    - Having a `void` pointer, a void return type, but no void typed variable

[14] RA Sebesta, Chapter 1.3

Evaluation

- Syntax design (Continuation)
  - Orthogonality[14]
    - Primitive types and `structs` can be returned from functions, but cannot (technically) be done for arrays.
    - Again, arrays as actual parameters in a function call
    - `void` pointers and return types
    - Continued existence of `structs`
  - Uniformity
    - Primitive types are NOT objects/classes
    - Without the explicit use of a (de)referencing operator and excluding pointers, any other type is pass-by-value, EXCEPT arrays, which uses pass-by-reference
    - Having a `void` pointer, a void return type, but no void typed variable
  - "I made up the term 'object-oriented', and I can tell you I didn't have C++ in mind" - Alan Kay

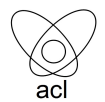
[14] RA Sebesta, Chapter 1.3



Evaluation

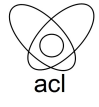
- Syntax design (Continuation)
  - Orthogonality[14]
    - Primitive types and `structs` can be returned from functions, but cannot (technically) be done for arrays.
    - Again, arrays as actual parameters in a function call
    - `void` pointers and return types
    - Continued existence of `structs`
  - Uniformity
    - Primitive types are NOT objects/classes
    - Without the explicit use of a (de)referencing operator and excluding pointers, any other type is pass-by-value, EXCEPT arrays, which uses pass-by-reference
    - Having a `void` pointer, a void return type, but no void typed variable
  - "I made up the term 'object-oriented', and I can tell you I didn't have C++ in mind" - Alan Kay
  - "C++ would make a decent teaching language if we could teach the ++ part without the C part." - Michael B. Feldman.

[14] RA Sebesta, Chapter 1.3



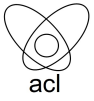
Evaluation

- Abstraction



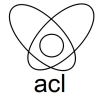
Evaluation

- Abstraction
  - Access grants to variables and procedures can be declared via `public` and `private`



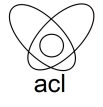
Evaluation

- Abstraction
  - Access grants to variables and procedures can be declared via `public` and `private`
  - `namespace` declarations allow additional scoping of variables



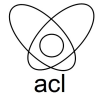
Evaluation

- Abstraction
  - Access grants to variables and procedures can be declared via `public` and `private`
  - `namespace` declarations allow additional scoping of variables
- Expressivity: Rich standard library allows for easier coding of fundamental algorithms and data structures



Evaluation

- Abstraction
  - Access grants to variables and procedures can be declared via `public` and `private`
  - `namespace` declarations allow additional scoping of variables
- Expressivity: Rich standard library allows for easier coding of fundamental algorithms and data structures
- Exception handling: `try`, `catch`, and `throw` keywords



Evaluation

- Restricted aliasing

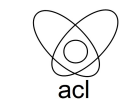
acl

[15] http://www.drdobbs.com/cpp/comparing-cc-compilers/184405450

[16] http://scienceblogs.com/goodmath/2006/11/02/the-c-is-efficient-language-fa/

Evaluation

- Restricted aliasing
  - Explicit pointers



[15] http://www.drdobbs.com/cpp/comparing-cc-compilers/184405450
[16] http://scienceblogs.com/goodmath/2006/11/02/the-c-is-efficient-language-fa/

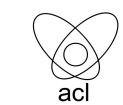
Evaluation

- Restricted aliasing
  - Explicit pointers
  - Methods allow pass-by-reference

[15] http://www.drdobbs.com/cpp/comparing-cc-compilers/184405450
[16] http://scienceblogs.com/goodmath/2006/11/02/the-c-is-efficient-language-fa/

Evaluation

- Restricted aliasing
  - Explicit pointers
  - Methods allow pass-by-reference
- Efficiency:

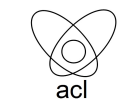


[15] http://www.drdobbs.com/cpp/comparing-cc-compilers/184405450
[16] http://scienceblogs.com/goodmath/2006/11/02/the-c-is-efficient-language-fa/

Evaluation

- Restricted aliasing
  - Explicit pointers
  - Methods allow pass-by-reference
- Efficiency:
  - Type checking: Employs both static (compile time) and dynamic (runtime) type checking



[15] http://www.drdobbs.com/cpp/comparing-cc-compilers/184405450
[16] http://scienceblogs.com/goodmath/2006/11/02/the-c-is-efficient-language-fa/

Evaluation

- Restricted aliasing
  - Explicit pointers
  - Methods allow pass-by-reference
- Efficiency:
  - Type checking: Employs both static (compile time) and dynamic (runtime) type checking
  - C++ compilers are known for their efficiency in terms of memory and compilation time[15] (but of course, there are those that are more efficient especially when heavy number-crunching is involved[16])

[15] http://www.drdobbs.com/cpp/comparing-cc-compilers/184405450
[16] http://scienceblogs.com/goodmath/2006/11/02/the-c-is-efficient-language-fa/

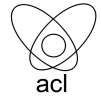
Object Oriented Programming Languages

Object Oriented Programming Languages

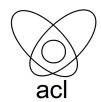
Smalltalk

C++

Java



Background

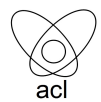


---

http://oracle.com.edgesuite.net/timeline/java/

Background

- Developed in 1991 by the Green Team led by ~~Ryan~~ James Gosling under the defunct Sun Microsystems

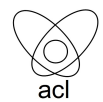


---

http://oracle.com.edgesuite.net/timeline/java/

Background

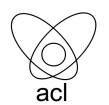
- Developed in 1991 by the Green Team led by ~~Ryan~~ James Gosling under the defunct Sun Microsystems
- Initially called Oak



http://oracle.com.edgesuite.net/timeline/java/

Background

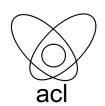
- Developed in 1991 by the Green Team led by ~~Ryan~~ James Gosling under the defunct Sun Microsystems
- Initially called Oak
- The main target of Java was to enhance Web experience



http://oracle.com.edgesuite.net/timeline/java/

Background

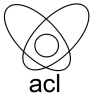
- Developed in 1991 by the Green Team led by ~~Ryan~~ James Gosling under the defunct Sun Microsystems
- Initially called Oak
- The main target of Java was to enhance Web experience
  - In 1994, Java (Oak) was used to develop WebRunner (now known as HotJava), a web browser that supports dynamic elements (e.g. video, reactive input-output mechanisms)
  - Achieved mainstream popularity in 1995



http://oracle.com.edgesuite.net/timeline/java/

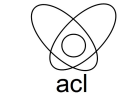
Evaluation

- Data types



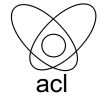
Evaluation

- Data types
  - Integers, floating and fixed point real numbers, characters, enumerations as primitive types



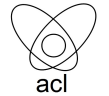
Evaluation

- Data types
  - Integers, floating and fixed point real numbers, characters, enumerations as primitive types
  - Support for multidimentional arrays



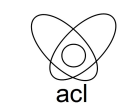
Evaluation

- Data types
  - Integers, floating and fixed point real numbers, characters, enumerations as primitive types
  - Support for multidimentional arrays
  - Allows for derived structures via class declarations



Evaluation

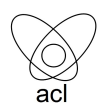
- Syntax design (Java 5.0[17])



[17] http://slebok.github.io/zoo/java/java-5/landman/extracted/index.html

Evaluation

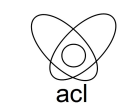
- Syntax design (Java 5.0[17])
  - Production rules: 302
  - Number of top alternatives: 302
  - Number of symbols: 1,782
  - Vocabulary
    - Nonterminal symbols: 110
    - Terminal symbols: 265



[17] http://slebok.github.io/zoo/java/java-5/landman/extracted/index.html

Evaluation

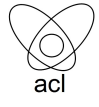
- Syntax design (Java 5.0[17])
  - Production rules: 302
  - Number of top alternatives: 302
  - Number of symbols: 1,782
  - Vocabulary
    - Nonterminal symbols: 110
    - Terminal symbols: 265
  - Grammar is similar to C++, but difference is that it was built primarily as an OOPL rather than something that was built on top of a PL with a different paradigm



[17] http://slebok.github.io/zoo/java/java-5/landman/extracted/index.html

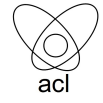
Evaluation

- Abstraction



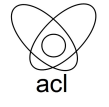
Evaluation

- Abstraction
  - Access grants to variables and procedures can be declared via `public`, `private`, and `protected`



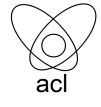
Evaluation

- Abstraction
  - Access grants to variables and procedures can be declared via `public`, `private`, and `protected`
  - `package` declaration for grouping together related classes



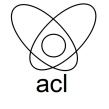
Evaluation

- Abstraction
  - Access grants to variables and procedures can be declared via `public`, `private`, and `protected`
  - `package` declaration for grouping together related classes
- Expressivity: Rich core API set allows for easier coding of fundamental algorithms and data structures



Evaluation

- Abstraction
  - Access grants to variables and procedures can be declared via `public`, `private`, and `protected`
  - `package` declaration for grouping together related classes
- Expressivity: Rich core API set allows for easier coding of fundamental algorithms and data structures
- Exception handling: `try`, `catch`, and `throw` keywords



Evaluation

- Restricted aliasing: Everything is pass-by-value[18,19]

[18] http://jonskeet.uk/java/passing.html
[19] http://javadude.com/articles/passbyvalue.htm
[20] http://www.csi.ucd.ie/staff/jmurphy/publications/1681.pdf
[21] http://www.ibm.com/developerworks/library/j-codetoheap/
[22] http://krebsonsecurity.com/tag/java/

Evaluation

- Restricted aliasing: Everything is pass-by-value[18,19]
- Efficiency

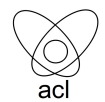
[18] http://jonskeet.uk/java/passing.html
[19] http://javadude.com/articles/passbyvalue.htm
[20] http://www.csi.ucd.ie/staff/jmurphy/publications/1681.pdf
[21] http://www.ibm.com/developerworks/library/j-codetoheap/
[22] http://krebsonsecurity.com/tag/java/

Evaluation

- Restricted aliasing: Everything is pass-by-value[18,19]
- Efficiency
  - Employs both static (compile time) and dynamic (runtime) type checking

[18] http://jonskeet.uk/java/passing.html
[19] http://javadude.com/articles/passbyvalue.htm
[20] http://www.csi.ucd.ie/staff/jmurphy/publications/1681.pdf
[21] http://www.ibm.com/developerworks/library/j-codetoheap/
[22] http://krebsonsecurity.com/tag/java/



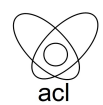


Evaluation

- Restricted aliasing: Everything is pass-by-value[18,19]
- Efficiency
  - Employs both static (compile time) and dynamic (runtime) type checking
  - Memory inefficiency[20,21]

[18] http://jonskeet.uk/java/passing.html
[19] http://javadude.com/articles/passbyvalue.htm
[20] http://www.csi.ucd.ie/staff/jmurphy/publications/1681.pdf
[21] http://www.ibm.com/developerworks/library/j-codetoheap/
[22] http://krebsonsecurity.com/tag/java/



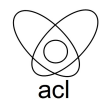


Evaluation

- Restricted aliasing: Everything is pass-by-value[18,19]
- Efficiency
  - Employs both static (compile time) and dynamic (runtime) type checking
  - Memory inefficiency[20,21]
  - Security vulnerabilities[22]

[18] http://jonskeet.uk/java/passing.html
[19] http://javadude.com/articles/passbyvalue.htm
[20] http://www.csi.ucd.ie/staff/jmurphy/publications/1681.pdf
[21] http://www.ibm.com/developerworks/library/j-codetoheap/
[22] http://krebsonsecurity.com/tag/java/





END OF SESSION 5