Principles
of
Programming Language

[BE SE-6th Semester]

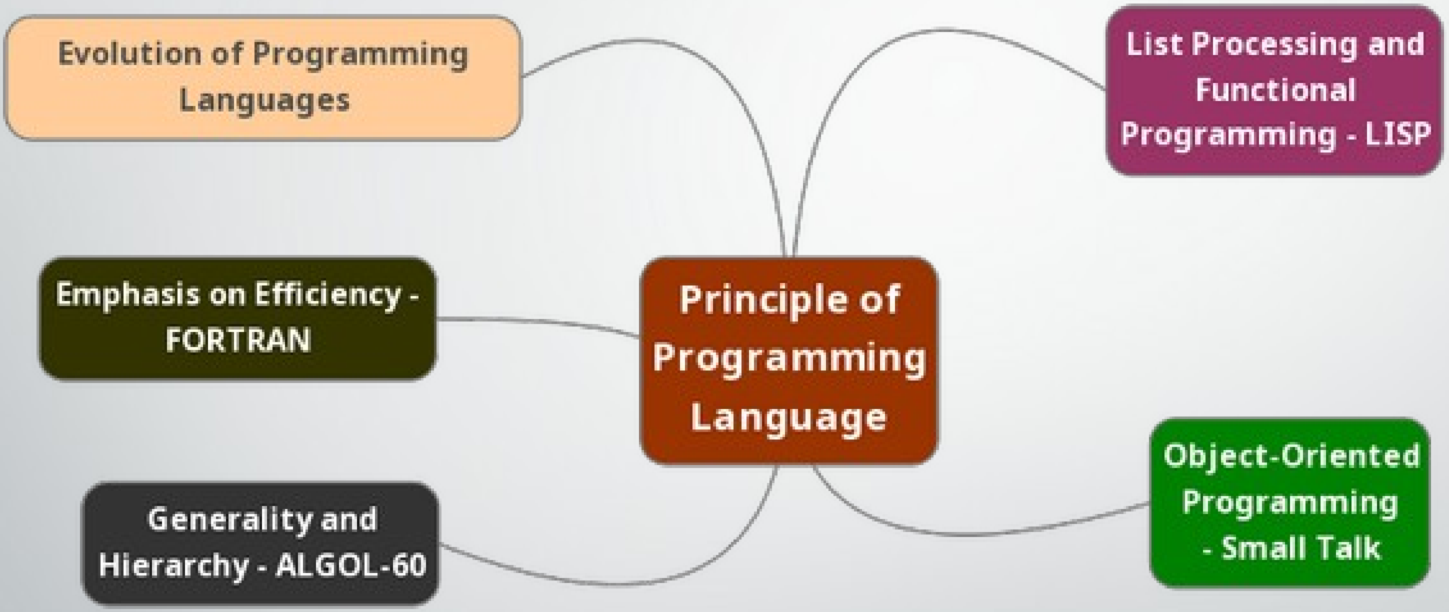
Rishi K. Marseni

Textbook:

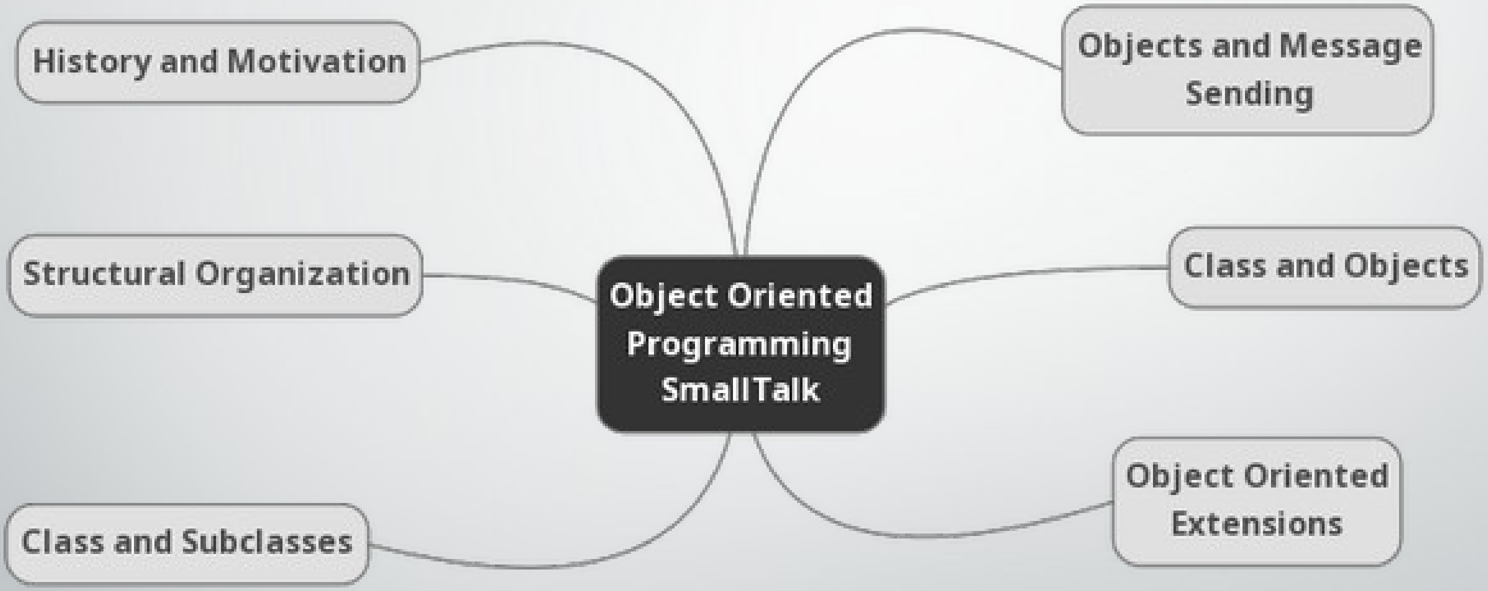
Principles of programming languages: design, evaluation, and implementation.

Author: Bruce J. MacLennan

Principle of Programming Language


Evolution of Programming Languages
List Processing and Functional Programming - LISP
Emphasis on Efficiency - FORTRAN
Principle of Programming Language
Generality and Hierarchy - ALGOL-60
Object-Oriented Programming - Small Talk

Object Oriented Programming
Small Talk


Object Oriented Programming SmallTalk
History and Motivation
Structural Organization
Class and Subclasses
Objects and Message Sending
Class and Objects
Object Oriented Extensions

Smalltalk Quotes

Common languages are tools, Smalltalk is a piece of art.

Alan Kay:

“The best way to predict the future is to invent it”

“Simple things should be simple. Complex things should be possible”

Object Oriented Programming

- First goal: Model the *objects* of the world
  - Noun-oriented
    - (in contrast to verb-orientation in Structured Programming)
  - Focus on the *domain* of the program
- Phases
  - Object-oriented **analysis**: Understand the domain
    - Define an object-based model of it
  - Object-oriented **design**: Define an implementation strategy
    - Design the solution
  - Object-oriented **programming**: Build it

Object Oriented Programming

- Primary object-oriented language concepts

  - dynamic lookup - polymorphism
  - encapsulation - abstraction
  - inheritance
  - sub typing

Objects

- An object consists of
  - hidden data
    - instance variables, also called member data
    - hidden functions also possible
  - public operations
    - methods or member functions
    - can also have public variables in some languages

hidden data	
msg_1	method_1
. . .	. . .
msg_n	method_n

- Object-oriented program:
  - Send messages to objects

Object Orientation

- Programming methodology
  - organize concepts into objects and classes
  - build extensible systems
- Language concepts
  - dynamic lookup - polymorphism
  - Abstraction-encapsulation
  - subtyping allows extensions of concepts
  - inheritance allows reuse of implementation

Dynamic Lookup

- In object-oriented programming,

  object <- message (arguments)

  code depends on object and message

- In conventional programming,

  operation (operands)

  meaning of operation is always the same

Fundamental difference between abstract data types and objects

Example

- Add two numbers x <- add (y)
  different add if x is integer, complex, etc.

- Conventional programming add (x, y)
  function add has fixed meaning

- Very important distinction:
  - Overloading is resolved at compile time,
  - Dynamic lookup at run time

Language concepts

- “dynamic lookup”
  - different code for different object
  - integer “+” different from real “+”
- encapsulation
- sub typing
- inheritance

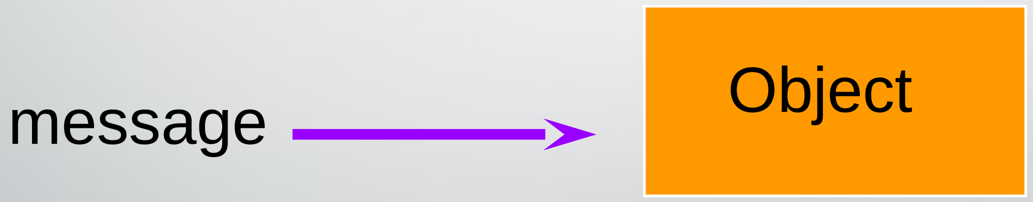
How’d we get from there to here?

- How did we move from structured to object-oriented?

- Alan Kay and his desire to make software better
    - More robust
    - More maintainable
    - More scalable

Encapsulation

- Builder of a concept has detailed view
- User of a concept has “abstract” view
- Encapsulation separates these two views



Encapsulation

Encapsulation

- How does it support our objectives?
  - Models real world
    - examples: driving a car
  - Reusability & Ease of use
    - other users need only know ‘interface’, not the implementation

Comparison with ADTs

- Traditional (non-OO) approach to encapsulation is through abstract data types

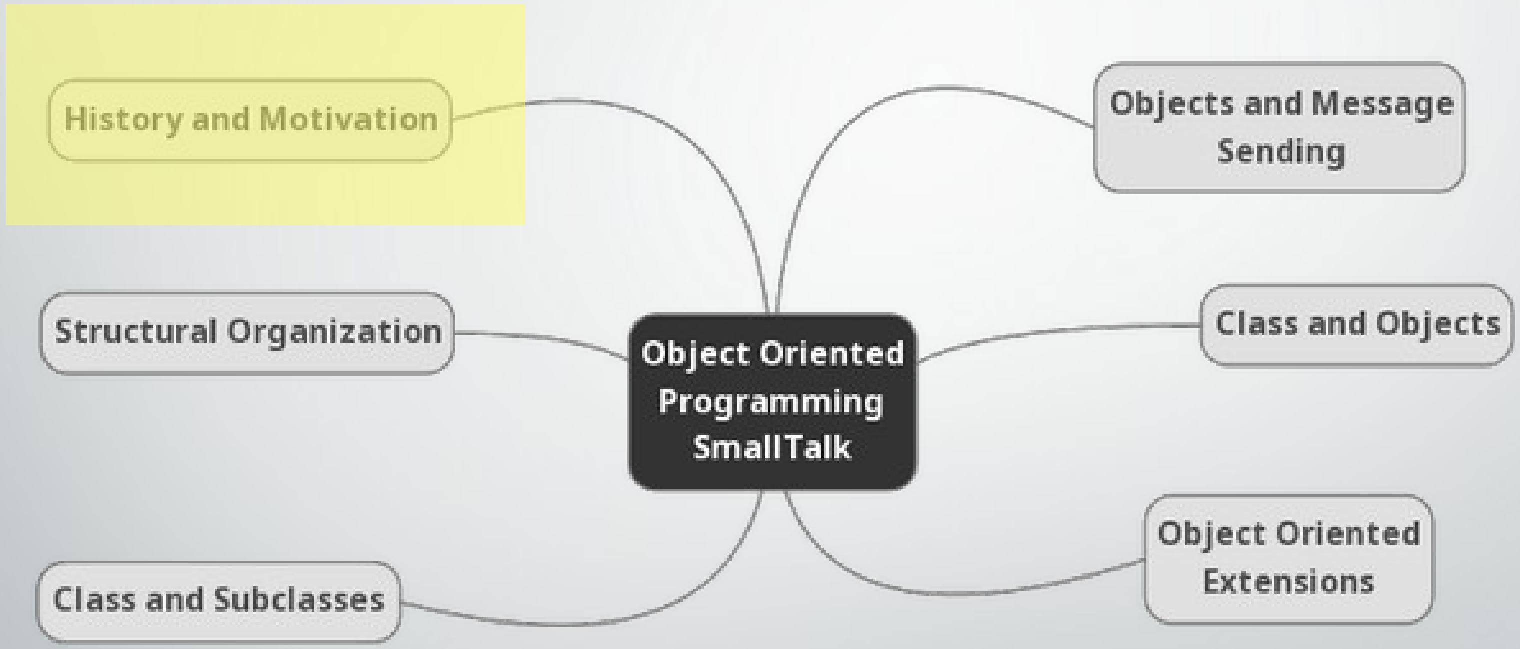
- Advantage
  - Separate interface from implementation
- Disadvantage
  - Not extensible in the way that OOP is

Abstract Data Types (ADTs)

- Guarantee invariants of data structure
  - only functions of the data type have access to the internal representation of data
- Limited “reuse”
  - No inheritance

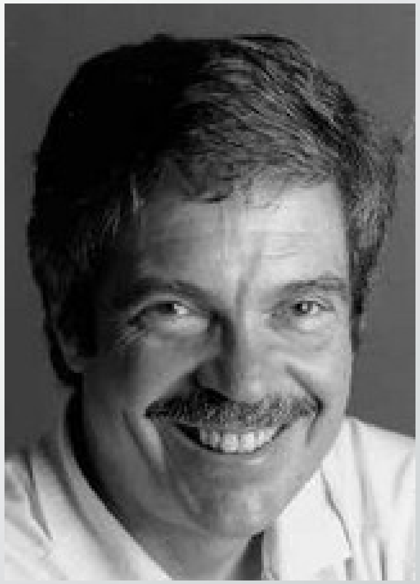
- Data abstraction is important part of OOP, innovation is that it occurs in an *extensible* form

Object Oriented Programming
Small Talk


History and Motivation
Objects and Message Sending
Structural Organization
Object Oriented Programming SmallTalk
Class and Objects
Object Oriented Extensions
Class and Subclasses

Alan Kay

- University of Utah PhD student in 1966
  - Read Sketchpad, Ported Simula
- Saw “objects” as the future of computer science
- His dissertation: Flex, an object-oriented *personal* computer
  - A *personal* computer was a radical idea then
  - How radical?



"There is no reason anyone would want a computer in their home." (Ken Olsen, Digital Equipment Corp, **1977**)

Alan Kay

Alan C. Kay: ”*In the 1990’s there will be millions of personal computers. They will be the size of notebooks of today, have high-resolution flat-screen reflective displays, weigh less than ten pounds, have ten to twenty times the computing and storage capacity of an Alto. Let’s call them Dynabooks. The purchase price will be about that of a color television set of the era, …*” (1971)

Alan Kay

- Alan Kay’s categorization of Programming Languages:
  - Agglutination of Features
    - Cobol, PL/1, Ada, etc.
  - Crystallization of Style
    - Lisp, APL, and Smalltalk

Birth of Objects → Sketchpad

- First object-oriented drawing program
- Master and instance drawings
  - Draw a house
  - Make two instances
  - Add a chimney to the master
  - Poof! The instances grow a chimney

Birth of Objects → Simula

- Simulation programming language from Norway, 1966 (Kristen Nygaard & Ole-Johan Dahl)
- Define an *activity* which can be instantiated as *processes*
- Each process has its own data and behavior
  - In real world, objects don't mess with each others' internals directly
- (Simulated) Multi-processing
  - No Universal Scheduler in the Real World

Birth of Objects

- Objects as models of real world entities
- Objects as Cells
  - Independent, indivisible, interacting -- in standard ways
- Scales well
  - Complexity: Distributed responsibility, *growing* complexity
  - Robustness: Independent
  - Supporting growth: Same mechanism everywhere
  - Reuse: Provide services, just like in real world

A Personal Computer for Children

- Flex, an object-oriented *personal* computer
- Alan Kay is referred to as the "father of the personal computer"
- An early prototype of personal computing
- Kay's ideas contributed to the transformation of the computer from a calculating machine to a communication medium

Flex

Xerox Alto I (1973)

Smalltalk Inventions

- Interface:
  - overlapping Windows
  - Icons, even iconic programming
  - Pop-up menus
  - Mouse as Pointing device
- Object-oriented programming
- Multimedia authoring environment: Drawing, music, animations

1981: Xerox releases Smalltalk-80

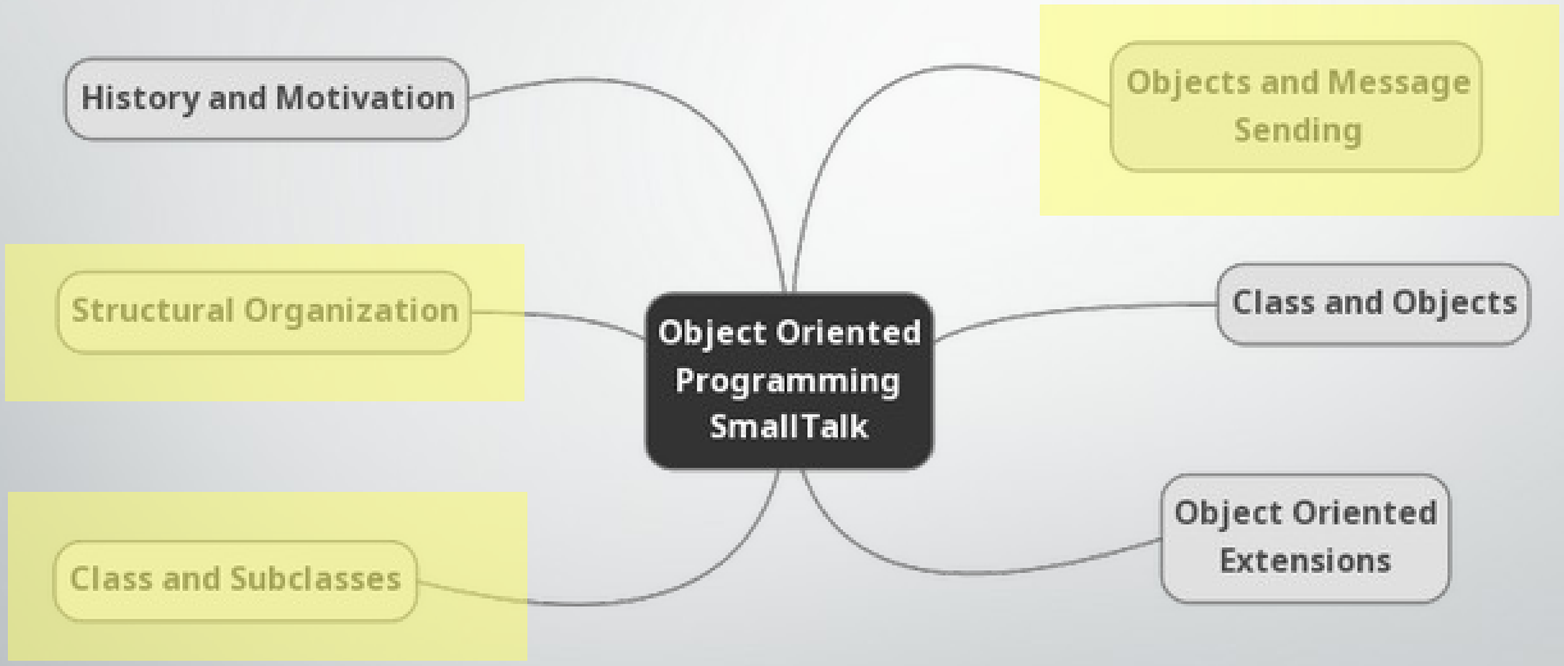
Why Smalltalk ?

- Pure object-oriented language
- Uniform
- Simple
- Powerful
- Dynamically typed (no type coercion…)
- No primitive types
- Syntax simple so force to think !!!!!
- Discuss design, OO modeling and not syntax!!

Why Smalltalk ?

- Object Culture
- Environment completely written in Smalltalk
- Powerful development tools
- Good to model real world applications

Object Oriented Programming Small Talk


History and Motivation
Objects and Message Sending
Structural Organization
Object Oriented Programming SmallTalk
Class and Objects
Object Oriented Extensions
Class and Subclasses

Part 2

- Structural Organization
- Classes and Subclasses
- Objects and Message Sending

Smalltalk: Interactive & Interpreted

- To satisfy Dynabook's requirements, it has to be highly interactive.
  → communication with either pointing or typing commands.
- Primary ways of defining things:
  - Bind a name to an object
    - X←3
    - y←x+1
  - Class definition

    x*2 => send message '*2' to object x

Important Ideas

1-Objects have a behavior.

2-Objects can be made to do things by sending them messages.

3-Repetitive operations can be simplified by using control structures.

Objects have a behavior

- Instantiation is also a message sent to some object.
- Which object should the message be sent to?
  - A universal system object responsible for all instantiations
  - The class relating the object.
- For the sake of Information Hiding Principle: Each class is responsible for its own instantiation.
- anotherScribe← pen newAt: 200@800

Class Definition

- To draw another box

  - Move the pen

  - Instantiate a new pen

    → Violates the Abstraction Principle

- A better solution:

  define a class *box* which can be instantiated any number of times

Example : box class

- Class definition written in tabular form

class name	box
instance var names	loc tilt size scribe
instance messages and methods	

```
shape ||
    scribe penup; goto: loc; turnTo: tilt; pendn.
    4 timesRepeat: [scribe go: size; turn: 90]

erase ||
    scribe color background.
    self shape
```

Class Group Related Objects

- In the real world, objects are individuals, they differ from one- another
- If all objects were totally different, it would’ve been impossible to understand the world or act effective
- Objects have many common properties and act similarly.

  → therefore we abstract these out
- The resulting abstraction, or class, retains the similar properties and omit particulars that distinguishes one individual from another

Class Group Related Objects

- *Class definition* specifies all of the properties & behaviors common to all instances of the class.
- *Instance variable* in the object contain the particular information.
- The behavior of the members of the class(messages being responded), is called the *protocol* of the class (is determined by the instance methods)

Hierarchical Classification

- When some objects respond to certain common messages:

  There is Subclass and Superclass
- Subclass *inherits* the properties and methods of the superclass
- All other classes are instances(perhaps indirectly) of the class object.

Overloading is implicit and inexpensive

- Ex. [3+5] the object 3 responds to the message +5 and returns 8
- Ex. [“book” + “keeper”] the object “book” responds to the message +”keeper” and returns “bookkeeper”
- There is no operator identification problem because the system always knows the class to which an object belongs

Multiple DT Representation

- Classes simplify having several representations for one abstract datatype

Ex. indexStack → array

linkStack → link list

They work with ‘push’ ‘pop’ ‘empty’

- Same protocol (interface) → can be used interchangeably
- Thus Smalltalk applies… principles!!
  - Information Hiding Principle
  - Manifest Interface Principle

Self-Displaying Objects

Every displayable object respond to the message *print* by returning a character string form of itself

Ex. w= 1+2i

z=2+5i

(w+z) print

➔ “3+7i”

Methods accept any object with proper protocol

- It is possible at any time to define a new class and have many existing methods already applicable to it.
- X+Y can work, either integer or polynomial because:

  The object x is responsible for responding to the message +y; if x is a number, it does simple addition; if x is polynomial, it does polynomial addition

Hierarchical subclasses preclude orthogonal classification

In Smalltalk , classes can be immediate subclasses of exactly *one* other class.

Solution?

- There are different sorts of classification in the real-world which are orthogonal

- Multiple Inheritance may be a solution; but:
  - What is decided a method or property exists in both superclasses?
  - What is decided if something is inherited directly and indirectly?

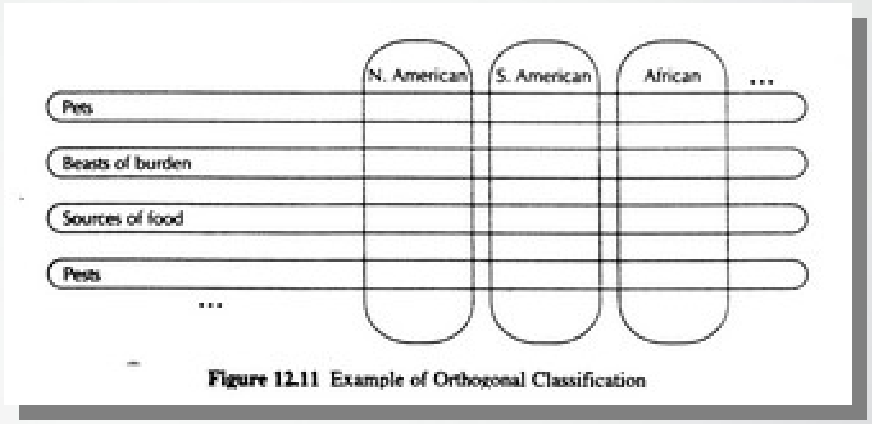


Figure 12.11 Example of Orthogonal Classification

Everything in Smalltalk is an object

- In Smalltalk, even classes are objects. This design decision satisfies… principles!
  - Regularity

Simplicity Principle

- A language should be simple as possible, There should be a minimum number of concepts, with simple rules for their combination.

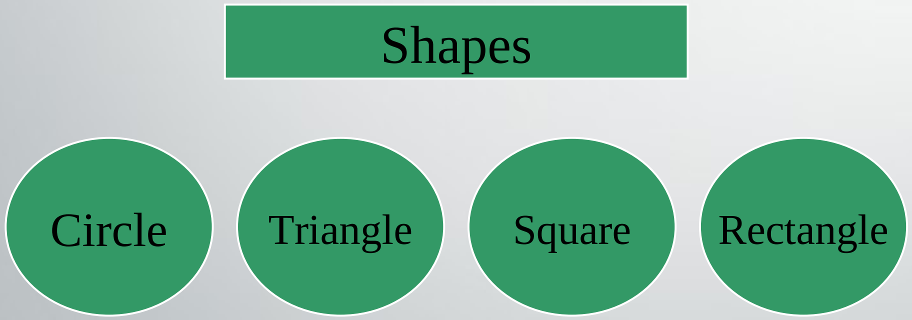
- In Smalltalk, even classes are objects. This design decision satisfies… principles!
  - Regularity
  - Simplicity

What are Objects?

- Objects have characteristics of both data (e.g. quantities, properties) and programs(they *do* things).
- The set of messages to which an object can respond is called its *protocol*
- When an object is sent a message, the Smalltalk system produces an error diagnostic if that message is not part of the object’s protocol.
- Objects are instantiated dynamically

Names Are Not Typed

- Variables in Smalltalk don’t have type!
- Why is this good?
  - Polymorphism!



```
shape1 draw.
shape2 draw.
shape3 draw.
```

Names Are Not Typed

- What about type checking?
  - Type checking occurs when a message is sent to the object bound to a name. If the object responds to that message(i.e. the message is in its protocol), then the message is legal; otherwise it is not.
- Smalltalk like LISP, has strong, but dynamic, typing.

Names Are Not Typed

- Does Smalltalk’s dynamic type checking violate the Security Principle?
  - No! Smalltalk system will allow a message to be sent to an object only if that object has a method to respond to the message.

Error Detection

- Instead of machine-code, Smalltalk has all of the source code available at run-time.
- In case of an error, Smalltalk can produce run-time diagnostics.
- A run-time error causes the program to be suspended. An offending class can be edited and quickly recompiled and the execution of the program can be resumed

Storage Utilization

- Every object is accessed through an *object reference, that* is, a pointer. Therefore all variables and parameters occupy the same amount of storage-one pointer.

Static typing:
more Documentation

- The designers of Smalltalk claim that well-named variables provide just as good documentation.
- For E.g. calling a parameter anInteger makes its intended value just as clear as a typed declaration like n: integer

Dynamic Typing & Flexibility

- Any object with the proper protocol can be passed to a method.
- A major application of Abstraction Principle: common algorithms can be factored out of a system without complicated mechanisms.

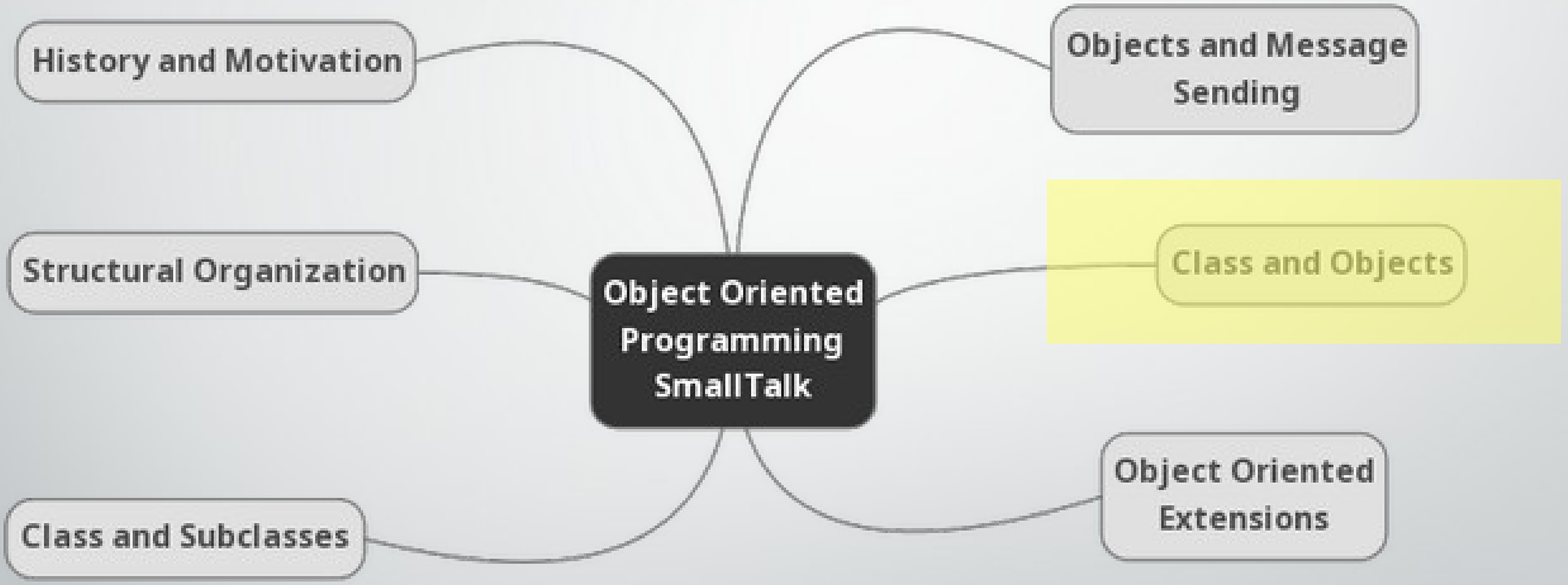
Three Kinds of Message Syntax

- Parameterless messages
  - B1 show
- Unary messages
  - x+y
- Messages with one or more parameters
  - Scribe grow:100

Satisfies the… Principle.

  - *Zero-One-Infinity*

Object Oriented Programming
Small Talk


History and Motivation
Structural Organization
Class and Subclasses
Object Oriented Programming SmallTalk
Objects and Message Sending
Class and Objects
Object Oriented Extensions

Part 3

- Object Representation
- Class Representation
- Activation Record Representation
- Message
  - Sending
  - Returning

Storage Manager

- Managing Free Space
  - Reference Counting with extension for cyclic structures

Interpreter

- Heart of the Smalltalk system
- Interpret Smalltalk written form or intermediate form (more efficient)
- Abstract data type manager for methods

Primitive Subroutines

- Collection of methods that, for performance reasons, are implemented in machine code
- Basic Input-Output functions, integer arithmetic, basic screen graphics operations, …

Three Central Ideas in Smalltalk

- Objects
- Classes
- Message sending

Objects Representation

- Abstraction and Information Hiding Principles
- Just information that varies from object to object (instance variables)
- A pointer to the data structure representing the class

Example : box class

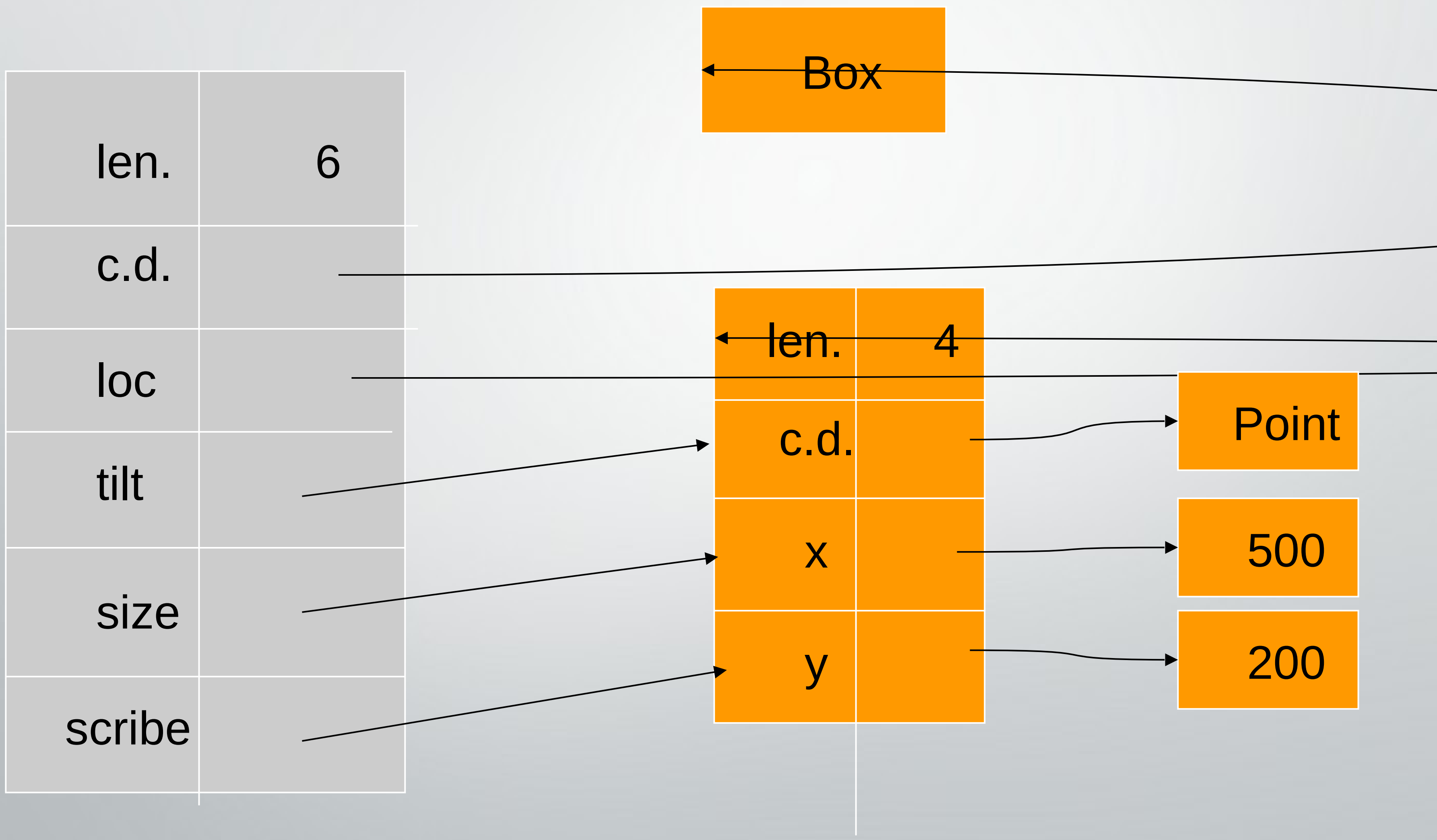
- Class definition written in tabular form

class name	box
instance var names	loc tilt size scribe
instance messages and methods	

```
shape ||
    scribe penup; goto: loc; turnTo: tilt; pendn.
    4 timesRepeat: [scribe go: size; turn: 90]

erase ||
    scribe color background.
    self shape
```

Objects Representation



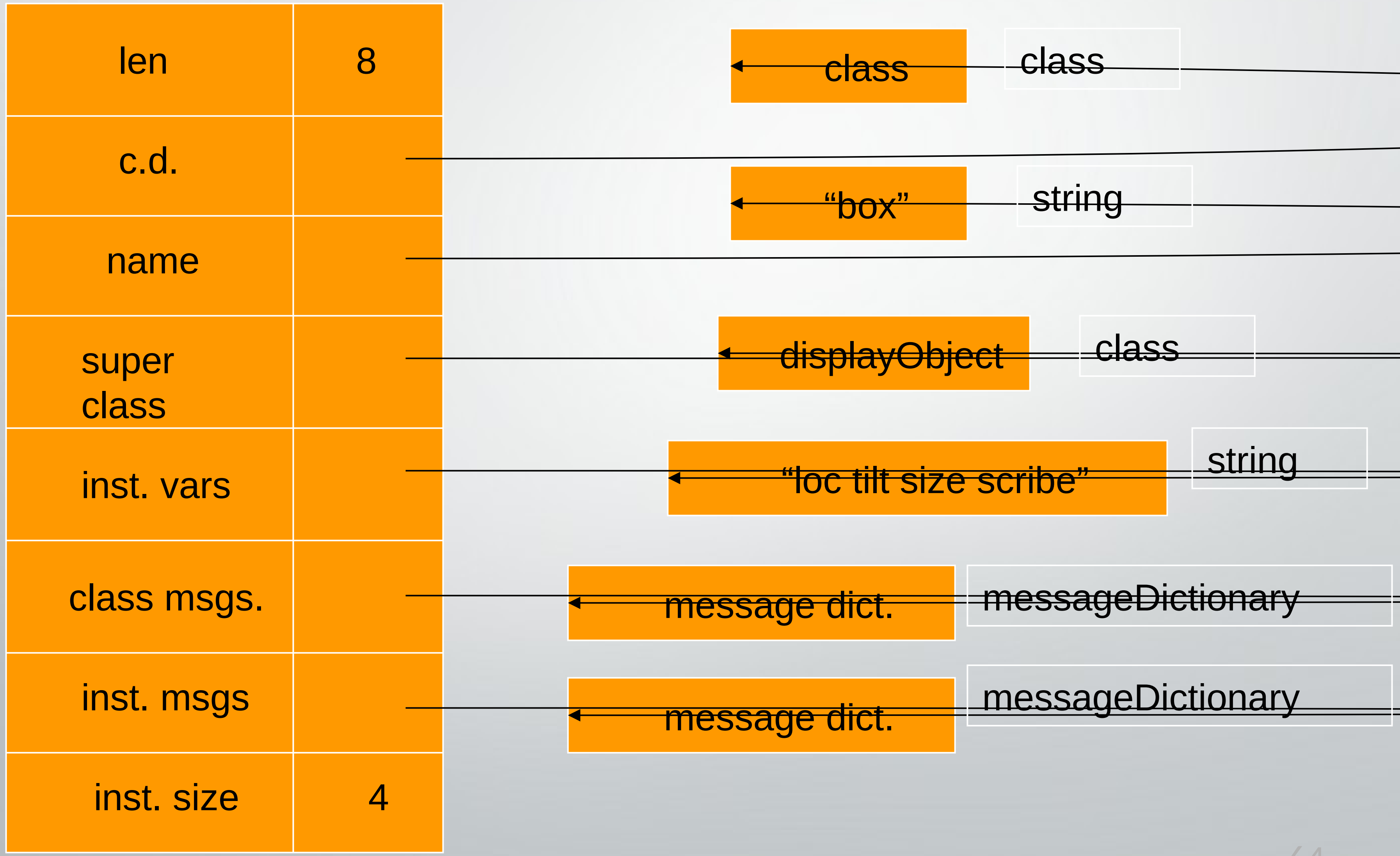
Class Representation

- Every thing is an object without exception

(Regularity Principle)

- An object of a class named “class”
- Information for representing
  - Class name
  - Super class name
  - Instance variable names
  - Class message dictionary
  - Instance message dictionary

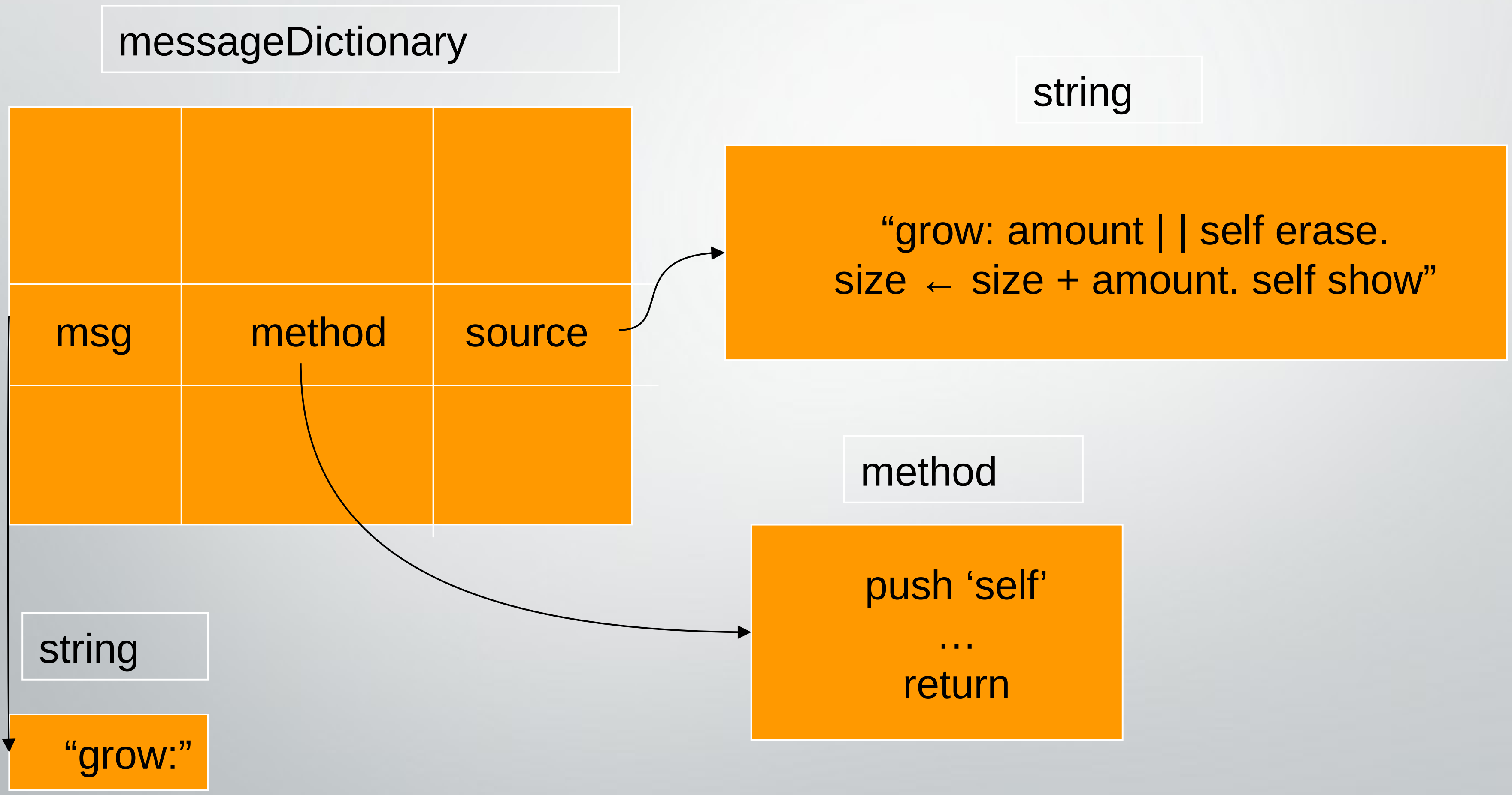
Class Representation



Message Dictionary

- Finding the message template in message dictionary by using hashing techniques
- Contains two entry for each message template
  - Source form
    - For editing and displaying class definitions
  - Compiled form
    - For rapidly interpreting

Message Dictionary



Message Sending Representation

message sending in Smalltalk and procedure calls in other languages

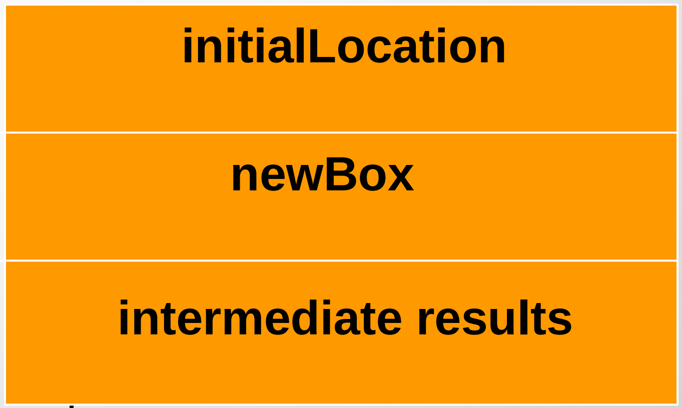
- Very similar implementation techniques
- Activation Record
- Some important differences

Activation Record Structure

- Sender Part
  - A dynamic link to the sender’s activation record (just a pointer)
- Instruction Part
  - Object pointer
  - Relative offset
  - Goes through the storage manager
    (Information Hiding Principle)

Activation Record Structure

- Environment Part
  - Local environment
    - Parameters
    - Temporary variables
    - Intermediate results
    - Example
      - newAt: initialLocation | newBox |
        newBox ← box new.
        …



Activation Record Structure

- Non-local environment
    - All other visible variables
    - Namely instance variables and class variables
    - A simple pointer to that object, ***static link***

Activation Record Structure

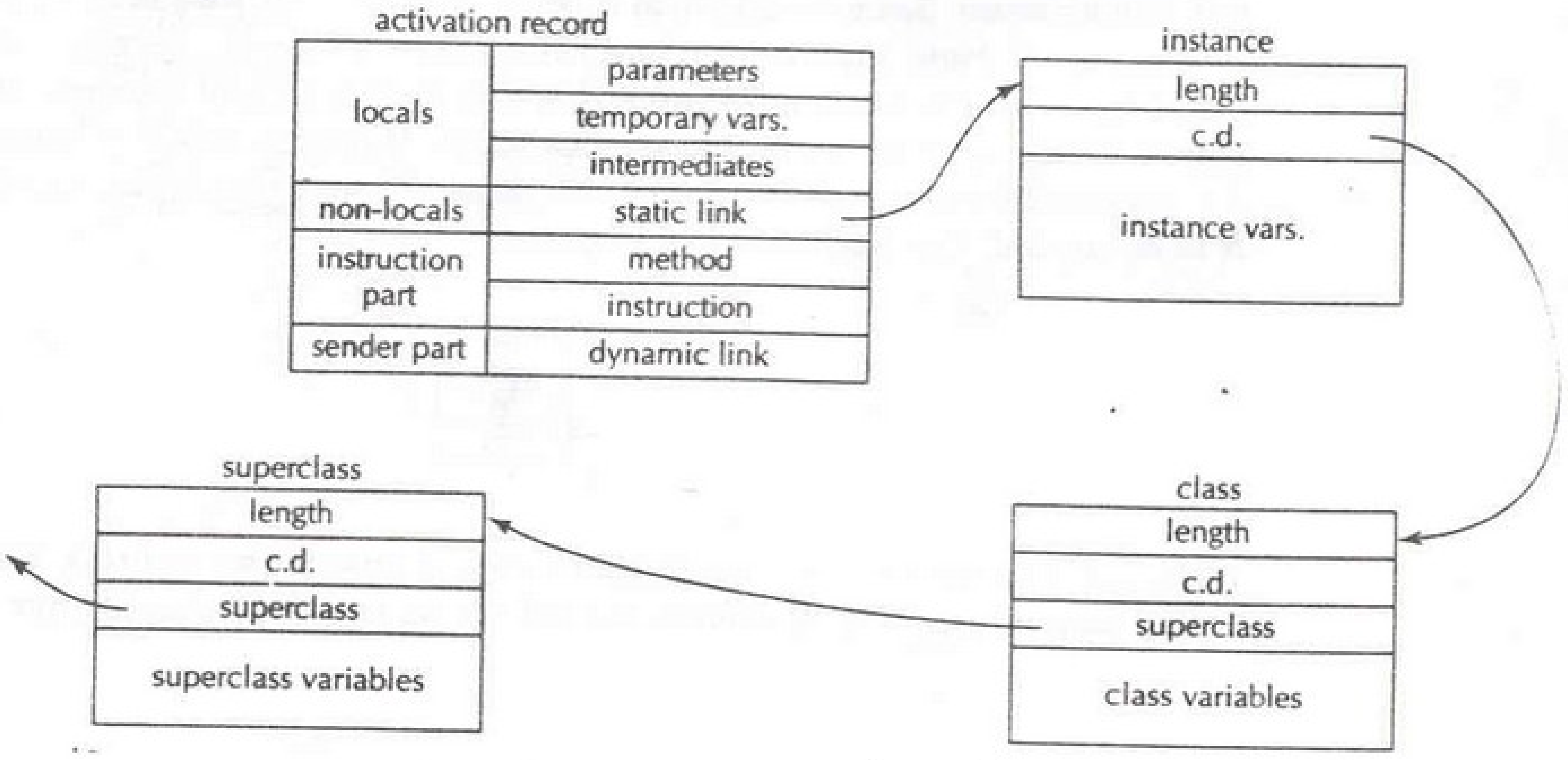


Figure 12.17 Parts of an Activation Record

Message Sending and Returning

- When a message is sent to an object
  - Create callee’s activation record
  - Identify the method by looking in the message dictionary for that class or its super class or ….
  - Transmit the parameters
  - Suspend the sender and saving its state in its activation record
  - Establish the dynamic link and activate receiver’s activation record

Message Sending and Returning

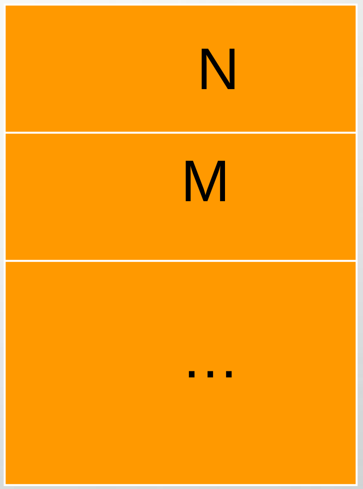
- Returning from the method
  - Transmit the returned object (if any) back to the sender
  - Resume execution of sender
- Why omitting the deallocation of the receiver’s activation record?
- Storage manager
- Information Hiding Principle
- Not in the stack
- Less efficient, but more simple and regular

Message Sending and Returning

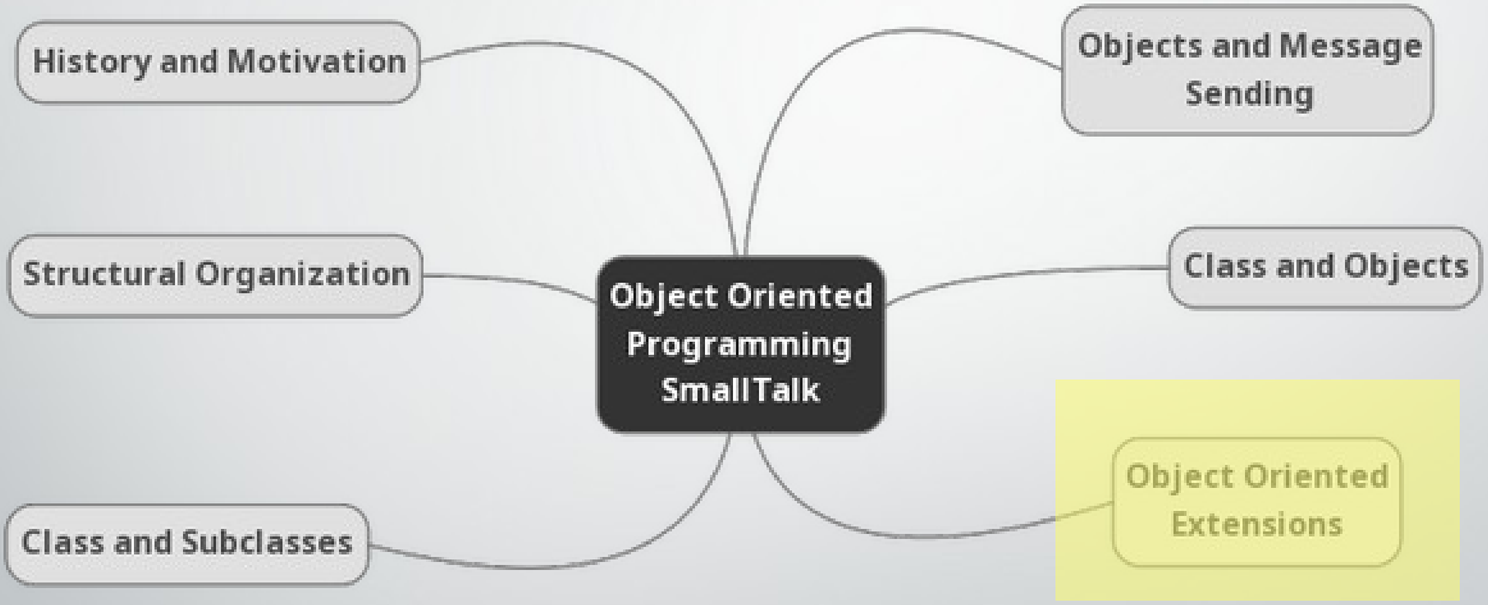
- Another reason, ***concurrency***
- The scheduler must wait for each task to return from its ***run*** method, **an important limitation**
- Real Smalltalk systems provide interrupt facility
- Time sharing

Message Sending and Returning

- What does happen if we use stack?
- Consider this example
- Two possibilities:
  - Popping N and M
  - Popping M from middle of the stack
- No one is correct
- Stack is appropriate for LIFO discipline



Object Oriented Programming
Small Talk


History and Motivation
Objects and Message Sending
Structural Organization
Object Oriented Programming SmallTalk
Class and Objects
Object Oriented Extensions
Class and Subclasses

Part 4

- Smalltalk Terminology
- Comparison with other OO languages
- Conclusions

Smalltalk Language Terminology

- Object Instance of some class
- Class Defines behavior of its objects
- Selector Name of a message
- Message Selector together with parameter values
- Method Code used by a class to respond to message
- Instance variable Data stored in object
- Subclass Class defined by giving incremental modifications to some super class

Smalltalk vs. C++ and Java ?

Smalltalk

- “Everything is an object”
- Objects are passed by reference
- Objects are the units of encapsulation

C++

- “Everything is a structure”
- Objects are passed by value (pointers)
- Classes are the units of encapsulation

Java

- “Almost everything is an object”
- Objects are passed by reference (no pointers)
- Classes are the units of encapsulation (like C++)

Syntax Differences

- Languages:

	C++ / Java	Smalltalk
Comments	/*comments*/ //comments	“comment”
Assignments	int max=100;	Variable **:=** value
Basic Types	“string”	‘string’
Self Reference	this this.getClass()	self self class

C++ compared to Smalltalk

- C++ is…
  - Mixed paradigm programming: objects, but can have functions, too
  - Compiled to native code (Recall: main goal is efficiency)
  - Based on traditional functions and stack-based scoping
  - Strongly typed
  - Storage is controlled by the programmer
- Smalltalk is…
  - Dynamic (feels like an interpreter)
  - Byte-code compiled
  - Persistent objects
  - Not at all typed
  - Garbage collection: storage is managed by the system

Java compared to the others

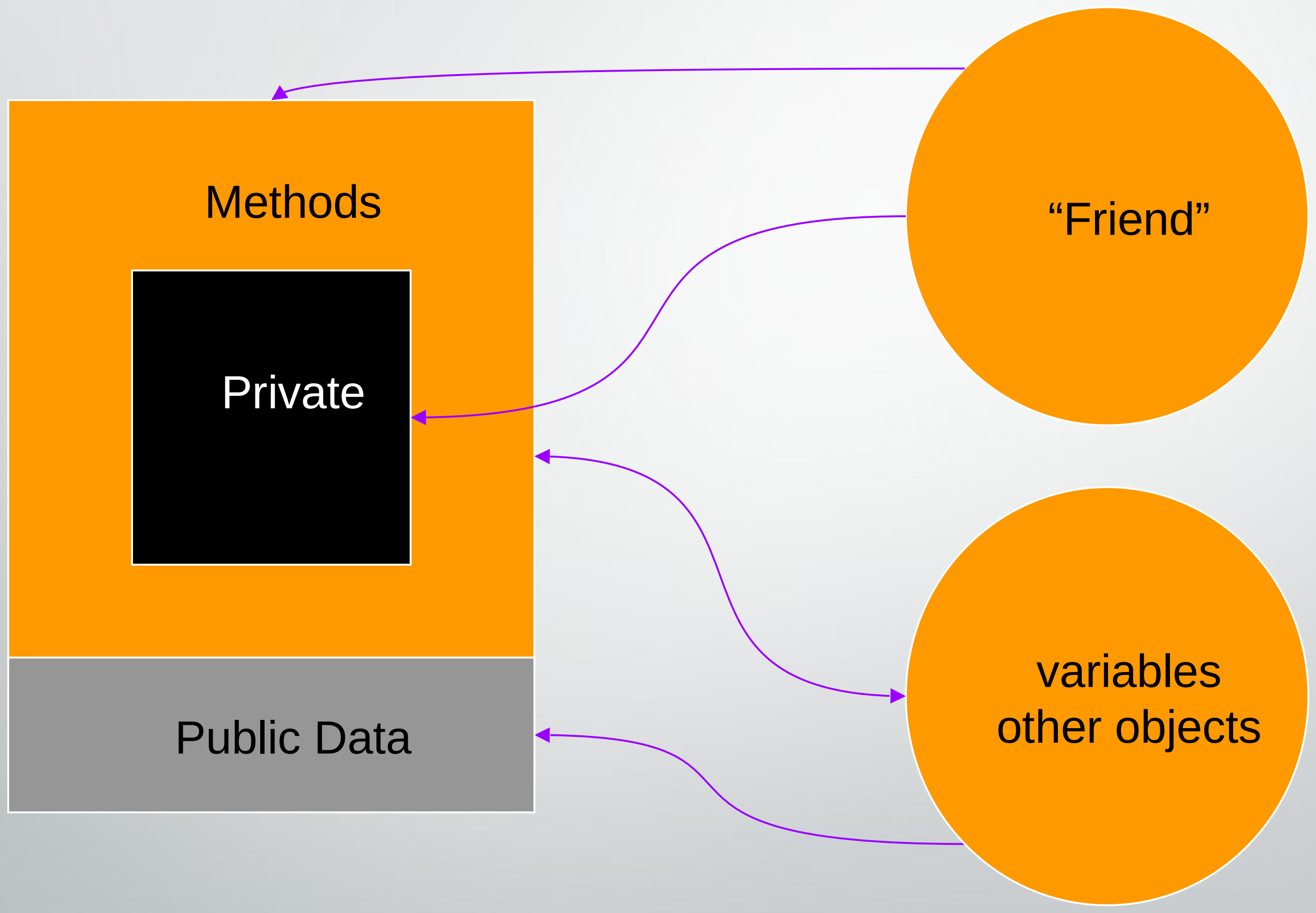
- Java looks like C++, but…
  - Mostly objects (no functions, but some primitive types)
  - Uses a VM
  - Objects are more like C++'s than Smalltalk’s
  - Even more strongly typed
  - Storage is handled by garbage collection

Encapsulation (Smalltalk vs. C++)

<u>In C++:</u>

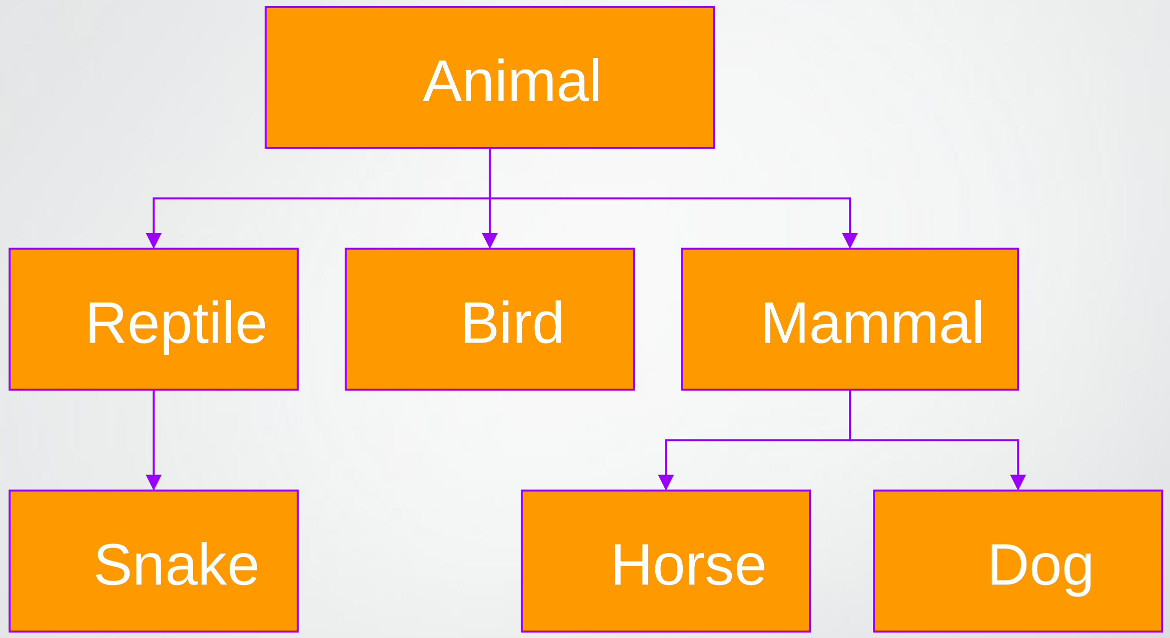
- not everything is an object
- pseudo message passing through member functions
- not all of an object’s state is encapsulated
  - public data variables
  - keyword ‘friend’

Encapsulation (Smalltalk vs. C++)



Inheritance (Smalltalk vs. C++)

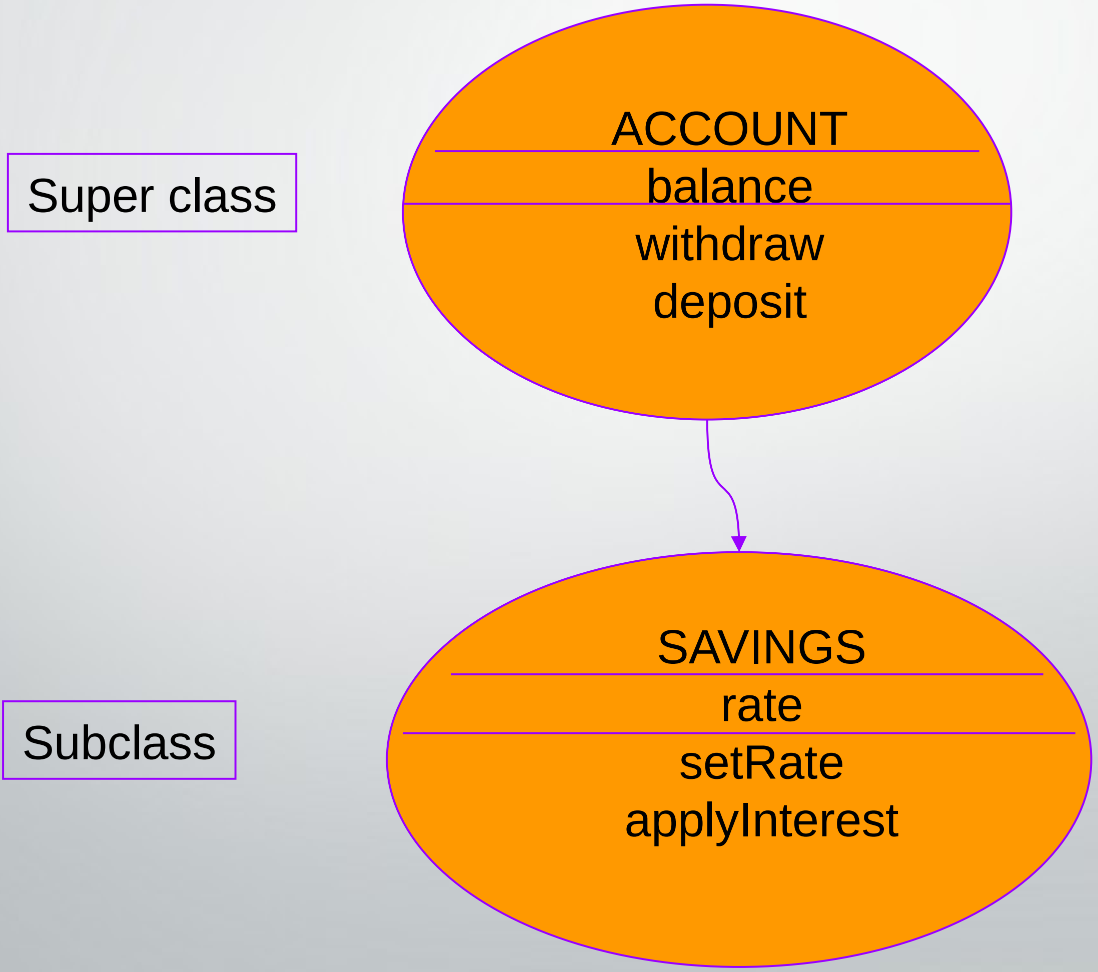
In Smalltalk:



- Generalization & Specialization
- Subclass & superclass
- Every class has a superclass

Inheritance (Smalltalk vs. C++)

In Smalltalk:



Inheritance (Smalltalk vs. C++)

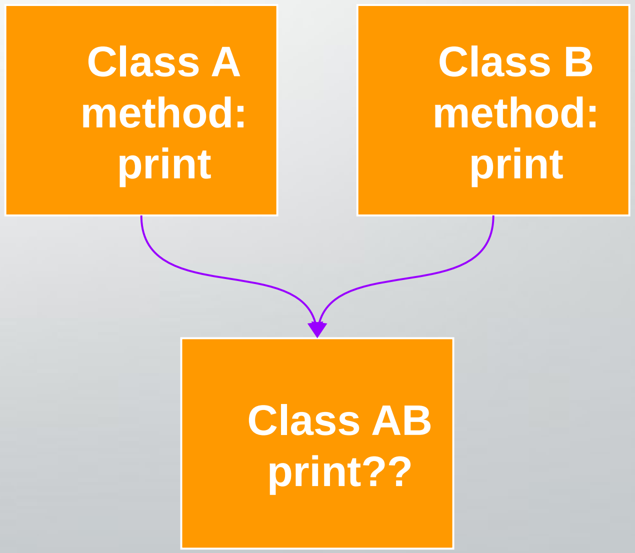
In Smalltalk:

- How does it support our objectives?
  - Models real world
    - group similar ‘things’ into class hierarchy
  - Reusability & Ease of use
    - create new classes by specializing existing ones

Inheritance (Smalltalk vs. C++)

- Inheritance in C++
  - public, private, protected
  - burden on programmer

- Multiple Inheritance
  - Supported in C++
  - Not supported in Smalltalk

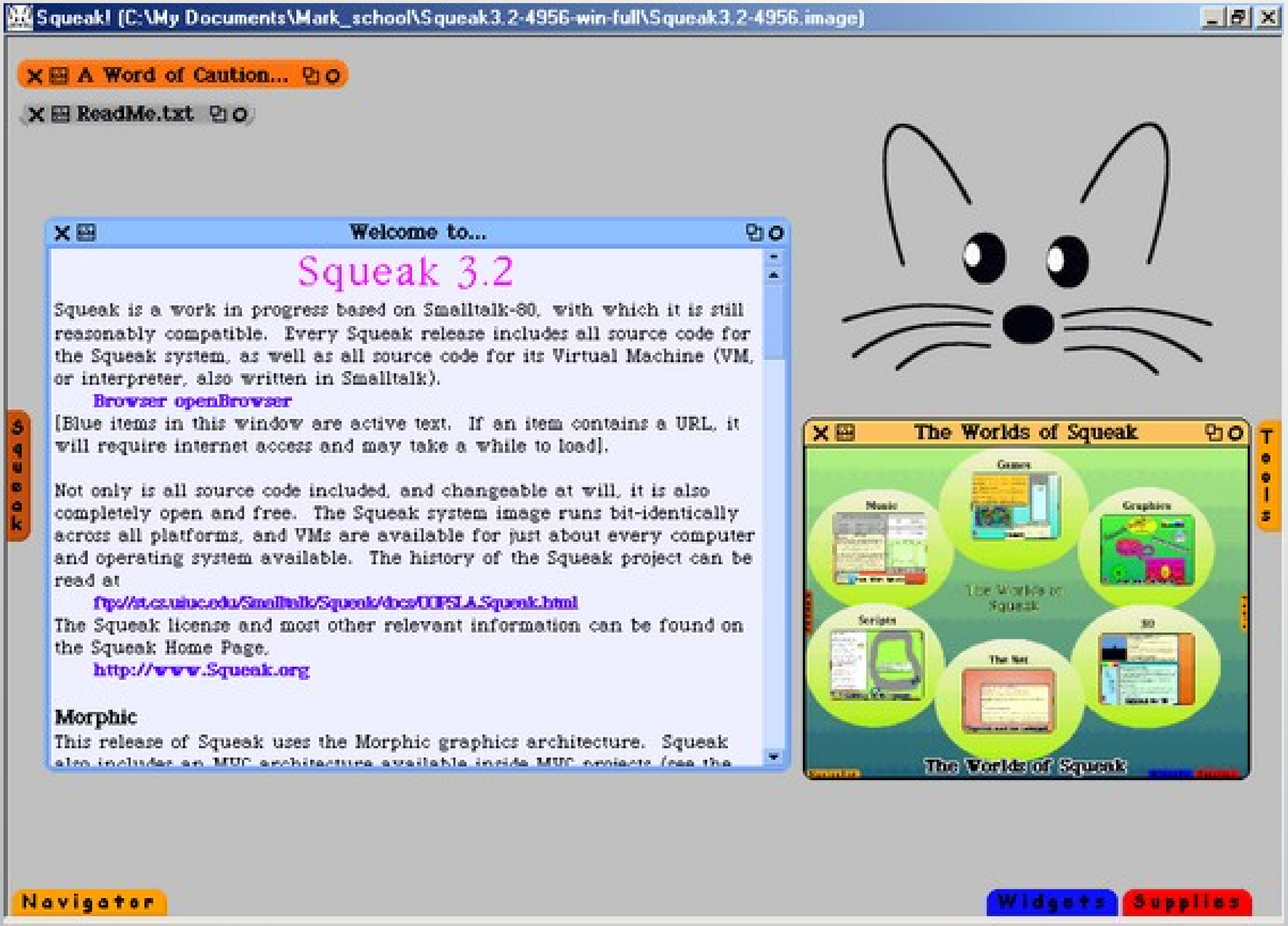


Free Implementations

- Squeak (Smalltalk for Win , Mac, etc.)
  – http://www.squeak.org/download/index.html

- GNU Smalltalk (Unix systems only)
  – http://www.gnu.org/software/smalltalk/smalltalk.html

Squeak Desktop



Squeak 3.2

Squeak is a work in progress based on Smalltalk-80, with which it is still reasonably compatible. Every Squeak release includes all source code for the Squeak system, as well as all source code for its Virtual Machine (VM, or interpreter, also written in Smalltalk).

Browser openBrowser

[Blue items in this window are active text. If an item contains a URL, it will require internet access and may take a while to load].

Not only is all source code included, and changeable at will, it is also completely open and free. The Squeak system image runs bit-identically across all platforms, and VMs are available for just about every computer and operating system available. The history of the Squeak project can be read at

ftp://st.cs.uiuc.edu/Smalltalk/Squeak/docs/OOPSLA.Squeak.html

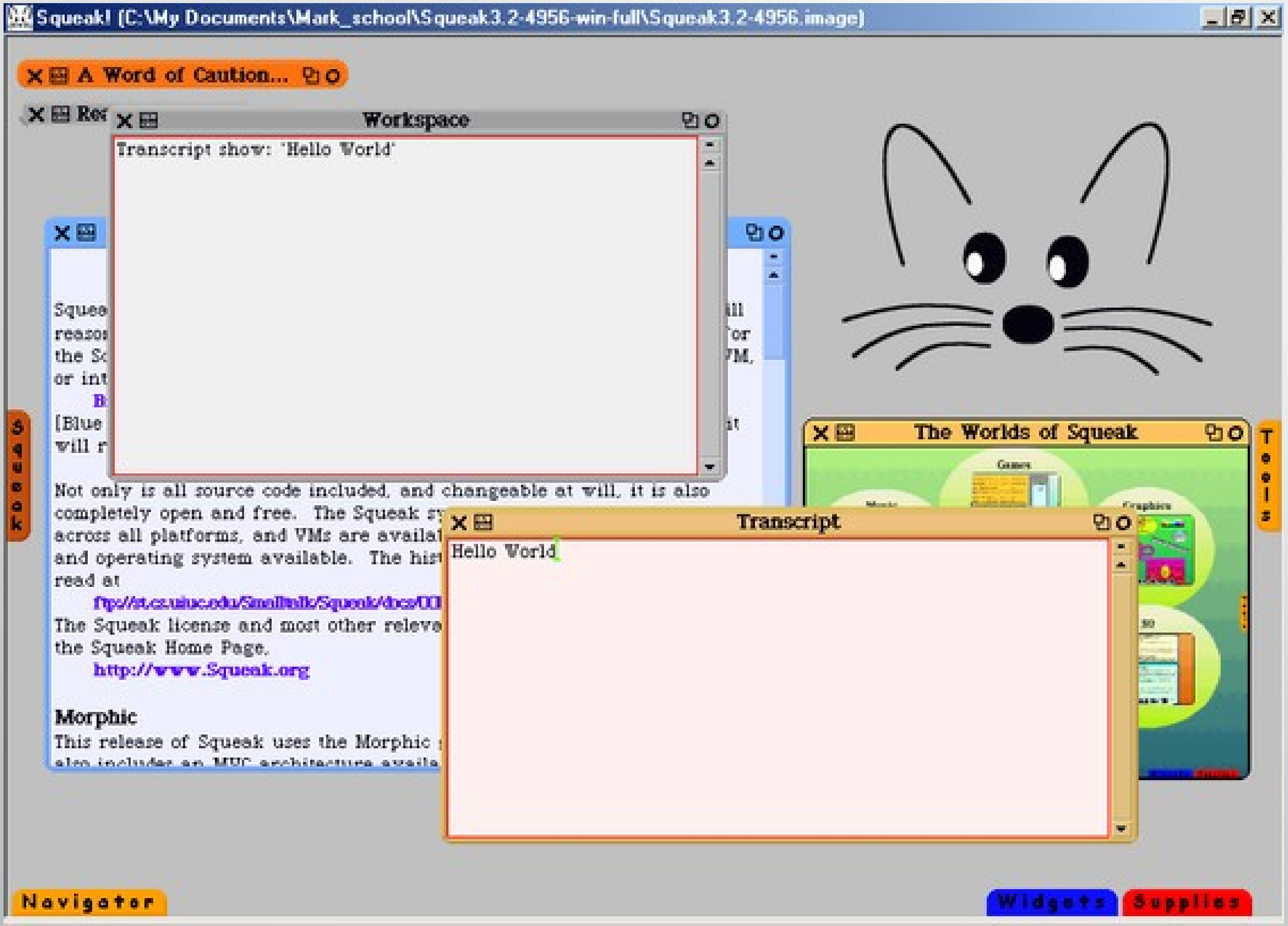
The Squeak license and most other relevant information can be found on the Squeak Home Page.

http://www.Squeak.org

Morphic

This release of Squeak uses the Morphic graphics architecture. Squeak also includes an MVC architecture available inside MVC projects (see the

Hello World Example



Workspace:

```
Transcript show: 'Hello World'
```

Transcript:

```
Hello World
```

Not only is all source code included, and changeable at will, it is also
completely open and free. The Squeak sy
across all platforms, and VMs are availal
and operating system available. The hist
read at
ftp://st.cs.uiuc.edu/Smalltalk/Squeak/docs/OO
The Squeak license and most other releva
the Squeak Home Page.
http://www.Squeak.org

Morphic
This release of Squeak uses the Morphic

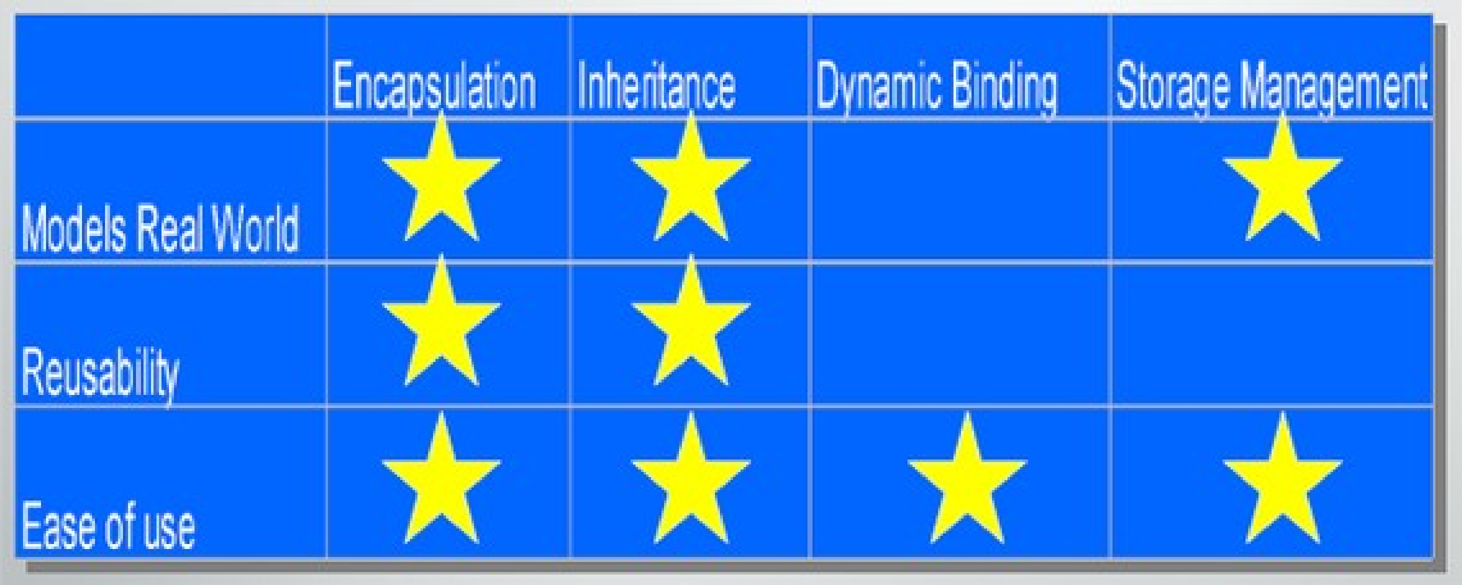
Squeak → Pros and Cons

- Pros
  - Developed by the inventors of Smalltalk
  - Open source
  - Fun
  - Multimedia, 3D, Sounds,….
- Cons
  - You may think that Smalltalk is not serious
  - Code is sometimes not so good

Costs and benefits of “true OO”

- Why is property of Ingalls test useful?
  - Everything is an object
  - All objects are accessed only through interface
  - Makes programs extensible
- What is implementation cost?
  - Every integer operation involves method call
    - Unless optimizing compiler can recognize many cases
  - Is this worth it?
    - One application where it seems useful ?
    - One application where it seems too costly?
    - Are there other issues? Security? (wait for Java final classes…)

Advantages



	Encapsulation	Inheritance	Dynamic Binding	Storage Management
Models Real World	★	★		★
Reusability	★	★		
Ease of use	★	★	★	★

Smalltalk Summary

- **Class**
  - creates objects that share methods
  - pointers to template, dictionary, parent class
- **Objects**: created by a class, contains instance variables
- **Encapsulation**
  - methods public, instance variables hidden
- **Sub typing:** implicit, no static type system
- **Inheritance**: subclasses, self, super

  Single inheritance in Smalltalk-76, Smalltalk-80

To Learn More About Smalltalk

- Lectures on Smalltalk
  - www.iam.unibe.ch/~ducasse/
- Lectures of Ralph Johnson
  - www.cs.uiuc.edu/users/cs497/lectures.html
  - Yes one of the Gang of Four Book (Design Patterns) is a Smalltalk Guru!!
- Lectures of Roger Withney
  - http://www.eli.sdsu.edu/courses/spring01/cs635/index.html

To Learn More About Smalltalk

Local Website	http://www.iam.unibe.ch/~scg/Resources/Smalltalk/
	http://www.iam.unibe.ch/~ducasse/PubHTML/Smalltalk.html
Local Wiki:	http://scgwiki.iam.unibe.ch:8080/SmalltalkWiki/
Cincom Smalltalk	http://www.cincom.com/smalltalk/
Squeak	http://www.squeak.org/
	http://www.cc.gatech.edu/squeak.1
Dolphin Smalltalk	http://www.object-arts.com/Home.htm
	http://www.smalltalk.org
	http://www.goodstart.com/index.shtml
	http://st-www.cs.uiuc.edu/
VisualWorks Wiki	http://brain.cs.uiuc.edu/VisualWorks/
VisualAge Wiki:	http://brain.cs.uiuc.edu/VisualAge/
Newsgroup:	comp.lang.smalltalk
ESUG	http://www.esug.org/
BSUG	http://www.bsug.org/
GSUG	http://www.gsug.org/
SSUG	http://www.iam.unibe.ch/~ssug/

To Learn More About Smalltalk

Smalltalk Books:

- Smalltalk by Example, Alec Sharp
  - Pdf at: http://www.iam.unibe.ch/~ducasse/WebPages/FreeBooks.html
- Best Smalltalk Pratices, K. Beck
- Smalltalk Pattern Design Companion, S.Alpert, K.Brown and B.Woolf
- Squeak, X. Briffault, S. Ducasse (fr)

To Learn More About Smalltalk

Books for Starting:

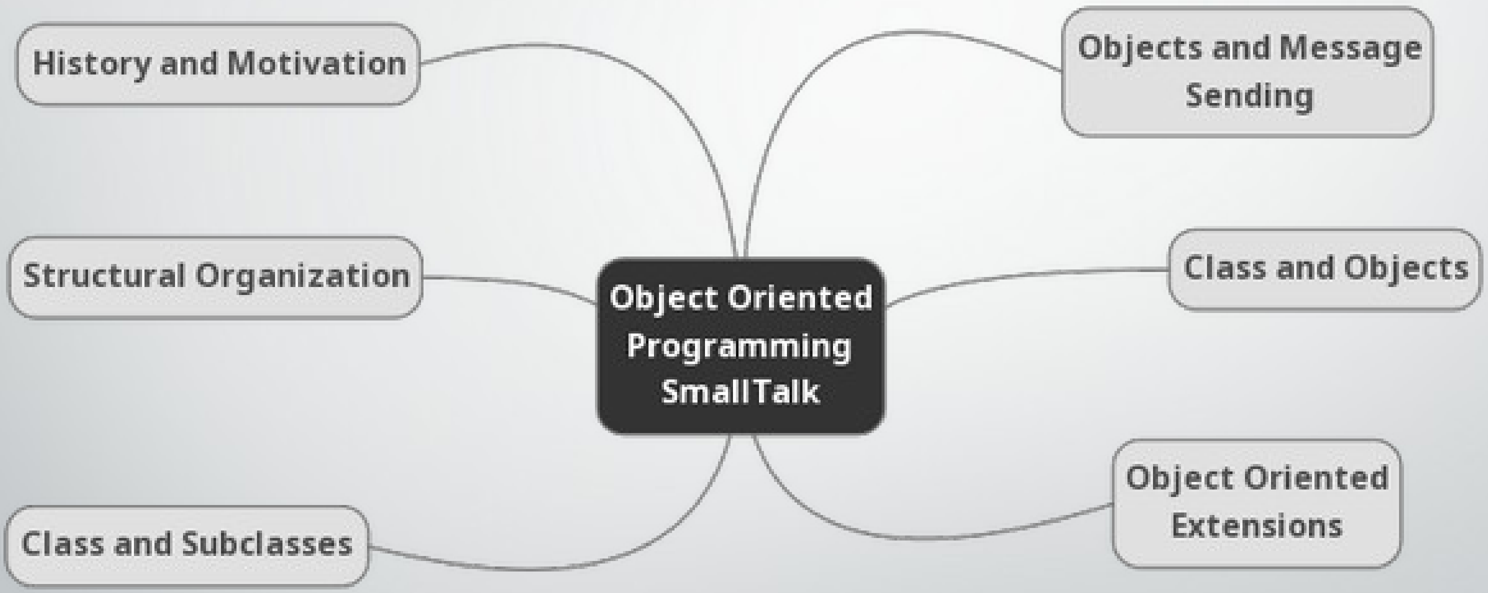
- On To Smalltalk, P. Winston, Addison-Wesley, 1998, 0-201-49827-8
- Smalltalk by Example : The Developer's Guide, A. Sharp, McGraw Hill, ISBN: 0079130364, 1997
- http://www.iam.unibe.ch/~ducasse/FreeBooks.html
- Smalltalk: an Introduction to application development using VisualWorks, T. Hopkins and B. Horan, Prentice-Hall,1995, 0-13-318387-4
- Joy of Smalltalk, Ivan Tomek
- http://brain.cs.uiuc.edu/VisualWorks/Joy+of+Smalltalk
- Chamond, Liu, Smalltalk, Objects, and Design, iUniverse.com, ISBN: 1583484906, 2000.

To Learn More About Smalltalk

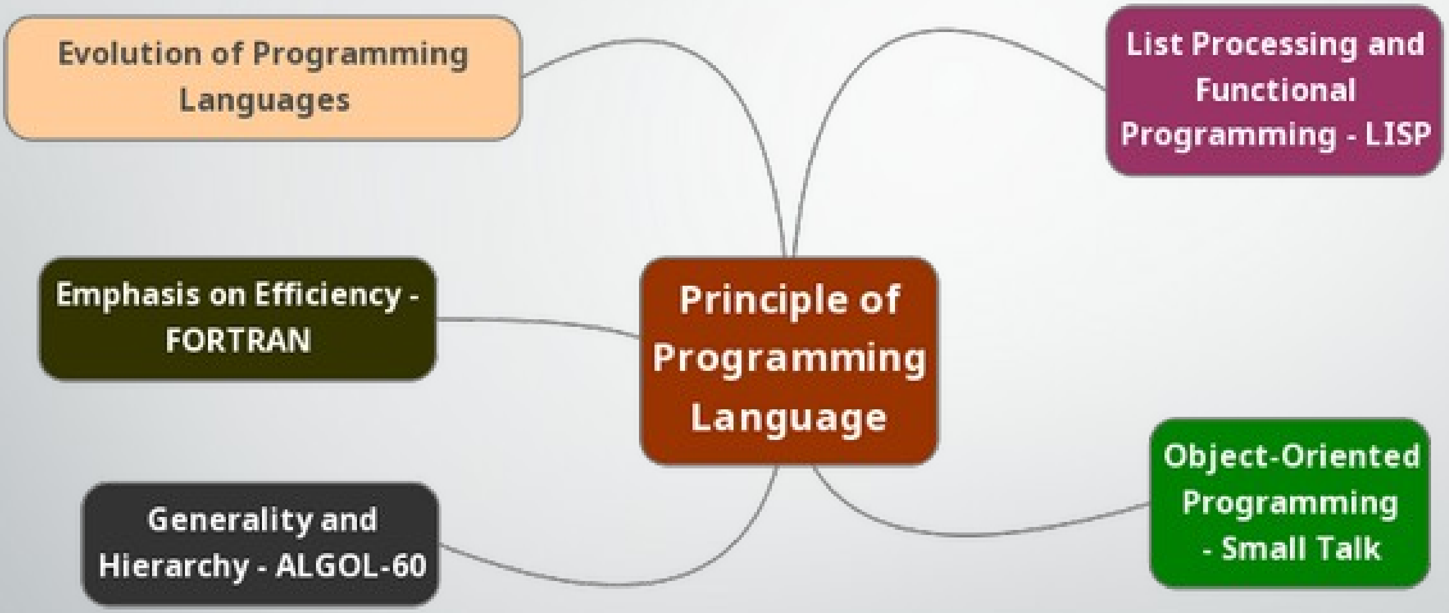
Advanced References

- Smalltalk Best Practice Patterns, K. Beck, Prentice Hall, 1997, ISBN 0-13-476904-x
- The Design Patterns Smalltalk Companion, S. Alpert and K. Brown and B. Woolf, Addison-Wesley, 1998,0-201-18462-1
- Smalltalk with Style, S. Skublics and E. Klimas and D. Thomas, Prentice-Hall, 1996, 0-13-165549-3.
- The Smalltalk Developer's Guide to VisualWorks, T. Howard, Sigs Books, 1995, 1-884842-11-9
- Mastering Envy/Developer, A. Knight, J. Pelrine, and A Chou., SIG Press.

Object Oriented Programming
Small Talk


Object Oriented Programming SmallTalk
History and Motivation
Structural Organization
Class and Subclasses
Objects and Message Sending
Class and Objects
Object Oriented Extensions

Principle of Programming Language


Evolution of Programming Languages
List Processing and Functional Programming - LISP
Emphasis on Efficiency - FORTRAN
Principle of Programming Language
Object-Oriented Programming - Small Talk
Generality and Hierarchy - ALGOL-60