UNIVERSITÄT
BERN

Programmierung 2
Object-Oriented Programming with Java

1. Introduction

Prof. O. Nierstrasz

Spring Semester 2011

P2 — Object-Oriented Programming

Lecturer:	Oscar Nierstrasz
Assistants:	Niko Schwartz, Aaron Karper, Dominique Rahm
WWW:	scg.unibe.ch/teaching/p2

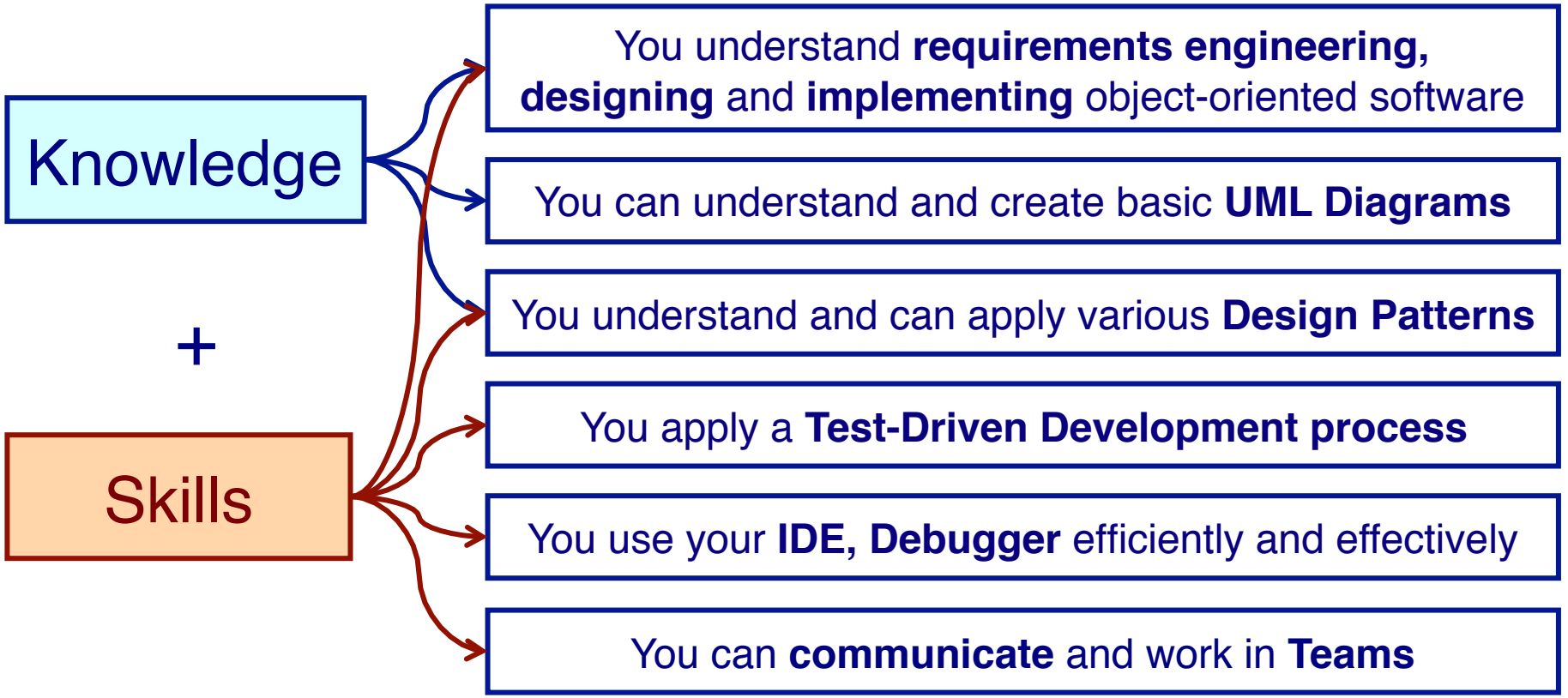
Roadmap

- Goals, Schedule
- What is programming all about?
- What is Object-Oriented programming?
- Foundations of OOP
- Why Java?
- Programming tools, version control

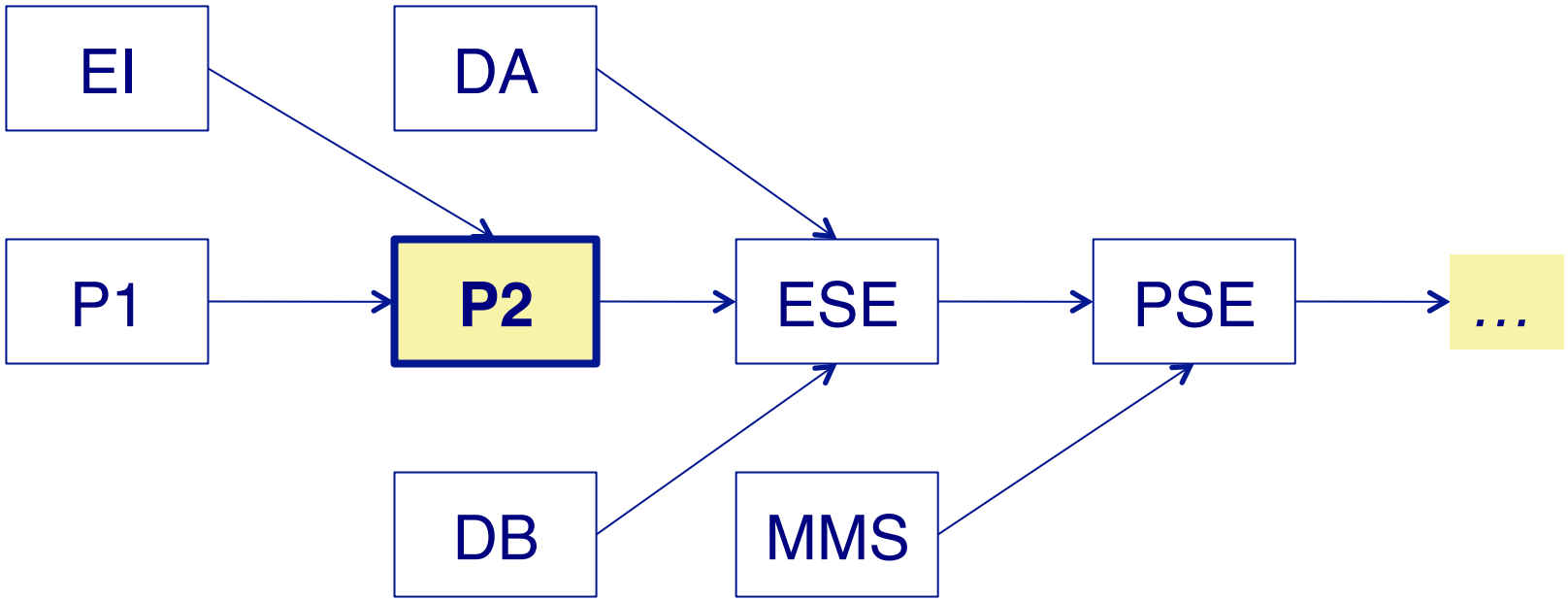
Roadmap

- **Goals, Schedule**
- What is programming all about?
- What is Object-Oriented programming?
- Foundations of OOP
- Why Java?
- Programming tools, version control

Your Learning Targets

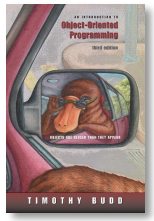


The Big Picture



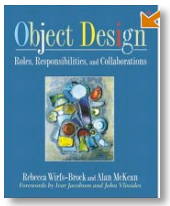
Recommended Texts

- ***Java in Nutshell: 5th edition***, David Flanagan, O'Reilly, 2005.

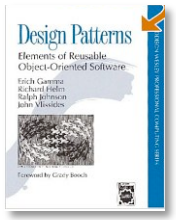


- ***An Introduction to Object-Oriented Programming***, Timothy Budd, Addison-Wesley, 2004.

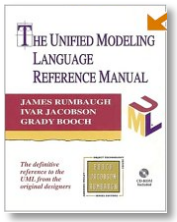
- ***Object-Oriented Software Construction***, Bertrand Meyer,Prentice Hall, 1997.



- ***Object Design - Roles, Responsibilities and Collaborations***, Rebecca Wirfs-Brock, Alan McKean, Addison-Wesley, 2003.



- ***Design Patterns: Elements of Reusable Object-Oriented Software***, Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides, Addison Wesley, Reading, Mass., 1995.



- ***The Unified Modeling Language Reference Manual***, James Rumbaugh, Ivar Jacobson, Grady Booch, Addison-Wesley, 1999

Schedule

1. Introduction
2. Object-Oriented Design Principles
3. Design by Contract
4. A Testing Framework
5. Iterative Development
6. Debugging and Tools
7. Inheritance and Refactoring
8. Advanced OO Design (lab)
9. GUI Construction
10. Guidelines, Idioms and Patterns
11. A bit of C++
12. A bit of Smalltalk
13. Guest Lecture — *Einblicke in die Praxis*
14. *Final Exam*

Roadmap

- Goals, Schedule
- **What is programming all about?**
- What is Object-Oriented programming?
- Foundations of OOP
- Why Java?
- Programming tools, version control

What is the hardest part of programming?

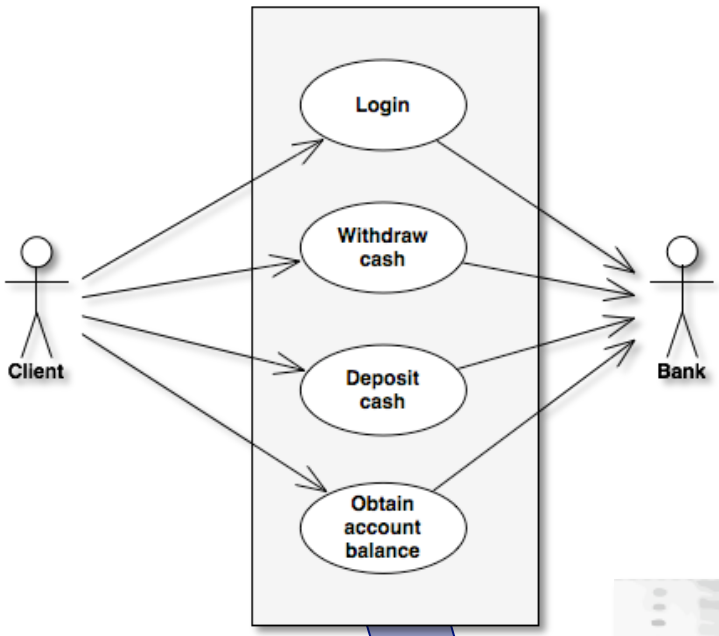
What constitutes programming?

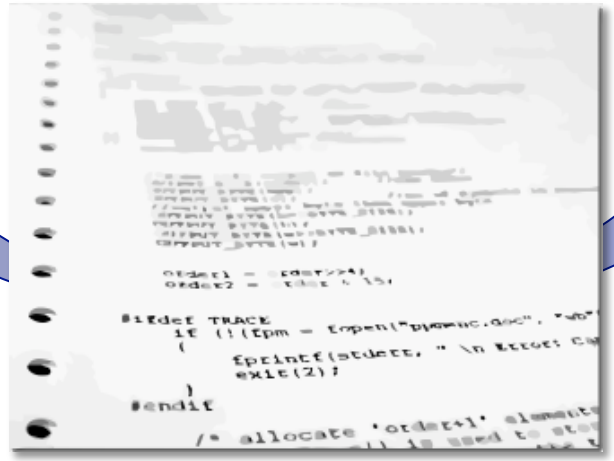
- Understanding requirements
- Design
- Testing
- Debugging
- Developing data structures and algorithms
- User interface design
- Profiling and optimization
- Reading code
- Enforcing coding standards
- ...

Roadmap

- Goals, Schedule
- What is programming all about?
- **What is Object-Oriented programming?**
- Foundations of OOP
- Why Java?
- Programming tools, version control

Programming is modeling

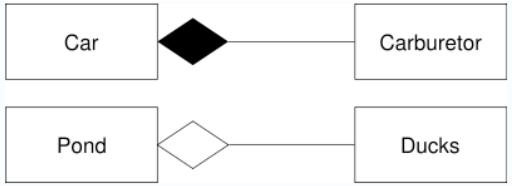




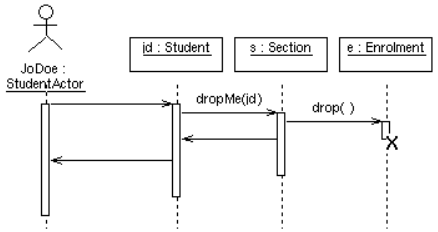
What is Object-Oriented Programming?

Encapsulation — Abstraction & Information Hiding

Composition — Nested Objects

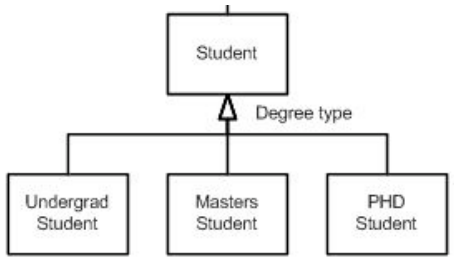


Distribution of Responsibility — *Separation of concerns (e.g., HTML, CSS)*



Message Passing — *Delegating responsibility*

Inheritance — *Conceptual hierarchy, polymorphism and reuse*

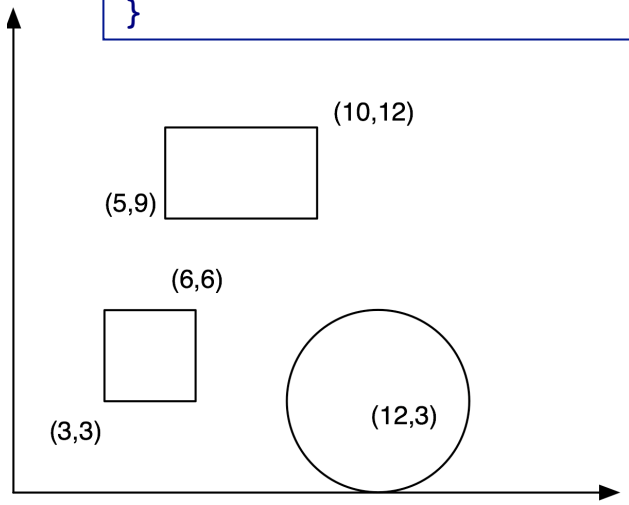


Procedural versus OO designs

Problem: compute the total area of a set of geometric shapes

```
public static void main(String[] args) {
    Picture myPicture = new Picture();
    myPicture.add(new Square(3,3,3));          // (x,y,width)
    myPicture.add(new Rectangle(5,9,5,3));     // (x,y,width,height)
    myPicture.add(new Circle(12,3,3));         // (x,y,radius)

    System.out.println("My picture has size " + myPicture.size());
}
```



How to compute the size?

Procedural approach: *centralize* computation

```
double size() {
    double total = 0;
    for (Shape shape : shapes) {
        switch (shape.kind()) {
        case SQUARE:
            Square square = (Square) shape;
            total += square.width * square.width;
            break;
        case RECTANGLE:
            Rectangle rectangle = (Rectangle) shape;
            total += rectangle.width * rectangle.height;
            break;
        case CIRCLE:
            Circle circle = (Circle) shape;
            total += java.lang.Math.PI * circle.radius * circle.radius / 2;
            break;
        }
    }
    return total;
}
```

Object-oriented approach: *distribute* computation

```
double size() {
   double total = 0;
   for (Shape shape : shapes) {
      total += shape.size();
   }
   return total;
}
```

```
public class Square extends Shape {
...
   public double size() {
      return width*width;
   }
}
```

What are the advantages and disadvantages of the two solutions?

Roadmap

- Goals, Schedule
- What is programming all about?
- What is Object-Oriented programming?
- **Foundations of OOP**
- Why Java?
- Programming tools, version control

Object-Oriented Design in a Nutshell

- Identify *minimal* requirements
- Make the requirements *testable*
- Identify objects and their *responsibilities*
- Implement and *test* objects
- Refactor to *simplify* design
- Iterate!

Responsibility-Driven Design

- Objects are responsible to *maintain information and provide services*
- A good design exhibits:
  - *high cohesion* of operations and data within classes
  - *low coupling* between classes and subsystems
- Every method should perform *one, well-defined task:*
  - High level of abstraction — write to an interface, not an implementation

Design by Contract

- Formalize client/server contract as *obligations*
- Class invariant — formalize valid state
- Pre- and post-conditions on all public services
  - *clarifies responsibilities*
  - *simplifies design*
  - *simplifies debugging*

Extreme Programming

Some key practices:

- Simple design
  - *Never anticipate functionality that you "might need later"*
- Test-driven development
  - *Only implement what you test!*
- Refactoring
  - *Aggressively simplify your design as it evolves*
- Pair programming
  - *Improve productivity by programming in pairs*

Testing

- Formalize requirements
- Know when you are done
- Simplify debugging
- Enable changes
- Document usage

Code Smells

- Duplicated code
- Long methods
- Large classes
- Public instance variables
- No comments
- Useless comments
- Unreadable code
- …

Refactoring

> *"Refactoring is the process of **rewriting** a computer program or other material to improve its structure or readability, while explicitly **keeping its meaning** or behavior."*
>
> *— wikipedia.org*

Common refactoring operations:

- Rename methods, variables and classes
- Redistribute responsibilities
- Factor out helper methods
- Push methods up or down the hierarchy
- Extract class
- …

Design Patterns

“a general repeatable solution to a commonly-occurring problem in software design.”

Example

- Adapter — “adapts one interface for a class into one that a client expects.”

Patterns:

- Document “best practice”
- Introduce standard vocabulary
- Ease transition to OO development

But …

- May increase flexibility at the cost of simplicity

Roadmap

- Goals, Schedule
- What is programming all about?
- What is Object-Oriented programming?
- Foundations of OOP
- **Why Java?**
- Programming tools, version control

Why Java?

Special characteristics

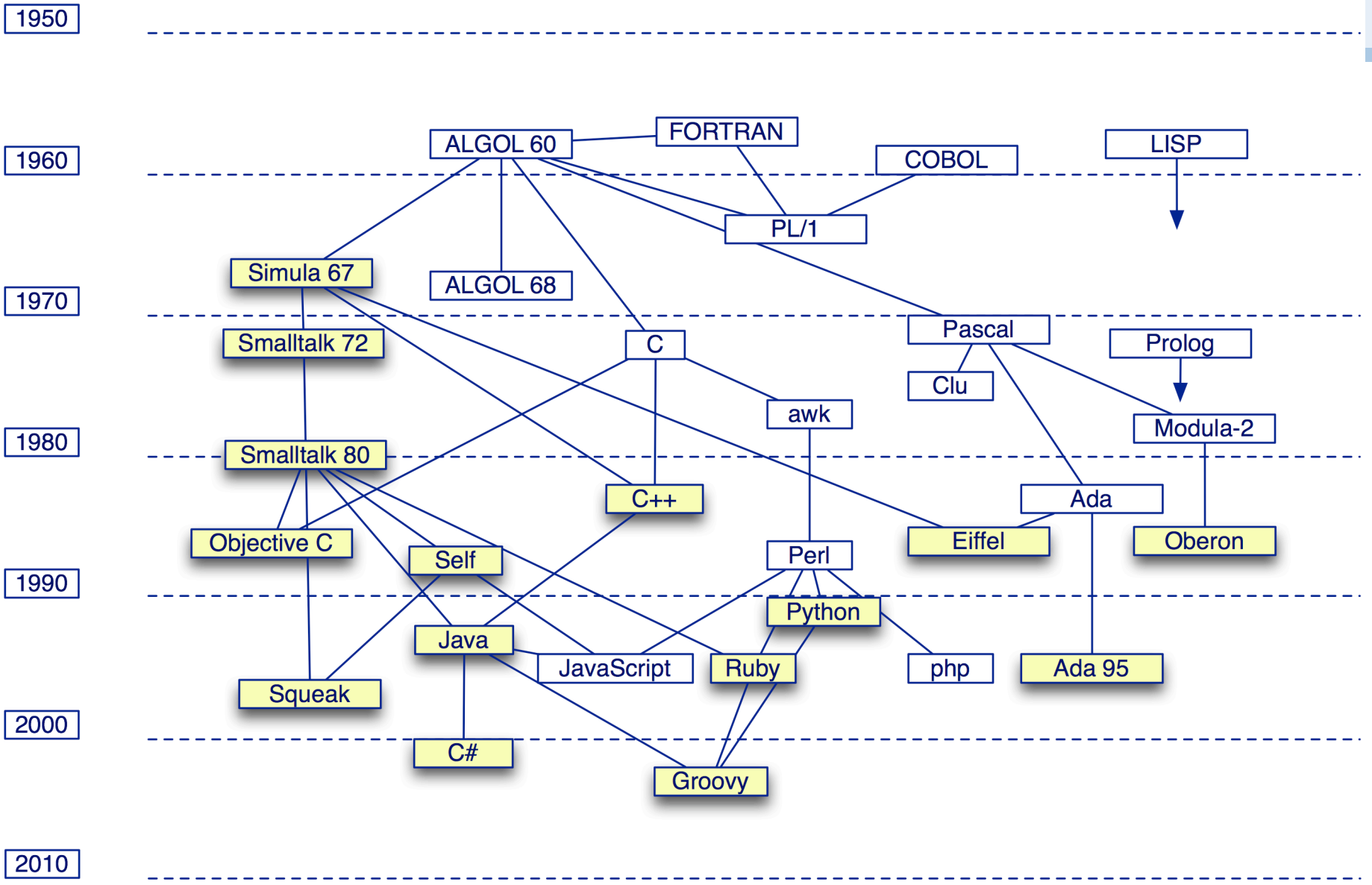
- Resembles C++ minus the complexity
- Clean integration of many features
- Dynamically loaded classes
- Large, standard class library

Simple Object Model

- “Almost everything is an object”
- No pointers
- Garbage collection
- Single inheritance; multiple subtyping
- Static and dynamic type-checking

Few innovations, but reasonably clean, simple and usable.

History


1950
1960
1970
1980
1990
2000
2010
ALGOL 60
FORTRAN
COBOL
LISP
PL/1
Simula 67
ALGOL 68
Smalltalk 72
C
Pascal
Prolog
Clu
awk
Modula-2
Smalltalk 80
C++
Ada
Objective C
Self
Perl
Eiffel
Oberon
Python
Java
JavaScript
Ruby
php
Ada 95
Squeak
C#
Groovy

Roadmap

- Goals, Schedule
- What is programming all about?
- What is Object-Oriented programming?
- Foundations of OOP
- Why Java?
- **Programming tools, version control**

Programming Tools

Know your tools!

- IDEs (Integrated Development Environment)— e.g., Eclipse,
- Version control system — e.g., svn,cvs, rcs
- Build tools — e.g., maven, ant, make
- Testing framework — e.g., Junit
- Debuggers — e.g., jdb
- Profilers — e.g., java -prof, jip
- Document generation — e.g., javadoc

Version Control Systems

A version control system keeps track of multiple file revisions:

- *check-in* and *check-out* of files
- *logging changes* (who, where, when)
- *merge* and *comparison* of versions
- *retrieval* of arbitrary versions
- *"freezing"* of versions as releases
- *reduces storage space* (manages sources files + multiple *"deltas"*)

Version Control

Version control enables you to make radical changes to a software system, with the assurance that ***you can always go back*** to the last working version.

✎ When should you use a version control system?

✔ *Use it whenever you have one available, for even the* ***smallest project!***

Version control is as ***important*** *as* ***testing*** *in iterative development!*

What you should know!

- *What is meant by “separation of concerns”?*
- *Why do real programs change?*
- *How does object-oriented programming support incremental development?*
- *What is a class invariant?*
- *What are coupling and cohesion?*
- *How do tests enable change?*
- *Why are long methods a bad code smell?*

Can you answer these questions?

- *Why does up-front design increase risk?*
- *Why do objects “send messages” instead of “calling methods”?*
- *What are good and bad uses of inheritance?*
- *What does it mean to “violate encapsulation”?*
- *Why is strong coupling bad for system evolution?*
- *How can you transform requirements into tests?*
- *How would you eliminate duplicated code?*
- *When is the right time to refactor your code?*

License

> http://creativecommons.org/licenses/by-sa/2.5/

creative commons

COMMONS DEED

Attribution-ShareAlike 2.5

You are free:

- to copy, distribute, display, and perform the work
- to make derivative works
- to make commercial use of the work

Under the following conditions:

Attribution. You must attribute the work in the manner specified by the author or licensor.

Share Alike. If you alter, transform, or build upon this work, you may distribute the resulting work only under a license identical to this one.

- For any reuse or distribution, you must make clear to others the license terms of this work.
- Any of these conditions can be waived if you get permission from the copyright holder.

Your fair use and other rights are in no way affected by the above.