

1

# Introduction to Programming

Bertrand Meyer

Slides revised: 24 October 2003  
(Not final version)

Chair of Software Engineering      Intro – Lecture 2

2

## German version of lecture slides

Folien für diese und alle Vorlesungseinheiten werden von nun an **auch in Deutsch** verfügbar sein.

Sie können die deutsche Folien auf der [Webseite](#) der Vorlesung finden.

Chair of Software Engineering      Intro – Lecture 2

3

## Announcements

- Some delays with opening accounts (the administration doesn't have all the information yet).
- Initial compilation takes a long time; this will be corrected but please be patient. It's only for the first compilation.
- Congratulations to those who were able to install the Mac version of EiffelStudio. Please note that it is still a beta version, not a released one.

Chair of Software Engineering      Intro – Lecture 2

4

## Announcements

- Two exercise groups switch rooms:
  - lions - Monday ML J37.1, Tuesday ETZ E9
  - cows - Monday ML F39, Tuesday ML J34.3
- Please contact Susanne during the break or after the lecture if
  - Your e-mail address is not ...@student.ethz.ch
  - You did not receive a subscription confirmation for your exercise group
  - You did not receive the fable
- If you are reading this on the Web site, watch for more announcements in class.

Chair of Software Engineering      Intro – Lecture 2

5

## Lecture 2: Dealing with objects

Chair of Software Engineering      Intro – Lecture 2

6

## Your first program!

1. Display a map of Paris with the Metro
2. Spotlight position of Louvre museum
3. Highlight line 8
4. Animate predefined route

Chair of Software Engineering      Intro – Lecture 2

## A class text

7

Software machine: **class**

Extend an existing class: **inherit**

Operations: **feature**

Feature name: **explore is**

Comment: **end**

Feature declaration: **do**

Pseudocode: **do**

```

class
  PREVIEW
inherit
  TOURISM
feature
  explore is
    -- Show city info and route.
    do
      "To be filled in (by you!)"
    end
  end
end

```

Keywords have a special meaning: **class**, **inherit**, **feature**, **is**, **do**, **end**.

Chair of Software Engineering | Intro - Lecture 2

## Magic?

8

- Class **TOURISM** is part of the supporting software
- It helps you learn by using predefined facilities (the "magic")
- Little by little pieces of the magic will be removed
- At the end, the magic will be gone

Chair of Software Engineering | Intro - Lecture 2

## Filling in the feature body

9

```

class
  PREVIEW
inherit
  TOURISM
feature
  explore is
    -- Show city info and route.
    do
      Paris,display
      Louvre,spotlight
      Line8,highlight
      Route1,animate
    end
  end
end

```

Chair of Software Engineering | Intro - Lecture 2

## Program formatting

10

Between adjacent elements:  
**break**: one or more spaces, "tabs", "carriage returns"

All kinds of break are equivalent

Typographical variations (**boldface**, *italics*, colors) do **not** affect meaning (**semantics**) of program

Breaks: class, inherit, feature, explore is, do, end, end

Tabs: Paris,display, Louvre,spotlight, Line8,highlight, Route1,animate

```

class
  PREVIEW
inherit
  TOURISM
feature
  explore is
    -- Show city info
    -- and route.
    do
      Paris,display
      Louvre,spotlight
      Line8,highlight
      Route1,animate
    end
  end
end

```

Chair of Software Engineering | Intro - Lecture 2

## Style rules

11

For indentation, use tabs, not spaces

Use this property to highlight the **structure** of the program, particularly through **indentation**

Chair of Software Engineering | Intro - Lecture 2

## Feature call

12

The fundamental mechanism of program execution: apply a "feature" to an "object"

Basic form: **your\_object . your\_feature**

Object (target of the call): Paris,display

Feature of the call: .display

```

class
  PREVIEW
inherit
  TOURISM
feature
  explore is
    -- Show city info
    -- and route.
    do
      Paris,display
      Louvre,spotlight
      Line8,highlight
      Route1,animate
    end
  end
end

```

Chair of Software Engineering | Intro - Lecture 2



## A distinct mode of expression

13

- *Paris*.display
- next\_message.send
- computer.shut\_down
- telephone.ring



## Predefined objects

14

- Paris, Louvre, Metro, and Route1 are names of predefined objects
- Defined in class TOURISM from which PREVIEW inherits.
- display, spotlight, highlight, and animate are features, applicable to these objects



## More style rules

15

- Class name: all upper-case
- Period in feature call: no space before or after
- Names of predefined objects: start with upper-case letters
- New names (for objects you define) start with lower-case letters

```

class
  → PREVIEW
  inherit
  → TOURISM
  feature
  explore is
    do
      -- Show city info
      -- and route.
    → Paris.display
    → Louvre,spotlight
    → Line8,highlight
    → Route1,animate
  end
end

```



## Object technology

16

- We work with objects
- Our style of programming: **Object-Oriented programming**
- Abbreviation: **O-O**
- More generally, "Object Technology": includes O-O databases, O-O analysis, O-O design...
- Software execution is made of operations on objects — feature calls

*your\_object*.*your\_feature*



## A distinct mode of expression

17

- *Paris*.display
- next\_message.send
- computer.shut\_down
- telephone.ring

Every operation applies to an object  
(the target of the call)



## How many...

18

... does it take to screw in a light bulb?



## How many...

19

... **object-oriented programmers**  
does it take to screw in a light bulb?



## What's an object?

20

It's a software notion: a machine which we know through the operations applicable to it.

Three kinds of object:

1. Some reflect **material** objects of the outside world: the Louvre, Paris, a metro car..
2. Some correspond to **abstract** notions from the outside world: a line, a route...
3. Some express purely **software** notions ("data structures")

A key attraction of object technology is its **modeling** power: connect software objects to objects of the problem domains  
You should not, however, confuse them  
In this course, "object" by default means **software** object



## Features, commands and queries

21

**Feature**: an operation available on a certain class of objects

Three kinds:

- **Command**
- **Query**
- **Creation procedure** (seen later)



## Queries

22

Goal: obtain **properties** of objects

*Should not modify* the object, or any other

Examples, for "route" objects:

- What is the origin (first station) of **Route1**?
- What is the end point of **Route1**?
- How many steps does **Route1** use?
- Which stations does **Route1** traverse?



## Commands

23

Goal: produce a **change** on an object, or several  
Examples, for "route" objects:

- Animate **Route1**
- Prepend (add at the beginning) a segment to **Route1**
- Append (add at the end) a segment to **Route1**.



## A command

24



## A query

25

Chair of Software Engineering

Intro – Lecture 2

## An object is a machine

26

Programs are machines  
They're made of smaller machines: **objects**

- During execution there may be many objects (e.g. millions)

Chair of Software Engineering

Intro – Lecture 2

## An object is a machine

27

- A machine, hardware or software, is characterized by the operations (“features”) users may apply

Chair of Software Engineering

Intro – Lecture 2

## Two views of objects

28

- An object has data, stored in memory.

“Bürkliplatz”
25
5
“Bucheggplatz”

- An object is a machine offering queries and commands.

The connection:

- The operations that the machine provides (1) access and modify the object's data (2).

Chair of Software Engineering

Intro – Lecture 2

## Objects: a definition

29

An **object** is a software machine allowing programs to access and modify a collection of data.

Chair of Software Engineering

Intro – Lecture 2

## Defining and classifying features

30

A **feature** is an operation that programs may apply to certain classes of objects.

- A feature that *accesses* an object is a **query**
- A feature that may *modify* an object is a **command**

Chair of Software Engineering

Intro – Lecture 2

## Using queries

31

- Queries are as important as commands
- Queries don't "do" anything, but yield a value, e.g. `Route1.origin` yields the starting point of `Route1`
- You may work with the return values of queries, e.g. display the starting point on the screen.

## Feature calls with arguments

32

- Task:
  - Show starting point of `Route1` on "console" window
- You need:
  - Predefined object `Console`.
  - Feature `show` applicable to `Console`.
  - The object `Route1`
  - Feature `origin` returning starting point and applicable to `Route1`
- The new feature call:
  - `Console.show (Route1.origin)`

## Extending the feature body

33

```
class PREVIEW
  inherit TOUR
  feature
    explore is
      -- Show city info, route, and the route's origin.
      do
        Paris,display
        Louvre,spotlight
        Line8,highlight
        Route1,animate
        Console.show (Route1 . origin)
      end
    end
end
```

## Features with arguments

34

`your_object.your_feature (some_argument)`

`some_argument` is a value that `your_feature` needs

Example: feature `show` must know what to show.

Same concept as function arguments in maths:

`cos (x)`

Features may have several arguments:

`x.f (a, b, c, d) -- Separated by commas`

In well written O-O software, most have 0 or 1 argument

## A distinct mode of expression

35

- `Paris.display`
- `next_message.send`
- `computer.shut_down`
- `telephone.ring`

Every operation applies to an object

## A distinct mode of expression

36

- `Paris.display`
- `next_message.send_to (recipient)`
- `computer.shut_down_after (3)`
- `telephone.ring_several (10, Loud)`

Every operation applies to an object and may take arguments

## Object technology

37

- Source: Simula 67 language, Oslo, mid-sixties
- Spread *very* slowly in the seventies
- Smalltalk, developed at Xerox PARC in late seventies, made O-O hip by combining it with visual technologies
- First OOPSLA conference in 1986 revealed O-O to the unwashed masses
- Spread quickly in 1990s through O-O languages like Objective C, C++, Eiffel, Java, as well as O-O tools, O-O databases, O-O analysis...
- Largely accepted today
- *Non* O-O approaches are also called "procedural".

## Eiffel

38

- Dates back to 1985 in first version
- Constantly refined and improved since then
- Fully object-oriented; not a hybrid with other approaches
- Focuses on quality, especially reliability, extendibility and reusability
- Emphasizing **simplicity**
- Used for many mission-critical projects in industry (see next)
- Based on concepts of "Design by Contract".
- Implementations: from Eiffel Software, Object Tools, University of Nancy ("SmartEiffel")
- International standard in preparation through ECMA

## Large Eiffel projects in industry

39



## Why use Eiffel?

40

- Simple, clean O-O model
- Enables you to focus on concepts, not language
- Little language "baggage"
- Development environment (EiffelStudio)
- Portability: Windows / Linux & others

## Scaling up

41

- One of the toughest issues in learning software is to find solutions that work well both "in the small" and "in the large".
- That's the goal for the techniques we teach in this course.

## An object has an interface

42



## An object has an implementation

43



Chair of Software Engineering

Intro - Lecture 2



## Information hiding

44



Chair of Software Engineering

Intro - Lecture 2



## To do next!

45

- Read chapter 3
- Read slides for next lecture
- For next week: read chapter 4

Chair of Software Engineering

Intro - Lecture 2



46

End lecture 2

Chair of Software Engineering

Intro - Lecture 2

