Einführung in die Programmierung
Introduction to Programming

Prof. Dr. Bertrand Meyer

Lecture 2: Dealing with Objects I
Our first program!

Display a map of Paris

Spotlight position of Louvre museum

Highlight line 8 of the metro

Animate a predefined route
A class text

/!

Class: a software machine

The class name

class  

PREVIEW

inherit

TOURISM

feature

explore

-- Show city info and route.

do

-- “To be filled in (by you!)”

end

end
Traffic library convention

Traffic classes have names of the form

```
TRAFFIC ACTUAL_CLASS_NAME
```

In these slides and in the book, for brevity, I omit the `TRAFFIC` part of the name

But you will need it to find the classes in the software
Another convention

For long names, use underscores “_”

TRAFFIC_STATION
Station_Paradeplatz  --  or: Station_Parade_Platz

We do not use “CamelCase”:

AShortButHardToDeCipherName

but underscores (sometimes called “Pascal_case”):

A_significantly_longer_but_still_perfectly_clear_name
A class text

Keywords have a special role: class, inherit, feature, do, end.

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

Software machine

Extend existing class

Feature declaration

Comment

Pseudocode

Operations

Feature name
Class `TOURISM` is part of the supporting software.

It helps you learn by using predefined facilities ("magic")

Little by little pieces of the magic will be removed.

At the end, the magic will be gone.
Filling in the feature body

class
    PREVIEW
inherit
    TOURISM
feature
    explore

    -- Show city info and route.
    do
    Paris.display
    Louvre.spotlight
    Line8.highlight
    Route1.animate
end
end
Program formatting

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 the effect (semantics) of programs
Style rule

For indentation, use tabs, not spaces

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

The fundamental mechanism of program execution: apply a “feature” to an “object”
Basic form: \textit{your\_object}\texttt{.your\_feature}
Predefined objects

Paris, Louvre, Line8, and Route1 are names of predefined objects

The objects are defined in class TOURISM from which PREVIEW inherits

display, spotlight, highlight and animate are features, applicable to these objects
More style rules

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

```java
class PREVIEW
{
    inherit TOURISM
{
        feature explore
{
            do
{
                Paris.display
{
                    Louvre.spotlight
{
                        Line8.highlight
{
                            Route1.animate
{
                                end
{
                                    end
{
                                        -- Show city info
                                        -- and route.
                                }
{
                            }
{
                        }
{
                    }
{
                }
{
            }
{
        }
{
    }
{
}
```
Object technology

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

*Paris.display*

*next_message.send*

*computer.shut_down*

*telephone.ring*

Every operation applies to an object
(the target of the call)
What’s an object?

Software notion: machine known through operations applicable to it.

Three kinds of object:

- “Physical objects”: reflect material objects of the modeled world or system
  
  Examples: the Louvre, Paris, a metro car..

- “Abstract objects”: describe abstract notions from the modeled world or system
  
  Examples: a line, a route...

- “Software objects”: represent pure software notions
  
  Examples: “data structures” such as arrays or lists

A key attraction of object technology is its *modeling* power: connect software objects to objects of the problem domain (“model”)

You should not, however, confuse them

In this course, “object” by default means *software object*
Two views of objects

1. An object has data, stored in memory.
2. An object is a machine offering operations (features)

The connection:
- The operations that the machine provides (2) access and modify the object’s data (1).
Features: commands and queries

Feature: an operation available on a certain class of objects

Three kinds:
- Command
- Query
- Creation procedure (seen later)
Queries

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 stations does Route1 have?
- Which stations does Route1 traverse?
Commands

Goal: produce a change on an object, or several objects

Examples, for “route” objects:

- Animate Route1
- Append (add at the end) a station to Route1
- Prepend (add at the beginning) a station to Route1
A command
A query
The command-query separation principle

Asking a question should not change the answer
An object is a machine

An executing program is a machine
It’s made of smaller machines: *objects*

During execution there may be many objects (e.g. millions)
An object is a machine

A machine, hardware or software, is characterized by the operations (“features”) users may apply.
Two views of objects

Two viewpoints:

1. An object has data, stored in memory.
2. An object is a machine offering operations (features: commands and queries)

The connection:

The operations that the machine provides (2) access and modify the object’s data (1).
Objects: a definition

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

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
Using queries

Queries are as important as commands

Queries don’t “do” anything, but yield a value, e.g. `Route1.origin` yields the starting station of `Route1`

You may work with the return values of queries, e.g. highlight the starting station on the screen
Features may have **arguments**

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

class PREVIEW
  inherit TOURISM

feature explore
  -- Show city info, a route, and route's origin.
  do
    Paris.display
    Louvre.spotlight
    Line8.highlight
    Route1.animate
    Console.show(Route1.origin)
  end
end
Features with arguments

`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:

\[ x.f(a, b, c, d) -- \text{Separated by commas} \]

In well written O-O software, most have 0 or 1 argument
A distinct mode of expression

Paris.display

next_message.send

computer.shut_down

telephone.ring

Every operation applies to an object
A distinct mode of expression

*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
Scaling up

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**
An object has an **implementation**
Information hiding

- animate
- append
- prepend
- first
- last
- count
- stations
The information hiding principle

The designer of every module must specify which properties are accessible to clients (public) and which are internal (secret).

The programming language must ensure that clients can only use public properties.
What we have seen so far

Basic concepts and constructs of object technology:

- Classes (a first view)
- Basic program text structure
- Objects
- Features
- Feature call
- Feature arguments

Methodological principles:

- Command-query separation
- Information hiding