# C Object-oriented Programming

## C.1 Overview

- Motivation for the OO paradigm
- Software-design methods
- Basic terms of OO programming
- The Evolution of the object model
- Fundamental concepts of the OO paradigm
- Objetct-oriented Analysis and Design






## C.2 References

**ABC83.** M. P. Atkinson, P. J. Bailey, K. J. Chisholm, W. P. Cockshot and R. Morrison, “An Approach to Persistent Programming”, *The Computer Journal*, Vol. 26, No. 4, pp. 360-365, 1983.

**Boo94.** Grady Booch, *Object-Oriented Analysis and Design (with Applications)*, Benjamin/ Cummings, Redwood (CA), 1994.

**CoY91a.** P. Coad, E. Yourdon. Object-Oriented Analysis. Prentice Hall, 1991.

**Coa91b.** P. Coad, E. Yourdon. Object-Oriented Design. Prentice Hall, 1991.

**CW85.** Luca Cardelli, Peter Wegner, "On Understanding Types, Data Abstraction, and Polymorphism", *Computing Surveys*, Vol. 17, No. 4, Dec. 1985.

**GHJ+97.** Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*, 10th print, Addison-Wesley, 1997

**Jac92.** I. Jacobson. Object-Oriented Software Enineering — A Use Case Driven Approach. Addison-Wesley, 1992.

**MaM88.** Ole Lehrmann Madsen, Birger Møller-Pedersen, "What object-oriented programming may be — an what it does not have to be", *ECOOP '88 – European Conference on OO Programming*, pp. 1 - 20, S. Gjessing, K. Nygaard [Eds.]; Springer Verlag, Oslo, Norway, Aug. 1988.






## C.2 References (2)

**Mey86.** Bertrand Meyer, "Genericity versus Inheritance", *Conference on Object-Oriented Programming Systems, Languages, and Applications - OOPSLA '87*, pp. 391 - 405, Portland (Oreg., USA), published as *SIGPLAN Notices*, Vol. 21, No. 11, Nov. 1986.

**Mey88.** Bertrand Meyer. *Object Oriented Software Construction.* Prentice Hall Inc., Hemel Hempstead, Hertfordshire, 1988.

**Oes97.** B. Oestereich. Objektorientierte Softwareentwicklung: Analyse und Design. Oldenbourg, 1997.

**Rum91.** J. Rumbaugh. Object-Oriented Modelling and Design. Prentice Hall, 1991.

**Str91.** Bjarne Stroustrup. *The C++ programming language*, 2. ed., Addison-Wesley, 1991.

**Str93.** Bjarne Stroustrup, "A History of C++", ACM SIGPLAN Notices, Vol. 28, No. 3, pp.271 - 297, Mar. 1993.

**Weg87.** Peter Wegner, “Dimensions of Object-Based Language Design”, *OOPSLA '87 – Conference Proceedings,* pp. 1-6, San Diego (CA, USA), published as *SIGPLAN Notices*, Vol. 22, No. 12, Dec. 1987.

**Weg90.** Peter Wegner, “Concepts and Paradigms of Object-Oriented Programming”, *ACM OOPS Messenger*, No. 1, pp. 8-84, Jul. 1990.






## C.2 References (3)

**BRJ98.** G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language Reference Manual.* Addison Wesley, 1998.

**RJB98.** J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language User Guide.* Addison Wesley, 1998.

**JBR98.** I. Jacobson, G. Booch, and J. Rumbaugh. *The Unified Software Development Process.* Addison Wesley, 1998.

# C.3 Motivation for the OO Paradigm

## 1 Goals

- Increasing complexity of large software
  - "industrial-strength" software [Boo94]
    - ➤ impossible for one developer to comprehend all details of its design
    - ➤ very long life span
    - ➤ many users depend on their proper functioning
    - ➤ many people responsible for maintenance and enhancement
- → Software Crisis
  - Hardware increasingly capable
  - Software becomes larger and larger
  - Costs for maintenance and enhancement rise dramatically
  - Not enough good software developers to create the software users need






# C.4 Software-Design Methods

## 1 Classification [Boo94]

- Top-down structured design (composite design)
- Object-oriented design

## 2 Classes of Programming Languages

… at least the most important ones

- Procedural / imperative
- Functional
- Object-oriented






## 1 Goals (2)

- Increase the productivity of programmers
  - Design patterns for repeatedly occurring problems
  - Reusage of existing software
  - Better extensibility of software by modularization and clear interfaces
  - Incremental development from small & simple to huge & complex systems
  - Better control over complexity and costs of software maintenance
- Shift from the needs of the machine to abstractions of the problem domain
  - Better understanding of the problem
  - Terminology of the problem domain is reflected in the software solution
    - ➤ better understanding of the solution



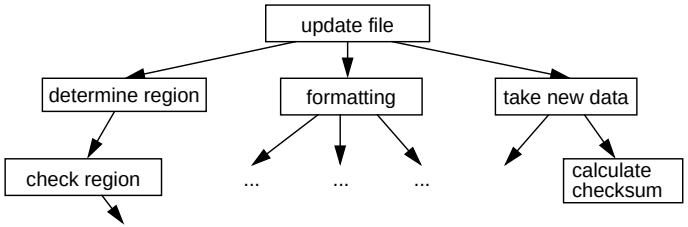



## 3 Top-Down Structured Design (Composite Design)

- Units of decomposition: Subroutine
- **Algorithmic decomposition**
- Not suitable for structuring today's large and complex software systems
- Top-down structured design cannot describe:
  - – data abstraction & information hiding
  - – concurrency
- Problems arise when applications are very complex or when object-oriented languages have to be used
- Widely used technique
- Procedural languages ideally suited for implementations

## 3 Top-Down Structured Design (2) (Composite Design)

- Example:









## 4 Object-oriented Design

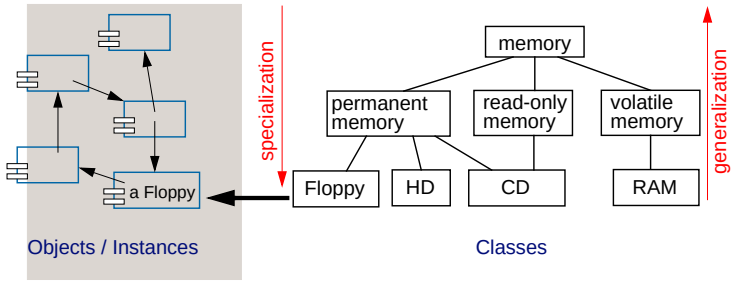
Bertrand Meyer:[Mey88]
*Computing systems perform certain actions on certain objects;*
*to obtain flexible and reusable systems,*
*it is better to base the structure of software*
*on the objects than on the actions.*






## 4 … Object-oriented Design (2)

- Software system is modeled as a collection of cooperating objects
- Each object is
  an instance of a class
  in a hierarchy of classes
- Example of a class hierarchy:









## 4 … Object-oriented Design (3)

- Concepts reflected in the structure of modern programming languages
  - ➤ Smalltalk
  - ➤ C++
  - ➤ Eiffel
  - ➤ Java
  - ➤ Ada
- General basis: object-oriented decomposition
- Advantages:
  - + Reusage of common mechanisms
    - ➔ software becomes smaller
  - + Modifications and improvements of the software become easier
  - + Results are less complex
  - + Better understanding of the principal's ideas

# C.5 Object-oriented Programming

## 1 Definition (Grady Booch)

OOP is a method of implementation in which programs are organized as

**cooperative collections of objects**,

each of which represents an

**instance of some class**,

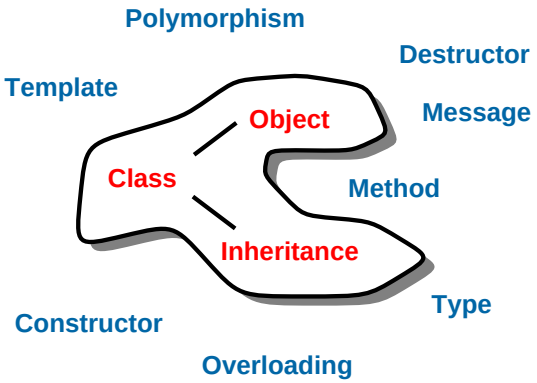
and whose classes are all members of a hierarchy of classes united via

**inheritance relationships**.






## 2 Basic Terms









## 3 Objects & Methods

- Software developer's view:
  - an object is a "thing" from the problem domain
    - has a state
    - has behavior
    - has a unique identity
- Program-technical point of view:
  - an encapsulated unit of data and functions that operate on this data
  - an object has a clear interface (operations = **methods**)

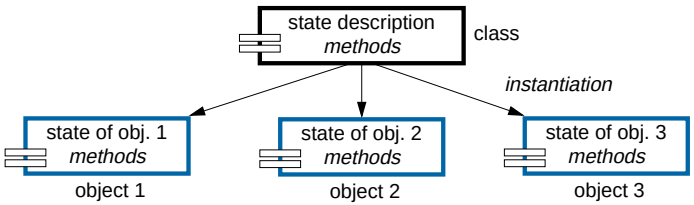

→ **object-based programming languages** [Weg87]






## 4 Classes

- Software developer's view:
  - a class is a set of objects with common structure and common behavior
- Program-technical point of view:
  - a class is a template for objects
    - each object is an instance of a class
    - object creation = *instantiation*




→ **class-based programming languages**
= objects & classes