# Object-oriented Programming

## *Overview*

- ❑ Motivation for the OO paradigm
- ❑ Software-design methods
- ❑ Basic terms of OO programming
- ❑ Basic concepts of the OO paradigm





# Motivation for OO

## *Software crisis*

- ❑ Software becomes more and more complex
- ❑ Less time & money for SW-development

## *Goals of OO*

- ❑ Increase productivity of the programmer by
  - ➔ Design patterns for problem solving
  - ➔ Reusage of software
  - ➔ Interfaces & modularity





# Object-oriented Programming

## *Definition (Grady Booch)*

OOP is 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

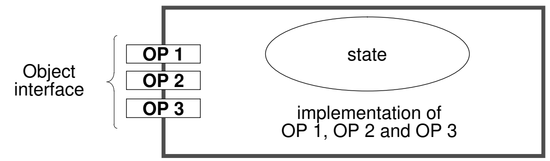
**inheritance relationships.**





# Objects, methods, & classes

- ❑ **Objekt** = Variables (state) + operations




- ❑ **Methods** = Operations of an object
- ❑ **Class** = Template for the creation of objects (instances)
  or: a set of objects that share a common structure & behavior

## Inheritance

- ❑ Deriving a "sub"-class from a single or from multiple "base"-classes
  - Adding methods and variables
  - Redefining methods
  - (Deleting methods and variables)
  - Late binding
- ❑ Advantage
  - Code reuse
- ❑ Inheritance in C++
  - No deletion
  - Visibility modifier: private, protected, public
  - Private vs. public inheritance





## Basic concepts of OOP

- ❑ Overview

| | |
|---|---|
| Encapsulation | Data abstraction |
| Modularity | Hierarchy |
| Typing | Concurrency |
| Persistence | Distribution |

- ❑ Encapsulation and data abstraction
  - Only methods should access variables
  - Abstract data type vs. object





## Basic concepts of OOP (2)

- ❑ Modularity
  - Groups of objects
  - Defined interfaces
  - Java packages
  - C++ files
- ❑ Hierarchy
  - Class hierarchy: *kind-of*
  - Object hierarchy: *part-of*



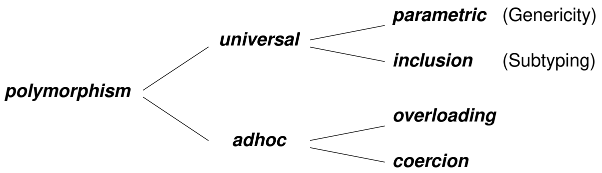


## Typing

- ❑ Characteristics of common properties (structure and behavior)
- ❑ Typing in OOP
  - Signatures of methods and semantics
  - Avoiding type errors by type checking
- ❑ Type hierarchy
  - Multiple classes implement a type
  - A class implements multiple types
- ❑ Subclassing vs. subtyping *(kind-of vs. is-a)*

# Polymorphism

❑ Classification




❑ Polymorphisms in C++
- Inclusion — public inheritance
- Parametric — templates
- Overloading — function and operator overloading
- Coercion — cast operators





# Dynamic Binding

❑ Method lookup
- depends on current instance,
- does not depend on the current type.

❑ A subclass (subtype) may redefine a method
- It will be looked up even in the context of the base-class (base-type) in case of dynamic binding

❑ C++
- **virtual** methods are dynamically bound
- nonvirtual methods are statically bound

❑ Java
- all methods are dynamically bound
- if a method is **final**ized it can be statically bound, but cannot be redefined in subclasses





# Abstract Classes

❑ A base class that does not define a method but only declares it:

pure virtual function

❑ We cannot create objects from abstract classes

❑ Available in both, C++ and Java

❑ Additionally in Java: Interfaces
- act as pure abstract classes (types)
- multiple inheritance of interfaces possible (but only single inheritance of classes)





# Genericity

❑ We can avoid dynamic type checking by (type-)parametrized classes (e.g., a stack)

❑ Generic class → actual class → instance of a class

❑ C++ templates
- One or mupltiple types are passed as parameters.
- They can be used in the context of a generic class and will be replaced with the current type-values when the actual class is created.

❑ Example:
- Integer-Stack
- String-Stack

# Concurrency and Persistence

- ❑ Concurrency
  - tendency to large, distributed applications → **concurrency**
  - orthogonal vs. nonorthogonal languages
  - uniform vs. nonuniform languages
  - Java: Thread classes, concurrency control by built-in monitors
- ❑ Persistence
  - Data may survive the environment of its creation and usage
  - Six levels of persistence
  - Overhead of making data persistence
  - OOP and persistence:

    — persistence of objects (realized by the run-time/operating system)

    — typing, inheritance, concurrency control available for persistent data