Special thanks to A. Vecchio for the previous version of these slides

Object-Oriented Programming: Basic Ideas

1

Java

Object-Oriented Programming

Object Oriented Programming (OOP) is a programming paradigm that is inspired to the way Man creates models for the comprehension of the real world.

The OOP adopt a number of mechanisms to control and to manage the complexity of a software project

Such a goal is pursued applying rules aimed at:

abstracting - generalizing - classifying

2

Special thanks to A. Vecchio for the previous version of these slides

Java

Basic Characteristics (I)

- **Encapsulation:** ability to tell apart the internal state (and behavior) fron the external state and behavior) of an object
  - **Data hiding:** ability to hide details on the internal state of an object
  - **Type Extendibility :** ability to add user-defined types to the native types of the language

3

Java

Basic Characteristics (II)

- **Inheritance:** ability to create new types by importing/reusing the description of existing types
- **Polymorphism:** ability to call the same functionality (possibly requiring different implementations depending on the use context) by means of a unique identifier. The proper implementation to use may be chosen either during the compilation phase or at runtime.

4

Java

Encapsulation and Classes

- The first important attempt to make use of encapsulation in a programming language has been done with the "ADT" concept (Abstract Data Type)
- The concept of "class" is a generalization of ADT, and it turns out to be more flexible.
- A class entity in Java can be created using the construct "class"
- A class entity in C++ can be created using the constructs "struct", "union", "class"

5

Java

Inheritance (I)

- Inheritance is a kind of relation that allow to organize classes within a program.
- Analyzing a new object to insert in a program we must cope with the following questions:
  - What are the similarities with the other objects?
  - And what about the differences?

6

Special thanks to A. Vecchio for the previous version of these slides

Java

Inheritance (II)

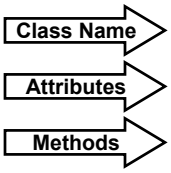
- **Classes can be organized according to a hierarchical model, containing *different levels*.**
- **The higher the level, the more generic the class; Each level contains more specific classes than the previous one.**
- **Inheritance, in OOP, is basically an abstraction mechanism.**

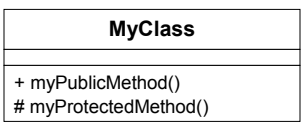
7

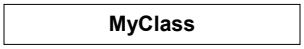
Java

Class Diagrams (I)

Many different graphicnotations are used in the literature: now we use a specific one, inspired to UML (Unified Modelling Language)







8

Special thanks to A. Vecchio for the previous version of these slides

Java

Class Diagrams (II)

In UML, different kinds of relations among classes are taken into account:

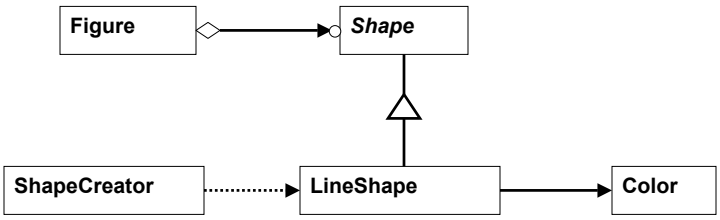
- **Derivation (Gen.-Spec.): in case a class is a direct subclass ("child") of a base class ("parent")**
- **Composition (aggregation, use): in case a class holds (refers), among its members, an instance(s) of another class.**
  - **exclusive (*aggregation*, part-of)**
  - **possibly shared (*acquaintance*)**
- **Association: *semantic* link among classes, characterized by a name, the roles of the involved classes, etc.**

9

Java

Class Diagrams (III)

Representation of different kinds of relations among classes



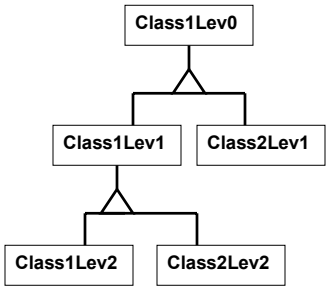
Upon the reference arrow,
it can be usually specified the corresponding field

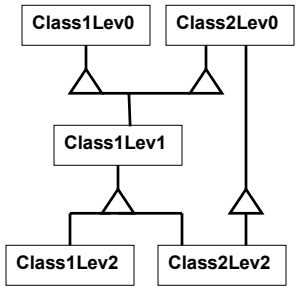
10

Java

Simple and Multiple Inheritance

- The organization of classes due to the inheritance relation is a *partial order*
- Two different kind of inheritance are used: *simple* and *multiple*





11

Java

The Overriding Mechanism

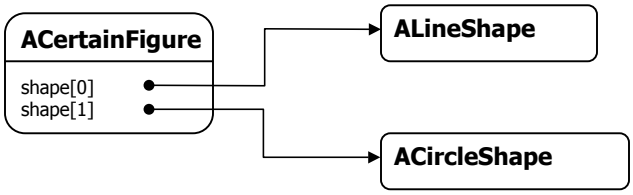
- In case the implementation of an inherited method had to be modified, a new method declaration (and definition) can be placed in the derived class, keeping both the name and the signature.
- The described operation is known as "*overriding*."
- As an "overridden" method is invoked, the executed version of the method depends on the actual type of the class instance used for the invocation.

12

Java

Object Diagrams

An *object diagram*
shows only *class instances*,
and it provides a view of (a portion of)
the program state (in terms of objects)
at a given moment in its execution.



13

Java

Polymorphism

- **In the assortment of programming languages, a number of mechanisms are present that show particular behaviors called "polymorphic" (I.e. same ID, multiple functionality).**
- **Also some OO languages make use of polymorphic mechanisms in order to enhance programming flexibility.**
- **OO languages, because of the adopted hierarchical organization of the classes, typically provide specific kinds of polymorphism that leverage the derivation relation.**

14

Special thanks to A. Vecchio for the previous version of these slides

Java

Different Kinds of Polymorphism

- **Ad hoc Polymorphism**
  - **coercion: a function/method/operator acts upon values of different types by converting them into the expected type**
  - **overloading: A specific function/method is called by taking into account its signature too**
- **Pure Polymorphism**
  - **parametric polymorphism: the type is left unspecified by the programmer, and it will be automatically instantiated later (at compilation time)**
  - **inclusion: Functions in the base type keep on working in the sub-type too. Thus the same function may have many different implementations, and the proper one (at a certain execution point) is chosen at runtime, by the identification of the actual sub-type.**

15

Java

Strategies for Class Reuse (I)

- **White-box reuse (reuse by inheritance)**
  - **pros:**
    - **It's simple to modify part of the implementation (by overriding)**
    - **It' defined statically, at compilation time**
  - **cons:**
    - ***"linheritance breaks encapsulation",* I.e. derived classes usually can (must) access the internals of the base class:**
    - **If something is changed within the base class, often this operation yields mandatory modifications in the derived class**
    - **The hierarchical relation hampers flexibility: the use of abstract classes is recommended**

16

Special thanks to A. Vecchio for the previous version of these slides

Special thanks to A. Vecchio for the previous version of these slides

Java

Strategies for Class Reuse (II)

- **Black-box reuse (reuse by composition)**
  - **pros:**
    - **Encapsulation is not broken (objects are accessed only by their interface)**
    - **Few dependencies by the implementation, as the implementation of an object is done in terms of interfaces of other objects**
    - **It can be done at runtime**
  - **cons:**
    - **Interfaces must be strictly respected, and thus they must be designed very accurately**

17

Java

Two Principles for OOP

According to Gamma, Helm, Johnson, Vlissides:

- ***program focusing on the interfaces, instead of implementations***
- ***prefer object composition, instead of class inheritance***

18