Object-Oriented Programming

Inheritance and polymorphism exercices

Anders Jonsson & Federico Heras
2023-24

Course topics

Topic 1 Introduction and the concept of objects

Topic 2 The object-oriented programming paradigm

Topic 3 Object modelling and relations between objects

Topic 4 Inheritance and polymorphism

Topic 5 Abstract classes and interfaces

Topic 6 Reuse and study of problems solved using objects

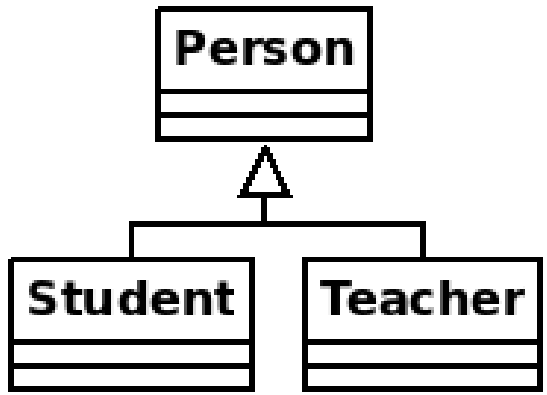
Inheritance

- Makes it possible to specialize the definition of an existing superclass
- All instance members of the superclass are inherited
- Reuse: the definition of the superclass is reused
- Inheritance also makes it possible to group common members
- Reuse: avoids duplication of attributes and methods

Polymorphism

- Treat a set of objects as if they belong to the same class
- Polymorphic variables: declared type $\neq$ instantiated type
- Overriding: redefine an inherited method in a subclass
- The method executed depends on the instantiated type
- Needs dynamic binding to work
- Casting: change declared type of variable storing an instance

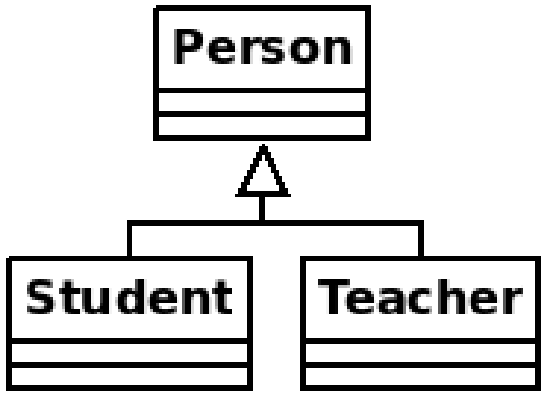
Upcast



- Always permitted: an instance of the subclass is also an instance of the superclass

```
Student s = new Student( "Eva", 20, 4 );
Person person = s; // upcast
```

Downcast



- Can fail: an instance of the superclass may not be of the correct subclass

```
Teacher t = new Teacher( "Oscar", 38 );
Person person = t; // upcast
Student student = (Student)person; // downcast
```

The Object class

- In Java, the Object class is a superclass of all other classes
- If a class header does not contain extends, by default the class inherits from Object!
- Consequently, when creating an instance of any class the constructor of Object is always executed
- The instance methods of Object can be applied to any instance:
  - `String toString()`
  - `boolean equals( Object obj )`
  - `int hashCode()`
  - `void wait()`

Exercise

- Define a class `ComparableObject` which represents objects with single attributes that can be compared
- Override methods `toString` and `equals` of the Object class
- Add a method `lessThan` that tests whether one instance is less than another

Exercise

- Define a class `ComparableVector` that represents vectors of comparable objects
- The class should contain methods `addObject` and `contains`
- Define a new class `SortedVector` that represents sorted vectors, by overriding the methods `addObject` and `contains`
- Define a new class `NoRepVector` for vectors that contain no repeated objects, by overriding the method `addObject`

Exercise

- Define a new class `SortingAlgorithm` with a method for sorting arrays
- Define subclasses that override the sorting method
- Add a relation between `ComparableVector` and `SortingAlgorithm` that makes it possible to sort the elements of `ComparableVector`

Exercise

- Design a class `Stream` that represents a data stream with inputs and outputs
- The class should be able to read input from the keyboard and write output on the screen
- Define a class `FileStream` that can read and write from files, redefining the methods for reading input and writing output

Exercise

- Design a graphical hierarchy able to draw two main shapes Circle and Rectangle. Use the Java classes JFrame, JPanel and Graphics
  - Define the Java hierarchy for a graphical application
  - Define the hierarchy for the shapes