Object-Oriented Programming

Inheritance and polymorphism

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

Classes and instances

- Fundamental units of object-oriented programming
- **Class**:
  - Abstract idea
  - Represents an object family
  - Defines the attributes and methods that are shared by all objects in this family
- **Instance**:
  - Concrete object
  - Belongs to an object family (that is, a class)
  - Assigns a concrete value to each attribute of the class

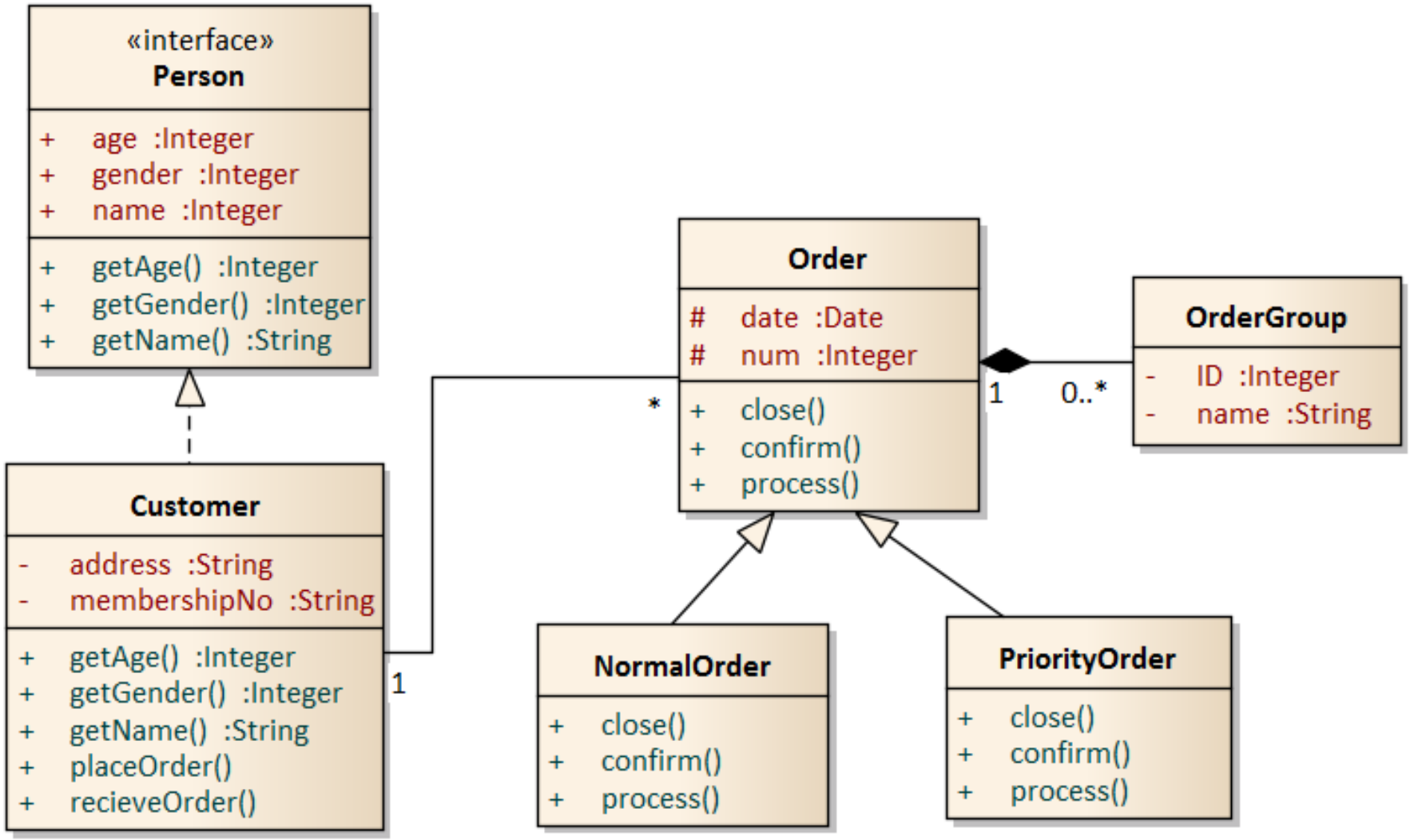
Example

```
public class Key {          // class header
   private String color; // attribute "color"
   private String shape; // attribute "shape"
   private Door door;    // attribute "door"

   // constructor method
   public Key( String c, String s, Door d ) {
      color = c;
      shape = s;
      door = d;
   }

   // getter
   public String getColor() {
      return color;
   }
}
```

Class diagram



Theory session 3

Object hierarchies

Inheritance

Visibility

Constructor methods

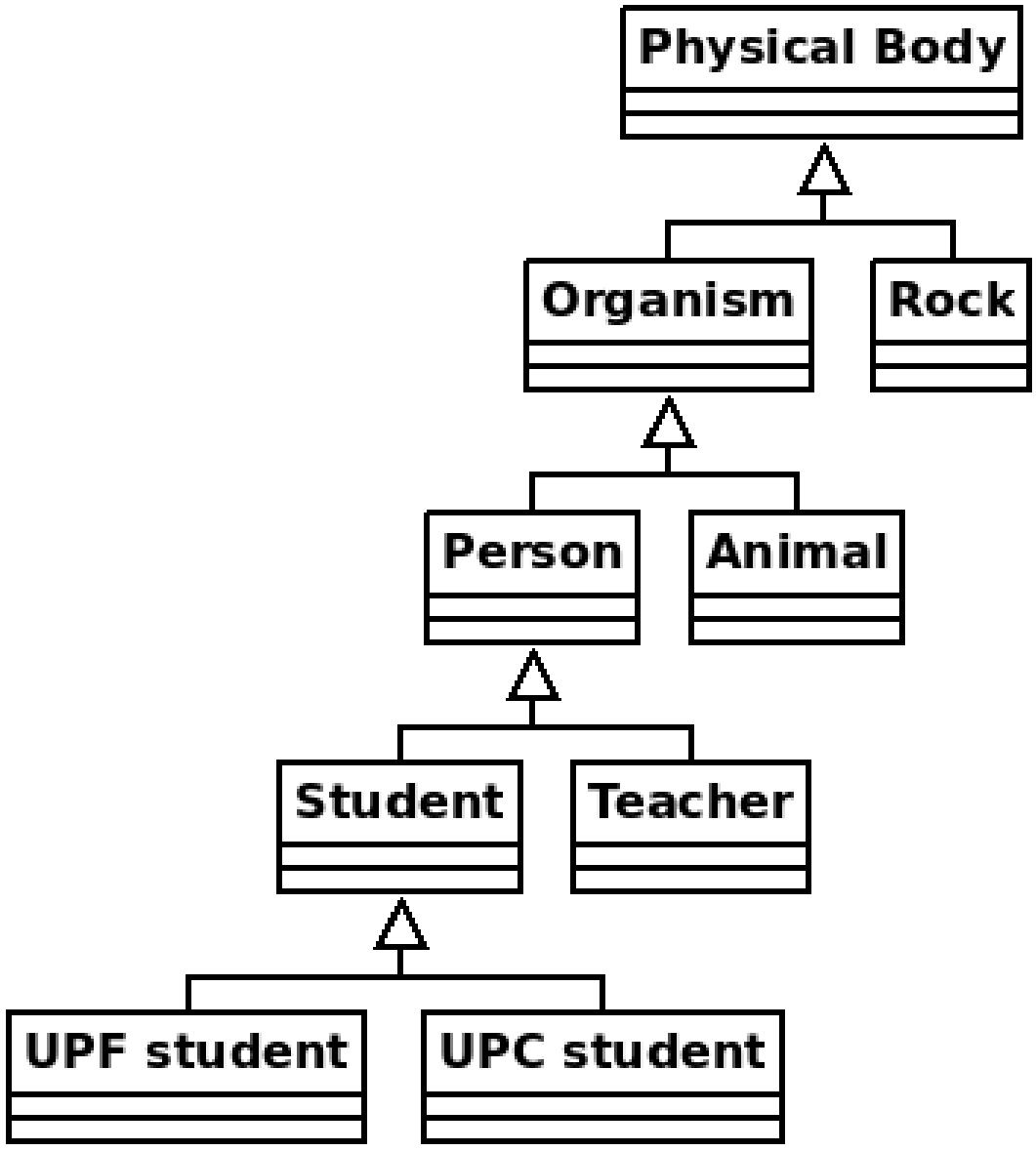
Intuition

- Instance: object that belongs to an object family
- A single object can belong to multiple families
- Example: John Smith, UPF student

Multiple families

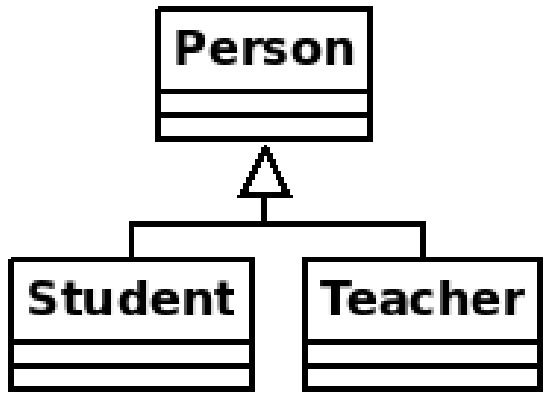
- John Smith is a UPF student
- Consequently, he is also a student
- Any student is also a person
- A person is an organism
- An organism is a physical body (that has mass and weight)

Hierarchy


Physical Body
Organism
Rock
Person
Animal
Student
Teacher
UPF student
UPC student

Definition of hierarchy

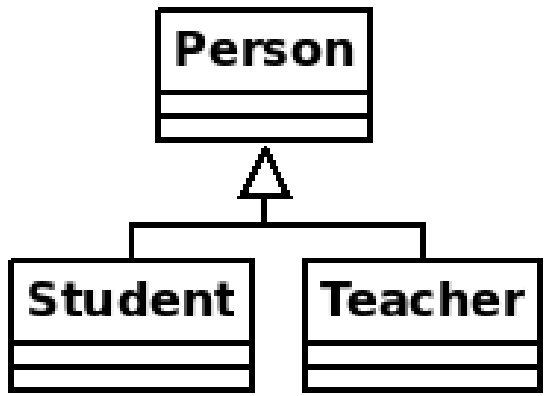
- Hierarchy: tree whose nodes are object families (classes)



- Example: Student is a child of Person in the hierarchy
  - Student is a subclass (derived class, child class) of Person
  - Person is a superclass (base class, parent class) of Student

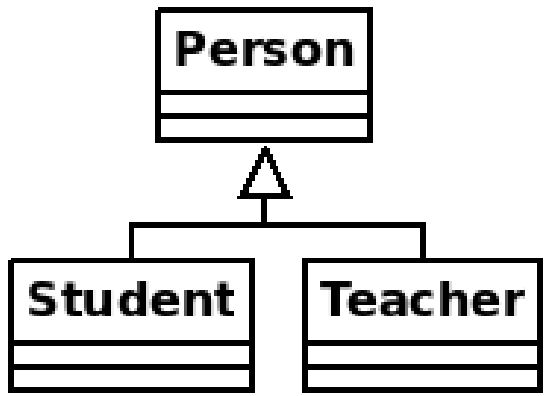
Relation among classes

- What does the relation subclass-superclass imply?



Relation among classes

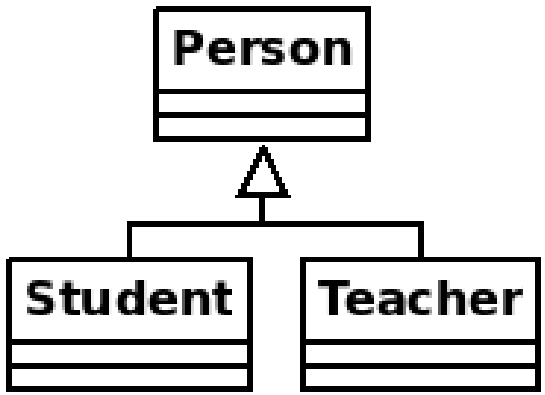
- What does the relation subclass-superclass imply?



- A person is not necessarily a student
- In contrast, each student is also a person
- $\Rightarrow$ an instance of Student is also an instance of Person

Set representation

- Each class represents a set of objects



- O(Student): set of objects represented by Student
- O(Person): set of objects represented by Person
- Relation subclass-superclass implies O(Student) $\subseteq$ O(Person)
- An instance cannot belong to two classes on the same level!

Exercise

- Design an object hierarchy that categorizes the animals in a zoo

Theory session 3

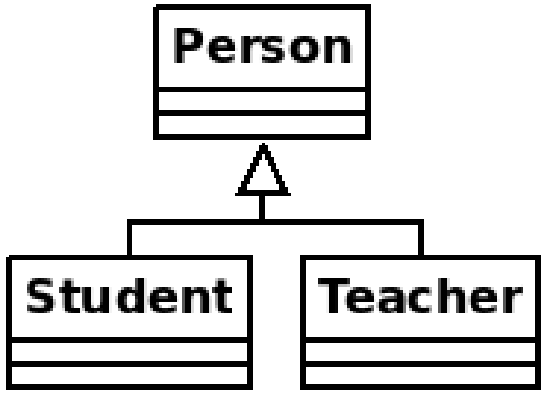
Object hierarchies

Inheritance

Visibility

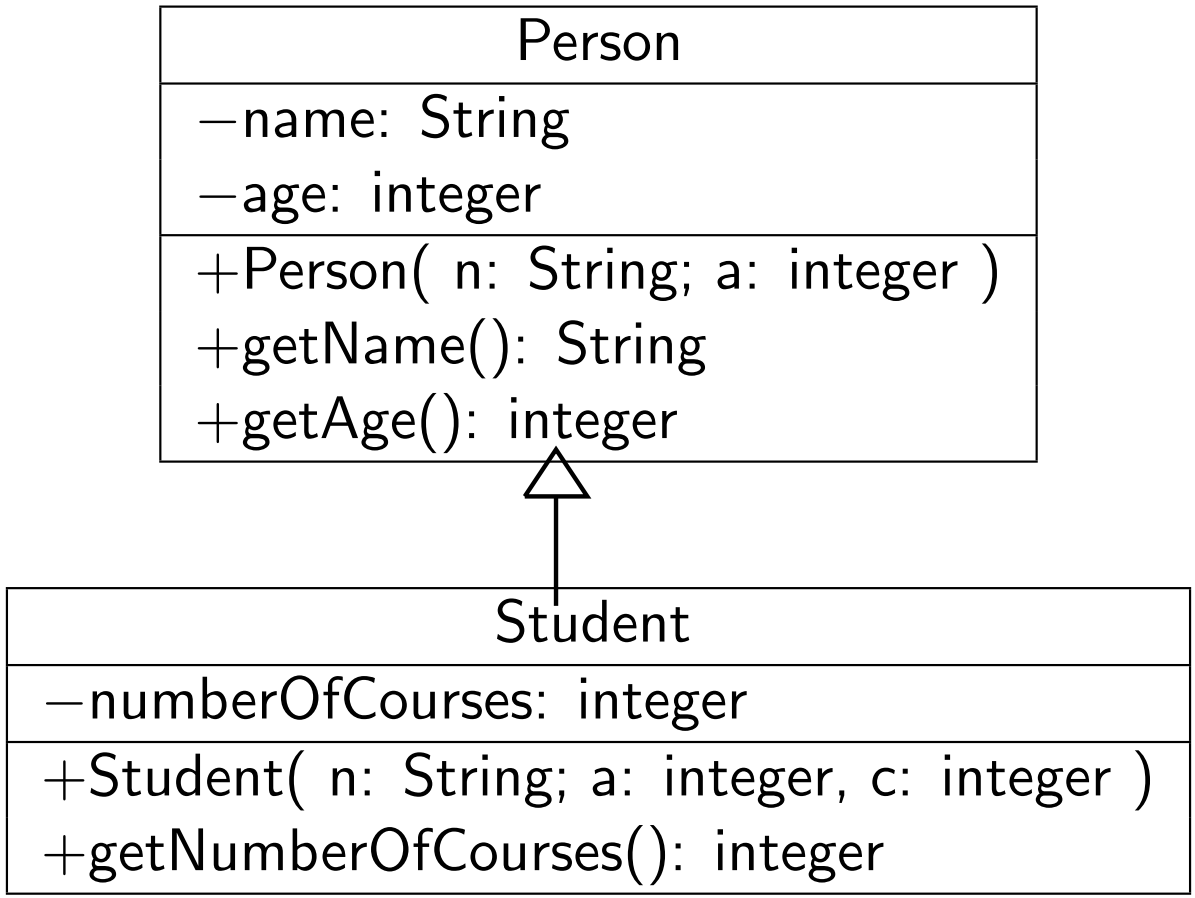
Constructor methods

Inheritance



- Each instance of Student is also an instance of Person
  $\Rightarrow$ the instance members of Person also apply to Student!
- Student inherits the instance members of Person
- It is not necessary to redefine these members in Student!
- Class members (defined using static) are not inherited in Java

Example


Person
−name: String
−age: integer
+Person(n: String; a: integer)
+getName(): String
+getAge(): integer
Student
−numberOfCourses: integer
+Student(n: String; a: integer, c: integer)
+getNumberOfCourses(): integer

Java code

```
public class Person {
     private String name; // attribute "name"
     private int age;     // attribute "age"

     public Person( String n, int a ) {
          name = n;
          age = a;
     }

     public String getName() {
          return name;
     }

     public int getAge() {
          return age;
     }
}
```

Inheritance

```
public class Student extends Person {
     private int numberOfCourses;

     public Student( String n, int a, int c ) {
          super( n, a );
          numberOfCourses = c;
     }

     public int getNumberOfCourses() {
          return numberOfCourses;
     }
}
```

Program

```
public class TestStudent {
     public static void main( String[] args ) {
          Student s1 = new Student( "Juan", 19, 5 );
          Student s2 = new Student( "Eva", 20, 4 );

          String name = s1.getName();
          int courses = s2.getNumberOfCourses();
     }
}
```

Inheritance in C++

```
class Student : public Person {
private:
     int numberOfCourses;

public:
     Student( std::string n, int a, int c )
          : Person( n, a )
          , numberOfCourses( c ) {
     }

     int getNumberOfCourses() {
          return numberOfCourses;
     }
};
```

Inheritance – Advantages

- 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
- Flexibility: new instance members can be defined in the subclass

Inheritance – Advantages

- Inheritance also makes it possible to group common members
- Example: our program includes classes Student and Teacher, both with attributes “name” and “age”
- We can define a new class Person with these attributes and make Student and Teacher subclasses of Person
- Reuse: avoids duplication of attributes and methods

Inheritance – Applicability

- There are different relations between classes
- It is important to know when to apply inheritance
- If the statement "an A is a B" holds, then A is naturally a subclass of B
- Examples:
  - A dog is a mammal: Dog can inherit from Mammal
  - A car is an engine: Car should not inherit from Engine

Inheritance - Disadvantages

- Overhead in method calls
  - The method is not necessarily found in the class itself!
- Additional work during design
  - Identify appropriate relations among classes
  - Danger of defining hierarchies that are too deep
- Difficult to change the hierarchy once defined
- Dislocation of code (difficult to know where members are defined)

Theory session 3

Object hierarchies

Inheritance

Visibility

Constructor methods

Visibility

- To each member we can associate a level of visibility
- Private visibility: internal member, restricted access
- Public visibility: external member, open access
- Mechanism for achieving encapsulation

Visibility and inheritance

- Each instance member of the superclass is inherited (private or not)
- Members with private visibility are inaccessible from the subclass
- Solution: protected visibility
- Members with protected visibility are accessible from the subclass

Visibility in Java

Modifier	Class	Package	Subclass	World
public	✓	✓	✓	✓
protected	✓	✓	✓	✗
no modifier	✓	✓	✗	✗
private	✓	✗	✗	✗

Java code

```
public class Person {
     protected String name; // attribute "name"
     protected int age;     // attribute "age"

     public Person( String n, int a ) {
          name = n;
          age = a;
     }

     public String getName() {
          return name;
     }

     public int getAge() {
          return age;
     }
}
```

Theory session 3

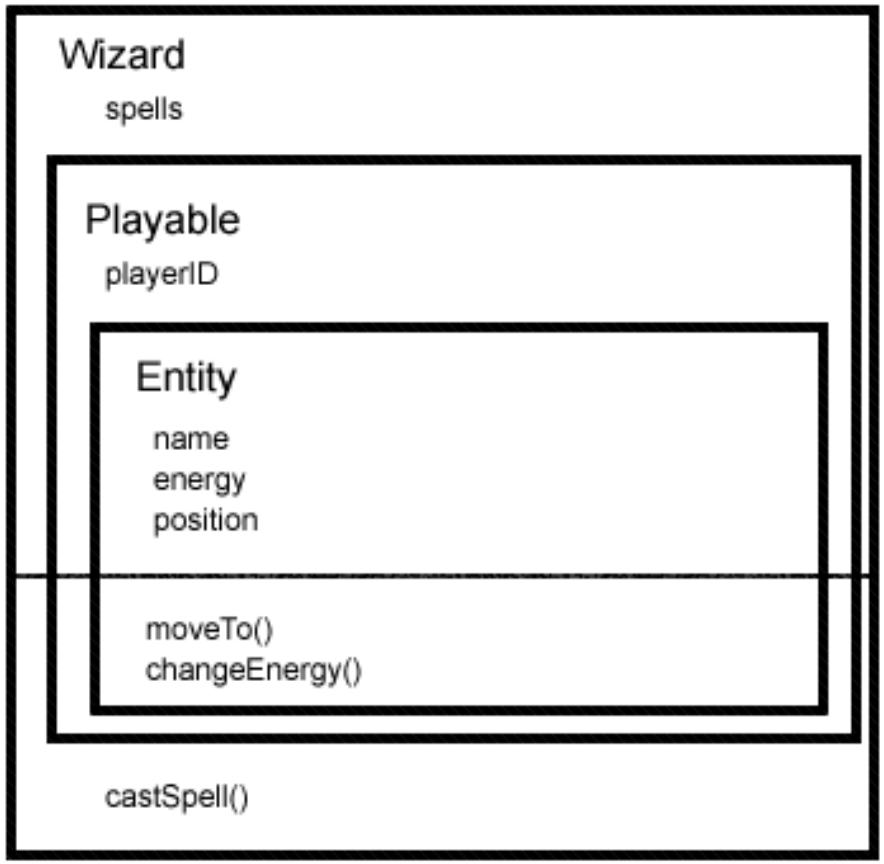
Object hierarchies

Inheritance

Visibility

Constructor methods

Constructor methods

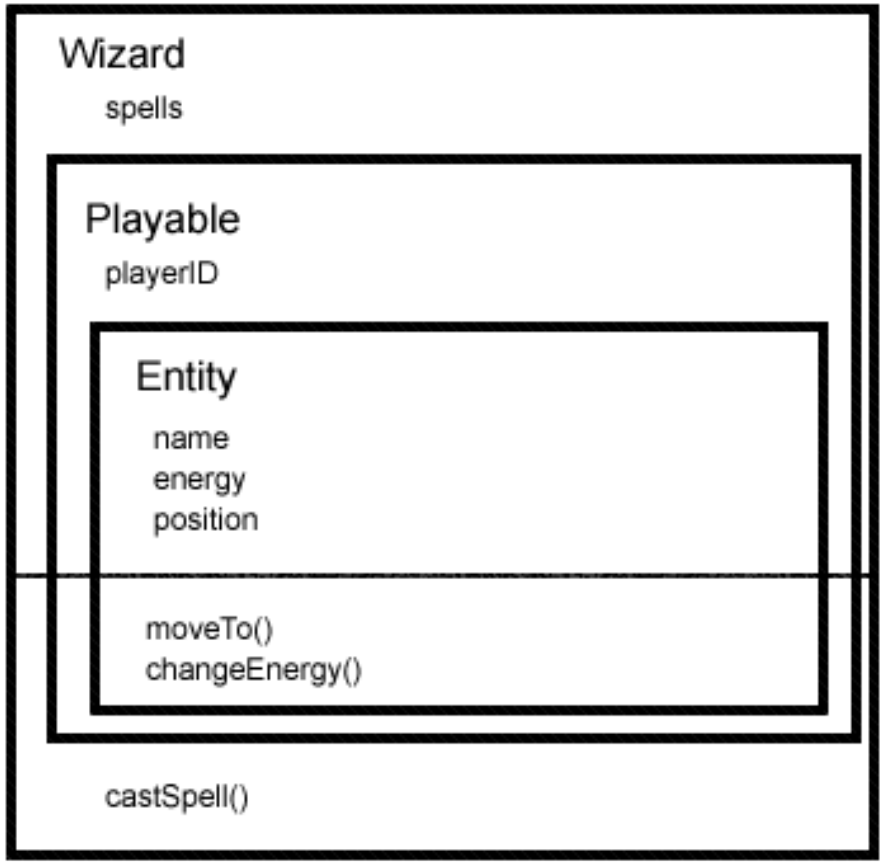


- An instance can contain members from multiple classes
- Which constructor should we apply to create an instance?

Constructor methods

- A constructor method always calls a constructor method of the superclass
- The constructor of the superclass executes first
- Recursive mechanism: constructor of the superclass calls the constructor of its superclass, etc.

Example



- Creating an instance of Wizard executes this sequence:
  - Constructor method of the class Entity
  - Constructor method of the class Playable
  - Constructor method of the class Wizard

The keyword super

- In Java, the constructor method can specify which constructor of the superclass is called using the keyword super
- The keyword super always appears on the first line!

```
public class Student extends Person {
        private int numberOfCourses;

        // constructor method
        public Student( String n, int a, int c ) {
                super( n, a );
                numberOfCourses = c;
        }
}
```

Constructor methods in Java

- If a class does not contain a constructor, the compiler adds the empty constructor
- If a constructor does not call the constructor of the superclass, the compiler adds the call super() on the first line
- Can cause confusing compiler errors

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()`

Inheritance or aggregation?

- Often we want to modify an existing class X
- Two main options:
  - Inheritance: define a subclass of X and add new members
  - Aggregation: include an attribute of type X in our class
- Which option is better?
  $\Rightarrow$ depends on whether inheritance is natural

Exercise

- Define a class Ellipse with methods for computing the circumference and area
- Define a class Circle that reuses the class Ellipse:
  - first by inheritance
  - then by aggregation

Summary

- Object hierarchy
- Inheritance
- Protected visibility
- Constructor methods