Object Oriented Programming

Inheritance II

Programación II
Facultad de Ingeniería
Universidad Austral

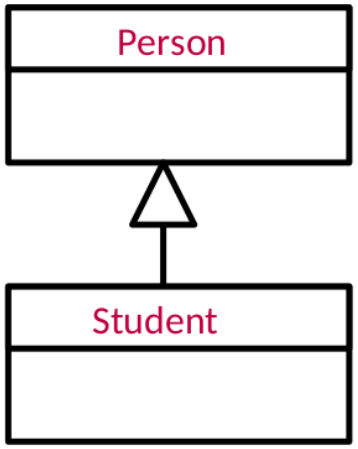
Casting

- Many languages support type casting between numeric types

```
int i = 7;
float f = (float) i; // f==7.0
double d = 3.2;
int i2 = (int) d; // i2==3
```

- With inheritance it is reasonable to type cast an object to any of the types above it in the inheritance tree…

Widening



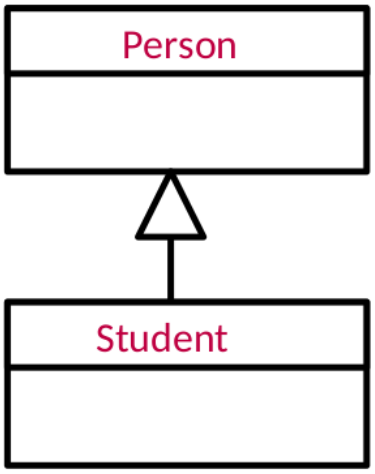
- Student is-a Person
- Hence we can use a Student object anywhere we want a Person object
- Can perform *widening* conversions (up the tree)

```
Student s = new Student()

Person p = s;
```

```
public void print(Person p) {...}

Student s = new Student();
print(s);
```

Implicit widening

Narrowing



- Narrowing conversions move down the tree (more specific)
- Need to take care...

OK because underlying object really is a Student

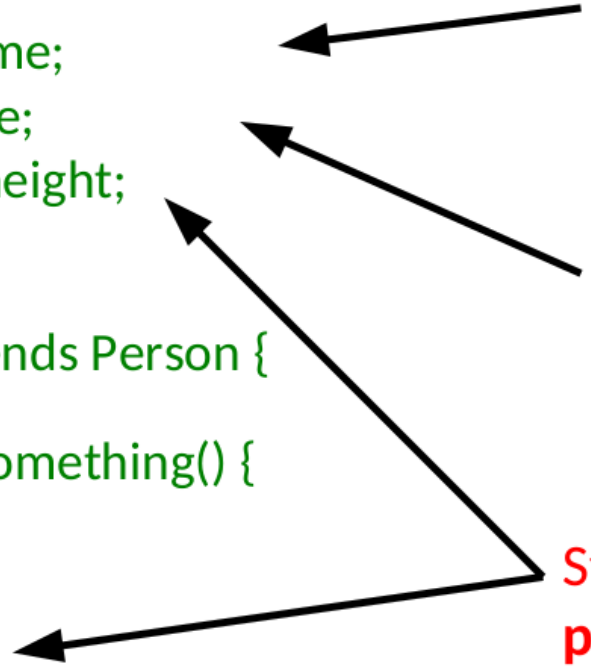
```
public void print(Person p) {
  Student s = (Student) p;
}

Student s = new Student();
print(s);
```

```
Person p = new Person();

Student s = (Student) p;
```

FAILS at runtime. Not enough info In the real object to represent a Student

Fields and Inheritance

```
class Person {
   public String name;
   protected int age;
   private double height;
}

class Student extends Person {

  public void do_something() {
    name=”Bob”;
    age=70;
    height=1.70;
  }

}
```



Student inherits this as a public variable and so can access it

Student inherits this as a protected variable and so can access it

Student inherits this but as a **private** variable and so cannot access it directly
This line doesn’t compile

Fields and Inheritance: Shadowing

```
class A {   public int x; }

class B extends A {
   public int x;
}

class C extends B {
  public int x;

  public void action() {
     // Ways to set the x in C
     x = 10;
     this.x = 10;

     // Ways to set the x in B
     super.x = 10;
     ((B)this).x = 10;

     // Ways to set the x in A
     ((A)this.x = 10;
  }
}
```

'this' is a reference to the current object

'super' is a reference to the parent object

all classes extend Object (capital O)

if you write 'class A {}' you actually get 'class extends Object {}'

Object a = new A(); // substitution principle

Don't write code like this. No-one will understand you!

Methods and Inheritance: Overriding

- We might want to require that every Person can dance. But the way a Lecturer dances is not likely to be the same as the way a Student dances...

Know the difference: overriding vs overloading

```
class Person {
  public void dance() {
    jiggle_a_bit();
  }
}

class Student extends Person {
  public void dance() {
    body_pop();
  }
}

class Lecturer extends Person {
  public void dance(int duration) {...}
}
```

Person defines an original implementation of dance()

Student overrides the original

Lecturer overloads the inherited dance() method

Lecturer inherits the original implementation and jiggles

Abstract Methods

- Sometimes we want to force a class to implement a method but there isn't a convenient default behaviour
- An abstract method is used in a base class to do this
- It has no implementation whatsoever

```
class abstract Person {
     public abstract void dance();
}
class Student extends Person {
     public void dance() {
        body_pop();
     }
}
class Lecturer extends Person {
     public void dance() {
        jiggle_a_bit();
      }
}
```

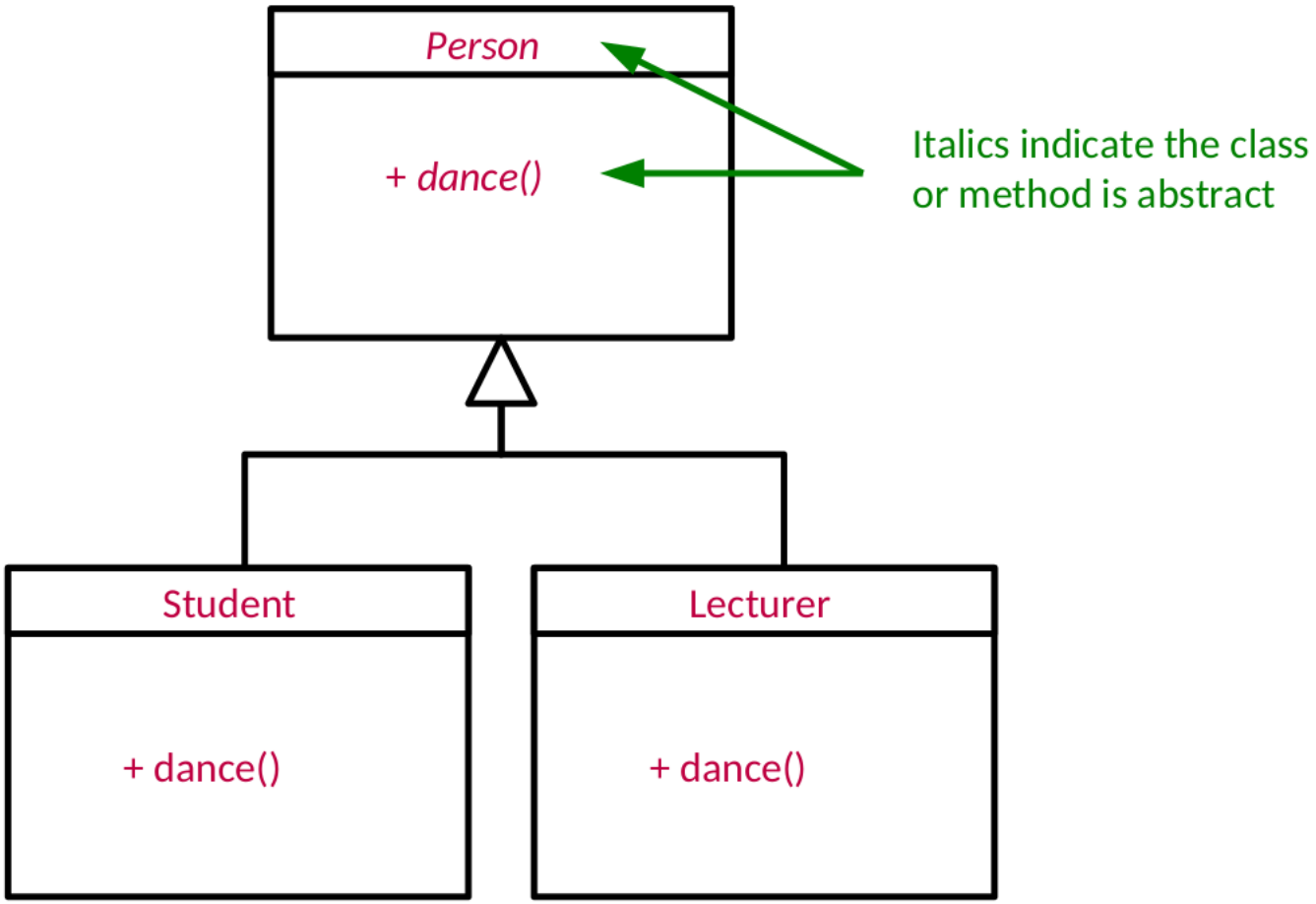
Abstract Classes

- Note that I had to declare the class abstract too. This is because it has a method without an implementation so we can't directly instantiate a Person.

```
public abstract class Person {
    public abstract void dance();
}
```

- All state and non-abstract methods are inherited as normal by children of our abstract class
- Interestingly, Java allows a class to be declared abstract even if it contains no abstract methods!

Representing Abstract Classes




Questions?