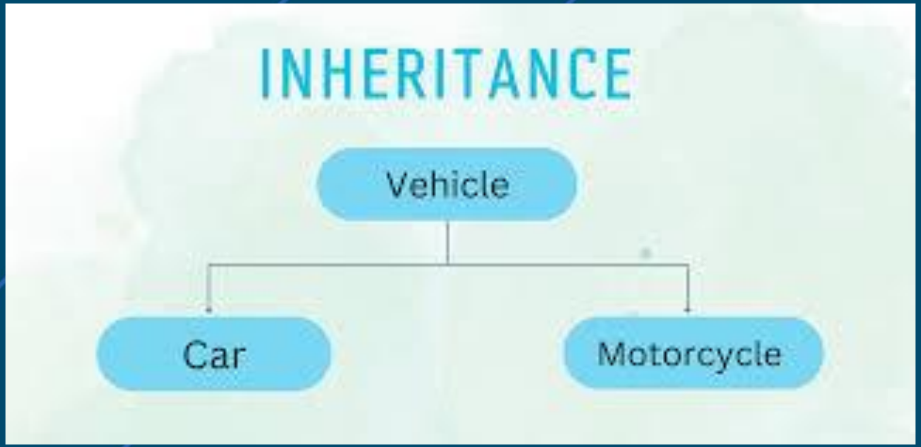
Féidearthachtaí as Cuimse
Infinite Possibilities



Inheritance

Object Oriented programming

OLLSCOIL TEICNEOLAÍOCHTA BHAILE ÁTHA CLIATH
TU DUBLIN
TECHNOLOGICAL UNIVERSITY DUBLIN

Inheritance

- Dictionary

`“To receive from predecessors”..`

Scenario

College system – stores details on staff and students

Scenario

College system – stores details on staff and students

Student - spec

// attributes, inc programme, year of study etc

// behaviour (methods) includes: getters, setters, etc

etc

Staff

// attributes inc line manager, department etc

// behaviour (methods) includes: getters, setters, etc

Let’s look at the code….

Scenario

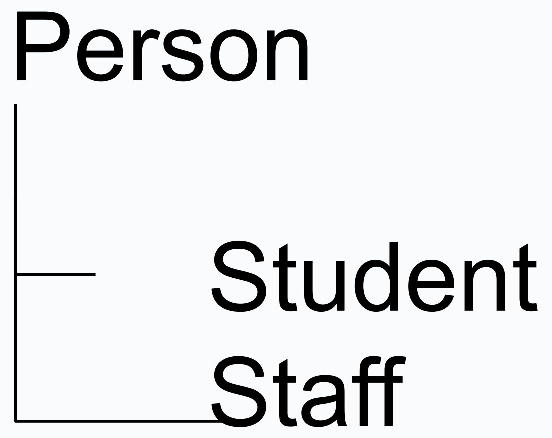
What's the code overlap?

What's the problem?

What's the solution?

Inheritance in OO

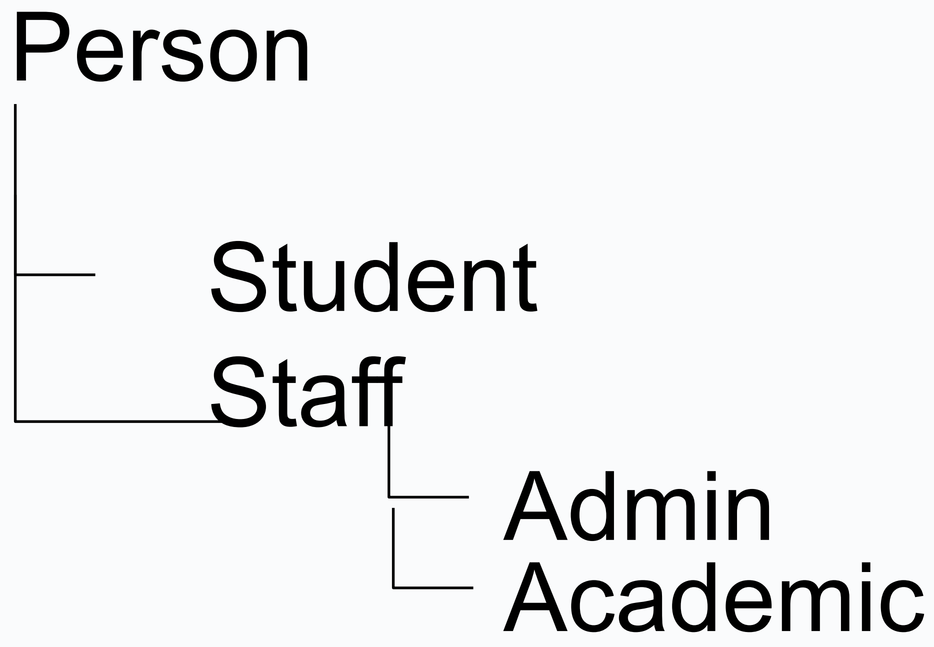
“Is type of “



Purpose : to re-use code (avoid rewriting new code)
A subclass inherits variables and methods from its parents

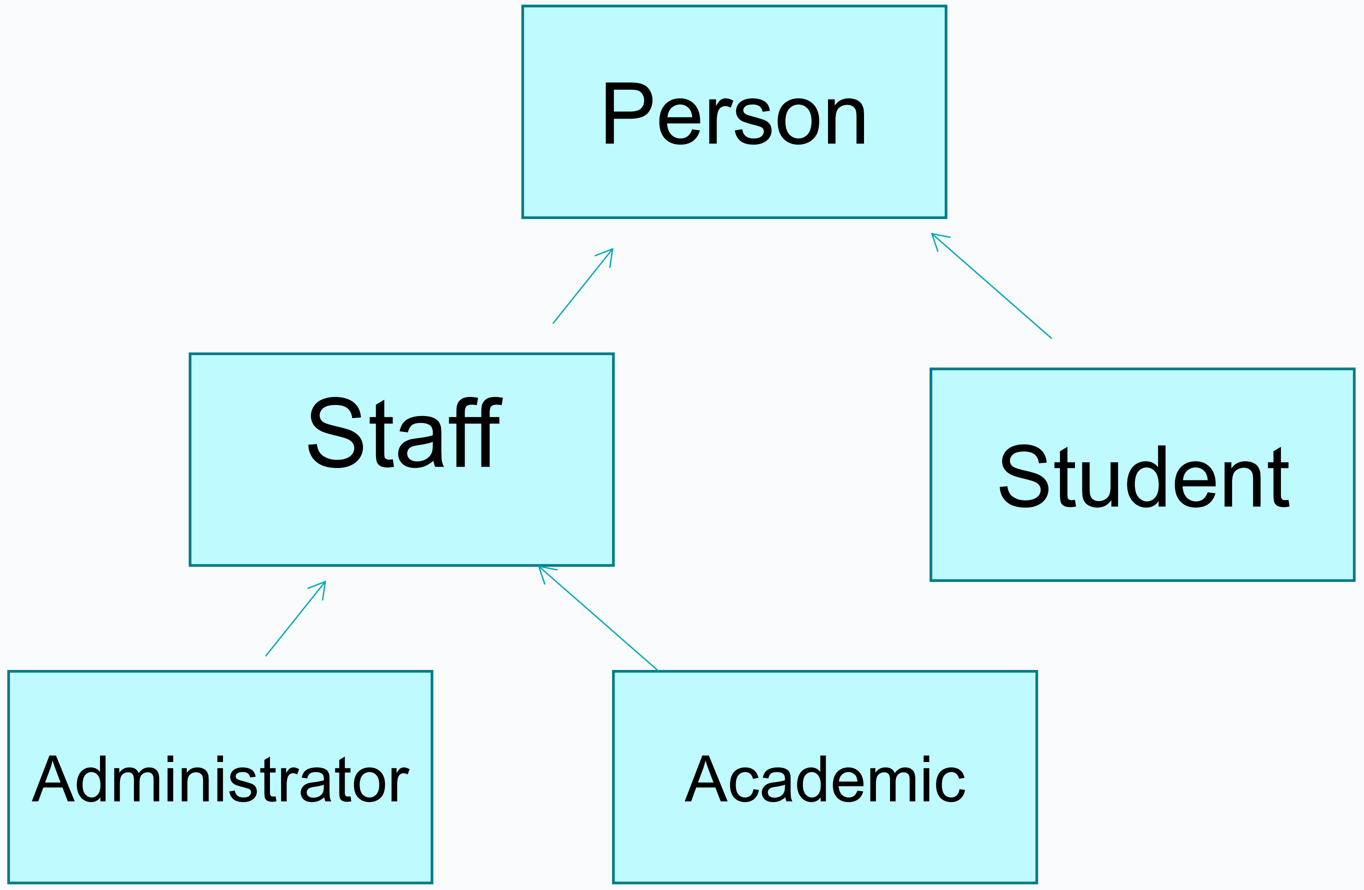
Inheritance in OO – multi layered

“Is type of “



Super classes and sub classes

How many
super classes and
sub classes
are shown?

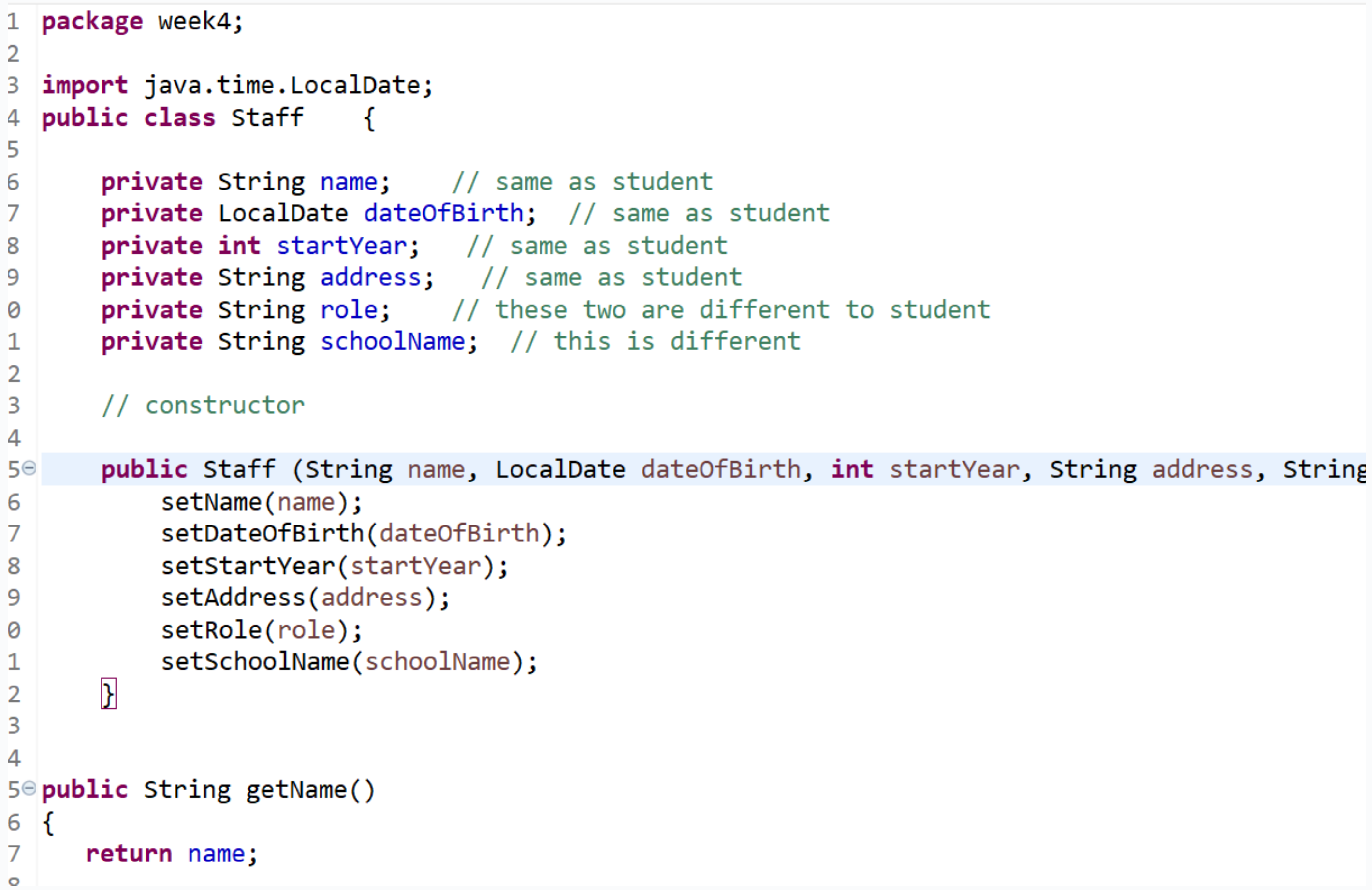


can be identified from the class hierarchy

Subclass

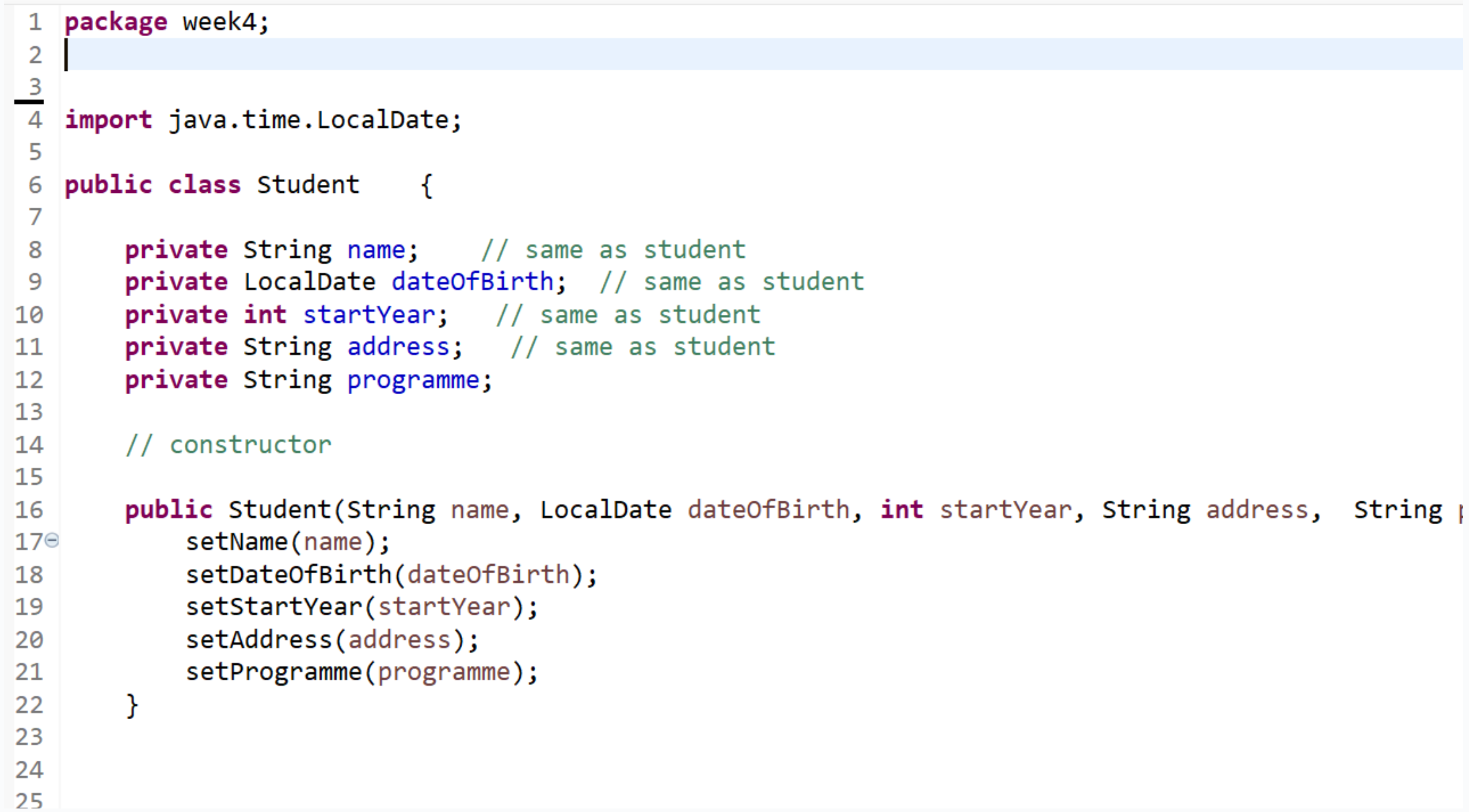
- Inherits the members (attributes and methods)
- *Adds its own specific members (attributes and members)*
- Overrides methods (behaviour) as needed
- Example: Person/ Student/ Staff etc

Class Student (Bright Space



```
package week4;

import java.time.LocalDate;
public class Staff    {

    private String name;    // same as student
    private LocalDate dateOfBirth;  // same as student
    private int startYear;   // same as student
    private String address;   // same as student
    private String role;    // these two are different to student
    private String schoolName;  // this is different

    // constructor

    public Staff (String name, LocalDate dateOfBirth, int startYear, String address, String
        setName(name);
        setDateOfBirth(dateOfBirth);
        setStartYear(startYear);
        setAddress(address);
        setRole(role);
        setSchoolName(schoolName);
    }


public String getName()
{
   return name;
```

Class Staff



```java
package week4;


import java.time.LocalDate;

public class Student    {

    private String name;    // same as student
    private LocalDate dateOfBirth;  // same as student
    private int startYear;   // same as student
    private String address;   // same as student
    private String programme;

    // constructor

    public Student(String name, LocalDate dateOfBirth, int startYear, String address,  String 
        setName(name);
        setDateOfBirth(dateOfBirth);
        setStartYear(startYear);
        setAddress(address);
        setProgramme(programme);
    }



```

Class Person --- code (BrightSpace)

Inherited by subclasses: all public methods (get*, set*, toString() unless overridden).

•**Not inherited:** constructors.

•**Private fields** are not directly accessible in subclasses (use getters/setters).

Summary of steps– Look at Code example on Brightspace

Concept	What it Does	Example
extends	Makes one class inherit from another	class Student extends Person
super(...)	Calls the parent class constructor	super(name, age)
super.toString()	Calls the parent class method	super.toString()
@Override	Indicates that we’re replacing a parent method	@Override public String toString()

To implement inheritance in java

```
public class Student **extends** Person
```

Note the constructor:
Use "**super**" to call constructor of superclass from subclass

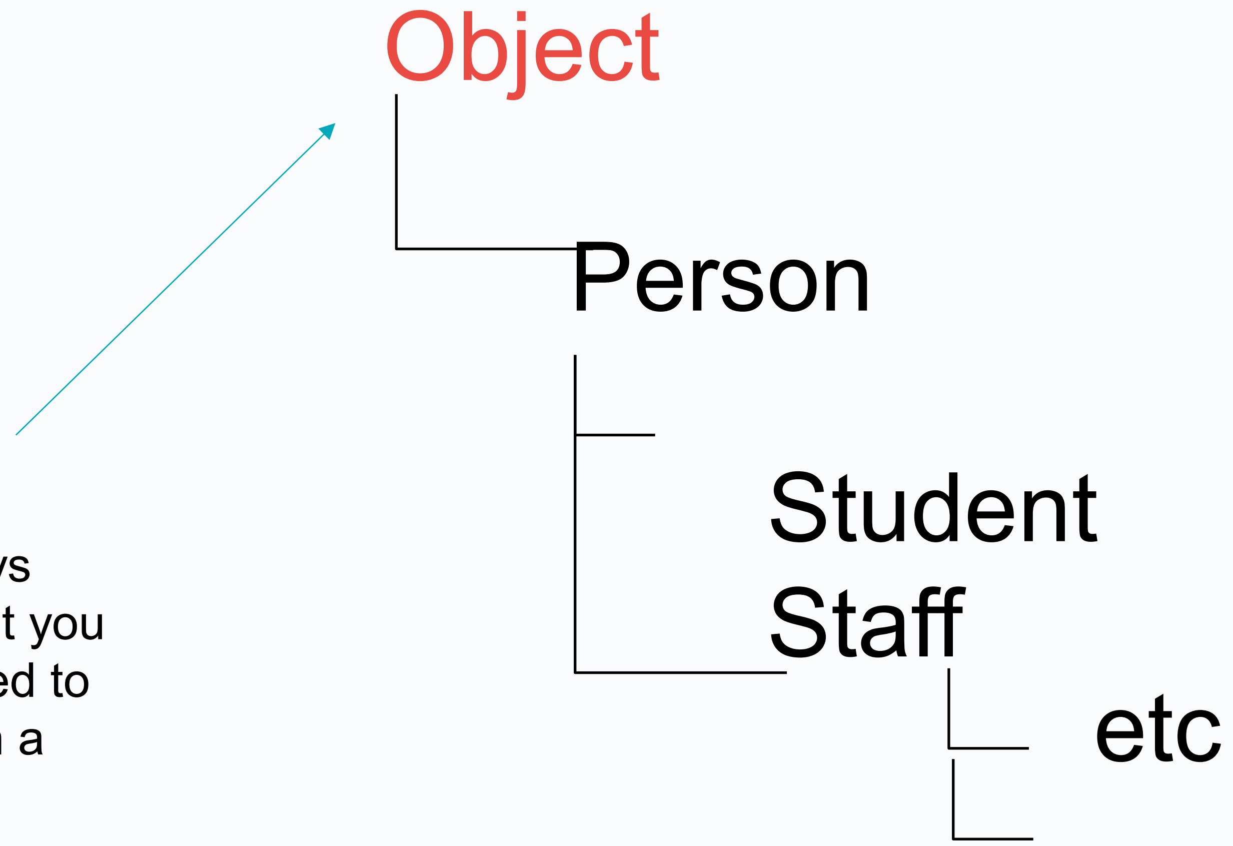
Examine the code – and write the Staff class

Note: “ Object” The root class at the top

inheritance allows us to create a new class from an existing class so that we can effectively reuse existing code. All classes in Java are inherited from a pre-written base class known as the Object class.

- The Object class
- “Adam and Eve” object
- A class with no superclass, extends this class
- toString() behaviour.. How is inheritance linked to this?

Object class

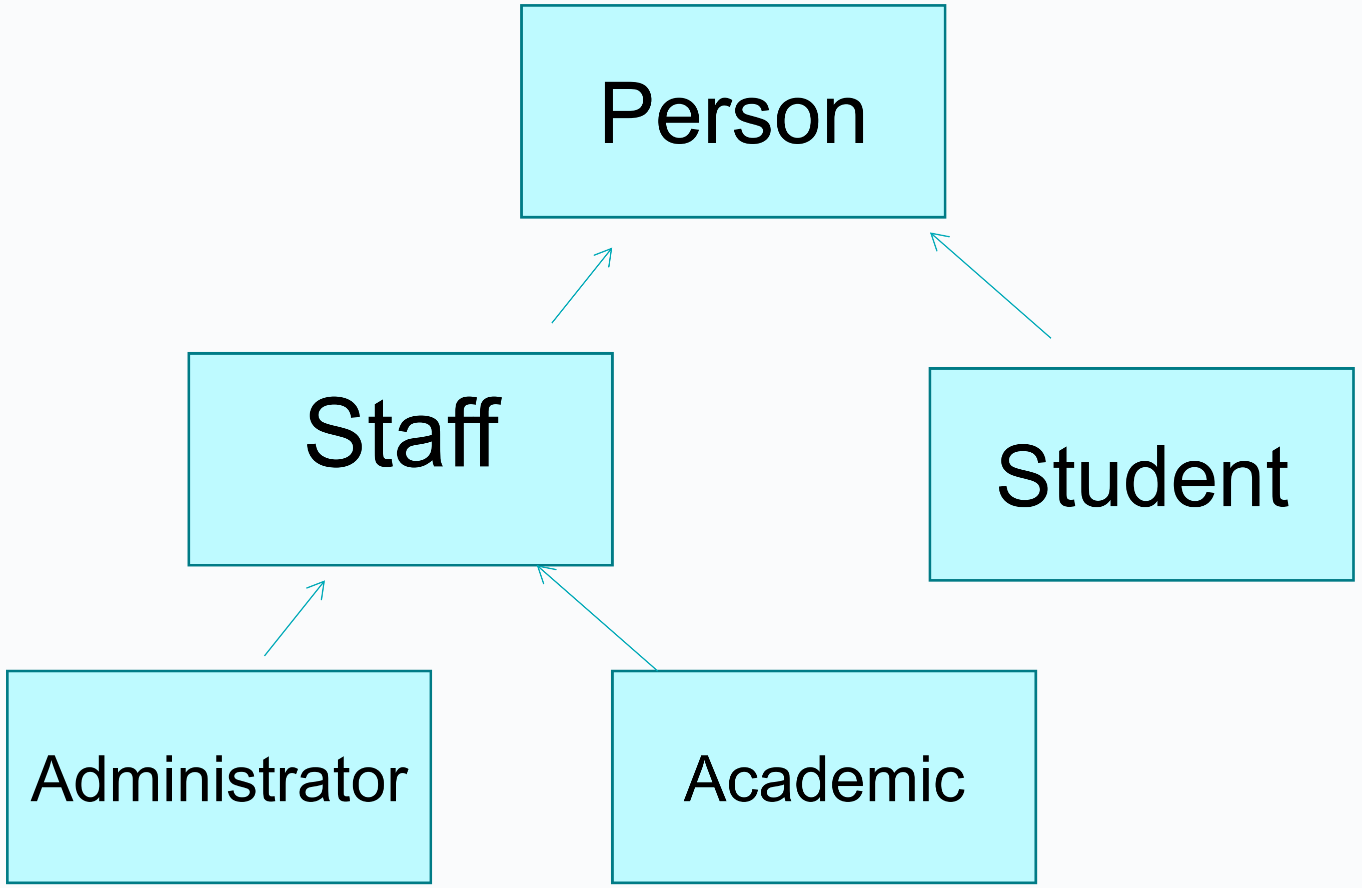


It's always there, but you don't need to draw it in a design !

Object “Types”

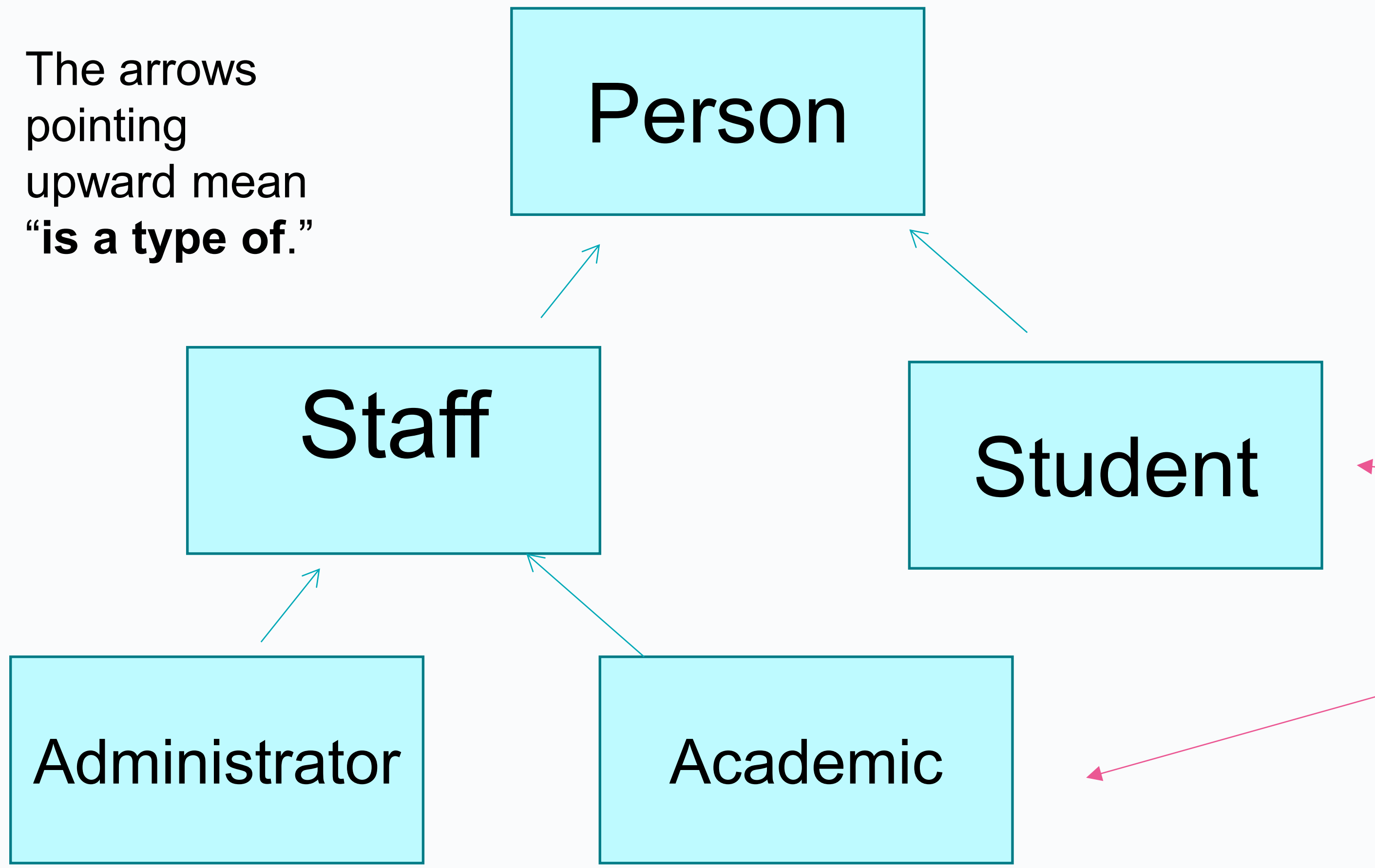
An important concept in java

Objects created from subclasses can be treated as objects of their parent classes —
this is the basis of **polymorphism** and **type hierarchy** in object-oriented programming.



Object Types

The arrows pointing upward mean "**is a type of**."



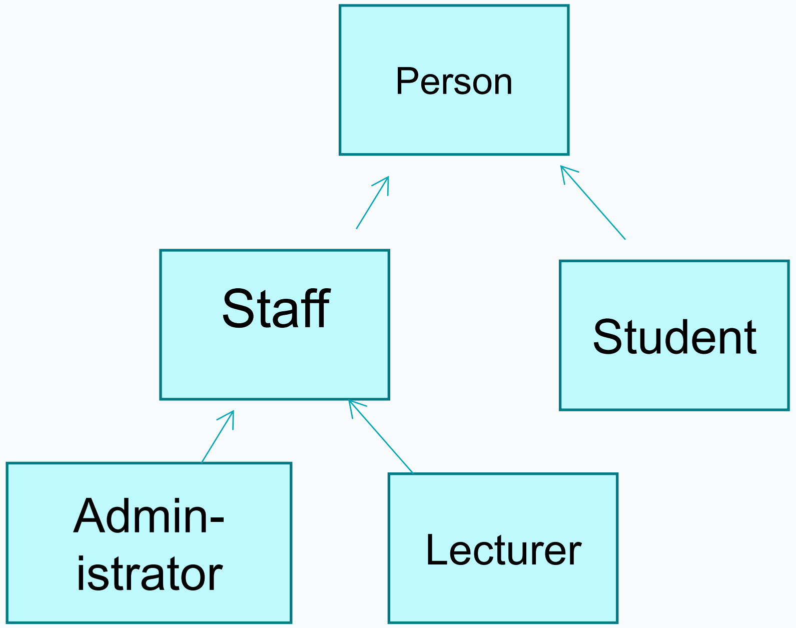
What is the object type(s) for an instance of this class

Or this class

Casting objects

“Casting” is taking an object of one type and converting into another type

In class hierarchies.. works a specific way:



Example

```
Person p1 = new Student();  // create a person object
Student s1 = (Student) p1;   // changes a person object called p1 into a Student object
```

Or upcasting

```
Student s1 = new Student();   // A Student object
Person p1 = s1;               // Upcasting: Student → Person
```

```
Person p1 = new Staff();
Staff  a1 = (Staff) p1;
```

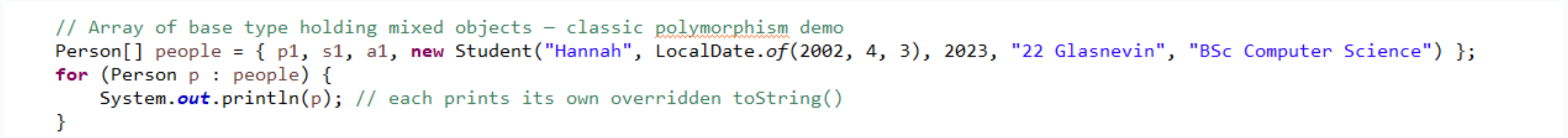
polymorphism

```
Person p = new Student(); // Upcasting
p.printInfo();             // Prints: "This is a Student"
```

Method Overriding

- Different classes in the hierarchy do things in “their own way” – i.e. have their own version of a method
- Note: Use **super.superclassmethod()** from the subclass method if the superclass does part of the work.
  - avoiding code repetition
- An example is the toString() method

Essentials of Method Overriding



```
// Array of base type holding mixed objects – classic polymorphism demo
Person[] people = { p1, s1, a1, new Student("Hannah", LocalDate.of(2002, 4, 3), 2023, "22 Glasnevin", "BSc Computer Science") };
for (Person p : people) {
    System.out.println(p); // each prints its own overridden toString()
}
```

- Same **method name**
- Same **parameter list**
- Same or **compatible return type**
- **Occurs between superclass and subclass**
- **@Override** annotation (recommended)
- **Access level** cannot be reduced
- **Static** and **final** methods **cannot** be overridden
- **Happens at runtime** (polymorphism)

Question

- What is the difference between method overriding and method overloading?

What we covered

- Inheritance
  - Why it’s used - No 1 reason: code re-use
  - How it’s used - “extends”
- “Object” class

- Object types / Casting
- Method overriding
- Polymorphism
- Abstract classes
- “final” keyword