An abstract network diagram with white nodes and lines on a dark blue background, representing a complex graph structure. The nodes are connected by thin white lines, forming a web-like pattern that fills the right side of the slide.

CS1101

Programming and Problem Solving

Dr. Gina Bai
Spring 2023

Logistics

- **ZY-9** on zyBook > Assignments
 - Due: **Wednesday, April 19**, at 11:59pm
- **PA11 - A, B** on zyBook > Chap 11
 - Due: **Thursday, April 20**, at 11:59pm

Start Early!!!

Logistics (The End of The Semester is Approaching...)

Sun	Mon	Tue	Wed	Thu	Fri	Sat
	April 17 Lecture	April 18 Last Recitation Sessions	April 19 Lecture ZY-9 Due Last Recitation Sessions	April 20 PA11 Due Immersion Showcase 5-7PM, @ FGH Atrium	April 21 Lecture	
	April 24 Final Review Last Day of Classes & Office Hours ZY-10 Due	April 25	April 26	April 27	April 28 Final Exam 7-10PM @ Sarratt Cinema	

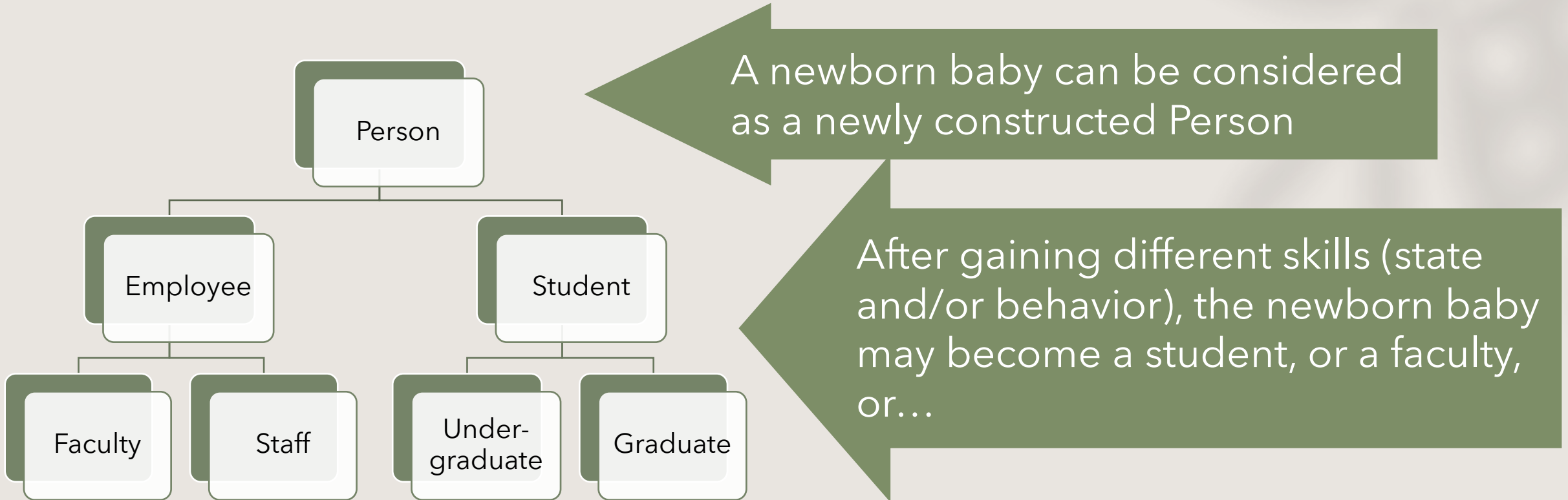
Four Main Principles of OOP

1. Abstraction
2. Encapsulation
- 3. Inheritance**
4. Polymorphism

Inheritance Basics

zyBook Chap 10

Inheritance – Real World Example

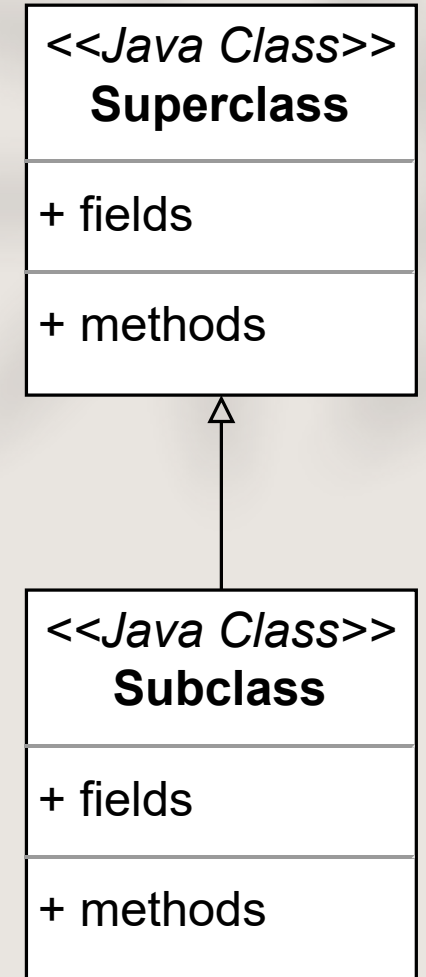
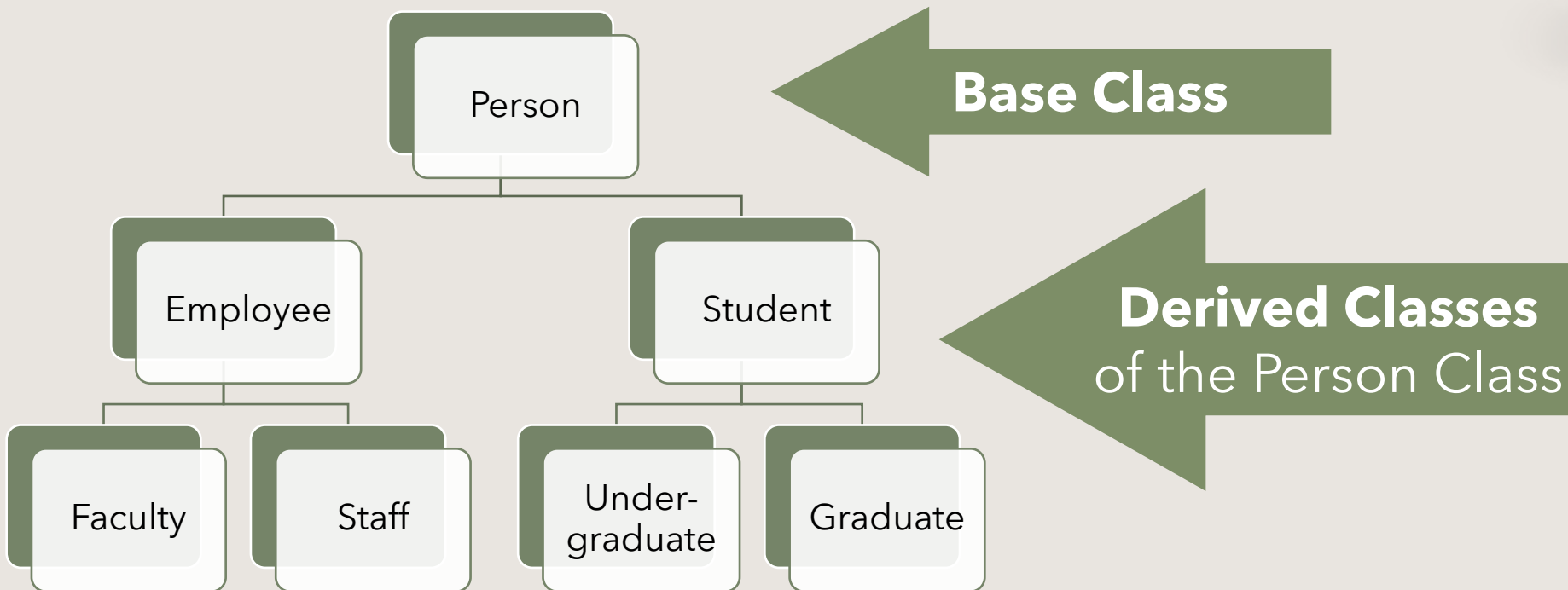


Inheritance

A programming technique in which a **derived class** **extends** the **functionality** of a **base class**, **inheriting** all of its **state** and **behavior**

- One class is an **extension/specialization** of another
- Advantages
 - Enhance Code Reusability
 - Reduce redundancy

Inheritance in Programming



Terminologies

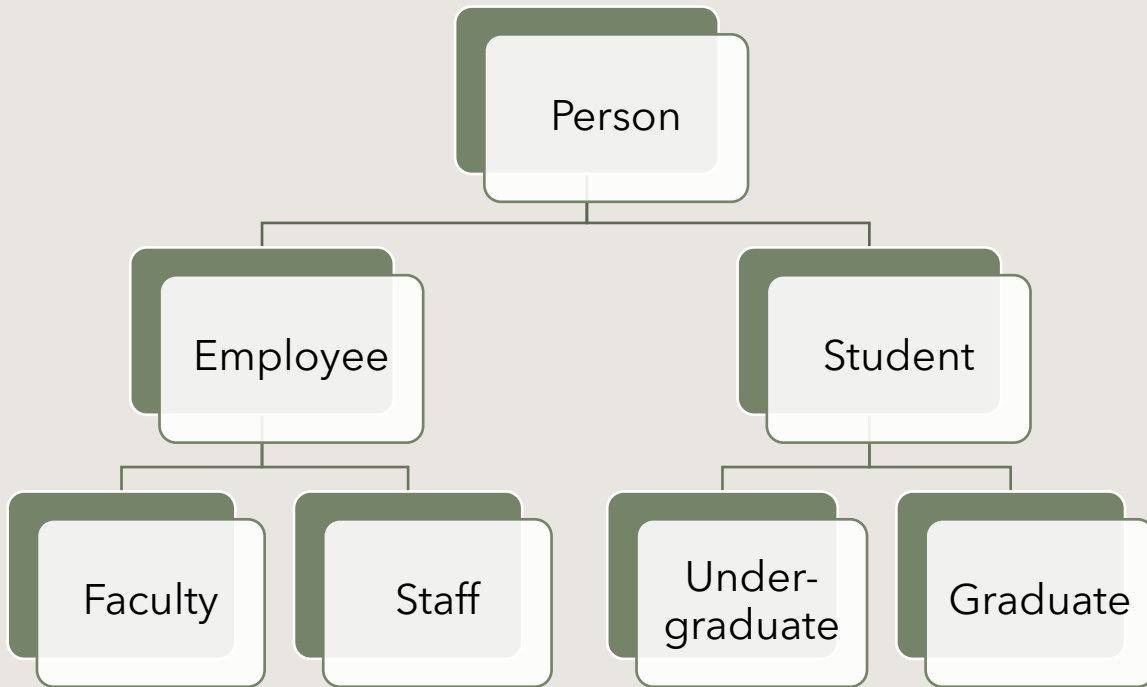
- **Subclass** / Child Class / Derived Class / Extended Class
 - A class which inherits the other class
- **Superclass** / Parent Class / Base Class
 - A class from where a subclass inherits the features
- Subclass **extends** Superclass

```
// <SuperclassName>.java
public class <SuperclassName> {
    // Some code...
}
// <SubclassName>.java
public class <SubclassName> extends <SuperclassName> {
    // Some code...
}
```

→ A Java keyword

→ The idea of "increases the functionality"

Syntax – Example



```
// Person.java  
public class Person {  
    /* ...Code Here... */  
}
```

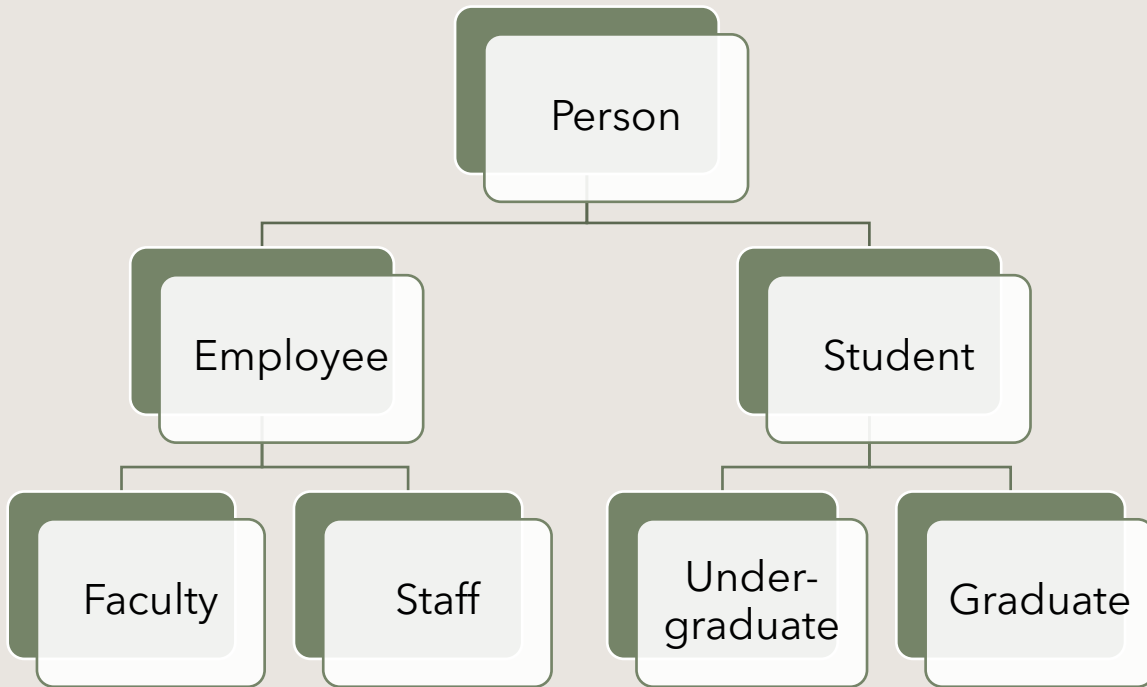
```
// Student.java  
public class Student extends Person {  
    /* ...Code Here... */  
}
```

```
// Employee.java  
public class Employee extends Person {  
    /* ...Code Here... */  
}
```

```
// Faculty.java  
public class Faculty extends _____ {  
    /* ...Code Here... */  
}
```

```
// Graduate.java  
public class Graduate extends _____ {  
    /* ...Code Here... */  
}
```

Syntax – Example



```
// Person.java  
public class Person {  
    /* ...Code Here... */  
}
```

```
// Student.java  
public class Student extends Person {  
    /* ...Code Here... */  
}
```

```
// Employee.java  
public class Employee extends Person {  
    /* ...Code Here... */  
}
```

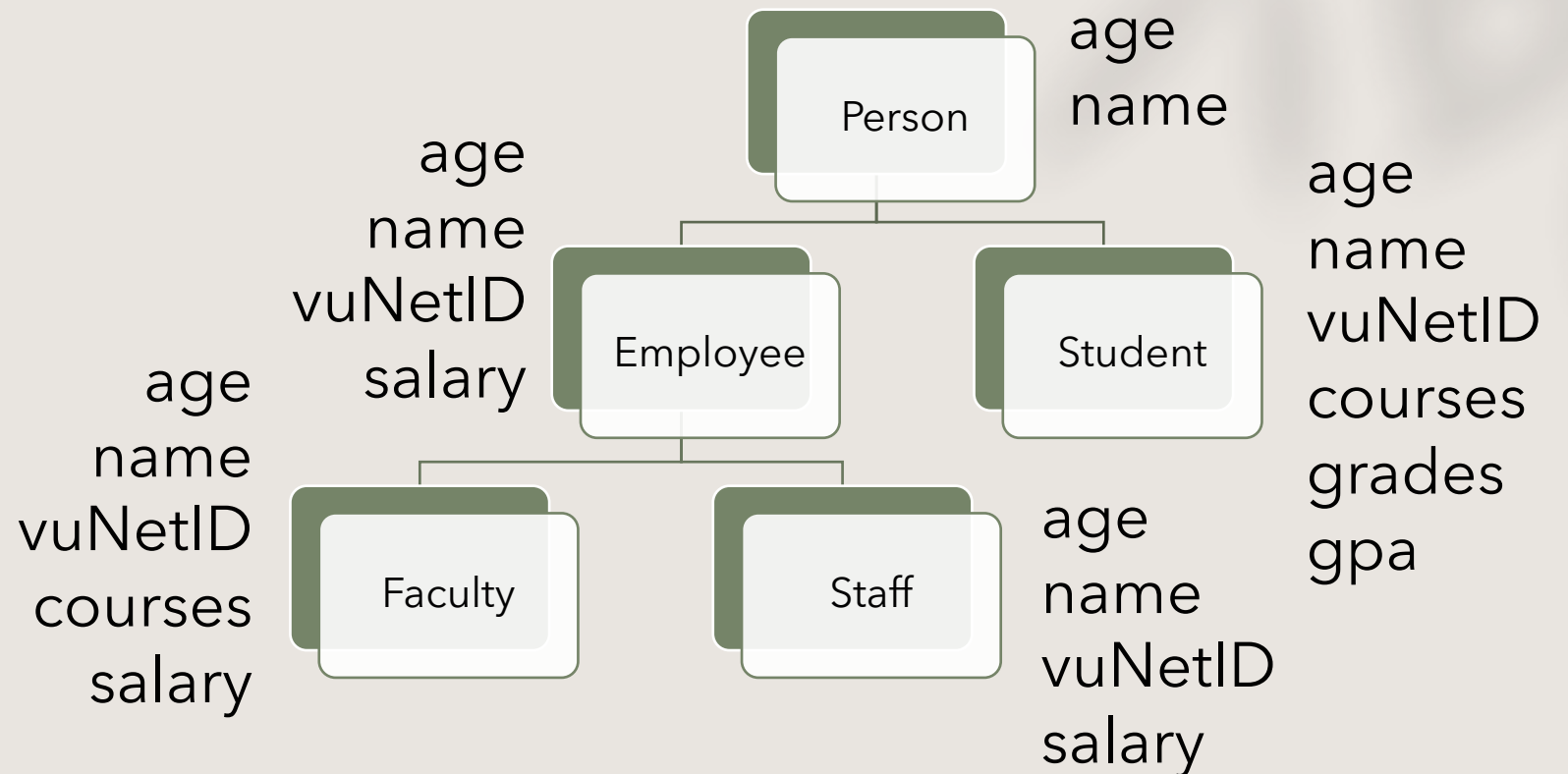
```
// Faculty.java  
public class Faculty extends Employee {  
    /* ...Code Here... */  
}
```

```
// Graduate.java  
public class Graduate extends Student {  
    /* ...Code Here... */  
}
```

Design Class Hierarchy

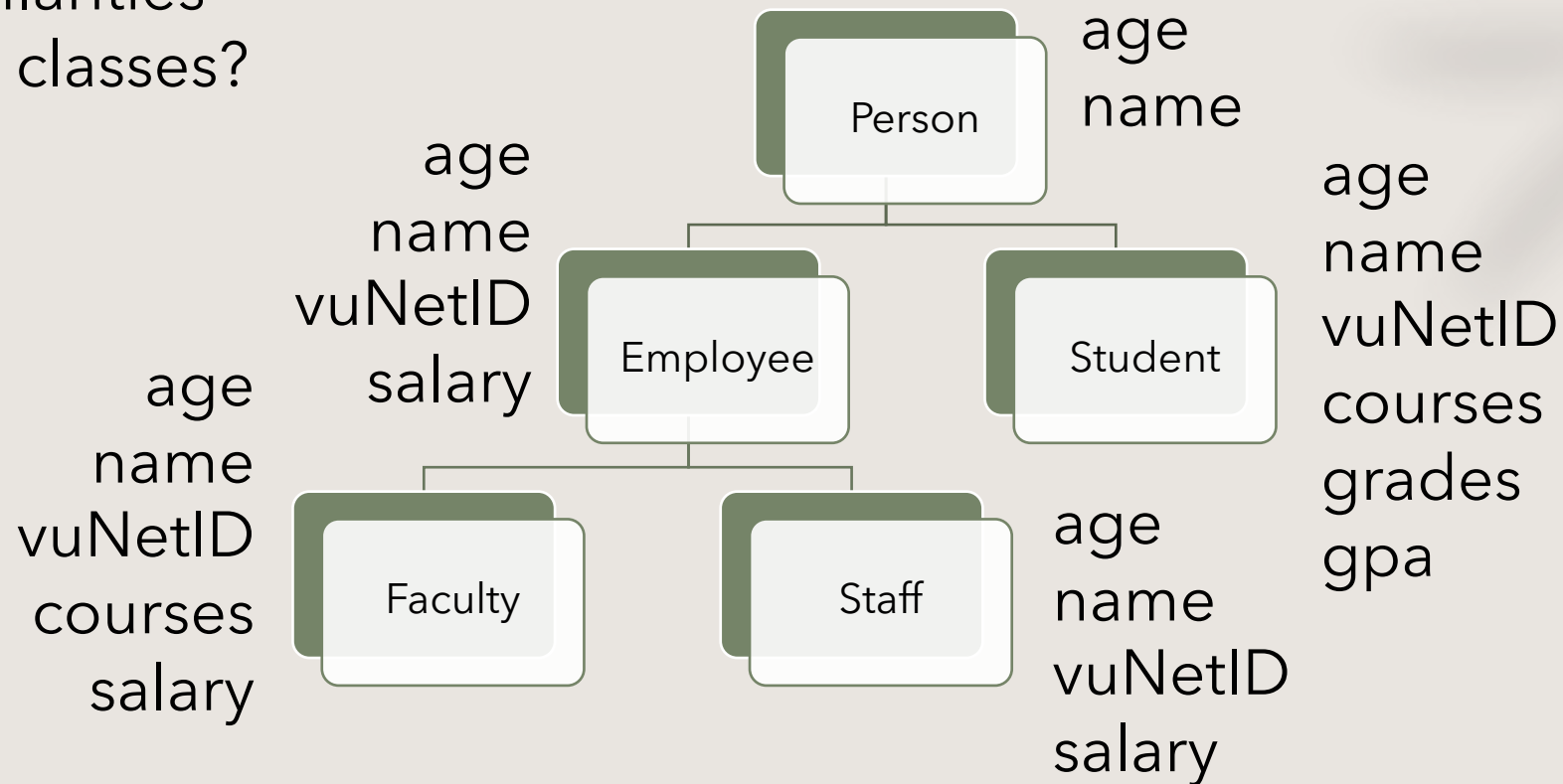
Q: Select reasonable fields for the classes.

- int age
- String name
- String vuNetID
- String[] courses
- String[] grades
- double gpa
- double salary



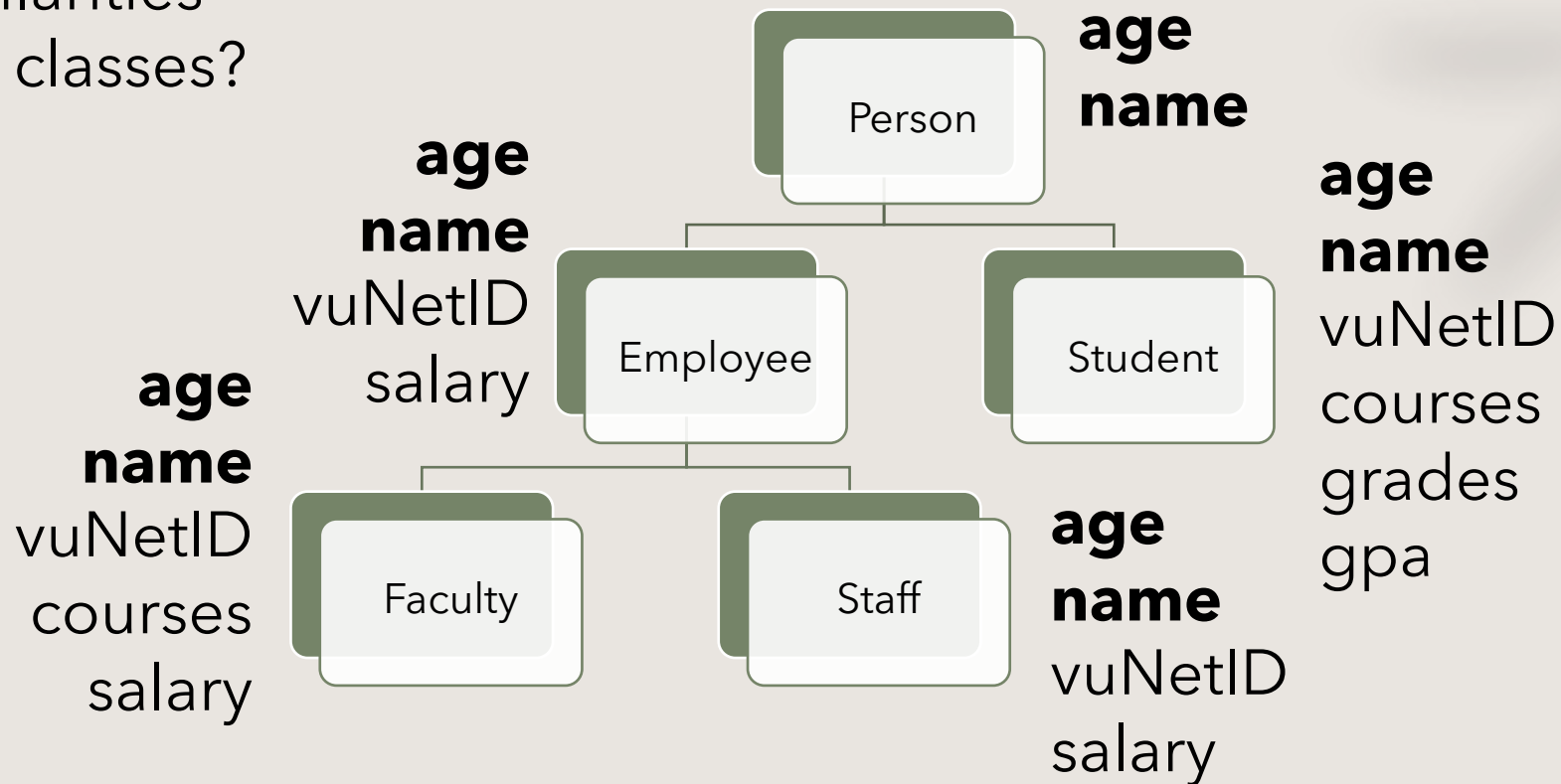
Design Class Hierarchy

Q: Any similarities among the classes?



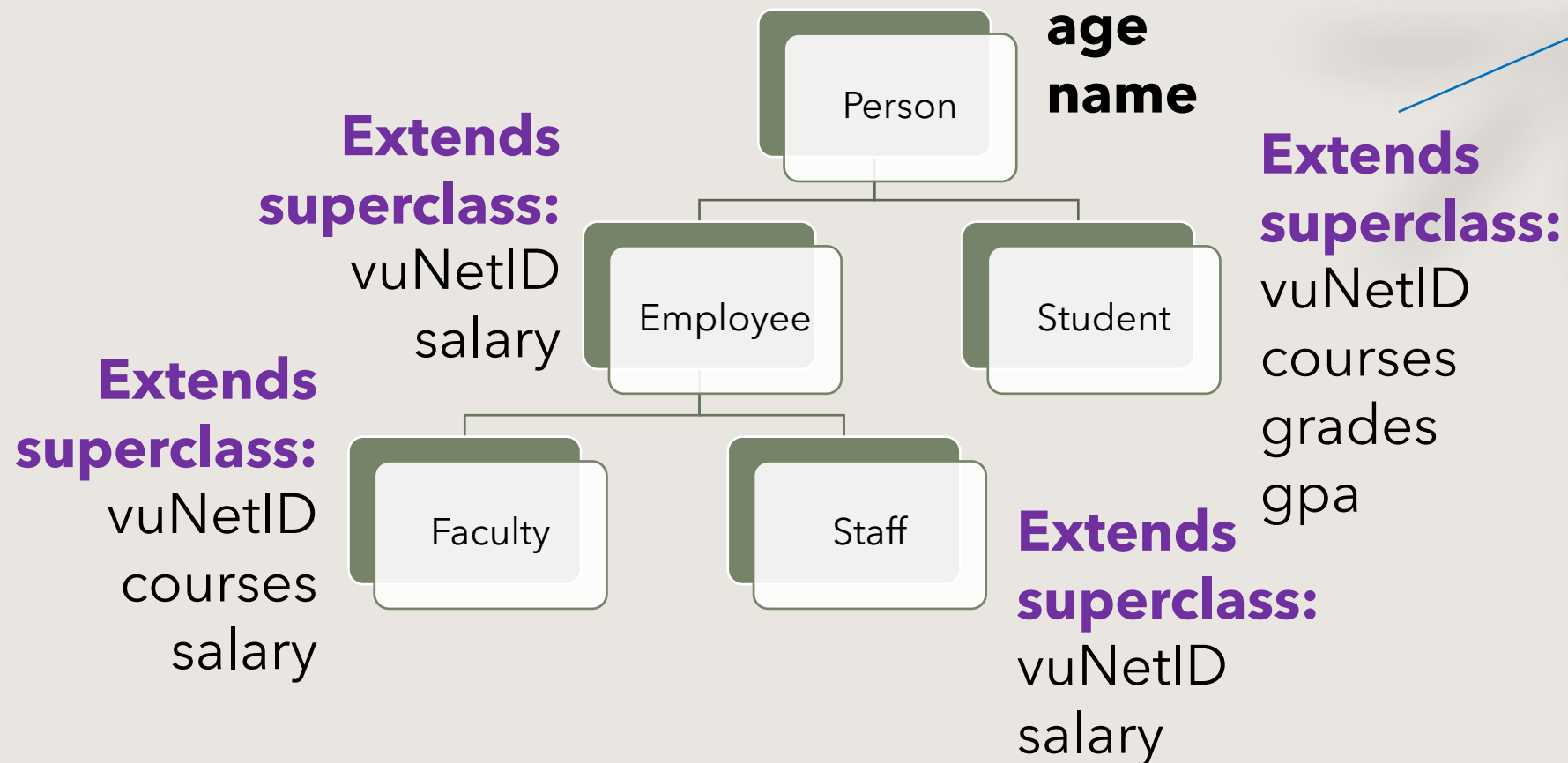
Design Class Hierarchy

Q: Any similarities among the classes?



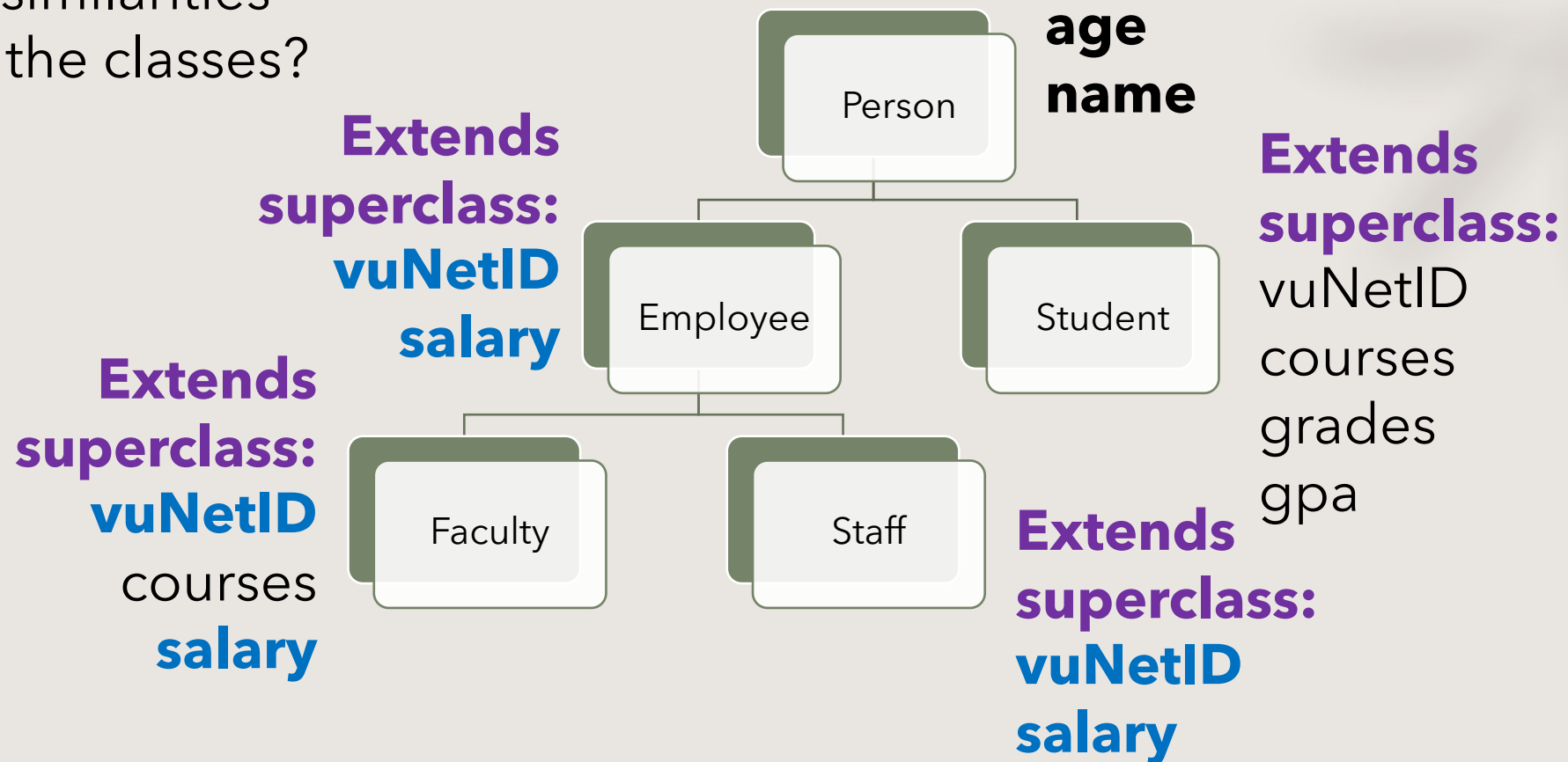
Design Class Hierarchy

- Inherits **age** and **name** from the superclass
- Extends the superclass with more fields

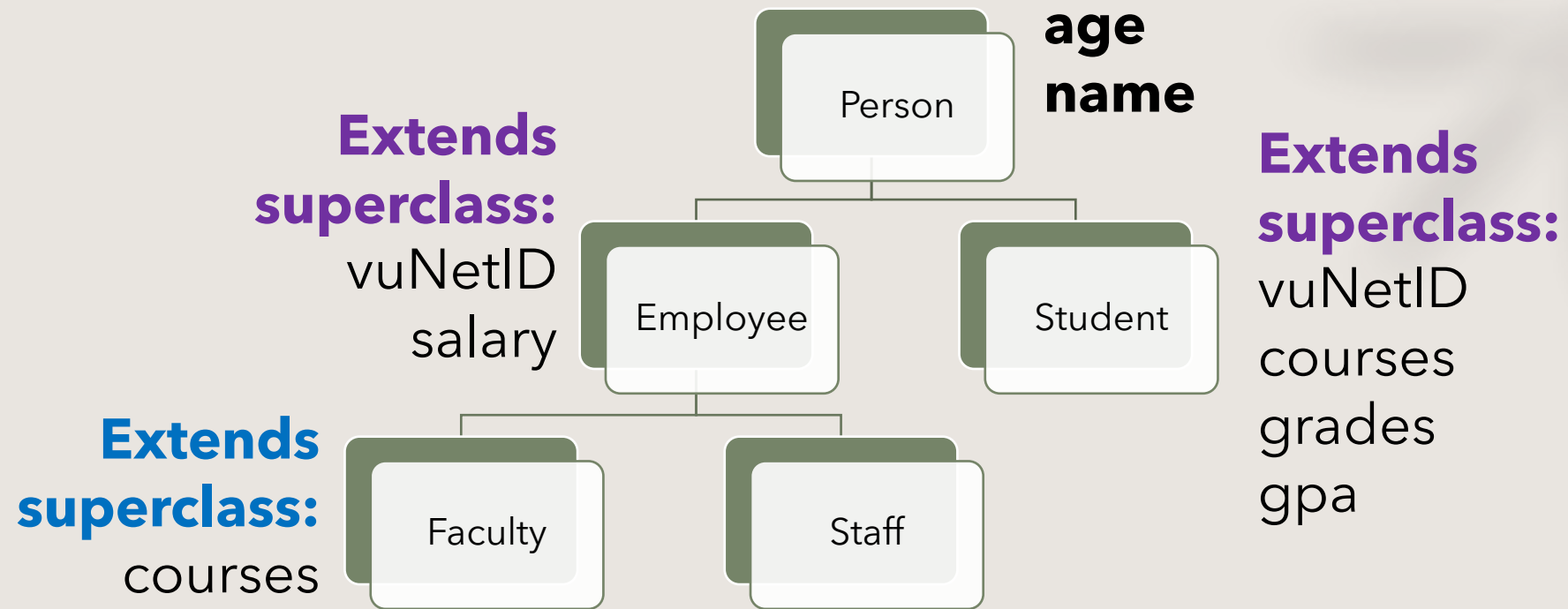


Design Class Hierarchy

Q: Any similarities among the classes?



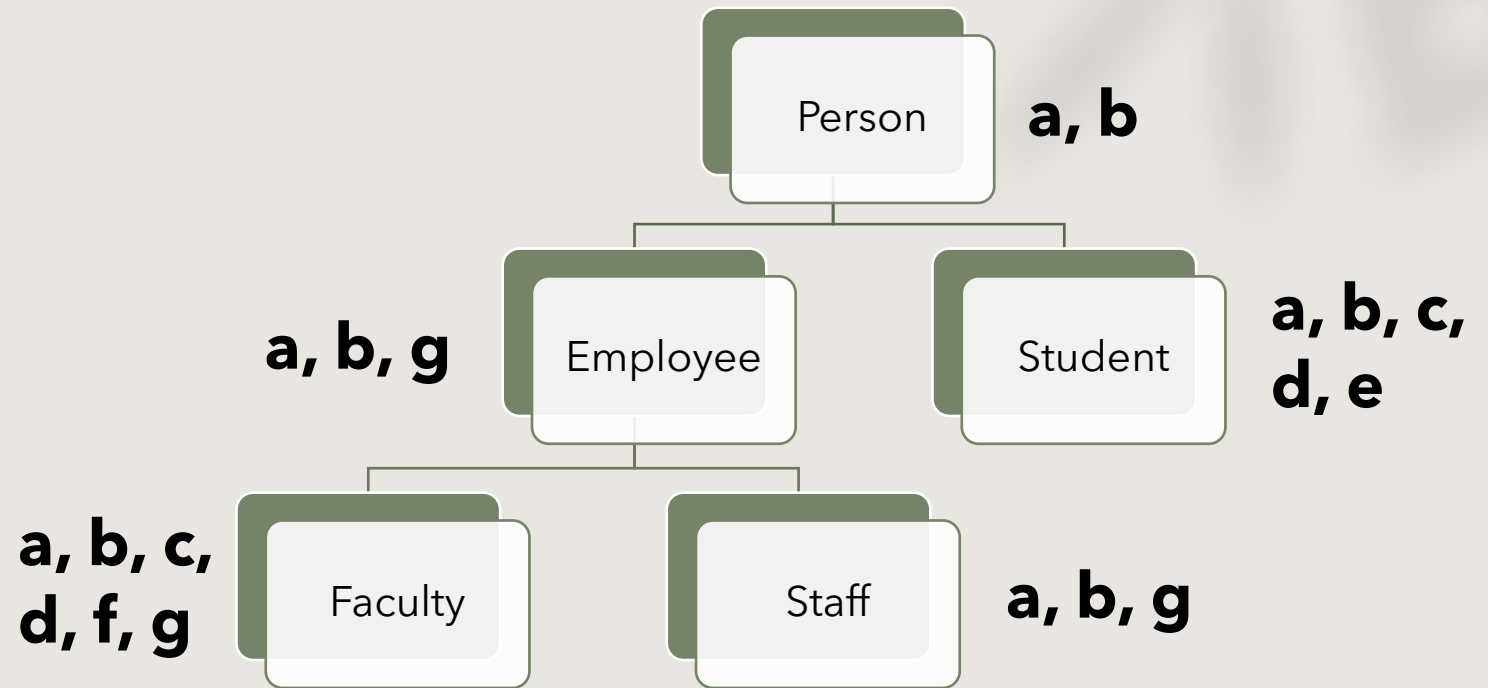
Design Class Hierarchy



Design Class Hierarchy

Q: Select reasonable methods for the classes.

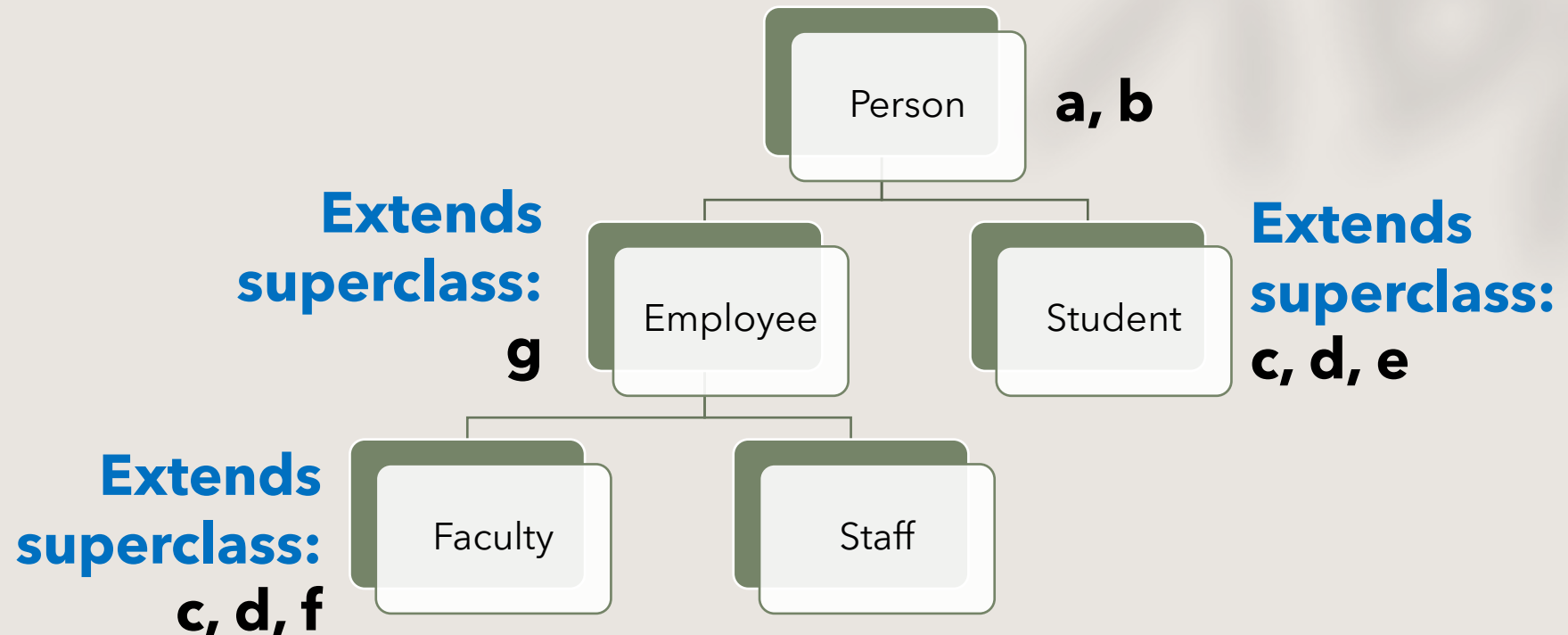
- a. toString()
- b. equals()
- c. addCourse()
- d. removeCourse()
- e. calcGPA()
- f. teachCourse()
- g. calcMonthlyPay()



Design Class Hierarchy

Q: Select reasonable methods for the classes.

- a. toString()
- b. equals()
- c. addCourse()
- d. removeCourse()
- e. calcGPA()
- f. teachCourse()
- g. calcMonthlyPay()



Subclass

- An instance of the superclass
 - **Superclass is constructed first**
- Can have new state and add new behavior
 - **Constructed/initialized after the superclass'** state and behavior
- If we don't like the superclass' behavior, we can override it
 - To implement a **new version of a method in the subclass** to replace code that would otherwise have been inherited from a superclass
 - E.g., toString() and equals() methods
 - Method name and signature **MUST** match exactly

Subclass and Keyword super

- An instance of the superclass
 - **Superclass is constructed first** via **super(<param>)**, which allows a derived class to call its superclass' constructor

Recap:

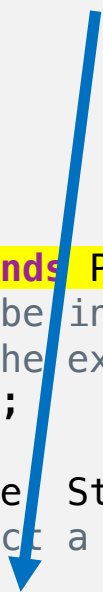
this(<param>) is used to call one constructor from another in the same Class

```
// In Person.java
public class Person {
    public int age;
    public String name;

    public Person(int age, String name) {
        this.age = age;
        this.name = name;
    }
}

// In Student.java
public class Student extends Person {
    // age and name will be inherited from Person
    // Hence, only list the extended field(s)
    public String vuNetID;

    public Student(int age, String name, String vuNetID) {
        // First, construct a Person object
        super(age, name);
        // Next, initialize the extended field
        this.vuNetID = vuNetID;
    }
}
```



Subclass

- Can have new state and add new behavior
 - Constructed/initialized **after** the superclass' state and behavior

```
// In Person.java
public class Person {
    public int age;
    public String name;

    public Person(int age, String name) {
        this.age = age;
        this.name = name;
    }
}

// In Student.java
public class Student extends Person {
    // age and name will be inherited from Person
    // Hence, only list the extended field(s)
    public String vuNetID;

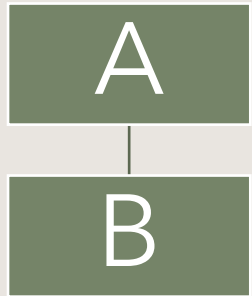
    public Student(int age, String name, String vuNetID) {
        // First, construct a Person object
        super(age, name);
        // Next, initialize the extended field
        this.vuNetID = vuNetID;
    }
}
```

Live Demo

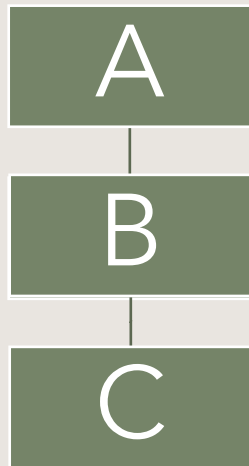


Types of Inheritance

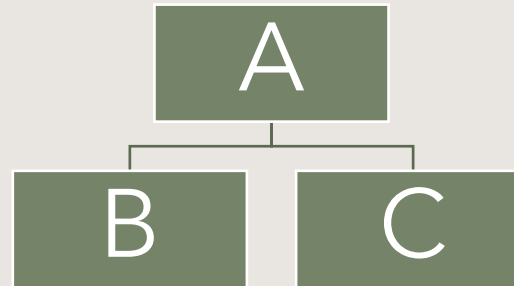
1) Single-level



2) Multi-level

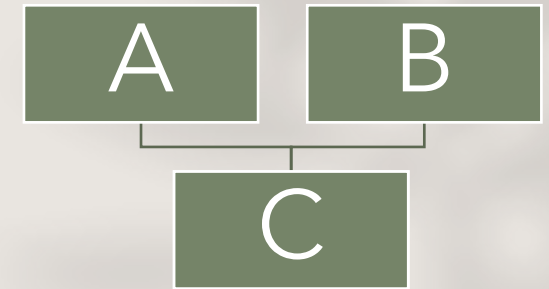


3) Hierarchical

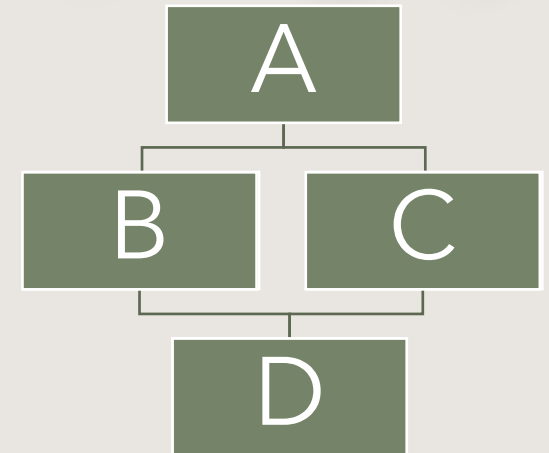


Supported through Class

4) Multiple



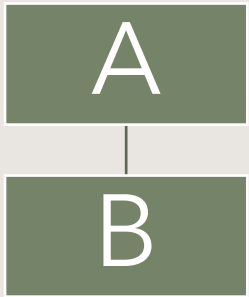
5) Hybrid



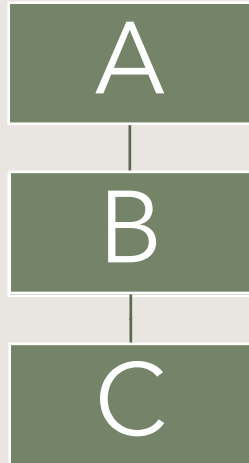
Supported through Interface
(zyBook Chap 16, optional)

Types of Inheritance

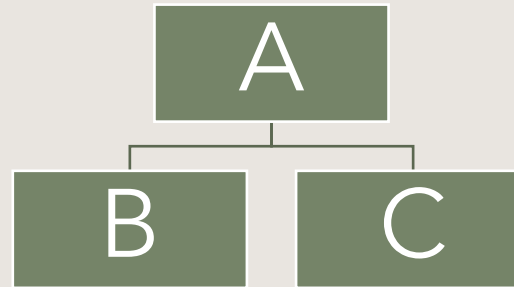
1) Single-level



2) Multi-level



3) Hierarchical

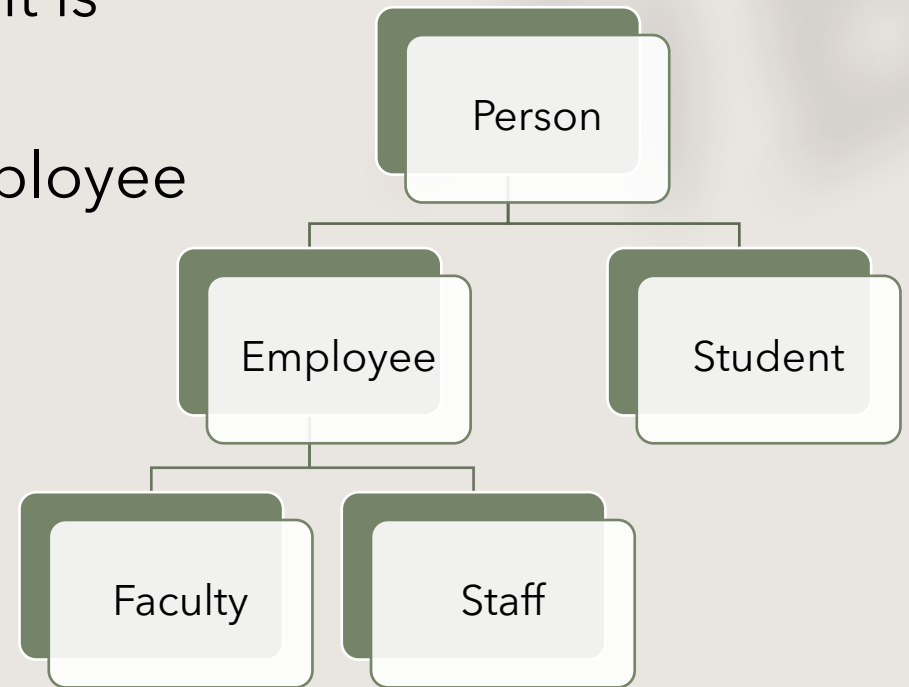


- A class can have ONLY ONE superclass
- A superclass may have many subclasses

Is-A Relationships

Is-A → The idea of **inheritance**

- Whenever one class **inherits** another class, it is called an Is-A relationship
 - A Student is a Person, a Faculty is an Employee
- **Unidirectional**
 - A Student is a Person, but not all Persons are Students



Has-A vs. Is-A Relationships

Has-A → The idea of **composition**

- Whenever an instance of one class is **used** in another class, it is called Has-A relationship
 - E.g., a Person has a String field, such as name.

Using Inheritance

- When should you use inheritance?
 - When you see **similarities between classes** that can be modeled in a hierarchy