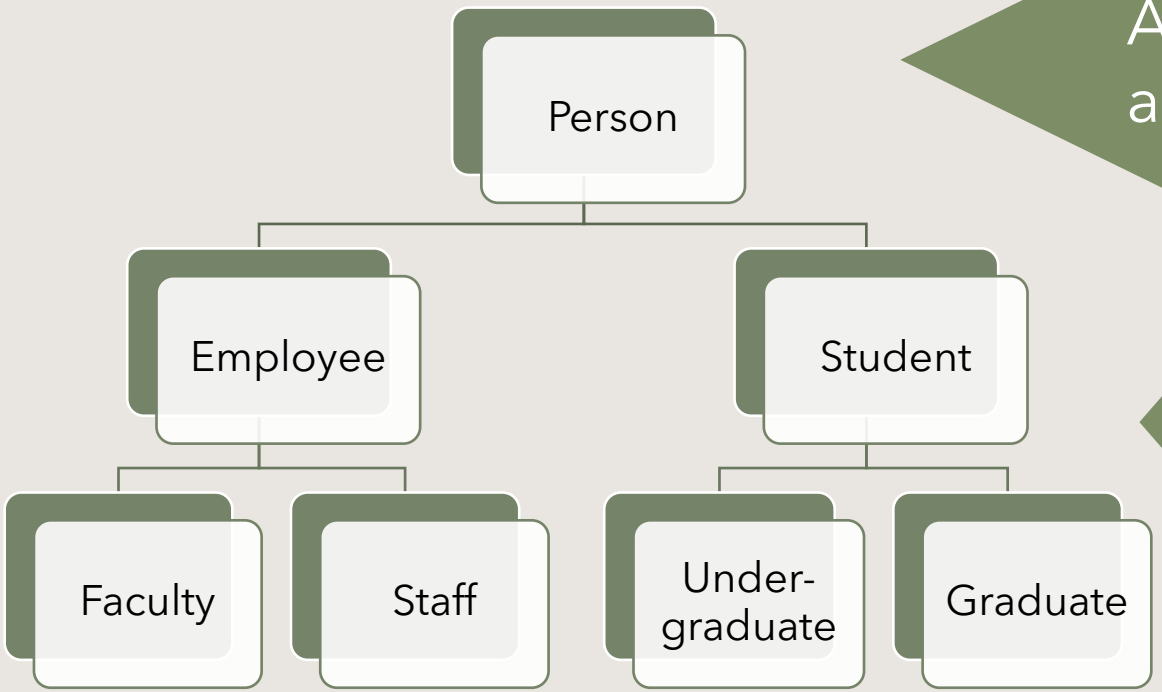
Inheritance Basics

zyBook Chap 10

Inheritance – Real World Example



A newborn baby can be considered as a newly constructed Person

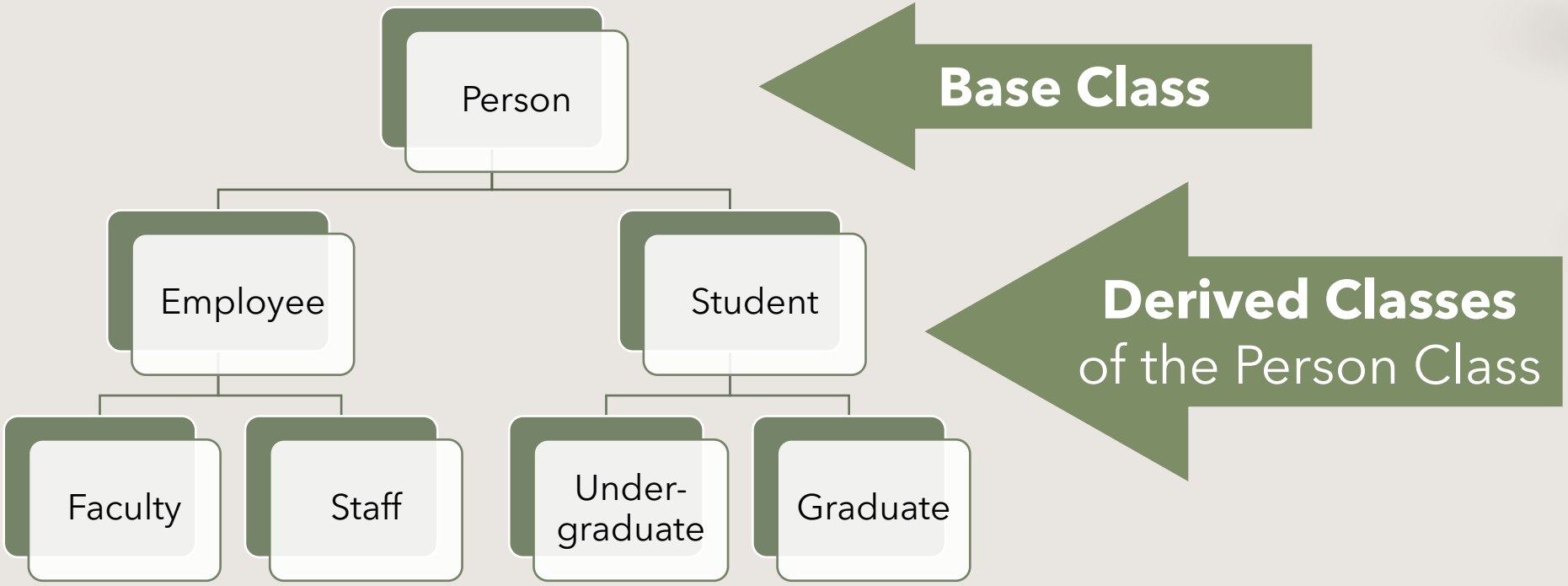
After gaining different skills (state and/or behavior), the newborn baby may become a student, or a faculty, or…

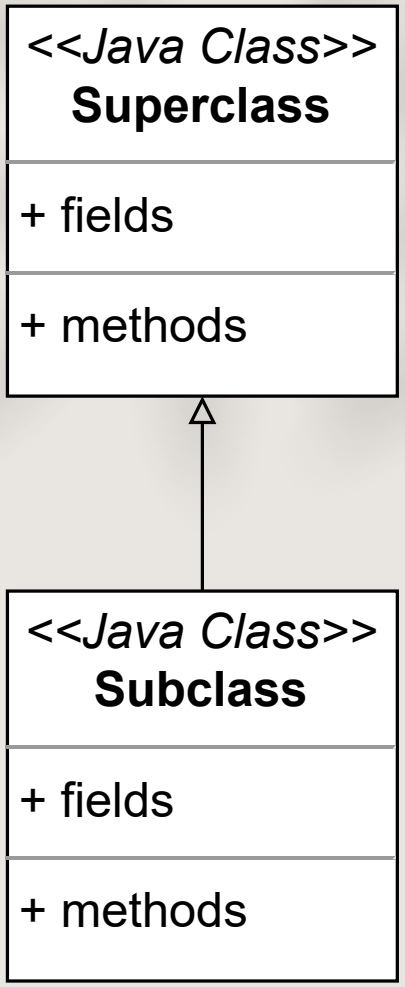
Inheritance

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





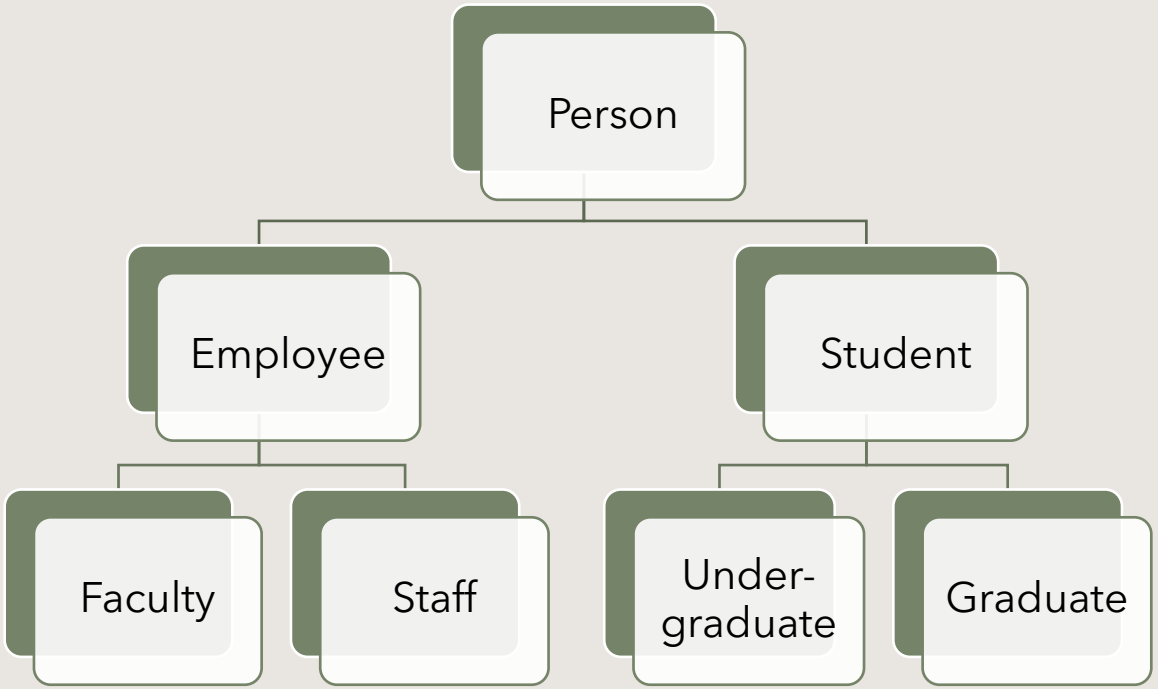
Terminologies

- **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

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

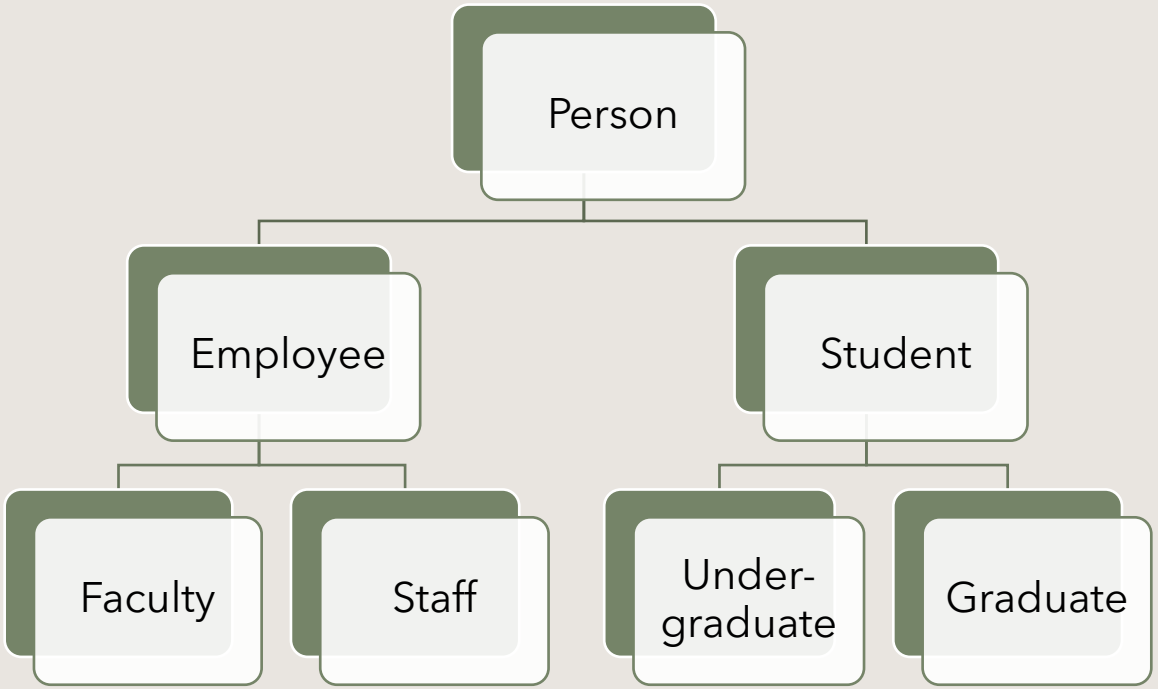
→ A Java keyword
→ The idea of "increases the functionality"

Syntax – Example




```java
// 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

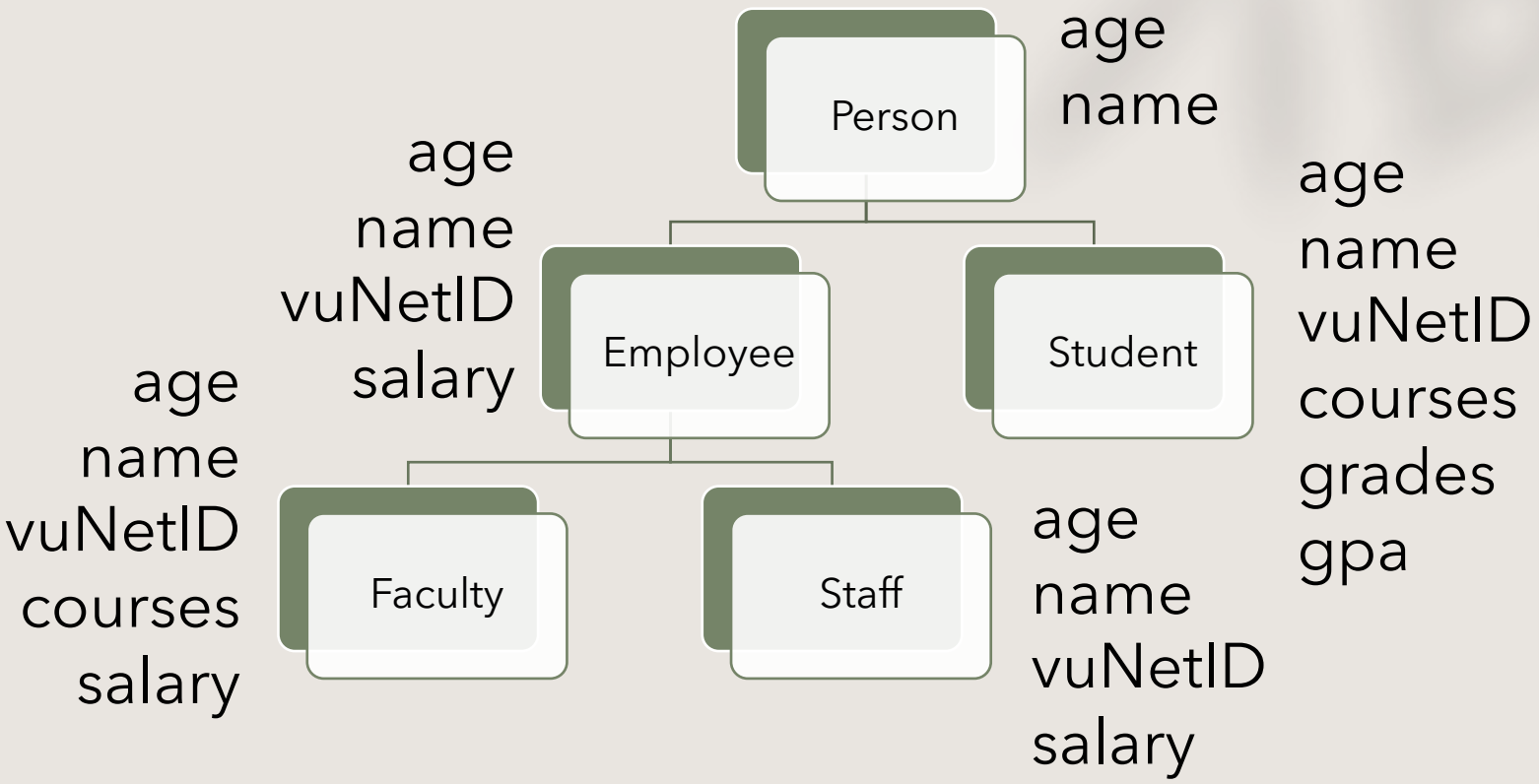



```java
// Person.java
public class Person {
    /* ...Code Here... */
}

// 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

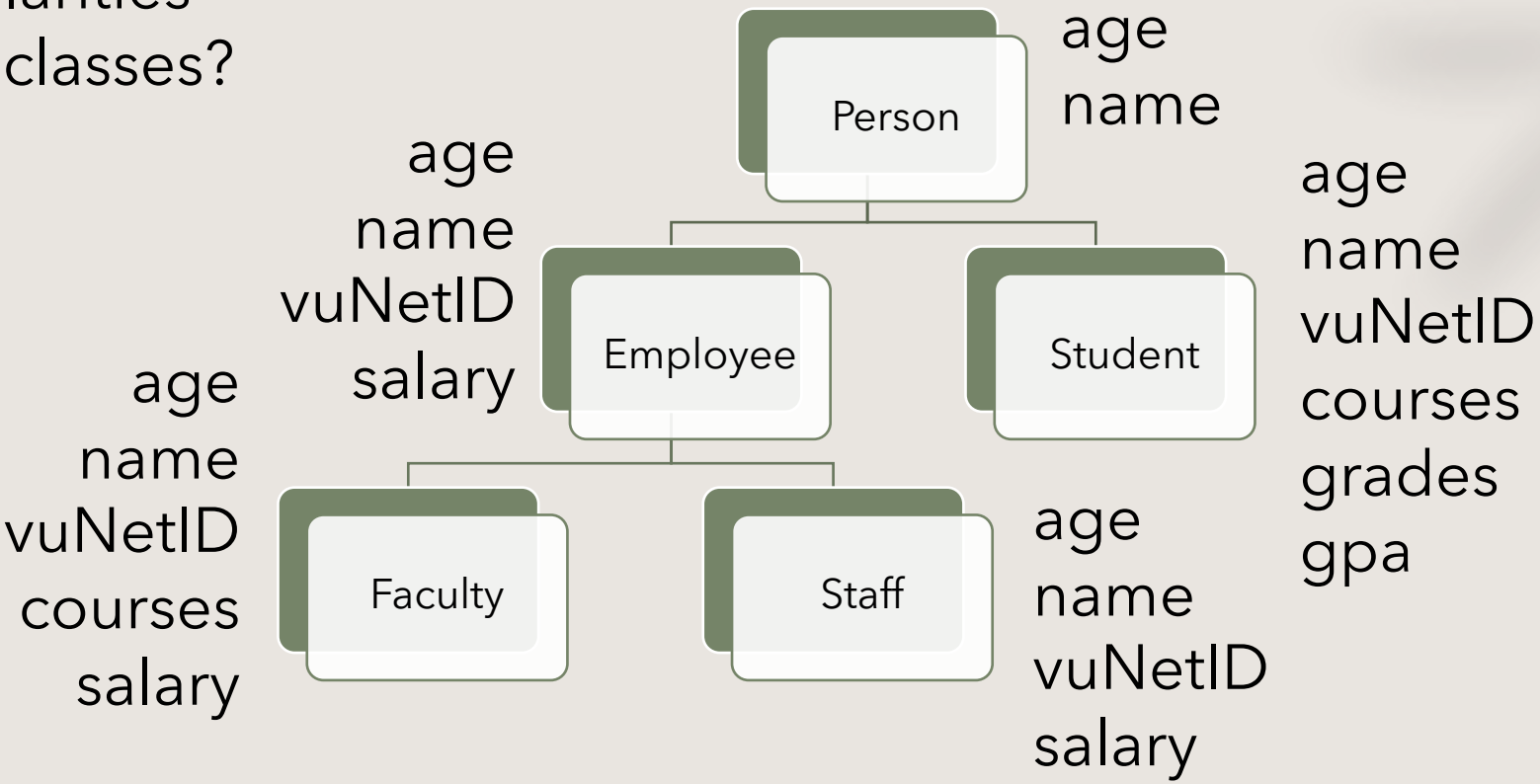
Q: Select reasonable fields for the classes.

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



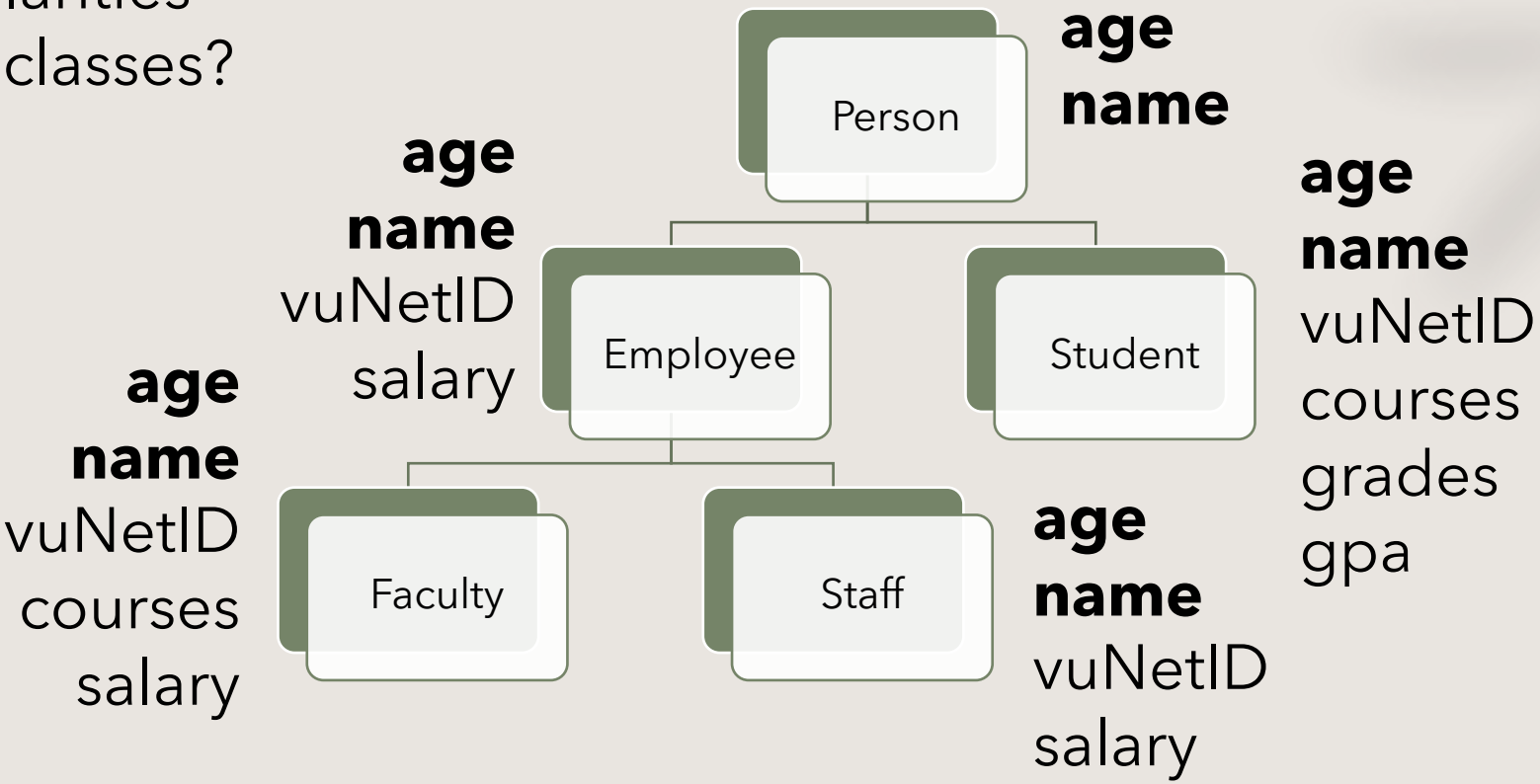
Design Class Hierarchy

Q: Any similarities among the classes?



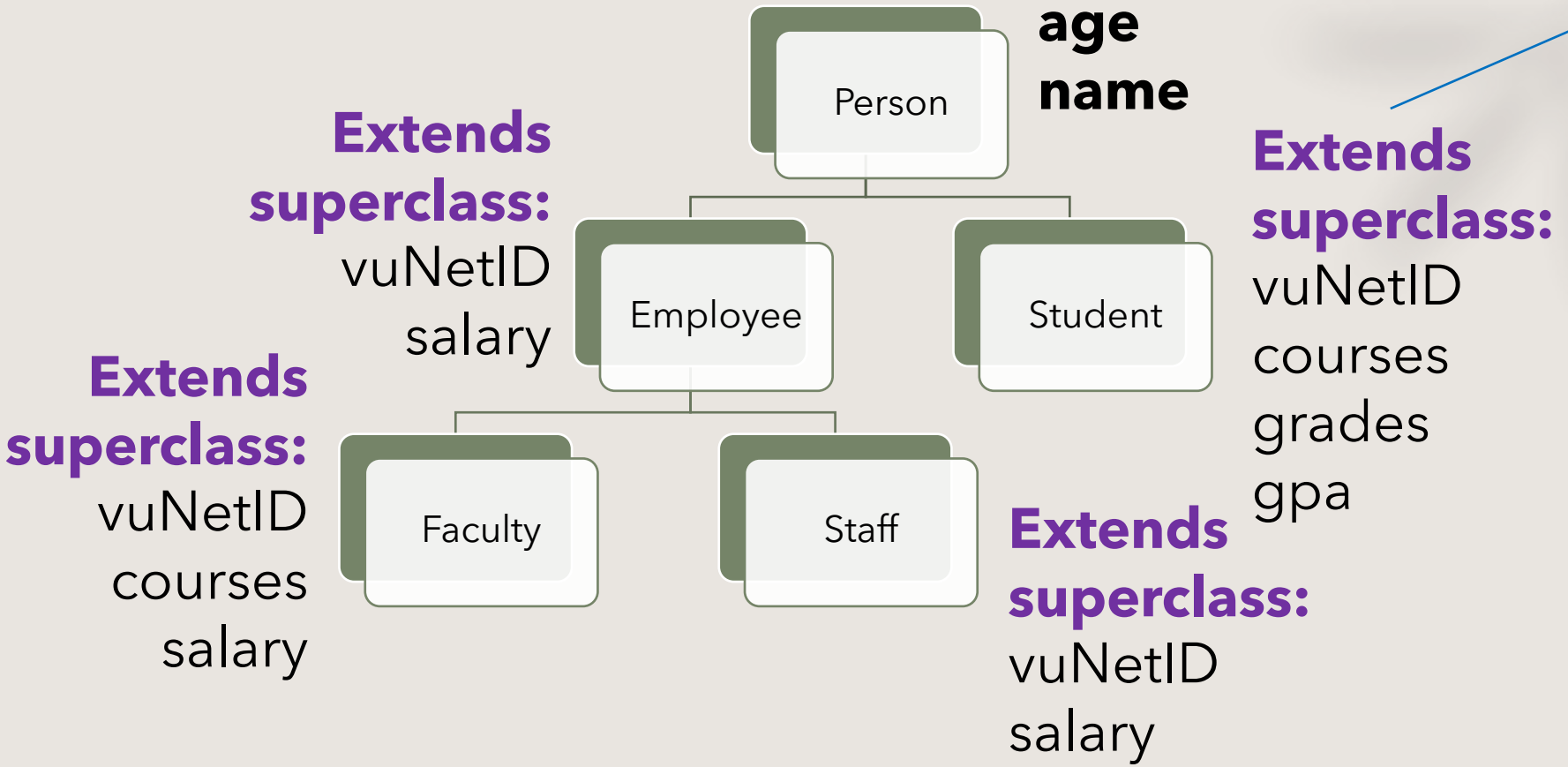
Design Class Hierarchy

Q: Any similarities among the classes?



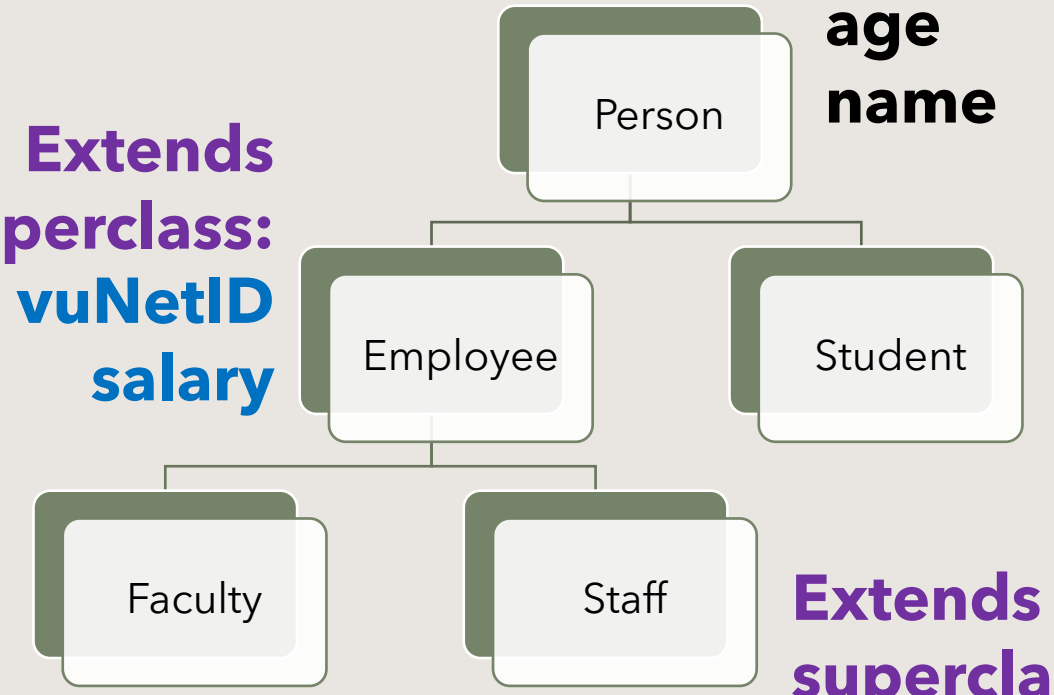
Design Class Hierarchy

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



Design Class Hierarchy

Q: Any similarities among the classes?



age
name

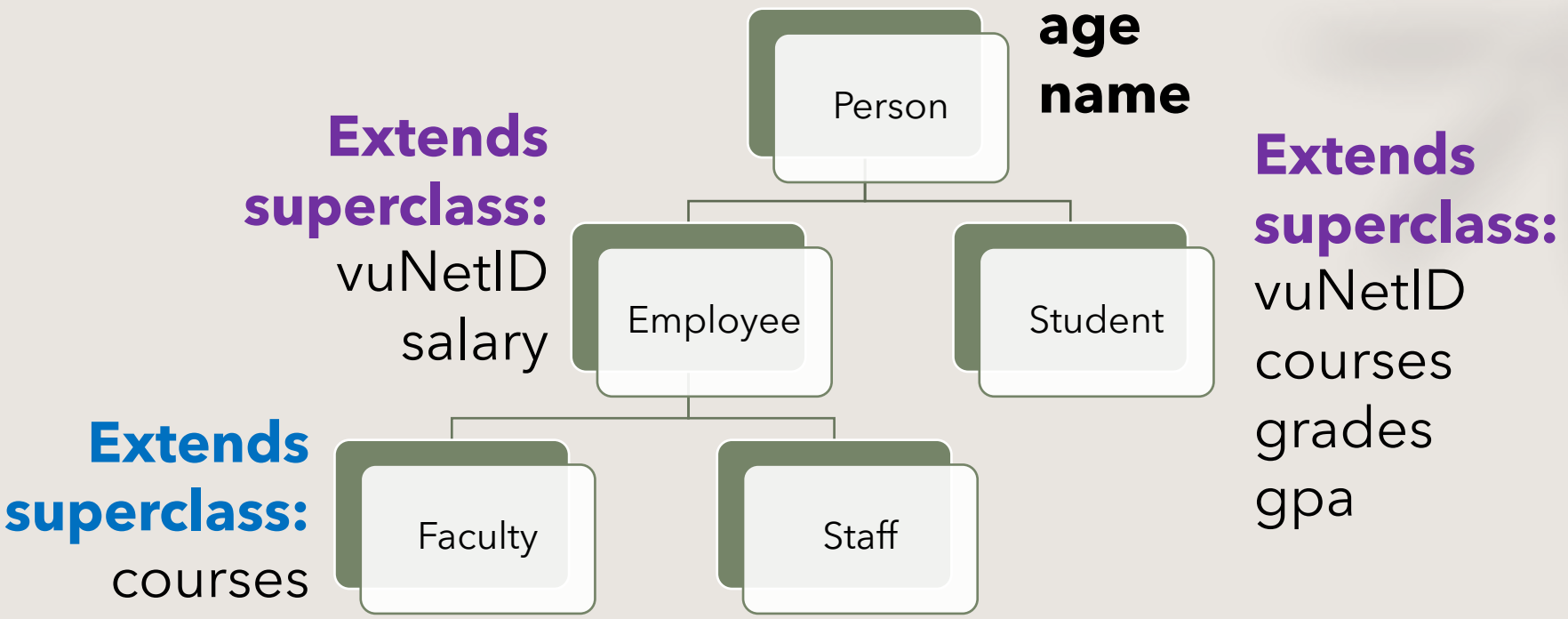
Extends superclass:
vuNetID
courses
grades
gpa

Extends superclass:
vuNetID
salary

Extends superclass:
vuNetID
courses
salary

Extends superclass:
vuNetID
salary

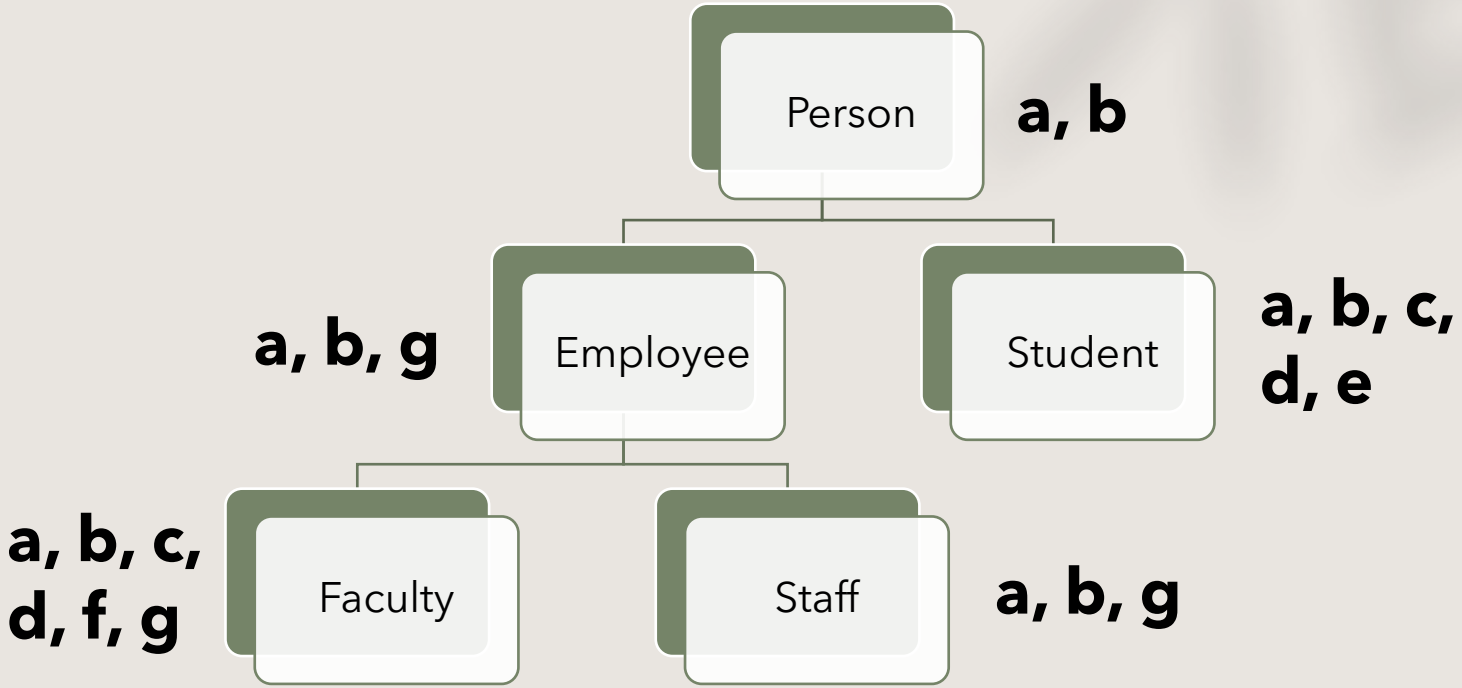
Design Class Hierarchy



Design Class Hierarchy

Q: Select reasonable methods for the classes.

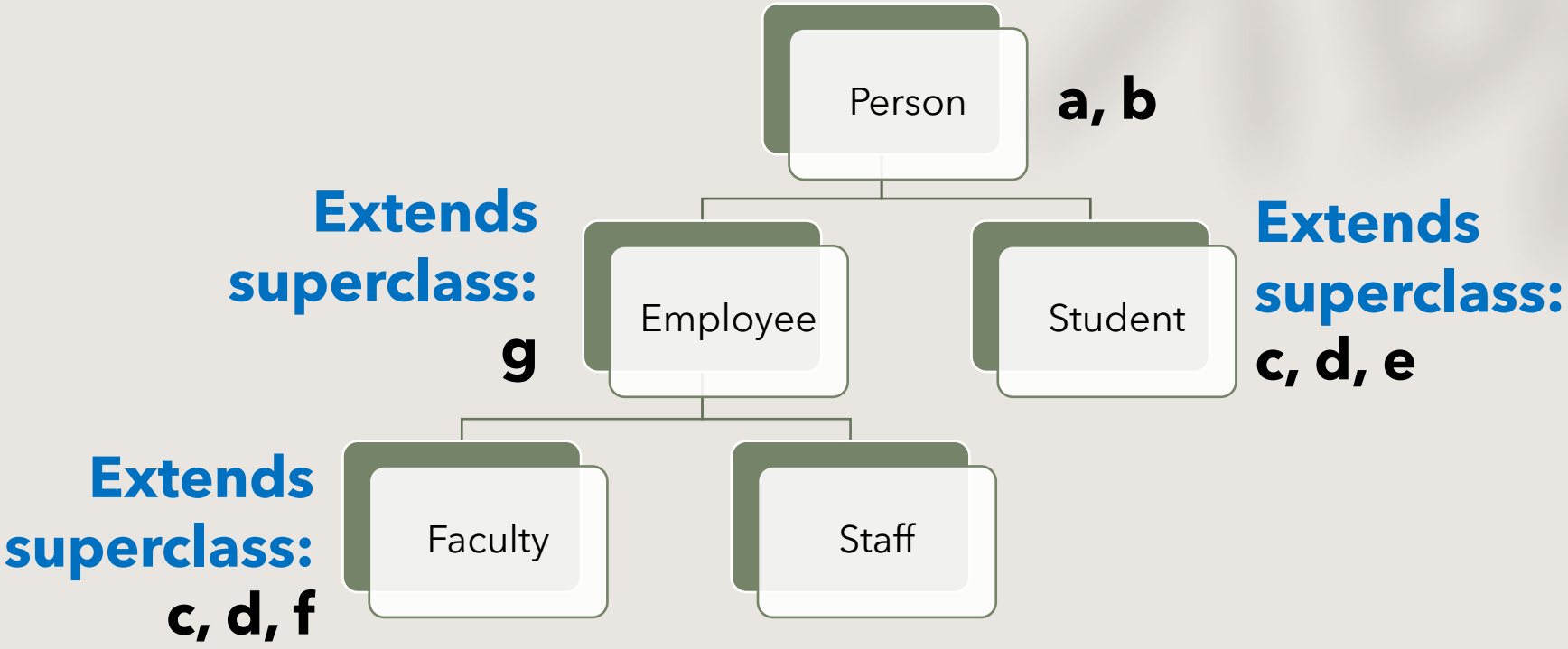
a. toString()
b. equals()
c. addCourse()
d. removeCourse()
e. calcGPA()
f. teachCourse()
g. calcMonthlyPay()



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

```java
// 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;
    }
}
```

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

Subclass

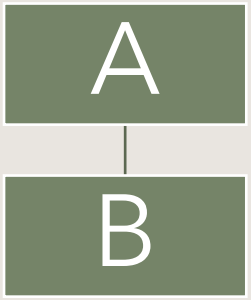
- Can have new state and add new behavior
  - Constructed/initialized **after** the superclass' state and behavior

```java
// 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;
    }
}
```

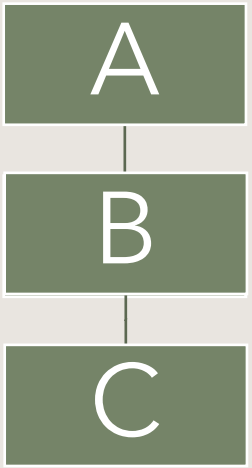
Live Demo

Types of Inheritance

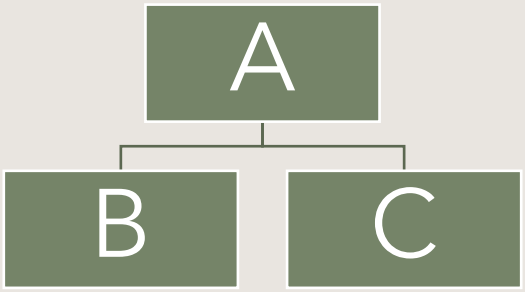
1) Single-level



2) Multi-level

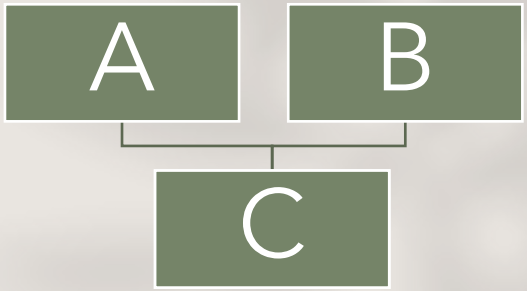


3) Hierarchical

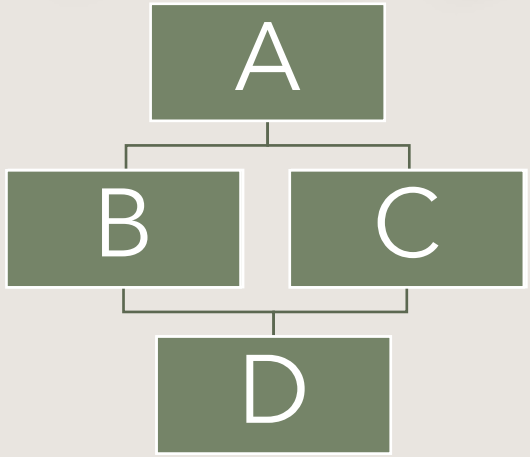


Supported through Class

4) Multiple



5) Hybrid

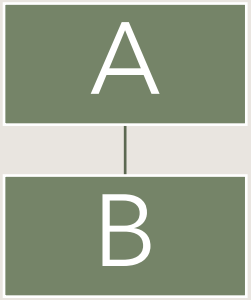


Supported through Interface

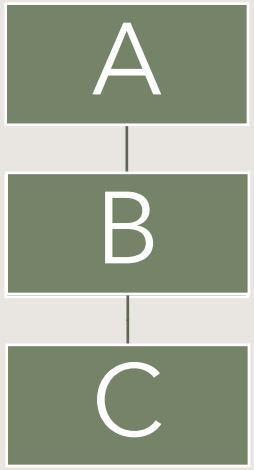
(zyBook Chap 16, optional)

Types of Inheritance

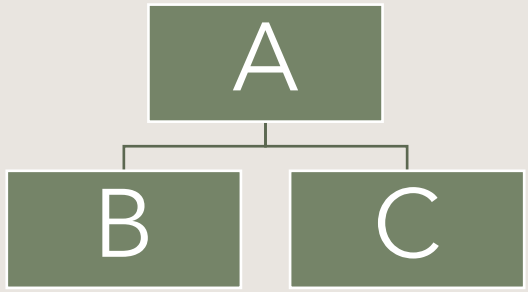
1) Single-level



2) Multi-level



3) Hierarchical

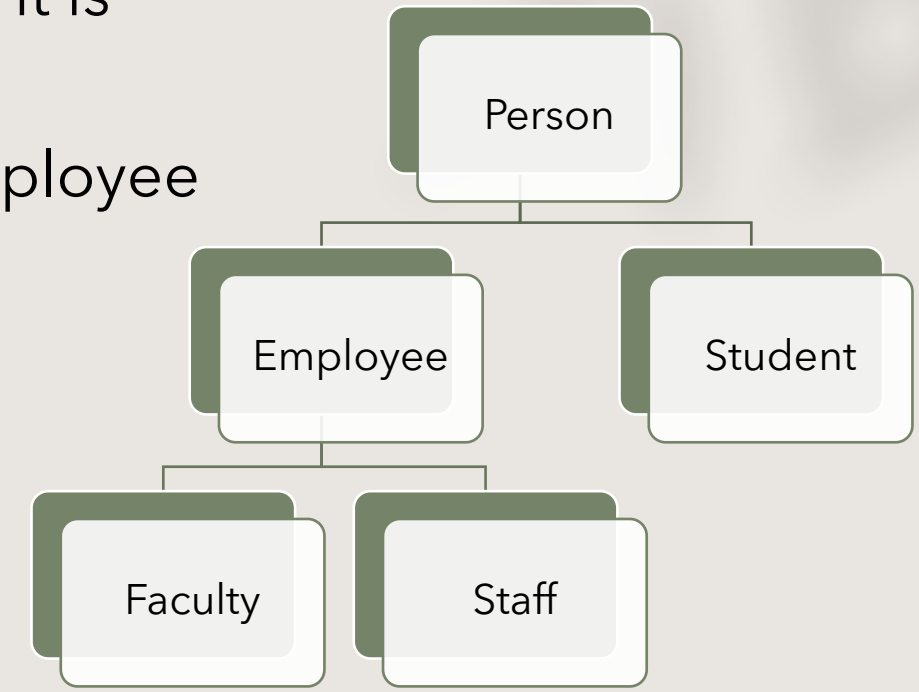


- A class can have ONLY ONE superclass
- A superclass may have many subclasses

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