COMPSCI 121: INHERITANCE Part 1.

SPRING 2020

WHAT YOU ALREADY KNOW!

Word Cloud generated from from zyBooks chapter headings.

Now:

INHERITANCE

SCENARIO 1: CAR DEALERSHIP

A car dealership maintains an inventory of cars and trucks. Let's model this in software with the Car and Truck classes.

Attributes of these classes:

Car: vehicle ID (VIN), engine type, color, num doors, weight, num passengers and luggage capacity.

Truck: vehicle ID (VIN), engine type, color, num doors, weight, hauling capacity, pickup/flatbed.

SCENARIO: CAR DEALERSHIP

Notice the attributes in common:

Car: vehicle ID (VIN), engine type, color, num doors, weight, num passengers and luggage capacity.

Truck: vehicle ID (VIN), engine type, color, num doors, weight, hauling capacity, pickup/flatbed.

Good design says we try not to duplicate code whenever possible. This cuts down on errors, simplifies the code.

SCENARIO: CAR DEALERSHIP

Use inheritance to *refactor* the common attributes out of the Car and Truck classes. Put them into a parent class called Vehicle.

Vehicle: vehicle ID (VIN), engine type, color, num doors, weight.

Car (extends Vehicle): num passengers and luggage capacity.

Truck(extends Vehicle): hauling capacity, pickup/flatbed.

SCENARIO: CAR DEALERSHIP

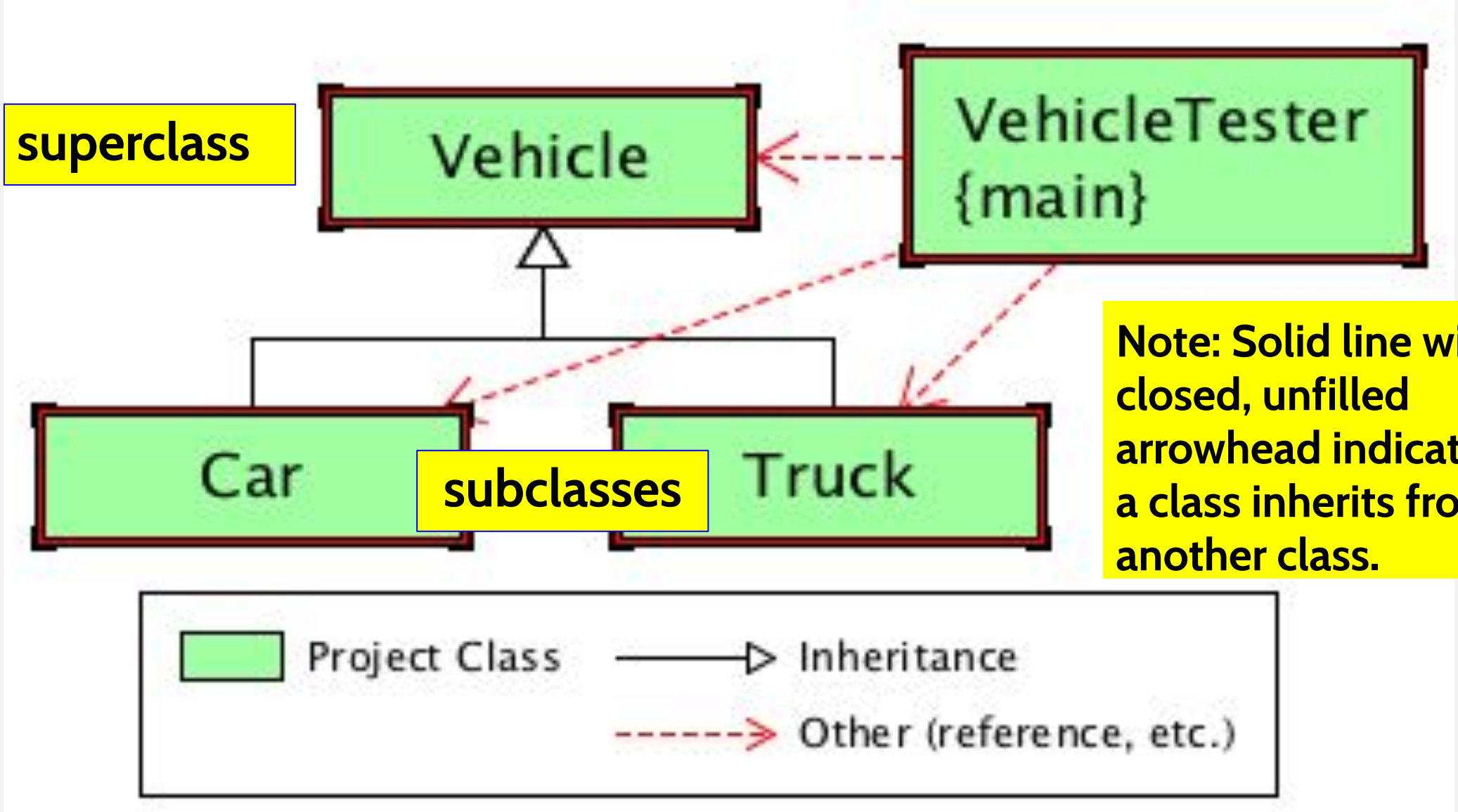
Vehicle: vehicle ID (VIN), engine type, color, num doors, weight.

Car (extends Vehicle): num passengers and luggage capacity.

Truck(extends Vehicle): hauling capacity, pickup/flatbed.

- The Car and Truck classes are subclasses of Vehicle.
- Therefore, they both *are* Vehicles, and share Vehicle attributes, but have their own special attributes.
- Car and Truck *specialize*, or **extend** the Vehicle class.

INHERITANCE EXAMPLE 1



Note: Solid line with closed, unfilled arrowhead indicates a class inherits from another class.

INHERITANCE TERMS

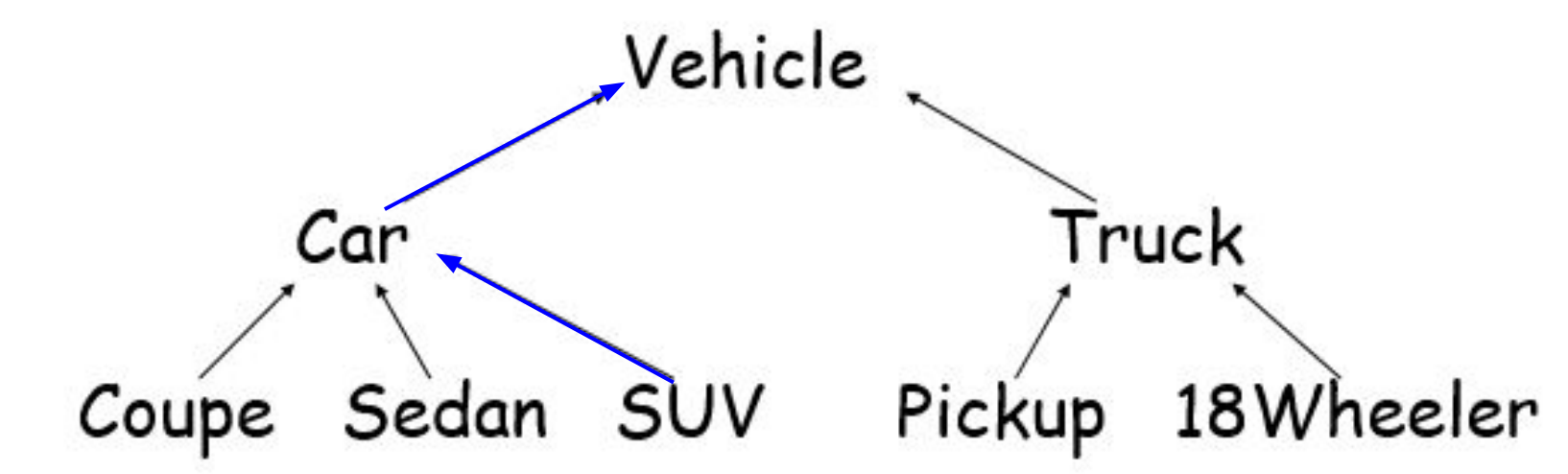
These are terms used to talk about inheritance:

- superclass, also called a base class.
- subclass, also called a derived class.
- About the relationship between superclass and subclss we can say:
  - "A subclass inherits from a superclass".
  - "A subclass is derived from a superclass".
  - "A subclass extends a superclass".

USE CASES FOR INHERITANCE

1. You have several classes that **share** many common attributes and/or methods.
   - you can refactor the **common code** into a superclass.
2. You are processing **many** different but related Objects in the same manner- e.g. toString().
3. You have several classes that perform similar tasks.
   - put the shared part of the task in a **superclass** and the **subclasses** specialize. E.g. reading different file types. They all locate and open the file.

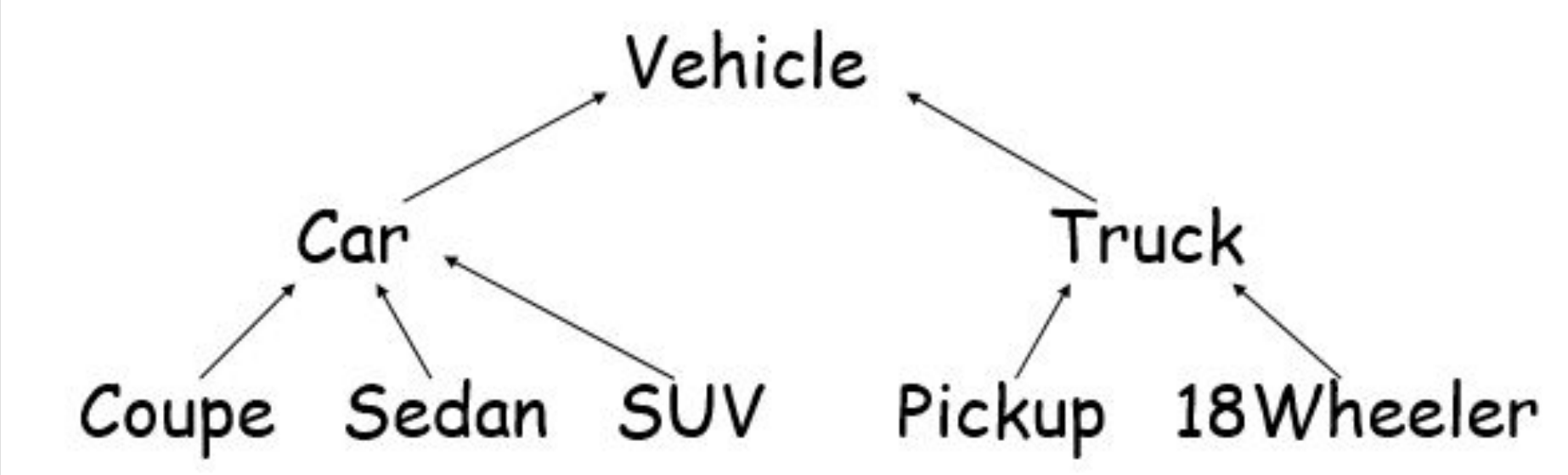
EXAMPLE 1 EXTENDED FURTHER



A class can `extend` another class that itself `extends` another class: SUV extends Car, which extends Vehicle.

We call inheritance an "is-a" relationship.
Notice the is-a nature of classes: An SUV is a Car, a Car is a Vehicle. But, a Vehicle *is not* a Car (note: ***works in one direction only!***)

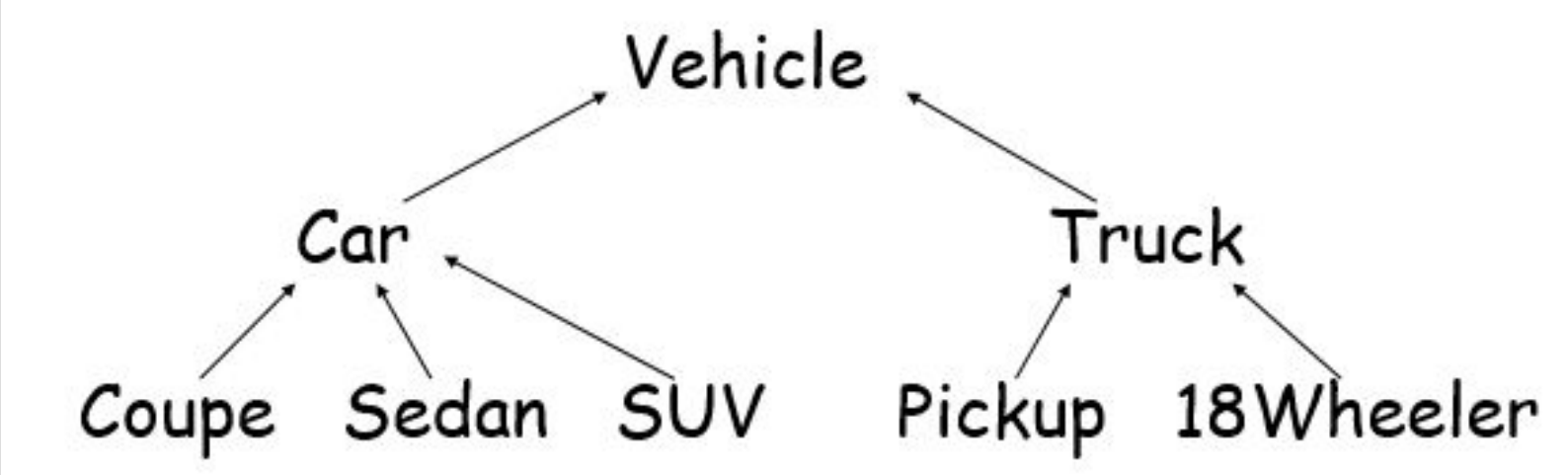
Clicker Question 1



Which of the following is *correct*?

A. class Car extends Vehicle.
B. class Sedan extends Vehicle
C. class Pickup extends 18Wheeler
D. class 18Wheeler extends Car

Clicker Question 1 Answer



Cannot inherit from 2 classes!

A. `class Car extends Vehicle`
B. `class Sedan extends Car and Vehicle`
C. `class Pickup extends 18Wheeler`
D. `class 18Wheeler extends Car`

SCENARIO 2

Create a program to store and use data for UMass personnel.

Q. What classes should we define?

Q. What attributes do we need?

Three types of personnel: **Employee, Student, Faculty.**

For each we store: **first, last names, address, email, phone.**

In addition we need:

for the Employee: the hourly rate.

for the Student: the GPA and major.

for the Faculty: the department.

Lecture code: PersonProject

DESIGN IDEA 1

Say we create **Employee, Student, Faculty** classes.
For all 3 classes we need to define the common attributes:
first, last names, address, email, phone, and we have to write `get`, `set` methods for each attribute in all 3 classes.

Is there a problem?

- Code should be defined in one place whenever possible.
- This avoids errors and makes adding new specialized classes easier.
- Also simplifies the code.

Same problem as in the Car Dealership scenario.

DESIGN IDEA 2

What if we create one class to represent all personnel- called Person?

What attributes do we need?

first name, last name, address, email, phone, hourly rate, GPA, major, department.

The problem?

For Employee: GPA `0.0`, major & department `null`

For Student: hourly rate `0.0` and department `null`

For Faculty: hourly rate and GPA `0.0`

Bad design: avoid null or undefined values.

DESIGN IDEA 3 WITH INHERITANCE

Create a superclass **Person** that contains the common attributes: first, last names, address, email, phone.

Then, create subclasses of Person and their attributes:
for the Employee (extends Person): the hourly rate.
for the Student (extends Person): the GPA and major.
for the Faculty (extends Person): the department.

-See the lecture code.

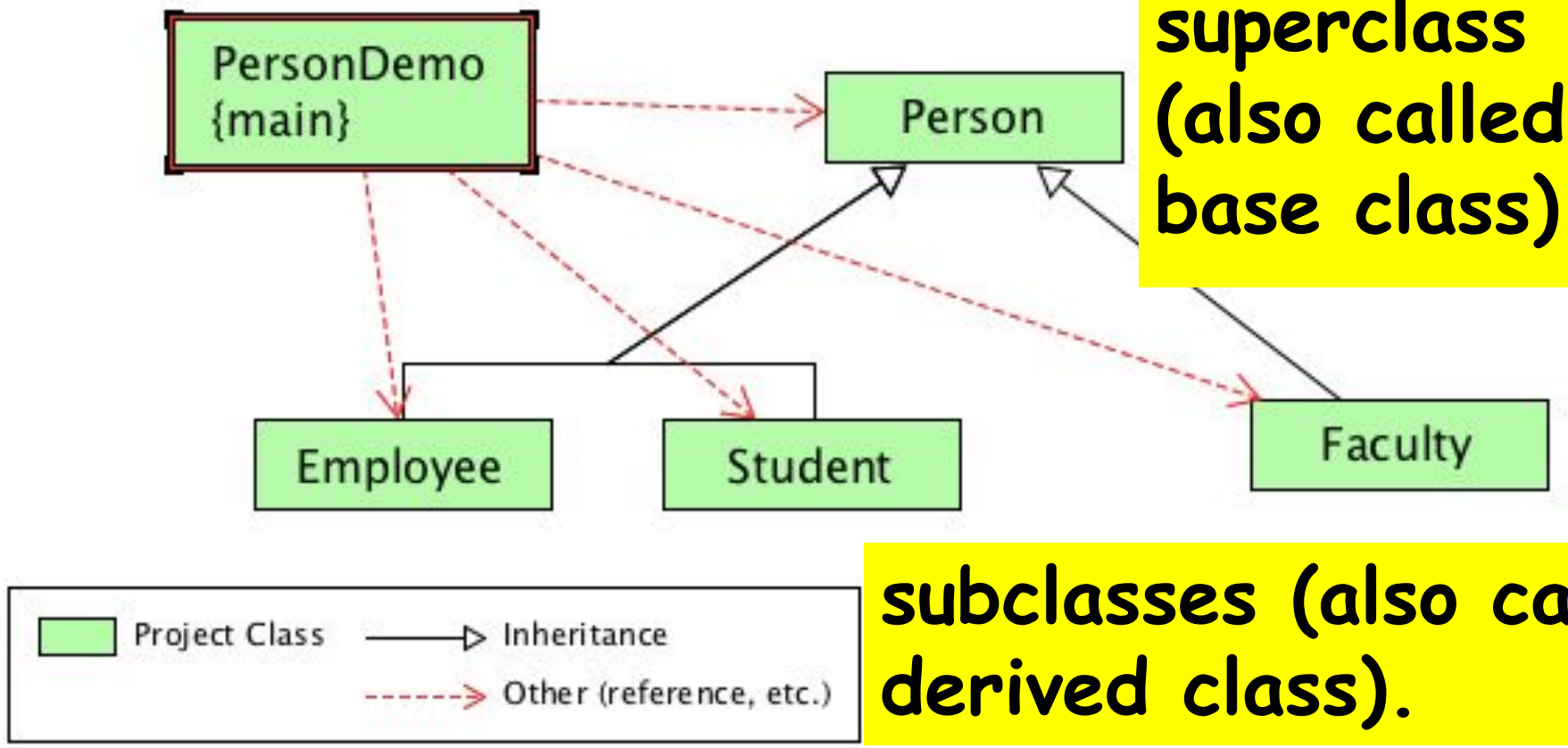
SOLUTION

With inheritance, we can **reuse the fields and methods** of the **superclass** **<u>without having to write and debug</u>** them in several places.

Notice the ***is-a*** nature of these:

- A Student ***is a*** Person
- A Faculty ***is a*** Person
- An Employee ***is a*** Person

EXAMPLE - UML diagram



superclass (also called base class).

subclasses (also called derived class).

The superclass is most general;
The subclasses are specialized (or "extended") versions of the superclass.

EXAMPLE OF SUPERCLASS CODE

```
1  public class Person{
2    private String fName;
3    private String lName;
4    private String address;
5    private String email;
6    private String phone;
7
8    public Person(String fName, String lName,String address,
9                  String email, String phone){
10     this.fName = fName;
11     this.lName = lName;
12     this.address = address;
13     this.email = email;
14     this.phone = phone;
15   }
16   public String getFirstName(){return fName;}
17   public String getLastName(){return lName;}
18   public String getAddress(){return address;}
19   public String getEmail(){return email;}
20   public String getPhone(){return phone;}
21   public String toString(){return fName+", "+lName
22                  +", "+address+", "+email+", "+phone;}
23 }
```

Common attributes

Constructor

Methods

SUBCLASS CODE TO CONNECT TO SUPERCLASS

```
1  public class Employee extends Person{
2
3    private double hourlyRate;
4
5    public Employee(String fName, String lName,String address,
6                    String email, String phone, double rate){
7       super(fName, lName, address, email, phone);
8       hourlyRate = rate;
9    }
10
11   public double getHourlyRate() { return hourlyRate; }
12   public void setHourlyRate(double newRate) { hourlyRate = newRate; }
13   public String toString(){
14          return super.toString()+", hourlyRate: "+hourlyRate;
15   }
16 }
```

Creates Inheritance

Note superclass parameters + subclass parameters in constructor.

Keyword super: Call to superclass constructor. This is required.

SUBCLASS METHOD OVERRIDE EXAMPLE

```
11    public double getHourlyRate() { return hourlyRate; }
12    public void setHourlyRate(double newRate) { hourlyRate = newRate; }
13
14    public String toString(){
15            return super.toString()+", hourlyRate: "+hourlyRate;
16    }
17 }
```

Subclass `toString()` method overrides the superclass `toString()`. It provides a different implementation, and it calls the superclass `toString()` method!

(this is not the same as method overloading)

POLYMORPHISM EXAMPLE: DRIVER CLASS

```java
public class PersonDemo {

  public static void main(String[] args){

   Student myStudent = new Student("Joe", "Smith", "12 Penny Lane", "joe@geemail.com",
                                                   "411 333-1234", 3.75, "Math");
   Person person1 = new Employee("Moe", "Smith","100 Easter Island", "moe@geemail.com",
                                                   "777 988-1234", 11.00);
   Person person2 = new Faculty("Edgar", "Poe", "9 Raven Circle", "poe@geemail.com",
                                                   "876 123-4455", "COMPSCI");
   System.out.println(myStudent.toString());
   System.out.println(person1.toString());
   System.out.println(person2.toString());
```

Polymorphism is the ability of an object to take on many forms or *behaviors*. Each subclass will exhibit it's own print behavior.

Here, a Person can exhibit many behaviors: Employee, Faculty, or Student.

Subclasses override superclass toString() method.

EXAMPLE: DRIVER CLASS

```java
public class PersonDemo {

  public static void main(String[] args){

   Student myStudent = new Student("Joe", "Smith", "12 Penny Lane", "joe@geemail.com",
                                                   "411 333-1234", 3.75, "Math");
   Person person1 = new Employee("Moe", "Smith","100 Easter Island", "moe@geemail.com",
                                                   "777 988-1234", 11.00);
   Person person2 = new Faculty("Edgar", "Poe", "9 Raven Circle", "poe@geemail.com",
                                                   "876 123-4455", "COMPSCI");
   System.out.println(myStudent.toString());
   System.out.println(person1.toString());
   System.out.println(person2.toString());
```

In Java:
Determining which program behavior to execute depends on the **data type** of the derived (subclass) Object, not the data type of the reference.

Subclasses override superclass toString() method.

SUBCLASS TOSTRING OVERRIDE CODE

```
Joe, Smith, 12 Penny Lane, joe@geemail.com, 411 333-1234, 3.75, Math
Moe, Smith, 100 Easter Island, moe@geemail.com, 777 988-1234, hourlyRate: 11.0
```

In the super class:

```
public String toString(){return fName+", "+lName
                    +", "+address+", "+email+", "+phone;}
```

In Student subclass

```
public String toString(){
    return super.toString()+", "+gpa+", "+ major;}
```

In Employee subclass

```
public String toString(){
        return super.toString()+", "+hourlyRate;}
```

Notice the call to the superclass toString()

Clicker Question 2

Which of these keyword must be used in the class definition to inherit a class?

```
public class Employee ____?____ Person{
```

A. `inherits`

B. `private`

C. `extends`

D. `super`

Clicker Question 2 Answer

Which of these keyword must be used in the class definition to inherit a class?

```
public class Employee ____?____ Person{
```

A. `inherits //no such keyword`

B. `private //incorrect usage`

C. `extends`

D. `super //incorrect usage`

EXAMPLE 3: INHERITANCE USAGE

```
Student myStudent = new Student("Joe", "Smith", "12 Penny Lane", "joe@geemail.com",
                                                "411 333-1234", 3.75, "Math");
Person person1 = new Employee("Moe", "Smith","100 Easter Island", "moe@geemail.com",
                                                "777 988-1234", 11.00);
Person person2 = new Faculty("Edgar", "Poe", "9 Raven Circle", "poe@geemail.com",
                                                "876 123-4455", "COMPSCI");

Person[] personnel = new Person[3];
personnel[0] = myStudent;
personnel[1] = person1;
personnel[2] = person2;

for(Person curPerson : personnel) {
   System.out.println(curPerson.toString());
}
```

Note: using superclass reference!

EXAMPLE 3: INHERITANCE USAGE – contd.

```
Person curPerson;
    for(int i=0; i<personnel.length; i++) {
      curPerson = personnel[i];
    System.out.println(curPerson.toString());
```

Note: using superclass reference!

Java will execute the implementation that is found in the derived Object.
In this case, the derived Objects are Student, Employee, Faculty.

```
Joe, Smith, 12 Penny Lane, joe@geemail.com, 411 333-1234, 3.75, Math
Moe, Smith, 100 Easter Island, moe@geemail.com, 777 988-1234, 11.0
Edgar, Poe, 9 Raven Circle, poe@geemail.com, 876 123-4455, COMPSCI
```

Printout shows Polymorphic behavior.

POLYMORPHISM SUMMARY

Polymorphism is not just about inheritance.

- **Compile-time Polymorphism** - compiler decides which of several identically-named methods to call based on the method's arguments.

  Example: method *overloading*.
- **Runtime polymorphism** - the determination of which method to call is made while the program is running.

  Example: subclasses *overriding* superclass method.

Clicker Question 3

```
Person personA;
personA = new Student("Zoe","Smith","12 Sylvan",
    "zoe@mymail.com”,"222 333-1234", 3.75,"Math");
System.out.println(personA.toString());
```

What is printed?

A. Zoe, Smith, 12 Sylvan, zoe@mymail.com, 222 333-1234, 3.75, Math

B. Compiler Error: Class mismatch

C. [LPerson;@7852e922

D. Zoe, Smith, 12 Sylvan, zoe@mymail.com, 222 333-1234

Clicker Question 3

```
Person personA;
personA = new Student("Zoe","Smith","12 Sylvan",
    "zoe@mymail.com”,"222 333-1234", 3.75,"Math");
System.out.println(personA.toString());
```

What is printed?

A. Zoe, Smith, 12 Sylvan, zoe@mymail.com, 222 333-1234, 3.75, Math

B. Compiler Error: Class mismatch

C. [LPerson;@7852e922

D. Zoe, Smith, 12 Sylvan, zoe@mymail.com, 222 333-1234

Java executes the Student implementation of toString().

INHERITANCE & REUSE SUMMARY

- The big picture: programming is about **REUSE**.
- A large OO program, say in Java, might use **hundreds or even thousands** of classes.
- A great many of these have been previously created, or are **variants** of classes that have been **previously created**.

WEEK 10 TO DO

- Complete zyBook chapter 10.
- Start the next project early - good for practising concepts.
- Visit online office hours for help.
- Post in Moodle or Piazza for help.