

# Inheritance

# This week

---

- **Project 2 Interviews:** until Nov 4<sup>th</sup>
- **Homework 7:** Nov 4<sup>th</sup>
- **Quiz 7:** due Sunday, **Nov 7th**
  
- **Project 3 - CYO**
  - Design Meeting: Nov 7<sup>th</sup> – Nov 9<sup>th</sup>
  - Code Skeleton due on **Nov 10<sup>th</sup>**

# Design Meetings

- Your idea: what's the story (at least a written outline)
- **Classes skeleton:**
  - Data members
  - Member function stubs
- **Main class:** “Game, World, Universe, ...”
- How you will meet requirements
  - What will you read from a file?
  - Where do the data come from/look like?
  - What will you write to a file?
- **Code skeleton: pseudocode**
  - Game flow -- what functions?
- Optional: Class diagrams

Account
- id: int - balance: double
+ Account() + Account(newID: int, initialBalance: double) + setId(newID: int): void + getId(): int + setBalance(newBalance: double): void + getBalance(): double + withdraw(amount: double): void + deposit(amount: double): void

# Practicum 3 – 100 points

---

- Coming up in week 13: Nov 14<sup>th</sup>- scantron
- Covers material from weeks 1 – 10
  - Chapter 2
  - Chapter 3
  - Chapter 4 - all sections except 4.10
  - Chapter 5 - all sections except 5.9 and 5.10
  - Chapter 6
  - Chapter 8 - sections 8.1, 8.2, 8.3
  - Chapter 9 - all sections except 9.10 and 9.11
- Practice Questions and Review guide released today

# Practicum Retake

---

- During the time allotted for finals – NO FINAL
- Retake practicums 1/2/3..
- Most recent score will be recorded

# Example

---

```
class Student
{
public:
    Student();
    void setMajor(string m);
    void display() const;
private:
    string major_;
};
```

# Example – can we create an instance of Student in main()

---

```
class Student
{
public:
    Student();
    void setMajor(string m);
    void display() const;
private:
    string major_;
};
```

# Today

---

- Inheritance



# Inheritance

---

- In the real-world, inheritance is the process by which one object (or class of objects) is assigned traits/characteristics based on a larger class that it belongs to.
- Can think of it as an “is a” relationship.

## Examples:

- Every car is a vehicle.
- Every sedan is a car.

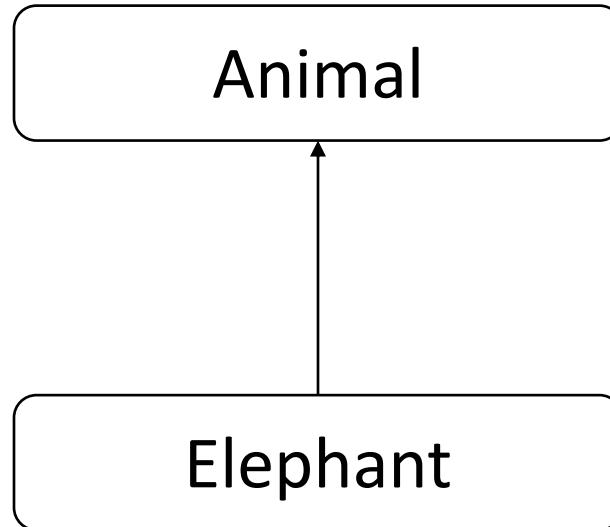
# Example

---

- We have elephant objects in our program...
- We now want to add Giraffes as well

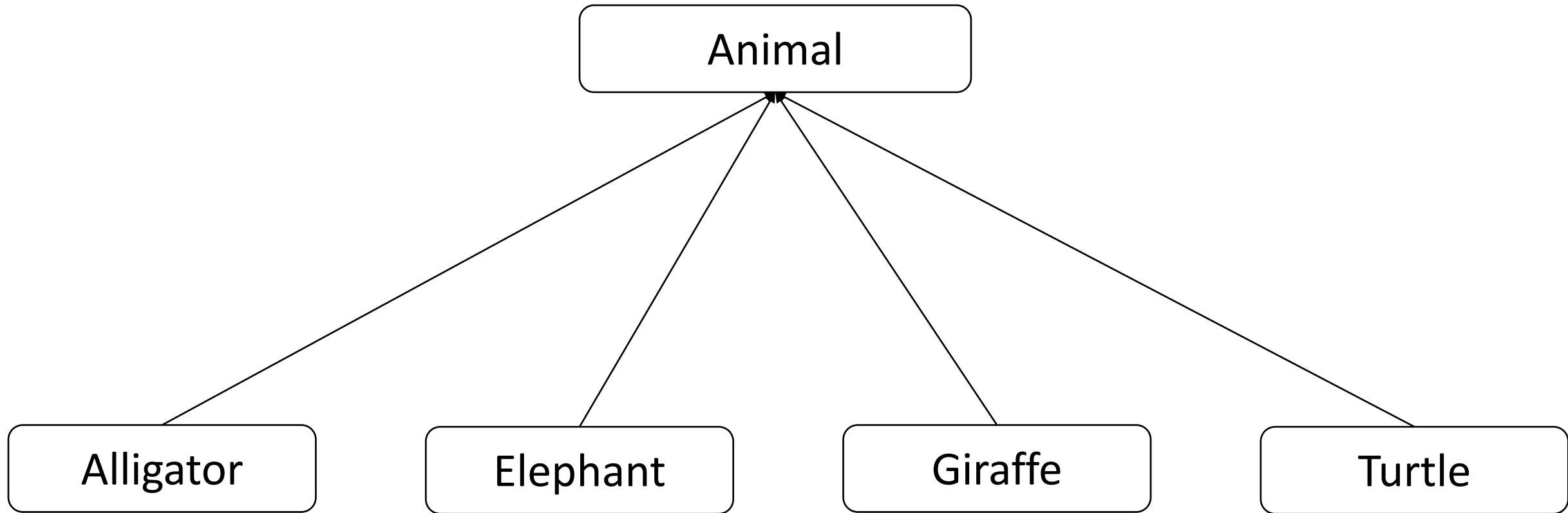
# Example – single inheritance

---



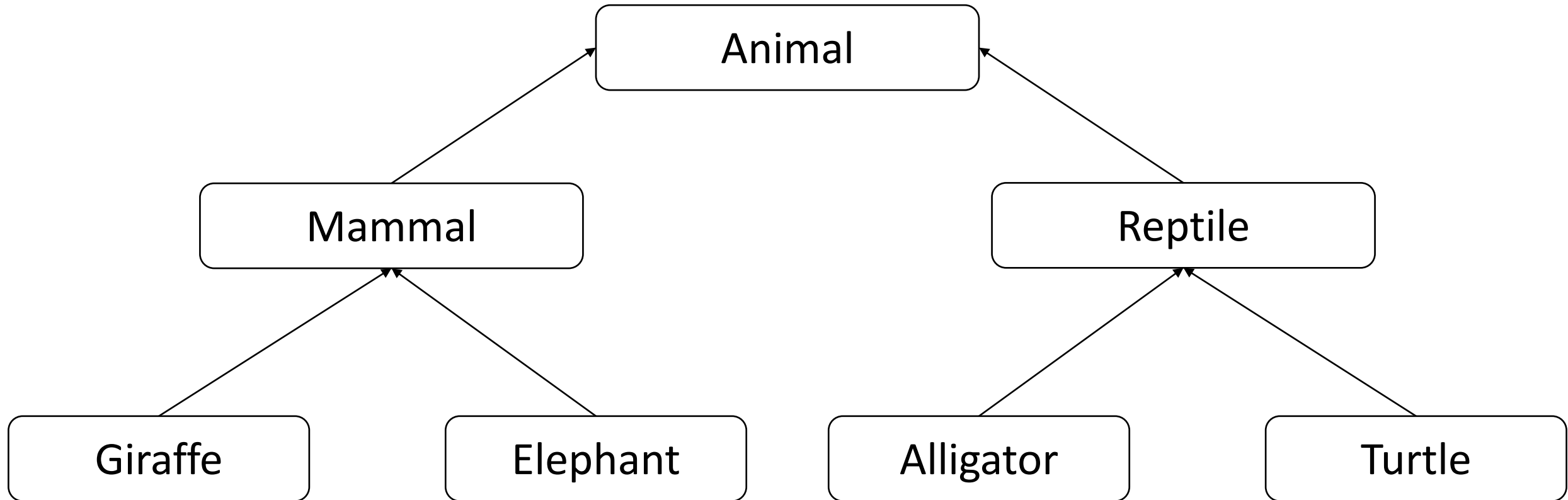
# Example – hierarchical inheritance

---



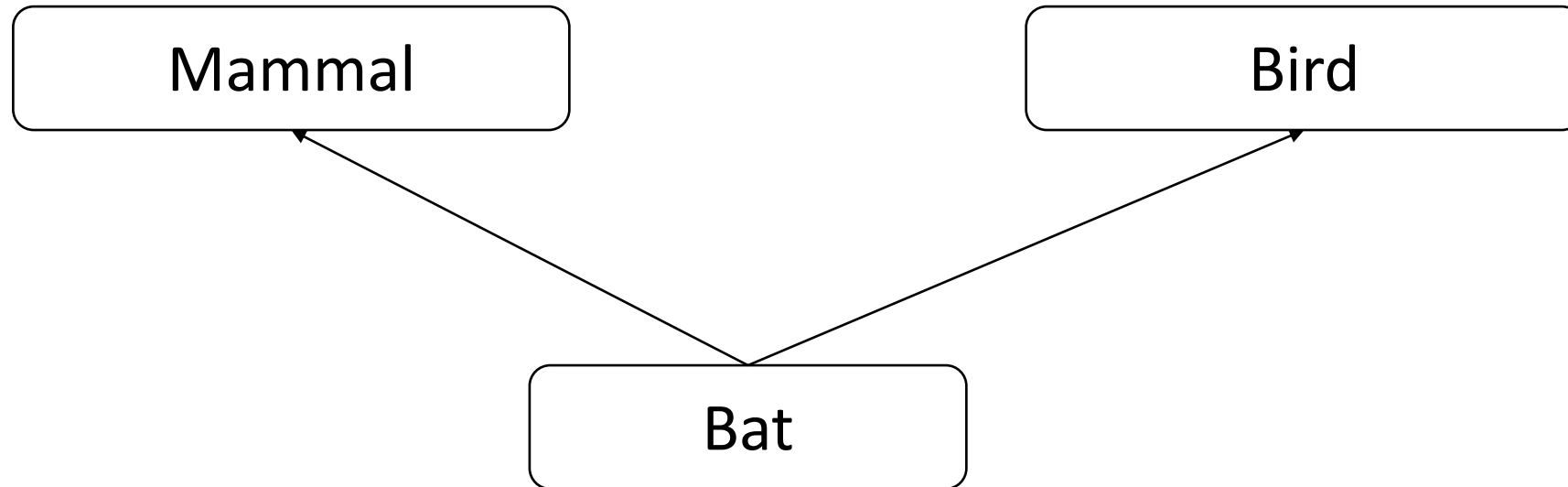
# Example – multilevel inheritance

---



# Example – multiple inheritance

---

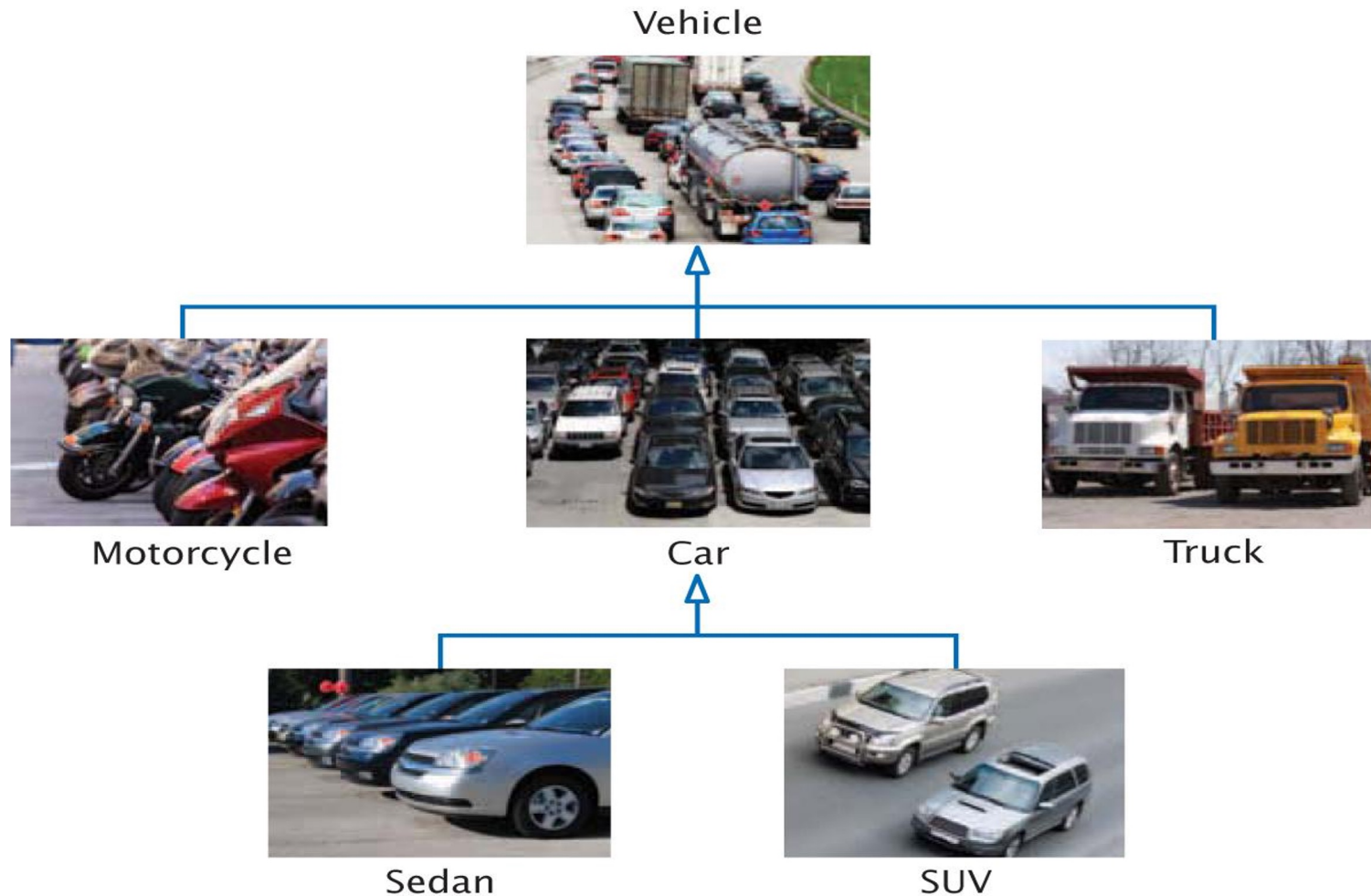


# Inheritance

---

- In object-oriented design, *inheritance* is a relationship between a more general class (called the **base class**) and a more specialized class (called the **derived class**).
- The derived class *inherits* data and behavior from the base class.
- Every car **is a** vehicle.
- IS-A denotes ***inheritance***.

# Inheritance: The IS-A Relationship



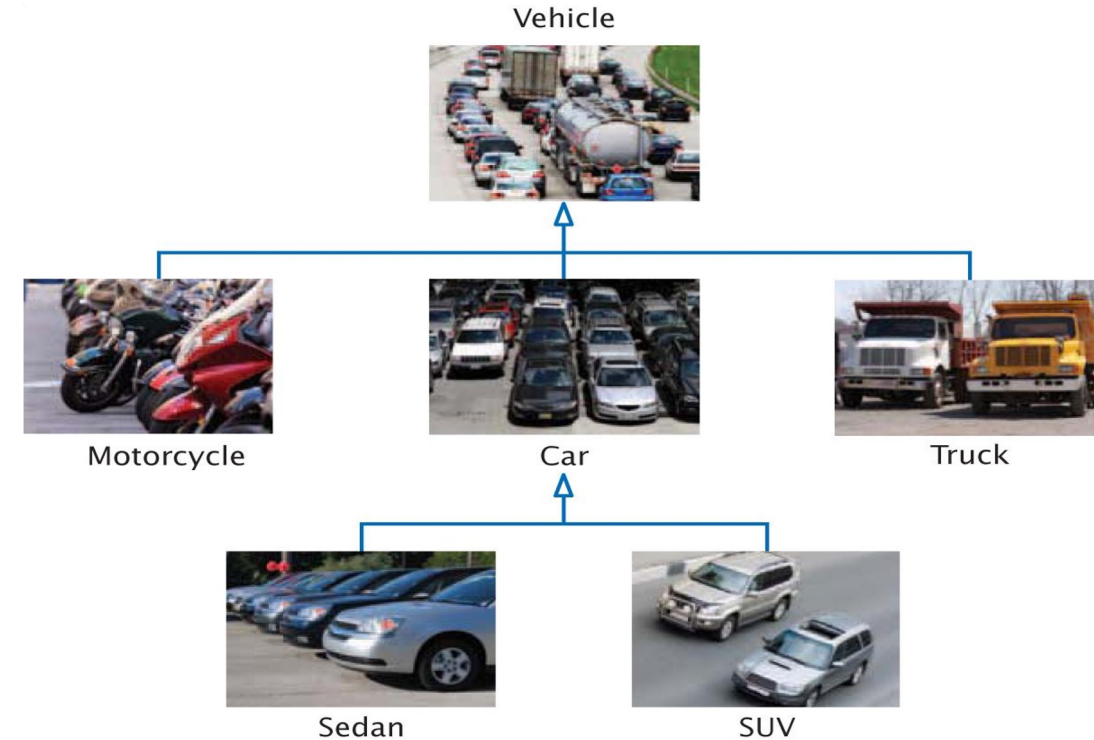
**Question:** In each of those examples, which is the base class and which is the derived class?



# Inheritance

Classes become more specific as you go deeper into the inheritance hierarchy.

- Vehicle is most general
- Car has all the properties of a vehicle, plus some specific to cars
- Sedan has all the properties of a car, plus some specific to sedans
  - Sedans and SUVs inherit properties specific to cars



# Substitution Principle

---

- The substitution principle states that you can always use a derived-class object when a base-class object is expected.
- You cannot use a base-class object when a derived-class object is expected

Example: `double gasMileage (Vehicle v)`

- Since a car IS-A vehicle, we can supply a **Car** object to such an algorithm or function, and it will work correctly.

# Person Hierarchy

---

People on a campus take on different roles:

- Student
  - Employee
  - Faculty
  - Staff
  - TA
- 
- Can we sketch out this inheritance hierarchy?

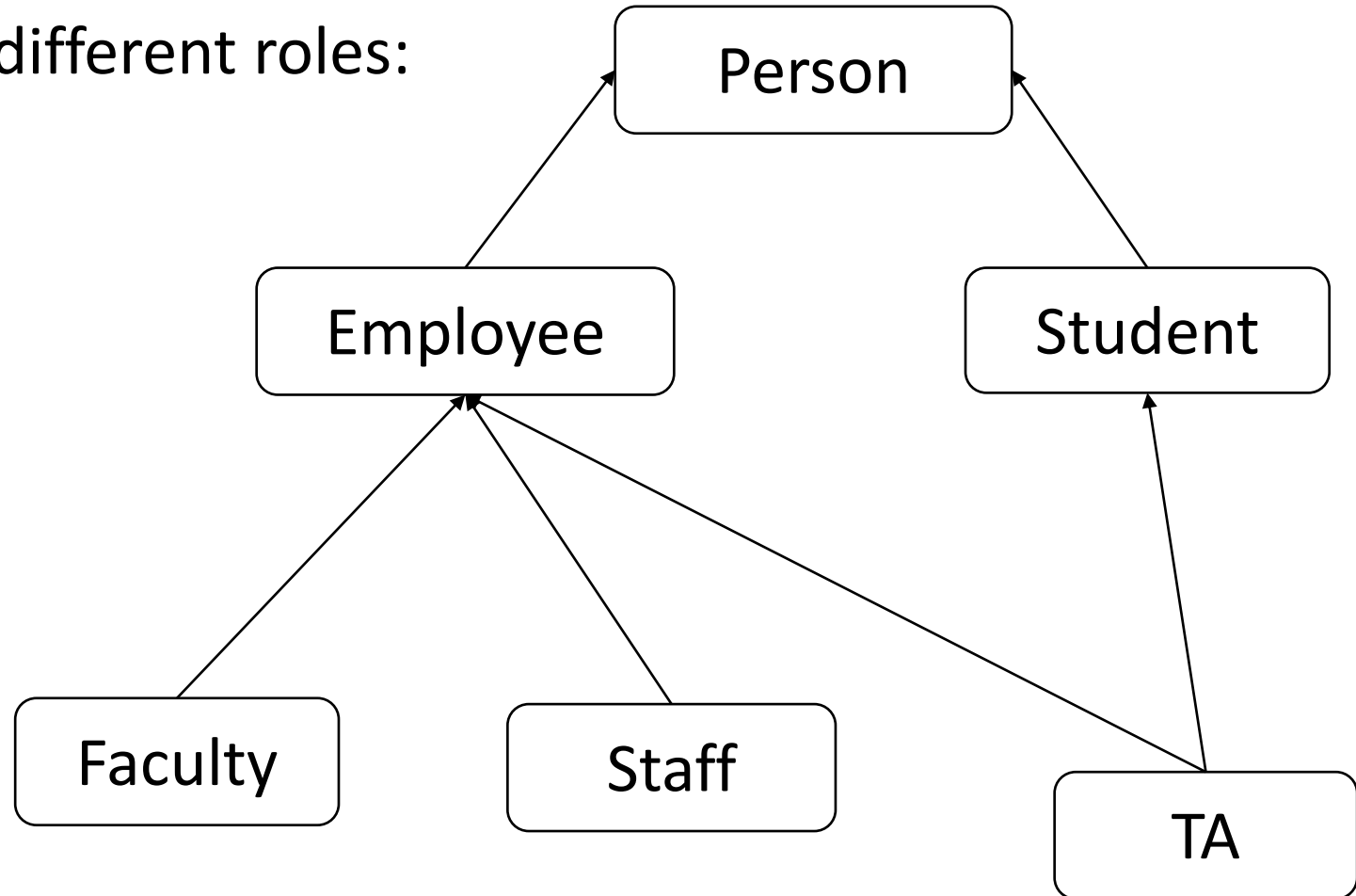
# Person Hierarchy

---

People on a campus take on different roles:

- Student
- Employee
- Faculty
- Staff
- TA

• Can we sketch out this inheritance hierarchy?



# Base class: Person

---

- Member functions: setName, setBirthday, display, constructor(s)
- Data members: name\_, birthday\_

# Base class: Person

---

- Member functions: setName, setBirthday, display, constructor(s)
- Data members: name\_, birthday\_

We want an object of person type to work like this:

- 1) First, the programmer sets the name and the birthday (stored in the Person object)
- 2) Then, the programmer asks the Person object info to be displayed to the user

# Implementing derived classes

---

- Now that Person class was the bare bones, most basic version of a person. We want different roles!
- Each role is a Person.  
(e.g., a student is a Person)
- So we start with the base class (Person) ...
- ... then write code for what makes each different type of role a special version of the more general Person type

# Implementing derived classes

---

Write code for what makes each different type of role a special version of the more general Person type

- Through inheritance, each of the derived classes have the data members and member functions that we set up in the base Person class
- AND we can define new stuff that makes the derived classes special!



# Derived classes: Student

---

```
class Student : public Person
{
    public:
        // new and/or changed member functions go here
    private:
        // additional data members go here
};
```

- The : denotes inheritance
- The public reserved word is needed for technical reasons
  - We want to inherit publicly, otherwise we wouldn't be able to use our Person member functions except within Student

# public inheritance

---

```
class Base
{
    public:
        int a;
    protected:
        int b;
    private:
        int c;
};
```

```
class Derived: public Base
{
    // a is public
    // b is protected
    // c is not accessible from Derived
};
```

# protected inheritance

---

```
class Base
{
    public:
        int a;
    protected:
        int b;
    private:
        int c;
};
```

```
class Derived: protected Base
{
    // a is protected
    // b is protected
    // c is not accessible from Derived
};
```

# private inheritance

---

```
class Base
{
    public:
        int a;
    protected:
        int b;
    private:
        int c;
};
```

```
class Derived: private Base
{
    // a is private
    // b is private
    // c is not accessible from Derived
};
```

# Derived classes: Student

---

Let's analyze what we need to do to make our specialized derived class.

- 1) We have name and birthday from the base Person class
- 2) But now need to set several data members that reside in Student...
- 3) ... which means we need to display the info a little bit differently.
  - we will do what is called overriding a member function  
(overriding = rewrite a specialized version for our derived class)

# Derived classes: Student

---

Member functions/data members we need to add/modify:

- Member functions: setMajor, display, constructor(s)
- Data members: major\_

# Derived classes: Student

---

```
class Student : public Person
{
public:
    Student();
    void setMajor(string m);
    void display() const;
private:
    string major_;
};
```

- We first specify the class we are inheriting from (Person) ...
- ... then only need to specify the differences (new or modified)
- You could slap this derived class definition right below the base class definition

```
class Person
{
public:
    Person();
    void setName(string person_name);
    void setBirthday(string date);
    void display() const;

private:
    string name_;
    string birthday_;
};
```

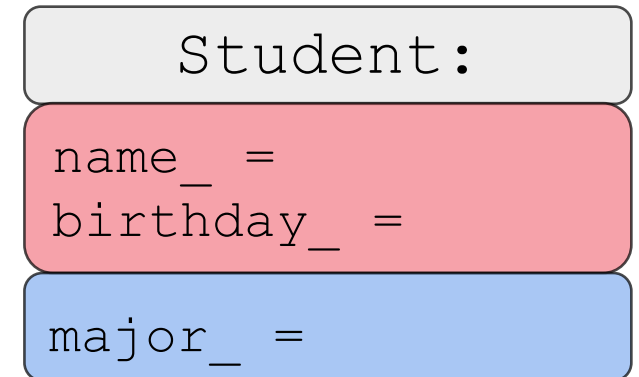
# Derived classes: Student

---

Student is one type, but made up of two

- One part is inherited from Person (name, birthday)
- Another part is new (major)

parts:



Member functions from the base class are public:

```
Student s1;
```

```
s1.setName("ABC"); // calls public member function of base  
class - okay!
```



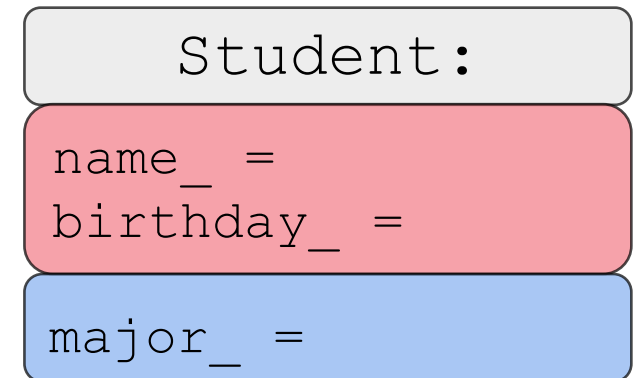
# Derived classes: Student

---

Student is one type, but made up of two

- One part is inherited from Person (name, birthday)
- Another part is new (major)

parts:



But data members from the base class are still private:

```
Student s1;
```

```
s1.name_ = "ABC"; // ERROR!
```

^-- here, the answer data member is private to the Question class, so even the derived type cannot access it

# Derived classes: Student

Moral of the story:

- When writing the Student member functions, we cannot directly access the private data members from the Person class
- Just like any other function, we must use our getters from the base class

But data members from the base class are still private:

```
Student s1;
```

```
s1.name_ = "ABC"; // ERROR!
```

^-- here, the answer data member is private to the Question class, so even the derived type cannot access it

parts:

Student:

name\_ =  
birthday\_ =

major\_ =

# Derived classes: overriding member functions

---

- Recall that our design requires that the display member function be rewritten in the Student class.  
→ This is called overriding a member function

**Person** display member function:

1) cout the name and birthday

**Student** display member function:

1) cout the name and birthday

2) then cout the major

# Derived classes: overriding member functions

---

The second part is easy – just print out the derived-class **major** data member:

```
void Student::display() const
{
    // Display the info
    ...
    // Display major
    cout << "Major: " << major << endl;
}
```

# Derived classes: overriding member functions

---

The first part *seems* easy -- just call the **display** function in the **Person** class

```
void Student::display() const
{
    // Display the info
    display(); // ERROR: calls the Student
              // version of display
    // Display major
    cout << "Major: " << major << endl;
}
```

# Derived classes: overriding member functions

---

The first part *seems* easy -- just call the **display** function in the **Person** class

```
void Student::display() const
{
    // Display the info
    Person::display(); // Calls the Person version
                        of display
    // Display major
    cout << "Major: " << major << endl;
}
```

We remedy this by forcing it to call the Person version by prefixing with Person::

# Derived classes: overriding member functions

---

- Note that we do not necessarily need to use the base class's function.
- We could have just rewritten the code to display the name and birthday.
- But in that case, we'd need a getter for the name and birthday data members, which are private.
- So we would need a new getter for name and birthday in the Person class

# Class Inheritance

---

***Hierarchies*** showing an **is a** relationship

- Ex: A car is a vehicle
- Ex: An apartment is a household

## **Base class vs derived class**

- Base = most general
- Derived = more specific

## **Derived classes**

- New member functions, data members
- Overriding old member functions



# Resources

---

- Textbook: 10.4.3, Common error 10.5
- <https://www.geeksforgeeks.org/access-modifiers-in-c/>
- <https://www.programiz.com/cpp-programming/public-protected-private-inheritance>
- Example 1, 2 and 3 only @ <https://www.programiz.com/cpp-programming/function-overriding>