

Lecture 34: Inheritance

Last time:

1. Introducing inheritance ...

Today:

1. Project #7 assigned
2. More inheritance



Project #7 Assigned!

- Project due Sunday, 4/29 at 11 pm
- Project is **closed**
 - You must complete the project by yourself
 - Assistance can only be provided by teaching assistants (TAs) and instructors
 - You must not look at other students' code
- Start now!
 - Read entire assignment from beginning to end before starting to code
 - Check out assignment now from CVS
 - Follow the instructions *exactly*, as much of grading is automated

Inheritance

- One class (**derived class**, **subclass**) is constructed by importing (**inheriting**) information from another (**base class**, **superclass**, **parent class**) and adding new / redefining existing information
- To derive a class D from a base class B, use:

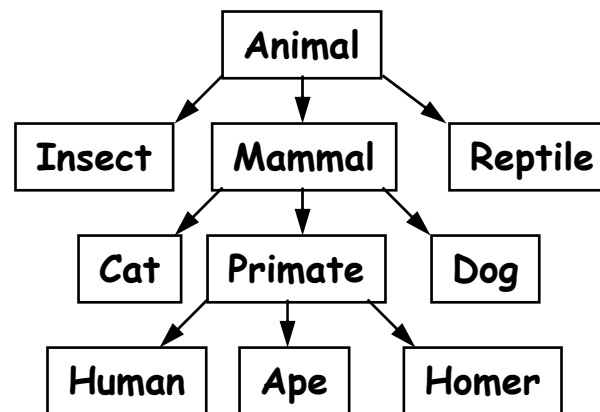
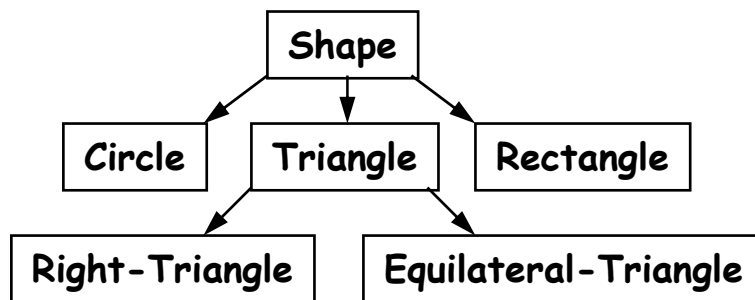
```
public class D extends B { ... }
```
- Example (we will look at this in next two slides):
 - Base class: `public class Shape`
 - Derived class: `public class Circle extends Shape`
- Derived class inherits all instance variables, methods from base class. It can also define new instance variables, methods
- In derived-class constructor, **super**(...) can be used to invoke constructor from base class
- Derived class can explicitly refer to entities from base class using `super`, e.g. `super.toString()`
- **Polymorphism**: object in derived class can be used anywhere base class is expected (a Student **“is a”** Person!)

Inheritance

- **Object Inheritance:** What does inheritance mean within the context of object-oriented programming?
- Suppose a **derived class, Circle**, comes from a **base class, Shape**:
 - Circle should have **all the instance variables** that Shape has. (E.g., Shape stores a color, and thus, Circle stores a color.)
 - Circle should have **all the methods** that Shape has (E.g., Shape has an accessor, `getColor()`, and thus, Circle has `getColor()`.)
 - Circle is allowed to define **new instance variables** and **new methods** that are particular to it:
 - (New) Circle Instance variables:** Center, radius.
 - (New) Methods:** `draw()`, `getArea()`, `getPerimeter()`.
- **Code reuse:** Code/Data that is common to all the derived classes can be stored in the base class. This allows us to **avoid code duplication**, and so makes development and maintenance easier.

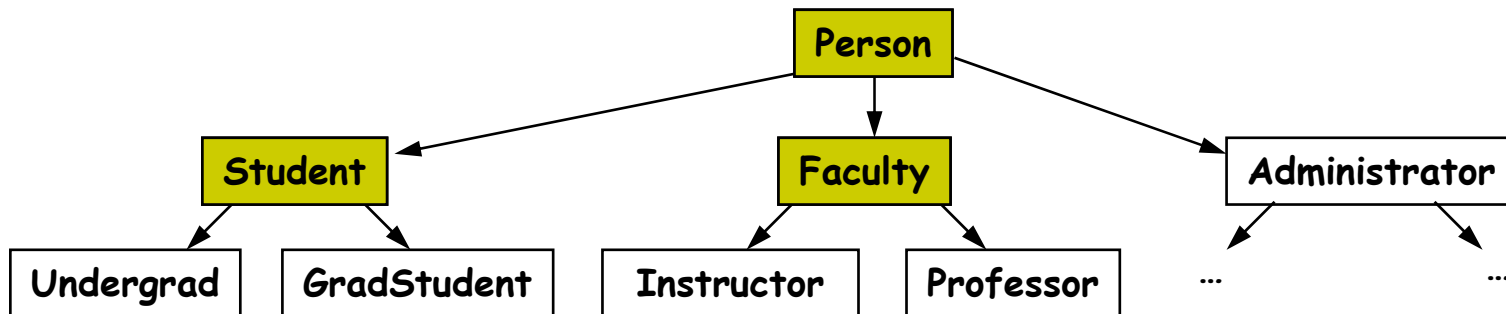
Inheritance More Generally

- Classes / objects have a natural “is-a” hierarchy
- Object-oriented programming provides mechanisms for exploiting this for
 - **Code re-use**
Common operations implemented in super classes
 - **Polymorphism**
Objects in subclasses can be used wherever superclass objects are needed



Example: People at University

- Base class: person
- Derived classes: student, professor, administrator





Base Class: Person (Part 1)

```
package university;
```

```
public class Person {
```

```
    private String name;           // person's name
    private String idNum;          // ID number
```

Instance variables

```
    public Person( ) {
        name = "No Name";
        idNum = "000-00-0000";
    }
```

Default constructor

```
    public Person( String n, String id ) {
        name = n;
        idNum = id;
    }
```

Standard constructor

```
    public Person( Person p ) {
        name = p.name;
        idNum = p.idNum;
    }
```

Copy constructor

```
    // ...other methods in part 2
```

```
}
```

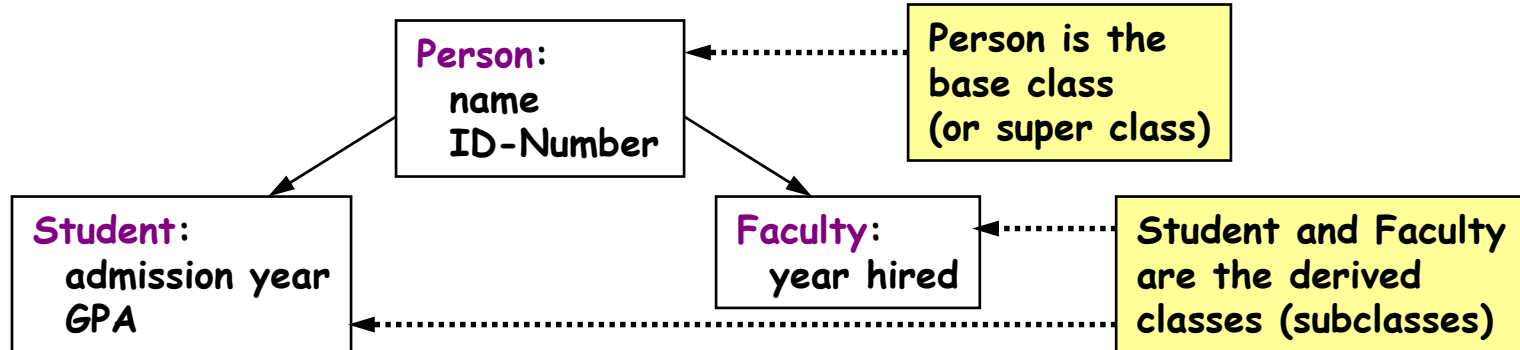
Interesting side note: Eclipse can set up Getters and Setters for you!

Source → Generate Getters and Setters

Not necessarily advisable!

Subclasses: Student, Faculty

- Each class **inherits** data, methods from Person
- Each adds special new data, methods appropriate for subclass



Derived Class Structure

- **Person:** (base class)

Instance Data: Name and ID-number.

String name

String idNum

Methods:

Constructors: default, standard, copy constructors.

Accessors/Setters: getName(), setName(), getIdNum(), setIdNum().

Standard methods: toString(), equals().

- **Student:** (derived from **Person**)

Instance Data: Admission year and GPA.

int admitYear

double gpa

Methods: (same structure as Person)

- **Faculty:** (derived from **Person**)

Instance Data: Year hired.

int hireYear

Methods: (same structure as Person)



Derived class: Student (Part 1)

```
package university;  
public class Student extends Person {
```

Tells Java that Student is derived from Person

```
private int admitYear;  
private double gpa;
```

Additional instance variables

```
public Student( ) {  
    super( );  
    admitYear = -1;  
    gpa = 0.0;  
}
```

Default constructor

This calls the default constructor for base class (superclass), Person, to set name and idNum.

```
public Student( String n, String id, int yr, double g ) {  
    super( n, id );  
    admitYear = yr;  
    gpa = g;  
}
```

Standard constructor

Calls Person constructor.

```
public Student( Student s ) {  
    super( s );  
    admitYear = s.admitYear;  
    gpa = s.gpa;  
}
```

Copy constructor

Calls Person copy constructor.

```
// ...other methods in part 2
```

```
}
```

Understanding Student

- extends specifies that Student is subclass of Person:
`public class Student extends Person`
- `super()`
 - When creating a new Student object, we need to initialize its base-class instance variables (from Person)
 - This is done by calling `super(...)`. E.g.
`super(name, id)` invokes constructor `Person(name, id)`
- `super(...)` must be the **first statement** of your constructor
 - If you **do not** call `super()`, Java will automatically invoke the base class's **default constructor**
 - If the base class's default constructor is undefined? **Error**
 - You must use `super(...)`, **not** `Person(...)`

Memory Layout and Initialization Order



- When you create a new derived class object:
 - Java allocates space for **base class** instance variables and **derived class** variables
 - Java initializes base class variables first, and then the derived class variables
- Example

```
Person ted = new Person("Ted Goodman", "111-22-3333" );
Student carole = new Student("Carole Goode", "123-45-6789",
                             2004, 4.0 );
```

