# CSC 143

Inheritance

## Introduction

- Classes have proven very useful
  - They help us write programs the way we view the world: interface vs implementation
- There is more to come! Java supports object oriented programming, i.e.,
  - Inheritance (sub classing)
  - Dynamic dispatch (polymorphism)
- Two very powerful programming tools!
  - They help us write less code

## Composition: has-a relationship

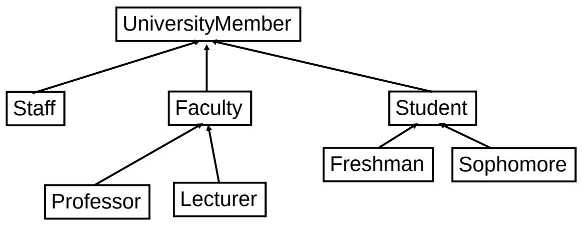
- We have used many times several classes together, e.g.,
  - Use an instance of a class as an instance field of another class

```
public class Landscape
{
 private Mountain m = new Mountain();
}
```

- This a "has-a" relationship
  - A **`Landscape`** "has-a" **`Mountain`**
- Also called aggregation

## Organization hierarchy

- Often, we classify things according to a hierarchy (from general to specific), e.g. part of the organization of a university



## Inheritance: is-a relationship

- Objects have also an "is-a" relationship
  - A Freshman "is-a" Student , a Lecturer "is-a" UniversityMember
- Java gives us the tools to implement an "is-a" relation. We can map our view of the world on the computer

## Inheritance in Java

- Inheritance is a way to encode the "is-a" relation in OO languages
  - Freshman declares that it "is-a" Student by inheriting from it
  - A derived class inherits from a base class by writing "**`extends BaseClassName`**" in the derived class declaration

derived class
(or subclass)

base class
(or superclass)

```
public class Freshman extends Student
 { /* Freshman-specific stuff here */}
```

## What is inherited? (1)

- The subclass inherits all of the public or protected or private members(data+methods) of its superclass (= everything but constructors)
- Members declared public or protected within the superclass can be used by the subclass as if they were declared within the subclass itself.
- protected member of a class: visible within the class itself, the package of the class, and the subclasses (even if they are in a different package). More on protected later.

## What is inherited? (2)

- private members of the superclass are unavailable outside the superclass scope.
- package level members of the superclass are unavailable outside the package scope. If the superclass and subclass are in the same package, the subclass can access any package level members of the superclass
- A subclass instance is also a superclass instance (is-a relation).
  - All methods of the superclass can be used on a subclass instance.

## Scope example

- Base.java

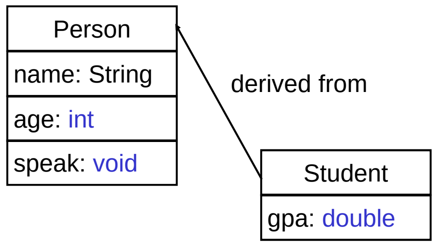
```
package org.seattlecentral;
 public class Base {
   public int i; //visible everywhere
   int j;//visible within the package
    protected int k; //visible within
    // the package and any derived class
   private int l;// visible within Base only
  }
```

- Derived.java

```
import edu.seattlecentral.Base;
 public class Derived extends Base{
   public void update(){
      i=4; // OK
      j=5; //Error, not in the same package
      k=6; // OK
      l=7; /*Error (not in Base)*/}}
```

## Example: Person, Student

- A person has a name, an age and can speak
- A student is a person with a gpa



## Person class

```
public class Person{
    public String name; // any name is OK
    private int age; // 0<=age<=130
    public Person(){
    //Call the other constructor
    this("someone",20);}
   public Person(String s, int a){
     name = s; setAge(a); }
    public void setAge(int a){
     if (a>=0 && a<=130) age=a;}
    public int getAge(){return age;}
    public void speak(){
     System.out.println("Hi, I am "+name);}
   }
```

## Student class

```
public class Student extends Person{
   private double gpa;
   public Student(){
    // Person() called automatically
    gpa = 3.5;}
    public Student(String s,int a,double g)
   {
     //Call the constructor Person (s,a)
     super(s,a); //Don't write Person(s,a)
     setGPA(g);
   }
    public void setGPA(double g){
     if (g>=0 && g<=4.0) gpa=g;}
    public double getGPA(){return gpa;}
   }
```

## Using Person and Student

```
Person p = new Person("Jane",25);
Student s = new Student("Robert",28,3.8);
p.speak(); // a person can speak
s.speak(); // a student can speak, since
             // a student is a person
```

## Static and dynamic types

```
// Can also write
 Person r = new Student("Sandra",28,3.9);
```

- The type used to declared r is Person = static type of r
- r points to a Student object. Student is the dynamic type of r.
- The static type of a variable never changes.
- The dynamic type may change

r = new Person(“Sandra, 28); // Dynamic type = Person

- Or may not be defined

  r = null;

## Method overriding

- How can we specialize speak for the Student class?
- Redefine speak within the Student class

```
public void speak(){
    super.speak();//let the Person speak
    // Add what the student say
    System.out.println("My gpa is "+gpa);
   }
```

- For any Student instance s, s.speak() invokes the speak method of the Student class. This is called method overriding.

## this keyword revisited

- this: two uses
  - a reference to the current object when within one of the object instance methods
  - Use this(*argument list*) when calling another constructor from a constructor (within the same class). The call must be the first executed statement in the constructor.

```
public Person(){
   System.out.println
   ("Default constructor");
   this(name,20); /*Error*/}
```

## super keyword

- super: two uses
  - a reference to the base object part of the current derived object within one of the derived object instance method.

```
super.speak();//In Student.speak()
```

  - Use super(*argument list*) when calling a base constructor from a derived constructor. The call must be the first executed statement in the constructor.

```
super(s,a);//In Student constructor
```

  - Can't chain the super calls

```
super.super.method();//Error
```

## Constructor calls

- When calling a constructor, Java
  - invokes the base class constructor (if necessary via an implicit call to super()).
  - initializes the instance variables of the current class
  - executes the statements in the constructor of the current class.

```
public class Derived extends Base{
    private int i=3;
    public Derived(){i++;}}
  Derived d = new Derived();
   // call Base(), set i to 3, increment i
   // What happens if Base is derived from
   // some other class?
```

## Overloading and Overriding

- Method overloading: same name but a different number or type of arguments.
  - A subclass can define some overloaded methods that augment the inherited overloaded methods provided by a superclass.
- Method overriding: define a method in the subclass with the same signature (return type and arguments) as the inherited method.
  - The method in the subclass shadows the method of the superclass.
  - Powerful when dealing with a hierarchy of objects (polymorphism: next slide)

## Polymorphism example

- Consider

```
Person[] group=new Person[2];
group[0]=new Person("Jane",25);
group[1]=new Student("Bob",26,3.9);
//OK, a Student is a Person
for(int i=0; i<2; i++)
  group[i].speak();
// What is printed?
```

## Polymorphism

- If there are several implementations of a method in the inheritance hierarchy of an object, the one in the most derived class always overrides the others.
- This is the case even if we refer to the object by way of a less derived type.

```
group[1].speak();
//invokes speak of Student = dynamic
// type of group[1]
```

## Dynamic binding and Polymorphism

- Polymorphism uses dynamic binding
  - The selection of the method is done at run time. The JVM starts looking in the dynamic type of the variable
- In Java, instance methods are by default dynamic (not the case in C++). To prevent overriding, declare the method final.

```
public final void speak()
{ /* speak code in Person */}
```

  - Can't write in Student class

```
public void speak(){/* code */}
// Compilation error
```

## Static binding and Overloading

- Overloading uses static binding
  - The selection of the method is done at compile time.
  - If in the Student class, we write instead

```
public void speak(String s){
//overload speak of Person
//takes a String as a formal parameter
  super.speak();
 System.out.println(s);}
```

  - Then

```
Person p = new Student();
p.speak(); //invokes Person.speak()
```

## Static methods (1)

- Reminder: static methods are not attached to any instance of a class. They belong to the class.
- A static method is invoked by using the class name. They are bound at compile time (That's why they are called static)

```
Math.sqrt(2);
```

- A static method can't be overridden.
  - You can't declare in a subclass as a non static method a superclass static method.
  - However, you can shadow a superclass static method in the subclass (unless the superclass method was declared final).

## Static methods (2)

```
 public class Base{
   public static void foo(){
    System.out.println("foo of Base");}
}

public class Derived extends Base{
 public void foo(){/*Code*/} // Error
 public static void foo(){ // OK
  System.out.println("foo of Derived");}
}
```

## final keyword

- final
  - When applied to a method, the method can't be overidden (or shadowed for a static method)
  - When applied to a class, the class can't be inherited.

```
public final class ThisIsIt{
 // class code }
public class MoreOfIt extends ThisIsIt {}
 // Error
```

- None of the methods in a final class are overridden. The compiler can optimize the use of the class methods (inlining).