Computer Science 22: Object Oriented Programming

Lecture #11: Inheritance

In this Lecture

- The concept of Inheritance in OOP
- Single Multi-level and Multiple Inheritance
- Effects of Inheritance
  - Abstract Classes
  - Final Classes
  - Final Methods

Four Pillars of Object Technology

- Abstraction – defines important characteristics of objects
- Encapsulation – hides implementation details and unimportant details from the user
- **Hierarchy** (**Inheritance**) – ranking/ordering of abstractions
- Modularity – grouping of related abstractions

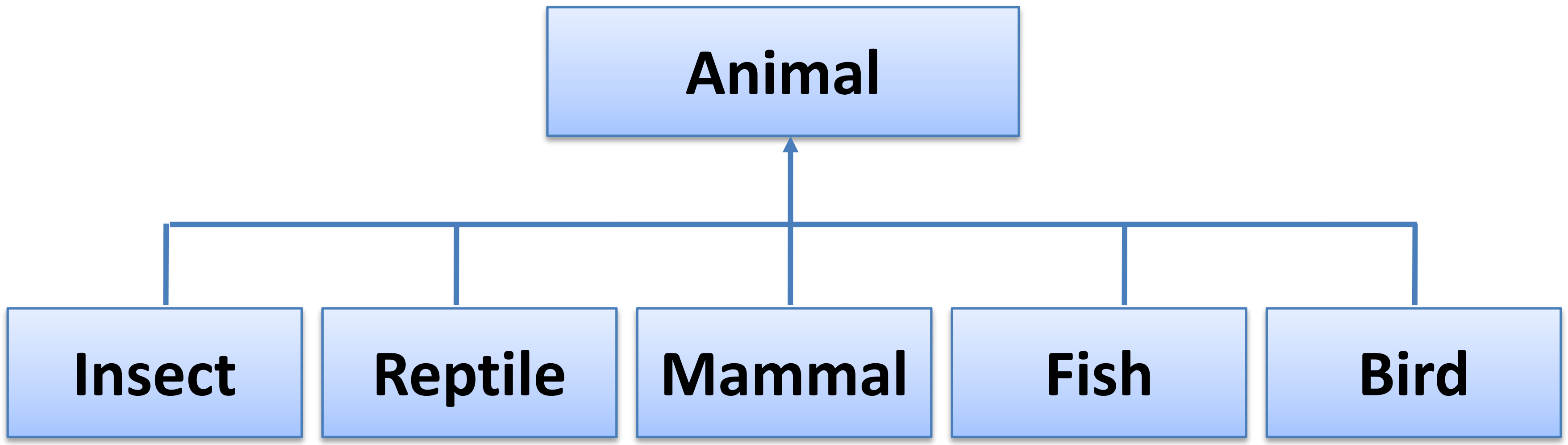
Inheritance

- Creating **derived classes** (**subclasses**) out of **base classes** (**superclasses**)
- Forms ranking or ordering of abstractions
- Simplifies our understanding of a problem
- Allows code reuse

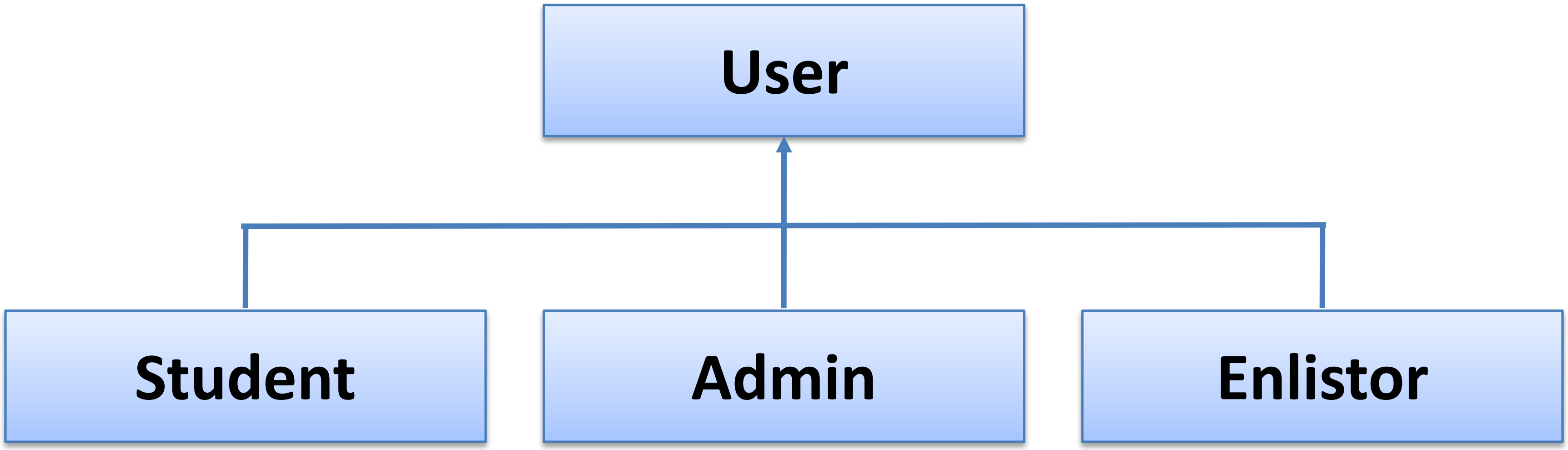
Inheritance in Modeling

- When two or more types of objects exist in a problem domain where instance of these types of objects
  - share common attributes AND behavior, but each have unique attributes and behavior (as demanded by their **specialization**)
  - can be **generalized** to a **super type** when we talk of these objects,
- Then we have a scenario by which can be modeled using inheritance

Inheritance in Modeling

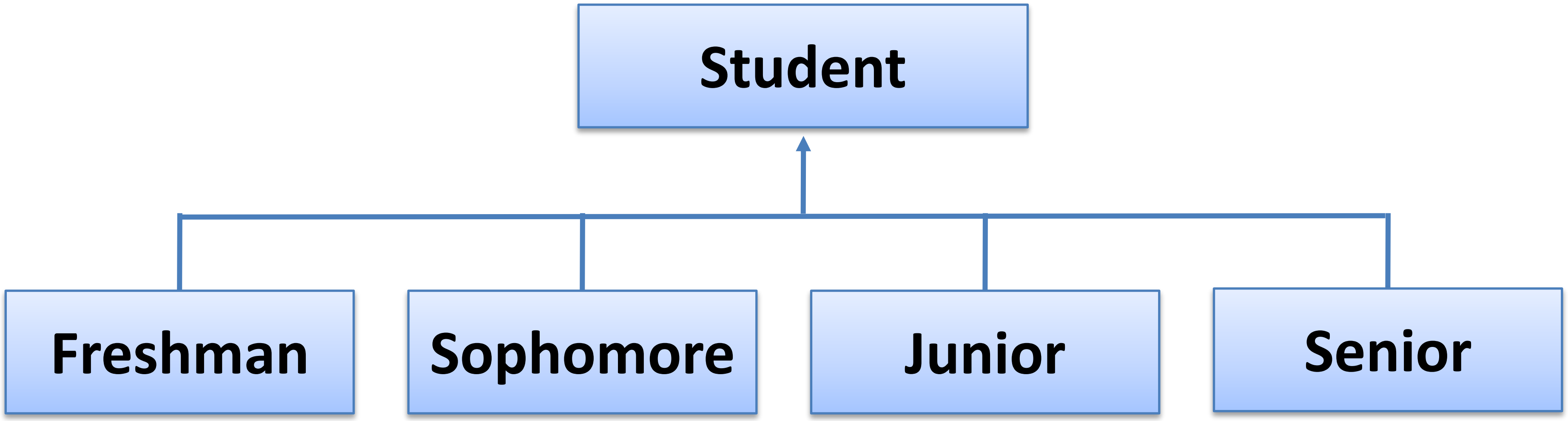


Inheritance in Modeling



Problem Domain: SystemOne

Inheritance in Modeling



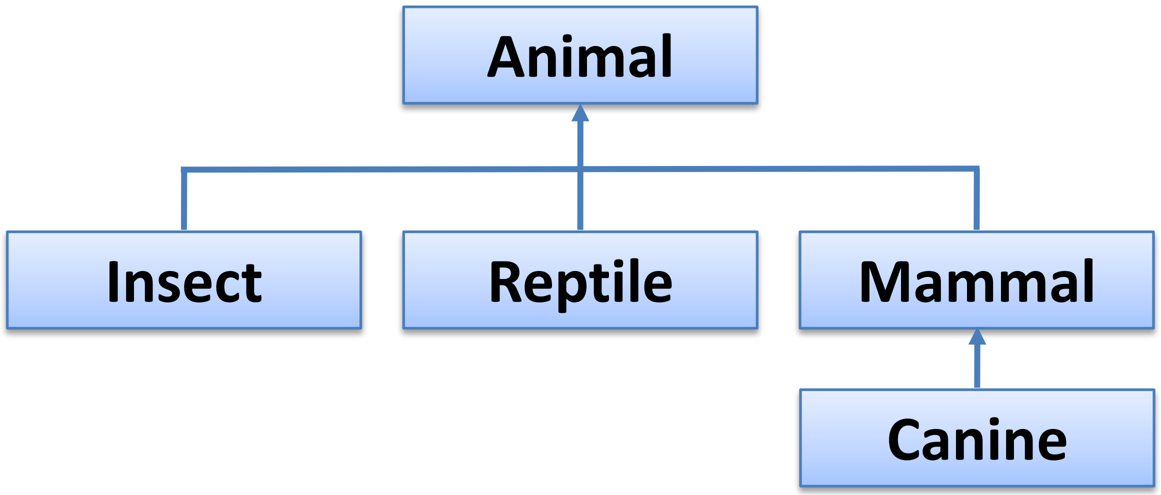
Note: May not be a good example

Inheritance

- A relationship between related types/classes
  - Superclass-subclass relationship
  - is-a/an, is-a-kind-of relationship
  - e.g., Enlistor is a kind of User in SystemOne
  - e.g., A reptile is an animal
  - e.g., Birds are animals

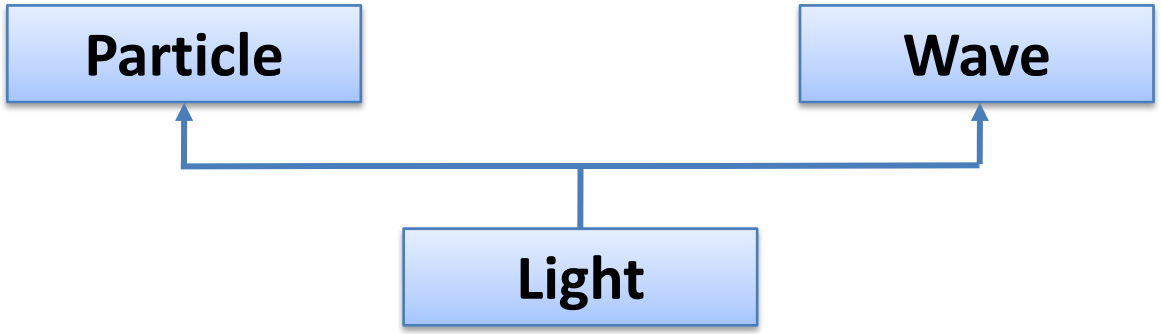
Kinds of Inheritance

- Single Inheritance
  - Subclasses are derived from one and only one superclass
  - Simple subtyping scheme
  - Java employs single inheritance



Kinds of Inheritance

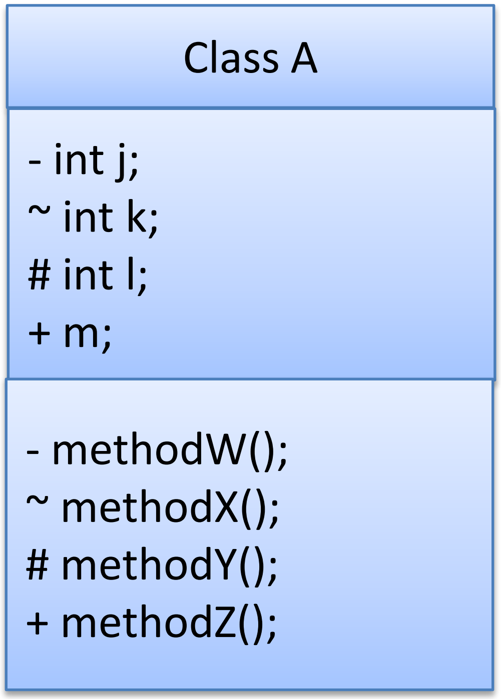
- Multiple Inheritance
  - A subclass may be derived from one or more super classes
  - Complicated to implement in a programming language
  - C++ employs multiple inheritance



Effects of Inheritance

- The derived class (subclass) takes over the access to attributes and methods (**public** and **protected** ones as well as **abstract methods**) from base class[es] (superclass[es])
- No need to redefine inherited methods – code from parent class is reused
- Less time coding subclasses
  - Important Note: Instances of the subclass are relatively larger (granularity) than instances of superclass[es]

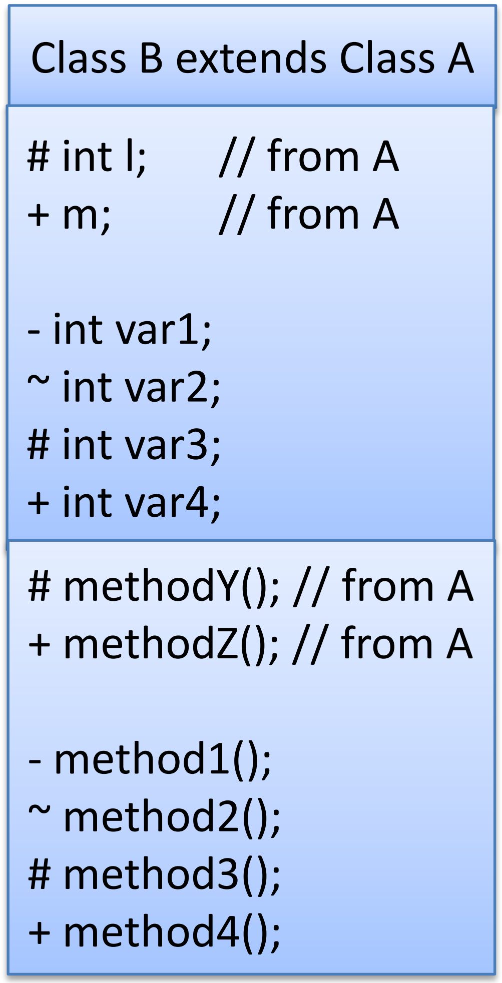
Effects of Inheritance



```
Class A

- int j;
~ int k;
# int l;
+ m;

- methodW();
~ methodX();
# methodY();
+ methodZ();
```



```
Class B extends Class A

# int l;        // from A
+ m;            // from A

- int var1;
~ int var2;
# int var3;
+ int var4;

# methodY(); // from A
+ methodZ(); // from A

- method1();
~ method2();
# method3();
+ method4();
```

Single Inheritance (Java)

```
public class Mammal {
  // contents of superclass
  // with implied contents from superclass:
  // java.lang.Object
}

public class Canine extends Mammal {
  // contents of subclass
  // with implied contents from superclass
}
```

Assignment

- Ask your lab instructor about the keyword **`super`**? How does it work? When is it used?

  Note: *You may also do your own research. The point is, you should know what the keyword is for by next meeting.*

Effects of Inheritance

```
class Boxer {
      public Boxer() { }
      public void punch() { … }
      public void jab() { … }
}

class KickBoxer extends Boxer {
      public  KickBoxer() { }
      public void kick() { … }
}
```

Effects of Inheritance

```
Boxer manny = new Boxer();
manny.punch();
manny.jab();
KickBoxer tosca = new KickBoxer();
tosca.punch(); //can do what “Boxers” do
tosca.jab();
tosca.kick();  //and here is something
               //only Kick Boxers do
```

Effects of Inheritance

- Superclass types can be bound to subclass instances.

```
Boxer frank = new KickBoxer();
//valid, frank is a kick boxer and
//kickboxers ARE boxers also.
//BUT frank is cast as a Boxer and “boxers”
//can't kick.
```

Inheriting Attributes

- All attributes are inherited, however, access modifier restrictions still apply.
  - **public** : public attributes (same)
  - **private** : private attributes **cannot be accessed** by subclass instances (as they are private to superclass instances only)
  - **protected**: protected attributes are **accessible** by subclasses only (so yes, these attributes can be accessed directly at the subclass level).

Inheriting Methods

- Same rules as inheriting attributes.
- **Method overriding**
  - This is done by redeclaring a method from the superclass and redefining its definition in the subclass

Method Overriding

```
class MySuperClass {
    //...
    public int process(
        float param) {
        //computation here
    }
    //...
}
```

```
class MySubClass extends
MySuperClass {
    //...
    public int process(
        float p) {
        //another version of
        //the computation here
    }
    //...
}
```

Abstract Classes and Inheritance

- **Abstract Classes**
  - Meant to represent abstract concept or entities
  - [Java, etc..] Abstract classes are designed only as **superclasses from which more specific and defined subclasses are derived from.**

Abstract Classes and Inheritance

```
public abstract class ChessPiece {
        //contents here...
        abstract void move();
        //other contents...
}

public class Knight extends ChessPiece {
        //contents here
        public void move() {
                //move in an L-shaped pattern
        }
        //other contents
}
```

Final Classes and Inheritance

- Final Classes
  - Designed to be a leaf node in the hierarchy. That is, we cannot derive subclasses from them

Final Classes and Inheritance

```
public final class MySubClass extends MySuperClass{
        // MySubClass can never be a parent/superclass
        // of another class
}
```

Final Methods and Inheritance

- A method declared as final can no longer be overriden

```
public class MySuperClass {
        public final int process(float param){
                //computation here
        }
}


public class MySubClass extends MySuperClass {
        public int process(float p){  // compiler error
                //computation here
        }
}
```