Computer Science 22: Object Oriented Programming

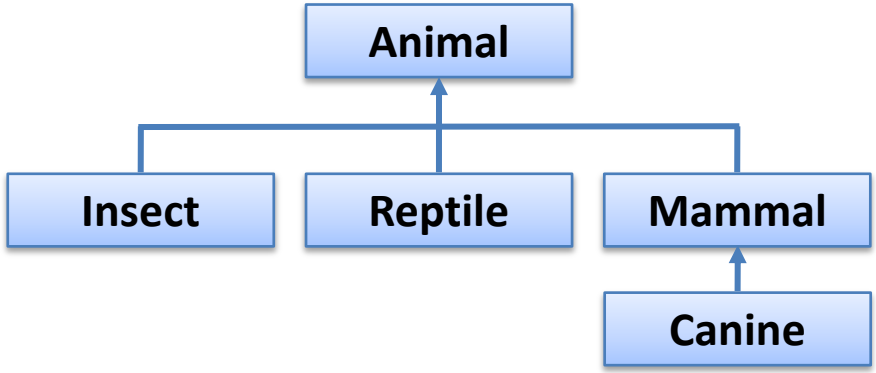
Lecture #12: Inheritance II

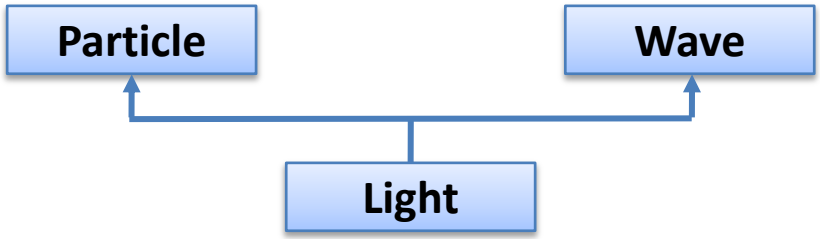
In This Lecture

- Types of Inheritance (Review)
- Interface
- Problems with Inheritance

Types of Inheritance

- **Single Inheritance**
  - A subclass is derived from one and only one superclass
- **Multiple Inheritance**
  - A subclass may be derived from more than one superclass





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
}
```

Multiple Inheritance (C++)

```
class Vehicle {
   //... contents here
}

class Car: public Vehicle {
   //Car derived from Vehicle
   //Single inheritance
}

class Jet { //... contents here }

class JetCar : public Car, Jet {
   //... contents here
}
```

Pseudo-Multiple Inheritance in Java

- Java cannot DO REAL multiple inheritance
- A Java class can extend only one superclass
  - ... but it may implement one or more **interfaces**
  - ... not really multiple inheritance but when it comes to **typing**, an objet now comes with more than one type

Interface

```
public interface C {
   //should be saved in a file called C.java
   public void aMethod(String s, int x, int y);
   public void anotherMethod(float f);
   public void yetAnotherMethod(String s1, String s2);
}

public interface D {
   //should be saved in a file called D.java
   public void setIsAlive(boolean d);
}

public interface E {
   //should be saved in a file called E.java
   public boolean negate(boolean d);
   public String joinCharacters(char c, char d);
}
```

Interface

```
public class A extends B implements C, D, E {
   // since class A 'implements' the interfaces C, D, and E,
   // it agrees to implement all the method signatures found in
   // these interfaces
   public void aMethod(String s, int x, int y) {...}
   public void anotherMethod(float f) {...}
   public void yetAnotherMethod(String s1, String s2) {...}
   public void setIsAlive(boolean d) {...}
   public boolean negate(boolean d) {...}
   public String joinCharacters(char c, char d) {...}

   // class A can still declare methods not in the interfaces
   public char myMethod() {...}
}
```

Effects of Implementing Interfaces

```
public class A extends B implements C, D, E {
   // instances of A can assume multiple types
}

C myC = new A();          // an instance of class A casted as C
myC.aMethod("gecko moria", 7, int 2);
myC.anotherMethod(7.13f);
myC.yetAnotherMethod("dracule mihawk", "bartholomew kuma");

D myD = new A();          // an instance of class A casted as D
myD.setIsAlive(true);

E myE = new A();          // an instance of class A casted as E
myE.negate(false);
myE.joinCharacters('J', 'R');

A myA = new A();          // what are the methods of myA
```

Pseudo-Multiple Inheritance in Java

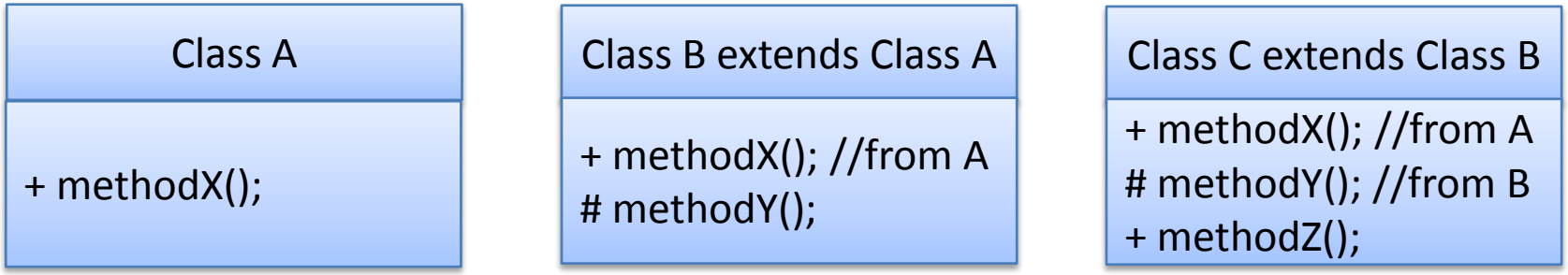
```
public class A extends B implements C, D, E {
   // instances of A can assume multiple types
}

C myC = new A();       // an instance of class A casted as C
D myD = new A();       // an instance of class A casted as D
E myE = new A();       // an instance of class A casted as E
```

an instance of A is an A (of course)
an instance of A is-a C
an instance of A is-a D
an instance of A is-an E

And so?

Problems with Inheritance

- **The Yo-yo problem**
  - When understanding the behavior of an object whose class comes from a hierarchy, we might have to go up and down the hierarchy to trace the behavior.



Problems with Inheritance

- **Dependency Problem**
  - Inheritance is also a dependency relationship in that stability and existence of a subclass demands the same from a superclass.
  - Therefore when “transporting” a subclass B, it must be transported with its superclasses.

Problems with Inheritance

- **Size and Efficiency Problem**
  - Note that subclasses are actually larger than their superclasses (although they might appear to be written with less code).
  - Sometimes, not all inherited methods are required in a given problem.
  - It does save us programming time but performance may suffer when dealing with larger than required objects.
  - Therefore, inheritance trees should be as shallow as possible.