CSD221 Lecture 3

## **Lecture3 Inheritance And Method Overloading**

[**Lecture3 Inheritance And Method Overloading**](#_uo41txirx83t) **1**

[**Objectives**](#_sceifzogbkph) **2**

[**Object Families**](#_ws4pszqk84id) **2**

[many real world scenarios involve the concept of "families".](#_lgv2d5fn0wjc) 2

[Objects are in one way or another related to other objects](#_lgv2d5fn0wjc) 2

["inheritance" is the concept that related objects "pass down"](#_lgv2d5fn0wjc) 2

[characteristics from one object to another](#_lgv2d5fn0wjc) 2

[children inherit the characteristics of the parent](#_lgv2d5fn0wjc) 2

["abstract classes" and "method overloading" are important concepts](#_lgv2d5fn0wjc) 2

[within the context of inheritance.](#_lgv2d5fn0wjc) 2

[**Generalization and Specialization**](#_id9wgr5y195a) **3**

["classes" are a generalized form of "objects"](#_lgv2d5fn0wjc) 3

[an "object" is an instance of a "class"](#_lgv2d5fn0wjc) 3

[ex; the object "Mustang" is an "instance" of a Car class.](#_lgv2d5fn0wjc) 3

["classes" can also be organized in a similar fashion](#_lgv2d5fn0wjc) 3

[example, a class hierarchy that describes the animal kingdom](#_lgv2d5fn0wjc) 3

[Kingdom (ex; animals)](#_lgv2d5fn0wjc) 3

[Phylum (ex; vertebrates)](#_lgv2d5fn0wjc) 3

[Class (ex; mammmals)](#_lgv2d5fn0wjc) 3

[Order (ex; carnivore)](#_lgv2d5fn0wjc) 3

[Family (ex; cat)](#_lgv2d5fn0wjc) 3

[Genus (ex; felix)](#_lgv2d5fn0wjc) 3

[Species (ex; felix leo)](#_lgv2d5fn0wjc) 3

[graphically....](#_m32qkx71cq9) 3

[Figure 1](#_3rj8l67s6eha) 3

[**Inheritance**](#_bxoyn6ybb2r7) **4**

[NOTE to Student \*\* You will create these classes in lab3.](#_d60qlxcy4akq) 5

[Figure 2](#_l24nbb4stcr7) 5

[Notes\*\*](#_w098t2skhthv) 5

[Activity 1](#_26fg4po7xe50) 6

[Feedback 1](#_j2zdiozhphtn) 6

[Figure 3](#_vbf998ic1osb) 7

[Activity 2](#_f1a8jx625i6h) 8

[**Implementing Inheritance in java.**](#_42wy177ffl5r) **14**

[Note to teacher\*\*](#_l2dlnu3f4qgo) 15

[Output](#_4fo0ofm7j0se) 15

[**Access Control**](#_fp03uds10jen) **16**

[**Abstract Classes**](#_3np2uqgqhsr) **17**

[**Overriding Methods**](#_tk4314wzv9mp) **17**

[**The "Object" class**](#_u14lzjjynifn) **18**

## 

## Objectives

### 

### 

### 

### 

## Object Families

### many real world scenarios involve the concept of "families".

### Objects are in one way or another related to other objects

### "inheritance" is the concept that related objects "pass down" characteristics from one object to another

### children inherit the characteristics of the parent

### "abstract classes" and "method overloading" are important concepts within the context of inheritance.

## Generalization and Specialization

### "classes" are a generalized form of "objects"

### an "object" is an instance of a "class"

### ex; the object "Mustang" is an "instance" of a Car class.

### "classes" can also be organized in a similar fashion

### example, a class hierarchy that describes the animal kingdom

### Kingdom (ex; animals)

### Phylum (ex; vertebrates)

### Class (ex; mammmals)

### Order (ex; carnivore)

### Family (ex; cat)

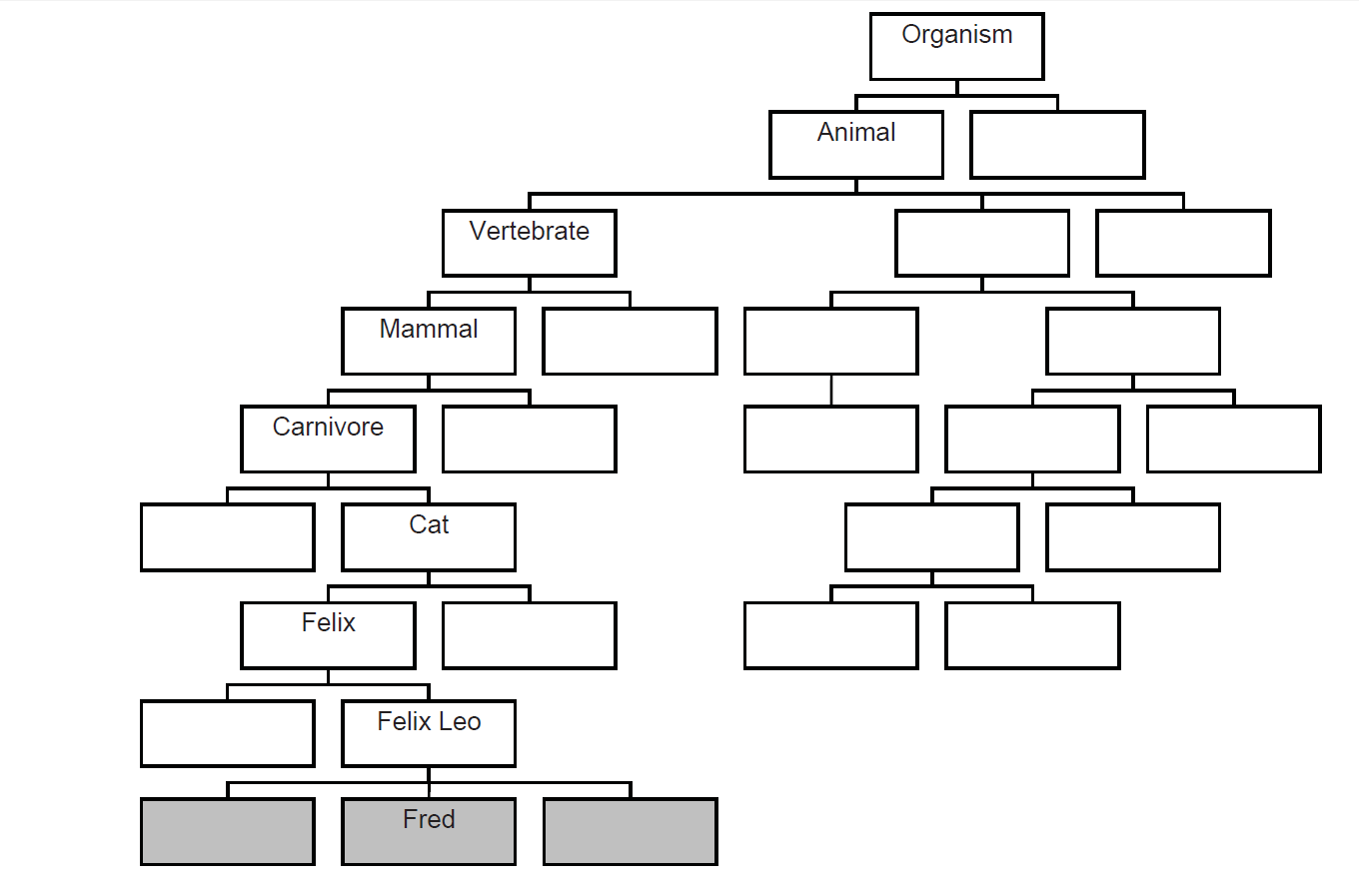
### Genus (ex; felix)

### Species (ex; felix leo)

### graphically....

### 

### **Figure 1**

****

* "Fred" is not a class, its an "instance" of a Felix Leo, an actual animal
* Fred **is a** Felix Leo **is a** felix **is a** carnivore
* carnivores "eat meat" so "Fred" has the characteristic of "eating meat".
* Fred **is a** Felix Leo **is a** felix **is a** carnivore **is a** mammal **is a** vertebrate
* vertebrates have a backbone so "Fred" has the characteristic "has a backbone"
* Note the **"is a"** relationship
* many real world relationships exhibit an "is a" relationship
  + A "Mustang" **is a** "Car" which **is a** "Vehicle"

## Inheritance

* in a class hierarchy, the higher up in the hierarchy, the more general the characteristics.
* The lower classes are more specific.
  + this is represented by the OOP terms "Generalization" and "Specialization"
  + so a Felix Leo is a more specific description of a Felix
  + a "Car" is a more specific description of a vehicle.
  + a "Mustang" is more specific than a "Car".
* "Inheritance" - all the characteristics of all the classes above a particular class are included in that class.
  + a mammal is a vertebrate is an animal is an organism.
* example;
  + books and magazines - both specific types of publication

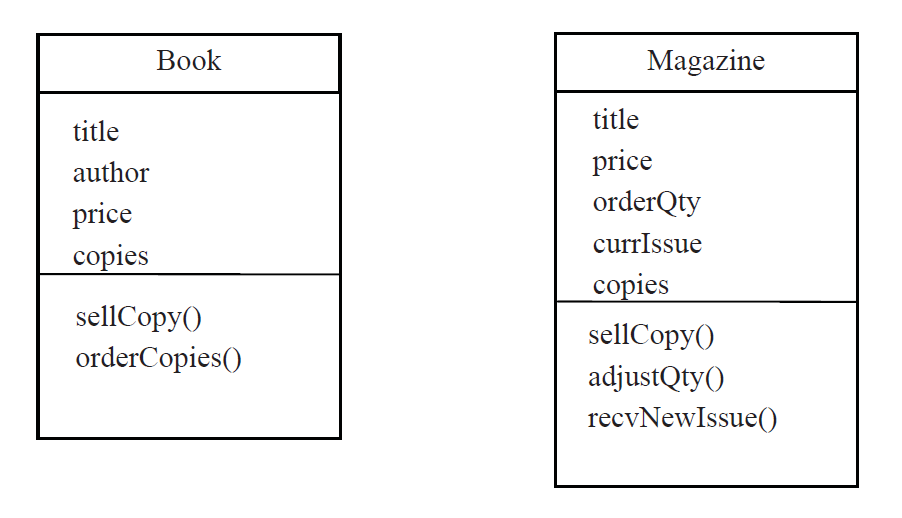
NOTE to Teacher \*\* Book, Publication and Magazine are in the

lecture3.w17 package in the

CSD221\_FredCarella\_public\_mvn\_git project

### NOTE to Student \*\* You will create these classes in lab3.

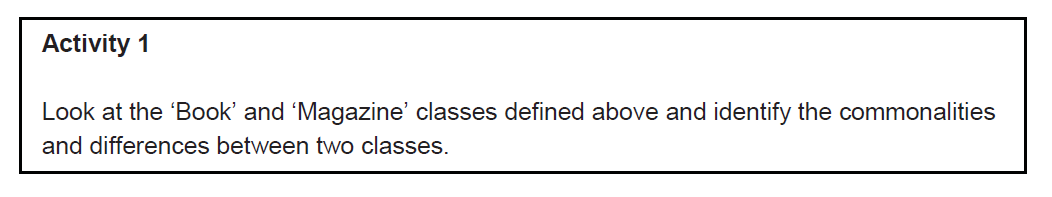
### Figure 2



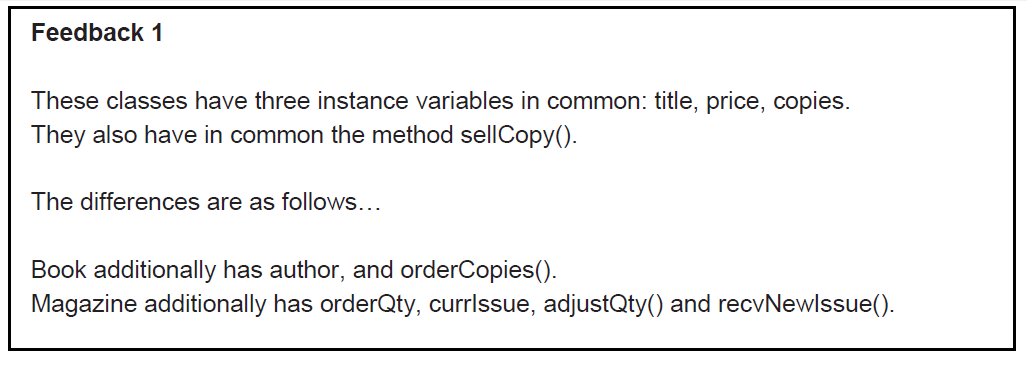
### **Notes\*\***

* Book:
  + title, author and price are obvious
  + copies == number of copies currently in stock
  + orderCopies(int num) == number of books to order and add to stock
* Magazine:
  + orderQty is the number of copies received of each new issue
  + currIssue is the date/period of the current issue (e.g. “January 2009”, “Fri 6 Jan”, “Spring 2009” etc.)
  + when a newIssue is received the old are discarded and orderQty copies are placed in stock.
    - Therefore recvNewIssue() sets currIssue to date of new issue and restores copies to orderQty.
  + adjustQty() modifies orderQty to alter how many copies of subsequent issues will be stocked.

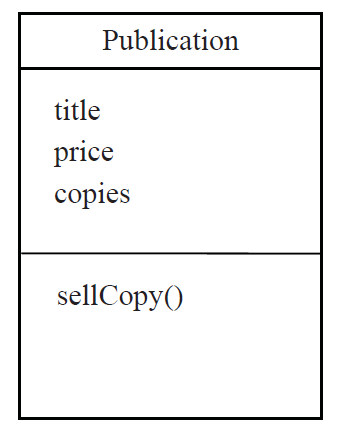
### **Activity 1**

****

### **Feedback 1**

****

* separate (or factor out) the common members into a superclass called publication

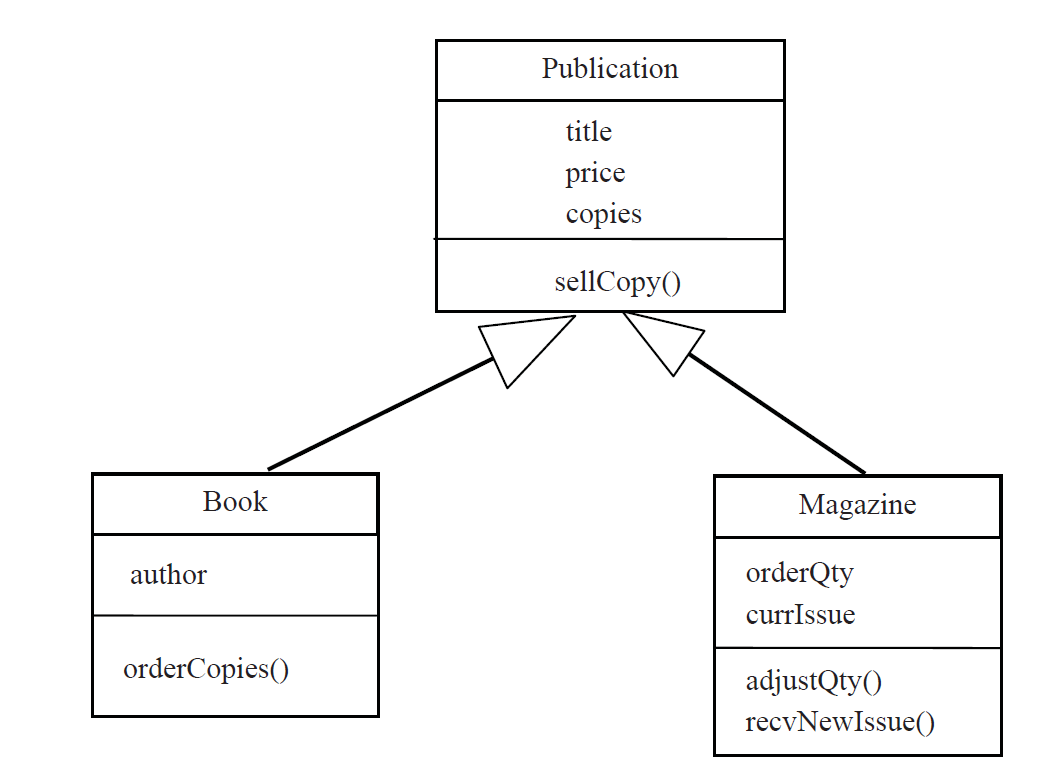


UML - the differences are specified in "subclasses" and are

illustrated in the UML diagram below.

The hollow arrow denotes "inheritance"

### **Figure 3**

****

Notes

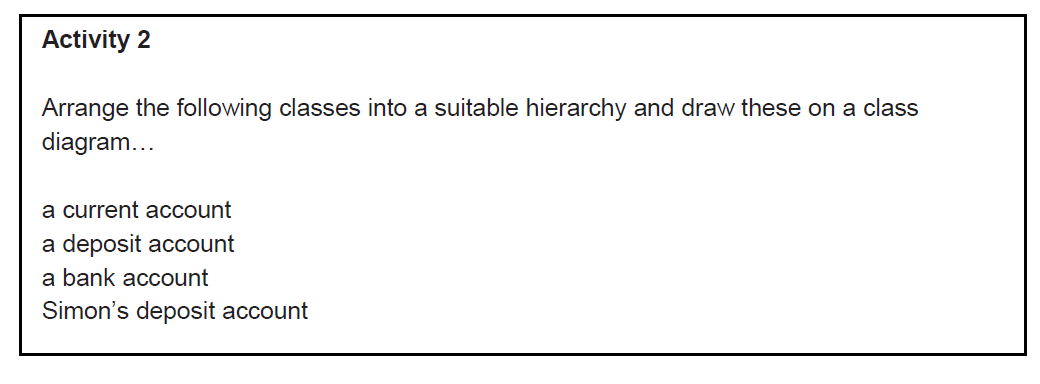
* the Subclass has the generalized superclass characteristics +
  + additional specialized characteristics.
* Thus the Book class has four instance variables (title, price,

copies and author) it also has two methods

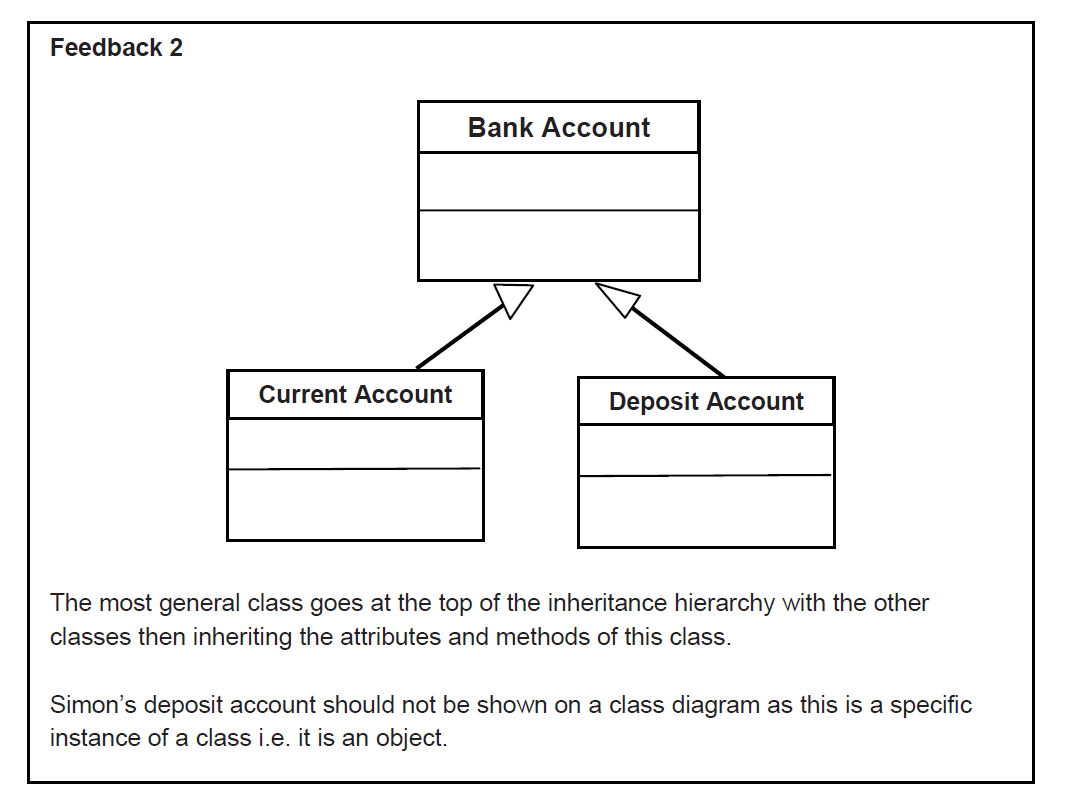
(sellCopy() and orderCopies()).

* The inherited characteristics are NOT listed in subclasses.
* The arrow shows they are acquired from superclass.

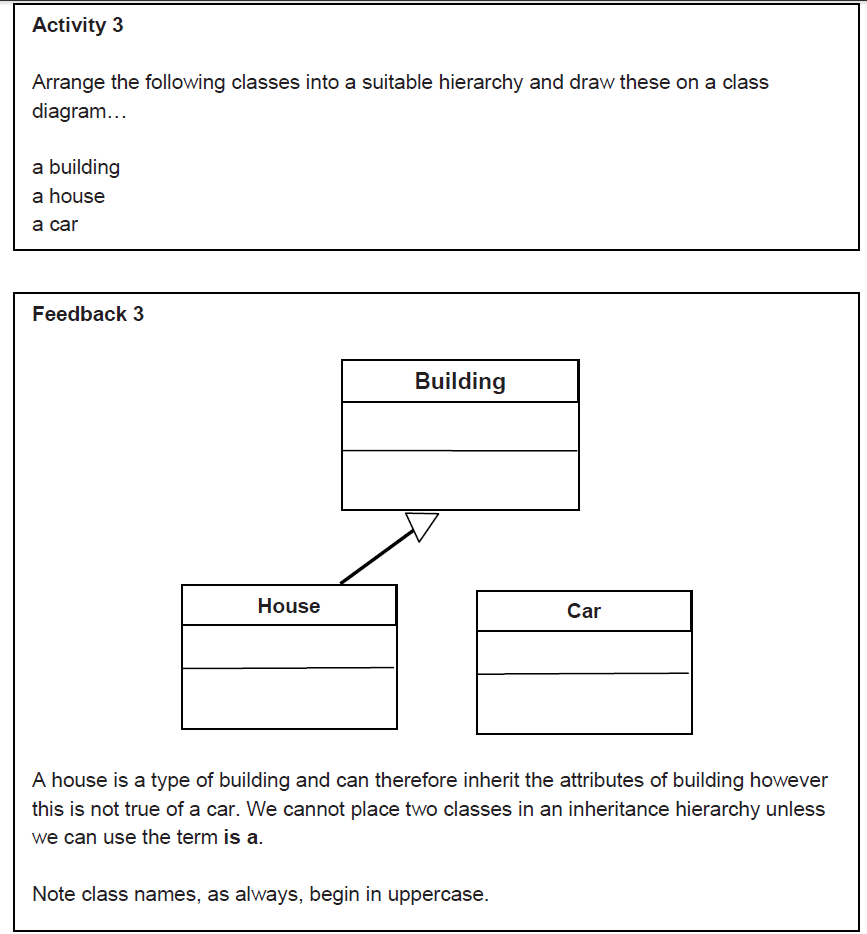
### **Activity 2**

****

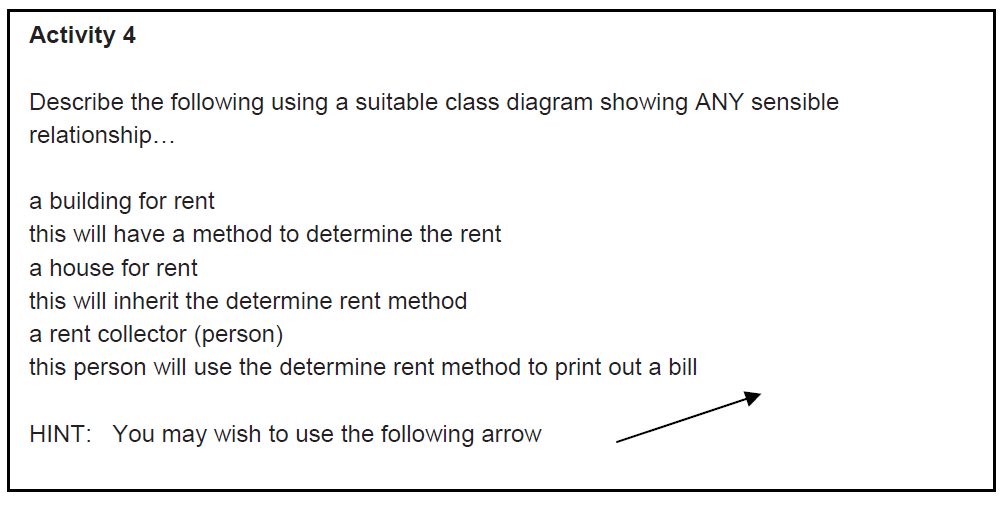
**Feedback 2**

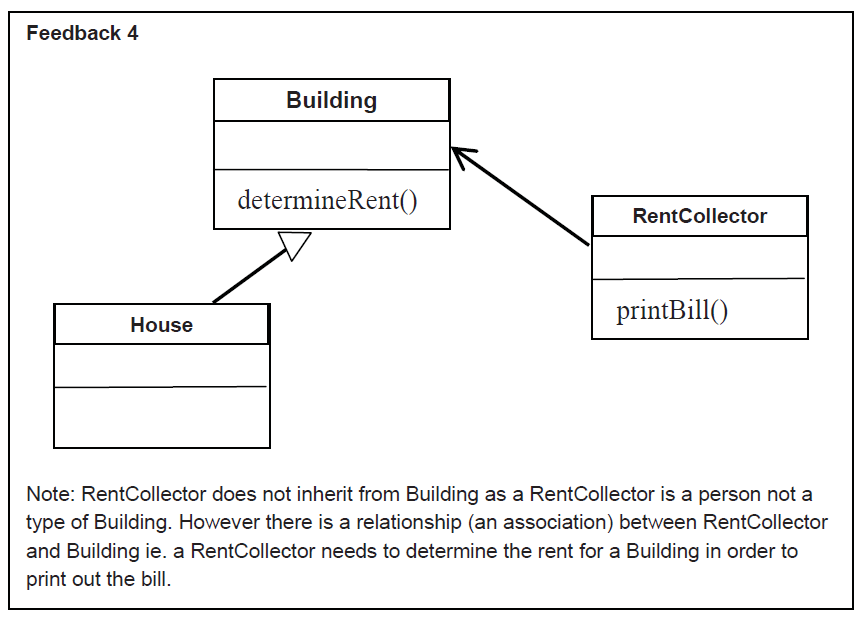
****

**Activity and Feedback 3**

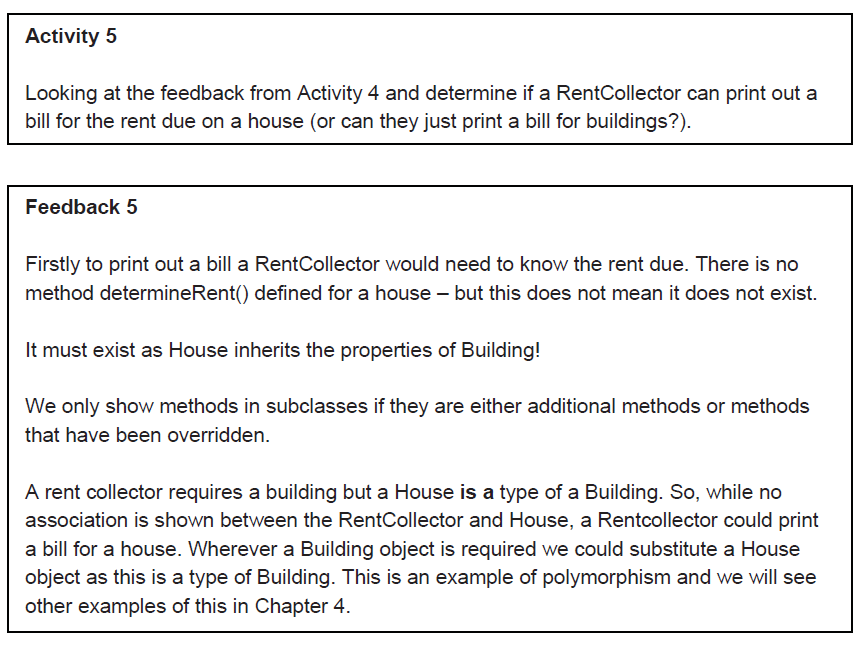
****

**Activity and Feedback 4**

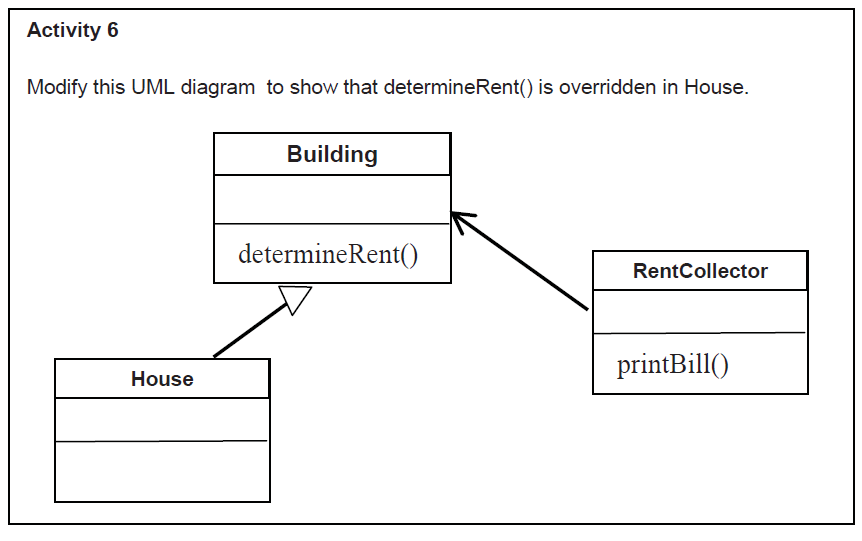
****

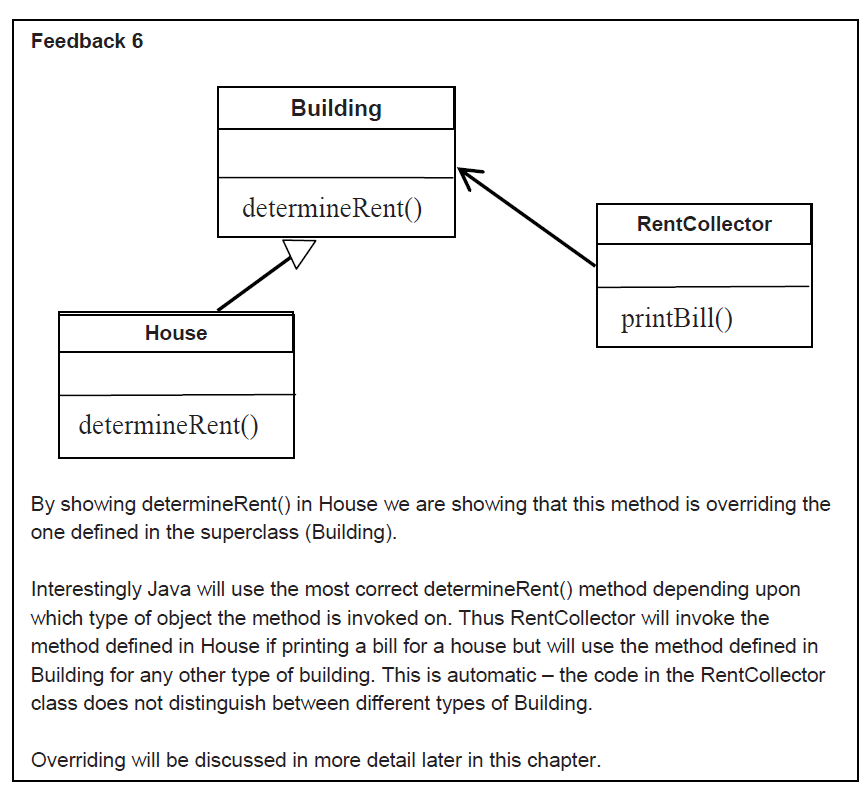
****

**Activity and Feedback 5**

****

**Activity and Feedback 6**

****

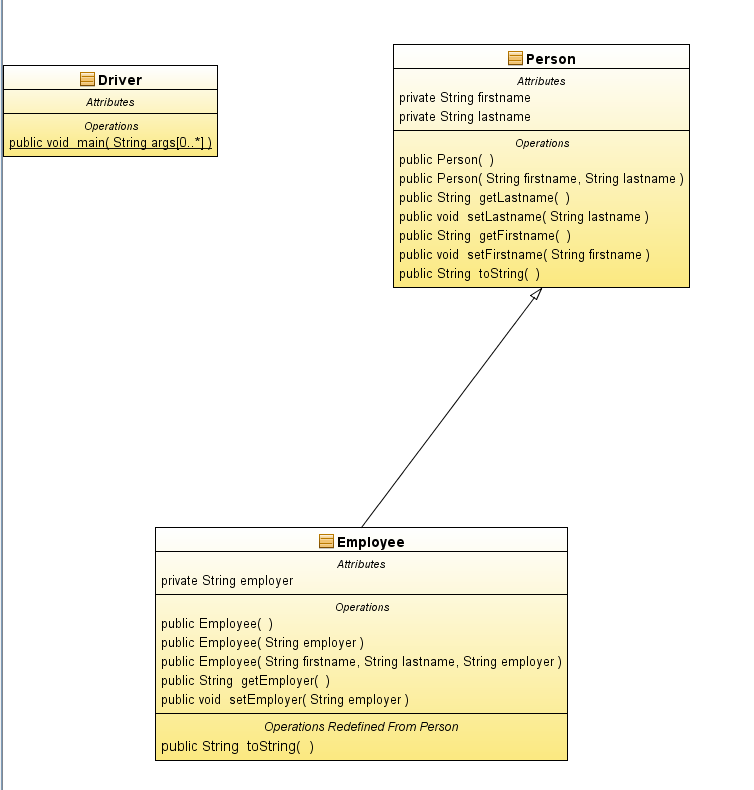
****

## **Implementing Inheritance in java.**

Checkout the zeus cvs project called **CSD221Winter2014FredCarellaNb671,** and refer to the source code in the ch3 package

* each class whether super or sub should initialize itself in its constructor
* refer to the ch3.implementinheritance package for the following
* discussion

The ch3 code implements the following UML diagram



### **Note to teacher\*\***

Review the code and then discuss the output of the program which shows debug output from

each constructor. Show the order in which they are called.

#### **Output**

------------------------

Person p=new Person();

Calling Person() constructor

System.out.println(p)==><default first name> <default last name>

------------------------

Person p2=new Person("Fred", "Carella");

Calling Person(String, String) constructor

System.out.println(p2)==>Fred Carella

------------------------

Employee e1=new Employee();

Calling Person() constructor

Calling Employee() constructor

System.out.println(e1)==><default first name> <default last name> works for <default employer>

------------------------

Employee e2=new Employee("Sault College");

Calling Person() constructor

Calling Employee(String) constructor

System.out.println(e2)==><default first name> <default last name> works for Sault College

------------------------

Employee e3=new Employee("Fred","Carella", "Sault College");

Calling Person(String, String) constructor

Calling Employee(String, String, String) constructor

System.out.println(e3)==>Fred Carella works for Sault College

------------------------

## **Access Control**

* to enforce encapsulation, make instance variables private and make public setters and getters for each variable to allow for public access.
* setters and getters allow you to restrict and/or validate data
  + for example, you can prevent "-ve" values on copies.
* if setters/getters are public, any other class can use them.
  + if you make them protected, only sub-classes can access them.

## **Abstract Classes**

* of the three classes, Publication, Magazine and Book, it makes sense to instantiate objects of type Book and Magazine but NOT Publication.
  + a Publication is too general.
  + if you wanted another class of Publication like a Calendar then you would sub-class Publication.
  + a Publication will never be instantiated.
  + a Publication is a class that describes all the generalized features of any Publication, you never instantiate one.
  + the fact that Publication is never intended to be instantiated

can be enforced by declaring it "abstract"

**public abstract class Publication {  
…  
}**

## **Overriding Methods**

* a subclass inherits the methods of its superclass
* a subclass can change the behaviour (ie; replace the superclass version of a method) of a superclasses methods by "overriding" the method
* example;
  + lets add a class called DiscMag which is a magazine with a disc attached to it.
  + when a new issue arrives, we want to check that the disc is attached
  + to implement this behaviour DiscMag should have its own version of the recvNewIssue method which adds a disc check before receiving the magazine.
  + overriding a method simply means to create a method in a

subclass which has the same signature of the superclass version of the method.

@Override

public void receiveNewIssue(Date newIssue) {

super.receiveNewIssue(newIssue);

System.out.println("Check that disc is attached.");

}

* NOTE\*\* we have 2 different methods implementing the same functionality (receive an issue) although, each, slightly differently. This is a part of what is called "polymorphism"

## **The "Object" class**

* every class inherits from (extends) the "Object" class, whether

explicitly stated or not.

* every class inherits "toString"
* toString automatically gets called whenever a String representation of an object is required, for example, when an object is passed to System.out.println.
* it’s common practise to override the toString() method so that you can customize the String representation of your object.

In magazine

@Override

public String toString() {

return super.toString()+"("+currIssue+")";

}

In book

@Override

public String toString() {

return super.toString()+" by "+ author;

}

In Publication

@Override

public String toString() {

return title;

}

