ENSF 593/594

8 – Class Relationships II

Inheritance

- Inheritance is a relationship among classes where a subclass inherits the structure and behavior of its superclass.
  - Defines the “is a” or generalization/specialization hierarchy.
  - Structure: instance variables.
  - Behavior: instance methods.

Inheritance (continued)

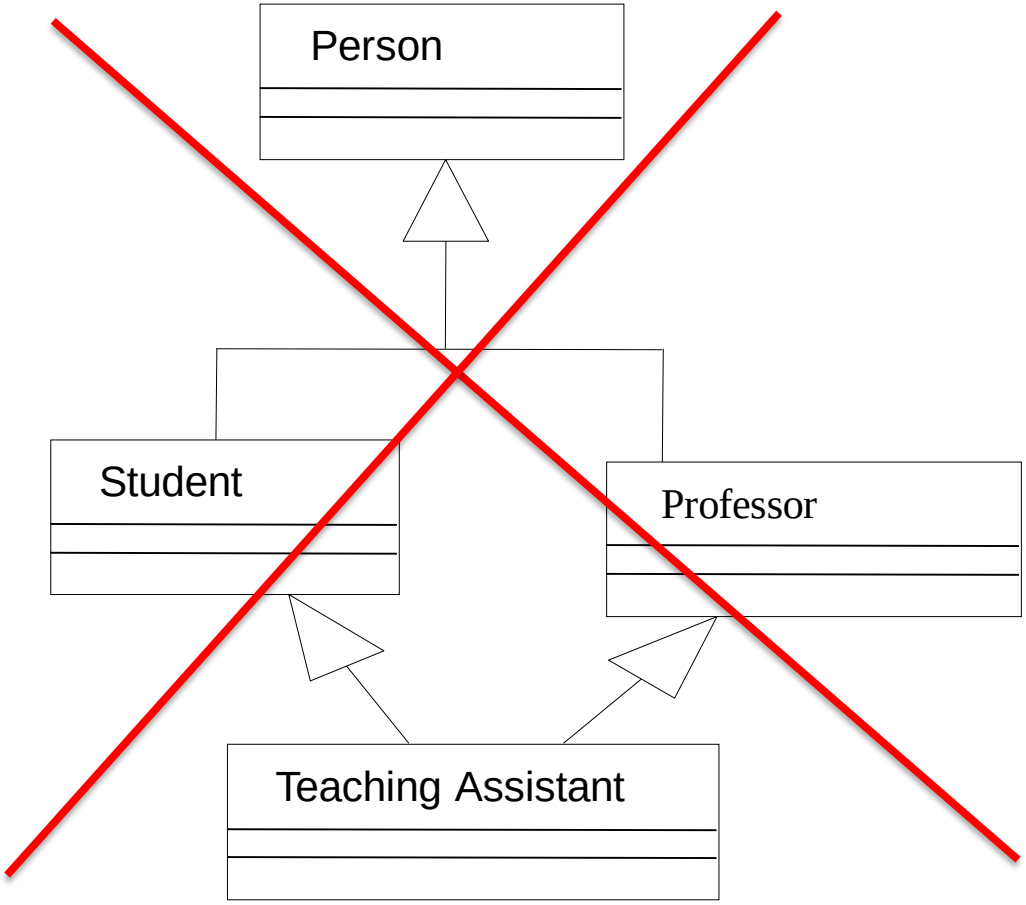
- Superclass instance variables are directly accessible to the subclass if they are declared **`protected`**.
  - If declared **`private`**, these variables can only be accessed by sending messages to `super`.

Inheritance (continued)

- There are **two types** of inheritance:
  - **Single inheritance:** any class has at most one superclass.
  - **Multiple Inheritance:** where a subclass may have more than one superclass.
- C++ supports multiple inheritance but Java does not.

Multiple Inheritance

- Java does NOT support multiple inheritance



Inheritance in Java

- Consider the following class Point:

```
public class Point {

        private int x, y;

        //more code here

}
```

Inheritance in Java (cont'd)

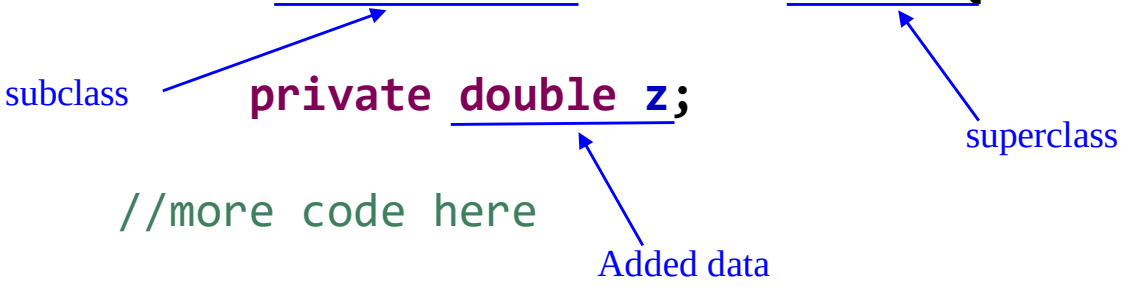
- Use the extend keyword to derive a class from an existing class:

```
class ThreeDPoint extends Point {

    private double z;

    //more code here

}
```



Implicit Subclassing

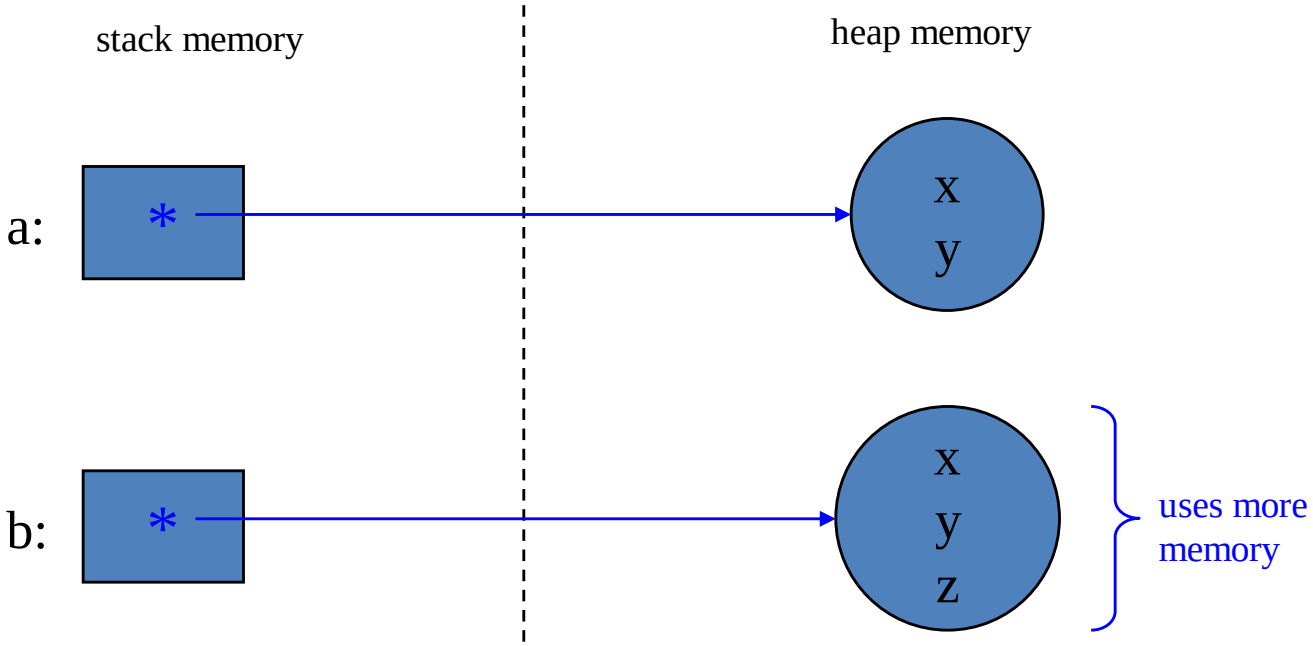
- If you don't use the extends keyword, the class implicitly **extends the class Object**.
- E.g. the following two declarations are **equivalent:**

```
public class Point {
}

public class Point extends Object {
}
```

Inheriting Structure – Example

```
Point a = new Point ();
ThreeDPoint b = new ThreeDPoint ();
```



Inheriting Behavior

- A subclass can invoke a method of any super class as long as, it has not been redefined.

Inheriting Behavior (continued)

```
class Point {
    protected double x, y;

    public double getx() {
        return x;
    }

    public double gety() {
        return y;
    }

    ...
```

Inheriting Behavior(continued)

```
    public double distance(Point p) {
        double xdiff = x - p.x;
        double ydiff = y - p.y;
        return Math.sqrt(xdiff * xdiff + ydiff * ydiff);
    }

    public void move(double deltaX, double deltaY) {
        x += deltaX;
        y += deltaY;
    }
}
```

Inheriting Behavior(continued)

```
class ThreeDPoint extends Point {
    private double z;

    public double getz() {
        return z;
    }

    public double distance(ThreeDPoint p) {
        double xdiff = x - p.x;
        double ydiff = y - p.y;
        double zdiff = z - p.z);
        return Math.sqrt(xdiff * xdiff +
                         ydiff * ydiff +
                         zdiff * zdiff);
    }
```

new method

replacement method

Inheriting Behavior(continued)

augmented method

```
public void move(double deltaX, double deltaY,
                 double deltaZ) {
    move(deltaX, deltaY);
    z += deltaZ;
}
```

code to customize the class's behavior

```
public static void main(String[] args) {
    ThreeDPoint a = new ThreeDPoint();
```

uses superclass's methods

```
    System.out.println("x = " + a.getX());
    System.out.println("y = " + a.getY());
    System.out.println("z = " + a.getZ());
}
}
```

Access when Extending

- If the instance variables in the superclass are private, you can only access them by using a superclass's accessor method.
- A protected instance variable is directly accessible to any subclasses.

Constructors in Extended Classes

- When constructing a subclass, you must invoke one of the superclass's constructors.
  - This ensures that the inherited structure is properly initialized.
- The superclass's constructor can be invoked explicitly using super().

- Example:

Constructors… (continued)

```
public class Point {
    protected double x, y;

    public Point(double xVal, double yVal) {
        x = xVal;
        y = yVal;
    }
}

public class ThreeDPoint extends Point {
    private double z;

    public ThreeDPoint(double xVal, double yVal, double zVal) {
        super(xVal, yVal);
        z = zVal;
    }
}
```

invokes Point's constructor

Constructors… (continued)

- super() must be the first statement in the constructor.
- If several constructors are used, at least one must invoke super(), and the others should use this().
- Example:

Constructors… (continued)

```
public class ThreeDPoint extends Point {
    private double z;

    public ThreeDPoint(double xVal, double yVal,
                       double zVal) {
        super(xVal, yVal);
        z = zVal;
    }

    public ThreeDPoint() {
        this(0.0, 0.0, 0.0);
    }
}
```

invokes the superclass's constructor

invokes the above constructor, which then invokes the superclass's constructor

Constructors… (continued)

- If the superclass doesn't explicitly define a constructor, and super() or this() are not used by the subclass, the superclass's default constructor is implicitly invoked at the beginning of the constructor.
  - Java provides a default no-arg constructor that invokes super() for you:

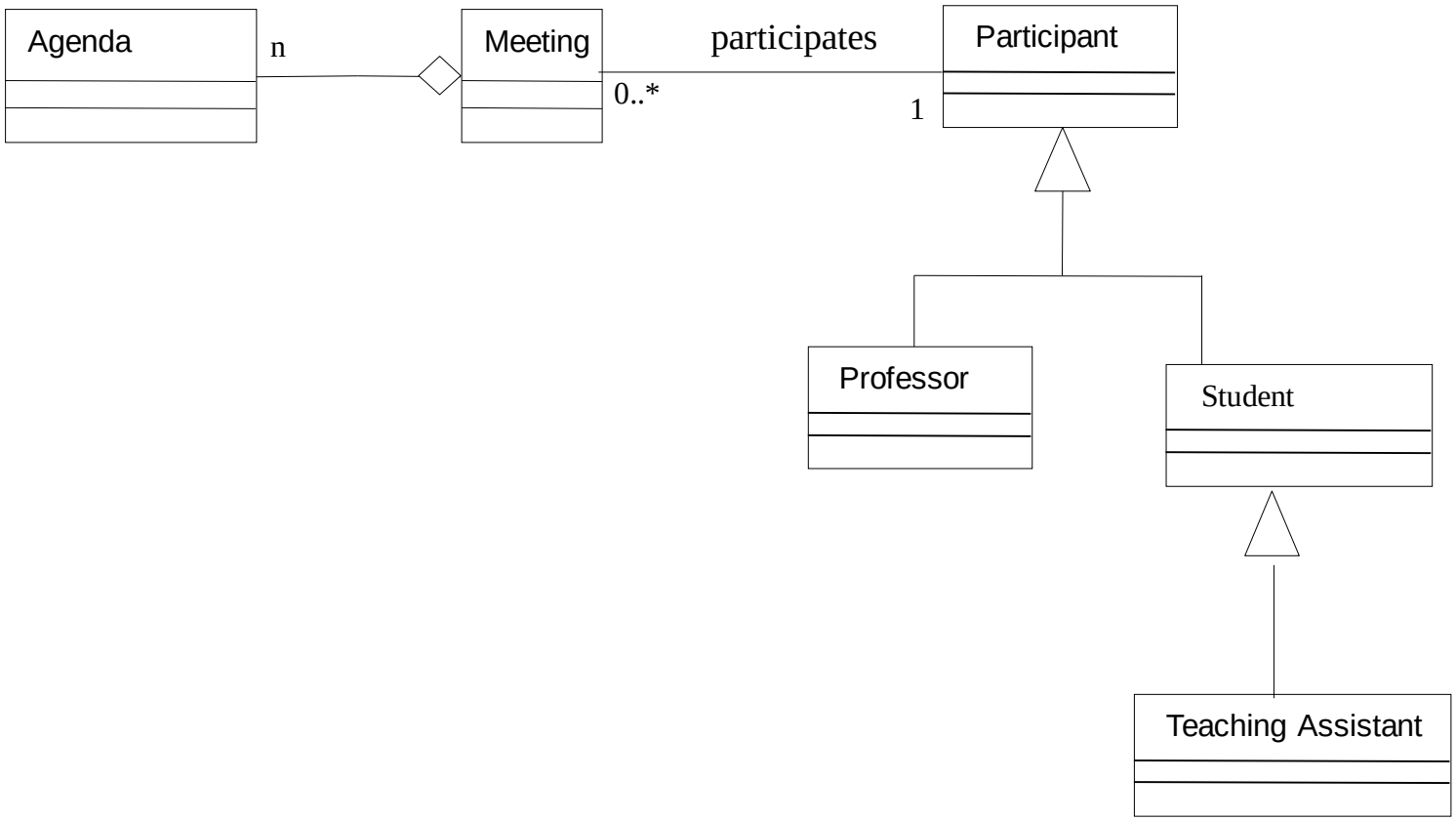
```
public class ExtendedClass extends SimpleClass {
    public ExtendedClass() {
        super();
    }
}
```

Class Exercise

Class Exercise

- Assume we want to design a meeting-scheduler software for the Faculty of Graduate Studies. You are responsible to design the following minimum requirements:
  - Students, Professors, and TAs should be able to participate in some of the meeting.
  - Each meeting may have several agenda

Class Diagram



What is Polymorphism

Polymorphism

- In Biology, the term polymorphism describes the characteristic of an element that may take on different forms
- In other words, the occurrence of more than one *form* or *morph*.
- Then, what does it mean in object-oriented programming?

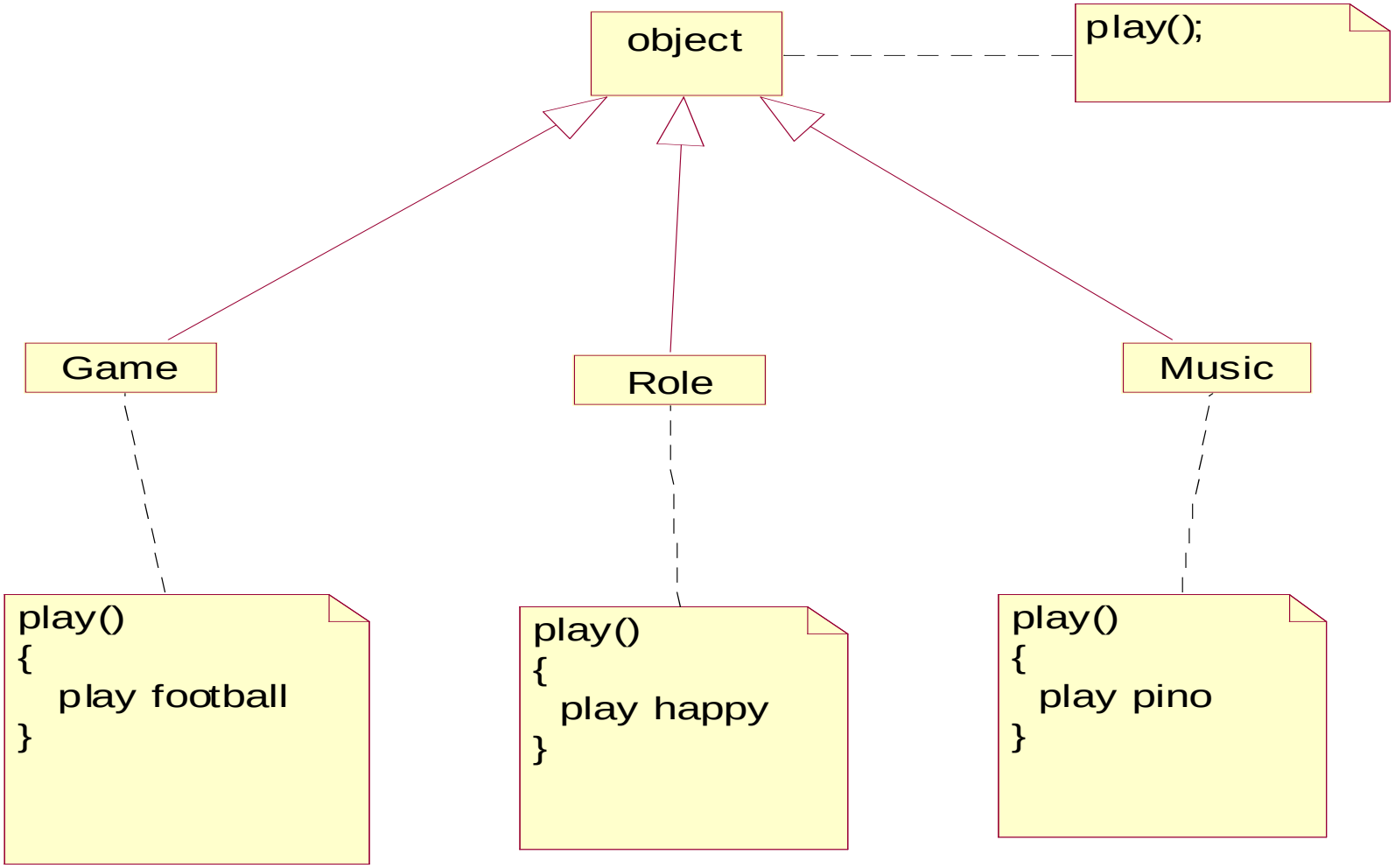
Polymorphism

- In in an O.O. language, polymorphism is the property that different objects may react to the same message.
  - –game.play()
  - –music.play()
  - –role.play

Polymorphism

- The concept of polymorphism works, when the classes are related in the inheritance tree; i.e. there is a common superclass where the method is defined.
- The subclasses *redefine* the method, so that behavior is specialized in some way.

Polymorphism



Abstract vs. Concrete Classes

- An abstract class cannot be instantiated.
  - The abstract superclass provides common structure and behavior. Subclasses are expected to add the necessary parts to make a useful class.
  - An abstract class may contain one or more abstract methods. The methods that do not have definitions.
- Example:

```
abstract class Shape { }
```

- Classes that are not abstract and their objects can be created are called concrete classes.

Final Classes

- A class marked final cannot be subclassed:

```
final class ClassName { }
```

- A final method cannot be redefined by any subclass.
- Final classes and methods are more secure, since their behavior cannot change.
- Final classes and methods can be optimized by the compiler.

Class Exercise

Scenario

- As part of a of a CAD application, you need to develop a few classes that represent shapes such as: Rectangle, Circle, Prism, etc..
  - Each shape must have a point of origin.
  - Each shape should know how to calculate its area, perimeter, and volume.
  - CAD should be able to use many shapes, and calculate the area, perimeter, or volume of any current or future shape in its method, calculator.