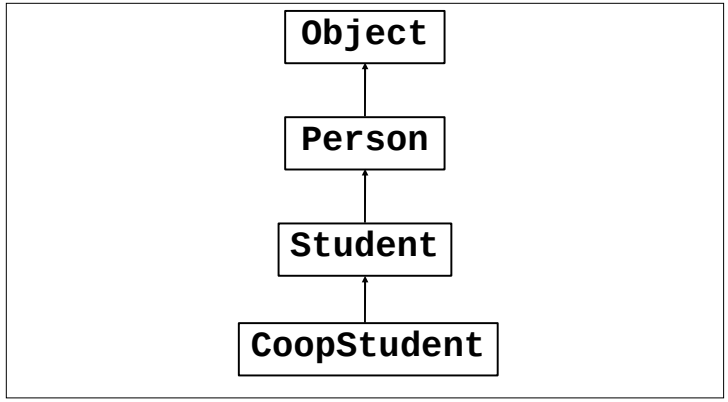
CS133: Developing Programming Principles

Lecture 16

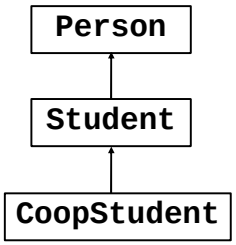
Types, more method resolution, casting

Recall from last day



Types

- An object of a derived class is also an object of **any** ancestor class.




```
CoopStudent marcus = new
  CoopStudent("Marcus",4321);
```

marcus **is a** `CoopStudent` object
marcus **is a** `Student` object
marcus **is a** `Person` object

Method resolution rules

```
Student kael =
   new CoopStudent("Kael", 4444);
```

The *reference (or static) type* is `Student`

The *object (or dynamic) type* is `CoopStudent`

The Rules:

- The reference type determines whether a method can be called.
- The object type determines which class's implementation of the method is called.

Types and method resolution

- Suppose we do the following:

```
Student kael = new
    CoopStudent("Kael", 4444);
```

- Remember that **`CoopStudent`**s are **`Student`**s
- What value is returned by **`kael.calcFees()`**?
- Answer: _____

Using method resolution

- Suppose we wish to keep track of all (4) UWaterloo students.
- Some may be coop, others would be “just” students.
- We would like to determine to total of all the fees for all these students.

Using method resolution (continued)

```
// array declaration and allocation

// fill array
// (Kael, Mike are in coop, Chantelle, Marcus are not)




// determine totalFees
```


Types and method resolution (continued)

- What about the CoopStudent method **`setWorkTerm`**, as in

```
kael.setWorkTerm(true);
```

- Answer: _____
- Why?

How to set the workterm?

- We need to tell Java to treat **`kael`** like a **`CoopStudent`**, not just a **`Student`**
- So, we can say:

Casting

- The same idea can be applied to numbers:

```
double mass = 3.00;
int massValue = mass;
```

Instead, we should say:

Question: What if **`mass = 3.67`** above?

Another inheritance example

- Recall the **`Coordinate`** class:

```
public class Coordinate
{ private int row;
  private int col;

  public Coordinate(int theRow, int theCol)
  { this.row = theRow;
    this.col = theCol;
  }
  public int getRow()
  { return this.row;
  }
  public int getCol()
  { return this.col;
  }
}
```


Augmenting `Coordinate`

- We could add some methods to **`Coordinate`**:

```
public double distance(Coordinate other)
{ int xDiff = this.col – other.col;
  int yDiff = this.row – other.row;
  int sqDist = xDiff * xDiff + yDiff * yDiff;
  return Math.sqrt(sqDist);
}
public String toString()
{ String s = "(" + this.row + "," + this.col + ")";
  return s;
}
public boolean equals(Coordinate other)
{ return (this.row == other.row) &&
         (this.col == other.col);
}
```

Extending the `Coordinate` class to include a colour

```
import java.awt.Color;
public class ColourCoordinate extends Coordinate
{ private static final Color DEFAULT_COLOUR =
                                        Color.BLACK;
  private Color colour;
  public ColourCoordinate(int row, int col,
                          Color aColour)
  { super(row, col);
    this.colour = aColour;
  }
  public ColourCoordinate(int rows, int cols)
  { this(row, col, DEFAULT_COLOUR);
  }
```


`ColourCoordinate` class continued

```
  public Color getColour()
  {
     return this.colour;
  }

  public void setColour(Color aColour)
  {
     this.colour = aColour;
  }

  public String toString()
  {

  }
```

`ColourCoordinate` class continued

```
  public boolean sameColour(
                   ColourCoordinate other)
  {
      return this.colour.equals(other.colour);
  }

  public boolean equals(ColourCoordinate other)
  {

  }
} // end class
```


Using the `ColourCoordinate` class: Valid or Invalid

```
Coordinate coord0 = new Coordinate(5, 4);
ColourCoordinate colour0 = new
         ColourCoordinate(0, 10, Color.RED);
Coordinate coord1 = colour0; _______________
ColourCoordinate colour1 = coord0;  _________

Object obj1 = colour0; ______________
Object obj2 = coord0; _______________

System.out.println(coord1.toString()); _____
System.out.println(colour0.toString()); ____
```

Using the `ColourCoordinate` class continued

```
ColourCoordinate colour2 = new
   ColourCoordinate(7, 6, Color.BLUE);

if (colour2.sameColour(colour0)) … _________
if (colour2.sameColour(coord0)) … __________

double d1 = coord1.distance(coord0); _______
double d2 = coord1.distance(colour0); ______

double d3 = colour0.distance(coord1); ______
double d4 = colour0.distance(colour2); _____
```


Summary

- Role of **`this`** and **`super`** in a derived (extended) class
- Resolving method calls
- Casting

ERROR: undefined
OFFENDING COMMAND:

STACK: