CS133: Developing Programming Principles

Lecture 15

More on Inheritance: `super`, `final`, method resolution, `Object`

The `CoopStudent` class

```
public class CoopStudent extends Student
{ private boolean isOnWorkTerm = false;
  private static final double COOP_FEE = 317.69;

  public CoopStudent(String newName, int sID)
  {  super(newName, sID);
  }
  public CoopStudent(String newName, String newAddress,
                     int sID)
  {  super(newName, newAddress, sID);
  }
  public double calcFees()
  {  return super.calcFees() + COOP_FEE;
  }
  // other methods omitted
}
```

Accessing the parent class

- Notice that we can use the constructor of **`Student`**
  - called **`super(newName, sID)`**
- Notice the method **`calcFees`** uses **`super`** in a different way
  - not as a constructor, but to indicate which class to look in for the (other) **`calcFees`** method

`super` as constructor

- Must be the first line in a constructor of a derived class.
- If it's not there, Java automatically inserts **`super()`** as the first line.
- Must be **`super(Arguments)`** if superclass constructor needs parameters.

Accessing the parent class continued

- Using **`super`** as the name of the object will use the method defined in the parent (base) class.
- Similar to **`this`**, but pretends it is an instance of the parent class.

The `calcFees` method of the `CoopStudent` class

So we can explain this method more fully:

```
public double calcFees()
{     return super.calcFees() + this.COOP_FEE;
}
```

Look in the super class

Call the **`calcFees`** method in the super class

`super` as object

- Can be used as an object name
  - **`super`**.***method(arguments)*** to access method in superclass
- When used as object name, may only be called once:
  - ***Can't do***

    `super.super.`*`method`*`(`*`arguments`*`)`.
- Only use **`super`** when overriding is involved.

Inheritance

- What if **`CoopStudent`** wants to access the method **`getName`**?
- **`getName`** is not in its (direct) parent class (i.e., **`Student`**) but it is in an ancestor class (i.e., **`Person`**).
- Example:

  Write a method **`upperName`** that prints the name of the **`CoopStudent`** in uppercase, one character per line.

upperName

```
public void upperName()
{




}
```


Inheritance of methods

- All **`public`** methods from parent classes are inherited.
- Child classes can override these **`public`** methods and also create new methods.
- **`private`** methods are not inherited, nor can they be accessed.

`private` methods not inherited

- Suppose **`Person`** had a **`private`** method, **`changeNameToFirstInitial`**.

```
public class Person
{  …
   private void changeNameToFirstInitial()
   {
     this.name = this.name.substr(0, 1);
   }
   …
}
```

- In **`Student`**, we cannot access this method, in the class definition or in a **`Student`** object.

Inheritance of instance variables

- Derived methods cannot access inherited **`private`** instance variables directly.
- Must use **`public`** mutator and accessor classes.
- Example:
  `name` variable in **`Person`** is accessible only by accessor/mutator methods.

final

- Recall that
  `final int myValue = 12;`
  makes **`myValue`** a constant that cannot be changed.

- We can use **`final`** in method signatures in a similar way.

Example

- In the **`Person`** class

```
public final void greeting()
{
  System.out.println("Hello World");
}
```

- No subclass (like **`Student`** or **`CoopStudent`**) can override this method!

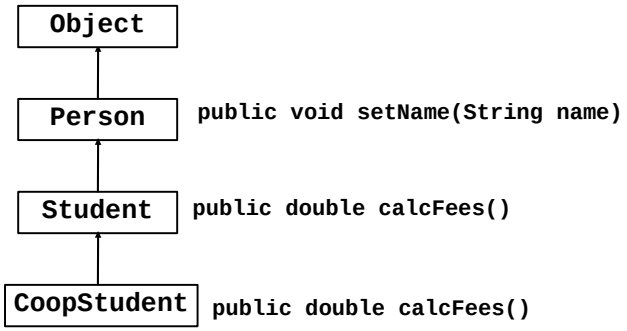
Resolving method calls

```
CoopStudent mike = new
          CoopStudent("Mike", 123456789);
mike.setName("Michael");
mike.getName();
```

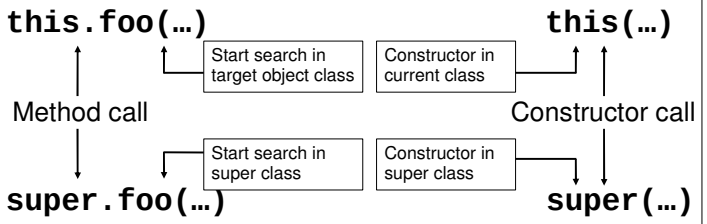
- How does the computer know where to find the methods to perform?
  - > Class hierarchy

Resolving method calls continued

```
mike.setName("Michael");
```



`this` vs. `super`



Note: **`this(…)`**, **`super(…)`** can only occur in constructor.

Note:**`this.foo(…)`** and **`super.foo(…)`** can occur in any method of the derived class.

The class `Object`

- Every class in Java extends **`Object`** by default.
- Provides the basic functionality you see in any object of any type.
- No need to extend it explicitly.

`Object` methods

- There are two methods in the **`Object`** class that we care about
  - **`toString()`** and **`equals(Object other)`**
- We typically override these methods with more meaningful functionality
- Example: add **`toString`** to the **`Student`** class

Summary

- Accessing parent methods
- **`private`** methods are not inherited
- **`final`** modifier for methods
- Resolving method calls
- The **`Object`** class

ERROR: undefined
OFFENDING COMMAND:

STACK: