CS133: Developing Programming Principles

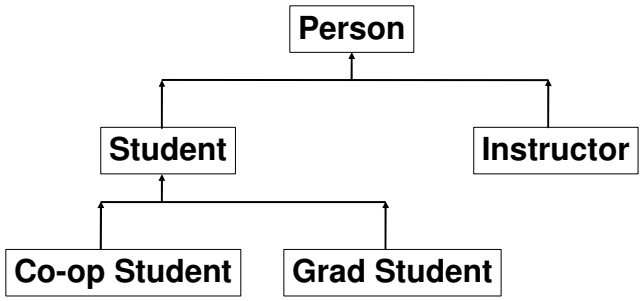
Lecture 14

Class hierarchies, super/subclasses, calling superclass constructors, overriding methods

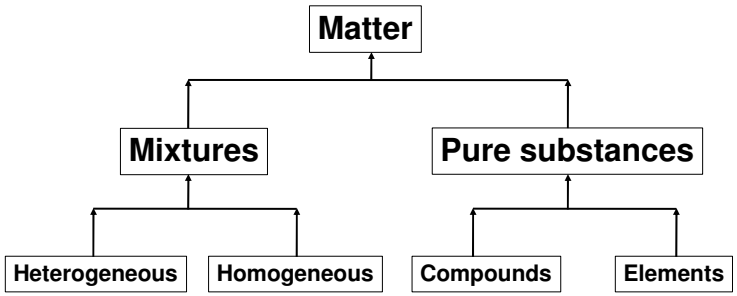
Hierarchies

- Information may be organized in a non-linear manner.
- Hierarchies represent dependencies or relationships in a non-linear, tree-like structure.
- Examples:
  - People/Students/Co-op students
  - Classification of matter

Example



Another example



Observations

- Notice that the root of the hierarchy is the most general (Person, Matter)
- As we move down through the tree, we become more specific
  - The attributes become increasingly well-defined.
  - Example: Co-op Student is more specific than Student, which is more specific than Person.

Observations continued

- Attributes are passed down the tree
  - Example: if person has an address, then students will also have an address.
- The CS/Java/O-O hierarchies will guarantee the above point
  - In “real life”, not all attributes are inherited

How does this relate to CS?

- We will treat these **classes** of things as **objects**, and relate them together using

  INHERITANCE

Person

- Suppose we know that all people have a name and an address.
- We can do the following to the name and address:
  - Set
  - Compare
  - Retrieve

The `Person` class

```
public class Person
{ private String name;
  private String address;

  public Person(String newName)
  {    this.name = newName;
       this.address = "";
  }
  public Person(String newName,
                String newAddress)
  {    this.name = newName;
       this.address = newAddress;
  }
```


The `Person` class

```
 public String getName()
  {    return this.name;
  }
  public void setName(String newName)
  {    this.name = newName;
  }
  public boolean sameName(Person other)
  {    return
           this.name.equalsIgnoreCase(
                                other.name);
  }
// Also getAddress and setAddress and
// sameAddress
```

`Person` as Superclass

- **`Person`** is called the **superclass** (or **base class or parent class**).
- This is the most **general** class.
- From this class, **subclasses** (or **derived classes** or **child classes**) can inherit properties (e.g., methods and attributes) from the super class.

`Person` continued

- Using the University hierarchy again, we will create a subclass of **`Person`** called **`Student`**.
- Students are people, except they have a student id, take courses and, of course, pay fees.

The `Student` class

```
public class Student extends Person
{ private int studentID;
  private String[] courses = new String[6];
  private int numCourses = 0;
  private static final double PER_COURSE_FEE = 256.74;

  public Student(String newName, int sID)
  { super(newName);
    this.studentID = sID;
  }
  public Student(String newName, String newAddress,
                 int sID)
  { super(newName, newAddress);
    this.studentID = sID;
  }
```

Indicates this is a subclass of Person

Call the constructor of the super class

Call the second constructor of the super class

The `Student` class continued

```
  public boolean equals(Student other)
  {
    return this.studentID == other.studentID &&
           this.sameName(other) &&
           this.sameAddress(other);
  }

  public String getStudentID()
  {
    return this.studentID;
  }

  public double calcFees()
  {
    return this.numCourses*PER_COURSE_FEE;
  }
```

`Student` class continued

super(*Arguments*)

- Calls the constructor of the immediate superclass if there is one that can accept the arguments.
  - Must be first statement in constructor
  - Otherwise, super() is inserted automatically
- Otherwise, there is a compile-time error.
- Here, **`Person`**'s constructors are called.

`Student` class continued

- A **`Student`** object will have a name, as well as the **`getName`** and **`setName`** methods even though they aren't explicitly defined in the class.
- **Inherits** them from the **`Person`** class.
- Notice that a **`Student`** object has access to the instance variables of other **`Student`** objects (e.g., for **`equals`**).

Example

```
> Student troy = new Student("Troy", 20000001);
> Student sandy = new Student("Sandy", "200 King St", 99);
> System.out.println(troy.getName());
  ___________________
> System.out.println(troy.calcFees());
  ___________________
> troy.setName("Troy Vasiga");
> System.out.println(troy.sameName(sandy));
  ___________________
> sandy.addCourse("CS 000");
> System.out.println(sandy.calcFees());
  ___________________
> sandy.getStudentID();
  ___________________
```

Overriding methods

- Sometimes when we specialize (i.e., create subclasses), we may want to restrict or modify the methods of the parent class for this particular subclass.
- So we may **override** a method in a subclass using the same signature as the super class method, but replacing the body of the method.

Co-op Student

- Suppose we extend the **`Student`** class to a subclass Co-op student.
  - Co-op students may be on a work term, or on-campus
  - Co-op students pay an additional Co-op fee

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;
  }
```

Override **`calcFees`** from **`Student`** class

The `CoopStudent` class

```
  public void setWorkTerm(boolean status)
  {  this.isOnWorkTerm = status;
  }
  public String location()
  { if (isOnWorkTerm) {
       return this.getName() +
          " is on a work term.";
    } else {
       return this.getName()+
          " is on campus.";
    }
  }
}
```

Example: `CoopStudent`

```
> CoopStudent chantelle = new CoopStudent("Chantelle",
                                                  5555555);
> System.out.println(chantelle.getName());
  ____________________
> chantelle.onWorkTerm(true);
> System.out.println(chantelle.calcFees());
  ____________________
> chantelle.addCourse("ECE 476");
> chantelle.onWorkTerm(false);
> System.out.println(chantelle.calcFees());
  ____________________
> System.out.println(chantelle.location());
```


Terminology

- A base class is also known as a **parent class** or the **superclass**.
- A derived class is also known as a **child class** or the **subclass**.
- An **ancestor class** is a class that is a parent or parent of a parent (etc.).
- If class A is an ancestor of class B, then class B is a **descendant class** of class A.

Terminology

- X **is a** Y

  means class X extends Y (or extends an earlier extension of Y).

- X **has a** Y

  means class X has an instance variable of type Y.

Definitions

- Overriding is when a method with the same signature as in a superclass is written in a subclass.
  - Example: **`CoopStudent`**'s **`calcFees`** overrides **`Student`**'s **`calcFees`**
- Overloading is when there are at least two different methods with the same name and return type, but with different types or numbers of parameters.
  - Example: constructor in **`Student`** is overloaded

Summary

- Inheritance
- Superclass, subclass
- Hierarchy
- **`super(…)`**
- Overriding methods

ERROR: undefined
OFFENDING COMMAND:

STACK: