Chapter 5

Inheritance (2)

Agenda

- Inheritance
  - The class Object
  - ”Is a “ Versus “Has a”
  - getClass Method
  - instanceOf Operator
  - How to Define equals Method

The class **Object**

- In Java, every class is a descendent of the class ***Object***
  - Every class has **Object** as its ancestor
  - Every object of every class is of type **Object**, as well as being of the type of its own class
- If a class is defined that is not explicitly a derived class of another class, it is still automatically a derived class of the class **Object**

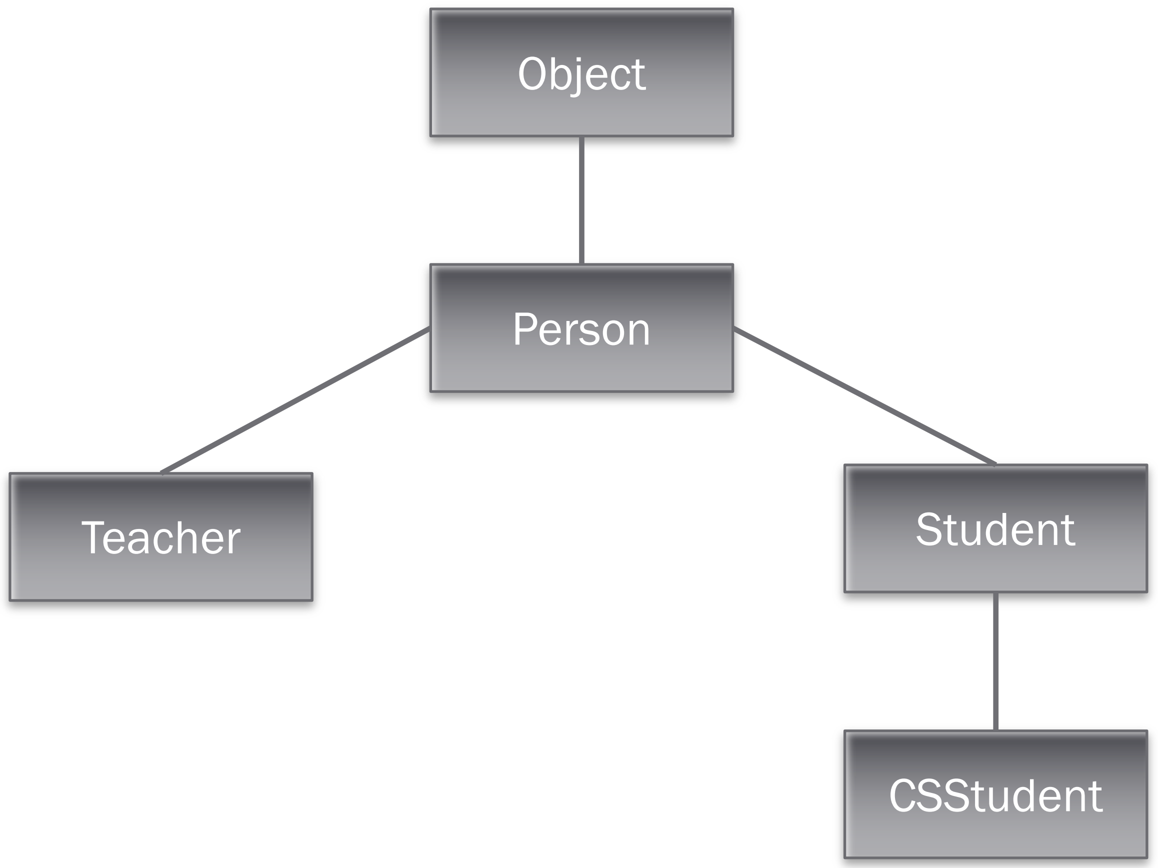
The class **Object**

- The class **Object** is in the package **java.lang** which is always imported automatically
- Having an **Object** class enables methods to be written with a parameter of type **Object**
  - A parameter of type **Object** can be replaced by an object of any class whatsoever
  - For example, some library methods accept an argument of type **Object** so they can be used with an argument that is an object of any class

The class **Object**

- The class **Object** has some methods that every Java class inherits
  - For example, the **equals** and **toString** methods
- Every object inherits these methods from some ancestor class
  - Either the class **Object** itself, or a class that itself inherited these methods (ultimately) from the class **Object**
- However, these inherited methods should be overridden with definitions more appropriate to a given class
  - Some Java library classes assume that every class has its own version of such methods

Class Hierarchy – Previous Example



Tip: "Is a" Versus "Has a"

- A derived class demonstrates an *"is a"* relationship between it and its base class
  - Forming an "is a" relationship is one way to make a more complex class out of a simpler class
  - For example, a **Teacher** "***is a***" **Person**
  - **Teacher** is a more complex class compared to the more general **Person** class

Tip: "Is a" Versus "Has a"

- Another way to make a more complex class out of a simpler class is through a *"has a"* relationship
  - This type of relationship, called *composition*, occurs when a class contains an instance variable of a class type
  - The **Teacher** class could contain an instance variable, **hireDate**, of the class **Date**, so therefore, a **Teacher** "***has a***" **Date.**

```
public class Teacher {
    Date hireDate;
}
```

```
public class Date {
    int day, month, year;
}
```

Object.getClass Method

- Return the runtime class of an object.
- If we have two classes (Teacher and Student) that extend the class Person, then:

```
Person p = new Teacher();
System.out.println(p.getClass()); // class Teacher

Person p = new Student();
System.out.println(p.getClass()); // class Student
```

instanceOf

- The **java instanceof operator** is used to test whether the object is an instance of the specified type (class or subclass or interface). It will return **true** *if it is a descendent of that class.*

- If we have two classes (Teacher and Student) that extend the class Person, then:

```
Person p = new Teacher();
System.out.println( p instanceOf Person ); // true

Person p = new Teacher();
System.out.println( p instanceOf Teacher ); // true

Person p = new Teacher();
System.out.println( p instanceOf Student ); // false
```

Define **equals** Method for Teacher

```
■ public class Person {
    protected String name;
  }

■ public class Teacher extends Person {
    protected String salary;

    public boolean equals(Teacher otherTeacher) {
      return name.equals(otherTeacher.name)
                    && salary == other.salary;
    }
  }
```

The Right Way to Define **equals**

- Since the **equals** method is always inherited from the class **Object**, methods like the following simply overload it:

```
public boolean equals(Teacher otherTeacher)
{ . . .}
```

- However, this method should be overridden, not just overloaded:

```
public boolean equals(Object otherObject)
{ . . .}
```

The original **equals** Method for the Class **Employee**

-
  ```
  public boolean equals(Teacher    otherTeacher)
    {
       return (name.equals(otherTeacher.name)
                   && salary == otherTeacher.salary);
    }
  ```

- ==============================

The right way to define equals:

```
public boolean equals(Object otherObject)
{ . . . }
```

The Right Way to Define **equals**

- The overridden version of **equals** must meet the following conditions
  - The parameter **otherObject** of type **Object** must be type cast to the given class (e.g., **Teacher)**
  - However, the new method should only do this if **otherObject** really is an object of that class, and if **otherObject** is not equal to **null**
  - Finally, it should compare each of the instance variables of both objects

A Better equals Method for the class Teacher

```
public boolean equals(Object otherObject)
  {
     if(otherObject == null)
        return false;

     else if(getClass( ) != otherObject.getClass( ))
        return false;

     else
     {
        Teacher otherTeacher = (Teacher)otherObject;
           return (name.equals(otherTeacher.name)
              && salary == otherTeacher.salary));
     }
  }
```

Note:

```
//   else if(!(OtherObject instanceof Teacher))
                    return false; . . .
```

getClass Versus instanceOf

- Here is an example using the class **Teacher**

```
if ( otherObject == null )
        return false;

else if ( getClass( ) != otherObject.getClass( ) )
        return false;

. . . //excerpt from bad equals method
else if ( ! ( OtherObject  instanceof   Teacher ) )
        return false; . . .
```

getClass Versus instanceOf

- Here is an example using the class **Teacher**

```
. . . //excerpt from bad equals method
else if(!(OtherObject  instanceof  Teacher))
   return false; . . .
```

- Now consider the following:

```
Person p = new Person("Joe");
Teacher t = new Teacher("Joe", 4586);

boolean testT = p.equals(t);// return True
boolean testP = t.equals(p);// return False
```

getClass Versus instanceOf

- **testT** will be **true**, because **t** is a **Teacher** with the same name as **p**
- However, **testP** will be **false**, because **p** is not a **Teacher**, and cannot be compared to **t**
- Note that this problem would not occur if the **getClass()** method were used instead, as in the previous **equals** method example

getClass Versus instanceOf

- Both the **instanceof** operator and the **getClass()** method can be used to check the class of an object
- However, the **getClass()** method is more exact
  - The **instanceof** operator simply tests the class of an object
  - The **getClass()** method used in a test with **==** or **!=** tests if two objects *were created with* the same class

+

Thanks!