8.2 Programming with inheritance

- Constructors in Derived Classes
- Call to an Overridden Method
- Assignment Compatibility
- "Is a" and "Has a" Relationships
- The Class Object
- toString()
- A Better equals Method

Example of Adding Constructor in a Derived Class: **`Student`**

```
public class Student extends Person
{
    private int studentNumber;
    public Student()
    {
        super();
        studentNumber = 0;
    }
…
```

- Two new constructors (one on next slide)
  - » default initializes attribute `studentNumber` to 0
- *`super`* must be [redacted] action in a constructor definition
  - » Included automatically by Java if it is not there
  - » *`super()`* calls the parent [redacted] constructor

Example of Adding Constructor in a Derived Class: **`Student`**

- Passes parameter `newName` to constructor of parent class
- Uses second parameter to initialize instance variable that is not in parent class.

```
public class Student extends Person
{
...
    public Student(String newName, int newStudentNumber)
    {
        super(newName);  // Person(newName);illegal
        studentNumber = newStudentNumber;
    }
...
```

Super()

- If you do not include a call to the base-class constructor, Java will automatically include a call to the [redacted] constructor of the base class as the first action of any constructor for a derived class.
  - » equivalent

```
    public Student()
    {
        super( );
        studentNumber = 0;
    }
=======================
    public Student()
    {
        studentNumber = 0;
    }
```

More about Constructors in a Derived Class

- Constructors can call other [redacted]
- Use **`super`** to invoke a constructor in [redacted] class
  - » as shown on the previous slide
- Use **`this`** to invoke a constructor [redacted] the class
  - » shown on the next slide
- Whichever is used must be the [redacted] action taken by the constructor
- Only one of them can be first, so if you want to invoke both:
  - » Use a call with `this` to call a constructor with `super`

Example of a constructor using

- `Student` class has a constructor with two parameters: `String` for the `name` attribute and `int` for the `studentNumber` attribute

```
public Student(String newName, int newStudentNumber)
  {
      super(newName);
      studentNumber = newStudentNumber;
  }
```

Another constructor within `Student` takes just a `String` argument and initializes the `studentNumber` attribute to a value of 0:

» calls the constructor with two arguments, initialName (`String`) and 0 (`int`), within the same class

```
public Student(String initialName)
{
    this(initialName, 0);
}
```

Call to an Overridden Method

- Use `super` to call a method in the parent class that was [redacted] (redefined) in the derived class
- Example: `Student` redefined the method `writeOutput` of its parent class, `Person`
- Could use `super.writeOutput()` to invoke the overridden (parent) method

```
public void writeOutput()
{
    super.writeOutput();
    System.out.println("Student Number : "
                                        studentNumber);
}
```

```
    public int getStudentNumber( )
    {
        return studentNumber;
    }
    public void setStudentNumber(int newStudentNumber)
    {
        studentNumber = newStudentNumber;
    }

    public void writeOutput( )  ///
    {
        System.out.println("Name: " + getName( ));
        System.out.println("Student Number: " + studentNumber);
    }
    public boolean equals(Student otherStudent)  //
    {
        return (this.sameName(otherStudent)
            && (this.studentNumber == otherStudent.studentNumber));
    }

    public String toString( )
            {
                return("Name: " + getName( )
                            + "\nStudent number: "
                            + studentNumber);
    }
}
```

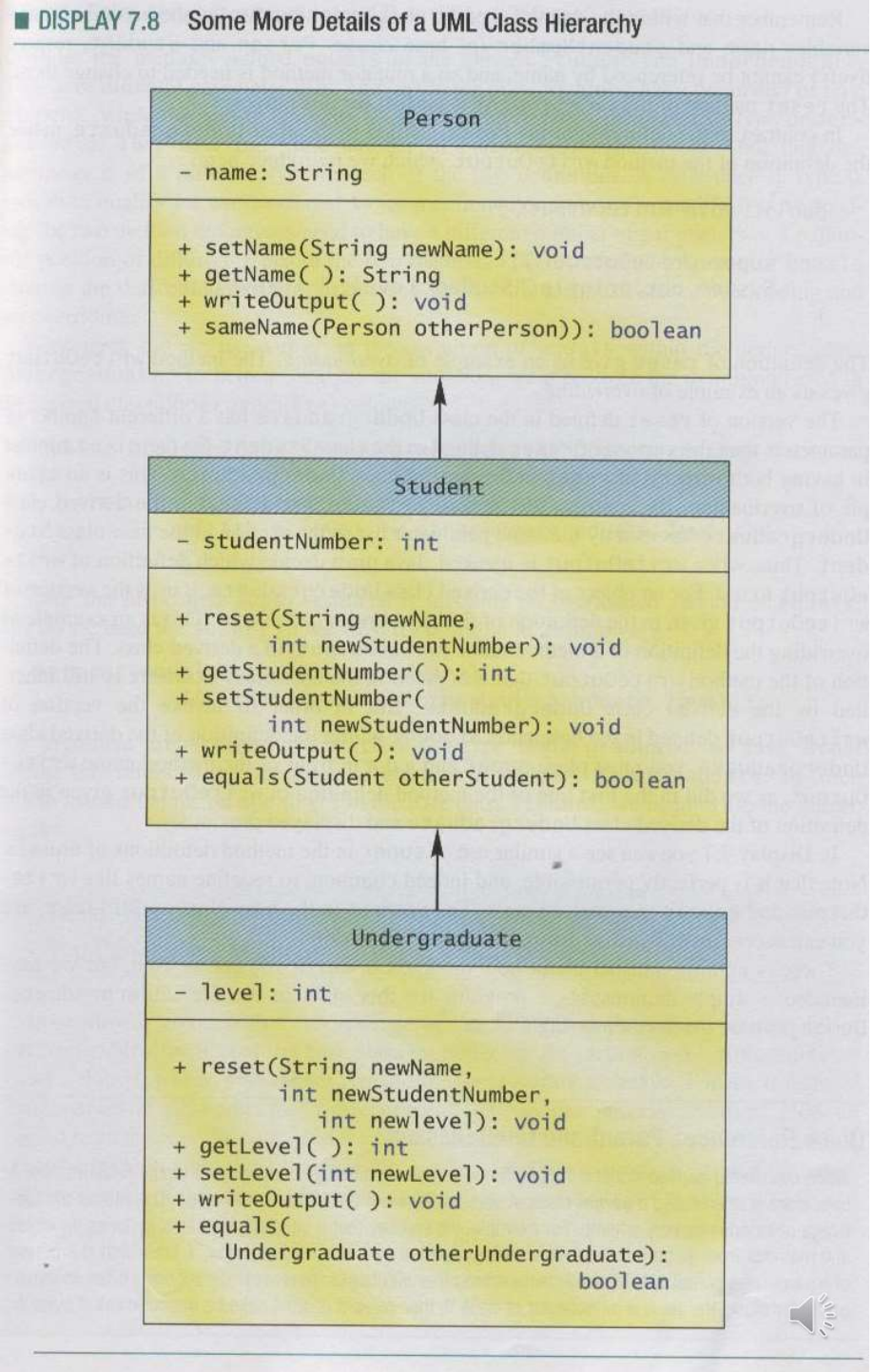
Listing 8.4 A Derived Class of a Derived Class - Undergraduate.java

```
// Listing 8.4 A Derived Class of a Derived Class

public class Undergraduate extends Student
{
   private int level; //1 for freshman, 2 for sophomore,
                      //3 for junior, or 4 for senior.

   public Undergraduate( )
   {
      super( );  //
      level = 1;
   }

   public Undergraduate(String initialName,
                 int initialStudentNumber, int initialLevel)
   {
      super(initialName, initialStudentNumber);   //
      setLevel(initialLevel); //Checks 1 <= initialLevel <= 4
   }
```

```
public void reset(String newName,
                  int newStudentNumber, int newLevel)
  {
        // invokes the method named reset in the base class Student
    reset(newName, newStudentNumber);
    setLevel(newLevel); //Checks 1 <= newLevel <= 4
  }
  public int getLevel( )
  {
    return level;
  }
  public void setLevel(int newLevel)
  {
    if ((1 <= newLevel) && (newLevel <= 4))
      level = newLevel;
    else
    {
      System.out.println("Illegal level!");
      System.exit(0);
    }
  }
  public void writeOutput( )  //
  {
    super.writeOutput( ); //
    System.out.println("Student Level: " + level);
  }
```

```
public boolean equals(Undergraduate otherUndergraduate)
   {
      return equals((Student)otherUndergraduate) &&
            (this.level == otherUndergraduate.level);
   }
/*  // Alternate version
   public boolean equals(Undergraduate otherUndergraduate)
   {
      return super.equals(otherUndergraduate) &&
            (this.level == otherUndergraduate.level);
   }
*/}
```

Figure 8.4 Some more Details of a UML class Hierarchy-Undergraduatedemo.java



You cannot use multiple supers

- You cannot repeat the use of super to invoke a method from some ancestor class other than a direct parent.

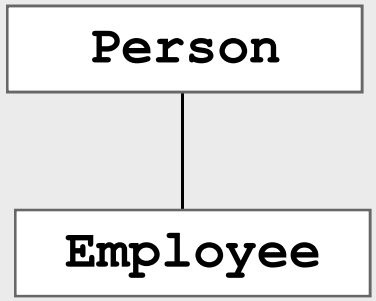
- Ex) super.super.writeOutput(); //ILLEGAL!!

What is the "Type" of a Derived class?

- Derived classes have more than one type
  - » they have the type of the derived class (the class they define)
  - » They also have the type of every ancestor class
    - – all the way to the top of the class hierarchy
  - » Every instances of derived classes is also an object of the ancestor class

```
Person josephine;
Employee boss = new Employee();
```

```
Person josephine;
Person boss = new Employee();
```



`Person` is the parent class of `Employee` in this example.

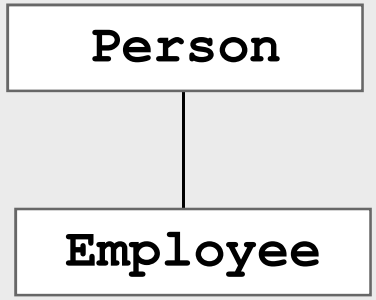
Assignment Compatibility

- **Can** assign an object of a derived class to a variable of any ancestor type

```
Person josephine;
Employee boss = new Employee();
josephine = boss;
```

- Can **not** assign an object of an ancestor class to a variable of a derived class type

```
Person josephine = new Person();
Employee boss;
boss = josephine;
```



`Person` is the parent class of `Employee` in this example.

An object Can have More than One Type

```
public class SomeClass
{
        public static void compareNumbers(Student s1, Student s2)
        {
                if (s1.getStudentNumber() == s2.getStudentNumber())
                        System.out.println(s1.getName()
                                + " has the same number as  "
                                + s2.getName());
                else
                        System.out.println(s1.getName()
                                + " has the different number as  "
                                + s2.getName());
        }
}
```

```
import java.util.Scanner;

public class UseSomeClass
{
   public static void main(String[] args)
   {
                    Scanner keyboard = new Scanner(System.in);
                    Student studentObject = new Student("Jane Doe", 1234);
                    System.out.println("Enter Name: ");
                    String undergradName = keyboard.next();
                    System.out.println("Enter student number : ");
                    int undergradStudentNumber = keyboard.nextInt();
                    Undergraduate undergradObject =
                              new
Undergraduate(undergradName,undergradStudentNumber,1);

                    SomeClass.compareNumbers(studentObject,
undergradObject);
          }
}
```

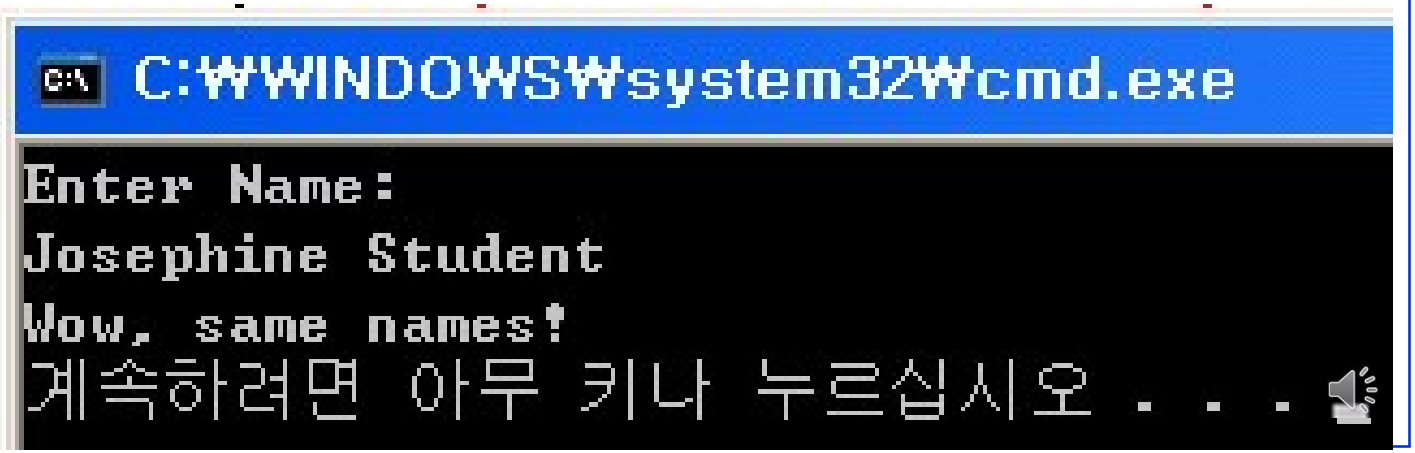
C:₩WINDOWS₩system32₩cmd.exe

```
Enter Name:
KKK
Enter student number :
1234
Jane Doe has the same number as  KKK
계속하려면 아무 키나 누르십시오 . . .
```

C:₩WINDOWS₩system32₩cmd.exe

```
Enter Name:
kkk
Enter student number :
1111
Jane Doe has the different number as  kkk
계속하려면 아무 키나 누르십시오 . . .
```

```
import java.util.Scanner;
public class UseSomeClass2
{
  public static void main(String[] args)
  {
            Scanner keyboard = new Scanner(System.in);
            Person joePerson = new Person("Josephine Student");
            System.out.println("Enter Name: ");
            String newName = keyboard.nextLine();

            Undergraduate someUndergrad =
                    new Undergraduate(newName,222,3);
            if (joePerson.hasSameName(someUndergrad))
                    System.out.println("Wow, same names! ");
            else
                    System.out.println("Different names ");
        }
}
```



```
public class Person
{
   private String name;

   public Person( )
   {
      name = "No name yet";
   }

   public Person(String initialName)
   {
      name = initialName;
   }

   public void setName(String newName)
   {
      name = newName;
   }

   public String getName( )
   {
      return name;
   }

   public void writeOutput( )
   {
      System.out.println("Name: " + name);
   }

   public boolean hasSameName(Person otherPerson)
   {
      return this.name.equalsIgnoreCase(otherPerson.name);
   }
}
```

```
public class UseSomeClass3
{
   public static void main(String[] args)
   {
                    Person p1, p2;
                    Student s = new Student();
                    Undergraduate ug = new Undergraduate();
                    p1 = s;
                    p2 = ug;

                    p1 = new Student();
                    p2 = new Undergraduate();

                    Student s2 = new Person();
                    Undergraduate ug2 = new Person();
                    Undergraduate ug3 = new Student();
         }
}
```

```
D:\My Documents\@@@@@@jv\ch07\UseSomeClass3.java:14:
                    Student s2 = new Person();
                      ^
D:\My Documents\@@@@@@jv\ch07\UseSomeClass3.java:15:
                    Undergraduate ug2 = new Person();
                            ^
D:\My Documents\@@@@@@jv\ch07\UseSomeClass3.java:16:
                    Undergraduate ug3 = new Student();
                            ^
3 errors

Tool completed with exit code 1
```

```
public class UseSomeClass4
{
   public static void main(String[] args)
   {
                    Undergraduate ug, ug2;
                    Person p = new Person();
                    ug = p;
                    Student s = new Student();
                    ug2= s;
          }
}
```

```
D:\My Documents\@@@@@@jv\ch07\UseSomeClass4.java:7:
incompatible types
found   : Person
required: Undergraduate
                    ug = p;
                ^
D:\My Documents\@@@@@@jv\ch07\UseSomeClass4.java:9:
incompatible types
found   : Student
required: Undergraduate
                    ug2= s;
                ^
2 errors

Tool completed with exit code 1
```

"Is a" and "Has a" Relationships

- Inheritance is useful for "is a" relationships.
  - » A student "is a" person.
  - » `Student` inherits from `Person`.

- Inheritance is usually **not** useful for "has a" relationships.
  - » A student "has a(n)" enrollment date.
  - » Add a `Date` object as an instance variable of `Student` instead of having `Student` inherit from `Date`.

- If it makes sense to say that an object of `Class1` "is a(n)" object of `Class2`, then consider using inheritance.

The Class Object

- Object Class as
  - » a class that is an ancestor of every class
  - » *All* classes derive from the original, predefined class `Object`
  - » `Object` is called the *Eve* class since it is the original class for all other classes

- The Class Object does have some methods that every Java class inherits.
  - » ex) equals()
  - » ex) toString()
  - » ex) clone()

toString()

```
// A Derived Class

public class Student extends Person
{
   private int studentNumber;
   .........
   public String toString( )
   {
            return("Name: " + getName( )
                        + "\nStudent number: "
                        + studentNumber);
   }
}
```

```
public class ToStringTest
{
   public static void main(String[] args)
   {
                    Student s = new Student("Joe Student", 2001);
                    System.out.println(s.toString());
                    System.out.println(s);
         }
}
```

```
C:WWINDOWSWsystem32Wcmd.exe
Name: Joe Student
Student number: 2001
Name: Joe Student
Student number: 2001
계속하려면 아무 키나 누르십시오 . .
```

A Better equals Method listing 8.5

- Listing 8.2 Student Class
  - » public boolean equals(Student otherStudent)
- Object Class.
  - » public boolean equals(Object otherObject)
- studentPar.equals(objectPar)
  - » Object objectPar 이라면
  - » ➔ invoke equals() method of Object Class (Error!!!)
  - » ➔ Java will use the definition of equals defined for the equals method **defined for the class Object.**

A subtle point about overloading and overriding

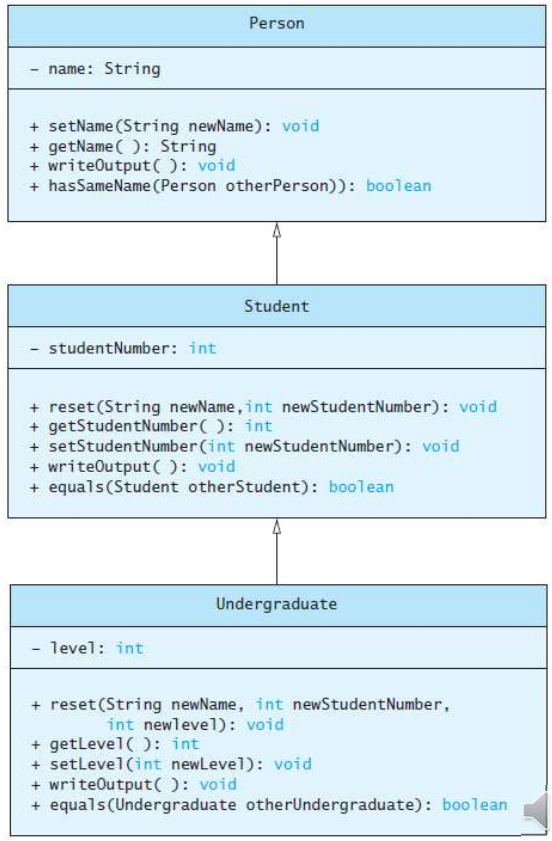
- Ex) Equals() of Undergraduate Class & Equals() of Student Class
  - » different parameter list (same number of parameters, different type)
    - – parameter of Equals of Student Class : Student Class
    - – parameter of Equals of Undergraduate Class : Undergraduate Class
  - » in some technical sense, overloading and not overriding
  - » Why did we use super in the definition of equals ?

```
public boolean equals(Undergraduate otherUndergraduate)
  {
      return (super.equals(otherUndergraduate)
           && (this.level == otherUndergraduate.level));
  }
```

// Listing 8.5. A better equals Method for the Class Student

```
public boolean equals (Object otherObject)
{
    boolean isEqual = false;
    if ((otherObject != null) &&
            (otherObject instanceof Student))
    {
        Student otherStudent = (Student) otherObject;
        isEqual = this.sameName (otherStudent) &&
            (this.studentNumber ==
                otherStudent.studentNumber);
    }

    return isEqual;
}
```

FIGURE 8.4 More Details of the UML Class Hierarchy Shown in Figure 8.2



끝