Chapter 8

●Inheritance

- 8.1 Inheritance Basics
- 8.2 Programming with Inheritance
- 8.3 Polymorphism
- 8.4 Interfaces and Abstract Classes
- 8.5 Graphics Supplement (Optional)

Objectives

- 1) Become familiar with inheritance in general
- 2) Learn how to define and use derived classes in Java
- 3) Learn about dynamic binding and polymorphism in general in Java.

Principles of OOP

- OOP - Object-Oriented Programming
- Principles discussed in previous chapters:
  - » Information Hiding
  - » Encapsulation
- In this chapter
  - » Inheritance

8.1 Inheritance Basics : Why OOP?

- 1)To try to deal with the [redacted] of programs
- 2)To apply [redacted] to simplify the tasks of writing, testing, maintaining and understanding complex programs
- 3) To increase [redacted]
  - » to reuse classes developed for one application in other applications instead of writing new programs from scratch ("Why reinvent the wheel?")
- Inheritance is a major technique for realizing these objectives

Inheritance Overview

- Inheritance allows you to define a very general class then later define more specialized classes by adding new detail
  - » the general class is called
- The specialized classes ***inherit*** all the properties of the general class
  - » specialized classes are ***derived*** from the base class
  - » they are called
- After the general class is developed you only have to write the "difference" or "specialization" code for each derived class
- A *class hierarchy:* classes can be derived from derived classes (child classes can be parent classes)
  - » any class higher in the hierarchy is an *ancestor class*
  - » any class lower in the hierarchy is a *descendent class*

An Example of Inheritance: a **`Person`** Class

The base class: Listing 8.1

- Constructors:
  - » a default constructor
  - » one that initializes the `name` attribute (instance variable)
- Accessor methods:
  - » `setName` to change the value of the `name` attribute
  - » `getName` to read the value of the `name` attribute
  - » `writeOutput` to display the value of the `name` attribute
- One other class method:
  - » `sameName` to compare the values of the `name` attributes for objects of the class
- Note: the methods are `public` and the `name` attribute `private`

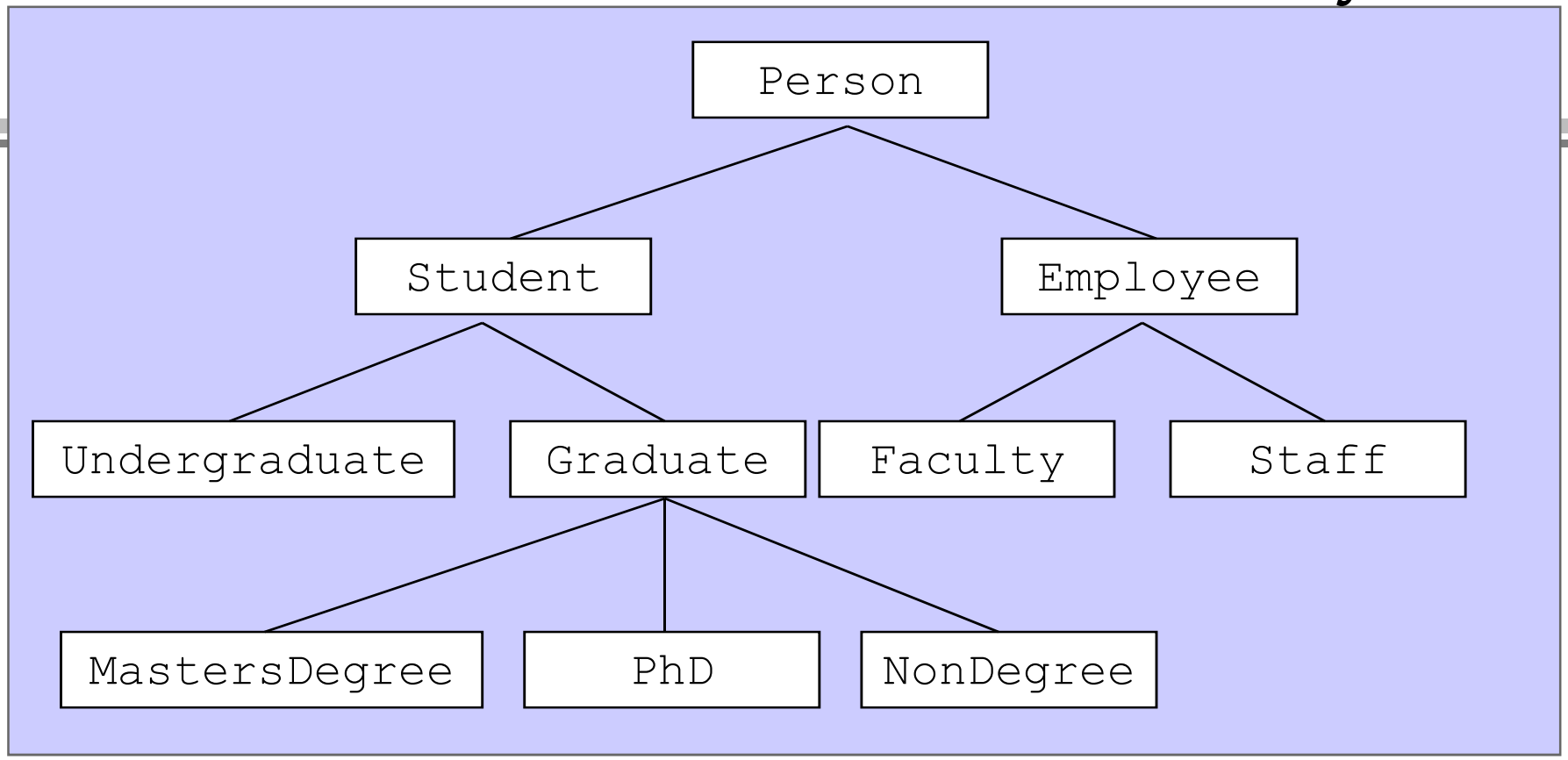
A `Person` Base Class

Listing 8.1

```
// The base class: Listing 8.1

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
sameName(Person otherPerson)
   {
      return
(this.name.equalsIgnoreCase(other
Person.name));
   }

}
```

Derived Classes: a Class Hierarchy



- The base class can be used to implement specialized classes
  - » For example: student, employee, faculty, and staff
- **Derived class :** Classes can be derived from the classes derived from the base class, etc., resulting in a *class hierarchy*

Derived Classes

```
public class Student          Person
```

- The keyword **extends** in first line indicates inheritance.
  - » Creates **derived class `Student`** from base class `Person`
- A derived class has the instance variables and methods of the base class that it extends.
  - » The `Person` class has a `name` instance variable so the `Student` class will also have a `name` instance variable.
  - » Can call the `setName` method with a `Student` object even though **`setName` is defined in `Person`** and not in `Student`:

```
Student s = new Student();
s.setName("Warren Peace");
```

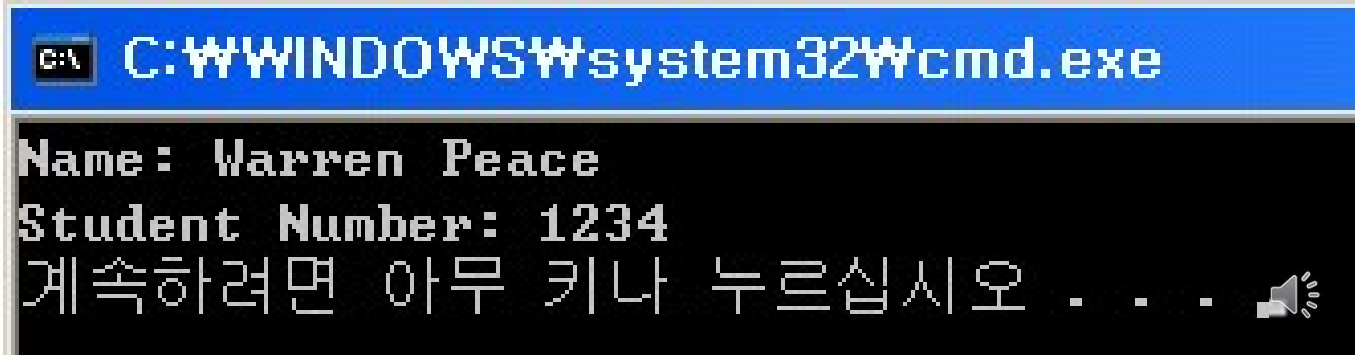
Listing 8.2 A Derived Class Student.java

```
// Listing 8.2 A Derived Class

public class Student extends Person
{
   private int studentNumber;

   public Student( )
   {
      super( );
      studentNumber = 0;//Indicating no number yet
   }
   public Student(String initialName, int initialStudentNumber)
   {
      super(initialName);
      studentNumber = initialStudentNumber;
   }

   public void reset(String newName, int newStudentNumber)
   {
      setName(newName);
      studentNumber = newStudentNumber;
   }
```

```
    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 8.3 demonstrating inheritance

```
// Listing 8.3 Demonstrating Inheritance


public class InheritanceDemo
{
   public static void main(String[] args)
   {
      Student s = new Student( );

      s.setName("Warren Peace");
                  // setName is inherited from the class Person
      s.setStudentNumber(1234);
      s.writeOutput( ); //@@
   }
}
```



Extending the Base Class

- A derived class **can add instance variables and/or methods** to those it inherits from its base class.
- Note that an instance variable for the student number has been added
  - » `Student` has this attribute in addition to `name`, which is inherited from `Person`

```
private int studentNumber;
```

- Student also adds several methods that are not in Person:
  - » `reset`, `getStudentNumber`, `setStudentNumber`, `writeOutput`, `equals`, and some constructors

Overriding Methods

- When a child class has a method that with the same signature as the parent class, the method in the child class ***overrides*** the one in the parent class.
- This is *overriding*, not overloading.
- Example:
  - » Both `Person` and `Student` have **a `writeOutput`** method with no parameters (same signature).
  - » When `writeOutput` is called with a `Student` calling object, the `writeOutput` in `Student` will be used, not the one in `Person`.

```
// The base class: Listing 8.1

public class Person
{
    private String name;


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

```
// Listing 8.2 A Derived Class

public class Student extends Person
{
    private int studentNumber;

    public void writeOutput( )  ///
    {
        System.out.println("Name: " + getName( ));
        System.out.println("Student Number: " + studentNumber);
    }
    …….
}
```

Overriding Versus Overloading

Overriding	Overloading
• Same method name	• Same method name
• signature • One method in ancestor, one in descendant	• signature • Both methods can be in class

The `final` Modifier

- Specifies that a method definition cannot with a new definition in a derived class
- Example:

```
public final void specialMethod()
{
...
```

- Used in specification of some methods in standard libraries
- Allows the compiler to generate more efficient code
- Can also declare an entire class to be final, which means it cannot be used as a base class to derive another class

`private` & `public` Instance Variables and Methods( )

- `private` instance variables from the parent class are not available by name in derived classes
  - » "Information Hiding" says they should not be
  - » use accessor methods to change them, e.g. `reset` for a `Student` object to change the `name` attribute

- `private` methods are **not** inherited!
  - » use `public` to allow methods to be inherited
  - » only helper methods should be declared `private`

Use of Private Instance Variables from the Base Class

- Valid Definition

```
public void reset (String newName, int newStudentNumber)
{
        setName(newName);
        studentNumber = newStudentNumber
}
```

- illigal definition

```
public void reset (String newName, int newStudentNumber)
{
        name = newName; //illegal !!
        studentNumber = newStudentNumber
}
```

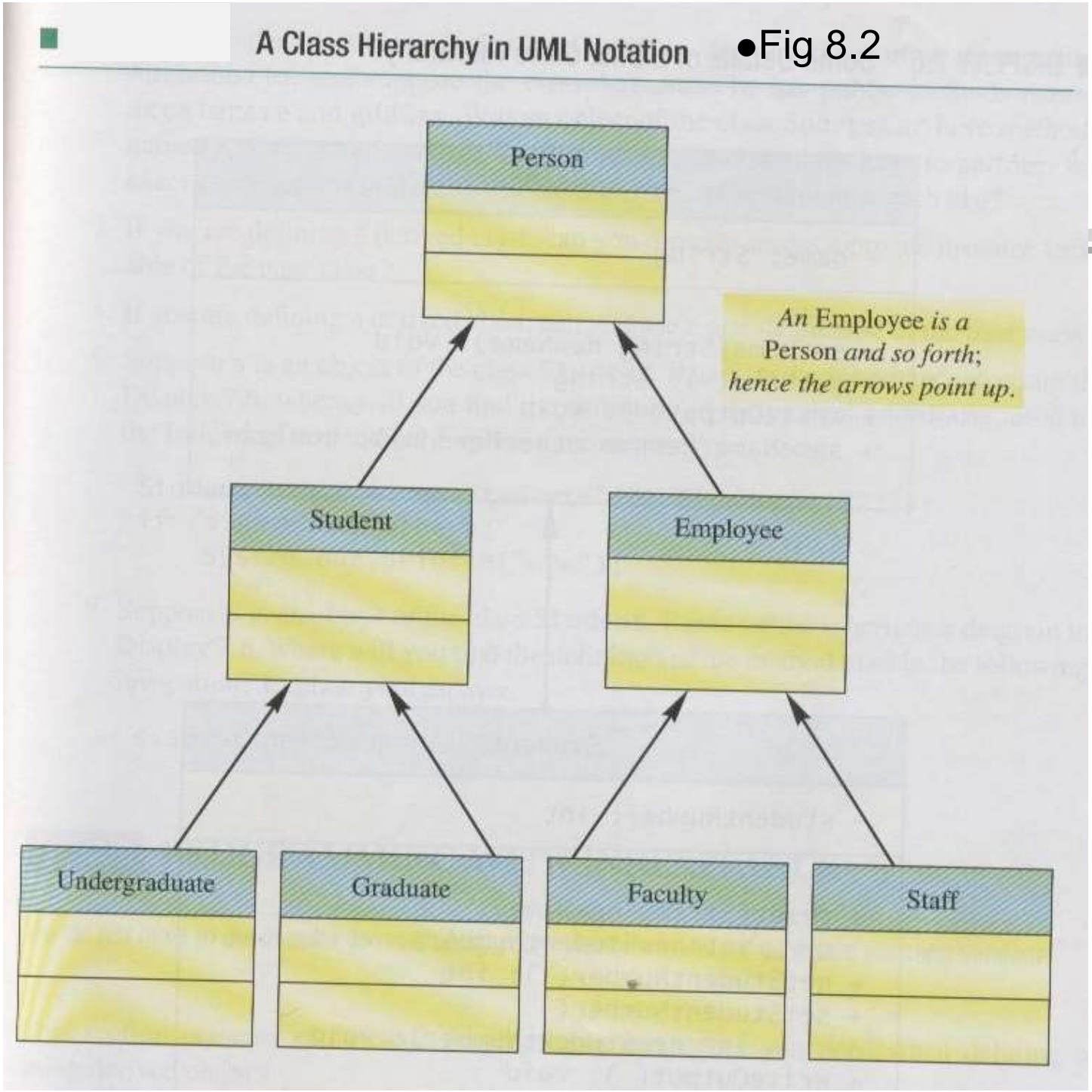
- private instance variable of a base class cannot be accessed in the definition of a method of a derived class.

UML inheritance Diagrams

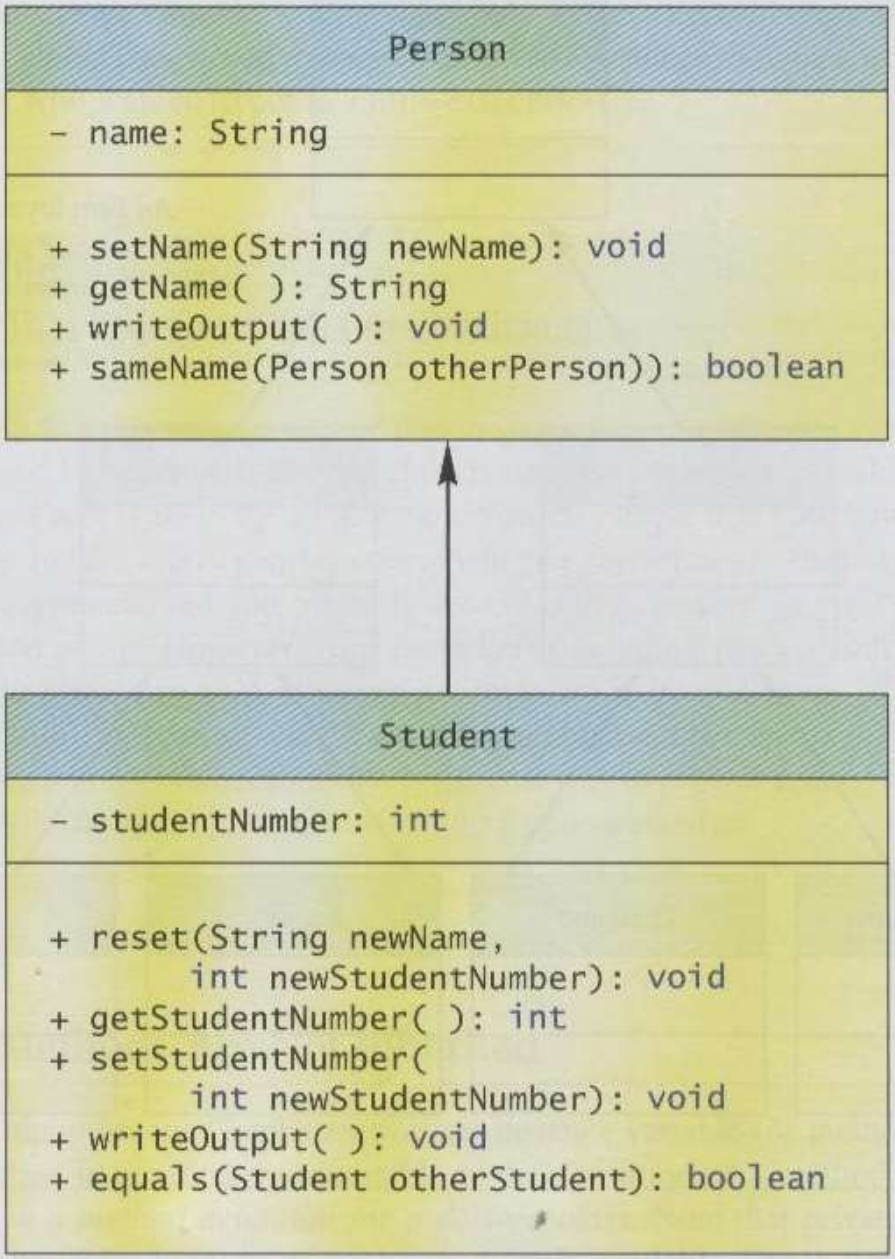
- Arrawheads
  - » relationship
  - » Helping in locating method definitions

A Class Hierarchy in UML Notation

●Fig 8.2



Some Details of a UML Class Hierarchy • Fig 8.3



- S.writeoutput();
- S.reset("…");

➔ in Student Class

- S.setName(..);

➔ in Person Class