ITP20003 Java Programming

# Class Inheritance and Polymorphism

(Chapter 8)

This slide is primary taken from the instructor's resource of Java: Introduction to Problem Solving and Programming, 7th ed. by Savitch and then edited partly by Shin Hong



hongshin@handong.edu

# What if ?

- Construct various lists such as a linked list, a doubly linked list, an array list, a list with a binary tree and then use them in various purposes
- Repeat the Tangram lab with 10 different kinds of figures
- Create linked lists to hold Integer, Long, Double, and various class objects
- Write classes for list, sequence, and queue

*Software Engineering is about multi-person development of multi-version programs* - David L. Parnas

# How to Manage Similar Classes Systematically? (1/2)

- Share interface
  - Declare a generic type by grouping the common interfaces, and declare each class as a *subtype* of the genetic type
    - E.g., every list has methods add(), get(), remove()
  - A common interface is implemented variously across subtypes
    - i.e., Polymorphism (c.f. morph = form)
  - Use a class with its generic interface, while not caring out its implementation

# How to Manage Similar Classes Systematically? (2/2)

- Share implementation
  - Define a new class by extending an existing class definition
    - Reuse the existing code for defining a new one
      - i.e., Class Inheritance
    - Allow a new class to have additional aspects
  - Reduce redundancy in program code



# Class Hierarchy in Java

- A class can be declared as a child of an existing class
  - A child class extends a parent class
    - Parent class and Child class, or Superclass and Subclass (derived class)
  - In Java, a class can have at most one parent class
    - There can be multiple ancestor classes (e.g.., grand parent, grand-grand parent)

- A child class inherits both interface and implementation of its parent
  - Subtype: an object of a child class IS A object of its parent class
  - Reuse: a child object has all fields and methods of its parent class
    - except private members
  - Overriding: an object may have its own definition of its interface method

# Inheritance Basics

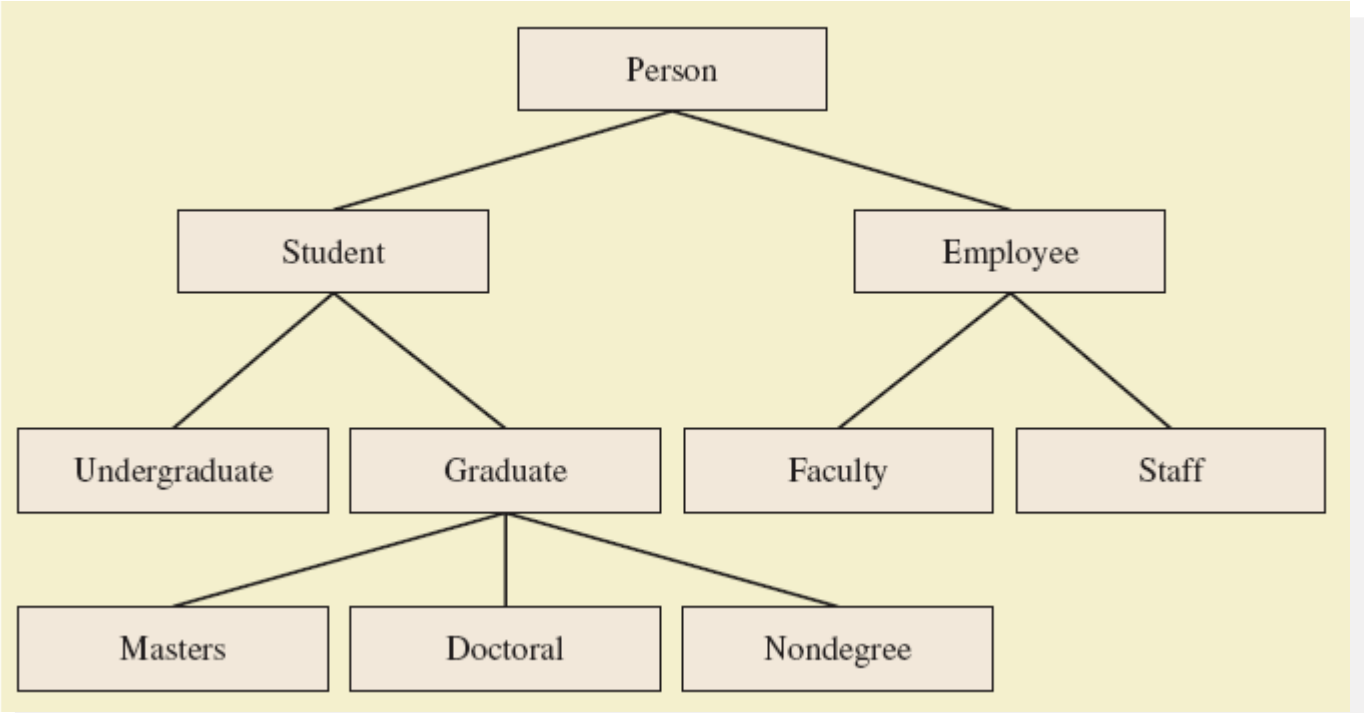
- Inheritance allows programmer to define a general class
- Later you define a more specific class
  - Adds new details to general definition
- New class inherits all properties of initial, general class
- View example class, listing 8.1
  **`class Person`**

http://cs.smu.ca/~porter/csc/228/notes/PowerPoints7thEd


*JAVA: An Introduction to Problem Solving & Programming, 7*th Ed. By Walter Savitch
ISBN 0133862119 

# Derived Classes

- Figure 8.1 A class hierarchy





*JAVA: An Introduction to Problem Solving & Programming, 7*th Ed. By Walter Savitch
ISBN 0133862119 

# Derived Classes

- Class **`Person`** used as a *base* class
  - Also called *superclass*
- Now we declare *derived* class **`Student`**
  - Also called *subclass*
  - Inherits methods from the superclass
- View derived class, listing 8.2
  **`class Student extends Person`**
- View demo program, listing 8.3
  **`class InheritanceDemo`**

Sample screen output

```
Name: Warren Peace
Student Number: 1234
```

# Overriding Method Definitions

- Note method **`writeOutput`** in class **`Student`**
  - Class Person also has method with that name
- Method in subclass with same signature overrides method from base class
  - Overriding method is the one used for objects of the derived class
- Overriding method must return same type of value


*JAVA: An Introduction to Problem Solving & Programming, 7*th Ed. By Walter Savitch

# Overriding Versus Overloading

- Do not confuse overriding with overloading
  - Overriding takes place in subclass – new method with same signature
- Overloading
  - New method in same class with different signature

# The `final` Modifier

- Possible to specify that a method cannot be overridden in subclass
- Add modifier final to the heading

```
public final void specialMethod()
```

- An entire class may be declared `final`
  - Thus cannot be used as a base class to derive any other class


*JAVA: An Introduction to Problem Solving & Programming, 7*th Ed. By Walter Savitch

# Private Instance Variables, Methods

- Consider private instance variable in a base class
  - It is not inherited in subclass
  - It can be manipulated only by public accessor, modifier methods
- Similarly, private methods in a superclass not inherited by subclass

# Constructors in Derived Classes

- A derived class does not inherit constructors from base class
  - Constructor in a subclass must invoke constructor from base class
- Use the reserve word **`super`**

  - Must be first action in the constructor

```
public Student(String initialName, int initialStudentNumber)
{
    super(initialName);
    studentNumber = initialStudentNumber;
}
```


*JAVA: An Introduction to Problem Solving & Programming, 7*th Ed. By Walter Savitch
ISBN 0133862119 

# The `this` Method – Again

- Also possible to use the **`this`** keyword
  - Use to call any constructor in the class

```
public Person()
{
    this("No name yet");
}
```

- When used in a constructor, this calls constructor in same class
  - Contrast use of **`super`** which invokes constructor of base class


*JAVA: An Introduction to Problem Solving & Programming, 7*th Ed. By Walter Savitch

# Calling an Overridden Method

- Reserved word **`super`** can also be used to call method in overridden method

```
public void writeOutput()
{
    super.writeOutput(); //Display the name
    System.out.println("Student Number: " + studentNumber);
}
```

- Calls method by same name in base class


*JAVA: An Introduction to Problem Solving & Programming, 7*th Ed. By Walter Savitch
ISBN 0133862119 

# Programming Example

- A derived class of a derived class
- View sample class, listing 8.4
  **`class Undergraduate`**
- Has all public members of both
  - **`Person`**
  - **`Student`**
- This reuses the code in superclasses

# Type Compatibility

- In the class hierarchy
  - Each **`Undergraduate`** is also a **`Student`**
  - Each **`Student`** is also a **`Person`**
- An object of a derived class can serve as an object of the base class
  - Note this is not typecasting
- An object of a class can be referenced by a variable of an ancestor type


*JAVA: An Introduction to Problem Solving & Programming, 7*th Ed. By Walter Savitch
ISBN 0133862119 

# Type Compatibility

- Be aware of the "is-a" relationship
  - A **`Student`** *is a* **`Person`**
- Another relationship is the "has-a"
  - A class can contain (as an instance variable) an object of another type
  - If we specify a date of birth variable for **`Person`** – it "has-a" **`Date`** object


*JAVA: An Introduction to Problem Solving & Programming, 7*th Ed. By Walter Savitch

# The Class `Object`

- Java has a class that is the ultimate ancestor of every class
  - The class `Object`
- Thus possible to write a method with parameter of type `Object`
  - Actual parameter in the call can be object of any type
- Example: method
  `println(Object theObject)`


*JAVA: An Introduction to Problem Solving & Programming, 7*th Ed. By Walter Savitch
ISBN 0133862119 

# The Class `Object`

- Class Object has some methods that every Java class inherits
- Examples
  - Method `equals`
  - Method `toString`
- Method `toString` called when `println(theObject)` invoked
  - Best to define your own `toString` to handle this


*JAVA: An Introduction to Problem Solving & Programming, 7*th Ed. By Walter Savitch

# A Better `equals` Method

- Programmer of a class should override method equals from `Object`
- View code of sample override, listing 8.8

```
public boolean equals
        (Object theObject)
```


*JAVA: An Introduction to Problem Solving & Programming, 7*th Ed. By Walter Savitch
ISBN 0133862119 

# Polymorphism

- Inheritance allows you to define a base class and derive classes from the base class
- Polymorphism allows you to make changes in the method definition for the derived classes and have those changes apply to methods written in the base class

# Polymorphism

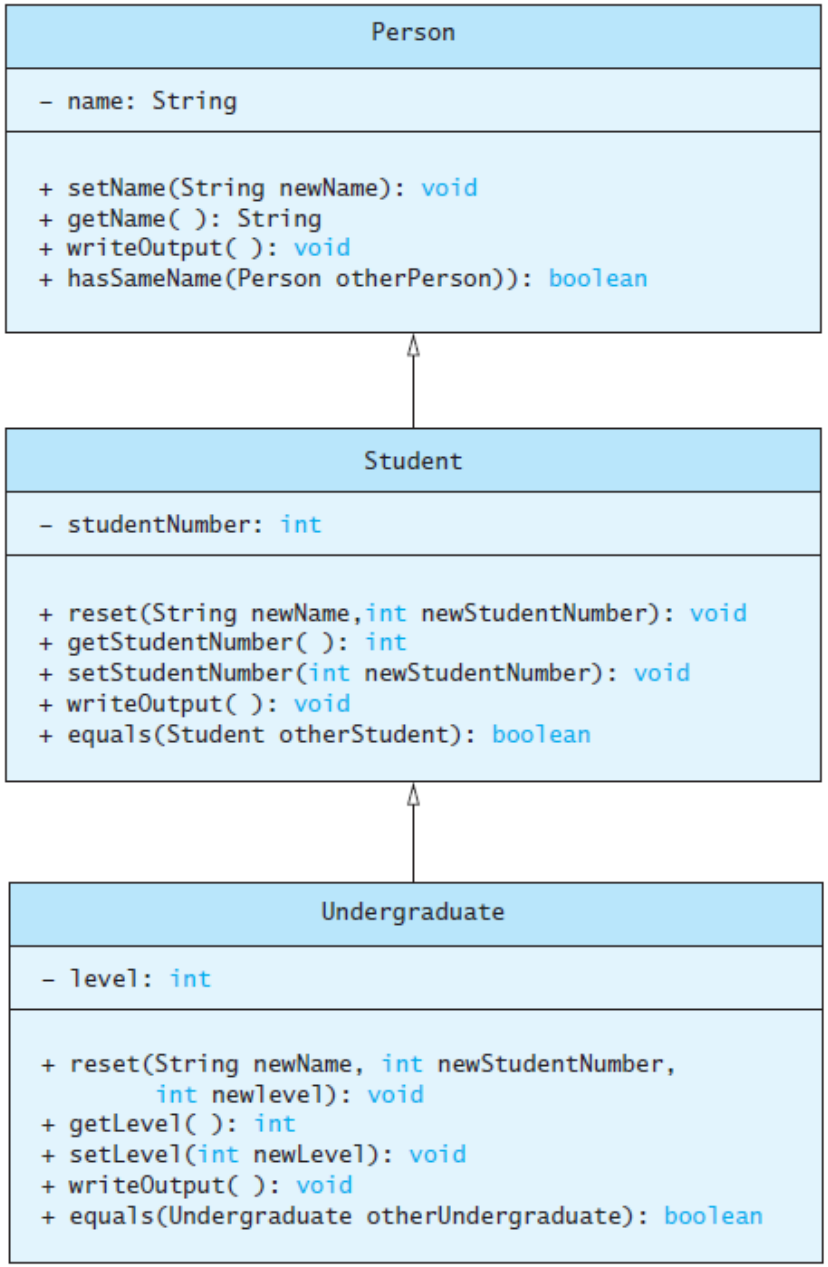
- Consider an array of **Person**

```
Person[] people = new Person[4];
```

- Since **Student** and **Undergraduate** are types of **Person**, we can assign them to **Person** variables

```
people[0] = new
  Student("DeBanque, Robin",
  8812);
```

```
people[1] = new
  Undergraduate("Cotty, Manny",
  8812, 1);
```





*JAVA: An Introduction to Problem Solving & Programming, 7*th Ed. By Walter Savitch

# Polymorphism

- Given:

```
Person[] people = new Person[4];
people[0] = new Student("DeBanque, Robin",
 8812);
```

- When invoking:

```
people[0].writeOutput();
```

- Which `writeOutput()` is invoked, the one defined for `Student` or the one defined for `Person`?
- Answer: The one defined for `Student`

# An Inheritance as a Type

- The method can substitute one object for another
  - Called *polymorphism*
- This is made possible by mechanism
  - *Dynamic binding*
  - Also known as *late binding*


*JAVA: An Introduction to Problem Solving & Programming, 7*th Ed. By Walter Savitch

# Dynamic Binding and Inheritance

- When an overridden method invoked
  - Action matches method defined in class used to create object using `new`
  - Not determined by type of variable naming the object
- Variable of any ancestor class can reference object of descendant class
  - Object always remembers which method actions to use for each method name


*JAVA: An Introduction to Problem Solving & Programming, 7*th Ed. By Walter Savitch

# Polymorphism Example

- View sample class, listing 8.6
  **`class PolymorphismDemo`**
- Output

```
Name: Cotty, Manny
Student Number: 4910
Student Level: 1

Name: Kick, Anita
Student Number: 9931
Student Level: 2
```

```
Name: DeBanque, Robin
Student Number: 8812

Name: Bugg, June
Student Number: 9901
Student Level: 4
```