SENG1110/SENG6110
Object Oriented Programming

THE UNIVERSITY OF NEWCASTLE AUSTRALIA

FACULTY OF ENGINEERING AND BUILT ENVIRONMENT

www.newcastle.edu.au

Lecture 10

Inheritance and Polymorphism

Outline

- Encapsulation
- Inheritance
- The class `Object`
- Polymorphism.
- Abstract methods and classes.
- Interfaces.
- Composition.

THE UNIVERSITY OF NEWCASTLE AUSTRALIA

Encapsulation

- Classes combine data (instance variables) and operations (methods) on that data.
- Methods can access the data, but internal state of an object cannot be manipulated directly (private instance variables)
- This process is called **encapsulation**.
- This is one of the important concepts in Object-oriented design

- Another way to say...

THE UNIVERSITY OF NEWCASTLE AUSTRALIA

Encapsulation

- Person **A** implements class `Course`.
- Person **B** can use class `Course`, without knowing **how** the methods were implemented.
- The details of the implementation are encapsulated in the method and hidden from person **B**.
- This is know as *information hiding* or **encapsulation**.
- If person **A** decides to change the implementation of the methods in class `Course`, the person **B** will not be affected (only if person **A** decides to change the signature ☹).

THE UNIVERSITY OF NEWCASTLE AUSTRALIA

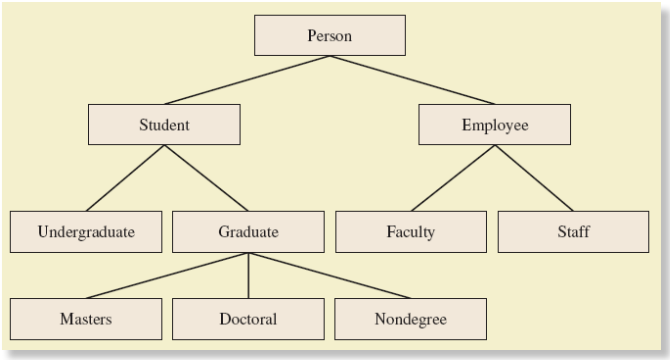
Inheritance

- Inheritance allows programmer to define a general class
- And later to define a more specific class
  - Adds new details to general definition
- New class inherits all properties of initial, general class

Inheritance

- Suppose you need to implement a class **`Employee`**.
- Suppose you have the class **`Person`**.
- Every **`Employee`** is a **`Person`**.
- So, an idea is to extend the definition of the class **`Person`**, adding the elements that are necessary.
- However, without making any physical changes to the class **`Person`**.
- This is the principle of **inheritance**

Derived Classes



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 in CodeSamplesWeek9
  `class Student extends Person`
- View demo program, CodeSamplesWeek9
  `class InheritanceDemo`

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

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

THE UNIVERSITY OF NEWCASTLE AUSTRALIA

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 UNIVERSITY OF NEWCASTLE AUSTRALIA

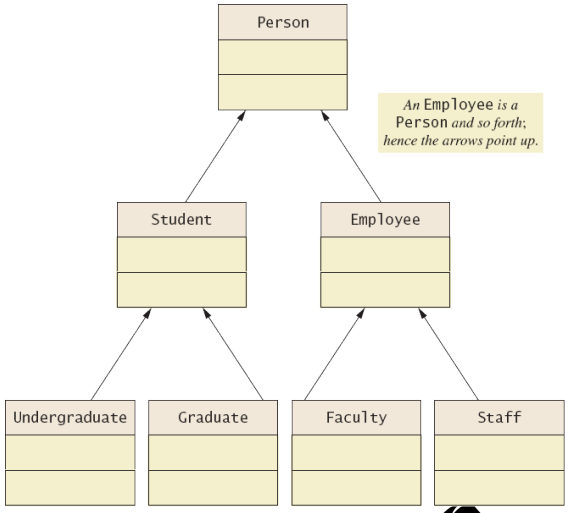
Private Instance Variables, Methods

- 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

THE UNIVERSITY OF NEWCASTLE AUSTRALIA

UML Inheritance Diagrams

A class hierarchy in UML notation



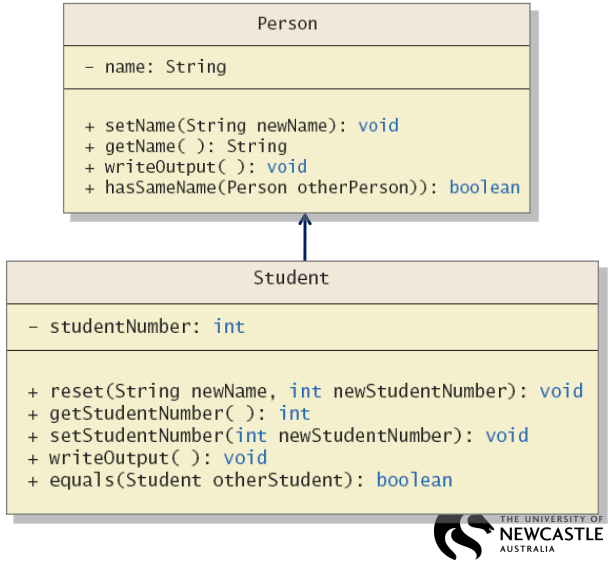
Constructors in Derived Classes

- 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`**

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


UML Inheritance Diagrams

Some details of UML class hierarchy



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

Calling an Overridden Method

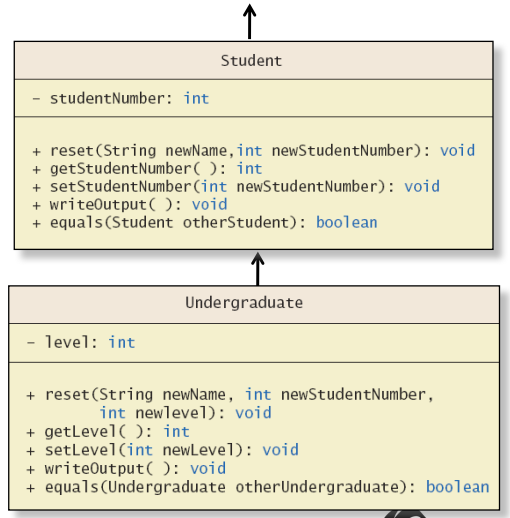
- Reserved word **`super`** can also be used to call method in overridden method

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

- Calls method by same name in base class

Programming Example

More details of the UML class hierarchy



Programming Example

- A derived class of a derived class
- View sample class, CodeSamplesWeek09
  `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

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

Inheritance – superclass (base) and subclass (derived)

- Members of the **subclass** cannot directly access the **private** members of the **superclass**.
- The **subclass** can include additional data and method members.
- The **subclass** can override (redefine) the public methods of the **superclass**.
  - However, this redefinition applies only to the objects of the **subclass** , not to the objects of the **superclass**.
- All data members of the **superclass** are also data members of the **subclass** .
- Similarly, the methods of the **superclass** (unless overridden) are also the methods of the subclass. (Remember first rule above when accessing a member of the **superclass** in the **subclass**).

Inheritance

- To write a method's definition of a **subclass**, specify a call to the public method of the **superclass**.
  - If **subclass** overrides public method of **superclass**, specify call to public method of **superclass** :

    ```
    super.methodName(parameter list)
    ```

  - If subclass does not override public method of **superclass**, specify call to public method of superclass:

    ```
    methodName(parameter list)
    ```


class Rectangle


```
public class Rectangle
{
    private double length;
    private double width;

    public Rectangle()
    {
          length = 0;
          width = 0;
    }
    public Rectangle(double l, double w)
    {
          setDimension(lw,);
    }
    public void setdimension(double l, double w)
    {
          if (l>=0) length = l;
          else      length = 0;
          if (w>=0) width = w;
          else       width = 0;
    }
```

class Rectangle


```
    public double getLength()
    {
          return length;
    }

    public double getWidth()
    {
          return width;
    }

    public double area()
    {
          return length*width;
    }

    public void print()
    {
          System.out.println("Length = "+length +" width = "+width);
    }
}
```


class Box


```
public Class Box extends Rectangle
{
    private double height;

    public Box()
    {
        super();
        height = 0;
    }

    public Box(double l, double w, double h)
    {
        super(l, w);
        height = h;
    }
```


class Box


```
    public void print()
    {
        super.print();
        System.out.print("; Height = " + height);
    }
    public void setDimension(double l, double w, double h)
    {
        super.setDimension(l, w);
        if (h >= 0)  height = h;
        else                 height = 0;
    }
    public double area()
    {
        return  2 * (getLength()*getWidth() +
                             getLength()*height +
                             getWidth()*height);
    }
}
```


Defining Constructors of the Subclass

- Call constructor of superclass:
  - Must be first statement.
  - Specified by super parameter list.

```
public Box()
{
    super();
    height = 0;
}

public Box(double l, double w, double h)
{
    super(l, w);
    height = h;
}
```


Protected Members of a Class

- `public x private`
- `protected`
  - If a member of a superclass needs to be accessed directly (only) by a subclass, the member is declared using the modifier **`protected`**

BClass


```
public class BClass {
          protected char bCh;
          private double bX;

    public BClass()  {
          bCh = '*';
          bX = 0.0;
          }
    public BClass(char ch, double u){
          bCh = ch;
          bX = u;
          }
    public void setData(double u){
          bX = u;
          }
    public void setData(char ch, double u){
          bCh = ch;
          bX = u;
    }
    public String toString()
    {
          return("Superclass: bCh = " + bCh + ", bX = " + bX + '\n');
    }
```


DClass


```
public class DClass extends BClass{
    private int dA;

    public DClass(){
          super();
          dA = 0;
    }
    public DClass(char ch, double v, int a){
          super(ch, v);
          dA = a;
    }
    public void setData(char ch, double v, int a){
          super.setData(v);
          bCh = ch; //initialize bCh using the assignment statement
          dA = a;
    }
    public String toString(){
          return (super.toString() + "Subclass dA = " + dA + '\n');
    }
}
```


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 <u>any</u> type

The Class `Object`

- Class Object has some methods that every Java class inherits
- Examples
  - Method `equals`
  - Method `toString`

A Better `equals` Method

- Programmer of a class should override method equals from `Object`
- View code of sample override, CodeSamplesWeek9

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

THE UNIVERSITY OF NEWCASTLE AUSTRALIA

`toString`

- Method `toString` called when `println(theObject)` invoked
  - Best to define your own `toString` to handle this
- Example

```
Person p = new Person("Anna");
System.out.println(p);
```


THE UNIVERSITY OF NEWCASTLE AUSTRALIA

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

THE UNIVERSITY OF NEWCASTLE AUSTRALIA

Polymorphism

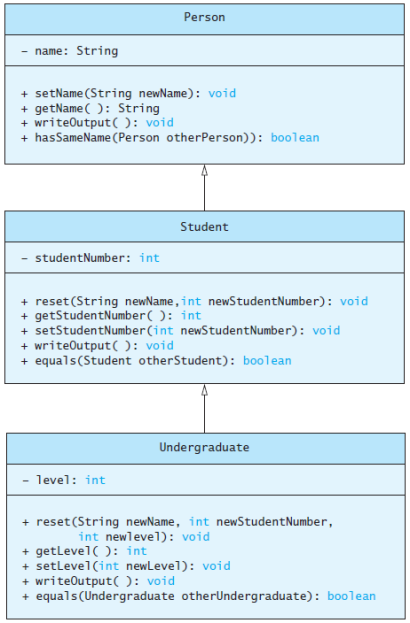
- Consider an array of **`Person`**

```
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);
```



Polymorphism Example

- View sample class, CodeSamplesWeek9

```
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
```


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`**

Polymorphism

- You cannot automatically make reference variable of subclass type point to object of its superclass.
- Suppose that `supRef` is a reference variable of a superclass type. Moreover, suppose that `supRef` points to an object of its subclass:
- You can use an appropriate cast operator on supRef and make a reference variable of the subclass point to the object.
  - On the other hand, if `supRef` does not point to a subclass object and you use a cast operator on supRef to make a reference variable of the subclass point to the object
  - Example…next slide…

Polymorphism


```
Person name, nameRef;
Employee employee, employeeRef;

name = new Person("John", "Blair");
employee = new Employee("Susan","Johnson",12.50,45);

nameRef = employee;

employeeRef = (Employee) name;

employeeRef = (Employee) nameRef;
```

illegal

legal

Polymorphism

- Operator `instanceof`: Determines whether a reference variable that points to an object is of a particular class type.
- This expression evaluates to true if p points to an object of the class BoxShape; otherwise it evaluates to false:

```
if( p instanceof BoxShape )
    System.out.println("p is an instance of BoxShape");
```


Abstract Methods

- A method that has only the heading with no body.
- Must be declared abstract.

```
public void abstract print();
public abstract object larger(object,object);
void abstract insert(int insertItem);
```


Abstract Classes

- A class that is declared with the reserved word `abstract` in its heading.
- An abstract class can
  - contain instance variables, constructors, finalizers, and non-abstract methods.
  - Can also contain abstract methods.
- If a class contains an abstract method, the class must be declared abstract.
- You cannot instantiate an object of an abstract class type. You can only declare a reference variable of an abstract class type.
- You can instantiate an object of a subclass of an abstract class, but only if the subclass gives the definitions of all the abstract methods of the superclass.

Abstract Class Example


```
public abstract class AbstractClassExample
{
    protected int x;
    public void abstract print();

    public void setX(int a)
    {
         x = a;
    }

    public AbstractClassExample()
    {
        x = 0;
    }
}
```


Class Interfaces

- Now consider different classes that implement this interface
  - They will each have the same behaviors
  - Nature of the behaviors will be different
- Each of the classes implements the behaviors/methods differently

THE UNIVERSITY OF NEWCASTLE AUSTRALIA

Interfaces

- A class that contains only abstract methods and/or named constants.
- Consider a set of behaviors for pets
  - Be named
  - Eat
  - Respond to a command
- We could specify method headings for these behaviors
- These method headings can form a class interface

THE UNIVERSITY OF NEWCASTLE AUSTRALIA

Java Interfaces

- A program component that contains headings for a number of public methods
  - Will include comments that describe the methods
- Interface can also define public named constants
- View CodeSamplesWeek9 interface Measurable

THE UNIVERSITY OF NEWCASTLE AUSTRALIA

Java Interfaces

- Interface name begins with uppercase letter
- Stored in a file with suffix `.java`
- Interface does not include
  - Declarations of constructors
  - Instance variables
  - Method bodies

Composition

- Another way to relate two classes.
- One or more members of a class are objects of another class type.
- "has-a" relation between classes.
  - For example, "every person has a date of birth."

Implementing an Interface

- To implement a method, a class must
  - Include the phrase
    ```
    implements Interface_name
    ```
  - Define each specified method
- View codeSamplesWeek9
  ```
  class Rectangle implements Measurable
  ```
- View codeSamplesWeek9
  ```
  class Circle implements Measurable
  ```


Your task

- Read
  - Lecture slides
  - Chapter 8
- Exercises
  - MyProgrammingLab
  - Computer lab exercises