Lecture 6

OOPS CONTINUED

Review

Remember Class Participation

UML Diagrams

The Unified Modeling Language (UML) is a visual language for capturing software designs and patterns. The first version of UML was defined 1994 and released by the Object Management Group (OMG) in 1997 as UML v.1.1. The syntax and a semantic of UML is defined by the OMG.

Class diagrams

A class diagram captures the static relationships of your software.

A class is represented by a rectangular box divided into compartments. A Compartment is an area in the box to write information. The first compartment holds the name, the second holds the attributes and the third is used for the operations

UML suggests that a class name:

- Starts with a capital letter
- is centered in the top compartment
- is written in a boldface font
- is written in italics if the class is abstract

Attributes

Attributes specifies details of a class and can be simple types or objects.

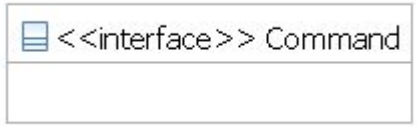
Inlined attributes are placed in the second compartment of the class. The notation for inline attribute is: `visibility name: type {multiplicity} {=default}`

Element	Values	Description
visibility	+	public Attribute
	-	private Attribute
	#	protected Attribute
	~	package Attribute
name	myName	Name of the attribute following the camelCase notation
type		Class name, interface or primitive types, e.g. int
multiplicity		Optional, if not specified then it is assumed to be 1, * for any value, 1,..,* for ranges.
default		Optional, default value of the attribute

Other Components

Interfaces

Interfaces are indicated via the stereotype

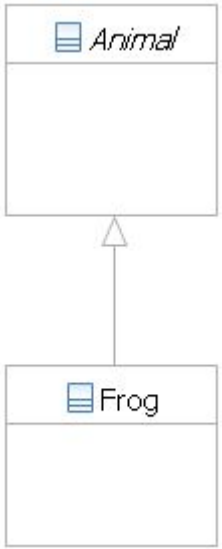


Generalisation

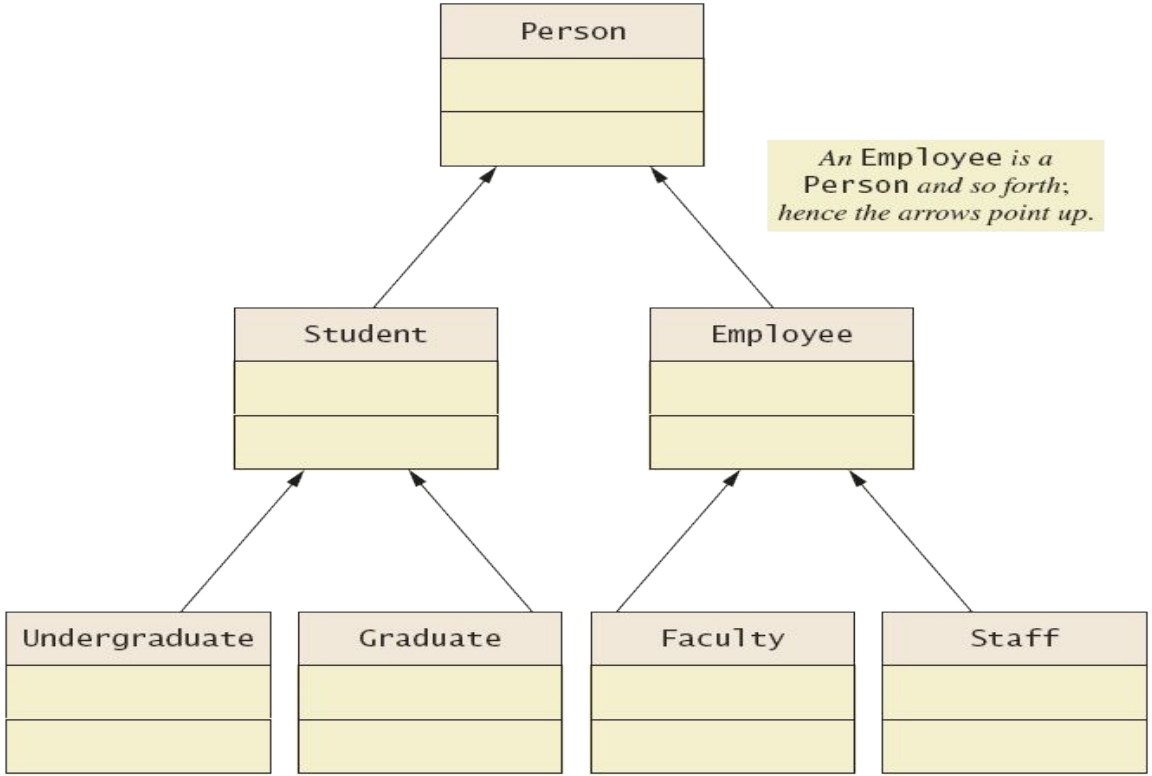
Represented by a solid line and a hollow triangular arrow

```
package animals;
public abstract class Animal {
}
```

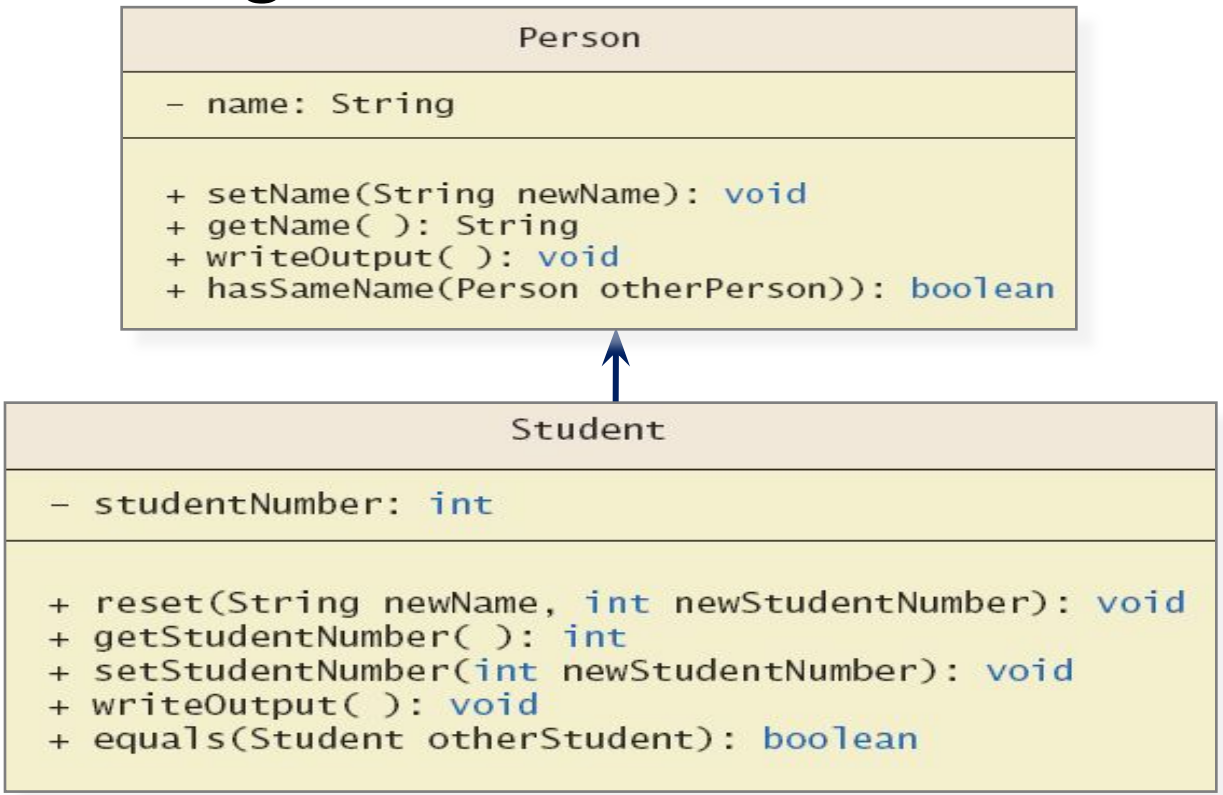
```
package animals;
public class Frog extends Animal {
}
```



Book Example

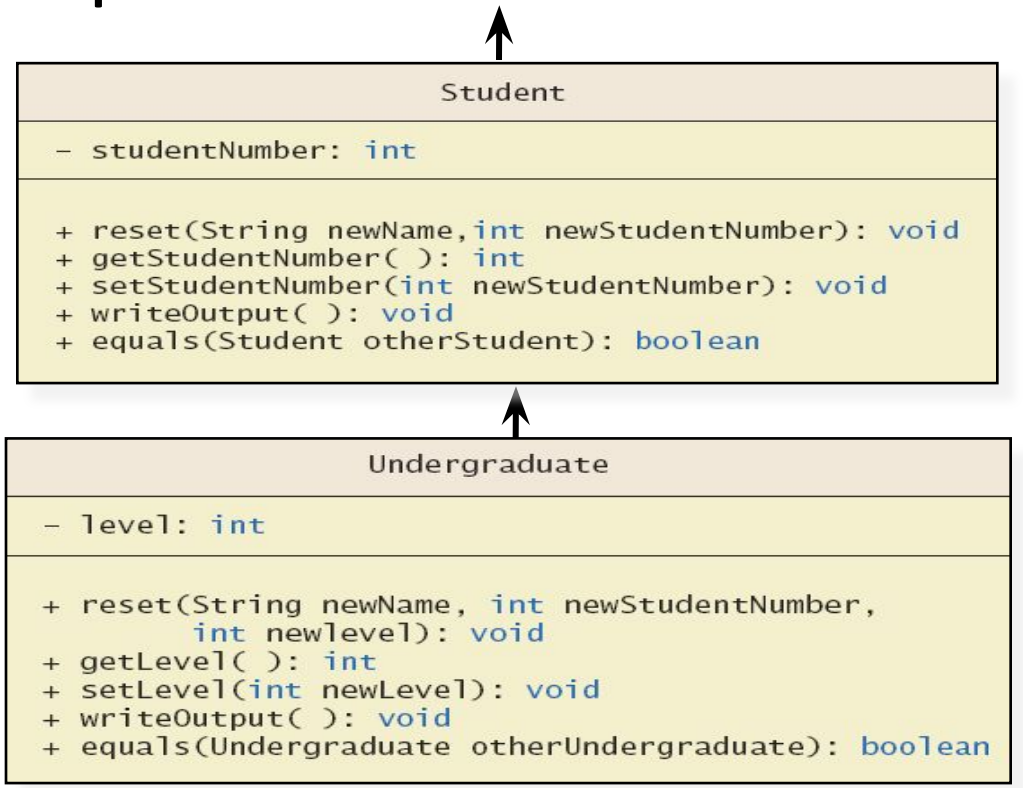


UML Inheritance Diagrams



Programming Example

- Figure 8.4 More details of the UML class hierarchy



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)`

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

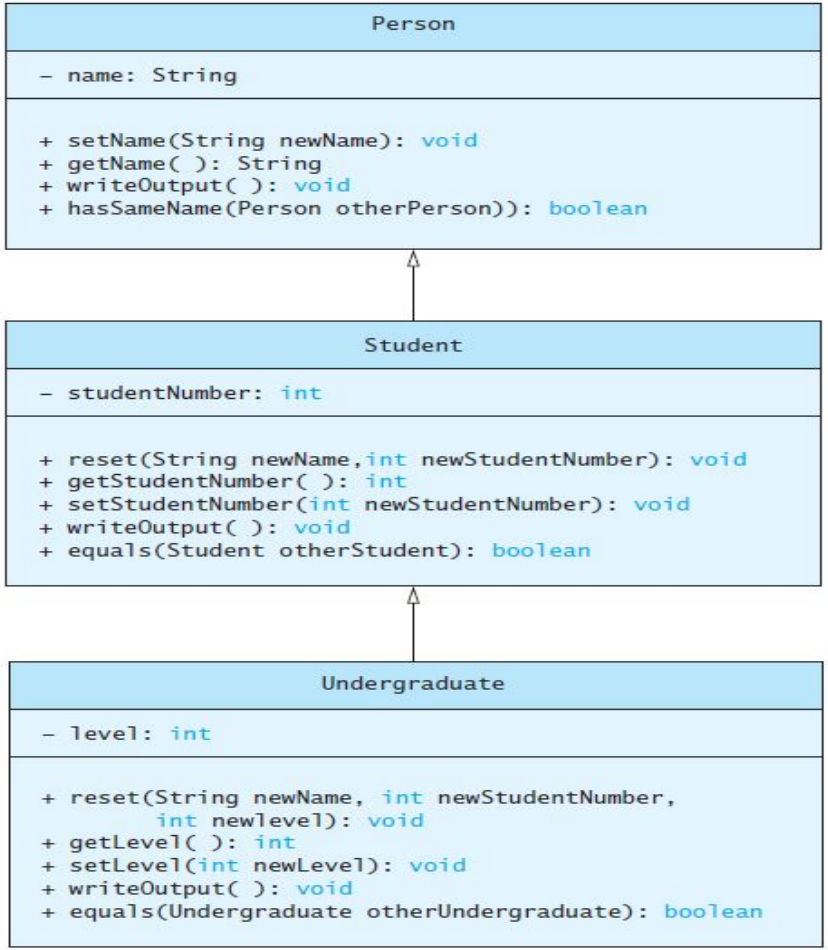
Polymorphism

- 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);
```



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*

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

An Inheritance as a Type

- Possible to write a method that has a parameter as an interface type
  - An interface is a reference type
- Program invokes the method passing it an object of any class which implements that interface

Comparable Interface

Java Comparable interface is used to order the objects of user-defined class.This interface is found in java.lang package and contains only one method named compareTo(Object). It provide single sorting sequence only i.e. you can sort the elements on based on single data member only

```
class Student implements Comparable<Student>{
int age;

Student(int age){

this.age=age;

}

public int compareTo(Student st){

if(age==st.age)

return 0;

else if(age>st.age)

return 1;

else

return -1; }
```

Summary

- An interface contains
  - Headings of public methods
  - Definitions of named constants
  - No constructors, no private instance variables
- Class which implements an interface must
  - Define a body for every interface method specified
- Interface enables designer to specify methods for another programmer

Summary

- Interface is a reference type
  - Can be used as variable or parameter type
- Interface can be extended to create another interface
- Dynamic (late) binding enables objects of different classes to substitute for one another
  - Must have identical interfaces
  - Called polymorphism

Summary

- Derived class obtained from base class by adding instance variables and methods
  - Derived class inherits all public elements of base class
- Constructor of derived class must first call a constructor of base class
  - If not explicitly called, Java automatically calls default constructor

Summary

- Within constructor
  - `this` calls constructor of same class
  - `super` invokes constructor of base class
- Method from base class can be overridden
  - Must have same signature
- If signature is different, method is overloaded

Summary

- Overridden method can be called with preface of **`super`**
- Private elements of base class cannot be accessed directly by name in derived class
- Object of derived class has type of both base and derived classes
- Legal to assign object of derived class to variable of any ancestor type

Summary

- Every class is descendant of class **`Object`**