CSY2030
Systems Design & Development

Inheritance

Structure of Lecture

- We will define the following Object-Oriented principles :
  - Inheritance
  - Polymorphic Variables
- The above will be supported by java programming

Inheritance

- Consider representing employees in a company:
  - all employees have a name and address
  - for programmers we store languages they know
  - for all offshore workers we store the date of safety certificate
    - for offshore chefs we store their catering qualifications
    - for offshore engineers we store their engineering title e.g mechanical eng, instrument eng.

Inheritance

In a traditional languages such as Pascal we might have:

```
employee = record
        name : string;
        address : string;
end;
```

```
programmer = record
        name : string;
        address : string;
        languages : list_array;
end;
```

```
offshore_worker = record
        name : string;
        address : string;
        offshore_date : string;
end;
```

```
offshore_chef = record
        name : string;
        address : string;
        offshore_date : string;
        qualification : string;
end;
```

```
offshore_engineer = record
        name : string;
        address : string;
        offshore_date : string;
        title : string;
end;
```

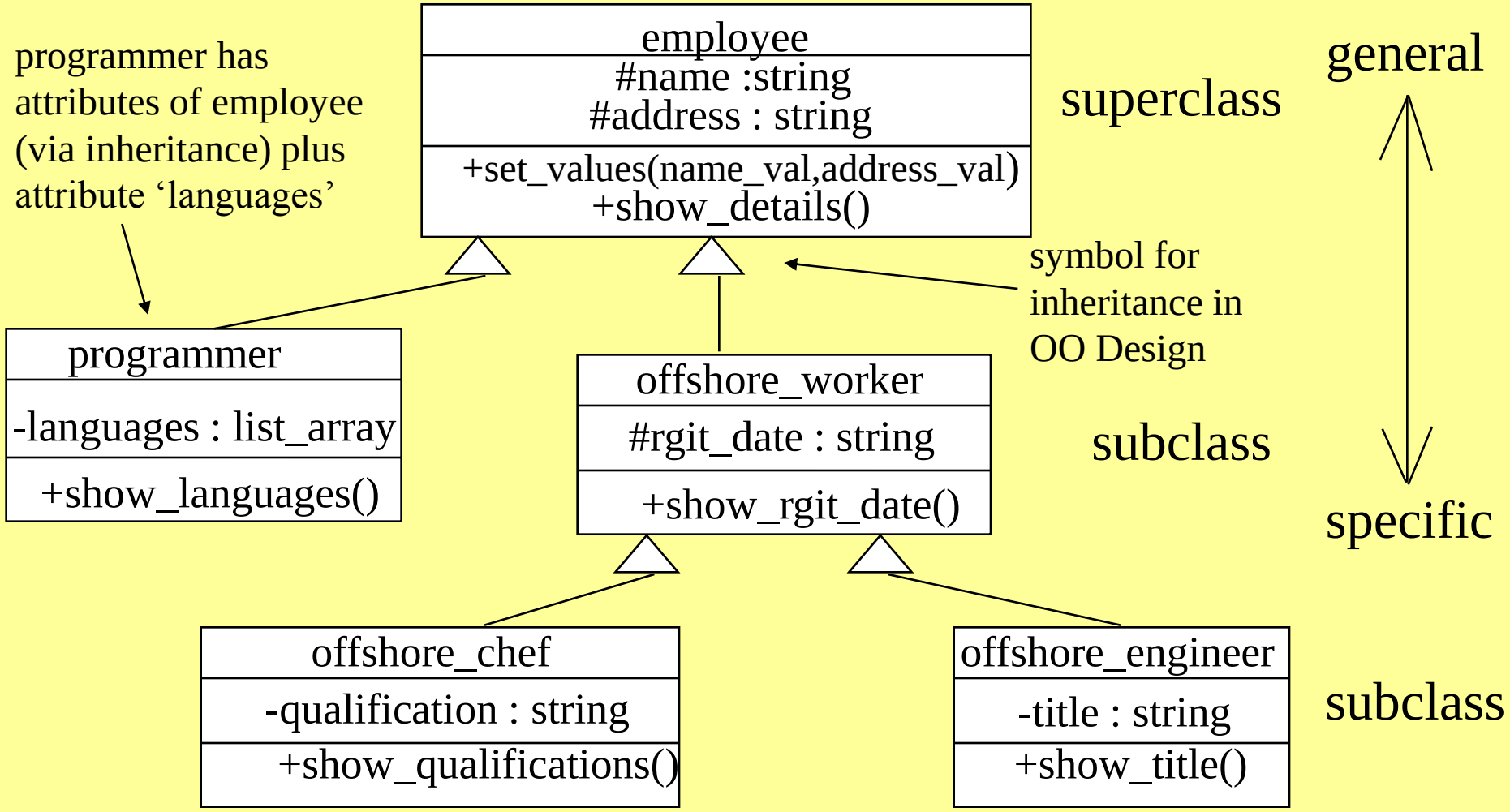
There is a lot of repetition here !! - better to have reuse.

Inheritance

- This is the process by which one class can acquire the properties of another class
  - supports classification
- Without classification, each class would have to define explicitly all of its characteristics
- With classifications, a class need only define those qualities that make it unique within its class.
- When inheriting methods, subclass can redefine an inherited method
  - this is called **overriding**

Inheritance

More efficient to have a hierarchy of classes:



Inheritance

- Object can be of type
  - superclass (most general class)
  - any subclass
- The relationship between a superclass and an inherited class is called an “is a” relationship
  - E.g programmer is a employee
  - E.g offshore_chef is a offshore_worker
- When using inheritence you are
  - inheriting attributes of a class(es)
  - inheriting methods of a class(es)
    - subclass’s methods can **override** the inherited method of the superclass i.e they are redefined in the subclass

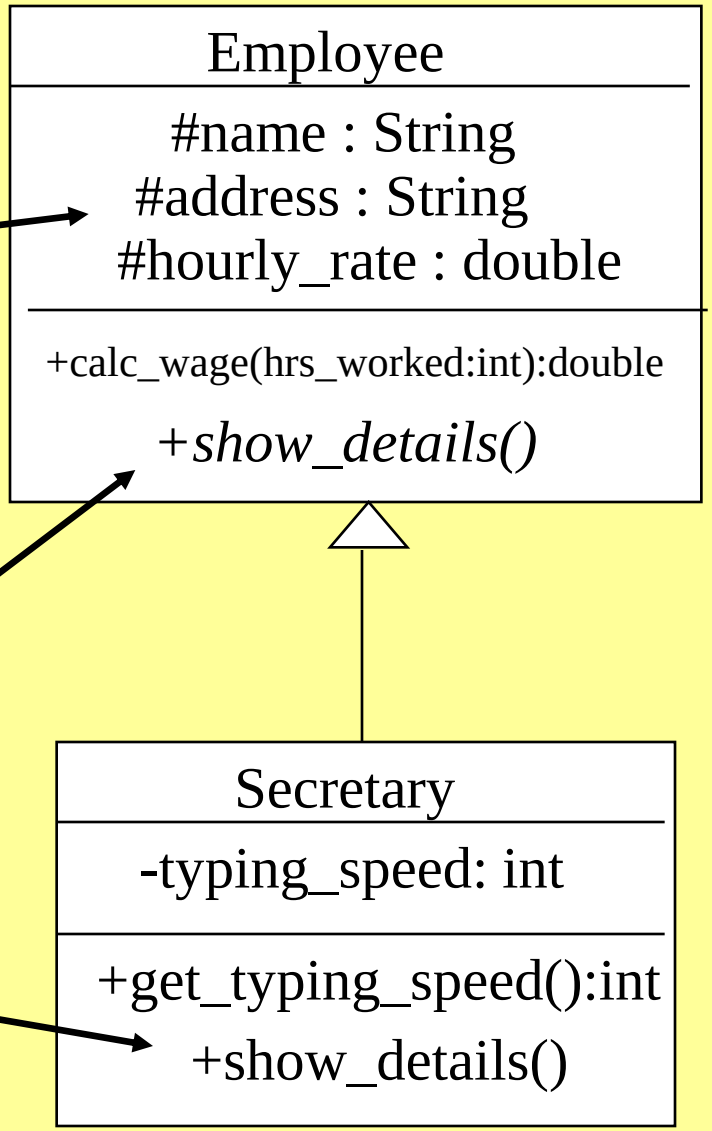
Inheritance

We use # to show that an attribute is **protected**

- This means it is private in superclass and accessible by subclasses
- If it was left private then subclass cannot access superclass's attributes

If method appears in both superclass and subclass then this means the method is redefined (replaced) in the subclass i.e the method in the subclass **overrides** the method of the superclass

- notation is to put superclass's method in italics



```
public class Employee {
    protected String name;
    protected String address;
    protected double hourly_rate;

    public Employee (String name_val,
          String address_val, double hr_rate_val)
    {
       name = name_val;
       address = address_val;
       hourly_rate = hr_rate_val;
    }

    public double calc_wage(int hrs_worked){
          return hrs_worked*hourly_rate;
    }

    public void  show_details(){
          System.out.println(name + ' ' +
                address + ' ' + hourly_rate);
     }
}
```

```
public class Secretary extends Employee {
    private int typing_speed;

    public Secretary (String name_val,
          String address_val, double hr_rate_val,
         int typing_speed_val)
    {
       super(name_val, address_val, hr_rate_val);
       typing_speed=typing_speed_val;
    }

    public int get_typing_speed(){
          return typing_speed;
    }

    public void  show_details(){
          System.out.println(name + ' ' +
                address + ' ' + hourly_rate
                + ' ' + typing_speed);
     }
}
```

Inheritance

- In superclass, attributes are **protected**
  - attributes are private to Employee and accessible by any subclass
- To inherit a class, use keyword **extends**
  - *public class Secretary extends Employee*
- Calling a method in the superclass is achieved by using the keyword **super**
  - *super(name_val, address_val, hr_rate_val)* calls the superclass's constructor
    - this save you writing the equivalent code in the constructor of Secretary

Inherited Objects

- Object of type subclass can call methods in the superclass or subclass e.g

Secretary Jean = new Secretary('Jean', 'Dundee', 5.00, 100)

System.out.println(Jean.calc_wages(10));

would give

50.00

Superclass method

System.out.println(Jean.get_typing_speed());

would give

100

Subclass methods

Jean.show_details()

would give

Jean Dundee 5.00 100

Polymorphic Variables

- This is when object is of type superclass/subclass
  - object whose type is a superclass can be an object of a subclass
    - it can call methods of superclass or overridden methods of subclass e.g

Employee Mary = new Secretary(“Mary”, “Stirling”, 5.00, 120);

System.out.println(Mary.calc_wage(10));

would give

50.00

Call superclass method

Mary.show_details();

would give

Mary Stirling 5.00 120

Call overridden method in subclass

Polymorphic Variables cont.

Cannot call method in subclass e.g

System.out.println(Mary.get_typing_speed()); ✗

However you can a method in the subclass if you cast the object to the subclass

System.out.println(((Secretary)Mary).get_typing_speed()); ✓

Cannot have an object of type subclass assigned to a superclass e.g

Secretary Mary = new Employee(“Mary”, “Stirling”, 5.00); ✗

Because not all Employees are Secretaries

Summary

- **Inheritance** allows re-use of attributes and methods
  - Allows new attributes and methods to be introduced
  - Allows methods to be re-defined in subclasses
    - This is called **overriding**
- Make attributes **protected** in superclass
  - This means attributes are still private in superclass and accessible by subclasses
- Objects can be of type superclass/subclass
  - This is **polymorphic variables**