CSY2030
Systems Design & Development

Classes, Objects and Polymorphism

Structure of Lecture

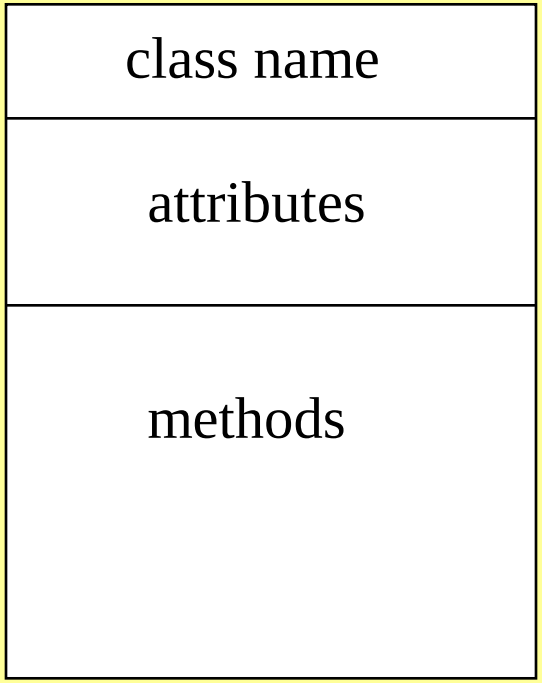
- We will define the following Object-Oriented principles :
  - Classes
  - Objects
  - Polymorphism
- The above will be supported by java programming

Classes

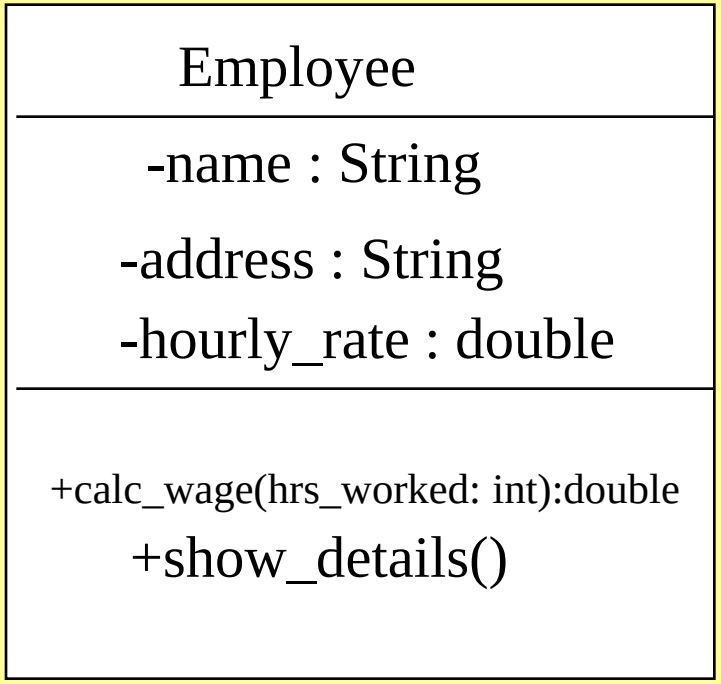
- A class is a user defined data type which encapsulates the following members:
  - data (also known as *attributes* or *fields*)
    - data should, where possible, be private i.e only accessible within the class
  - operations on the data (also known as *methods / member functions*)
    - in general, operations are public i.e accessible outwith the class - they are the interface to the data
- Variables whose type are classes are called objects
  - objects are a.k.a. *instances* of a class

Classes

Conceptually



e.g



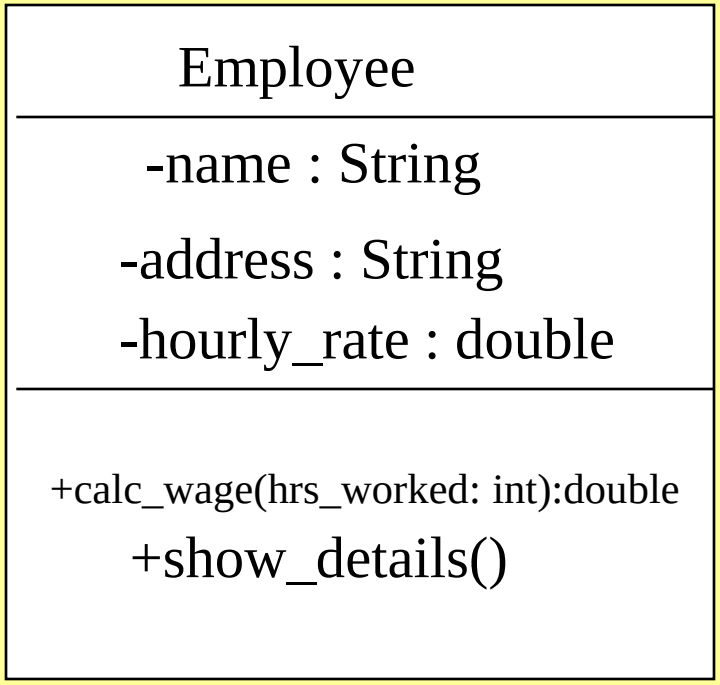
(part of object-oriented design
i.e UML - see future lectures)

Anything private is preceeded by a –
Anything public is preceeded by a +

can create employee objects bob and fred (variables) of type employee i.e

Employee bob, fred;

Classes




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

    public Employee (String name,
                String address, double hourly_rate)
    {
        this.name = name;
        this.address = address;
        this.hourly_rate = hourly_rate;
    }

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

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

Classes

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

    public Employee (String name,
                String address, double hourly_rate)
    {
        this.name = name;
        this.address = address;
        this.hourly_rate = hourly_rate;
    }

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

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

Attributes - they are private i.e only accessible in class a.k.a **Information Hiding**

Method with same name as class is called a **constructor** - this method is called when a new object is created. Constructors don’t have a return type

this method is of type double i.e it must have the keyword return in it followed by an expression which is a double

this method is of type void i.e it must not have the keyword return in it - voids return nothing and is used, in general, for displays

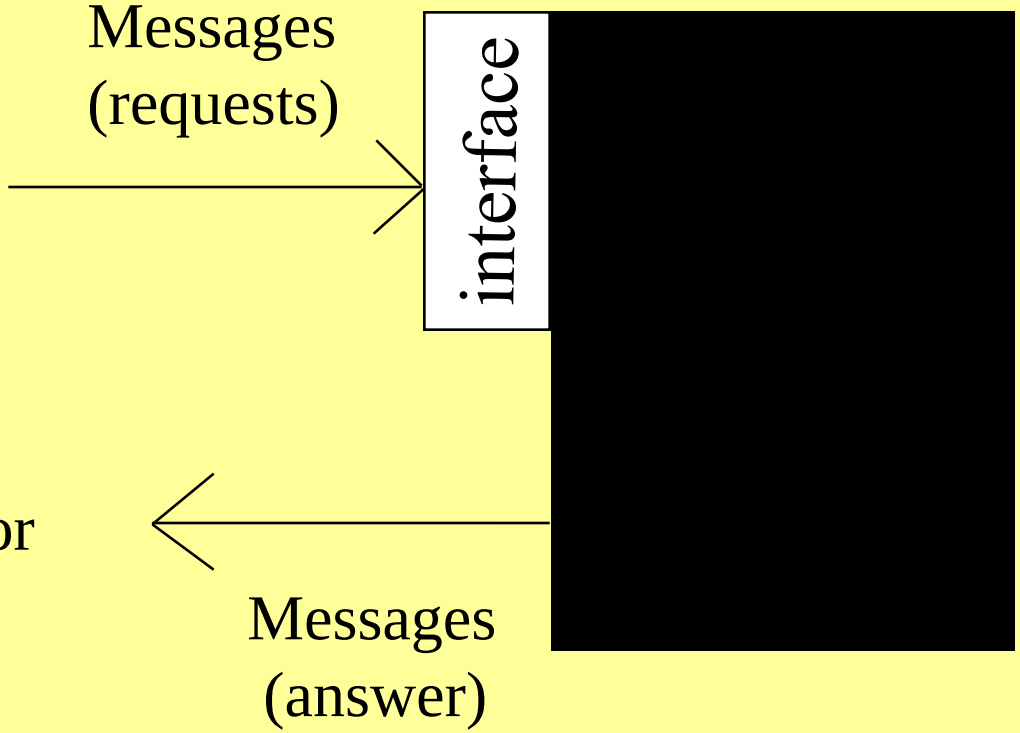
Objects

- Within an object, some of the data and/or methods may be private to the object
  - they are inaccessible to anything outside object
  - allows protection
- Some of the data and/or methods are public
  - they are accessible to anything outside the object (a.k.a the interface)
  - public data should be minimal or not at all
- Can think of objects as a black box….

Objects

An object is considered a black box

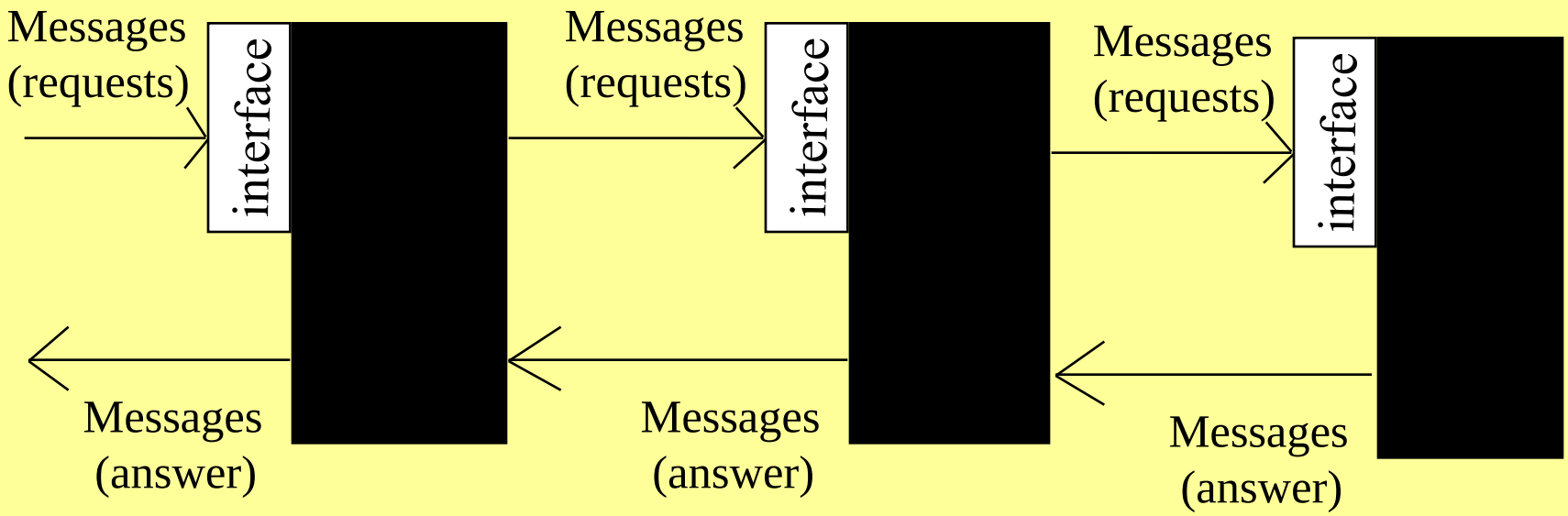
- implementation is hidden
- send object a message (via an interface) and you either
  1. get back an answer and/or
  2. get a change inside the object



Consider a calculator.....

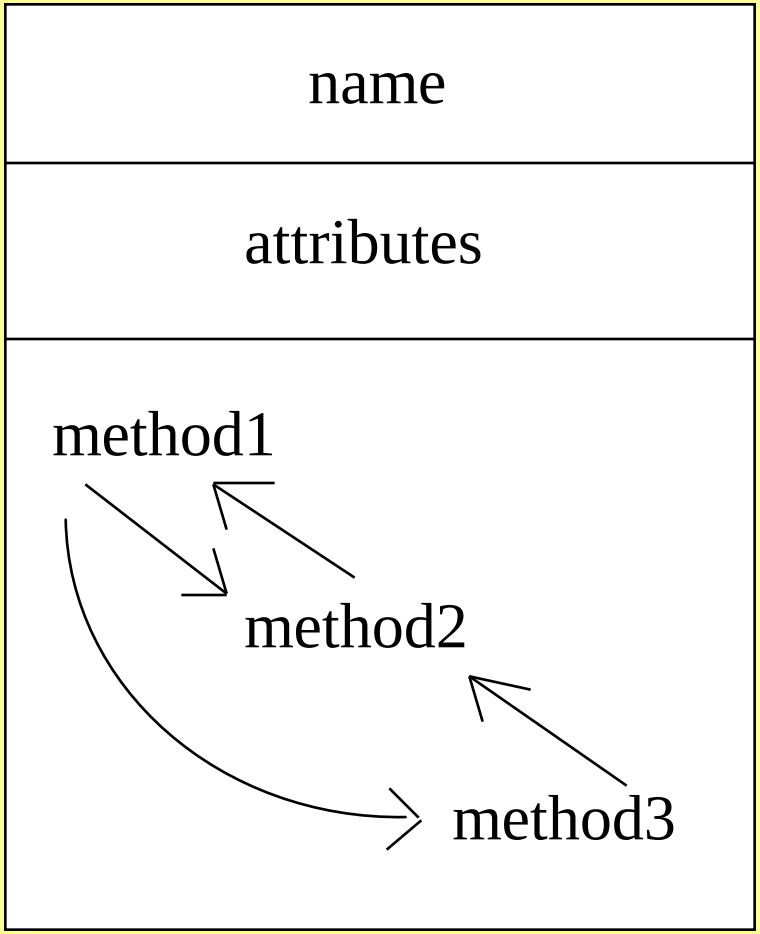
Objects

Can get message passing between objects:



Objects

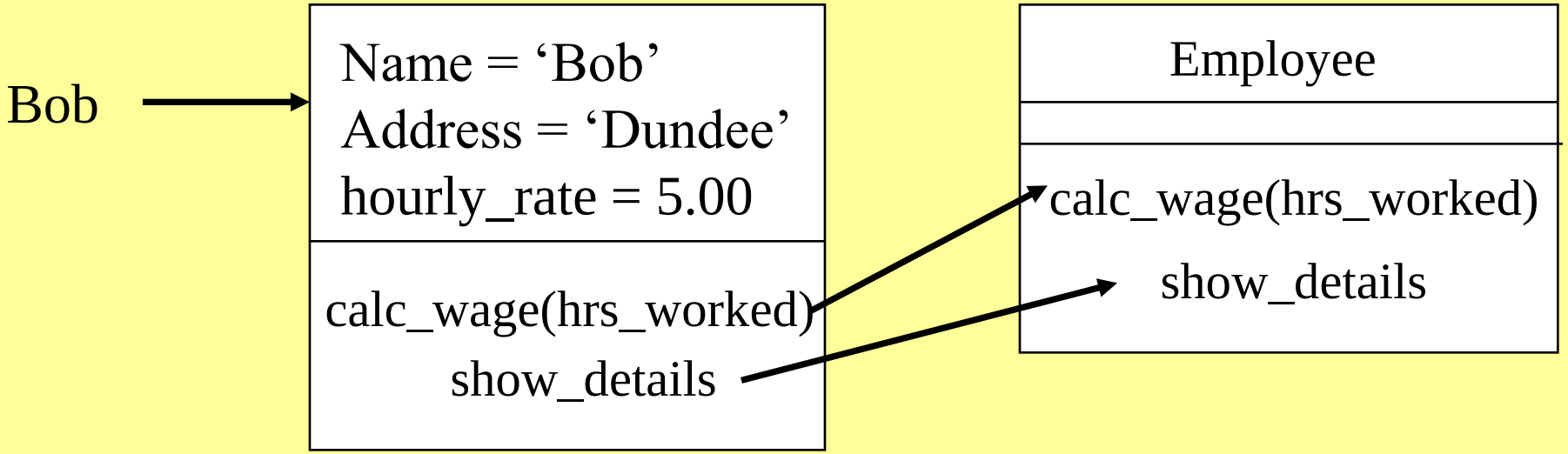
Methods inside an object can also call one another



Objects

- Objects are created using the **new** function
  - this allocates memory to an object
  - calls the class's constructor
  - creates a reference to memory e.g

Employee Bob = new Employee("Bob","Dundee",5.00);



Objects

- When we call methods we are said to *pass a message* to an object
  - i.e call the methods in the class
  - achieved by using the *dot* operator e.g

```
Bob.show_details();
```

would give:

Bob Dundee 5.00

```
System.out.println(Bob.calc_wage(10));
```

would give:

50.00

Calling method calc_wage with argument 10

Private vs public properties

- Anything **public** (like our methods) in a class called be called/accessed by the dot operator e.g
  - Bob.show_details();

- Anything **private** (like our attributes) in a class cannot be called/accessed by the dot operator e.g
  - Bob.hourly_rate=100; // this is not allowed

- To deal with private and public properties we should have **setter** and **getter** methods

Setter methods

- Setter methods generally set attributes which are private
  - You do this because you cannot access private attributes directly
- Could have following setter methods to the Employee class:

```
public void setName(String name){
        this.name=name;
}

public void setAddress(String address){
        this. address = address;
}

public void setHourlyRate(double hourly_rate){
        this.hourly_rate = hourly_rate;
}
```

Getter methods

- Getter methods generally retrieve the value of attributes which are private
  - You do this because you cannot access private attributes directly
- Could add the following getter methods to our Employee class:

```
public String getName(){
        return name;
}

public String getAddress(){
        return address;
}

public double getHourlyRate(){
        return  hourly_rate;
}
```

Setter + Getter methods

- Could have following code inside the main method:

```
Employee Bob = new Employee("Bob","Dundee",5.00);
Bob.setAddress("Northampton");
System.out.println("Bob lives in " + Bob.getAddress());
```

This would output:

```
Bob lives in Northampton
```

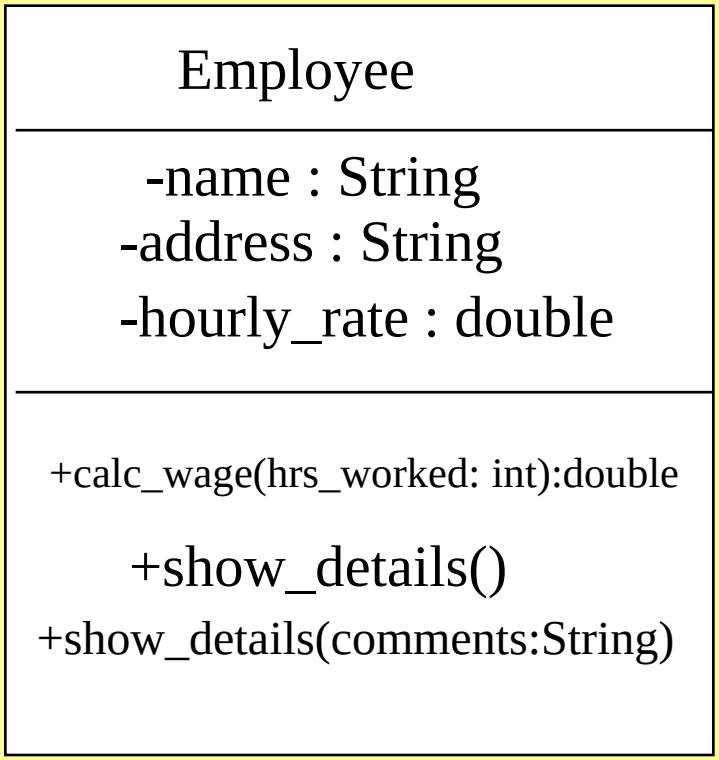
Polymorphism

- Characterised by the phrase ‘one interface, multiple methods’
  - allows one interface to be used with a general class of actions
  - specific action is determined by the exact nature of the situation
- Consider a calculator (interface)
  - the + (addition) button knows how to add integers and/or real numbers

Polymorphism in Java

- This is 2 or more methods with the same name
  - difference between them are their parameters
  - such methods are called **overloaded** methods
- Achieved in Java by doing the following:
  - each overloaded method is defined in the class as normal with it parameters

Polymorphism



Note that we list the polymorphic methods as normal

```
public class Employee {
    private String name;
    private String address;
    private 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 void  show_details(String comments){
        System.out.println(name + ‘ ‘ + address + ‘ ‘ + hourly_rate+ ‘ ‘ + comments);
     }
}
```

Polymorphism in Java

- We pass messages to an object
  - i.e call the methods in the class
  - e.g

Bob.show_details();

Knows to call show_details method with no arguments

would give:

Bob Dundee 5.00

Knows to call show_details method with a string argument

whereas

Bob.show_details(“good worker”);

would give:

Bob Dundee 5.00 good worker

Defining a class in Eclipse

- To create a class in Eclipse, go to *file* → *new* → *Class* and give your class a name e.g Employee
- Do not check the "public static void main()" checkbox as you have been previously
- This will create an empty class:

```
public class Employee {

}
```

- Once you have an empty class you can add attributes and methods to it

Objects in Eclipse

- Once this class has been defined you can use the class to create a person object to store the information about an individual
- You should create the instance of the person object in a method outside the class such as the main method
  - This means you need to create another file with the main method in it
- In summary each goes in its own file

1) Define a class with the fields you want to store in a separate file with the name <class name>.java
2) Define another class which has the main method to create an instance of the class using the new keyword and the class name

Summary

- **Classes** are user-defined types made up of attributes and methods
- Instances of classes are **objects**
  - They are created using the **new** function
    - This calls the class's **constructor**
- Keep attributes **private**
  - Set them with **setter** methods and access them with **getter** methods
- Make methods **public** to
- **Polymorphism** is 2 or methods with the same name but different paramters
  - These methods are called **overloaded** methods
- Call methods using **dot operator**