



Lecture 4: Inheritance

What is Inheritance?

- In the real world:
 - We have general terms for objects in the real world, example “Vehicle”
 - Vehicles have wheels, they move, you can ride them, etc.
 - There are many specific types of “Vehicles”
 - Cars, bicycle, trucks, busses etc.
 - They all share (inherit) attributes of a vehicle
 - But each is more specific:
 - Cars have 4 wheels, carry 5 people
 - Bicycles have 2 wheels, carry 1 person

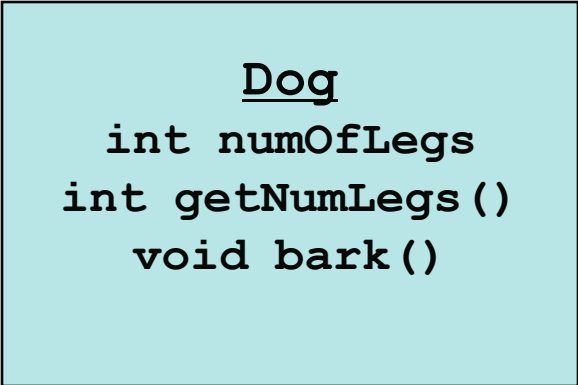
What is Inheritance?

In software:

- Objects that are derived from other object "resemble" their parents by *inheriting* both state (fields) and behaviour (methods).
- Parents are more general than children
- Children refine parents class specification for different uses

Dog Class

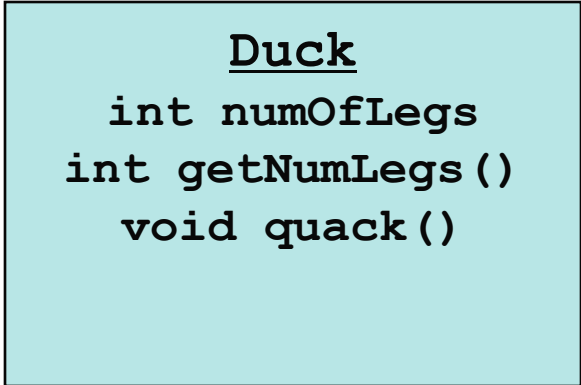
```
public class Dog {  
    private int numOfLegs;  
  
    public Dog(int legs){  
        numOfLegs = 4;  
    }  
  
    public int getNumLegs(){  
        return numOfLegs;  
    }  
  
    public String bark(){  
        return "Woof";  
    }  
}
```



```
Dog  
int numOfLegs  
int getNumLegs()  
void bark()
```

Duck Class

```
public class Duck {  
    private int numOfLegs;  
  
    public Cat(int legs){  
        numOfLegs = 2;  
    }  
  
    public int getNumLegs(){  
        return numOfLegs;  
    }  
  
    public String quack(){  
        return "quack";  
    }  
}
```



```
Duck  
int numOfLegs  
int getNumLegs()  
void quack()
```

Problem: Code Duplication

- `Duck` and `Dog` have the `numOfLegs` field and the `getNumLegs` method in common
- Classes often have a lot of state and behaviour in common
- Result: lots of duplicate code!

Solution: Inheritance

- Inheritance allows you to write new classes that inherit from existing classes
- The existing class whose properties are inherited is called the "parent" or **superclass**
- The new class that inherits from the super class is called the "child" or **subclass**
- Result: Lots of **code reuse!**

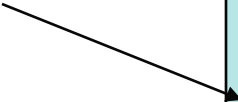
```
Dog  
int numOfLegs  
int getNumLegs()  
void bark()
```

```
Duck  
int numOfLegs  
int getNumLegs()  
void quack()
```



```
Animal  
int numOfLegs  
int getNumLegs()
```

superclass



subclass



subclass



```
Dog  
void bark()
```

```
Duck  
void quack()
```


Animal Superclass

```
public class Animal {
    public int numOfLegs;

    public Animal(int numOfLegs) {
        this.numOfLegs = numOfLegs;
    }

    public int getNumLegs() {
        return this.numOfLegs;
    }
}
```

Inheritance Rules

- Use the **extends** keyword to indicate that one class inherits from another
- The subclass inherits public (and *protected*) fields and methods of the superclass
- Use the **super** keyword in the subclass constructor to call the superclass constructor

Dog Class

```
public class Dog extends Animal {  
  
    public Dog() {  
        super(4);  
    }  
  
    public String bark() {  
        return "Woof";  
    }  
}
```

Duck Class

```
public class Duck extends Animal {  
    public Duck() {  
        super(2);  
    }  
  
    public String quack(){  
        return "Quack";  
    }  
}
```

Is-A Relationship

- Inheritance defines an “is-a” relationship
 - Dog *is an* Animal
 - Duck *is an* Animal
 - One way relationship
 - Animal is not a Dog! (Remember this when coding!)
- The derived class inherits access to methods and fields from the parent class
 - Use inheritance when you want to reuse code

Aside: Has-A Relationship

- When one class has a field of another class (or primitive type)
 - Animal has an int
- Do not confuse with inheritance!

Inheritance Review 1

What is the output of the following?

```
Dog d = new Dog();  
Duck u = new Duck();
```

```
System.out.println("A dog has " +  
    d.getNumLegs() + d.bark());
```

```
System.out.println("A duck has " +  
    u.getNumLegs() + u.quack());
```

(Dog and Duck inherit the `getNumLegs()` method from the `Animal` super class, but get `bark` and `quack` from their own class)

Which Lines Don't Compile?

```
public static void main(String[] args) {
    Animal a1 = new Animal(4);
    a1.getNumLegs();
    a1.bark();    // Animal does not have bark
    a1.quack();  // Animal does not have quack

    Dog a2 = new Dog();
    a2.getNumLegs();
    a2.bark();
    a2.quack();  // Dog does not have a quack

    Duck du = new Duck();
    du.getNumLegs();
    du.bark();   // Duck does not have bark
    du.quack();

}
```


Subclass Constructor

- The first thing a subclass constructor must do is call *a constructor* in the superclass.
- If the subclass constructor does not do this, then the default superclass constructor (with no arguments) will be called implicitly.

Implicit Super Constructor Call

If I have this `Food` class:

```
public class Food {
    private boolean raw;
    public Food() {
        raw = true;
    }
}
```

then this `Beef` subclass:

```
public class Beef extends Food {
    private double weight;
    public Beef(double w) {
        weight = w
    }
}
```

is equivalent to:

```
public class Beef extends Food {
    private double weight;
    public Beef(double w) {
        super();
        weight = w
    }
}
```

Inheritance Review 2

```
public class A {  
    public A() { System.out.println("I'm A"); }  
}
```

```
public class B extends A {  
    public B() { System.out.println("I'm B"); }  
}
```

```
public class C extends B {  
    public C() { System.out.println("I'm C"); }  
}
```

What does this print out?

```
C x = new C();
```

Overriding Methods

- Subclasses can *override* methods in their superclass

```
class Therm {  
    protected double celsius;  
  
    public Therm(double c) {  
        celsius = c;  
    }  
  
    public double getTemp() {  
        return celsius;  
    }  
}
```

```
class ThermUS extends Therm {  
    public ThermUS(double c) {  
        super(c);  
    }  
  
    // degrees in Fahrenheit  
    public double getTemp() {  
        return celsius * 1.8 + 32;  
    }  
}
```

- What is the output of the following?

```
ThermUS thermometer = new ThermUS(100);  
System.out.println(thermometer.getTemp());
```

212

Calling Superclass Methods

When you override a method, you can call the superclass's copy of the method by using the syntax `super.method()`

```
class Therm {  
    private double celsius;  
  
    public Therm(double c) {  
        celsius = c;  
    }  
  
    public double getTemp() {  
        return celsius;  
    }  
}
```

```
class ThermUS extends Therm {  
  
    public ThermUS(double c) {  
        super(c);  
    }  
  
    public double getTemp() {  
        return super.getTemp()  
            * 1.8 + 32;  
    }  
}
```

Remember Casting?

- "Casting" means "promising" the compiler that the object will be of a particular type.
 - So the compiler should go ahead and convert
- You can cast a variable to the type of the object that it references to use that object's methods.

```
Animal a2 = new Dog();  
a2.bark(); //Animal does not have a bark method  
-> ((Dog) a2).bark();
```

- The casting will fail if the variable doesn't reference an object of that type.

Which Castings Will Fail?

```
public static void main(String[] args) {  
    Animal a1 = new Dog();  
    ((Dog) a1).bark(); //a1 changed to Dog  
    ((Duck) a1).quack(); //a1 is not a Cat  
  
    Animal a2 = new Duck();  
    ((Duck) a2).quack(); //a2 changed to Duck  
    ((Dog) a2).bark(); //Dog is not a Dog
```

Programming Example

A company has a list of Employees.

It asks you to provide a payroll sheet for all employees.

- Different types of employees
 - manager, engineer, software engineer.
 - Manager straight Salary
 - Engineer Hourly
- You have an old Employee class but need to add very different data and methods for managers and engineers.

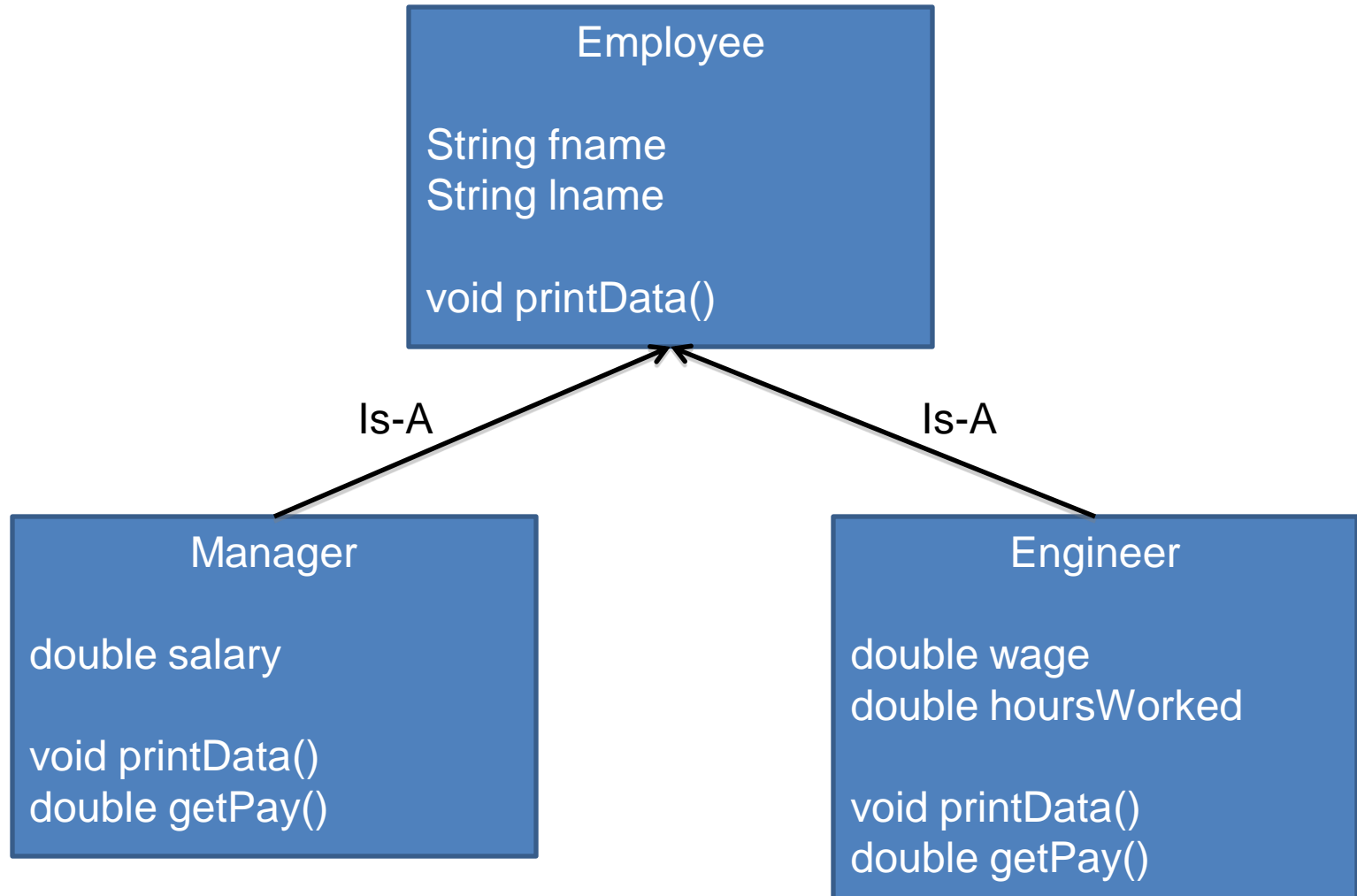
Employee Class

This is a simple super or base class.

```
class Employee {
    // Fields
    private String firstName, lastName;

    // Constructor
    public Employee(String fName, String lName) {
        firstName= fName; lastName= lName;
    }
    // Method
    public void printData() {
        System.out.println(firstName + " " + lastName);
    }
}
```

Inheritance



Engineer Subclass

```
class Engineer extends Employee {
    private double wage;
    private double hoursWorked;
    public Engineer(String fName, String lName,
                    double rate, double hours) {
        super(fName, lName);
        wage = rate;
        hoursWorked = hours;
    }
    public double getPay() {
        return wage * hoursWorked;
    }
    public void printData() {
        super.printData();           // PRINT NAME
        System.out.println("Weekly pay: $" +
                           getPay()); }
}
```

Manager Subclass

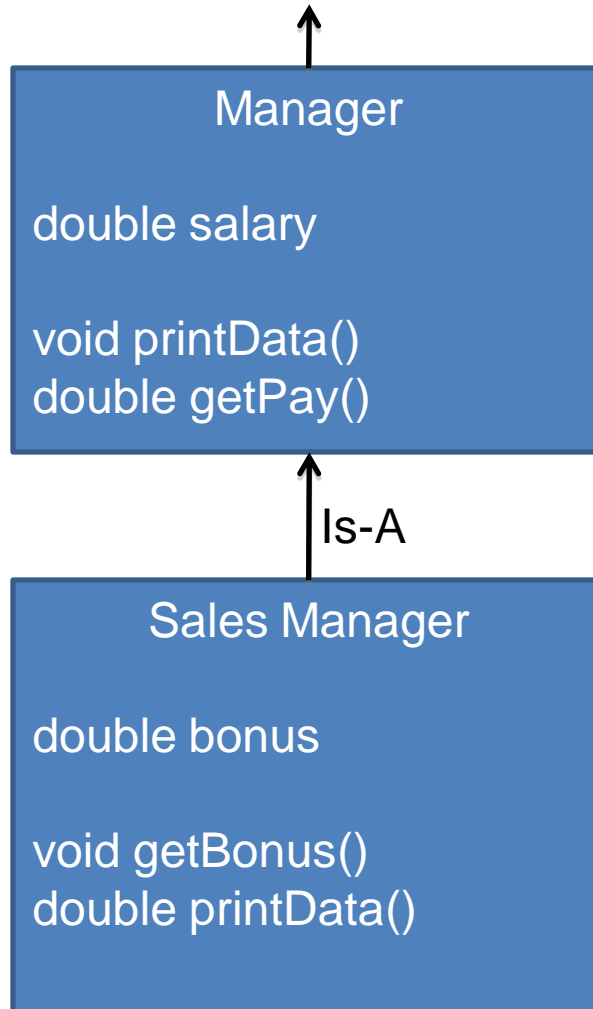
```
class Manager extends Employee {
    private double salary;

    public Manager(String fName, String lName, double sal){
        super(fName, lName);
        salary = sal; }

    public double getPay() {
        return salary; }

    public void printData() {
        super.printData();
        System.out.println("Monthly salary: $" + salary);}
}
```

More Inheritance



SalesManager Class

```
class SalesManager extends Manager {
    private double bonus;          // Bonus Possible as commission.

    // A SalesManager gets a constant salary of $1250.0
    public SalesManager(String fName, String lName, double b) {
        super(fName, lName, 1250.0);
        bonus = b; }

    public double getBonus() {
        return bonus; }

    public void printData() {
        super.printData(); //Print from both Super Classes
        System.out.println("Bonus Pay: $" + getBonus()); }
}
```

SalesManager

SalesManager

Manager

Employee

```
public Employee(String, String)
public void printData()
```

```
public Manager(String, String, double)
public void printData()
public double getPay()
```

```
private double bonus
```

```
public SalesManager(String, String, double)
public double getBonus()
public void printData()
```

Main Method

```
public class PayRoll {
    public static void main(String[] args) {
        Engineer fred      = new Engineer("Fred", "Smith", 12.0, 8.0);
        Manager ann        = new Manager("Ann", "Brown", 1500.0);
        SalesManager mary  = new SalesManager("Mary", "Kate", 2000.0);

        Employee[] employees = new Employee[3];
        employees[0]= fred;
        employees[1]= ann;
        employees[2]= mary;
        for (int i=0; i < 3; i++)
            employees[i].printData();
    }
}
```

**Java knows the
object type and
chooses the
appropriate method
at run time**

Output from main method

Fred Smith

Weekly pay: \$96.0

Ann Brown

Monthly salary: \$1500.0

Mary Barrett

Monthly salary: \$1250.0

Bonus: \$2000.0

Note that we could not write:

```
employees[i].getPay();
```

because `getPay()` is not a method of the superclass `Employee`.

In contrast, `printData()` is a method of `Employee`, so Java can find the appropriate version, starts from subclass (most inherited) and works the way up for method

instanceof Operator

- How about if you want to test if an object is of a specific class?
- Use the instanceof operator
 - returns true if an object is of the class
 - returns true if an object is a subclass of the class
- Form:
obj instanceof Class

instanceof Example

```
Employee emp = new Employee("first", "last");
```

```
Engineer eng = new Engineer("Fred", "Smith",  
                             12.0, 8.0);
```

```
Manager mana = new Manager("Ann", "Brown", 1500.0);
```

```
SalesManager salesm = new SalesManager("Mary", "Kate",  
                                         2000.0);
```

emp instanceof Employee	true
emp instanceof Engineer	false
mana instanceof Employee	true
eng instanceof Engineer	true
salesm instanceof Manager	true

instanceof Example

```
public class PayRoll {
    public static void main(String[] args) {
        Engineer fred      = new Engineer("Fred", "Smith", 12.0, 8.0);
        Manager ann        = new Manager("Ann", "Brown", 1500.0);
        SalesManager mary  = new SalesManager("Mary", "Kate", 2000.0);

        Employee[] employees = new Employee[3];
        employees[0]= fred;
        employees[1]= ann;
        employees[2]= mary;
        for (int i=0; i < 3; i++)
            if (employees[i] instanceof SalesManager)
                System.out.println(employees[i].getBonus());
    }
}
```

Object Class

- All Java classes implicitly inherit from `java.lang.Object`
- So every class you write will automatically have methods in `Object` such as `equals`, `hashCode`, and `toString`.
- We'll learn about the importance of some of these methods in later lectures.