MIT AITI 2004 – Lecture 12

Inheritance

What is Inheritance?

- In the real world: We inherit traits from our mother and father. We also inherit traits from our grandmother, grandfather, and ancestors. We might have similar eyes, the same smile, a different height . . . but we are in many ways "derived" from our parents.

- In software: Object inheritance is more well defined! Objects that are derived from other object "resemble" their parents by ***inheriting*** both state (fields) and behavior (methods).

Dog Class

```
public class Dog {
    private String name;
    private int fleas;

    public Dog(String n, int f) {
        name = n;
        fleas = f;
    }

    public String getName() { return name; }

    public int getFleas() { return fleas; }

    public void speak() {
        System.out.println("Woof");
    }
}
```

May want to replace "Woof" with whatever the "dog sound" is in your country.

Cat Class

```
public class Cat {
    private String name;
    private int hairballs;

    public Cat(String n, int h) {
        name = n;
        hairballs = h;
    }

    public String getName() { return name; }

    public int getHairballs() { return hairballs; }

    public void speak() {
        System.out.println("Meow");
    }
}
```

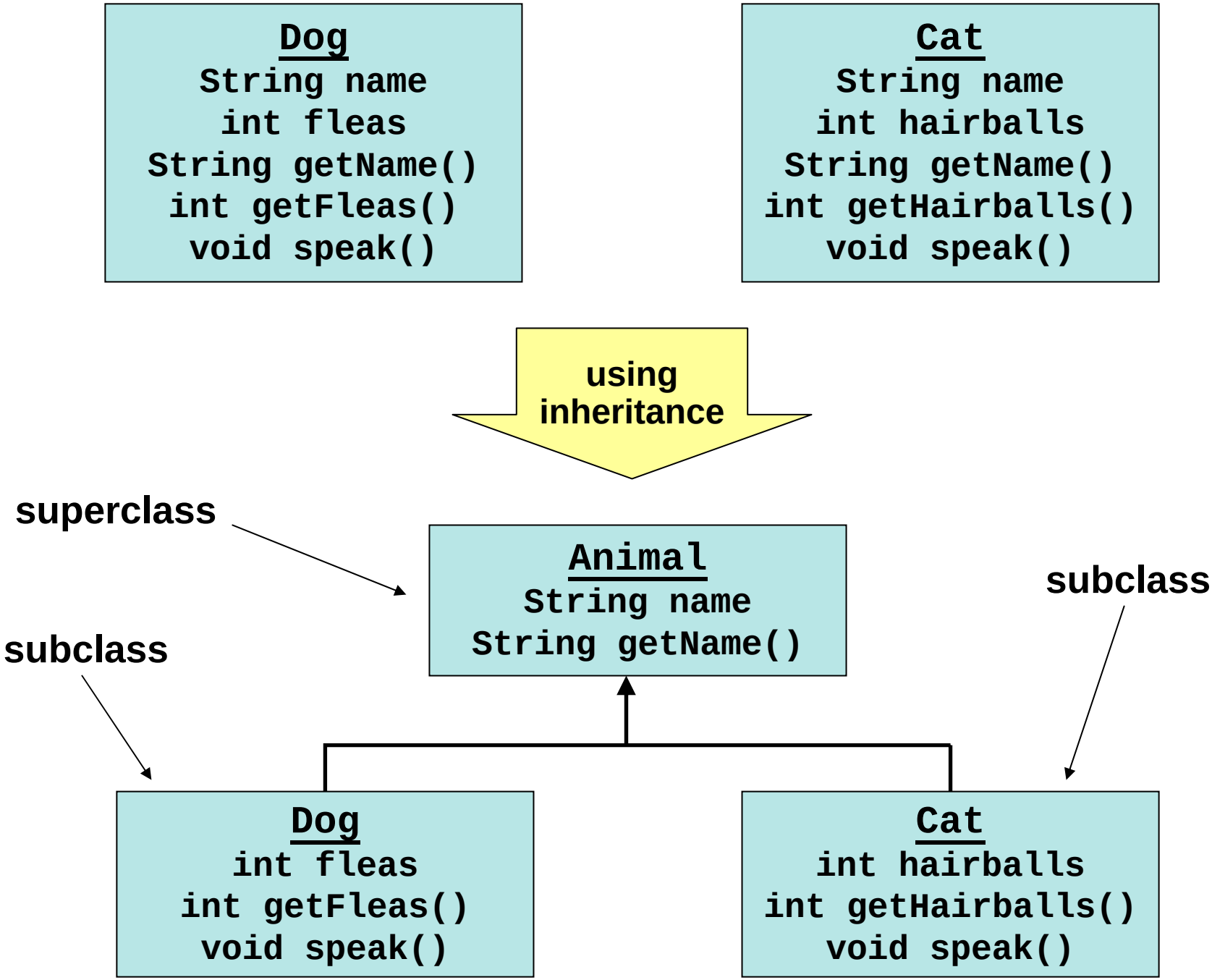
Mention that fields should be private, but not private just for example
May want to replace "Meow" with whatever the "cat sound" is in your country.

Problem: Code Duplication

- **`Dog`** and **`Cat`** have the **`name`** field and the **`getName`** method in common
- Classes often have a lot of state and behavior 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
String name
int fleas
String getName()
int getFleas()
void speak()
Cat
String name
int hairballs
String getName()
int getHairballs()
void speak()
using inheritance
superclass
Animal
String name
String getName()
subclass
subclass
Dog
int fleas
int getFleas()
void speak()
Cat
int hairballs
int getHairballs()
void speak()

Animal Superclass

```
public class Animal {

    private String name;

    public Animal(String n) {
        name = n;
    }

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

Dog Subclass

```
public class Dog extends Animal {

    private int fleas;

    public Dog(String n, int f) {
        super(n); // calls Animal constructor
        fleas = f;
    }

    public int getFleas() {
        return fleas;
    }

    public void speak() {
        return System.out.println("Woof");
    }
}
```

May want to replace "Woof" with whatever the "dog sound" is in your country.

Cat Subclass

```
public class Cat extends Animal {

    private int hairballs;

    public Cat(String n, int h) {
        super(n); // calls Animal constructor
        hairballs = h;
    }

    public int getHairballs() {
        return hairballs;
    }

    public void speak() {
        return System.out.println("Meow");
    }
}
```

May want to replace "Meow" with whatever the "cat sound" is in your country.

Inheritance Quiz 1

- What is the output of the following?

```
Dog d = new Dog("Rover" 3);
Cat c = new Cat("Kitty", 2);
System.out.println(d.getName() + " has " +
           d.getFleas() + " fleas");
System.out.println(c.getName() + " has " +
           c.getHairballs() + " hairballs");
```

```
Rover has 3 fleas
Kitty has 2 hairballs
```

(Dog and Cat inherit the getName method from Animal)

may want to change names to pet names in your country

Inheritance Rules

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

Subclass Constructor

- The first thing a subclass constructor must do is call the superclass constructor
- This ensures that the superclass part of the object is constructed before the subclass part
- If you do not call the superclass constructor with the **super** keyword, and the superclass has a constructor with no arguments, then that superclass constructor will be called implicitly.

example of the third point on the next page

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 Quiz 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();
```

```
I'm A
I'm B
I'm C
```

Overriding Methods

- Subclasses can *override* methods in their superclass

```
class Therm {
  public double celsius;

  public Therm(double c) {
    celsius = c;
  }

  public double getTemp() {
    return celcius;
  }
}
```

```
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? **212**

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

May want to change Meow and Roar to whatever the animal sounds are in your country

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) {
    celcius = c;
  }

  public double getTemp() {
    return celcius;
  }
}
```

```
class ThermUS extends Therm {

  public ThermUS(double c) {
    super(c);
  }

  public double getTemp() {
    return super.getTemp()
              * 1.8 + 32;
  }
}
```

Note that the celsius field can now be private as it should be!

Access Level

- Classes can contain fields and methods of four different access levels:
  - **`private:`** access only to the class itself
  - **`package:`** access only to classes in the same package
  - **`protected:`** access to classes in the same package and to all subclasses
  - **`public:`** access to all classes everywhere

when protected comes up, tell them this one is new.

Variable Type vs Object Type

- Variables have the types they are given when they are declared and objects have the type of their class.
- For an object to be assigned to a variable is must be of the same class *or a subclass* of the type of the variable.
- You may not call a method on a variable if it's type does not have that method, even if the object it references has the method.

Which Lines Don't Compile?

```
public static void main(String[] args) {
  Animal a1 = new Animal();
  a1.getName();
  a1.getFleas();      // Animal does not have getFleas
  a1.getHairballs();  // Animal does not have getHairballs
  a1.speak();         // Animal does not have speak
  Animal a2 = new Dog();
  a2.getName();
  a2.getFleas();      // Animal does not have getFleas
  a2.getHairballs();  // Animal does not have getHairballs
  a2.speak();         // Animal does not have speak
  Dog d = new Dog();
  d.getName();
  d.getFleas();
  d.getHairballs();   // Dog does not have getHairballs
  d.speak();
}
```

May want to go through slides 8, 9, and 10 to remind them of the Animal, Dog, and Cat classes

Remember Casting?

- "Casting" means "promising" the compiler that the object will be of a particular type
- You can cast a variable to the type of the object that it references to use that object's methods without the compiler complaining.
- The cast 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 Animal();
  ((Dog)a1).getFleas();       // a1 is not a Dog
  ((Cat)a1).getHairballs();   // a1 is not a Cat
  ((Dog)a1).speak();          // a1 is not a Dog

  Animal a2 = new Dog();
  ((Dog)a2).getFleas();
  ((Cat)a2).getHairballs();   // a2 is not a Cat
  ((Dog)a2).speak();

  Dog d = new Dog();
  ((Cat)d).getHairballs();    // d is not a Cat
}
```

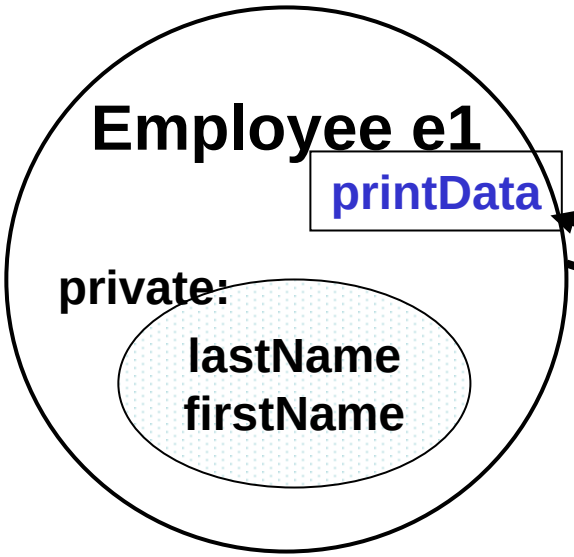
May want to go through slides 8, 9, and 10 to remind them of the Animal, Dog, and Cat classes

Programming Example

- **A Company has a list of Employees. It asks you to provide a payroll sheet for all employees.**
  - Has extensive data (name, department, pay amount, …) for all employees.
  - Different types of employees – manager, engineer, software engineer.
  - You have an old Employee class but need to add very different data and methods for managers and engineers.
    - Suppose someone wrote a name system, and already provided a legacy Employee class. The old Employee class had a printData() method for each Employee that only printed the name. We want to reuse it, and print pay info.

REVIEW PICTURE

Encapsulation **Message passing** **"Main event loop"**




```
public … Main(…){
Employee e1…("Mary","Wang");
...
e1.printData();
// Prints Employee names.
...
}
```

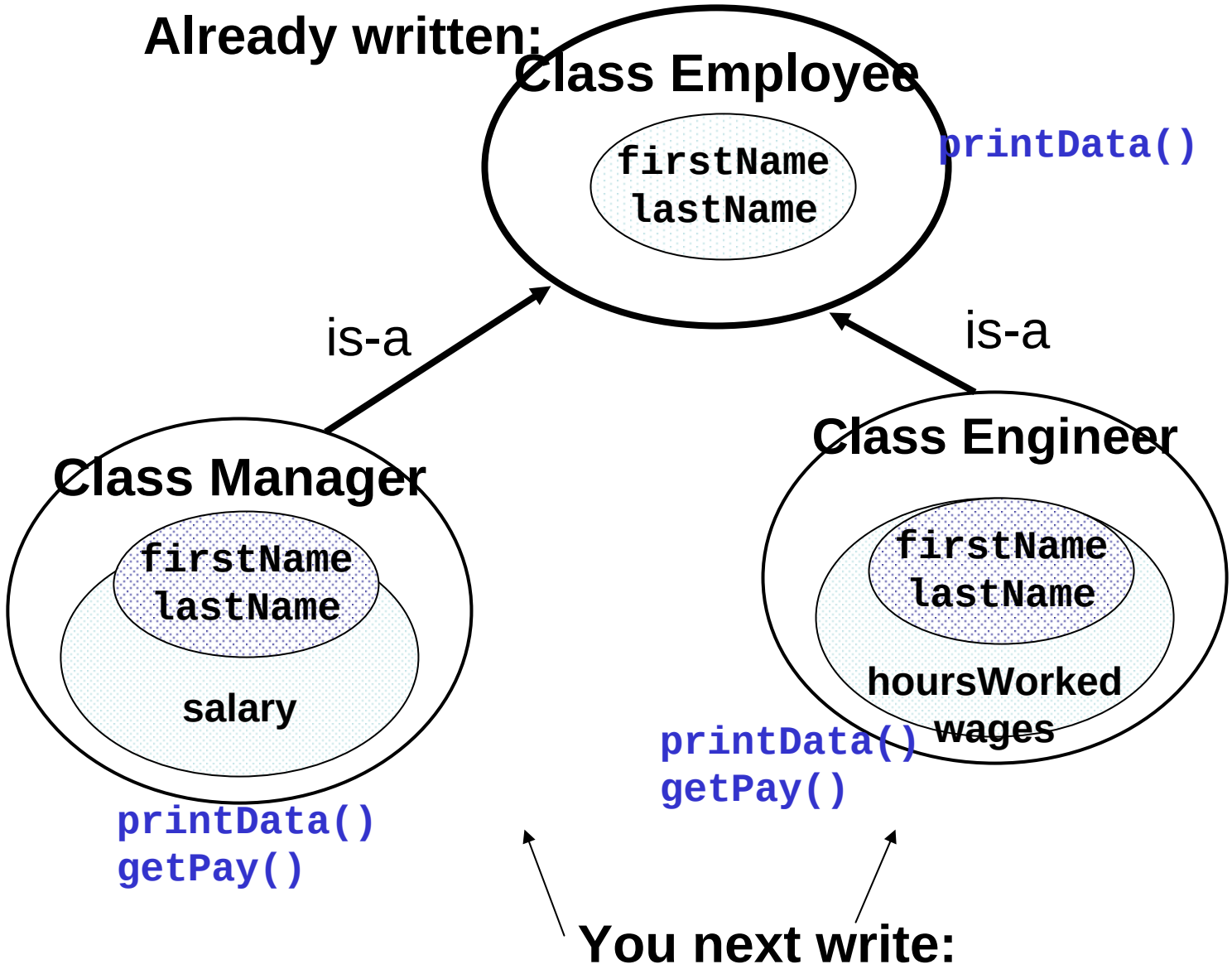
Employee class

This is a simple super or base class.

```
class Employee {
    // Data
    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 class

Subclass or (directly) derived class

```
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 class

Subclass or (directly) derived class

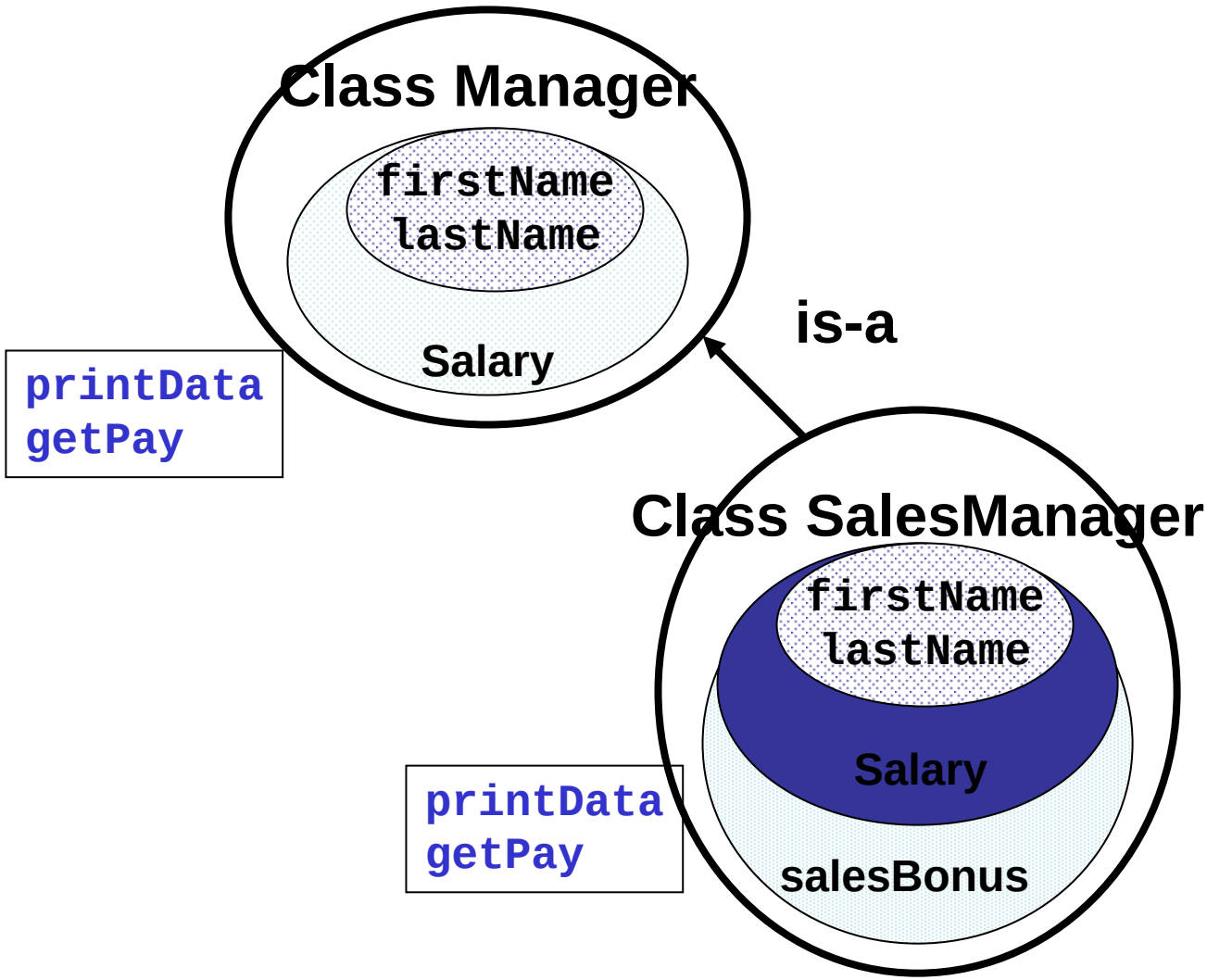
```
class Manager extends Employee {
    private double salary;

    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);}
}
```

Inheritance…



SalesManager Class

(Derived class from derived 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 getPay() {
       return 1250.0; }

   public void printData() {
       super.printData();
       System.out.println("Bonus Pay: $" + bonus; }
}
```

Main method

```
public class PayRoll {
 public static void main(String[] args) {
   // Could get Data from tables in a Database.
   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);


   // Polymorphism, or late binding
   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.

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.