Lecture 12

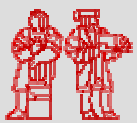
Inheritance

MIT-AITI Kenya 2005
June 28th, 2005

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.

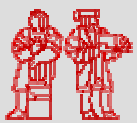
- In software: Object inheritance is more well defined! Objects that are derived from other object "resemble" their parents by ***inheriting*** both state (fields) and behavior (methods).



©2005

Masai Class

```
public class Masai {
    private String name;
    private int cows;

    public Masai(String n, int c) {
        name = n;
        cows = c;
    }

    public String getName() { return name; }

    public int getCows() { return cows; }

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



©2005

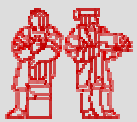
Kikuyu Class

```
public class Kikuyu {
    private String name;
    private int money;

    public Kikuyu(String n, int m) {
        name = n;
        money = m;
    }

    public String getName() { return name; }

    public int getMoney() { return money; }

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

©2005

Problem: Code Duplication

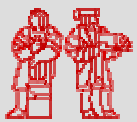
- **`Dog`** and **`Cat`** have the **`name`** field and the **`getName`** method in common
- Classes often have a lot of state and behavior in common
- Result: lots of duplicate code!



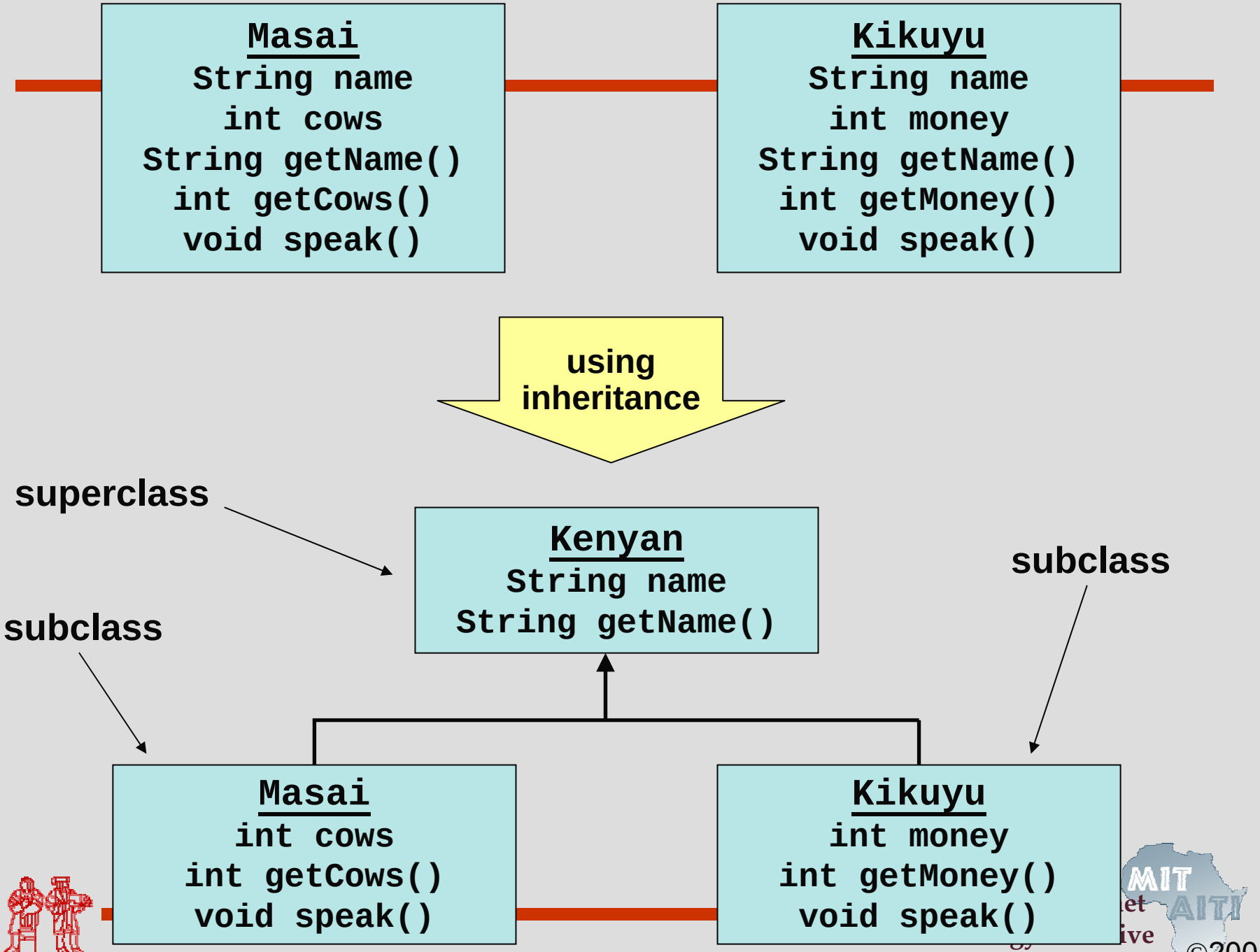
©2005

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**!



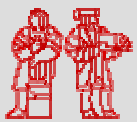
©2005


Masai
String name
int cows
String getName()
int getCows()
void speak()
Kikuyu
String name
int money
String getName()
int getMoney()
void speak()
using inheritance
superclass
Kenyan
String name
String getName()
subclass
subclass
Masai
int cows
int getCows()
void speak()
Kikuyu
int money
int getMoney()
void speak()

©2005

Kenyan Superclass

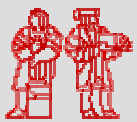
```
public class Kenyan {

    private String name;

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

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



©2005

Masai Subclass

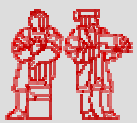
```
public class Masai extends Kenyan {

    private int cows;

    public Masai(String n, int c) {
        super(n); // calls Kenyan constructor
        cows = c;
    }

    public int getCows() {
        return cows;
    }

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



©2005

Kikuyu Subclass

```
public class Kikuyu extends Kenyan {

    private int money;

    public Kikuyu(String n, int m) {
        super(n); // calls Kenyan constructor
        money = m;
    }

    public int getMoney() {
        return money;
    }

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



©2005

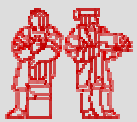
Inheritance Quiz 1

- What is the output of the following?

```
Masai d = new Masai(“Johnson" 23);
Kikuyu c = new Kikuyu(“Sheila", 2200);
System.out.println(d.getName() + " has " +
            d.getCows() + " cows");
System.out.println(c.getName() + " has " +
            c.getMoney() + " shillings");
```

```
Johnson has 23 cows
Sheila has 2200 shillings
```

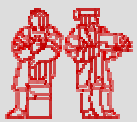
(Masai and Kikuyu inherit the getName method from the Kenyan super class)



©2005

Inheritance Rules

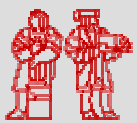
- Use the **`extends`** keyword to indicate that one class inherits from another
- The subclass inherits *all* the fields and methods of the superclass
- Use the **`super`** keyword in the subclass constructor to call the superclass constructor



©2005

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.



©2005

Implicit Super Constructor Call

If I have this **Food** class:

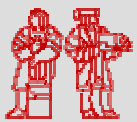
```
public class Food {
    private boolean raw;
    public Food() {
        raw = true;
    }
}
```

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
    }
}
```



©2005

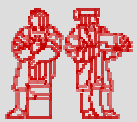
Inheritance Quiz 2

```
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
```



©2005

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());
```

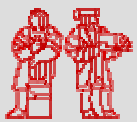
©2005

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;
  }
}
```

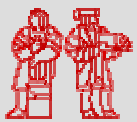
```
class ThermUS extends Therm {

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

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



©2005

Which Lines Don't Compile?

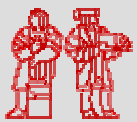
```
public static void main(String[] args) {
  Kenyan a1 = new Kenyan();
  a1.getName();
  a1.getCows();        // Kenyan does not have getCows
  a1.getMoney();       // Kenyan does not have getMoney
  a1.speak();          // Kenyan does not have speak
  Kenyan a2 = new Masai();
  a2.getName();
  a2.getCows();        // Kenyan does not have getCows
  a2.getMoney();       // Kenyan does not have getMoney
  a2.speak();          // Kenyan does not have speak
  Masai d = new Masai();
  d.getName();
  d.getCows();
  d.getMoney();        // Masai does not have getMoney
  d.speak();
}
```



©2005

Remember Casting?

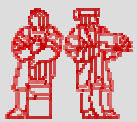
- "Casting" means "promising" the compiler that the object will be of a particular type
- You can cast a variable to the type of the object that it references to use that object's methods without the compiler complaining.
- The cast will fail if the variable doesn't reference an object of that type.



©2005

Which Castings Will Fail?

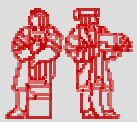
```
public static void main(String[] args) {
   Kenyan a1 = new Kenyan();
   ((Masai)a1).getCows();    //a1 is not a Masai
   ((Kikuyu)a1).getMoney(); //a1 is not a Kikuyu
   ((Masai)a1).speak();      //a1 is not a Masai

   Kenyan a2 = new Masai();
   ((Masai)a2).getCows();
   ((Kikuyu)a2).getMoney(); //a2 is not a Kikuyu
   ((Masai)a2).speak();

   Masai d = new Masai();
   ((Kikuyu)d).getMoney();  //d is not a Kikuyu
}
```



©2005

Programming Example

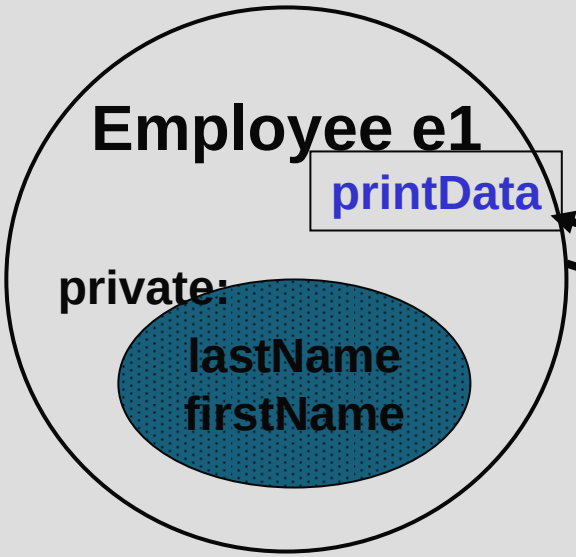
- **A Company has a list of Employees. It asks you to provide a payroll sheet for all employees.**
  - Has extensive data (name, department, pay amount, …) for all employees.
  - Different types of employees – manager, engineer, software engineer.
  - You have an old Employee class but need to add very 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.



©2005

REVIEW PICTURE

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



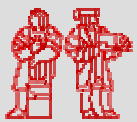

```
public … Main(…){
Employee e1…("Mary","Wang");
...
e1.printData();
// Prints Employee names.
...
}
```


©2005

Employee class

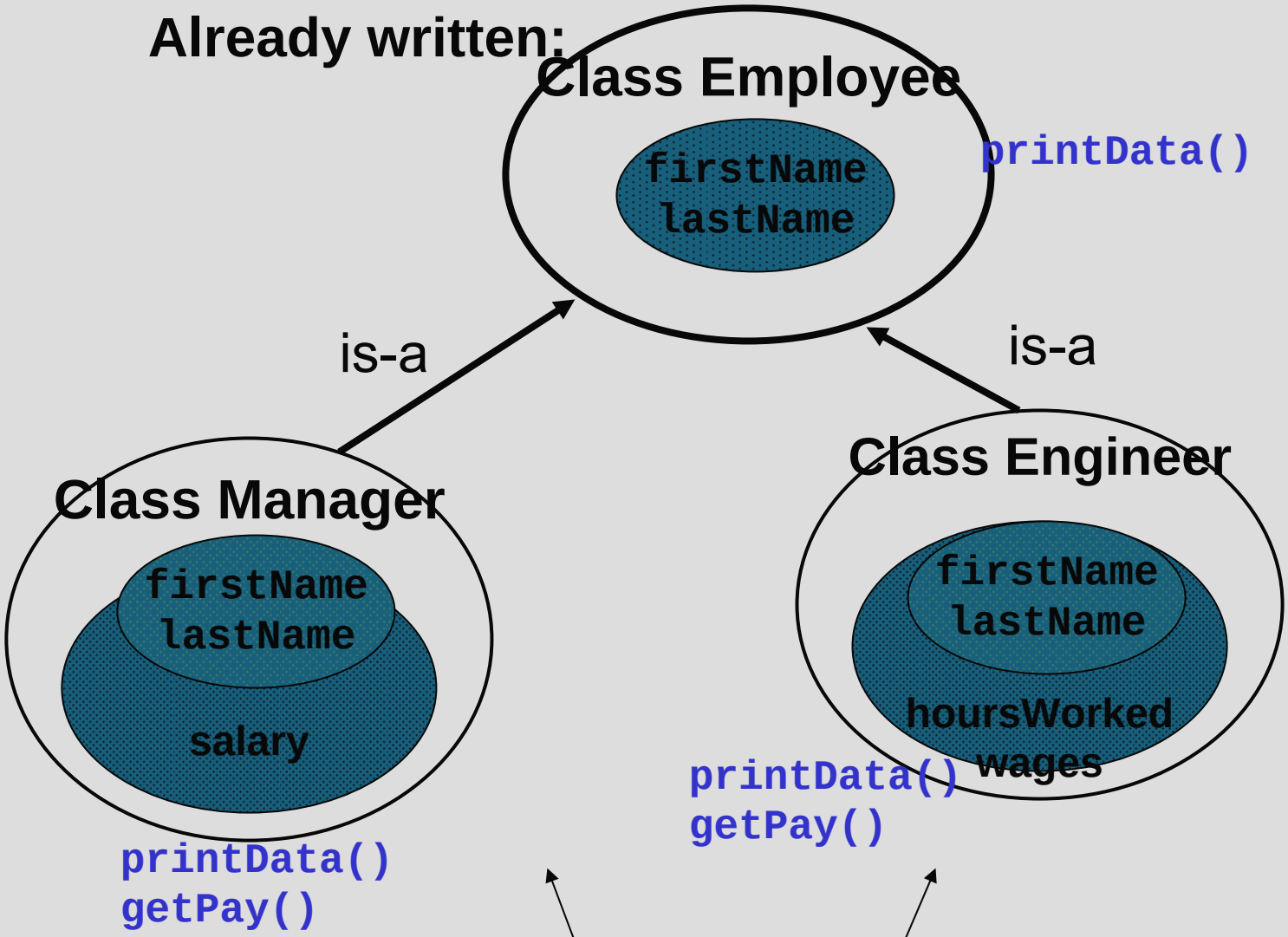
This is a simple super or base class.

```
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);}
}
```



©2005

Inheritance



©2005

Engineer class

Subclass or (directly) derived class

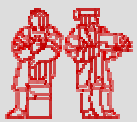
```
class Engineer extends Employee {
    private double wage;
    private double hoursWorked;
    public Engineer(String fName, String lName,
                    double rate, double hours) {
        super(fName, lName);
        wage = rate;
        hoursWorked = hours;
    }
    public double getPay()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

©2005

Manager class

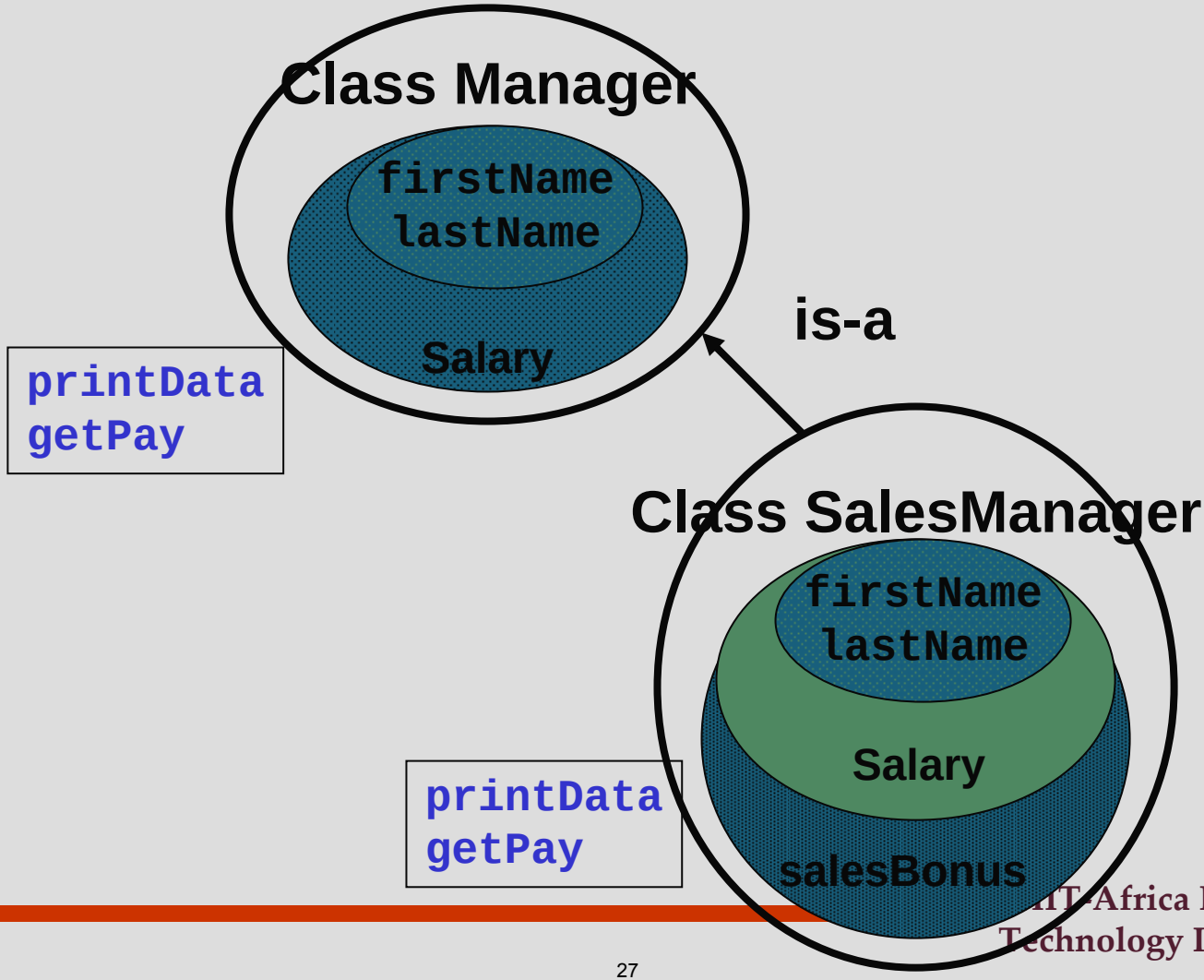
Subclass or (directly) derived class

```
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);}
}
```



©2005

Inheritance…



MIT Africa Internet Technology Initiative
©2005

SalesManager Class

(Derived class from derived class)

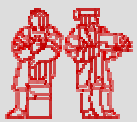
```
class SalesManager extends Manager {
   private double bonus;     // Bonus Possible as commission.

   // 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; }
}
```

©2005

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



©2005

Output from main method

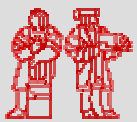
```
Fred Smith
Weekly pay: $96.0
Ann Brown
Monthly salary: $1500.0
Mary Barrett
Monthly salary: $1250.0
Bonus: $2000.0
```

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.



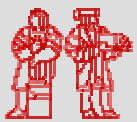
©2005

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.



©2005

MIT OpenCourseWare
http://ocw.mit.edu

EC.S01 Internet Technology in Local and Global Communities
Spring 2005-Summer 2005

For information about citing these materials or our Terms of Use, visit: http://ocw.mit.edu/terms.