CS 145

Chapter 9 – Polymorphism

Inheritance

- On a non-computer level, what is the meaning of the word **INHERITANCE?**

- But what does that mean for computer and how do we use it.

Code Reuse

Hierarchies of related objects

Take a look

- Imagine we start with the following class. What can you tell me

```
public class Employee
{
	public String name;

	public double getHours()
	{  return 40;}

	public String getJob()
	{   return name + " and I work here. "; }
}
```

A special Employee

- Great, but what if we want to have a class for the Boss

```
public class Boss
{
     public String name;

     public double getHours()
     {  return 40;}

     public String getJob()
     {   return name + " and I am the boss. "; }
}
```

Compare and Contrast

```
public class Employee
{
 public String name;

 public double getHours()
      {  return 40;}

  public String getJob()
  { return name + " and I
work here.“; }
}
```

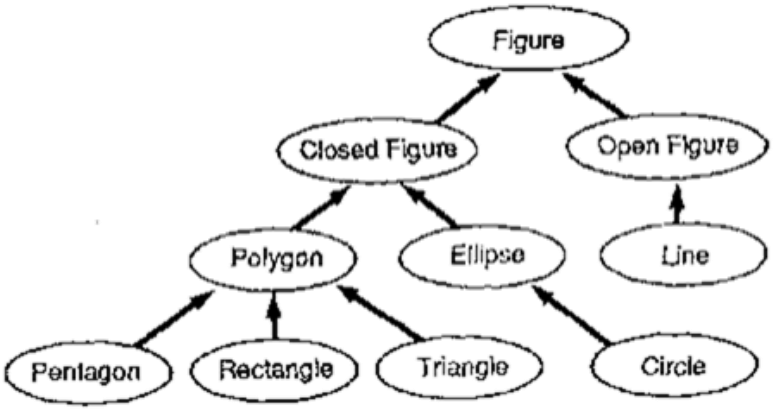
```
public class Boss
{
  public String name;

  public double getHours()
      {  return 40;}

  public String getJob()
   {   return name + " and I am
the boss.“; }
}
```

Boss **is a** "type" of employee

Is-a relationships, hierarchies

- **is-a relationship**: A hierarchical connection where one category can be treated as a specialized version of another.
  - every marketer *is an* employee
  - every legal secretary *is a* secretary

- **inheritance hierarchy**: A set of classes connected by is-a relationships that can share common code.



What we DON'T want to do…

```
public class Employee
{
 public String name;
 public boolean isBoss;
 ...
  public String getJob()
  { if (isBoss) // do this.
    else // do this.  }
}
```

What we DON'T want to do…

```
public class Employee
{
 public String name;
 public boolean isBoss, isSect, isManager
                isJanitor, isTemp, isFired;
 ...
  public String getJob()
  { if (isBoss) // do this.
     else (isSect) // do this.
     else (isManager) // do this.
     else (isJanitor) // do this.
```

So instead of…

```
public class Boss
{
  private String name;

  public double getHours()
     {  return 40;}

  public String getJob()
     {   return name + " and I am the
boss." }
}
```

Inheritance

- **inheritance**: A way to form new classes based on existing classes, taking on their attributes/behavior.
  - a way to group related classes
  - a way to share code between two or more classes

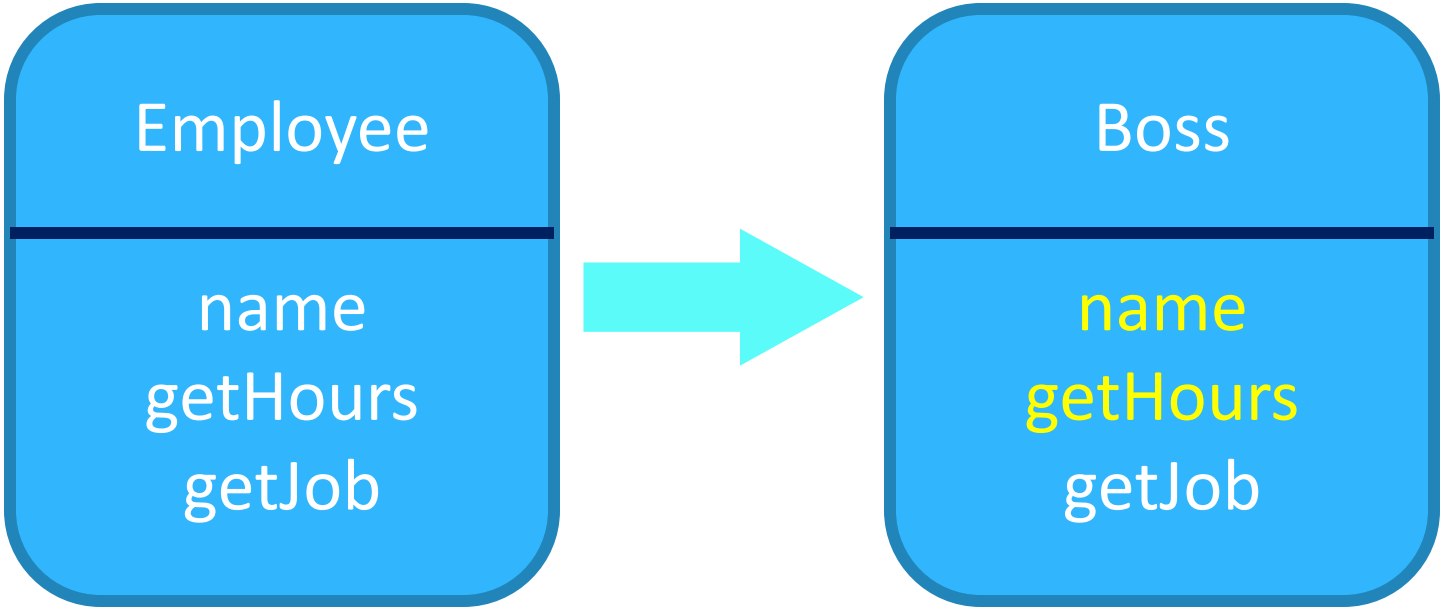
- One class can *extend* another, absorbing its data/behavior.
  - **superclass**: The parent class that is being extended.
  - **subclass**: The child class that extends the superclass and inherits its behavior.
    - Subclass gets a copy of every field and method from superclass

So instead of…

```
public class Boss extends Employee
{
   public String getJob()
    {   return name + " and I am the boss." }
}
```

- Wait… what happened to
  - `private String name;`
  - `public double getHours()`

Inheritance



The SUPERCLASS

The SUBCLASS

Override

- To write a new version of a method in a subclass that replaces the superclass's version
- An overridden method must have the exact same structure as the original method, so that is can be run instead. (no special syntax required. Just write a new version of it in the subclass)
- This allows you to change the action without changing how it is called.
  - *Note, There is a difference between **override** and **overload**.*

Overriding

```
class Dog{
    public void bark(){
        System.out.println("woof ");
    }
}
class Hound extends Dog{
    public void sniff(){
        System.out.println("sniff ");
    }

    public void bark(){
        System.out.println("bowl");
    }
}
```

Same Method Name,
Same parameter

Overloading

```
class Dog{
    public void bark(){
        System.out.println("woof ");
    }

    //overloading method
    public void bark(int num){
        for(int i=0; i<num; i++)
            System.out.println("woof ");
    }
}
```

Same Method Name,
Different Parameter

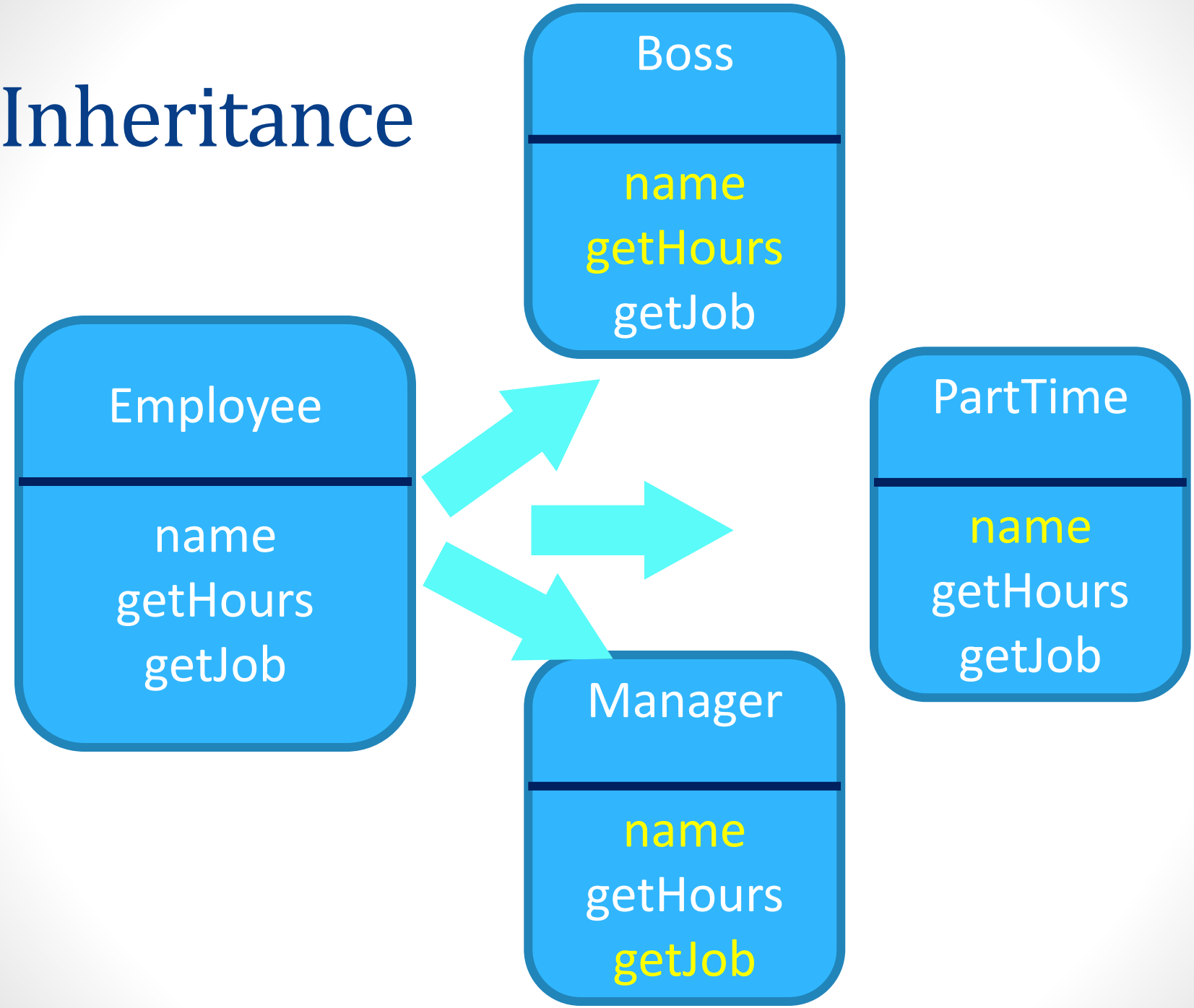
A special Employee #2

```
public class Manager extends Employee
{
     public double getHours()
     {  return 60;}

}
```

A special Employee #3

```
public class PartTime extends Employee
{
     public double getHours()
     {  return 20;}
    public String getJob()
    {   return name + ":Part time"; }
}
```

Inheritance



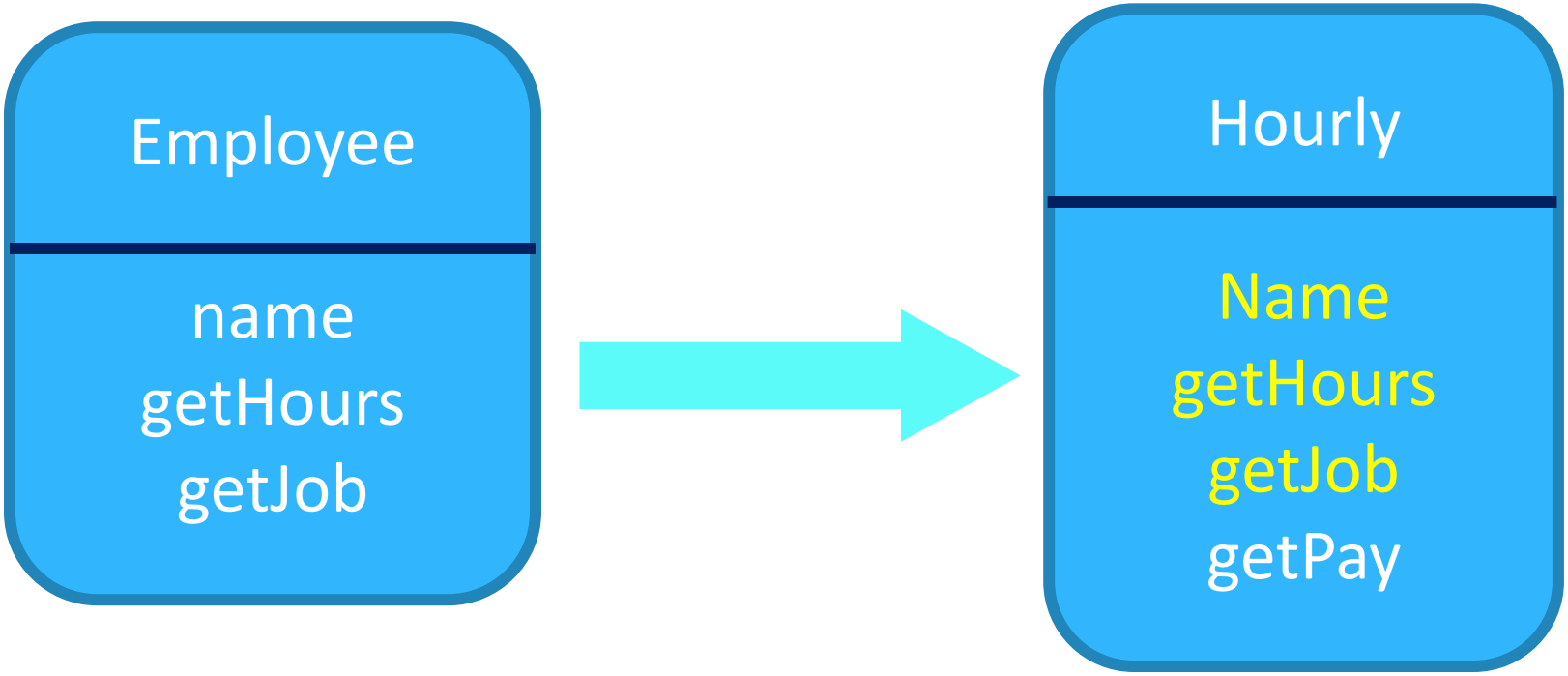
Note

- It is okay not to override anything
- Or you can override everything.
- A subclass can override every class method, or can be exactly the same. You can also add methods for only special subclasses

A special Employee #4

```
public class Hourly extends Employee
{
    public double getPay()
    {   return getHours() * 10.40 }
}
```

Inheritance



Interacting with the SuperClass

9.2

Changes to common behavior

- Let's return to a version of the company/employee example.

- Imagine a company-wide change affecting all employees.

  Example: Everyone is given an extra day of vacation.
  - The base employee vacation is 5 days +1 = 6 days.
  - Legal secretaries get 2 days more than a normal employee.
  - Marketers get 4 days more than a normal employee.

- We must modify our code to reflect this policy change.

Modifying the superclass

```
// A class to represent employees

public class Employee {
    public int getHours() {
        return 40;              // works 40
    }

    public double getVacation() {
        return 5 + 1;           // 5 + bonus
     }

    ...
}
```

- Are we finished?

An unsatisfactory solution

```
public class LegalSecretary extends Employee {
    public double getVacation() {
        return 5 + 2 + 1;
    }
    ...
}

public class Marketer extends Employee {
    public double getVacation() {
        return 5 + 4 + 1;
    }
    ...
}
```

- Problem: The subclasses' vacation days are based on the Employee vacation days, but the `getVacation` code does not reflect this.

Calling overridden methods

- Subclasses can call overridden methods with `super`

  `super.method(parameters)`

  - Example:

```
public class LegalSecretary extends Employee {
    public double getVacation() {

        double baseVacation = super.getVacation();
        return baseVacation + 2 ;
    }
    ...
}
```

Improved subclasses

```
public class Marketer extends Employee {

        public double getVacation() {
        return super.getVaction() + 4;
    }
}

public class Lawyer extends Marketer {

    public int getVacation () {
        return super.getVacation () * 2;
    }
}
```

```
public class Marketer extends Employee {
        public double getSalary () {
            return super.getSalary()+10000;
        }
        public void advertise () {
                System.out.println ("Act now, while supplies last!");
        }
}
```

```
public class Janitor extends Employee {
    public int getHours () {
            return super.getHours () * 2;
    }

    public double getSalary () {
        return super.getSalary()-10000;
    }

    public int getVacationDays (){
        return super.getVacationDays()/2;
    }
    public void clean() {
        System.out.println ("Workin' for the man.");
    }

}
```

```
public class HarvardLawyer extends Lawyer {
    public int getVacationDays () {
        return super.getVacationDays() + 3;
    }
    public double getSalary () {
        return super.getSalary () * 1.2;
    }
    public String getVacationForm () {
        return super.getVacationForm()+super.getVacationForm() + super.getVacationForm() + super.getVacatio
```