Subclasses

Concepts:

- The subclass and inheritance: subclass B of class A inherits fields and methods from A. A is a superclass of B. Keyword **extends** is used to define a subclass.
- Using the constructor of a superclass.
- Access modifier **protected**
- Overriding methods

Readings from Weiss:

classes, etc.: Sections 3.1-3.5

subclasses: Section: 4.1-4.2

packages: Section 3.6

Class Employee

```
// An instance of Employee contains a person's name,
// salary, and year  hired. It has a constructor and
// methods for raising the salary, printing the data, and
// retrieving the person's name and the year hired.
public class Employee {
  private String name;      // The person's name
  private double pay;       // The person's yearly salary
  private int hireDate;     // The year hired

  // Constructor: a person with name n, salary s, and
  // year d hired
  public Employee(String n, double s, int d)  {
       name= n;
       pay= s;
       hireDate= d;
  }

  // = the person's name
  public String getName()
     {return name;}
```

Class Employee, continued

```
  // Raise this Employee s salary by p perce nt
  public void raiseSalary(double p)
      {pay= pay * (1 + p/100.0);}

  // = this Employee s pay
  public double getPay()
      { return pay; }

  // = the year this Employee was hired
  public int getHireDate()
      {return hireDate;}

// = a String containing the data for this Employee
  public String toString()  {
     return name  + "  "  +
              pay  + "  "  +
              hireDate;
    }
}
```


Our task

Modify class Employee to take into account three different kinds:

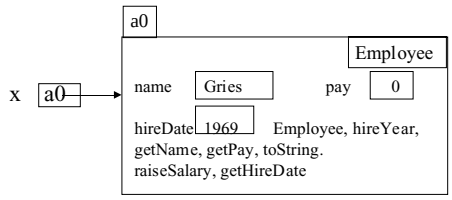
- **VIPS** (e.g. CEO, president, vice president): Need a field *bonus*, since VIPS get (big) bonuses. Get a yearly salary.
- **Salaried**: Expected to work overtime whenever necessary, without extra pay.
- **Regular**: Time cards! Have an hourly wage instead of a yearly salary. (Need also to record the number of hours worked on each day, but we ll forego that here and assume 40 hours per week.

Subclass VIP

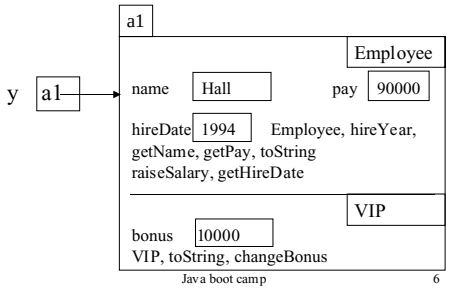
```
// An instance of VIP contains a VIP's data
public class VIP extends Employee {
  private double bonus;    // The VIP's bonus

  // Constructor: a VIP with name n, year d hired,
  // yearly pay s, and bonus b
  public VIP(String n, int d, double s, double b) {
     super (n, s, d);
     bonus= b;
  }

  // = a String containing the data for this VIP
  public String toString()  {
      return  VIP   + getName() +       + getPay() +
              +  getHireDate() +     +   bonus;
  }

  // Change this VIP s bonus to b
  public void changeBonus(double b)
     {bonus= b;}
}
```


An instance of class VIP has every field and method that an instance of class Employee does, plus the ones that are declared in VIP.

Employee x; **new** Employee(Grie s , 1969);



Employee y= **new** VIP(Hall ,1994, 90000,1000);



```
// An instance of Salaried contains a salaried person s data
public class Salaried extends Employee {
  // Constructor: instance with name n, hire date d, yearly
  // pay s
   public Salaried(String n, double s, int d, double b)
      { super (n, s, d); }

   // = a String containing the data for the person
   public String toString()
       {  return  Salari ed   + super.toString();  }

  // = yearly pay
  public double getPay()
      {  return super.getPay();  }
}
```

Important points:
(1) Overriding a method of the superclass.

(2) Calling a constructor of the superclass.

(3) calling other methods of the superclass.


```
// An instance of Hourly contains an hourly person s data
public class Hourly extends Employee {
  // private double hourlyPay; // pay for one hour
  // private int numHours;       // no. hours worked in year

 /* Try writing suitable methods to finish this class. You
     need a suitable constructor, methods to access private
     fields, a toString method, and function getPay.

     Your constructor should initialize the inherited
     fields correctly!!! (to the product of hourlyPay and
     numHours).

     Place all four classes --Employee, VIP, Salaried, and
     Hourly-- into files and compile them in a Java project,
     so that you can get the syntax bugs out of your code.

 */
}
```

Use of protected

A **public** field of Employee can be referenced from anywhere.

A **private** field can be referenced only from instances of Employee.

A **protected** field can be referenced only in the same package and in subclasses.

A field without a modifier can be referenced only in the same package.

Packages are not discussed now!

For now, use **protected** for instance variables that you want a subclass to be able to reference.

Referencing methods

Rule: When looking for a nonstatic method in an instance, start at the bottom of the instance and search upwards. In other words, the method called is the lowest one in the class hierarchy in the instance.

VIP y;

y= **new** VIP(Hall , 1983, 90000, 1000);

y.getName() refers to method getName of its superclass, Employee. This is because getName is not defined in VIP.

y.getBonus() refers to method getBonus of VIP. This is because getBonus is defined in VIP.

y.toString() refers to method toString of VIP. This is because toString is defined in VIP.

Method toString of superclass Employee has been **overridden** in class VIP.

Calling an overridden method

```
public class VIP {
 private double bonus; // ...

  // = a String containing the data for this VIP
  public String toString()  {
     return  VIP    + getName  ()  +        + getPay()  +
              + getHire  Date() +      +   bonus;
  }
```

There is already toString in superclass Employee. How to call it?

```
  // = a String containing the data for this VIP
  public String toString()  {
     return  VIP    +  super.toString() +      +  bonus;
  }
```

To call an overridden method, prefix the method name with

super.

Principle: rely as much as possible on fields and methods of the superclass.

Summary of this

Assume an instance a0 of some class contains a method m. Within the body of m: **this** refers to instance a0.

(0) You can precede a reference to any field or method by **this** . , without chan ging its meaning:

this.toString() is equivalent to toString()

Also: // Set this Employee s pay to pay

```
public void setPay(double pay)
   {  this.pay= pay;  }
```

(1) You can use **this** as an argument, to pass the name a0 as an argument:

bigger(**this**, e)

(2) You can use **this** to refer to another constructor:

```
// Constructor: an Employee with name n, pay d,
// and hire date 2001
public Employee(String n, double d)
  { this (n, d, 2000); }
```

Summary of super

Assume an instance of some subclass C contains a method m. Within the body of m:

super refers to the superclass of C.

(0) You can precede a reference to a method by **super** . to refer to the method of the superclass (instead of the overriding method toString in C, if there is one). This is a way to override the overriding method.

```
super.toString()
```

(2) You can use **super** to refer to a constructor of the superclass, to initialize the fields of the superclass:

```
// Constructor: a VIP with name n, pay d,
// hire date 2001, and bonus b
public VIP(String n, double d, int b) {
  super(n, d, 2001);
  bonus= b;
}
```


Class Object; the class hierarchy

Class Object is automatically the superclass of all classes that don t extend anything. In our example, we have:

```
Object
    Employee
        VIP
        Salaried
        Regular
```

Class object has (at least) two methods

```
// = thi s Object has the same name as b
//   (we give a possible implementation)
public boolean equals(Object b) {
    if ( b == null)
        return false;
    return this == b;
}
// = a String representation of this Object
public String toString()
```