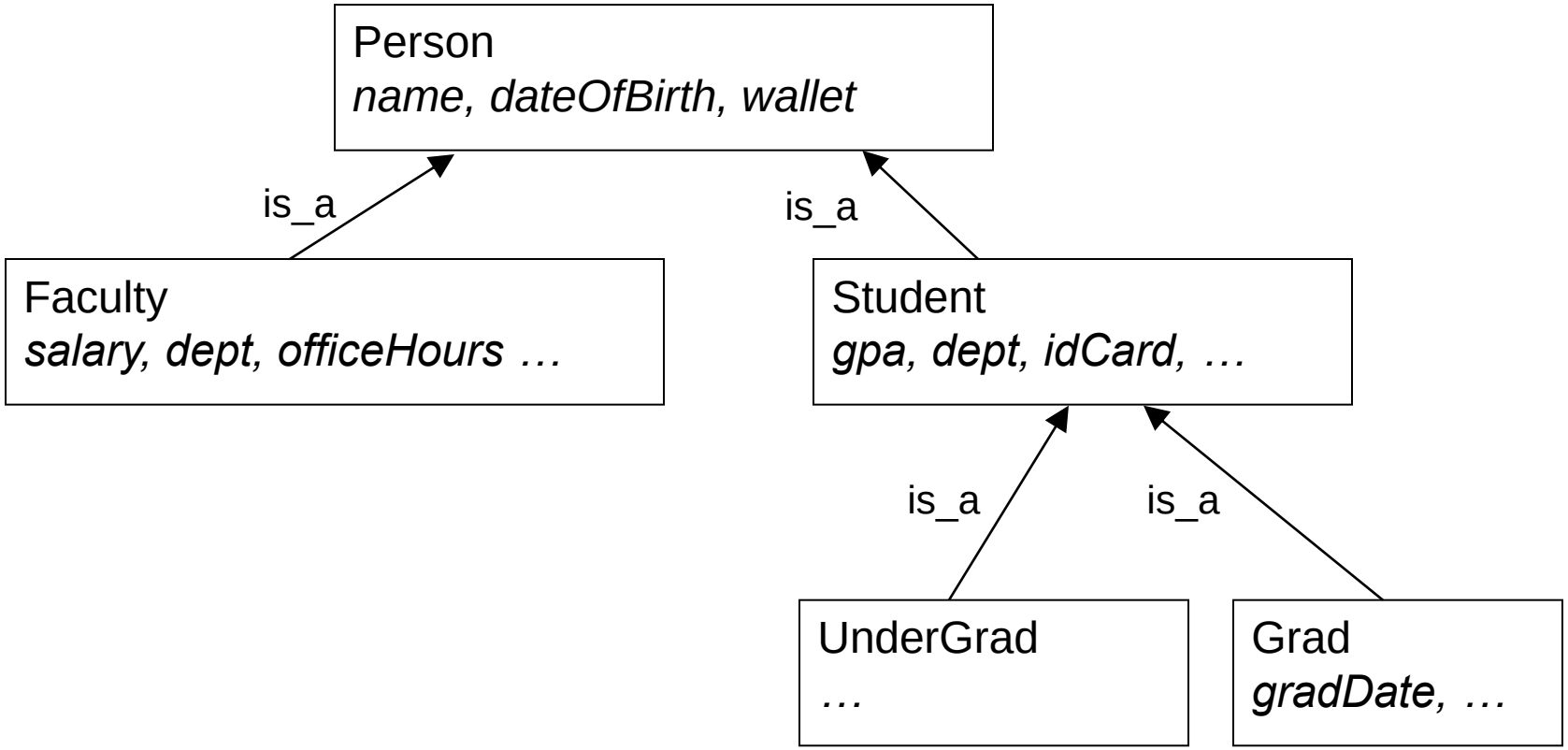
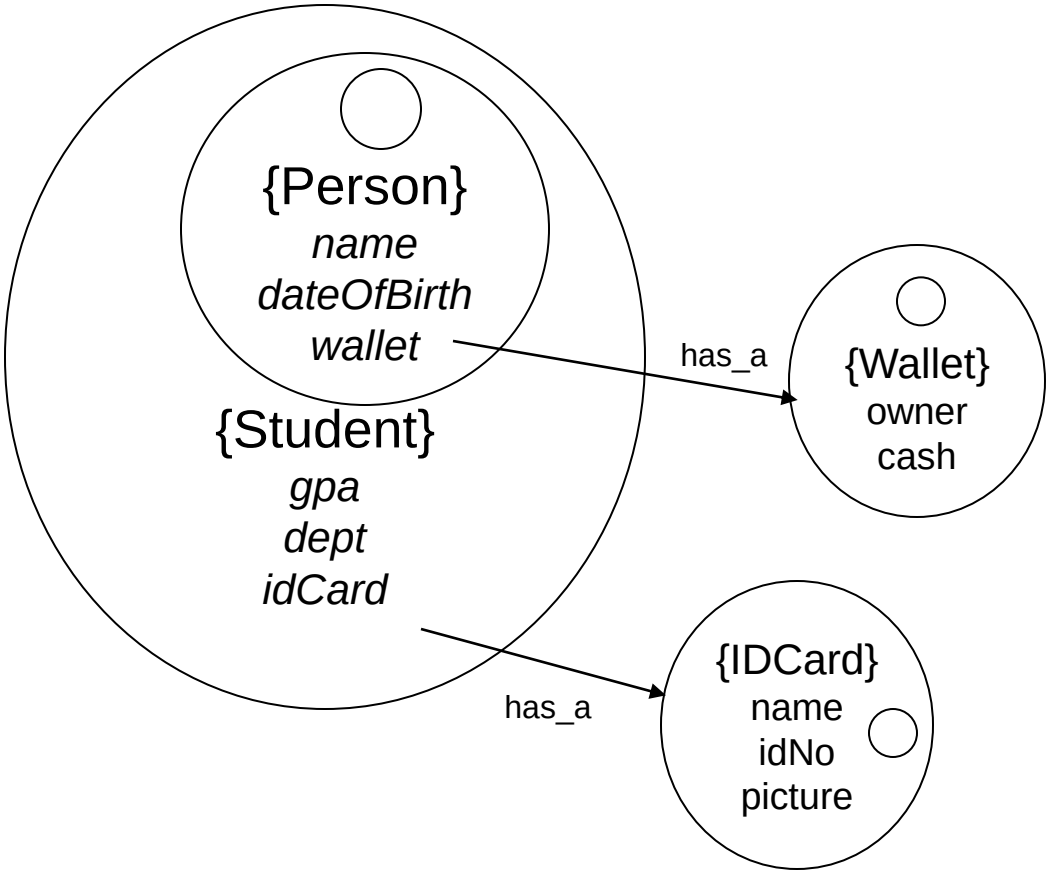
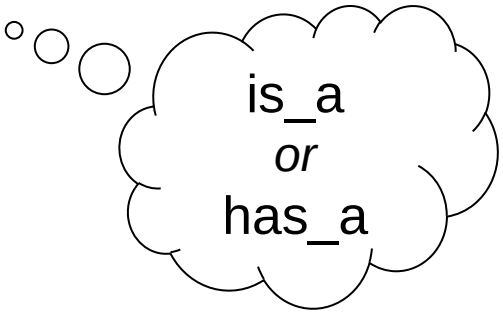
OOP Summary

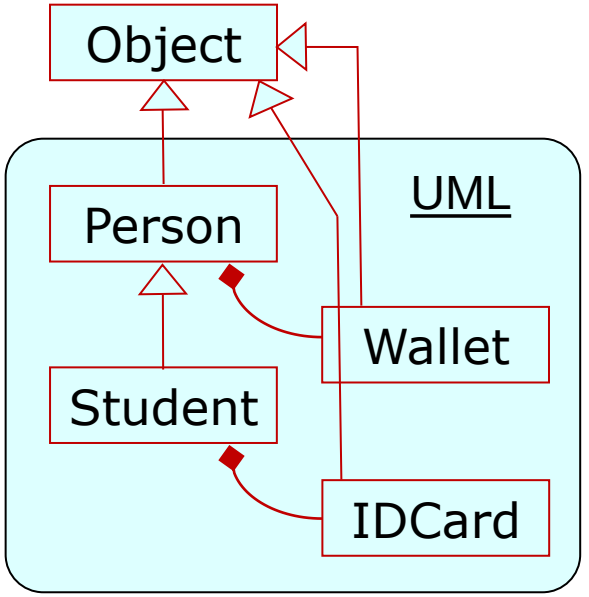
Inheritance Hierarchy



Inheritance & Composition







Example Java Code

```
public class Person {

        String  name;
        Date    dateOfBirth;
        Wallet  wallet;

        public Person ( String name, Date dob) {
             this.name = name;
             dateOfBirth = dob;
             wallet = null;
        }

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

Example Java Code

```
public class Student extends Person {

        double  gpa;
        String   dept;
        IDCard  id;

        public Student ( String name, Date dob,
                                IDCard id, String dept) {
            super( name, dob);
            this.id = id;
            this.dept = dept;
            gpa = 0;
        }

        public double getGpa() {
            return gpa;
        }
}
```

Student is_a Person

With additional properties

Call parent constructor to save writing code again!

Additional methods for new properties

Have direct access to non-private properties & methods of parent classes

Sub-class Constructors

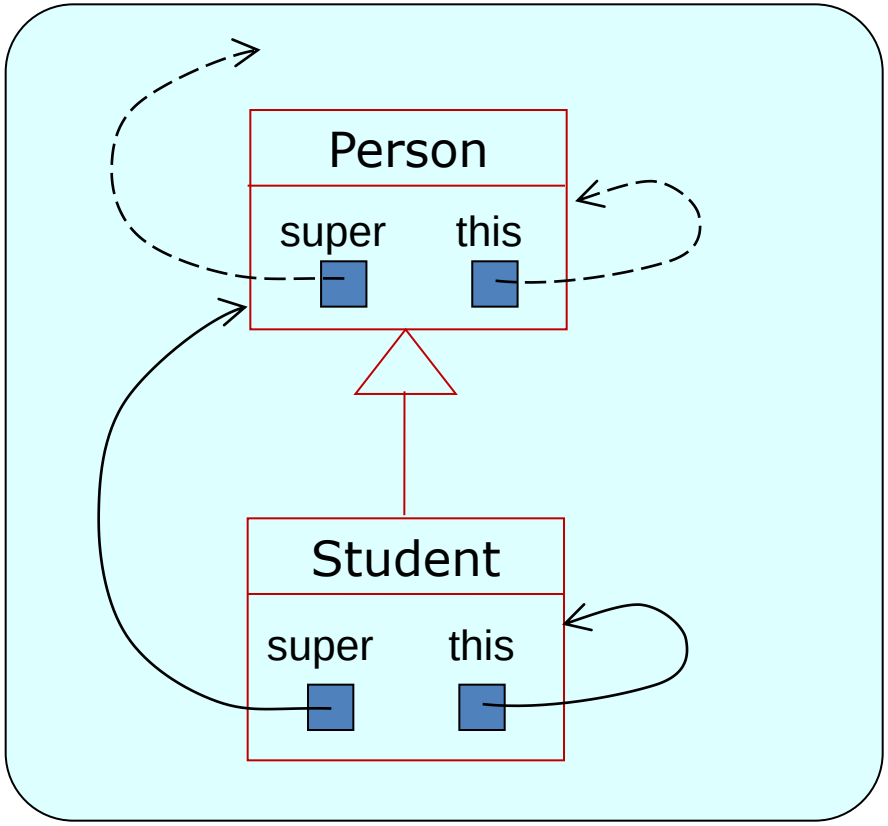
- Call to parent (super) constructor must be first statement in constructor (Java builds instances in layers, inside to out)

- If omitted Java automatically calls defaults constructor (one with no param's)

Note: Java classes can only have one parent.

Java is a *single-inheritance* language, as opposed to a *multiple-inheritance* language (such as C++) which can have many parents!

super vs. this



super refers to non-private constructors, properties & methods in parent class

this refers to properties & methods in current class

Extended Type Checking

- Can now match object of type or sub-type
- Distinguish type of reference vs. type of object

```
Object      x;      // can hold anything
Person      y;      // can hold Person, Student, …
Student     z;      // only Student and sub-classes
```

```
y = new Person( _____);
z = new Student( ____);

y.getName();

z.getName();
```

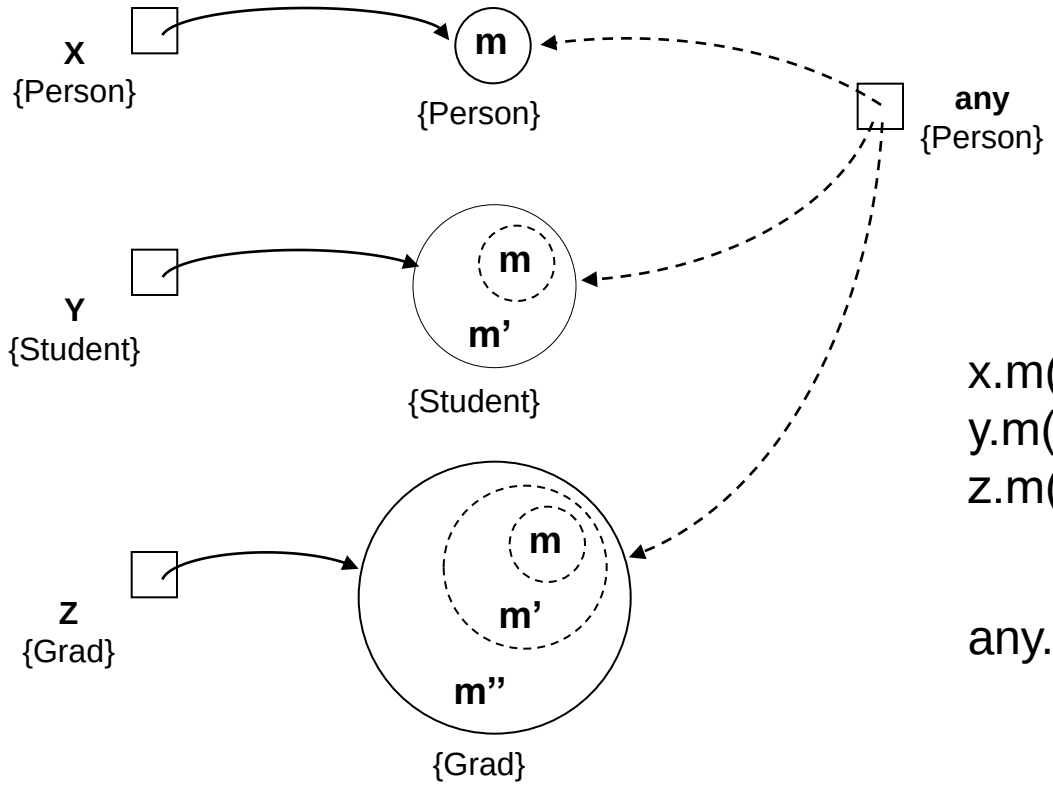
```
z.getGPA();

y.getGPA();

( (Student) y ).getGPA();
```

Polymorphism

- Everyone does *it* their way!

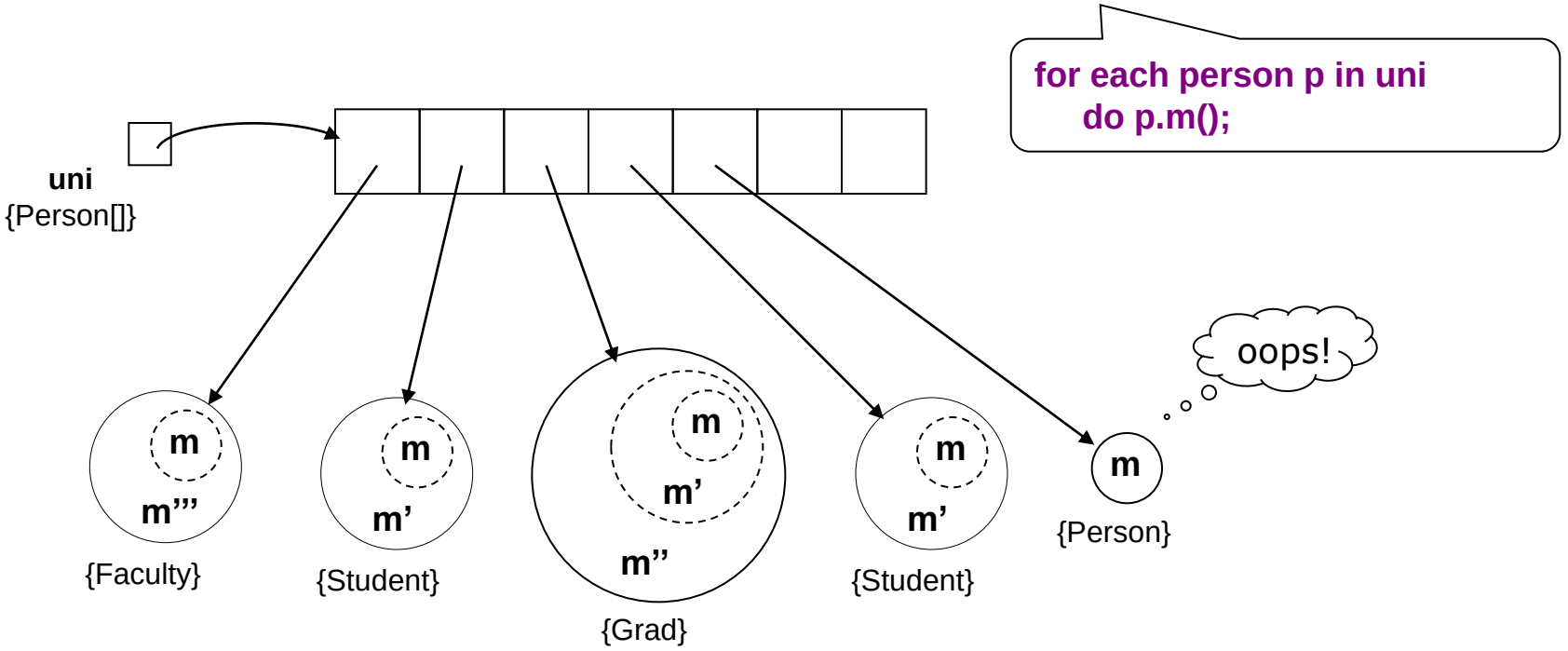



```
x.m();  // does m
y.m();  // does m'
z.m();  // does m''

any.m();  // depends on
                object type!
```

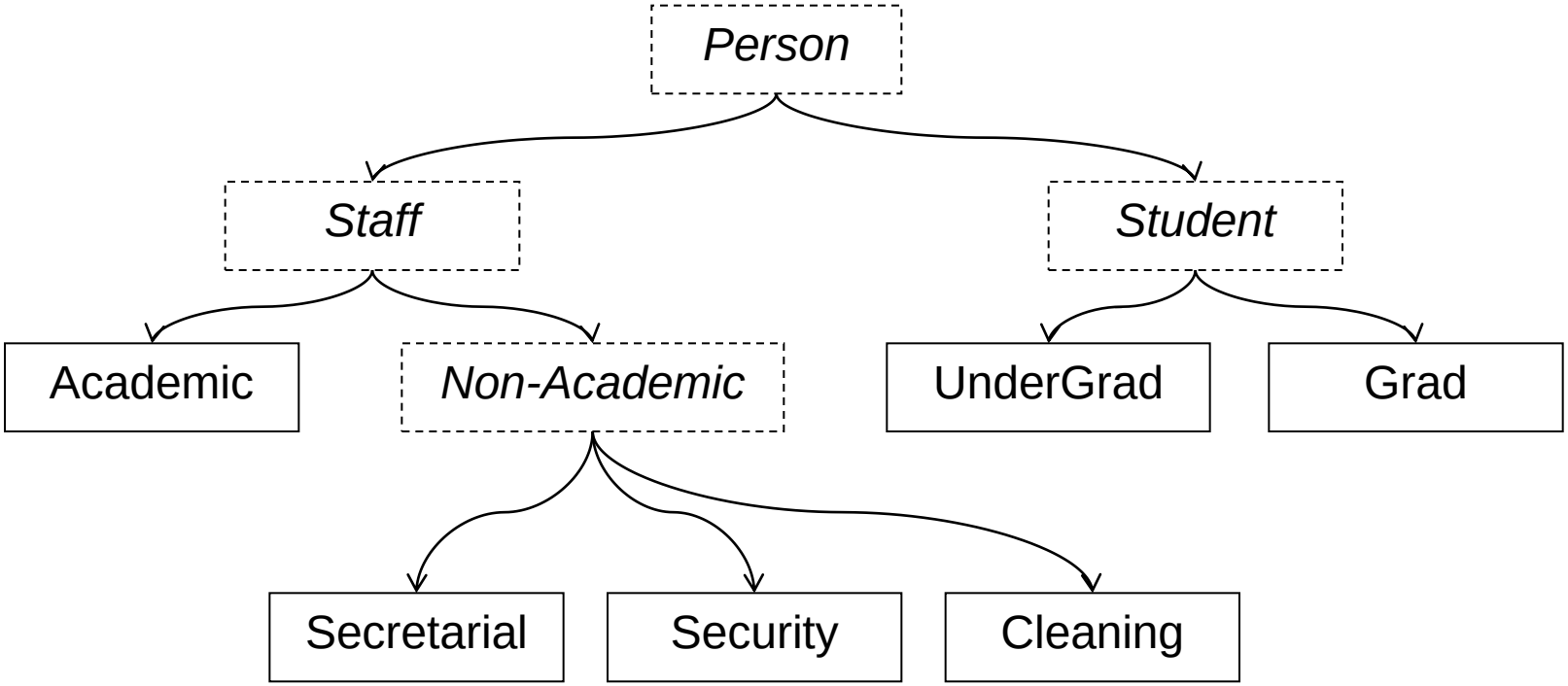
Polymorphic collections

- Objects in a collection all do *it* their way!



University People…

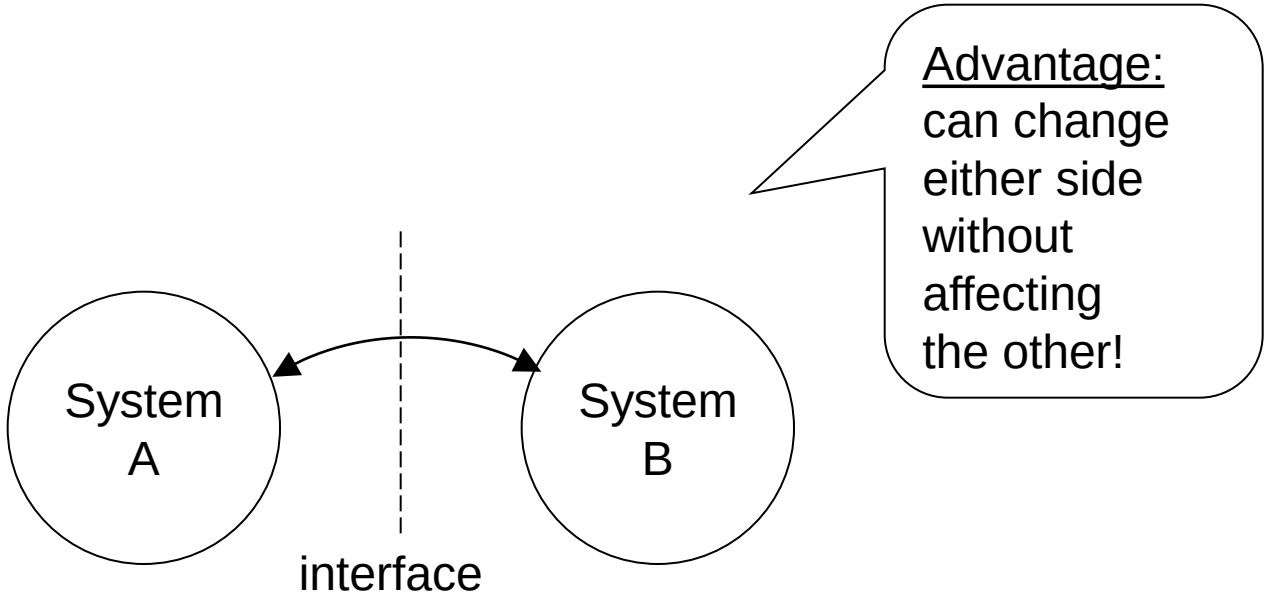
- Abstract vs. Concrete classes



Interfaces

- An interface

  is the boundary between two systems
  through which they connect/communicate
- Often standardised



Java Interfaces

- Declare with *public interface X {…}*
  - restricted to constants & abstract methods only
  - cannot be instantiated
  - guarantees any *implementing* class has specified methods (else cannot be instantiated)

- Classes extend one class & implement interfaces

  e.g. public class U extends V implements X,Y,Z {…}

- Can view as a special form of class

  so class U is_a V, is_a X, is_a Y, is_a Z

  i.e. a form of multiple inheritance

Simple example…

```
public interface Pointable {
   public boolean contains( int x, int y);
}

public class Circle extends TwoDShape implements Pointable {

   int   radius;

   public Circle( int radius) {
      super();
      this.radius = radius;
   }

   public int getRadius() {
      return radius;
   }

   public boolean contains( int x, int y) {
      // set result true iff x,y inside circle...
      return result;
   }
}
```

Define *interface* with abstract method

Define class that *implements* interface

Required method... won't compile if omitted

Another example…

- Multiple inheritance?

Notebook
- *is_a* Computer
- *is_a* ElectricalDevice
- *is_a* NetworkedDevice

ElectricCooker
- *is_a* Cooker
- *is_a* ElectricalDevice

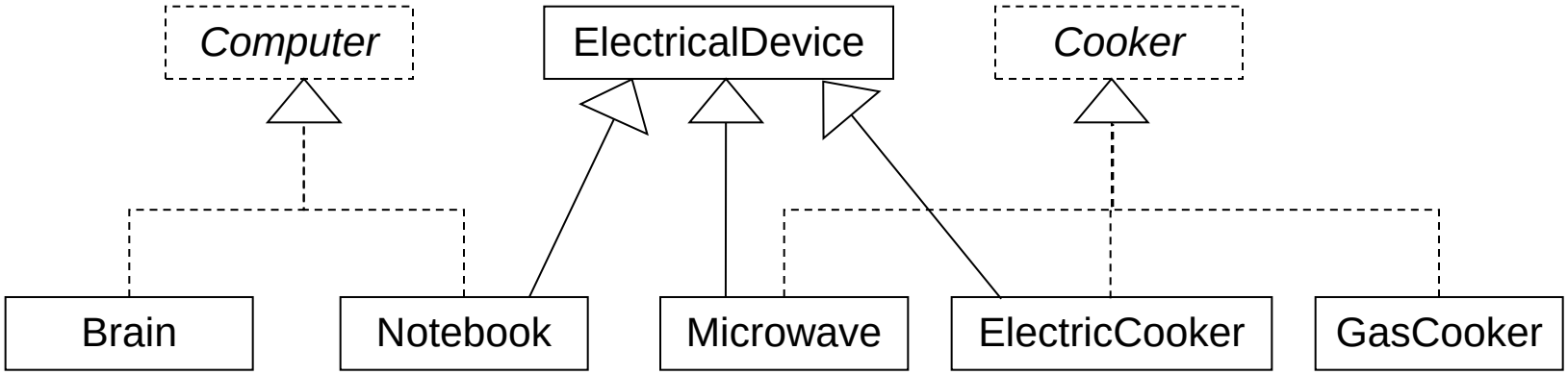
Microwave
- *is_a* Cooker
- *is_a* ElectricalDevice

SmartPhone
- *is_a* Phone
- *is_a* Computer
- *is_a* NetworkedDevice

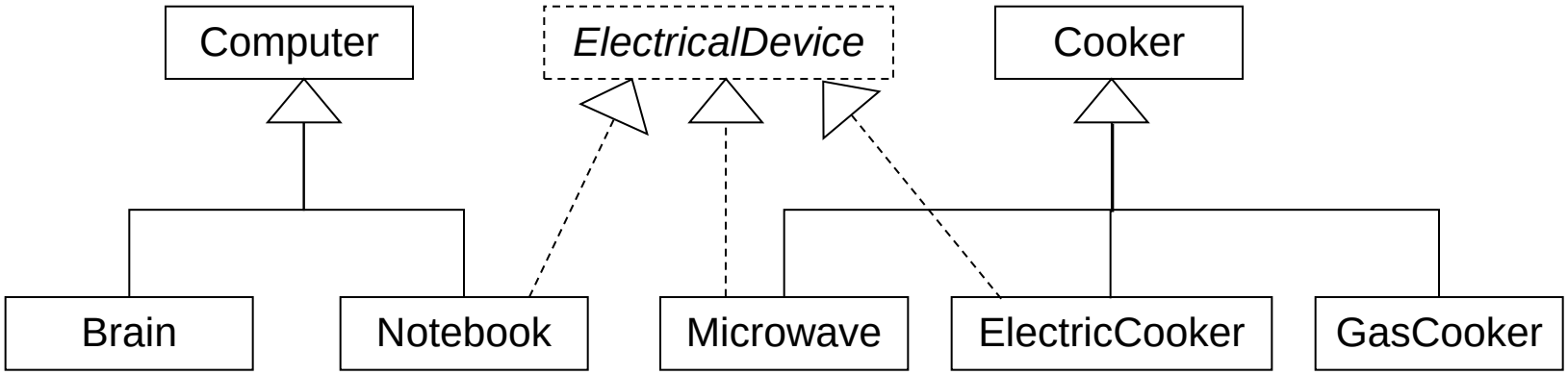
Brain
- *is_a* Computer
- *is_a* BiologicalDevice

GasCooker
- *is_a* Cooker
- *is_a* GasDevice

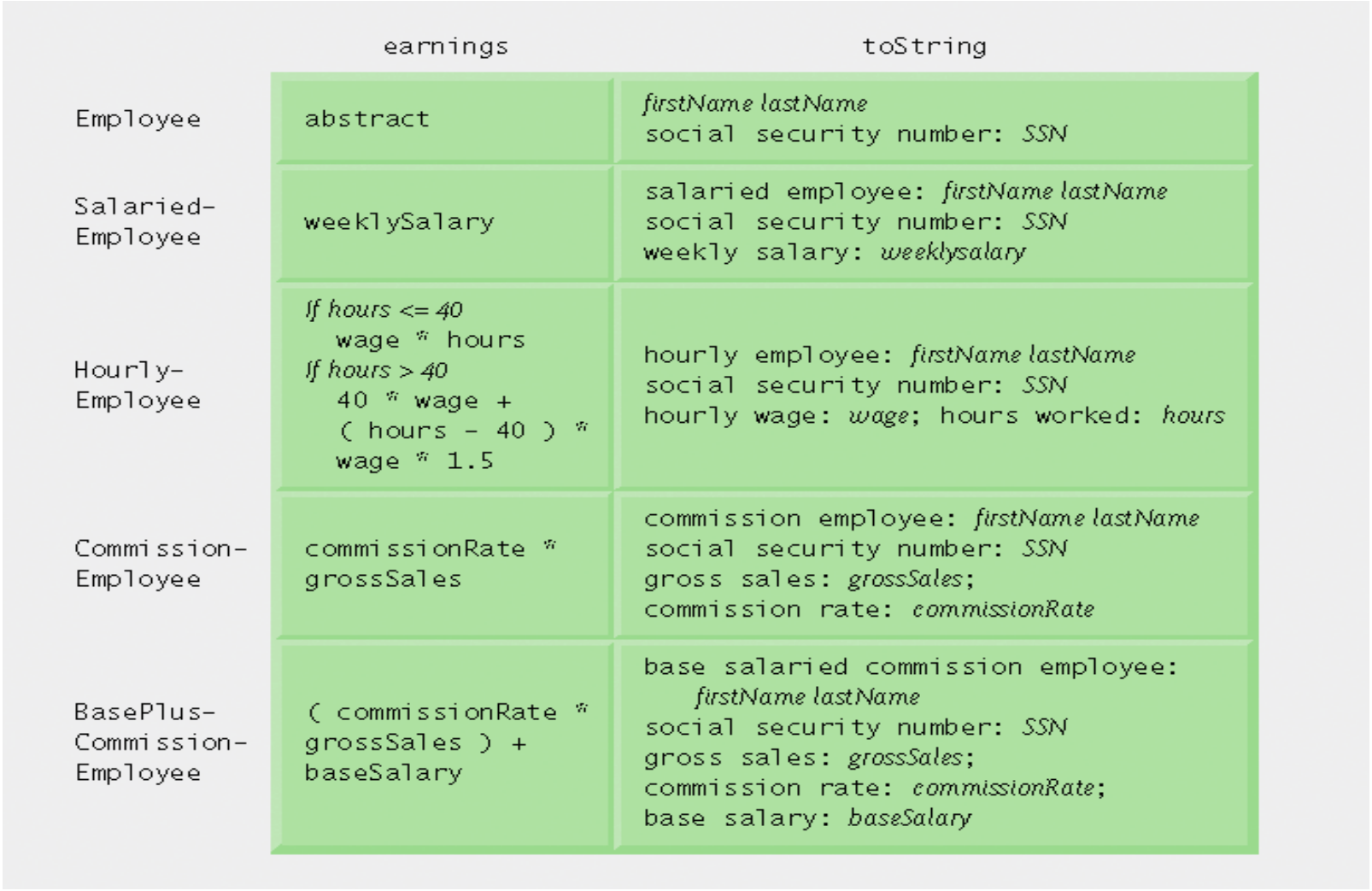
One soln with interfaces



Another soln with interfaces



Polymorphism Example



	earnings	toString
Employee	abstract	*firstName lastName* social security number: *SSN*
Salaried-Employee	weeklySalary	salaried employee: *firstName lastName* social security number: *SSN* weekly salary: *weeklysalary*
Hourly-Employee	*If hours <= 40* wage * hours *If hours > 40* 40 * wage + (hours - 40) * wage * 1.5	hourly employee: *firstName lastName* social security number: *SSN* hourly wage: *wage*; hours worked: *hours*
Commission-Employee	commissionRate * grossSales	commission employee: *firstName lastName* social security number: *SSN* gross sales: *grossSales*; commission rate: *commissionRate*
BasePlus-Commission-Employee	(commissionRate * grossSales) + baseSalary	base salaried commission employee: *firstName lastName* social security number: *SSN* gross sales: *grossSales*; commission rate: *commissionRate*; base salary: *baseSalary*

Fig. 10.3 | Polymorphic interface for the `Employee` hierarchy classes.

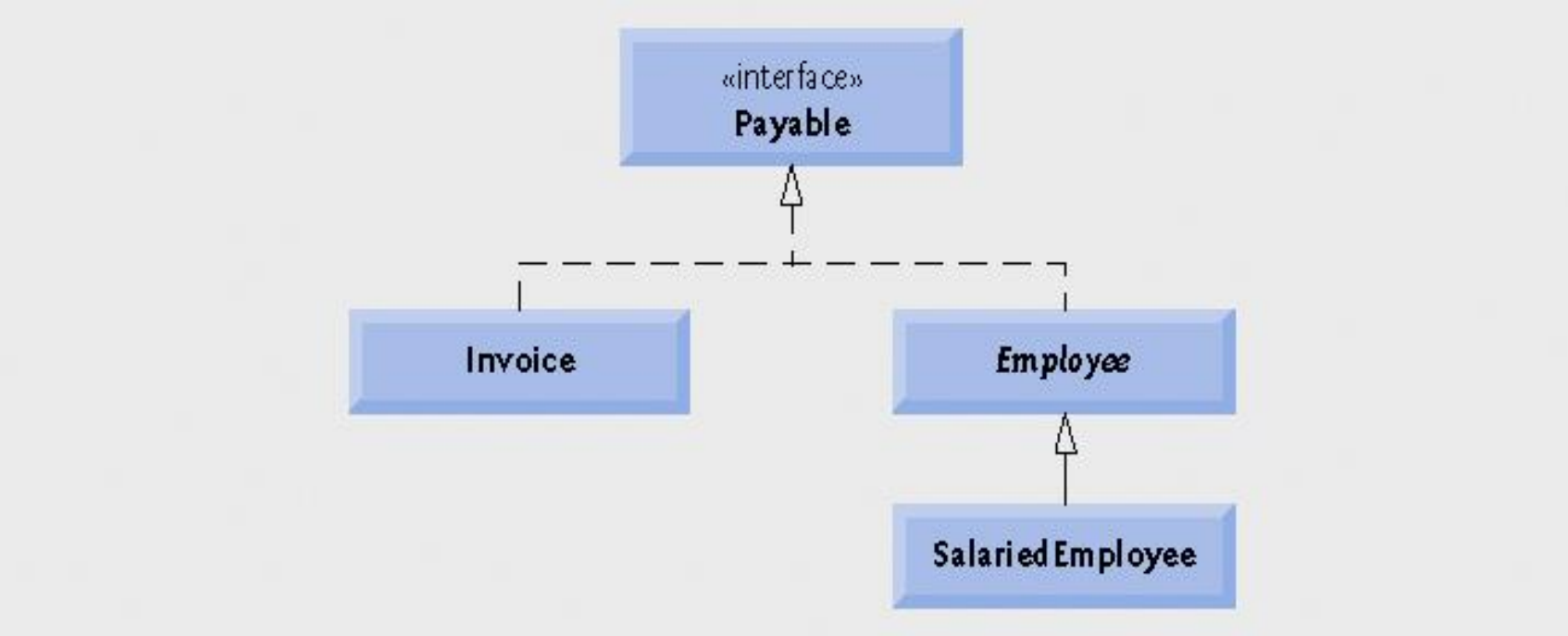


Fig. 10.10 | `Payable` interface hierarchy UML class diagram.

Outline

- Payable.java
-

```
1 // Fig. 10.11: Payable.java
2 // Payable interface declaration.
3
4 public interface Payable
5 {
6    double getPaymentAmount(); // calculate payment; no implementation
7 } // end interface Payable
```

Declare interface **Payable**

Declare **getPaymentAmount** method which is implicitly **public** and **abstract**

Outline

- Invoice.java
- (1 of 3)

```
1  // Fig. 10.12: Invoice.java
2  // Invoice class implements Payable.
3
4  public class Invoice implements Payable
5  {
6     private String partNumber;
7     private String partDescription;
8     private int quantity;
9     private double pricePerItem;
10
11    // four-argument constructor
12    public Invoice( String part, String description, int count,
13       double price )
14    {
15       partNumber = part;
16       partDescription = description;
17       setQuantity( count ); // validate and store quantity
18       setPricePerItem( price ); // validate and store price per item
19    } // end four-argument Invoice constructor
20
21    // set part number
22    public void setPartNumber( String part )
23    {
24       partNumber = part;
25    } // end method setPartNumber
26
```

Class **Invoice** implements interface **Payable**

```
27    // get part number
28    public String getPartNumber()
29    {
30       return partNumber;
31    } // end method getPartNumber
32
33    // set description
34    public void setPartDescription( String description )
35    {
36       partDescription = description;
37    } // end method setPartDescription
38
39    // get description
40    public String getPartDescription()
41    {
42       return partDescription;
43    } // end method getPartDescription
44
45    // set quantity
46    public void setQuantity( int count )
47    {
48       quantity = ( count < 0 ) ? 0 : count; // quantity cannot be negative
49    } // end method setQuantity
50
51    // get quantity
52    public int getQuantity()
53    {
54       return quantity;
55    } // end method getQuantity
56
```

Outline

- Invoice.java
- (2 of 3)

Outline

- Invoice.java
- (3 of 3)

```
57    // set price per item
58    public void setPricePerItem( double price )
59    {
60       pricePerItem = ( price < 0.0 ) ? 0.0 : price; // validate price
61    } // end method setPricePerItem
62
63    // get price per item
64    public double getPricePerItem()
65    {
66       return pricePerItem;
67    } // end method getPricePerItem
68
69    // return String representation of Invoice object
70    public String toString()
71    {
72       return String.format( "%s: \n%s: %s (%s) \n%s: %d \n%s: $%,.2f",
73          "invoice", "part number", getPartNumber(), getPartDescription(),
74          "quantity", getQuantity(), "price per item", getPricePerItem() );
75    } // end method toString
76
77    // method required to carry out contract with interface Payable
78    public double getPaymentAmount()
79    {
80       return getQuantity() * getPricePerItem(); // calculate total cost
81    } // end method getPaymentAmount
82 } // end class Invoice
```

Declare **`getPaymentAmount`** to fulfill contract with interface **`Payable`**

Outline

- Employee.java
- (1 of 3)

```
1  // Fig. 10.13: Employee.java
2  // Employee abstract superclass implements Payable.
3
4  public abstract class Employee implements Payable
5  {
6     private String firstName;
7     private String lastName;
8     private String socialSecurityNumber;
9
10    // three-argument constructor
11    public Employee( String first, String last, String ssn )
12    {
13       firstName = first;
14       lastName = last;
15       socialSecurityNumber = ssn;
16    } // end three-argument Employee constructor
17
```

Class **Employee** implements interface **Payable**

```
18    // set first name
19    public void setFirstName( String first )
20    {
21       firstName = first;
22    } // end method setFirstName
23
24    // return first name
25    public String getFirstName()
26    {
27       return firstName;
28    } // end method getFirstName
29
30    // set last name
31    public void setLastName( String last )
32    {
33       lastName = last;
34    } // end method setLastName
35
36    // return last name
37    public String getLastName()
38    {
39       return lastName;
40    } // end method getLastName
41
```

Outline

- Employee.java
- (2 of 3)

Outline

- Employee.java
- (3 of 3)

```
42    // set social security number
43    public void setSocialSecurityNumber( String ssn )
44    {
45       socialSecurityNumber = ssn; // should validate
46    } // end method setSocialSecurityNumber
47
48    // return social security number
49    public String getSocialSecurityNumber()
50    {
51       return socialSecurityNumber;
52    } // end method getSocialSecurityNumber
53
54    // return String representation of Employee object
55    public String toString()
56    {
57       return String.format( "%s %s\nsocial security number: %s",
58          getFirstName(), getLastName(), getSocialSecurityNumber() );
59    } // end method toString
60
61    // Note: We do not implement Payable method getPaymentAmount here so
62    // this class must be declared abstract to avoid a compilation error.
63 } // end abstract class Employee
```

`getPaymentAmount` method is not implemented here

Outline

- SalariedEmployee
- .java

- (1 of 2)

Class **SalariedEmployee** extends class **Employee** (which implements interface **Payable**)

```
1  // Fig. 10.14: SalariedEmployee.java
2  // SalariedEmployee class extends Employee, which implements Payable.
3
4  public class SalariedEmployee extends Employee
5  {
6     private double weeklySalary;
7
8     // four-argument constructor
9     public SalariedEmployee( String first, String last, String ssn,
10       double salary )
11    {
12       super( first, last, ssn ); // pass to Employee constructor
13       setWeeklySalary( salary ); // validate and store salary
14    } // end four-argument SalariedEmployee constructor
15
16    // set salary
17    public void setWeeklySalary( double salary )
18    {
19       weeklySalary = salary < 0.0 ? 0.0 : salary;
20    } // end method setWeeklySalary
21
```

Outline

- SalariedEmployee
- .java

```
22    // return salary
23    public double getWeeklySalary()
24    {
25       return weeklySalary;
26    } // end method getWeeklySalary
27
28    // calculate earnings; implement interface Payable method that was
29    // abstract in superclass Employee
30    public double getPaymentAmount()
31    {
32       return getWeeklySalary();
33    } // end method getPaymentAmount
34
35    // return String representation of SalariedEmployee object
36    public String toString()
37    {
38       return String.format( "salaried employee: %s\n%s: $%,.2f",
39          super.toString(), "weekly salary", getWeeklySalary() );
40    } // end method toString
41 } // end class SalariedEmployee
```

Declare **`getPaymentAmount`** method instead of **`earnings`** method

Outline

- PayableInterface
- Test.java

(1 of 2)

```
1  // Fig. 10.15: PayableInterfaceTest.java
2  // Tests interface Payable.
3
4  public class PayableInterfaceTest
5  {
6     public static void main( String args[] )
7     {
8        // create four-element Payable array
9        Payable payableObjects[] = new Payable[ 4 ];
10
11       // populate array with objects that implement Payable
12       payableObjects[ 0 ] = new Invoice( "01234", "seat", 2, 375.00 );
13       payableObjects[ 1 ] = new Invoice( "56789", "tire", 4, 79.95 );
14       payableObjects[ 2 ] =
15          new SalariedEmployee( "John", "Smith", "111-11-1111", 800.00 );
16       payableObjects[ 3 ] =
17          new SalariedEmployee( "Lisa", "Barnes", "888-88-8888", 1200.00 );
18
19       System.out.println(
20          "Invoices and Employees processed polymorphically:\n" );
21
```

Declare array of **`Payable`** variables

Assigning references to **`Invoice`** objects to **`Payable`** variables

Assigning references to **`SalariedEmployee`** objects to **`Payable`** variables

Outline

- PayableInterface
- Test.java

```
22       // generically process each element in array payableObjects
23       for ( Payable currentPayable : payableObjects )
24       {
25          // output currentPayable and its appropriate payment amount
26          System.out.printf( "%s \n%s: $%,.2f\n\n",
27             currentPayable.toString(),
28             "payment due", currentPayable.getPaymentAmount() );
29       } // end for
30    } // end main
31 } // end class PayableInterfaceTest
```

Call **`toString`** and **`getPaymentAmount`** methods polymorphically

```
Invoices and Employees processed polymorphically:

invoice:
part number: 01234 (seat)
quantity: 2
price per item: $375.00
payment due: $750.00

invoice:
part number: 56789 (tire)
quantity: 4
price per item: $79.95
payment due: $319.80

salaried employee: John Smith
social security number: 111-11-1111
weekly salary: $800.00
payment due: $800.00

salaried employee: Lisa Barnes
social security number: 888-88-8888
weekly salary: $1,200.00
payment due: $1,200.00
```

10.7.1 Developing a `Payable` Hierarchy

- `Payable` interface
  - Contains method `getPaymentAmount`
  - Is implemented by the `Invoice` and `Employee` classes
- UML representation of interfaces
  - Interfaces are distinguished from classes by placing the word “interface” in guillemets (« and ») above the interface name
  - The relationship between a class and an interface is known as realization
    - A class “realizes” the methods of an interface

Outline

- Employee.java
- (1 of 3)

```
1  // Fig. 10.4: Employee.java
2  // Employee abstract superclass.
3
4  public abstract class Employee
5  {
6     private String firstName;
7     private String lastName;
8     private String socialSecurityNumber;
9
10    // three-argument constructor
11    public Employee( String first, String last, String ssn )
12    {
13       firstName = first;
14       lastName = last;
15       socialSecurityNumber = ssn;
16    } // end three-argument Employee constructor
17
```

Declare **`abstract`** class **`Employee`**

Attributes common to all employees

```
18    // set first name
19    public void setFirstName( String first )
20    {
21       firstName = first;
22    } // end method setFirstName
23
24    // return first name
25    public String getFirstName()
26    {
27       return firstName;
28    } // end method getFirstName
29
30    // set last name
31    public void setLastName( String last )
32    {
33       lastName = last;
34    } // end method setLastName
35
36    // return last name
37    public String getLastName()
38    {
39       return lastName;
40    } // end method getLastName
41
```

Outline

- Employee.java
- (2 of 3)

Outline

- Employee.java
- (3 of 3)

```
42    // set social security number
43    public void setSocialSecurityNumber( String ssn )
44    {
45       socialSecurityNumber = ssn; // should validate
46    } // end method setSocialSecurityNumber
47
48    // return social security number
49    public String getSocialSecurityNumber()
50    {
51       return socialSecurityNumber;
52    } // end method getSocialSecurityNumber
53
54    // return String representation of Employee object
55    public String toString()
56    {
57       return String.format( "%s %s\nsocial security number: %s",
58          getFirstName(), getLastName(), getSocialSecurityNumber() );
59    } // end method toString
60
61    // abstract method overridden by subclasses
62    public abstract double earnings(); // no implementation here
63 } // end abstract class Employee
```

`abstract` method **`earnings`** has no implementation

Outline

- SalariedEmployee
- .java
- (1 of 2)

```
1  // Fig. 10.5: SalariedEmployee.java
2  // SalariedEmployee class extends Employee.
3
4  public class SalariedEmployee extends Employee
5  {
6     private double weeklySalary;
7
8     // four-argument constructor
9     public SalariedEmployee( String first, String last, String ssn,
10       double salary )
11    {
12       super( first, last, ssn ); //
13       setWeeklySalary( salary ); // validate and store salary
14    } // end four-argument SalariedEmployee constructor
15
16    // set salary
17    public void setWeeklySalary( double salary )
18    {
19       weeklySalary = salary < 0.0 ? 0.0 : salary;
20    } // end method setWeeklySalary
21
```

Class **SalariedEmployee** extends class **Employee**

Call superclass constructor

Call **setWeeklySalary** method

Validate and set weekly salary value

Outline

- SalariedEmployee
- .java

(2 of 2)

```
22    // return salary
23    public double getWeeklySalary()
24    {
25       return weeklySalary;
26    } // end method getWeeklySalary
27
28    // calculate earnings; override abstract method earnings in Employee
29    public double earnings()
30    {
31       return getWeeklySalary();
32    } // end method earnings
33
34    // return String representation of S
35    public String toString()
36    {
37       return String.format( "salaried employe
38          super.toString(), "weekly salary", getWeeklySalary() );
39    } // end method toString
40 } // end class SalariedEmployee
```

Override **`earnings`** method so **`SalariedEmployee`** can be concrete

Override **`toString`** method

Call superclass's version of **`toString`**

Outline

- HourlyEmployee
- .java
- (1 of 2)

```
1  // Fig. 10.6: HourlyEmployee.java
2  // HourlyEmployee class extends Employee.
3
4  public class HourlyEmployee extends Employee
5  {
6     private double wage; // wage per hour
7     private double hours; // hours worked for week
8
9     // five-argument constructor
10    public HourlyEmployee( String first, String last, String ssn,
11       double hourlyWage, double hoursWorked )
12    {
13       super( first, last, ssn );
14       setWage( hourlyWage ); // validate hourly wage
15       setHours( hoursWorked ); // validate hours worked
16    } // end five-argument HourlyEmployee constructor
17
18    // set wage
19    public void setWage( double hourlyWage )
20    {
21       wage = ( hourlyWage < 0.0 ) ? 0.0 : hourlyWage;
22    } // end method setWage
23
24    // return wage
25    public double getWage()
26    {
27       return wage;
28    } // end method getWage
29
```

Class **HourlyEmployee** extends class **Employee**

Call superclass constructor

Validate and set hourly wage value

Outline

- HourlyEmployee
- .java

- (2 of 2)

```
30    // set hours worked
31    public void setHours( double hoursWorked )
32    {
33       hours = ( ( hoursWorked >= 0.0 ) && ( hoursWorked <= 168.0 ) ) ?
34          hoursWorked : 0.0;
35    } // end method setHours
36
37    // return hours worked
38    public double getHours()
39    {
40       return hours;
41    } // end method getHours
42
43    // calculate earnings; override abstract method earnings in Employee
44    public double earnings()
45    {
46       if ( getHours() <= 40 ) // no overtime
47          return getWage() * getHours();
48       else
49          return 40 * getWage() + ( gethours() - 40 ) * getWage() * 1.5;
50    } // end method earnings
51
52    // return String representation of HourlyEmployee object
53    public String toString()
54    {
55       return String.format( "hourly employee: %s\n%s: $%,.2f; %s: %,.2f",
56          super.toString(), "hourly wage", getWage(),
57          "hours worked", getHours() );
58    } // end method toString
59 } // end class HourlyEmployee
```

Validate and set hours worked value

Override **`earnings`** method so **`HourlyEmployee`** can be concrete

Override **`toString`** method

Call superclass's **`toString`** method

Outline

```
1  // Fig. 10.7: CommissionEmployee.java
2  // CommissionEmployee class extends Employee.
3
4  public class CommissionEmployee extends Employee
5  {
6     private double grossSales; // gross weekly sales
7     private double commissionRate; // commission percentage
8
9     // five-argument constructor
10    public CommissionEmployee( String first, String last, String ssn,
11       double sales, double rate )
12    {
13       super( first, last, ssn );
14       setGrossSales( sales );
15       setCommissionRate( rate );
16    } // end five-argument CommissionEmployee constructor
17
18    // set commission rate
19    public void setCommissionRate( double rate )
20    {
21       commissionRate = ( rate > 0.0 && rate < 1.0 ) ? rate : 0.0;
22    } // end method setCommissionRate
23
```

Class **CommissionEmployee** extends class **Employee**

CommissionEmployee.java

(1 of 3)

Call superclass constructor

Validate and set commission rate value

```
24    // return commission rate
25    public double getCommissionRate()
26    {
27       return commissionRate;
28    } // end method getCommissionRate
29
30    // set gross sales amount
31    public void setGrossSales( double sales )
32    {
33       grossSales = ( sales < 0.0 ) ? 0.0 : sales;
34    } // end method setGrossSales
35
36    // return gross sales amount
37    public double getGrossSales()
38    {
39       return grossSales;
40    } // end method getGrossSales
41
```

Validate and set the gross sales value

Outline

- CommissionEmployee.java
- (2 of 3)

Outline

- CommissionEmployee.java

```
42    // calculate earnings; override abstract method earnings in Employee
43    public double earnings()
44    {
45       return getCommissionRate() * getGrossSales();
46    } // end method earnings
47
48    // return String representation of CommissionEmployee object
49    public String toString()
50    {
51       return String.format( "%s: %s\n%s: $%,.2f; %s: %.2f",
52          "commission employee", super.toString(),
53          "gross sales", getGrossSales(),
54          "commission rate", getCommissionRate() );
55    } // end method toString
56 } // end class CommissionEmployee
```

Override **`earnings`** method so **`CommissionEmployee`** can be concrete

Override **`toString`** method

Call superclass's **`toString`** method

Outline

- BasePlusCommissionEmployee.java
- (1 of 2)

```
1  // Fig. 10.8: BasePlusCommissionEmployee.java
2  // BasePlusCommissionEmployee class extends CommissionEmployee.
3
4  public class BasePlusCommissionEmployee extends CommissionEmployee
5  {
6     private double baseSalary; // base salary per week
7
8     // six-argument constructor
9     public BasePlusCommissionEmployee( String first, String last,
10       String ssn, double sales, double rate, double salary )
11    {
12       super( first, last, ssn, sales, rate );
13       setBaseSalary( salary ); // validate and store base salary
14    } // end six-argument BasePlusCommissionEmployee constructor
15
16    // set base salary
17    public void setBaseSalary( double salary )
18    {
19       baseSalary = ( salary < 0.0 ) ? 0.0 : salary; // non-negative
20    } // end method setBaseSalary
21
```

Class **BasePlusCommissionEmployee** extends class **CommissionEmployee**

Call superclass constructor

Validate and set base salary value

Outline

- BasePlusCommissionEmployee.java

(2 of 2)

```
22    // return base salary
23    public double getBaseSalary()
24    {
25       return baseSalary;
26    } // end method getBaseSalary
27
28    // calculate earnings; override method earnings in CommissionEmployee
29    public double earnings()
30    {
31       return getBaseSalary() + super.earnings();
32    } // end method earnings
33
34    // return String representation of BasePlusCommissionEmployee object
35    public String toString()
36    {
37       return String.format( "%s %s; %s: $%,.2f",
38          "base-salaried", super.toString(),
39          "base salary", getBaseSalary() );
40    } // end method toString
41 } // end class BasePlusCommissionEmployee
```

Override **`earnings`** method

Call superclass's **`earnings`** method

Override **`toString`** method

Call superclass's **`toString`** method

Outline

- PayrollSystemTest
- .java

- (1 of 5)

```
1  // Fig. 10.9: PayrollSystemTest.java
2  // Employee hierarchy test program.
3
4  public class PayrollSystemTest
5  {
6     public static void main( String args[] )
7     {
8        // create subclass objects
9        SalariedEmployee salariedEmployee =
10          new SalariedEmployee( "John", "Smith", "111-11-1111", 800.00 );
11       HourlyEmployee hourlyEmployee =
12          new HourlyEmployee( "Karen", "Price", "222-22-2222", 16.75, 40 );
13       CommissionEmployee commissionEmployee =
14          new CommissionEmployee(
15          "Sue", "Jones", "333-33-3333", 10000, .06 );
16       BasePlusCommissionEmployee basePlusCommissionEmployee =
17          new BasePlusCommissionEmployee(
18          "Bob", "Lewis", "444-44-4444", 5000, .04, 300 );
19
20       System.out.println( "Employees processed individually:\n" );
21
```

Outline

- PayrollSystemTest
- .java

- (2 of 5)

```
22       System.out.printf( "%s\n%s: $%,.2f\n\n",
23          salariedEmployee, "earned", salariedEmployee.earnings() );
24       System.out.printf( "%s\n%s: $%,.2f\n\n",
25          hourlyEmployee, "earned", hourlyEmployee.earnings() );
26       System.out.printf( "%s\n%s: $%,.2f\n\n",
27          commissionEmployee, "earned", commissionEmployee.earnings() );
28       System.out.printf( "%s\n%s: $%,.2f\n\n",
29          basePlusCommissionEmployee,
30          "earned", basePlusCommissionEmployee.earnings() );
31
32       // create four-element Employee array
33       Employee employees[] = new Employee[ 4 ];
34
35       // initialize array with Employees
36       employees[ 0 ] = salariedEmployee;
37       employees[ 1 ] = hourlyEmployee;
38       employees[ 2 ] = commissionEmployee;
39       employees[ 3 ] = basePlusCommissionEmployee;
40
41       System.out.println( "Employees processed polymorphically:\n" );
42
43       // generically process each element in array employees
44       for ( Employee currentEmployee : employees )
45       {
46          System.out.println( currentEmployee ); // invokes toString
47
```

Assigning subclass objects to supercalss variables

Implicitly and polymorphically call **`toString`**

Outline

```
48          // determine whether element is a BasePlusCommissionEmployee
49          if ( currentEmployee instanceof BasePlusCommissionEmployee )
50          {
51             // downcast Employee reference to
52             // BasePlusCommissionEmployee reference
53             BasePlusCommissionEmployee employee =
54                ( BasePlusCommissionEmployee ) curren
55
56             double oldBaseSalary = employee.getBaseSalary();
57             employee.setBaseSalary( 1.10 * oldBaseSalary );
58             System.out.printf(
59                "new base salary with 10%% increase is: $%,.2f\n",
60                employee.getBaseSalary() );
61          } // end if
62
63          System.out.printf(
64             "earned $%,.2f\n\n", currentEmployee.ear
65       } // end for
66
67       // get type name of each object in employees array
68       for ( int j = 0; j < employees.length; j++ )
69          System.out.printf( "Employee %d is a %s\n", j,
70             employees[ j ].getClass().getName() );
71    } // end main
72 } // end class PayrollSystemTest
```

If the **currentEmployee** variable points to a **BasePlusCommissionEmployee** obje

Downcast **currentEmployee** to a **BasePlusCommissionEmployee** reference

Give **BasePlusCommissionEmployee** s a 10% base salary bonus

Polymorphically call **earnings** method

Call **getClass** and **getName** methods to display each **Employee** subclass object's class name

```
Employees processed individually:

salaried employee: John Smith
social security number: 111-11-1111
weekly salary: $800.00
earned: $800.00

hourly employee: Karen Price
social security number: 222-22-2222
hourly wage: $16.75; hours worked: 40.00
earned: $670.00

commission employee: Sue Jones
social security number: 333-33-3333
gross sales: $10,000.00; commission rate: 0.06
earned: $600.00

base-salaried commission employee: Bob Lewis
social security number: 444-44-4444
gross sales: $5,000.00; commission rate: 0.04; base salary: $300.00
earned: $500.00
```

Outline

- PayrollSystemTest
- .java

- (4 of 5)

Outline

- PayrollSystemTest
- .java

(5 of 5)

```
Employees processed polymorphically:

salaried employee: John Smith
social security number: 111-11-1111
weekly salary: $800.00
earned $800.00

hourly employee: Karen Price
social security number: 222-22-2222
hourly wage: $16.75; hours worked: 40.00
earned $670.00

commission employee: Sue Jones
social security number: 333-33-3333
gross sales: $10,000.00; commission rate: 0.06
earned $600.00

base-salaried commission employee: Bob Lewis
social security number: 444-44-4444
gross sales: $5,000.00; commission rate: 0.04; base salary: $300.00
new base salary with 10% increase is: $330.00
earned $530.00

Employee 0 is a SalariedEmployee
Employee 1 is a HourlyEmployee
Employee 2 is a CommissionEmployee
Employee 3 is a BasePlusCommissionEmployee
```

Same results as when the employees were processed individually

Base salary is increased by 10%

Each employee's type is displayed