

Java 1

Lecture 06-Inheritance and Polymorphism
Sayed Ahmad Sahim

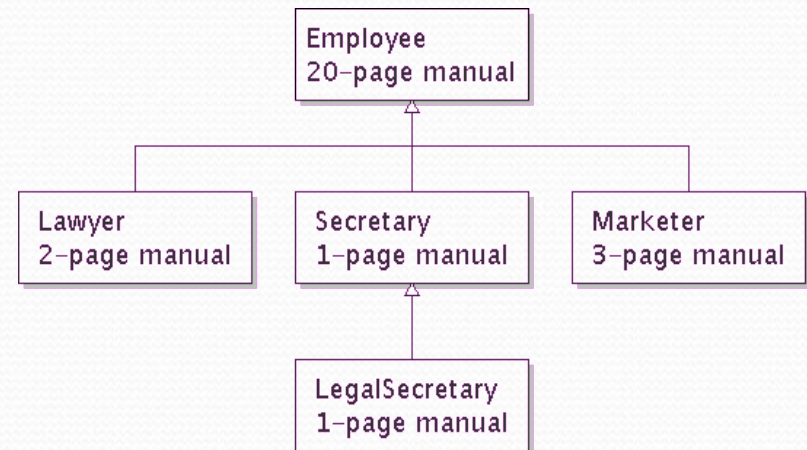
11.October.2017

The software crisis

- **software engineering:** The practice of developing, designing, documenting, testing large computer programs.
- Large-scale projects face many issues:
 - getting many programmers to work together
 - getting code finished on time
 - avoiding redundant code
 - finding and fixing bugs
 - maintaining, improving, and reusing existing code
- **code reuse:** The practice of writing program code once and using it in many contexts.

Law firm employee analogy

- common rules: hours, vacation, benefits, regulations ...
 - all employees attend a common orientation to learn general company rules
 - each employee receives a 20-page manual of common rules
- each subdivision also has specific rules:
 - employee receives a smaller (1-3 page) manual of these rules
 - smaller manual adds some new rules and also changes some rules from the large manual

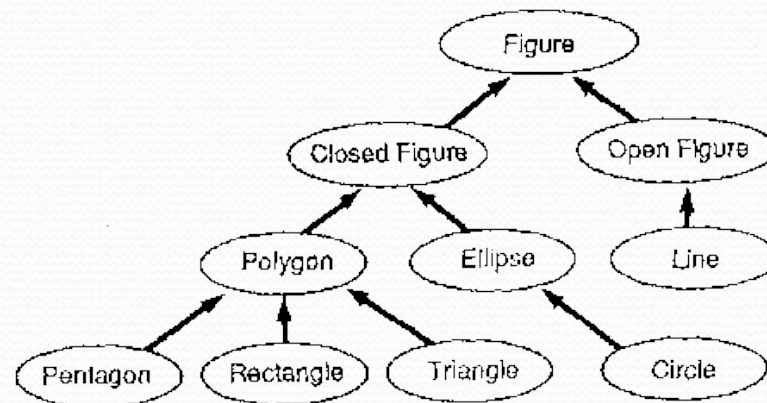


Separating behavior

- Why not just have a 22 page Lawyer manual, a 21-page Secretary manual, a 23-page Marketer manual, etc.?
- Some advantages of the separate manuals:
 - maintenance: Only one update if a common rule changes.
 - locality: Quick discovery of all rules specific to lawyers.
- Some key ideas from this example:
 - General rules are useful (the 20-page manual).
 - Specific rules that may override general ones are also useful.

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.



Employee regulations

- Consider the following employee regulations:
 - Employees work 40 hours / week.
 - Employees make \$40,000 per year, except legal secretaries who make \$5,000 extra per year (\$45,000 total), and marketers who make \$10,000 extra per year (\$50,000 total).
 - Employees have 2 weeks of paid vacation leave per year, except lawyers who get an extra week (a total of 3).
 - Employees should use a yellow form to apply for leave, except for lawyers who use a pink form.
- Each type of employee has some unique behavior:
 - Lawyers know how to sue.
 - Marketers know how to advertise.
 - Secretaries know how to take dictation.
 - Legal secretaries know how to prepare legal documents.

An Employee class

// A class to represent employees in general (20-page manual).

```
public class Employee {  
    public int getHours() {  
        return 40;           // works 40 hours / week  
    }  
  
    public double getSalary() {  
        return 40000.0;      // $40,000.00 / year  
    }  
  
    public int getVacationDays() {  
        return 10;           // 2 weeks' paid vacation  
    }  
  
    public String getVacationForm() {  
        return "yellow";     // use the yellow form  
    }  
}
```

- Exercise: Implement class Secretary, based on the previous employee regulations. (Secretaries can take dictation.)

Redundant Secretary class

// A redundant class to represent secretaries.

```
public class Secretary {  
    public int getHours() {  
        return 40;           // works 40 hours / week  
    }  
  
    public double getSalary() {  
        return 40000.0;      // $40,000.00 / year  
    }  
  
    public int getVacationDays() {  
        return 10;           // 2 weeks' paid vacation  
    }  
  
    public String getVacationForm() {  
        return "yellow";     // use the yellow form  
    }  
  
    public void takeDictation(String text) {  
        System.out.println("Taking dictation of text: " + text);  
    }  
}
```


Desire for code-sharing

- takeDictation is the only unique behavior in Secretary.
- We'd like to be able to say:

// A class to represent secretaries.

```
public class Secretary {  
    copy all the contents from the Employee class;  
  
    public void takeDictation(String text) {  
        System.out.println("Taking dictation of text: " + text);  
    }  
}
```

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

Inheritance syntax

```
public class name extends superclass {
```

- Example:

```
public class Secretary extends Employee {  
    ...  
}
```

- By extending Employee, each Secretary object now:
 - receives a getHours, getSalary, getVacationDays, and getVacationForm method automatically
 - can be treated as an Employee by client code (seen later)

Improved Secretary code

// A class to represent secretaries.

```
public class Secretary extends Employee {  
    public void takeDictation(String text) {  
        System.out.println("Taking dictation of text: " + text);  
    }  
}
```

- Now we only write the parts unique to each type.
 - Secretary inherits getHours, getSalary, getVacationDays, and getVacationForm methods from Employee.
 - Secretary adds the takeDictation method.

Implementing Lawyer

- Consider the following lawyer regulations:
 - Lawyers who get an extra week of paid vacation (a total of 3).
 - Lawyers use a pink form when applying for vacation leave.
 - Lawyers have some unique behavior: they know how to sue.
- Problem: We want lawyers to inherit *most* behavior from employee, but we want to replace parts with new behavior.

Overriding methods

- **override:** To write a new version of a method in a subclass that replaces the superclass's version.
 - No special syntax required to override a superclass method. Just write a new version of it in the subclass.

```
public class Lawyer extends Employee {  
    // overrides getVacationForm method in Employee class  
    public String getVacationForm() {  
        return "pink";  
    }  
    ...  
}
```

- Exercise: Complete the Lawyer class.
 - (3 weeks vacation, pink vacation form, can sue)

Lawyer class

```
// A class to represent lawyers.
public class Lawyer extends Employee {
    // overrides getVacationForm from Employee class
    public String getVacationForm() {
        return "pink";
    }

    // overrides getVacationDays from Employee class
    public int getVacationDays() {
        return 15;           // 3 weeks vacation
    }

    public void sue() {
        System.out.println("I'll see you in court!");
    }
}
```

- Exercise: Complete the Marketer class. Marketers make \$10,000 extra (\$50,000 total) and know how to advertise.

Marketer class

// A class to represent marketers.

```
public class Marketer extends Employee {  
    public void advertise() {  
        System.out.println("Act now while supplies last!");  
    }  
  
    public double getSalary() {  
        return 50000.0;           // $50,000.00 / year  
    }  
}
```


Levels of inheritance

- Multiple levels of inheritance in a hierarchy are allowed.
 - Example: A legal secretary is the same as a regular secretary but makes more money (\$45,000) and can file legal briefs.

```
public class LegalSecretary extends Secretary {  
    ...  
}
```

- Exercise: Complete the LegalSecretary class.

LegalSecretary class

// A class to represent legal secretaries.

```
public class LegalSecretary extends Secretary {  
    public void fileLegalBriefs() {  
        System.out.println("I could file all day!");  
    }  
  
    public double getSalary() {  
        return 45000.0;           // $45,000.00 / year  
    }  
}
```


Polymorphism

Polymorphism

- **polymorphism:** Ability for the same code to be used with different types of objects and behave differently with each.
 - `System.out.println` can print any type of object.
 - Each one displays in its own way on the console.
 - `CritterMain` can interact with any type of critter.
 - Each one moves, etc. in its own way.

Coding with polymorphism

- A variable of type T can hold an object of any subclass of T .

```
Employee ed = new Lawyer();
```

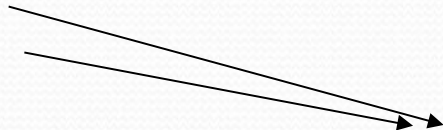
- You can call any methods from Employee on ed.
 - You can *not* call any methods specific to Lawyer (e.g. sue).
- When a method is called on ed, it behaves as a Lawyer.

```
System.out.println(ed.getSalary());           // 50000.0  
System.out.println(ed.getVacationForm());     // pink
```

Polymorphism and parameters

- You can pass any subtype of a parameter's type.

```
public class EmployeeMain {  
    public static void main(String[] args) {  
        Lawyer lisa = new Lawyer();  
        Secretary steve = new Secretary();  
        printInfo(lisa);  
        printInfo(steve);  
    }  
  
    public static void printInfo(Employee empl) {  
        System.out.println("salary = " + empl.getSalary());  
        System.out.println("days = " + empl.getVacationDays());  
        System.out.println("form = " + empl.getVacationForm());  
        System.out.println();  
    }  
}
```

A diagram consisting of two arrows. The first arrow originates from the `printInfo(lisa);` line in the `main` method and points to the `printInfo` method signature. The second arrow originates from the `printInfo(steve);` line in the `main` method and also points to the `printInfo` method signature. This illustrates that both `Lawyer` and `Secretary` objects are passed to the same `printInfo` method, which is defined to accept an `Employee` parameter.

OUTPUT:

```
salary = 50000.0  
vacation days = 21  
vacation form = pink
```

```
salary = 50000.0  
vacation days = 10  
vacation form = yellow
```


Polymorphism and arrays

- Arrays of superclass types can store any subtype as elements.

```
public class EmployeeMain2 {  
    public static void main(String[] args) {  
        Employee[] e = { new Lawyer(), new Secretary(),  
                          new Marketer(), new LegalSecretary() };  
        for (int i = 0; i < e.length; i++) {  
            System.out.println("salary: " + e[i].getSalary());  
            System.out.println("v.days: " + e[i].getVacationDays());  
            System.out.println();  
        }  
    }  
}
```

Output:

```
salary: 50000.0  
v.days: 15  
salary: 50000.0  
v.days: 10  
salary: 60000.0  
v.days: 10  
salary: 55000.0  
v.days: 10
```

Polymorphism problems

- 4-5 classes with inheritance relationships are shown.
- A client program calls methods on objects of each class.
- You must read the code and determine the client's output.
- This could be a question for the final exam!

A polymorphism problem

- Assume that the following four classes have been declared:

```
public class Foo {  
    public void method1() {  
        System.out.println("foo 1");  
    }  
    public void method2() {  
        System.out.println("foo 2");  
    }  
    public String toString() {  
        return "foo";  
    }  
}  
  
public class Bar extends Foo {  
    public void method2() {  
        System.out.println("bar 2");  
    }  
}
```

A polymorphism problem

```
public class Baz extends Foo {
    public void method1() {
        System.out.println("baz 1");
    }
    public String toString() {
        return "baz";
    }
}

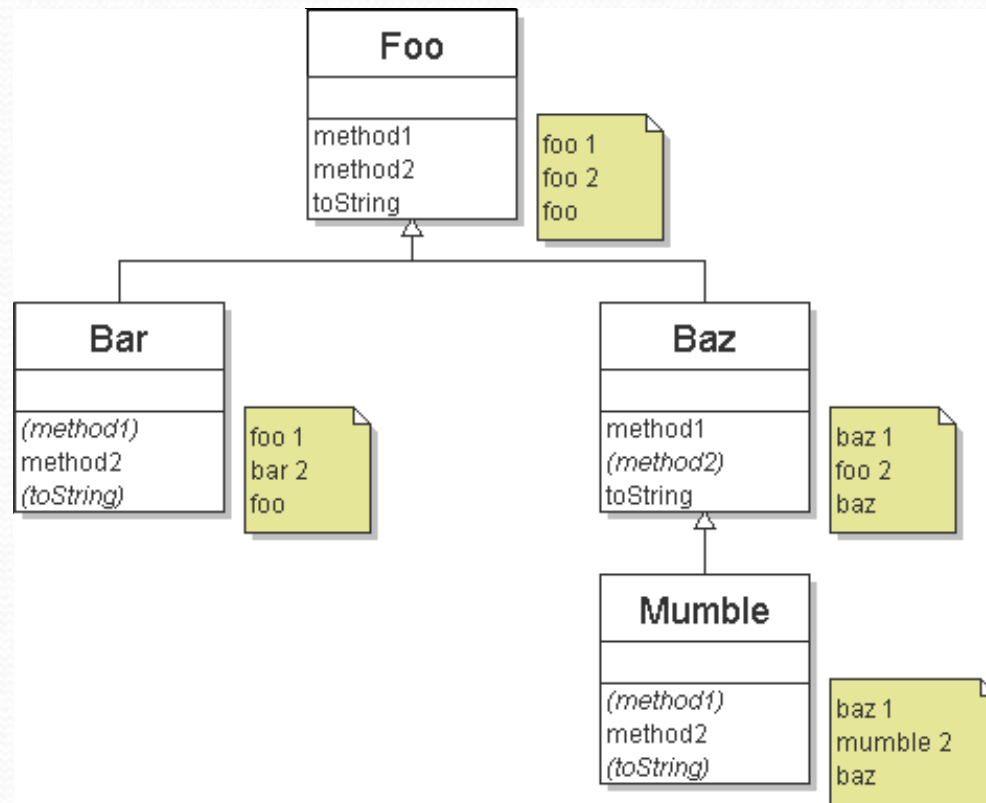
public class Mumble extends Baz {
    public void method2() {
        System.out.println("mumble 2");
    }
}
```

- What would be the output of the following client code?

```
Foo[] elements = {new Foo(), new Bar(), new Baz(), new Mumble()};
for (int i = 0; i < elements.length; i++) {
    System.out.println(elements[i]);
    elements[i].method1();
    elements[i].method2();
    System.out.println();
}
```


Diagramming the classes

- Add classes from top (superclass) to bottom (subclass).
- Include all inherited methods.



Finding output with tables

method	Foo	Bar	Baz	Mumble
method1	foo 1	<i>foo 1</i>	baz 1	<i>baz 1</i>
method2	foo 2	bar 2	<i>foo 2</i>	mumble 2
toString	foo	<i>foo</i>	baz	<i>baz</i>

Polymorphism answer

```
Foo[] elements={new Foo(), new Bar(), new Baz(), new Mumble()};  
for (int i = 0; i < elements.length; i++) {  
    System.out.println(elements[i]);  
    elements[i].method1();  
    elements[i].method2();  
    System.out.println();  
}
```

- Output:

```
foo  
foo 1  
foo 2  
  
foo  
foo 1  
bar 2  
  
baz  
baz 1  
foo 2  
  
baz  
baz 1  
mumble 2
```