

hi


Building Java Programs

Chapter 9
Lecture 9-1: Inheritance

Copyright 2010 by Pearson Education

The software crisis


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



Copyright 2010 by Pearson Education

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



```

graph TD
    A[Employee 20-page manual] --> B[Lawyer 2-page manual]
    A --> C[Secretary 1-page manual]
    A --> D[Marketer 3-page manual]
    C --> E[LegalSecretary 2-page manual]
  
```

Copyright 2010 by Pearson Education

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.

Copyright 2010 by Pearson Education

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.

```

graph TD
    Figure --> ClosedFigure
    Figure --> OpenFigure
    ClosedFigure --> Polygon
    ClosedFigure --> Ellipse
    ClosedFigure --> Line
    Polygon --> Paragon
    Polygon --> Rectangle
    Polygon --> Triangle
    Line --> Circle
  
```

Copyright 2010 by Pearson Education

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.

Copyright 2010 by Pearson Education

bye

1

hi

An Employee class

```
// A class to represent employees in general (20-page manual).
public class Employee {
    public int getHours() { // works 40 hours / week
        return 40;
    }

    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.)

Copyright 2010 by Pearson Education 7

Redundant Secretary class

```
// A redundant class to represent secretaries.
public class Secretary {
    public int getHours() { // works 40 hours / week
        return 40;
    }

    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);
    }
}
```

Copyright 2010 by Pearson Education 8

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);
    }
}
```

Copyright 2010 by Pearson Education 9

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

Copyright 2010 by Pearson Education 10

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)

Copyright 2010 by Pearson Education 11

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.

Copyright 2010 by Pearson Education 12


bye

2

hi

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.



13

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)

14

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.

15

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
    }
}

```

16

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.

17


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
    }
}

```



18

bye

hi

Interacting with the Superclass (super)

reading: 9.2

19

Copyright 2010 by Pearson Education

Changes to common behavior

- Let's return to our previous company/employee example.
- Imagine a company-wide change affecting all employees.

Example: Everyone is given a \$10,000 raise due to inflation.

- The base employee salary is now \$50,000.
 - Legal secretaries now make \$55,000.
 - Marketers now make \$60,000.
- We must modify our code to reflect this policy change.

20

Copyright 2010 by Pearson Education

Modifying the superclass

```
// 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 50000.0; // $50,000.00 / year
    }
    ...
}
```

- Are we finished?
- The `Employee` subclasses are still incorrect.
 - They have overridden `getSalary` to return other values.

21

Copyright 2010 by Pearson Education

An unsatisfactory solution

```
public class LegalSecretary extends Secretary {
    public double getSalary() {
        return 55000.0;
    }
    ...
}
public class Marketer extends Employee {
    public double getSalary() {
        return 60000.0;
    }
    ...
}
```

- Problem: The subclasses' salaries are based on the `Employee` salary, but the `getSalary` code does not reflect this.

22

Copyright 2010 by Pearson Education

Calling overridden methods

- Subclasses can call overridden methods with `super`
`super.method(parameters)`
- Example:

```
public class LegalSecretary extends Secretary {
    public double getSalary() {
        double baseSalary = super.getSalary();
        return baseSalary + 5000.0;
    }
    ...
}
```
- Exercise: Modify `Lawyer` and `Marketer` to use `super`.

23

Copyright 2010 by Pearson Education

Improved subclasses

```
public class Lawyer extends Employee {
    public String getVacationForm() {
        return "pink";
    }
    public int getVacationDays() {
        return super.getVacationDays() + 5;
    }
    public void sue() {
        System.out.println("I'll see you in court!");
    }
}
public class Marketer extends Employee {
    public void advertise() {
        System.out.println("Act now while supplies last!");
    }
    public double getSalary() {
        return super.getSalary() + 10000.0;
    }
}
```

24

Copyright 2010 by Pearson Education

bye

4