Lecture 14: Inheritance

Building Java Programs: A Back to Basics Approach
by Stuart Reges and Marty Stepp

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

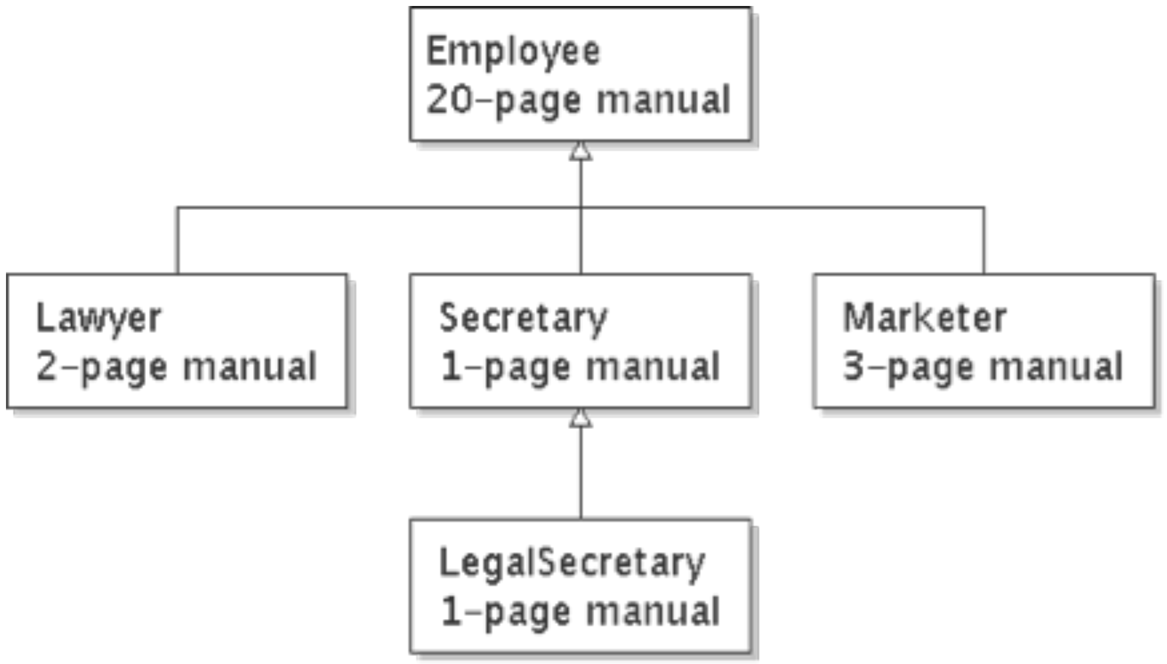
- **code reuse**: The practice of writing program code once and using it in many contexts.

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

Law firm employee analogy

- 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.?

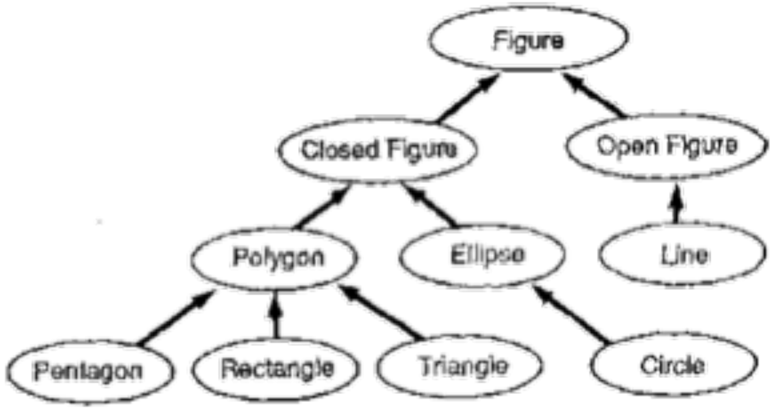
- Some advantages of the separate manuals:
  - maintenance: Only one update if a common rule changes.
  - locality: Quick discovery of all rules specific to lawyers.

- 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
    }
}
```

– Implement class `Secretary`, based on the previous employee regulations. (Secretaries can take dictation.)

Secretary class

Notice the redundancy.

```
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";
    }
    ...
}
```

Have we done this before? Answer:toString()

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!");
    }
}
```

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 {
    ...
}
```

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
    }
}
```

Example

```
public static void main(String[] args) {
  Employee a = new Employee();
  Employee b = new Marketer();//a marketer is-an employee
  Employee c = new LegalSecretary(); //a legal secretary is an
                                   //employee
  Lawyer d = new Lawyer();
  Lawyer e = new Employee(); // compile error
                     //not every employee is a lawyer.
  LegalSecretary f = new Secretary(); //compile error

  double  salary = a.getSalary(); //40000
  double salary2 = b.getSalary();  //50000, use overwritten
                                       //version
  double salary3 = c.getSalary();  //45000, use overwritten
                                       //version
  a.sue(); //error, no sue method for Employee a

  b.advertise();// error, even though b is a Marketer.
       //b is an Employee reference.
       //(need to cast, later lecture)

  d.sue();
```

Lab 1

Write the superclass Student and subclass GradStudent.

•The Student class has a private string `name` and public integer id. It has **no constructors.**

•Student has `getName(), setName (String n),printWelcome()` PUBLIC methods. `printWelcome()` prints `“Welcome”.`

Lab 1

Write the subclass GradStudent.

•The GradStudent class has a private string `dissertationTopic`. It has **no constructors.**

•GradStudent has `getTopic(), setTopic (String t),printWelcome()` public methods. `printWelcome()` overrides the same method from the superclass Student and prints `“Welcome to Graduate School”.`

Lab 1(continued)

Write the driver class to create a Student and a GradStudent and prints out their names and welcome messages. Print out private variables to see the error messages.