# Building Java Programs

Inheritance

**reading: 9.1 – 9.2**

Copyright 2010 by Pearson Education

# 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.)

Copyright 2010 by Pearson Education

# 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);
    }
}
```

Copyright 2010 by Pearson Education

# 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

# 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

# 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

# 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

# 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.

Copyright 2010 by Pearson Education

# 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)

Copyright 2010 by Pearson Education

# 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.

Copyright 2010 by Pearson Education

# 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
    }
}
```

Copyright 2010 by Pearson Education

# 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.

Copyright 2010 by Pearson Education

# 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
    }
}
```

Copyright 2010 by Pearson Education

# Interacting with the Superclass (`super`)

**reading: 9.2**

Copyright 2010 by Pearson Education

# Changes to common behavior

- 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.

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.

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.

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`.

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;
    }
}
```

Copyright 2010 by Pearson Education

# Inheritance and constructors

- Imagine that we want to give employees more vacation days the longer they've been with the company.
  - For each year worked, we'll award 2 additional vacation days.
  - When an Employee object is constructed, we'll pass in the number of years the person has been with the company.
  - This will require us to modify our `Employee` class and add some new state and behavior.
  - Exercise: Make necessary modifications to the `Employee` class.

Copyright 2010 by Pearson Education

# Modified `Employee` class

```
public class Employee {
    private int years;

    public Employee(int initialYears) {
        years = initialYears;
    }

    public int getHours() {
        return 40;
    }

    public double getSalary() {
        return 50000.0;
    }

    public int getVacationDays() {
        return 10 + 2 * years;
    }

    public String getVacationForm() {
        return "yellow";
    }
}
```

Copyright 2010 by Pearson Education

# Problem with constructors

- Now that we've added the constructor to the `Employee` class, our subclasses do not compile. The error:

```
Lawyer.java:2: cannot find symbol
symbol  : constructor Employee()
location: class Employee
public class Lawyer extends Employee {
       ^
```

  - The short explanation: Once we write a constructor (that requires parameters) in the superclass, we must now write constructors for our employee subclasses as well.

  - The long explanation: (next slide)

Copyright 2010 by Pearson Education

# The detailed explanation

- Constructors are not inherited.
  - Subclasses don't inherit the `Employee(int)` constructor.
  - Subclasses receive a default constructor that contains:

```
public Lawyer() {
    super();          // calls Employee() constructor
}
```

- But our `Employee(int)` replaces the default `Employee()`.
  - The subclasses' default constructors are now trying to call a non-existent default `Employee` constructor.

Copyright 2010 by Pearson Education

# Calling superclass constructor

```
super(parameters);
```

- Example:

```
public class Lawyer extends Employee {
    public Lawyer(int years) {
        super(years);  // calls Employee constructor
    }
    ...
}
```

- The `super` call must be the first statement in the constructor.

- Exercise: Make a similar modification to the `Marketer` class.

Copyright 2010 by Pearson Education

# Inheritance and fields

- Try to give lawyers $5000 for each year at the company:

```
public class Lawyer extends Employee {
    ...
    public double getSalary() {
        return super.getSalary() + 5000 * years;
    }
    ...
}
```

- Does not work; the error is the following:

```
Lawyer.java:7: years has private access in Employee
        return super.getSalary() + 5000 * years;
                                          ^
```

- Private fields cannot be directly accessed from subclasses.
  - One reason: So that subclassing can't break encapsulation.
  - How can we get around this limitation?

Copyright 2010 by Pearson Education

# Improved `Employee` code

Add an accessor for any field needed by the subclass.

```
public class Employee {
    private int years;

    public Employee(int initialYears) {
        years = initialYears;
    }

    public int getYears() {
        return years;
    }
    ...
}
public class Lawyer extends Employee {
    public Lawyer(int years) {
        super(years);
    }

    public double getSalary() {
        return super.getSalary() + 5000 * getYears();
    }
    ...
}
```

Copyright 2010 by Pearson Education