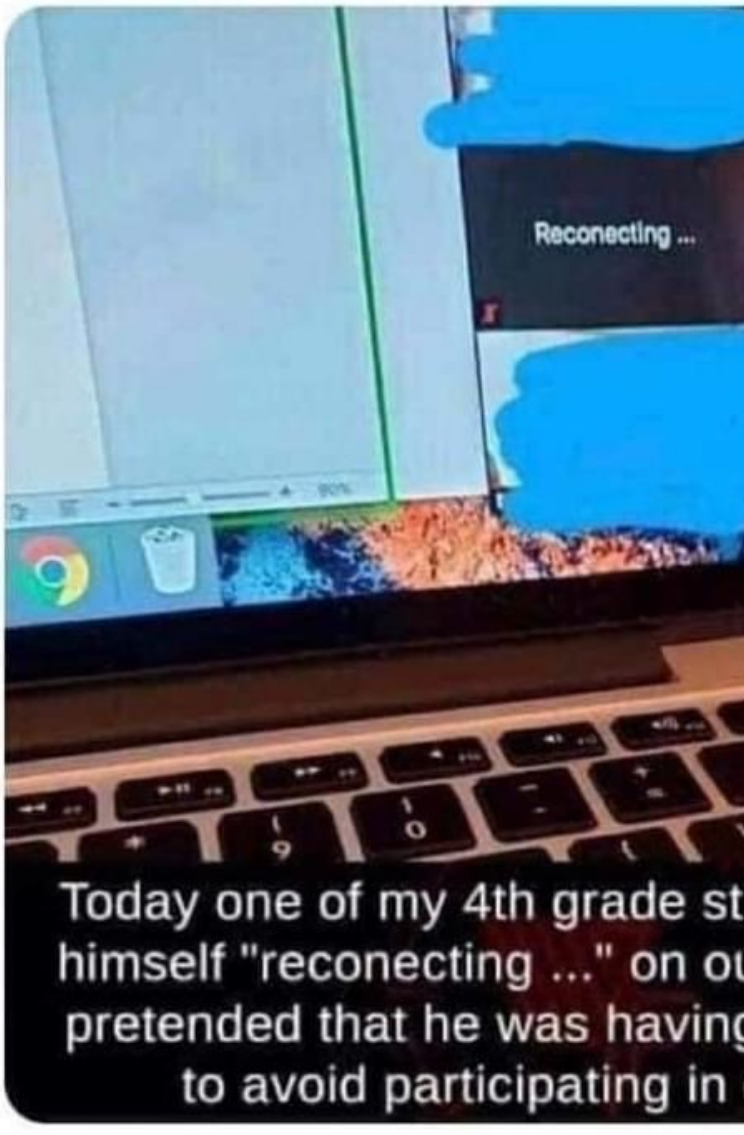
Advanced Programming Techniques in Java

The future of IT is in g



Object Oriented Programming

Lecture 11

Class Objectives

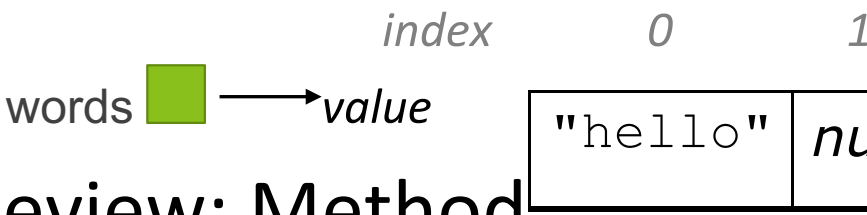
- OOD
- Inheritance Basics (9.1)

Review: Looking before you leap

- You can check for `null` before calling an object'

```
String[] words = new String[5];
words[0] = "hello";
words[2] = "goodbye";    // words[1], [3], [
```

```
for (int i = 0; i < words.length; i++) {
    if (words[i] != null) { words[i] =
    words[i].toUpperCase();
}
}
```



Review: Method Overloading

- There are three ways to overload a method
- Number of parameters `add (int, int)` `add (int, int, int)`

- Data type of parameters `add (int, int) add (int, double)`
- Sequence of data type of parameters `add(int, double) add(double, int)`

Review: Encapsulation

- Encapsulation is a principle of wrapping data (va
unit
- It is one of the four OOP concepts

- Encapsulation
- Inheritance
- Polymorphism
- Abstraction

Review: Encapsulation example 1

```
public class Account { private
   int account_number; private
   int account_balance;

   public void showData() { //code
      to show data
   }

   public void deposit(int a) {
       if (a < 0){
          //show error
       } else { account_balance =
       account_balance + a; }
}
```

- Approach 1 and Approach 2 fail
- You never expose your data to an external party (wh
  - The entire code can be thought as capsule

Review: Point class

```
public class Point{
     private int x;
     private int y;

     public Point(){
          this(0, 0);
     }

     public Point(int x, int y){
          setLocation(x, y);
     }

     public double distanceFromOrigin(){
          return Math.sqrt(x * x + y * y);
     }

     public int getX(){
          return x;
```

```
… publ
      r
      }

      p

      }

      p

      }

      p
     dy

…
}
```

```
        }
...
```

Benefits of encapsulation

- Abstraction between object and clients
- Protects object from unwanted access
- Example: Can't fraudulently change the points coor
- `getX`(), `getY`() return just a copy of the coordinates

Benefits of encapsulation (cont.)

- Can change the class implementation later
- Example: `Point` could be rewritten in polar coordin
  the same methods
- Client calls to `getX`() and `getY`() do not need to cha
- We just change their internal implementation
- Example: Only allow `Points` with non-negativ
- Ca

Class invariants

- **Class invariant** is an assertion about an object's st
of the object
- An invariant can be thought of as a postcondition on ever
class
- e.g.: "No BankAccount object's balance can be negative”

- **<u>Example</u>**: Suppose we want to ensure that all `Poin`
never negative
- We must ensure that a client cannot construct a `Point c`

- We must ensure that a client cannot move an existing `Po`

Pre/postconditions

- **Precondition**: Something that you assume to be tru
- **Postcondition**: Something you promise to be true w
- Pre/postconditions are often documented as comments

```
// Sets this Point's location to b
// Precondition: newX >= 0 && newY
Postcondition: x >= 0 && y >= 0 pu
setLocation(int newX, int newY) {
y = newY;
}
```

Violated preconditions

- What if your precondition is not met?
- Sometimes the client passes an invalid value to yo

```
Point pt = new Point(5, 17);
```

```
Scanner console = new Scanner(System.in);
coordinates: "); int x = console.nextInt()
a negative number? int y = console.nextInt
```

- How can we prevent the client from misusing our o

Dealing with violations

- One way to deal with this problem would be to retu
  are encountered
- However, it is not possible to do something similar in th
- A more common solution is to have your object ***th***

- **Exception** is a Java object that represents an error
- When a precondition of your method has been violated in your code
- This will cause the client program to halt

Throwing exceptions example

- Throwing an exception, general syntax: **`throw new type> ();`** or **`throw new <except. ("<message>");`**
- The ***<message>*** will be shown on the console when the p

```
// Sets this Point's location to be the g
// Throws an exception if newX or newY is
// Postcondition: x >= 0 && y >= 0
public void setLocation(int x, int y) thr
IllegalArgumentException{ if (x < 0 || y
    { throw new IllegalArgumentException(
    } this.x =
    x; this.y =
    y;
}
```

`Point` class and invariants

- Ensure that no `Point` is constructed with negativ

```
public Point(int x, int y) throws
IllegalArgumentException{ if (x < 0 || y < 0)
     throw new IllegalArgumentException();
     } this.x =
     x; this.y =
     y;
}
```

- Ensure that no `Point` can be moved to a negativ

```
public void translate(int dx, int dy) throws
IllegalArgumentException { if (x + dx < 0 || y
     dy < 0) { throw new
     IllegalArgumentException();
     } x +=
     dx; y +=
     dy;
}
```

Eliminating Redundancy

```
public class Point {
   private int x;
   private int y;
   public Point(int initialX, int init
        setLocation(initialX, initialY
    }
   ....
   public void translate (int dx, int
        setLocation(x + dx, y + dy);
```

```
    }
        public void setLocation
    throws x, int y){ if (x < 0
    || y < 0) { throw new IllegalArgume
        } this.x =
        x; this.y
        = y;
    }
}
```

Final `Point` class

Final `Point`

```
public class Point {
   private int x;
   private int y;

   public Point() { this(0, 0);    // cal
         constructor
   }

   public Point(int x, int y) {
         setLocation(x, y);
   }

   public double distanceFromOrigin() {
           return Math.sqrt(x * x + y * y
   }
   public void translate(int dx, int dy)
         setLocation(x + dx, y + dy);
   }
 …    }
```

Final `Point` class (cont.)

Final `Point`

```
public class Point {
   …
   public boolean equals(Object o) {
         if (o instanceof Point) {
         Point other = (Point) o;
             return x == other.x && y == o
         } else {  // not a Point object
             return false;
         }
   } public int getX()
    { return x;
   }

   public int getY() {
         return y;
   }
   ...
}
```

Final `Point` class (cont.)

Final `Point`

```
public class Point {
   …

   public void setLocation(int x, int y) th
   IllegalArgumentException { if (x < 0 ||
         { throw new IllegalArgumentExceptio
         } this.x =
         x; this.y =
         y;
   }

   public String toString() { return "("
   + x + ", " + y + ")"; }


}
```

Inheritan

Inheritance

The importance of code reuse

- **Software engineering** is the practice of designin testing and maintaining large computer programs many issues:
- Getting many programmers to work together
- Avoiding redundant code
- Finding and fixing bugs
- Maintaining, improving, and reusing existing code

- **Code reuse** is the practice of writing program coo contexts ▪ Inheritance is an important concept o **reusability** ▪ It allows a software developer to d one
- One class acquires the properties of another class
- Like a child inherits the traits of the parents

Inheritance

- The existing class is called the ***parent class***, or ***s***
- The derived class is called the ***child class*** or ***sub***
- The child class inherits the methods and data defined fo

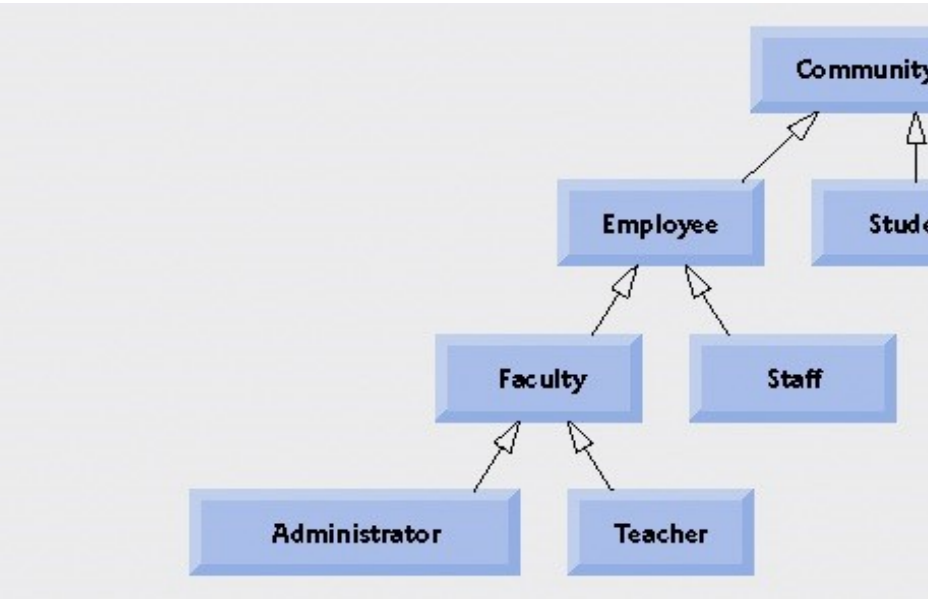
Inheritance

- *Software reuse* is at the heart of inheritance
- We are using existing software components to cre
- We capitalize on all the effort that went into the design, software
- The programmer can add new variables or metho inherited ones

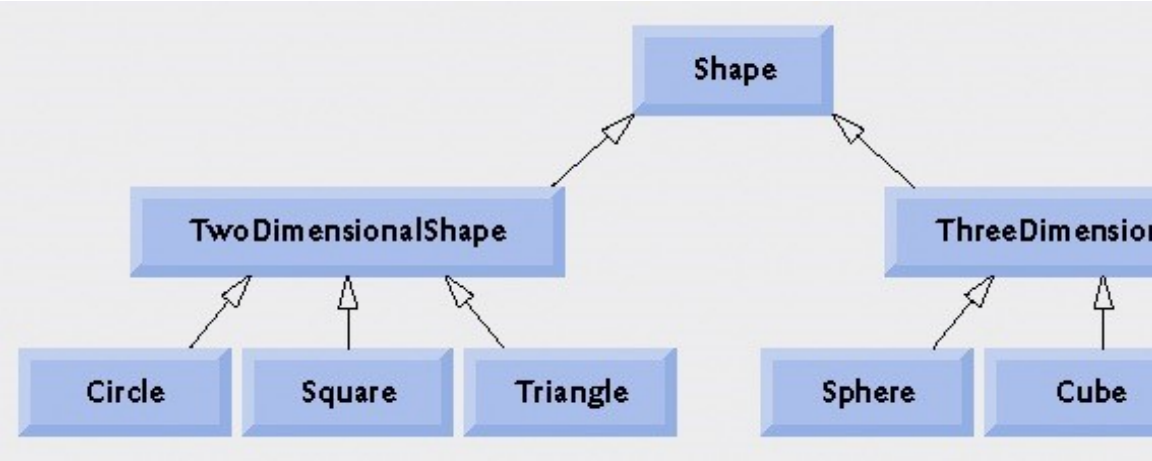
Inheritance

- Inheritance relationships often are shown graphica
  open arrowhead pointing to the parent class

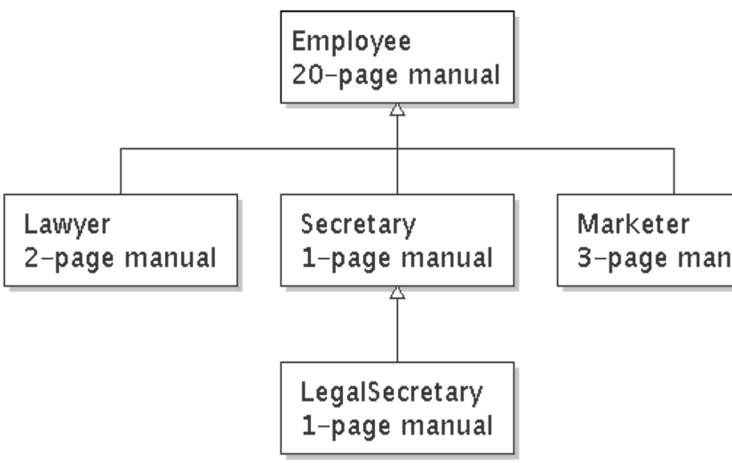
Inheritance example



Inheritance example



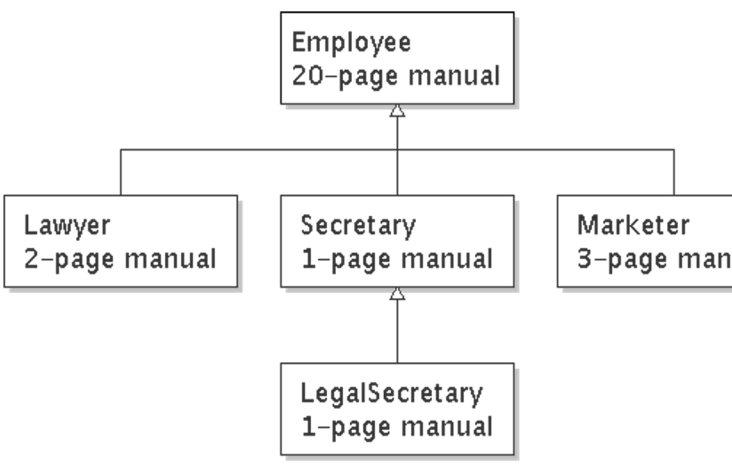
Law firm employee hierarchy



- Common rules: hours, vacation, benefits, regulati
- All employees attend a common orientation to learn ge
- Each subdivision also has specific rules

- We can have a 22-page Lawyer manual, a 21-pag
Marketer manual, etc.?

Law firm employee hierarchy



- Common rules: hours, vacation, benefits, reg
- All employees attend a common orientation to lear

- Each employee receives a 20-page manual of com
  has specific rules:
- Employee receives a smaller (1-3 page) manual o
- Smaller manual adds some new rules and also
  manual

Separating behavior

- Why not just have a 22-page Lawyer manual, a
  page Marketer manual, etc.?
- Some advantages of the separate manuals:
- **Maintenance**: Only one update if a common rule cha
- **Locality**: Quick discovery of all rules specific to lawye

- Some key ideas from this example:
- General rules are useful (the 20-page manual)
- Specific rules that may override general ones are also

Is-a relationships

- **Is-a relationship** is a hierarchical connection wh
specialized version of another
- Every marketer **is-an** employee
- Every legal secretary **is-a** secretary

- **Inheritance hierarchy** is a set of classes connec
share common code

Employee regulations

- Employee regulations:
- **Employee** works 40 hours / week
- **Employee** makes $40,000 per year, except **Legal Secretary** who makes $5,000 extra per year ($45,000 total), and **Marketer** who makes $10,000 extra pe
- **Employee** have 2 weeks of paid vacation leave per year, total of 3)
- **Employee** should use a yellow form to apply for leave, ex

- Each type of employee has some unique beha
- **Lawyer** knows how to sue
- **Marketer** knows how to advertise
- **Secretary** knows how to take dictation

- **Legal Secretary** knows how to prepare legal documents

`Employee` class

```
// A class to represent employees

public class Employee { public int
    return 40;           // works
    }

    public double getSalary() { re
        40000.0;      // $40,000.0
    }

    public int getVacationDays() {
        // 2 weeks' paid vacation
    }

    public String getVacationForm(
    "yellow";     // use the yello
}
```

`Secretary` class `Secretary.ja`

```
// A redundant class to represent secre
public class Secretary { public int get
return 40;            // works 40 hours/
    }

    public double getSalary() { return
        // $40,000.00/year
    }

    public int getVacationDays() { retu
        // 2 weeks' paid vacation
    }

    public String getVacationForm() { r
        "yellow";     // use the yellow
    }

    public void takeDictation(String te
        System.out.println("Taking dict
    }
```

```
}
```

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 tex
        System.out.println("Taking dict
    }
}
```

Inheritance

- **Syntax**

```
public class <subclass name> extends <superc
```

- **Example**

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

  - By extending `Employee`, each `Secretary` ol
  - Receives a `getHours`, `getSalary`, `getVacatior` method **automatically**

- Can be treated as an `Employee` by client code (se

Improved `Secretary` class

```
// A class to represent secretaries
public class Secretary extends Employee
{

    public void takeDictation(String text)
        System.out.println("Taking dictati
    }
}
```

- We only write the parts unique to each type of e
- `Secretary` inherits `getHours`, `getSalary`, `getV` `getVacationForm` methods from Employee
- Secretary adds the `takeDictation` method

Client program example

```
public class EmployeeMain { public static void

    methods of the System.out.println("Employee:"); I

        Employee employee1 = new Employee();
        System.out.print(employee1.getHours() + ",
        System.out.printf("$%.2f, ", employee1.get
        System.out.print(employee1.getVacationDays
        System.out.println(employee1.getVacationFo

        System.out.print("Secretary: ");

        Secretary employee2 = new Secretary();
        System.out.print(employee2.getHours() + ",
        System.out.printf("$%.2f, ", employee2.get
        System.out.print(employee2.getVacationDays
        System.out.println(employee2.getVacationFo
        employee2.takeDictation("CS12b example");
```

```
    }
}
```

The only method declared s

```
Employee: 40, $40000.00, 10, yellow
Secretary: 40, $40000.00, 10, yellow
Taking dictation of text: CS12b example
```

Implementing `Lawyer`

- Lawyer regulations:
- Gets an extra week of paid vacation (a total of 3)
- Uses a pink form when applying for vacation leave
- Has some unique behavior: they know how to sue

- We want lawyer to inherit most behavior from er
  with new behavior

Overriding methods

- **Override**: To write a new version of a method in a
superclass's version
- No special syntax required to override a superclass me
subclass

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

Employee regulations

- Employee regulations:
- **Employee** works 40 hours / week
- **Employee** makes $40,000 per year, except **Legal Secretary** who makes $5,000 extra per year ($45,000 total), and **Marketer** who makes $10,000 extra per
- **Employee** have 2 weeks of paid vacation leave per year, e of 3)
- **Employee** should use a yellow form to apply for leave, exc

- Each type of employee has some unique behav
- **Lawyer** knows how to sue
- **Marketer** knows how to advertise

- **Secretary** knows how to take dictation
- **Legal Secretary** knows how to prepare legal documents

`Marketer` class

- Marketer makes $10,000 extra ($50,000 total) and k
advertise

```
// A class to represent marketers public
class Marketer extends Employee { public
void advertise() {
        System.out.println("Act now whil
    }
     // overrides getSalary from Employe
    public double getSalary() { return
    50000.0;      // $50,000 / year }
}
```

`LegalSecretary` class

- Legal secretary makes $5,000 extra per year ($45,00
knows how to prepare legal documents

```
// A class to represent legal secretarie
public class LegalSecretary extends Secr
{ public void fileLegalBriefs() {
        System.out.println("I could file
    }

    public double getSalary() { return
    45000.0;      //$45,000.00 }
```

Client program

```
public class EmployeeMain2 {

  public static void main(String[] args) {
     System.out.println("Lawyer:");
     Lawyer employee3 = new Lawyer();
     System.out.print(employee3.getHours() + ",
     System.out.printf("$%.2f, ", employee3.get
     System.out.print(employee3.getVacationDays
     System.out.println(employee3.getVacationFo
    employee3.sue();

     System.out.print("Legal Secretary: ");
     LegalSecretary employee4 = new LegalSecret
     System.out.print(employee4.getHours() + ",
     System.out.printf("$%.2f, ", employee4.get
```

```
        System.out.print(employee4.getVacationDays
        System.out.println(employee4.getVacationFo
        employee4.takeDictation("CS12b example");
       employee4.fileLegalBriefs();
    }
}
```