Section 10

MATERIAL PULLED FROM LAST SECTION AND LAST YEAR’S SLIDES
Today’s Agenda

Administrivia

Review Design Patterns

Design Pattern Worksheet

Course Review
Administrivia

Friday – Demos and Course Wrap-Up
  ◦ It’s not too late! Nominate your extra credit work to show off in class!

Wednesday – 8:30AM Exam (sorry)
Design Patterns

Creational patterns: get around Java constructor inflexibility
- Sharing: singleton, interning, flyweight
- Telescoping constructor fix: builder
- Returning a subtype: factories

Structural patterns: translate between interfaces
- Adapter: same functionality, different interface
- Decorator: different functionality, same interface
- Proxy: same functionality, same interface, restrict access
- All of these are types of wrappers
Design Patterns

What pattern would you use to...

- add a scroll bar to an existing window object in Swing
  - Decorator
- We have an existing object that controls a communications channel. We would like to provide the same interface to clients but transmit and receive encrypted data over the existing channel.
  - Proxy
Worksheet time!

Solutions will be posted online
Course Review
Stronger vs Weaker (one more time!)

Requires more?

weaker

Promises more? (stricter specifications on what the effects entail)

stronger
Stronger vs Weaker

@requires key is a key in this
@return the value associated with key
@throws NullPointerException if key is null

A. @requires key is a key in this and key != null
   @return the value associated with key
   WEAKER

B. @return the value associated with key if key is a key in this, or null if key is not associated with any value
   NEITHER

C. @return the value associated with key
   @throws NullPointerException if key is null
   @throws NoSuchElementException if key is not a key this
   STRONGER
Subtypes & Subclasses

Subtypes are substitutable for supertypes

If Foo is a subtype of Bar, G<Foo> is a NOT a subtype of G<Bar>
  - Aliasing resulting from this would let you add objects of type Bar to G<Foo>, which would be bad!
  - Example:
    ```java
    List<String> ls = new ArrayList<String>();
    List<Object> lo = ls;
    lo.add(new Object());
    String s = ls.get(0);
    ```

Subclassing is done to reuse code (extends)
  - A subclass can override methods in its superclass
Typing and Generics

<<?> is a wildcard for unknown
  ◦ Upper bounded wildcard: type is wildcard or subclass
    ◦ Eg: List<?> extends Shape>
    ◦ Illegal to write into (no calls to add!) because we can’t guarantee type safety.
  ◦ Lower bounded wildcard: type is wildcard or superclass
    ◦ Eg: List<?> super Integer>
    ◦ May be safe to write into.
Subtypes & Subclasses

class Student extends Object { ... }
class CSEStudent extends Student { ... }

List<Student> ls;
List<? extends Student> les;
List<? super Student> lss;
List<CSEStudent> lcse;
List<? extends CSEStudent> lecse;
List<? super CSEStudent> lscse;
Student scholar;
CSEStudent hacker;

ls = lcse;  ✗
les = lscse;  ✗
lcse = lscse;  ✗
les.add(scholar);  ✗
lscse.add(scholar);  ✗
lss.add(hacker);  ✓
scholar = lscse.get(0);  ✗
hacker = lecse.get(0);  ✓
Subtypes & Overriding

class Foo extends Object {
    Shoe m(Shoe x, Shoe y){ ... }
}

class Bar extends Foo {...}
### Method Declarations in Bar

<table>
<thead>
<tr>
<th>Method Declaration</th>
<th>Objects</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>FootWear m(Shoe x, Shoe y) { ... }</code></td>
<td><code>Object</code></td>
</tr>
<tr>
<td><code>Shoe m(Shoe q, Shoe z) { ... }</code></td>
<td><code>Footwear</code></td>
</tr>
<tr>
<td><code>HighHeeledShoe m(Shoe x, Shoe y) { ... }</code></td>
<td><code>Foo</code></td>
</tr>
<tr>
<td><code>Shoe m(FootWear x, HighHeeledShoe y) { ... }</code></td>
<td><code>Shoe</code></td>
</tr>
<tr>
<td><code>Shoe m(FootWear x, FootWear y) { ... }</code></td>
<td><code>Bar</code></td>
</tr>
<tr>
<td><code>Shoe m(Shoe x, Shoe y) { ... }</code></td>
<td><code>HighHeeledShoe</code></td>
</tr>
</tbody>
</table>
Method Declarations in Bar

- The result is method overriding
- The result is method overloading
- The result is a type-error
- None of the above

<table>
<thead>
<tr>
<th>Method Declaration</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>FootWear m(Shoe x, Shoe y) { ... }</td>
<td>type-error</td>
</tr>
<tr>
<td>Shoe m(Shoe q, Shoe z) { ... }</td>
<td>overriding</td>
</tr>
<tr>
<td>HighHeeledShoe m(Shoe x, Shoe y) { ... }</td>
<td>overriding</td>
</tr>
<tr>
<td>Shoe m(FootWear x, HighHeeledShoe y) { ... }</td>
<td>overloading</td>
</tr>
<tr>
<td>Shoe m(FootWear x, FootWear y) { ... }</td>
<td>overloading</td>
</tr>
<tr>
<td>Shoe m(Shoe x, Shoe y) { ... }</td>
<td>overriding</td>
</tr>
<tr>
<td>Shoe m(HighHeeledShoe x, HighHeeledShoe y) { ... }</td>
<td>overloading</td>
</tr>
<tr>
<td>Shoe m(Shoe y) { ... }</td>
<td>overloading</td>
</tr>
<tr>
<td>Shoe z(Shoe x, Shoe y) { ... }</td>
<td>none (new method declaration)</td>
</tr>
</tbody>
</table>
Exam
You got this!
We believe in you!

Wednesday 8:30AM!

Set you alarms now!