Object-Oriented Design in Java
William Marsh

Overview
• Classes and Interfaces
  – inheritance and composition
  – interfaces
• Case study: reusable list
  – Java collections
• Packages in Java and UML

Java Classes and Interfaces
• Composition versus Inheritance
• Widening and narrowing
• Interfaces

Composition and Inheritance

Overriding Methods
• Alternative implementation of method of the superclass
  • same name
  • same number of parameters
  • same parameter types and
    same return types
• If parameters are different, the method is overloaded rather than overridden

Widening
• Every subclass object is also a superclass object

```java
Deposit d = ... ;
Transaction t ;
t = d ;
```
• Every ‘Deposit’ is also a ‘Transaction’
Narrowing
• What about:
  Deposit d;
  Transaction t = ...;
  d = t;  

  compilation error
• Not all transactions are deposits!
  Deposit d;
  Transaction t = ...;
  d = (Deposit) t;  

  down cast

Quiz 1
• How to prevent a method being over-ridden?
• How to require a method to be over-ridden?
• How to call method from the super class?

Quiz 2 - Result of this Programme?
```java
class A {
    int x;
    A(int z) {
        x = z;
    }
    int m(int y){
        return x + y ;
    }
}

class B extends A {
    B(int z) {
        super(z);
    }
    int m(int y){
        return x - y ;
    }
}

A o = new A(1);
System.out.println(o.m(1));
```

Quiz 3 - Result of this Programme?
```java
class A {
    int x;
    A(int z) {
        x = z;
    }
    int m(int y){
        return x + y ;
    }
}

class B extends A {
    B(int z) {
        super(z);
    }
    int m(int y){
        return x - y ;
    }
}

class B o = new B(1);
System.out.println(o.m(1));
```

Quiz 4 - Result of this Programme?
```java
class A {
    int x;
    A(int z) {
        x = z;
    }
    int m(int y){
        return x + y ;
    }
}

class B extends A {
    B(int z) {
        super(z);
    }
    int m(int y){
        return x - y ;
    }
}

class B o = new B(1);
System.out.println(o.m(1));
```

Interfaces
• An ‘interface’ is similar to a class
  - methods but no attributes
  - signature but no implementation
• A ‘contract’
• Class ‘C’ implements interface ‘I’
  - class must have both methods
```java
class C implements I {
    void m1() { ... }
    void m2() { ... }
}```
Interfaces as Types - ‘Known As’

- C can be ‘known as’ I

```java
class C implements I {
    void m1() { ... }
    void m2() { ... }
}
```

C cVar = new C();
I iVar = cVar;

Interface v. Abstract Class

- Class can implement multiple interfaces
- Abstract class may contain implementation

```
<interface> I
m1() m2()

C
m1() m2()
```

```
<interface> I
m1() m2()

<interface> J
m3() m4()
```

C
m1() m2()

Why are Interfaces Important?

- FACT: interfaces are important!
- Circumstance
  - This program will work with any kind of object
  - … provided it can do an ‘m’
- Solution
  - Declare an interface ‘I’ with method ‘m’
  - … require the object to be an ‘I’
- Many examples
  - GUI programming using Swing

Case Study - Reusable Lists

Simple Solution: No reuse
Reusable Solution: More complex

List of Transactions?

- List code integrated with application code
- No reuse
- No abstraction

Reusable List

- Abstraction
  - ‘list’ is an abstraction:
    - head, tail, append
    - more than one list implementation
    - need an interface
- Reuse
  - class for each implementation
List Interface

- Methods to provide the 'list abstraction'

```java
public interface List {
    public int size();
    public boolean isEmpty();
    public Object head();
    public void append(Object item);
    public Object removeHead();
}
```

Polymorphism

- 'Many forms'
- List should be polymorphic
  - element of any type
- Java solution: elements have type Object
  - polymorphism by inheritance

**Quiz 5**

- How to hold an integer in a list?
  - basic types are not objects

List Implementation

```java
public class LinkedList implements List {
    public int size() { ... }
    public boolean isEmpty() { ... }
    public Object head() { ... }
    public void append(Object item) { ... }
    public Object removeHead() { ... }
}
```

**Quiz 6**

- Implement 'size' and 'isEmpty'

Linked List Implementation

```java
public class LinkedList implements List {
    protected int count;
    protected Node first;
    ... static protected class Node {
        Object element;
        Node next;
    }
    public Object head() { ... }
    public void append(Object item) { ... }
    public Object removeHead() { ... }
}
```

**Quiz 7**

Iterating through the list

- List contains 'Object' - only the variable name suggests content
- Program to the 'List' interface not implementation

```java
class Account {
    List transactions;
    /* what is in the list? */
    printTransactionsAmounts() {
        ...
    }
}
```

**Quiz 8**

- Current position in the list
  - 'cursor'
  ```java
  void reset() {
      Move to the start of the list
  }
  Object next() {
      Return the element in the current
      position and advance the position
  }
  boolean hasNext() {
      Return true if not at the end
  }
  ```
Quiz 9 – Using ‘Iteration Methods’

- Rewrite `printTransactionAmounts`

```
List trans; /* list of transactions */
printTransactionsAmounts() {
    ...
}
```

Idea 1 – Add ‘Iteration Methods’ to List Interface

```
protected Node cursor;
void reset() { cursor = first; }
boolean hasNext() { return (cursor != null); }
Object next() {
    Node n;
    if (cursor != null) {
        n = cursor.element;
        cursor = cursor.next;
    }
    return n;
}
```

Idea 2 – Separate ‘Iterator Interface’

```
public interface Iterator {
    void reset();
    boolean hasNext();
    Object next();
}
```

- ‘LinkedList’ implements ‘List’ and ‘Iterator’

Idea 3 – Separate the Iterator

```
public interface List {
    ...
    public Iterator iterator();
}
```

Java Collections

Java provides a set of ‘collection’ classes; what are they for and how are they used?
Java Collection Classes

• Good news!
  – difficult algorithms implemented for you
• Interfaces
  
  ![Diagram of Collection classes]

• Implementations
  
  ![Diagram of implementation classes]

Using Collections

• Easy to use

```java
List trans = new LinkedList();
printTransactionsAmounts() {
  Iterator I = trans.iterator();
  while (i.hasNext()) {
    Transaction t = (Transaction) i.getNext();
    System.out.println(t.getAmount());
  }
}
```

Java 1.5

• Java 1.5 has new features to make
  – The program on the previous slide shorter
  – The Java language more complex
• The concepts illustrated by the collection class remain important
  – Only the example becomes obsolete

More Topics

• ‘Map’ interface and implementations
  – finite functions
• How sorted sets and map are handled
• Lab sheet 2 – ‘Using Java Collections’

Sorted Sets – How?

• `Comparator` interface

```java
public interface Comparator {
  int compare(Object o1, Object o2);
}
```

• `SortedSet` implementations have a constructor which takes a comparator

```java
public TreeSet(Comparator c)
```
Packages in Java and UML

- Package visibility
- Two design ideas

Java Packages

```java
package myPackage;
import anotherPackage.ClassName;
public class MyClass { … }
```

- Each class in a file
- Package
  - group of related classes
  - directory! (see Jia p137)

UML Package Diagram

```
packageX
+ ClassA
+ ClassB
packageY
```

- How useful?

Design Idea: Facade

- Present a simpler interface to the outside world
- Can do this with ‘package’ visibility

Design Idea: Factory Method

- Rather than a constructor, provide a method of class A which returns an instance of class B
  - Class B has a constructor, but it is not visible

- Quiz: already seen an example of this – where?
- Quiz: why do interfaces often contain factory methods?

Summary
<table>
<thead>
<tr>
<th>Summary</th>
<th>Reading</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Java language</td>
<td>• General</td>
</tr>
<tr>
<td>– Interfaces</td>
<td>– Jia chapters 5 &amp; 6</td>
</tr>
<tr>
<td>– Inner classes</td>
<td>– Bennet et al. chapters 11 &amp; 13</td>
</tr>
<tr>
<td>– Packages</td>
<td>• Collections</td>
</tr>
<tr>
<td>– etc</td>
<td>– Jia section 8.2</td>
</tr>
<tr>
<td>• ... used to design reusable software</td>
<td>• On the web</td>
</tr>
<tr>
<td>– Java library</td>
<td>– Collections</td>
</tr>
<tr>
<td>• Is reusable software simpler and easier?</td>
<td><a href="http://www.javaworld.com/javaworld/jw-12-1998/jw-12-techniques.html">http://www.javaworld.com/javaworld/jw-12-1998/jw-12-techniques.html</a></td>
</tr>
<tr>
<td></td>
<td>Article on interfaces and abstract classes</td>
</tr>
<tr>
<td></td>
<td>Article on interfaces and composition</td>
</tr>
</tbody>
</table>