6.005
elements of
software
construction

decoupling & interfaces

Daniel Jackson
september 19, 2007
intro
topics for today

last time
- hierarchical naming, scope of vars, importance of minimizing scope
- access modifiers, quoter example (slides you read after class)

role of names in software design
- how does a module name functionality in another module?

locality
- localizing changes within modules
- a form of “separation of concerns”

interfaces in Java
- have seen idea already
- today, see role in design of plugins
locality
software design

biggest challenge in software design

• locality: understand modules independently
• maintaining locality as software grows

how to design for locality

• assign each specification feature to at most one module
  so change to a feature only affects one module
• avoid dependences (coupling) between modules
  so change doesn’t propagate
feature-module assignment

example: change in high-vol feature will impact amp and speakers
note couplings between modules too

see: Nam Suh, Axiomatic Design: Advances and Applications, Oxford University Press, 2001;
David Parnas, On the Criteria to Be Used in Decomposing Systems Into Modules, CACM, 1972.
avoiding coupling

a balancing act

• modules connected by rich channels --> flexible, easy access
• modules connected by thin channels --> changes are contained

solution

• as thin as possible, but no thinner
• module-level software design = design of specs
interfaces in Java
what interfaces give you

what interfaces give you

• anonymous use -- the “better” scenario from above
• implementation determined at runtime, by runtime type of object passed
• so constructor call determines choice of implementation

applications

• can design program with “plugins” to parameterize functionality
  class can assume partial properties of objects
  example: java.util.TreeSet takes elements that implement Comparable

marker interfaces

• declare no methods
• used to expose spec properties (eg. java.util.RandomAccess)
• or as hack to add functionality (eg. java.io.Serializable)

© Daniel Jackson 2007
declaring an interface

declare List interface

```java
package java.util;
public interface List {
    boolean add (Object e);
    void clear ();
    ...
}
```

declare ArrayList class

```java
package java.util;
public class ArrayList implements List {
    boolean add (Object e) {...}
    void clear () {...}
    ...
}
```

‘implements’ claim causes compile-time check

• ensures that object of type ArrayList has methods of interface List

© Daniel Jackson 2007
using an interface

now use class only in constructor

' can switch to another class, eg. LinkedList, with edit in just one place

```java
package music;
import java.util.List;
import java.util.ArrayList;

class MusicMachine {
    boolean percussionMode, recordMode;
    List<Character> recording = new ArrayList<Character>();
    ...

    public void noteKeyPressed (char key) {
        Midi.play(key);
        if (recordMode) recording.add(key);
    }

    public void playKeyPressed () {
        for (char k: recording) Midi.play(k);
    }

    public void recordKeyPressed () {
        recordMode = !recordMode;
        if (recordMode) recording.clear();
    }
}
```
quote generation example
quote generation problem

problem

• want to obtain stock quotes for some ticker symbols
• produce both RTF and HTML output
• put ask price in bold if change since open $\geq \pm 1\%$
design challenge

separate functionality and minimize coupling

- **Quoter** doesn't know about generating text
- **QuoteDisplayer** doesn't know about HTML or RTF

just invokes QD with tickers & selects output format

‘dependency diagram’ arc means “knows name of”

obtains and outputs quotes

obtains quotes

obtains and outputs quotes
text generator

key design idea

' develop generic interface for text formatting
package generator;

public interface Generator {
    public void open () throws Exception;
    public void close ();
    public void newLine ();
    public void toggleBold ();
    public void toggleItalic ();
    public void write (String s);
}
public class RTFGenerator implements Generator {
    boolean italic, bold;
    String filename;
    PrintStream stream;

    public RTFGenerator (String filename) {
        this.filename = filename;
    }

    public void open() throws FileNotFoundException {
        FileOutputStream fos = new FileOutputStream (filename);
        stream = new PrintStream(fos);
        stream.println ("{\rtf1\mac");
    }

    public void close() {
        stream.println ("}");
        stream.close();
    }

    public void newLine () {
        stream.println ("\\");
    }

    public void toggleBold() {
        stream.println (bold ? "\\f\\b0" : "\\f\\b");
        bold = !bold;
    }

    ...
public class Quoter {
    private URL url;
    private String open, ask;
    private int change;

    public Quoter (String symbol) throws MalformedURLException {
        url = new URL("http://quote.yahoo.com/d/quotes.csv?s="+symbol+"&f=oap2");
    }

    public String getOpen () {return open;}
    public String getAsk () {return ask;}
    public int getChange () {return change;}

    public void obtainQuote () throws IOException {
        BufferedReader in = new BufferedReader(new InputStreamReader(url.openStream()));
        String csv = in.readLine();
        in.close();
        StringTokenizer tokenizer = new StringTokenizer(csv, ",");
        open = tokenizer.nextToken();
        ask = tokenizer.nextToken();
        change = (int) (100 * (Float.valueOf(ask)-Float.valueOf(open)) / Float.valueOf(open));
    }
}

© Daniel Jackson 2007
public class QuoteDisplayer {
    Set<String> symbols = new HashSet<String>();
    Generator gen;

    public QuoteDisplayer (Generator gen) {this.gen = gen;}
    public void addSymbol (String symbol) {symbols.add (symbol);}

    public void generateOutput () throws Exception {
        gen.open ();
        for (String symbol: symbols) {
            Quoter q = new Quoter (symbol);
            q.obtainQuote();
            gen.write (symbol + " : ");
            gen.toggleItalic ();
            gen.write ("opened at ");
            gen.toggleItalic ();
            gen.write (q.getOpen ());
            gen.toggleItalic ();
            gen.write (" and is currently trading at ");
            gen.toggleItalic ();
            boolean bigChange = Math.abs (q.getChange()) >= 1;
            if (bigChange) gen.toggleBold();
            gen.write (q.getAsk ());
            if (bigChange) gen.toggleBold();
            gen.newLine();
        }
        gen.close();
    }
}
public class QuoteApp {

    public static void main(String[] args) throws Exception {

        Generator rtfg = new RTFGenerator("myQuotes.rtf");
        QuoteDisplayer disp = new QuoteDisplayer(rtfg);
        disp.addSymbol("AAPL");
        disp.addSymbol("INTC");
        disp.addSymbol("JAVA");
        disp.addSymbol("MSFT");
        disp.generateOutput();

        Generator htmlg = new HTMLGenerator("myQuotes.html");
        disp = new QuoteDisplayer(htmlg);
        disp.addSymbol("AAPL");
        disp.addSymbol("INTC");
        disp.addSymbol("JAVA");
        disp.addSymbol("MSFT");
        disp.generateOutput();
    }
}
Exercise

Which modules would you need to modify to...

- handle new RTF syntax for italics?
- put ask price in bold if down since open?
- use Google Finance instead of Yahoo?
- add year-to-date change to report?
what's happening

- *QuoteDisplayer* uses plugin to generate formatted text
- ignorant of whether it's using HTML or RTF generator
- refers to generator only by interface name *Generator*
Costs of Knowledge

Illustrates general principle

- Suppose module M1 used module M2
- If M1 knows M2's internals, then M2 is not a replaceable component
- If M1 knows M2's name, then two versions of M2 cannot coexist
the joy of ignorance

good (left)

- M1 depends only on service S2 provided by M21
- can switch to M22 by modifying refs in M1, or renaming M22 to M21

better (right)

- M1 depends only on S2, and doesn't even name M21
- M21 and M22 can coexist; different approaches for configuring
review
summary

locality
- achieved by containing functions within modules
- and by limiting coupling between modules

interfaces
- key mechanism for achieving decoupling
- class depends only on spec of another class

dependency diagram
- show code modules as boxes
- arc from A to B when A uses a name declared in B, or the name B itself