6.005 Elements of Software Construction

Fall 2008

6.005 elements of software construction

Decoupling and Interfaces

Rob Miller
Fall 2008

Today's Topics

principles and concepts of system design

- modularity
- decoupling
- information hiding

a new notation

- module dependency diagram

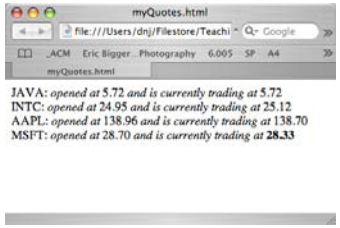
case study: designing a stock quoter

- using interfaces to decouple modules

Quote Generation Problem

problem

- obtain stock quotes for some ticker symbols
- produce both RTF and HTML output
- put the *ask* price in bold if the change since open is ≥ ± 1%



Design Tasks

tasks, for each ticker symbol:

- download quote information from web site
- parse to extract stock quotes
- write to file in RTF or HTML format

parsing

- minimize parsing by choosing a site with a simple format
- Yahoo offers stock quotes in comma-separated-values (CSV) format

example

- http://quote.yahoo.com/d/quotes.csv?s=aapl&f=noa
- returns the string `"APPLE INC",130.75,125.20`

Downloading & Parsing

```
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=noa");
  }
  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.open..));
    String csv = in.readLine();
    in.close();

    String[] fields = csv.split(“,”);
    open = fields[1];
    ask = fields[2];
    change = (int)(100 *(Float.valueOf(ask)-Float.valueOf(open))
                    / Float.valueOf(open));
  }
}
```

why are the fields of Quoter private?

Quoter is a state machine. Draw it. What design pattern does it use?

BufferedReader is also a state machine. Draw it. What design pattern does it use?

Modularity and Decoupling

modularity is essential for managing complexity

- system is divided into parts (modules) that can be handled separately and recombined in other combinations

coupling

- degree of dependence between parts of the system
- an important measurement of modularity

decoupling achieved so far

- the website (Yahoo) and its format (CSV) have been decoupled from the rest of the system

next step

- design the part of the system that generates the report
- report can be either HTML or RTF

Design Option #1

just build two formatters that use Quoter

```
public class HTMLFormatter {
    private final Set<String> symbols = new HashSet<String> ();
    ...
    public void generateOutput () throws IOException {
        PrintStream out = new PrintStream(new FileOutputStream (...));
        out.println("<html>");
        for (String symbol: symbols) {
            Quoter q = new Quoter (symbol);
            q.obtainQuote();
            out.println(symbol + ": "
                        + "<i>opened at</i> " + q.getOpen ()
                        + "<i> and is currently trading at </i>");
            boolean bigChange = Math.abs (q.getChange()) >= 1;
            if (bigChange) out.println("<b>");
            out.println(q.getAsk ());
            if (bigChange) out.println("</b>");
            out.println("<br>");
         }
         out.close();
     }
}
```

How would the RTF version differ? What's undesirable about this choice?

Design Option #2

build one formatter that takes a flag (RTF or HTML)

- tests flag to determine flow of control

```
public class Formatter {
    public enum Format { HTML, RTF };
    private final Format format;
    ..
    public void generateOutput () throws IOException {
       PrintStream out = new PrintStream(new FileOutputStream (...));
       out.println(format == HTML ? "<html>“ : "{\\rtf1\\mac“);
       for (String symbol: symbols) {
          ...
          boolean bigChange = Math.abs (q.getChange()) >= 1;
          if (bigChange) out.println(format == HTML ? "<b>“ : "\\f\\b“);
          out.println(q.getAsk ());
          if (bigChange) out.println(format == HTML ? "</b>“ : "\\f\\b0“);
          out.println("<br>");
       }
       ...
    }
}
```

Is this a wise way to test the format flag?

What's undesirable about this choice?

A Better Solution

factor out responsibilities for report generation

- generator: knows *how* to put in bold, italics, etc.
- formatter: knows *what* to put in bold, italics, etc.

designing the generator

- make it a state machine!
- two versions, one RTF and one HTML
- but *same interface*

A Principle

localize each design decision in exactly one place

- more crudely: "don't repeat yourself"

why?

- ready for change: if decision needs to change, there's only one place
- ease of understanding: don't have to think about the details of that decision when working on the rest of the system
- safety from bugs: fewer places to change means less chance of omission

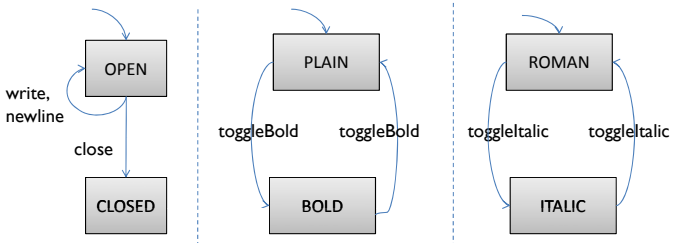
variations on the same idea

- Information hiding: localizing design decision and protecting the rest of the system from it
- Encapsulation: wrapping code up into a module that hides information
- Separation of concerns: responsibility for a feature is given to one module, not spread across system

Generator Machine

key design idea

- develop generic interface for text formatting



A Generator

```
public class RTFGenerator implements Generator {
  private boolean italic;
  private boolean bold;
  private final String filename;
  private 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; }
  ...
}
```

The Big Question

how to make formatter independent of generator?

- we want them decoupled
- so we can plug in different generators
- without changing the formatter's code

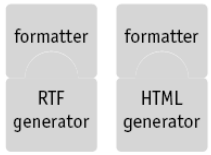
solution

- formatter doesn't refer to a particular generator class
- it refers to an **interface** instead

Interfaces, in Pictures

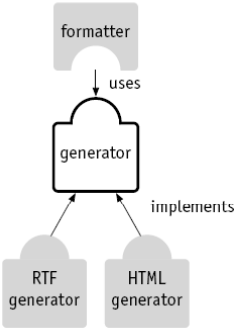
what we want

- two ways to configure formatter



how does formatter refer to generators?

- with an interface



Generator Interface

```
/**
 * Interface for generator with basic text formatting.
 * Typically a stream is passed to the constructor.
 */
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 {
    public void open() throws FileNotFoundException { ... }
    ...}
public class HTMLGenerator implements Generator {
    public void open() throws FileNotFoundException { ... }
    ...}
```


Using the Generator Interface

```
public class QuoteFormatter {
  private final Set<String> symbols = new HashSet<String> ();
  private final Generator generator;

  public QuoteFormatter(Generator generator) {
    this.generator = generator ;
  }
  public void addSymbol (String symbol) {
    symbols.add (symbol);
  }
  public void generateOutput () throws Exception {
    generator.open ();
    for (String symbol: symbols) {
      Quoter q = new Quoter (symbol);
      q.obtainQuote();
      generator.write (symbol + ": ");
      generator.toggleItalic ();
      generator.write ("opened at ");
      generator.toggleItalic ();
      ...
    generator.close();
  }
}
```

an object implementing Generator is plugged into the formatter

no mention of HTMLGenerator or RTFGenerator anywhere!

Putting Everything Together

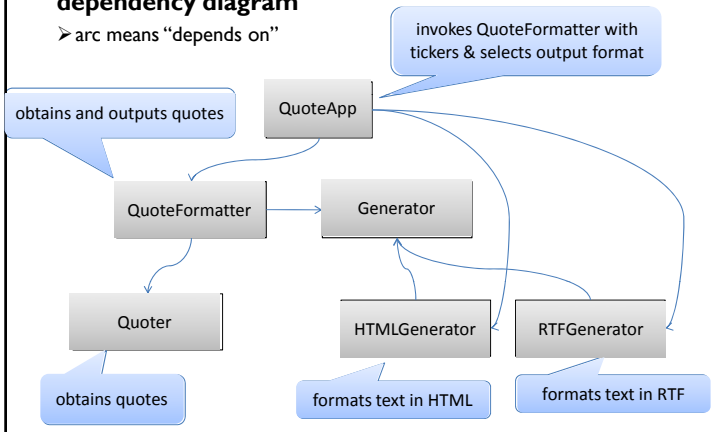
```
public class QuoteApp {

  public static void main(String[] args) throws Exception {
    Generator rtfg = new RTFGenerator ("myQuotes.rtf");
    QuoteFormatter formatter = new QuoteFormatter(rtfg);
    formatter.addSymbol ("AAPL");
    formatter.addSymbol ("INTC");
    formatter.addSymbol ("JAVA");
    formatter.addSymbol ("MSFT");
    formatter.generateOutput ();

    Generator htmlg = new HTMLGenerator ("myQuotes.html");
    formatter = new QuoteFormatter(htmlg);
    formatter.addSymbol ("AAPL");
    formatter.addSymbol ("INTC");
    formatter.addSymbol ("JAVA");
    formatter.addSymbol ("MSFT");
    formatter.generateOutput ();
  }
}
```

plugin is selected here

What We've Achieved

dependency diagram

- arc means "depends on"



Exercise

which modules would you need to modify to...

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

An Interface is a Specification

a general strategy

- client should only know about the **specification** of the service it uses
- so decouple the client from the service by interposing the specification

in Java:

- the specification is declared by an interface
- the service is plugged in by passing an object implementing that interface

specification is a contract

- we'll see more about this idea in later lectures

Other Uses of Interfaces

decoupling from choice of representation

- very common and important

```
List<NoteEvent> recording = new ArrayList<NoteEvent>();
recording.add(...);

List<NoteEvent> recording = new LinkedList<NoteEvent> ();
recording.add(...);
```

"marker" interfaces

- declare no methods
- used to expose specification properties (e.g. java.util.RandomAccess)
- or as a hack to add functionality (e.g. java.io.Serializable)

Summary

system design principles

- modularity
- decoupling using interfaces

dependency diagrams

- show essence of code design
- missing dependences are the interesting ones!