Today’s Topics

**principles and concepts of system design**
- modularity
- decoupling
- information hiding

**More examples of module dependency diagrams**

**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 $\geq \pm 1\%$

Design Tasks?
public class WebQuoter {
    private URL url;
    private String open, ask;
    private int change;

    public WebQuoter (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 WebQuoter private?

What happens if a get method is called before obtainQuote()?

Decoupling achieved so far?
Design Option #1

just build two formatters that use WebQuoter

```java
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) {
            WebQuoter q = new WebQuoter (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 design option?
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>" : "\rtf\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 option?
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*

This solution follows a principle
Generator State Machine

key design idea

- develop generic interface for text formatting

![Diagram of state machine showing transitions between OPEN, CLOSED, PLAIN, BOLD, ROMAN, and ITALIC states, with transition actions such as write, newline, toggleBold, toggleItalic, and close.]
public class RTFGenerator {
    private boolean italic, 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; }

    public void write(String s) {
        stream.println (s); } ...}
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 { ... }
    ...}
public class RTFGenerator implements Generator {
    private boolean italic, 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 ("{\\rtfl\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 void write(String s) {
        stream.println (s); }
}
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) {
            WebQuoter q = new WebQuoter(symbol);
            q.obtainQuote();
            generator.write(symbol + " : ");
            generator.toggleItalic();
            generator.write("opened at ");
            generator.toggleItalic();
            ...
            generator.close();
        }
    }
}
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();
    }
}

What Does the Module Dependence Diagram Look Like?
Exercise

which modules would you need to modify to...

➢ handle new RTF syntax for bold?

```java
public class RTFGenerator implements Generator {
    private boolean italic, 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 toggleBold() {
        stream.println (bold ? "\f\b0" : "\f\b");
        bold = !bold; }

    public void write(String s) {
        stream.println (s); } ...
```
Exercise

which modules would you need to modify to...

➢ put the ask price in bold if the stock went down since open?

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

    public QuoteFormatter(Generator generator) {
        this.generator = generator;
    }

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

Look at change and toggle bold
Exercise

which modules would you need to modify to...

➢ use Google Finance instead of Yahoo?

```java
public class WebQuoter {
    private URL url;
    private String open, ask;
    private int change;

    public WebQuoter (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;}

    ...
}
```
Exercise

which modules would you need to modify to...

➢ add year-to-date change to the report?
  • QuoteFormatter and WebQuoter
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