Midterm Reminders

The midterm is next Tuesday: March 4, 7-9 p.m.
If you have a conflict and haven't told us, send e-mail TODAY!
(conflicts with other Queens courses or exams)

Location: Stirling Hall rooms A, B and C
By last name (regardless of section):

Last Name	Room
A-J	Stirling A
K-O	Stirling B
P-Z	Stirling C

Midterm covers all material through Friday.
Readings for all topics on web site.
Monday: review, Q&A

Design Pattern #5: Builder

- This is a "creational" design pattern: has to do with how objects are created
- Situation: a program is creating a data structure (or file)
- Two aspects to the job:
  - figuring out what to create (from parsing an input file, computing, changing an existing data structure, etc.)
  - creating the data structure
- Builder pattern separates these two aspects

Example: RTF converter

- This example based on example in courseware
- Task: read an RTF file and convert it to another format
- RTF = **R**ich **T**ext **F**ormat
  - a simple, portable representation for formatted text
  - many word processing programs will read and write RTF (Word, WordPad, Word Perfect, etc.)
  - format is text with commands interspersed

Examples of RTF

Line in RTF file:

```
Mary had a \b little \b0 lamb.
```

Produces:

Mary had a **little** lamb.

Another way:

```
Mary had a {\b little} lamb.
```

Produces the same result.

- More commands to change fonts, color, lots of different formatting features.
- Example code on web recognises a tiny subset of RTF.

Task: Convert RTF to HTML

- read in an RTF file and build an HTML file to show roughly the same thing
- straightforward way looks like this:

```
while (not at the end of RTF file):
  read a bit more of the RTF file
  figure out what HTML commands are needed
     to get the same effect
   write the HTML commands to the output
     file
```

- Two things going on here:
  - parsing and understanding the RTF
  - building the HTML file

Builder: Separation of Concerns

- The Builder pattern separates these two aspects of the task
- A Builder object knows how to create an HTML file
  - provides methods such as:
    - `addString(String s)`
    - `addChar(char c)`
    - `setBold(boolean on)`
    - `setFontSize(double size)`
- A Director object reads and parses the RTF and calls methods in the Builder to write the HTML

Sequence of Events (1)

- User creates a Builder, specifying the output file.
  - Builder writes HTML header stuff to file.

Sequence of Events (2)

- User creates a Director, specifying the input file and the Builder object to use.

Sequence of Events (3)

- User calls Director's **`construct()`** method.
- construct method takes charge of the rest of the conversion
  - reads RTF
  - calls methods in Builder

Sequence of Events (4)

- Example: construct reads the normal character **`'a'`**
  - calls **`builder.addChar('a')`**
  - Builder writes **`'a'`** to the HTML file

Sequence of Events (5)

- Another Example: construct reads the special character **`'\'`**
  - reads more characters to see that the command is **`'\b'`**
  - calls **`builder.setBold(true)`**
  - Builder writes **`"<b>"`** to the HTML file

Sequence of Events (6)

- When input file is exhausted, construct calls **`builder.finish()`**
- finish method writes ending HTML tags and closes file

Advantages

- separation of concerns
  - one class handles the RTF format
  - another hands the HTML format
- changes to a format require change to only one part of the code

Disadvantages

- a bit more initial programming effort to separate code in this way, may be awkward:
- slight performance penalty: extra method calls

Expanded Scenario

- Need to be able to read RTF files and translate into several formats:
  - HTML
  - plain text (ignore fonts, bold, etc.)
  - GUI component for display on screen
  - internal data structure (parse tree?) for further processing
- Now Builder pattern is much more useful

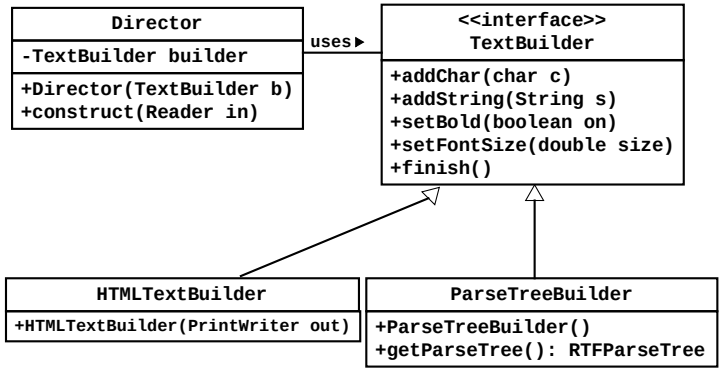
Without Builder Pattern

- Four methods (or classes):
  - translate RTF to HTML
  - translate RTF to plain text
  - read RTF and generate a GUI component
  - read RTF and generate a parse tree
- Each method reads and parses RTF input.
  - logic for RTF input intertwined with logic for each kind of output
  - a nuisance for programmer, even with cut-and-paste
  - changes or corrections to RTF reading must be done in 4 places

With Builder Pattern

- Much easier.
- Same Director for all 4 translations.
- Four Builders, one for each output format
- If change is needed to RTF format reading, it is made in one place (the Director)
- HTML and plain text Builders create files
- GUI and parse tree Builders create objects in memory
  - Builder has additional method to let user retrieve the object

Class Diagram (with 2 Builders)



Program to Translate RTF to HTML

```
<open input file inFile>
<open output file outFile>
TextBuilder builder = new HTMLTextBuilder(outFile);
Director dir = new Director(builder);
dir.construct(inFile);
```

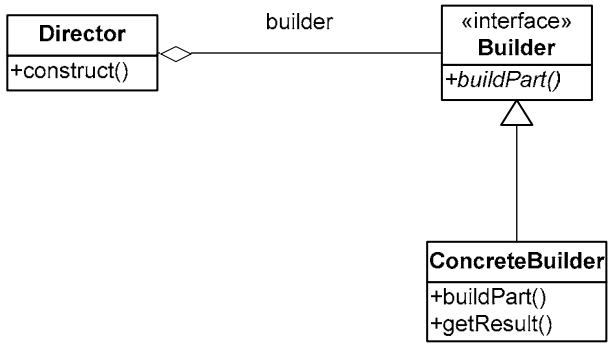

Program to Create an RTF Parse Tree

```
<open input file inFile>
TextBuilder builder = new ParseTreeBuilder();
Director dir = new Director(builder);
dir.construct(inFile);
RTFParseTree tree = builder.getParseTree();
```

Example Code On Web

- Program to convert from one of three input formats:
  - plain text
  - RTF
  - list of student records
- to one of three output formats:
  - plain text
  - RTF
  - HTML
- Without Builder might take 9 conversion methods
  - one for each combination of input and output format
- With Builder it takes 3 Directors and 3 Builders
- Uses a TextDirector interface with three implementations

General Form of Builder Pattern



Another Example: Using Java Files

- Want program to read a Java program and create a parse tree
- Another program to pretty-print a Java program (write to a file, no parse tree needed)
- Yet another program to read a Java program and generate statistics
- Use one Director for parsing the Java program
- Builder for creating "parse tree":
  - parse tree Builder really creates an internal tree
  - pretty-printing Builder just writes output to a file
  - statistics Builder records statistics for user to query
- Code for parsing a Java file shared

More About Directors

- In most examples, a director is reading from a file – translating from one format to another
- Also possible to have a director creating a data structure from scratch.
- Example: Java GUI class generator
  - reads specifications for GUI components
  - creates file with Java class to create the GUI
- Can use pretty-printing Builder from previous example to generate the Java class files
- Director will read specifications and call Builder methods