**Today’s topics**

Java

Syntax of Computer Language

Upcoming

More Java

Reading

*Great Ideas, Chapter 2*

---

**Grammar**

- English and other natural languages have structure
  
  
  \[
  \text{\texttt{S}} \rightarrow \text{\texttt{NOUN- PHRASE} \ NOUN-PH RASE} \\
  \text{\texttt{NOUN- PHRASE}} \rightarrow \text{\texttt{NOUN}} \mid \text{\texttt{ARTICLE} \ NOUN} \\
  \text{\texttt{VERB-PH RASE}} \rightarrow \text{\texttt{VERB}} \mid \text{\texttt{VERB} \ NOUN-PH RASE} \\
  \text{\texttt{NOUN}} \rightarrow \text{\texttt{DOG}} \mid \text{\texttt{FLEAS}} \mid \text{\texttt{PERSON}} \mid \ldots \\
  \text{\texttt{VERB}} \rightarrow \text{\texttt{RAN}} \mid \text{\texttt{BIT}} \mid \ldots
  \]

- Process of taking sentence and fitting it to grammar is called *parsing*

  DOG BIT PERSON

  \[
  \text{\texttt{NOUN} \ VERB \ NOUN} \\
  \text{\texttt{NOUN-PH RASE} \ VERB-PH RASE} \\
  \text{\texttt{S}}
  \]

- Parsing English is complex because of *context dependence*

---

**Formal specifications**

- Need a precise notation of syntax of a language
- Grammars can also be used for generation
- Grammar rule:
  
  \[
  \text{\texttt{name} \rightarrow sequence of letters and/or digits that begins with a letter}
  \]

- Grammar use:
  
  \[
  \text{\texttt{name} \rightarrow guessB} \\
  \text{\texttt{name} \rightarrow msg42}
  \]

- Substitute as many times as necessary. All *legal* statements can be generated this way

- Want: \texttt{person = firstn + " " + lastn;}
  
  - How do we get this from our grammar?

---

**A Grammar for Java**

- Need a set of rules
- Our first one was a good start:
  
  \[
  \text{\texttt{name} \rightarrow any string of alphanumeric symbols that begins with a letter}
  \]

- Let’s add something to define a simple statement:
  
  \[
  \text{\texttt{statement} \rightarrow \texttt{name} = \texttt{expression} ;}
  \]

- And then work on the details:
  
  \[
  \text{\texttt{expression} \rightarrow \texttt{string-expression} \mid \texttt{int-expression} \mid \texttt{oth-expression}}
  \]

  \[
  \text{\texttt{string-expression} \rightarrow \texttt{string}}
  \]

  \[
  \text{\texttt{string} \rightarrow \texttt{string} \rightarrow \texttt{name}}
  \]

  \[
  \text{\texttt{string} \rightarrow \texttt{string} \rightarrow \texttt{"any sequence of characters"}}
  \]
A Simple Statement

- Now have enough to generate a statement like:
  ```
  msg = "hello";
  ```
- Start with:
  ```
  <statement> => <name> = <expression> ;
  ```
- Then using: <name> => any string of alphanumeric symbols that begins with a letter

```
msg = <expression> ;
```
- Then, using: <expression> => <string-expression> | <int-expression> | <oth-expression>

```
msg = <string-expression> ;
```
- Using: <string-expression> => <string>

```
msg = <string> ;
```
- Using: <string> => “any sequence of characters”

```
msg = "hello" ;
```

A Grammar for Java

- Including more rules to describe programs we have:
  1. <name> => any string of alphanumeric symbols that begins with a letter
  2. <statement> => <name> = <expression> ;
  3. <statement> => <name> = new <class> ( <arguments> ) ;
  4. <statement> => <name> , <method> ( <arguments> ) ; |
  ```
  <method> ( <arguments> ) ;
  ```
  5. <arguments> => possibly empty list of <expression>s separated by commas
  6. <expression> => <string-expression> | <int-expression> | <oth-expression>
  7. <string-expression> => <string-expression> + <oth-expression>
  8. <string-expression> => <string>
  9. <string> = "any sequence of characters"
  10. <string> = <name>

Using our Grammar

- Use this to generate:
  ```
  person = firstn + " " + lastn;
  ```

<table>
<thead>
<tr>
<th>Rule</th>
<th>Statement being Generated</th>
</tr>
</thead>
<tbody>
<tr>
<td># 2:</td>
<td>&lt;statement&gt; =&gt; &lt;name&gt; = &lt;expression&gt; ;</td>
</tr>
<tr>
<td>1:</td>
<td>&lt;statement&gt; =&gt; person = &lt;expression&gt; ;</td>
</tr>
<tr>
<td>6:</td>
<td>&lt;statement&gt; =&gt; person = &lt;string-expression&gt; ;</td>
</tr>
<tr>
<td>7:</td>
<td>&lt;statement&gt; =&gt; person = &lt;string-expression&gt; + &lt;str-expression&gt; ;</td>
</tr>
<tr>
<td>8:</td>
<td>&lt;statement&gt; =&gt; person = &lt;string&gt; + &lt;str-expression&gt; ;</td>
</tr>
<tr>
<td>10:</td>
<td>&lt;statement&gt; =&gt; person = &lt;name&gt; + &lt;str-expression&gt; ;</td>
</tr>
<tr>
<td>1:</td>
<td>&lt;statement&gt; =&gt; person = firstn + &lt;str-expression&gt; ;</td>
</tr>
<tr>
<td>7:</td>
<td>&lt;statement&gt; =&gt; person = firstn + &lt;str-expression&gt; + &lt;str-expression&gt; ;</td>
</tr>
<tr>
<td>8:</td>
<td>&lt;statement&gt; =&gt; person = firstn + &lt;string&gt; + &lt;str-expression&gt; ;</td>
</tr>
<tr>
<td>9:</td>
<td>&lt;statement&gt; =&gt; person = firstn + &quot; &quot; + &lt;str-expression&gt; ;</td>
</tr>
<tr>
<td>8:</td>
<td>&lt;statement&gt; =&gt; person = firstn + &quot; &quot; + &lt;string&gt; ;</td>
</tr>
<tr>
<td>10:</td>
<td>&lt;statement&gt; =&gt; person = firstn + &quot; &quot; + &lt;name&gt; ;</td>
</tr>
<tr>
<td>1:</td>
<td>&lt;statement&gt; =&gt; person = firstn + &quot; &quot; + lastn ;</td>
</tr>
</tbody>
</table>

Proving Grammatical Correctness

- Why go through the process we went through?
  - Shows that desired statement can be generated from this grammar
  - Actually proves that the statement is grammatically correct!
  - Same rigor as a mathematical proof
  - (Does not prove that logic is correct, though)
- Actually need more rules to handle the level of Java we’ve covered so far
  - Summary of rules shown on pages 78-80 of Great Ideas
  - Also give an example for a complete applet
  - Too long to go through in class – Please Read!