Today’s topics

Parsing
Java Programming

Reading
Great Ideas, Chapter 3 & 4

A Grammar for Java

- Need a set of rules
- Our first one was a good start:
  - `<name> => any string of alphanumeric symbols that begins with a letter`
- Let’s add something to define a simple statement:
  - `<statement> => <name> = <expression> ;`
- And then work on the details:
  - `<expression> => <string-expression> | <int-expression> | <oth-expression>`
  - `<string-expression> => <string>
    - `<string> => <name>
    - `<string> => "any sequence of characters"
  - `<int-expression> => <string>
  - `<oth-expression> => <string>

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`
    - `<expression> = <string-expression> ;`
    - Using: `<string-expression> => <string>`
      - `<string> = "any sequence of characters"
      - `<string> = <name>`
### Using our Grammar

<table>
<thead>
<tr>
<th>Rule</th>
<th>Statement being Generated</th>
</tr>
</thead>
<tbody>
<tr>
<td>#2</td>
<td><code>&lt;statement&gt; =&gt; &lt;name&gt; = &lt;expression&gt;;</code></td>
</tr>
<tr>
<td>1:</td>
<td><code>&lt;statement&gt; =&gt; person = &lt;expression&gt;;</code></td>
</tr>
<tr>
<td>6:</td>
<td><code>&lt;statement&gt; =&gt; person = &lt;str-expression&gt;;</code></td>
</tr>
<tr>
<td>7:</td>
<td><code>&lt;statement&gt; =&gt; person = &lt;str-expression&gt; + &lt;str-expression&gt;;</code></td>
</tr>
<tr>
<td>8:</td>
<td><code>&lt;statement&gt; =&gt; person = &lt;string&gt; + &lt;str-expression&gt;;</code></td>
</tr>
<tr>
<td>10:</td>
<td><code>&lt;statement&gt; =&gt; person = &lt;name&gt; + &lt;str-expression&gt;;</code></td>
</tr>
<tr>
<td>1:</td>
<td><code>&lt;statement&gt; =&gt; &lt;statement&gt; =&gt; person = firstn + &quot; &quot; + lastn;</code></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
  - (Doesn’t 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 78-79 of *Great Ideas*
  - Also give an example for a complete applet
  - Too long to go through in class – Please Read!

### Decision trees

- **If-Then statements**
  ```java
  if (logical expression) {
    "true" actions
  }
  ```
- **If-Then-Else statements**
  ```java
  if (logical expression) {
    "true" actions
  } else (logical expression 2) {
    "false" actions
  }
  ```
- **Logical expressions** analogous to yes or no questions
  - `true` or `false`
- **Statements that are true**
  - `(5 < 7)`
  - `(100 == 100)`
  - `(100 != 10)`
  - `(10 <= 10)`
- **Statements that are false**
  - `(-2 > -1)`
  - `(10 != 10)`

### A decision tree

- He received the *Physics Price* in 1921.
- Look up the *Peace Prize* in 1991.
- Try the *Medicine Prize* in 1962.
- Would you prefer a humanitarian?
  - Would you like to read about Einstein?
  - Would you like to read about a scientist?
More Java Syntax

- **Assignment statement**
  
  ```java
  variable = expression;
  ```

- **Method invocation**
  - Also called function or procedure
  - Invoking also called “calling” a function
  - Methods can take arguments
  
  ```java
  button.setText("This text is an argument");
  init();
  ```

- **Variable declaration**
  
  ```java
  VariableType variableName;
  Button choice;
  ```

Java Details

- **Java tutorial** [http://java.sun.com/docs/books/tutorial](http://java.sun.com/docs/books/tutorial)
  
  1. Do “Your First Cup of Java” and create your First Applet
  2. Go to “Learning the Java Language” and read “Language Basics”

- **Variable**: an item of data named by an identifier

- **Operators**
  - Arithmetic
  - Relational and conditional
  - Assignment
  - Other

- **Expression**: a series of variables, operators, and method calls that evaluates to a single value

Dealing with numbers

- **Primitive data type**: `int`
  - Does not require a new statement to create
  - Primitive types not classes
  - Must declare
  - Should `initialize` (Java sets to 0)
  - Other primitive types include: boolean, char, double

- **Operations using integers**
  - `+`, `-`, `*`, `/`, `%`
  - Operator Precedence

Some arithmetic details

- **Java adheres to traditional order of operations**
  - `*` and `/` have higher precedence than `+` and `-`
    
    ```java
    int x = 3 + 5 * 6;   int y = (3 + 5) * 6;
    ```

  - Parentheses are free, use them liberally

- **Arithmetic expressions are evaluated left-to-right in the absence of parentheses**
    
    ```java
    int x = 3 * 4 / 6 * 2; int y = (3*4)/(6*2);
    ```

- **There are limits on int and double value, be aware of them.**
Types for Numbers

- The type String is not a built-in type, technically it’s a class

- There are many numerical types in Java. We’ll use two
  - int, represents integers: {...,-2,-1,0,1,2,3,...}
    - Conceptually there are an infinite number of integers, but the range is limited to \([-2^{31}, 2^{31}-1]\) or \([Integer.MIN_VALUE, Integer.MAX_VALUE]\)
    - Alternatives? Why is range limited?
  - double, represents real numbers like \(\pi, \sqrt{2}\)
    - Not represented exactly, so expressions like \(100 \times 0.1\) may yield unexpected results
    - Double precision floating point numbers, another type float exists, but it’s a terrible choice (generates poor results)

GIGO: program as good as its data?

- In calculations involving floating point numbers it’s easy to generate errors because of accumulated approximations:
  - What is \(10^{23} + 1\)?
  - When is \((x + y) + z\) different from \(x + (y + z)\)?

- The type int is severely constrained on 16-bit computers, e.g., running DOS, largest value is \(32,767\) \((2^{15}-1)\)
  - Even on 32-bit machines, how many seconds in a millennium? \(60 \times 60 \times 24 \times 365 \times 1000\), problems?
  - On UNIX machines time is measure in seconds since 1970, problems?
  - What was Y2K all about?

What arithmetic operations exist?

- Syntax and semantics for arithmetic operations
  - Addition, subtraction: + and −, int and double
    - \(23 + 4\) \(x + y\) \(d - 14.0 + 23\)
  - Multiplication: \(*\), int and double
    - \(23 \times 4\) \(y \times 3.0\) \(d \times 23.1 \times 4\)
  - Division: \(/\), different for int and double
    - \(21 / 4\) \(21 / 4.0\) \(x / y\)
  - Modulus: \(\%\), only for int
    - \(21 \% 4\) \(17 \% 2\) \(x \% y\)

- Mixed type expressions are converted to “higher” type
  - Associativity of operators determines left-to-right behavior

- Use parentheses liberally
  - Without () use operator precedence, *, /, % before +, −

Dealing with text

- Strings are a class and not a primitive datatype
- Declaration:
  - String message;
- String Constants
  - "Good Morning World!"
- String Assignment
  - message = "It's Friday";
Manipulating Strings

- Methods for manipulation
  - int length()
  - int indexOf(String st)
  - String substring(int start, int end)
- Getting String Data from user
  - The TextField class has getText() method
  - Use:
    - message = mg.getText();
    - where mg is a TextField and message is a String

Evaluating expressions

- Order of precedence
  - Operators  Associativity  Type
  - ()  left to right  Parentheses
  - * / %  left to right  Multiplicative
  - + -  left to right  Additive
  - < <= > >=  left to right  Relational
  - == !=  left to right  Equalities
  - =  right to left  Assignment

- Automatic type conversion
  - Values of one type are promoted to another compatible type as part of the computation process

- You can convert $T_f$ degrees Fahrenheit to $T_c$ degrees Celsius using the formula:
  $$T_c = \frac{5}{9}(T_f - 32)$$

- Given the following expression:
  $$\text{double } Tc = (Tf - 40.0) \times \left(\frac{5}{9}\right)$$
  If $T_f$ is -40.0 what is $T_c$?
  1. -40.0
  2. 0.0
  3. 40.0
  4. error
  5. unknown

More expressions

- int $n = 1 - 2 * 3 - 4 + 5$;
  - What is $n$?
    1. -4
    2. -2
    3. 0
    4. 2
    5. 4
    6. error

- int $n = 12 + \text{"hello"}$
  1. 0
  2. 12
  3. 17
  4. unknown
  5. error

Repeating code

- Repeating code is bad
- Writing repetitive code is tedious
- Debugging repetitive code is hard
- Avoid repeating code through:
  - Subroutines/methods
  - Loops

- int $x = 8 * (7 - 6 + 5) \% (54 + 3 / 2) - 1$;
  - What is $x$?
    1. -1
    2. 0
    3. 2
    4. 3
    5. error
    6. something else
Loops

- If statements need to repeat, then you probably need a loop
- Describe portion of program as:
  - Repeat
  - Continue until
  - For each value from 1 to n
  - For every object of a set, do something
- We have already used iteration by using the buttons
  - How?

Problems

- We want to:
  - Print out all numbers from 0 up to 100 incrementing by 0.5 each time
  - Sum up the numbers from 1 to 100
  - ...

- New Java syntax
  - New object type TextArea which is basically a big scrolling textbox
  - TextArea is 80 character wide and 20 rows high text box with 20 rows
    ```java
    tArea = new TextArea(20,80);
    ```
  - Add characters to the end of the TextArea using append
    ```java
    tArea.append("Hello\n");
    ```
  - '"\n" is called a newline character which moves the next character to
    the next line

Anatomy of a while loop

- While loops are one way to get rid of repetitive code
- Print out numbers up to 100 by increments of 0.5

```
x = 0.0;
while (x < 100)
{
  x = x + 0.5;
  tArea.append("x = " + x);
  tArea.append("\n");
}
```

Another loop

- Summing the numbers 1 ... 100
  ```java
  int sum = 0;
  int k = 0;
  while (k < 100)
  {
    k = k + 1;
    sum = sum + 1;
  }
  ```
- Other Loop designs
  - Count down
  - Stopping and starting at computed values
  - Data dependent loop
Arrays

- Aggregate data type
- Deal with items of same type
  - Lists
  - numbers
  - words ...
- Analogies
  - Mailboxes in post office
  - CD racks with slots
- Simplifies naming
- Allows use of loops
- Required for many mathematical and statistical problems
- Multiple elements or cells

Using arrays

- subscript or index to access element
  
  ```java
  x[5] = 20;
  foo.setText("Result is " + x[5]);
  ```
- Often used in loops
  
  ```java
  int k = 0; sum = 0;
  while (k < 10)
  {
    k = k + 1;
    sum = sum + name[k];
  }
  ```

Creating Arrays

- Declaration
  ```java
  double weights[];
  ```
- Definition
  ```java
  weights = new double[50];
  ```
- Combine
  ```java
  double weights[] = new double[50];
  ```

```java
int num[] = new int[6];

num[1] = 2; num[5] = 13; ?
```

Arrays & Loops

```java
int k = 2;
while (k<6)
{
  num[k] = k*k;
  k = k+1;
}
```

```
<table>
<thead>
<tr>
<th></th>
<th>21</th>
<th>4</th>
<th>9</th>
<th>16</th>
<th>25</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>21</td>
<td></td>
<td></td>
<td></td>
<td>13</td>
</tr>
</tbody>
</table>
```
Functions/Methods

- **Function example:** distance from point \((x,y)\) to origin
- **Function declaration**
  - Name of the function
  - Type of each argument to the function with a descriptive name for each argument
  - The type of value a function returns

Function calling mechanics

1. The value of each argument are computed
2. The value of each argument is copied into the corresponding formal parameter
3. The statements in the function body are evaluated until a return statement appears
4. The value of the return expression is evaluated
5. The calling program continues, with the returned value substituted in place of the call

Functions can return strings

```java
String WeekDay(int day)
{
    if (0 == day)  {
        return "Sunday";
    }
    else if (1 == day) {
        return "Monday";
    }
    else if (2 == day) {
        return "Tuesday";
    }
    else if (3 == day) {
        return "Wednesday";
    }
    ...
}
```

- **What function call looks like?**
  ```java
  String dayName;
  int dayNum = 4;
  dayName = WeekDay(dayNum);
  result.setText(WeekDay(5));
  int j = WeekDay(0);
  result.setText(WeekDay(2.1));
  String s = WeekDay(22);
  WeekDay(3);
  ```

- **Which is/are ok? Why?**

Think about it

**Puzzle: Toggling Frogs**

- You have 100 light switches, numbered 1-100, and 100 frogs, also numbered 1-100.
- Whenever a frog jumps on a light switch, it toggles a light between on and off. All lights are initially off.
  - frog #1 jumps on every light switch (ie turning them all on).
  - frog #2 jumps on every 2nd light switch, toggling some of them back off.
  ...
  - frog #k jumps on every kth light switch.
- After 100 frogs, which lights are on?

**Game: Don’t be last**

- You and a friend have a stack of 10 coins.
- On each person’s turn, they remove either 1 or 2 coins from the stack.
- The person who removes the last coin wins.
- What is a winning strategy? Should you go first or second?