

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"*

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> + <string-expression>`
 8. `<string-expression>` => `<string>`
 9. `<string>` = *"any sequence of characters"*
 10. `<string>` = `<name>`

Using our Grammar

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

Rule Statement being Generated

```
#
2: <statement> => <name> = <expression>;
1: <statement> => person = <expression>;
6: <statement> => person = <str-expression>;
7: <statement> => person = <str-expression> + <str-expression>;
8: <statement> => person = <string> + <str-expression>;
10: <statement> => person = <name> + <str-expression>;
1: <statement> => person = firstn + <str-expression>;
7: <statement> => person = firstn + <str-expression> + <str-expression>;
8: <statement> => person = firstn + <string> + <str-expression>;
9: <statement> => person = firstn + " " + <str-expression>;
8: <statement> => person = firstn + " " + <string>;
10: <statement> => person = firstn + " " + <name>;
1: <statement> => <statement> => person = firstn + " " + lastn;
```

CompSci 001

5.5

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!

CompSci 001

5.6

Decision trees

- If-Then statements

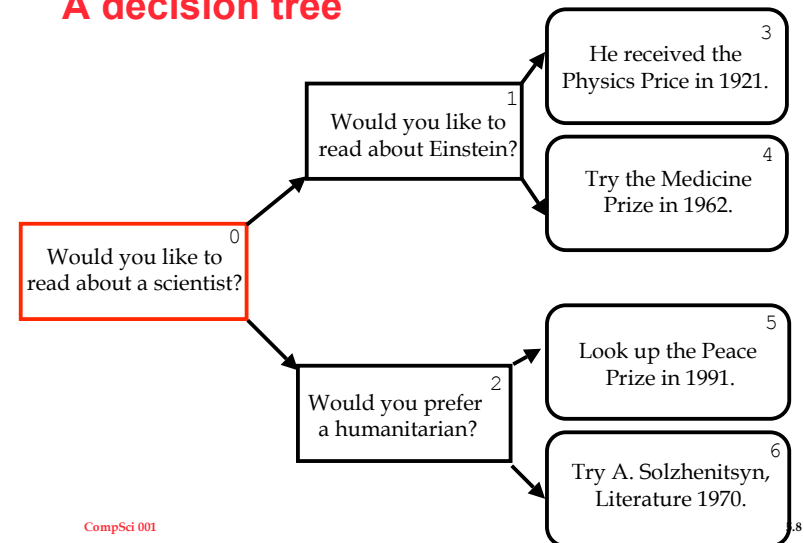
```
if (logical expression)
{
    "true" actions
}
```
- If-Then-Else statements

```
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
 - $(-2 > -1)$
 - $(10 != 10)$

CompSci 001

5.7

A decision tree



CompSci 001

5.8

More Java Syntax

- **Assignment statement**
`variable = expression;`
- **Method invocation**
 - Also called function or procedure
 - Invoking also called “calling” a function
 - Methods can take *arguments*

```
button.setText("This text is an argument");  
init();
```
- **Variable declaration**
`VariableType variableName;`
`Button choice;`

Java Details

- **Java tutorial** <http://java.sun.com/docs/books/tutorial>
 - Do “Your First Cup of Java” and create your First Applet
 - 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 `-`

```
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**

```
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: $\{\dots-3,-2,-1,0,1,2,3,\dots\}$
 - 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 $100 * 0.1$ 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? $60 * 60 * 24 * 365 * 1000$, 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`
 $23 * 4$ $y * 3.0$ $d * 23.1 * 4$
 - 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	Relationals
== !=	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:

$$T_c = (5/9)*(T_f - 32)$$

- **Given the following expression:**

```
double Tc = (Tf - 40.0) * (5/9)
```

If Tf is -40.0 what is Tc?

☒ -40.0

☒ 0.0

☒ 40.0

☒ error

☒ 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 + "hello"
```
1. 0
  2. 12
  3. 17
  4. unknown
  5. error

```
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

## Repeating code

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

## Loops

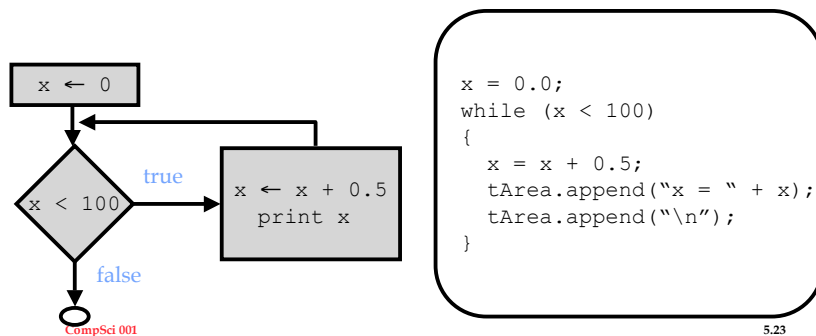
- If statements need to repeat, then you probably need a loop
- Describe portion of program as:
  - Repeat
  - Continue until
  - For each value from 1 to n
  - For every object of a set, do something
- We have already used iteration by using the buttons
  - How?

## 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
  - `tArea` is 80 character wide and 20 rows high text box with 20 rows  
`TextArea tArea = new TextArea(20,80);`
  - Add characters to the end of the `TextArea` using `append`  
`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



## Another loop

- Summing the numbers 1 ... 100

```
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
  - `x[5] = 20;`
  - `foo.setText("Result is " + x[5]);`
- Often used in loops

```
int k = 0; sum = 0;
while (k < 10)
{
 k = k + 1;
 sum = sum + name[k];
}
```

## Creating Arrays

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

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

|                          |                           |   |
|--------------------------|---------------------------|---|
| <code>num[1] = 2;</code> | <code>num[5] = 13;</code> | ? |
|--------------------------|---------------------------|---|

|   |    |   |   |   |    |
|---|----|---|---|---|----|
| ? | 21 | ? | ? | ? | 13 |
|---|----|---|---|---|----|

## Arrays & Loops

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

|   |    |   |   |    |    |
|---|----|---|---|----|----|
| ? | 21 | 4 | 9 | 16 | 25 |
|---|----|---|---|----|----|

## 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

- ☞ The value of each argument are computed
- ☞ The value of each argument is copied into the corresponding *formal parameter*
- ☞ The statements in the function body are evaluated until a return statement appears
- ☞ The value of the return expression is evaluated
- ☞ The calling program continues, with the returned value substituted in place of the call

## Functions can return strings

```
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";
 }
 ...
}
```

- Shorter (code) alternatives?
  - Is shorter better?

### ● What function call looks like?

```
String dayName;
int dayNum = 4;
dayName = WeekDay(dayNum);
```

### ● Which is/are ok? Why?

```
result.setText(WeekDay(5));

int j = WeekDay(0);

result.setText(WeekDay(2.1));

String s = WeekDay(22);

WeekDay(3);
```

## 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?