Values and Variables

Georgia Tech

Values


Unbeatable
We'll match the price of any local competitor's printed ad for an identical product. See manager for restrictions.
Prices
$000 00
GEORGIA

Computing

Computing is any purposeful activity that marries the representation of some dynamic domain with the representation of some dynamic machine that provides theoretical, empirical or practical understanding of that domain or that machine.
– Isbell, et. al., (Re)Defining Computing Curricula by (Re)Defining Computing, SIGCSE Bulletin, Volume 41, Number 4, December 2009

Georgia Tech

Models, Languages, Machines

Computing is fundamentally a modelling activity.

- A model is a representation of some information, physical reality, or a virtual entity in a manner that can then be interpreted, manipulated, and transformed.
- A language is a means of representation.
  - A language enables reasoning and manipulation of the model.
- A computational machine allows us to execute our models.

Georgia Tech

Languages and Computation

Every powerful language has three mechanisms for combining simple ideas to form more complex ideas:(SICP 1.1)

- primitive expressions, which represent the simplest entities the language is concerned with,
- means of combination, by which compound elements are built from simpler ones, and
- means of abstraction, by which compound elements can be named and manipulated as units.

In this lecture we'll focus on primitive expressions and basic abstraction.

Georgia Tech

A Model of Course Average

```
public class CourseAverage {
    public static void main(String[] args) {
        double hwAvg = 74.2;
        double examAvg = (81 + 91 + 93 + 89) / 4;
        double courseAvg = (.2 * hwAvg) + (.8 * examAvg);
        System.out.println("Course Average: " + courseAvg);
    }
}
```

- Values 74.2, 81, 93, 95, 89 are primitive expressions
- hwAvg, examAvg are abstractions which name values
- Value assigned to courseAvg computed by combining primitive values
- Our model of course average is expressed in a language that allows us to reason about, manipulate, and run the model on a machine

Georgia Tech

Identifiers

An identifier is a string of characters used as a name for a class, method or variable.

- Can contain letters, digits, and the underscore symbol and may not start with a digit.
- Case-sensitive: `this` is not the same as `This`.

Reserved and Keywords

Java reserves some identifers.

- Keywords are used by Java, like `class`, `public`, `if` and so on.
- Reserved words aren't currently used, like `goto` and `const`
- You can't use reserved or keywords for your own identifiers.
- Full list is here: `http://docs.oracle.com/javase/tutorial/java/nutsandbolts/_keywords.html`

Georgia Tech

Variable Declarations

A variable is an identifier that names a value. A variable has:

- a type, and
- a storage location for the variable's value.

Variables must be declared before use. Here's a declaration:

```
float twoThirds;
```

- `float` is the variable's type, `twoThirds` is the variable name

The value of `twoThirds` after the declaration statement above depends on whether `twoThirds` is an instance variable or a local variable. More on that later.

Assignment Statements

= is the assignment operator.

- The identifier on the left side of a = must be a variable identifier (an lvalue)
- The right side of the = must be an expression
- An expression has a value
- 2 + 3 is an expression. It has the value 5
- A variable is also an expression. It has whatever value it was last assigned

Declarations and Assginments

Here's a declaration followed by an assignment:

```
float twoThirds;
twoThirds = 2/3;
```

Usually combine declaration and assignment into an initialization statement:

```
float twoThirds = 2/3;
```

Type Compatibility

Legal assignments:

```
int x = 1;                    // int literal
float y = 3.14159f;           // float literal
double z = 3.1415;            // double literal
boolean thisSentence = false; // boolean literal
String goedel = "incomplete"; // String literal
```

Illegal assignments:

```
int x = 1.0;              // 1.0 is a double value
float y = 3.14159;        // 3.14159 is a double value
boolean thisSentence = 1; // 1 is an int value
```

Java is Statically Typed

Core concepts:

- every value has a type
- every variable has a type
- assignment of values to variables must be type compatible at compile-time

Georgia Tech

Syntax and Semantics

- Syntax - the form to which source code must conform
- Semantics - the meaning of the code, i.e., what it does

```
public class Expressions {
    public static void main(String[] args) {
        float twoThirds = 2/3;
        System.out.println(twoThirds);
    }
}
```

- Code inside `main` conforms to Java syntax: a sequence of statements, each ending with a semicolon
- Meaning of the program, its semantics, is: initialize the variable `twoThirds` with the value `.667` and print it to console (or so we think . . .)

Compile and run Expressions.java and see what it prints.

Type Conversion

When we run `Expressions.java` we get this:

```
$ javac Expressions.java
$ java Expressions
0.0
```

What happened?

- `twoThirds` is a `float`, so it can hold fractional values.
- But 2 and 3 are literal representations of `int` values.
- 2/3 performed integer division, resulting in a value of `0`.
- `float` variables can hold integer values, so Java performed automatic conversion to `float` upon assignment to `twoThirds` – `0.0`.

Type Conversion Rules

The previous example showed an implicit widening conversion

- `float` is wider than `int` because all intergers are also floating point values.
- Java will perform widening conversions automically because no precision is lost.
- To perform a narrowing conversion, you must explicitly cast the value.

Georgia Tech

Type Conversion Examples

This won't compile because an `int` can't hold a fractional value; converting may cause a loss of precision (note that we're using `double` values by including a decimal part):

```
int threeFourths = 3.0/4.0;
```

You have to cast the `double` to an int:

```
int threeFourths = (int) (3.0/4.0);
```

What happens if we leave off the parentheses around `(3.0/4.0)`?

Georgia Tech

Exercise: What do I need on the final?

Write a program called `FinalTarget` that calculates the score you need on the final exam in order to acheive a given target course average.

Georgia Tech

Integral Primitive Types

- `byte`: 8-bit signed two's complement integer. Min value of -128 and max value of 127 (inclusive).
- `short`: 16-bit signed two's complement integer. Min value of -32,768 and max value of 32,767 (inclusive).
- `int`: 32-bit signed two's complement integer. Min value of -2,147,483,648 and max value of 2,147,483,647 (inclusive). `int` generally default choice for integer values
- `long`: 64-bit signed two's complement integer. Min value of -9,223,372,036,854,775,808 and max value of 9,223,372,036,854,775,807 (inclusive).

Georgia Tech

Floating Point Primitive Types

- `float`: single-precision 32-bit IEEE 754 floating point.
- `double`: double-precision 64-bit IEEE 754 floating point. Its range of values is beyond the scope of this discussion, but is specified in the Floating-Point Types, Formats, and Values section of the Java Language Specification. For decimal values, `double` is generally the default choice.

Floating point types should never be used for precise values, such as currency. For that, you will need to use the java.math.BigDecimal class instead. Numbers and Strings covers BigDecimal and other useful classes provided by the Java platform.

Georgia Tech

boolean and char

- `boolean`: The boolean data type has only two possible values: true and false. Use this data type for simple flags that track true/false conditions. This data type represents one bit of information, but its "size" isn't something that's precisely defined.
- `char`: The char data type is a single 16-bit Unicode character. It has a minimum value of `'\u0000'` (or 0) and a maximum value of `'\uffff'` (or 65,535 inclusive).

Georgia Tech

Shortcut Assignment Statements

Like C and C++, Java allows shortcut assignments:

- A binary operation that updates the value of a variable:

```
x += 2; // same as x = x + 2;
```

- Pre- and post-increment and decrement:

```
x++; // post-increment; same as x = x + 1;
--x; // pre-decrement; same as x = x - 1;
```

Be Careful with Shorcut Assignments

Pre-increment: variable incremented before used in expression

```
int x = 1;
int y = ++x;
// x == 2, y == 2;
```

Post-increment: variable incremented after used in expression

```
int x = 1;
int y = x++;
// x == 2, y == 1;
```

What's the value of `x` after `x = x++`?

Georgia Tech

Precedence and Associativity

If an expression contains no parentheses, Java evaluates expressions according to precedence in a three-step process:

- Associate operands with operators, starting with highest-precedence operators. This step effectively parenthesizes expression
- Evaluate subexpressions in left to right order (possibly in multiple sweeps if deeply nested)
- Evaluate outer "top-level" operation once all subexpressions have been evaluated

Evaluation Example

`6 + 7 * 2 - 12` is evaluated in the following steps:

1. Associate highest precedence operators with operands:

```
6 + (7 * 2) - 12
```

1. '+' and '-' have same precedence, associate left to right:

```
(6 + (7 * 2)) - 12
((6 + (7 * 2)) - 12)
```

1. Evaluate subexpressions like you learned in high school:

```
((6 + 14) - 12)
(20 - 12)
8
```

Side-Effects in Expressions

Consider the following expression evaluation for `n = 2`:

```
((result = (++n)) + (other = (2*(++n))))
((result = 3) + (other = (2*(++n))))
(3 + (other = (2*(++n))))
(3 + (other = (2*4))) // n was 3 from the first pre-increment
(3 + (other = 8))
(3 + 8)
11
```

- An assignment statement has the value that was assigned
- Pre-increment (`++n`) means `n` is incremented before it's used in the expression in which it appears
- Three side-effects: result = 3, other = 8, and n = 4

Don't write code like this!

String Values

A String is a sequence of characters.

- String literals are enclosed in double quotes

```
"foo"
```

- String variables

```
String foo = "foo";
```

Note that, unlike the other types we've seen, String is capitalized. String is a class.

String Concatenation

The + operator is overloaded to mean concatenation for `String` objects.

- Strings can be concatenated

```
String bam = foo + bar + baz; // Now bam is "foobarbaz"
```

- Primitive types can also be concatenated with `Strings`. The primitive is converted to a String

```
String s = bam + 42; // s is "foobarbaz42"
String t = 42 + bam; // t is "42foobarbaz"
```

Note that + is only overloaded for ~String~s.

The String Class

`String` acts like primitive thanks to syntactic sugar provided by the Java compiler, but it is defined as a class in the Java standard library

- See `http://docs.oracle.com/javase/8/docs/api/java/lang/String.html` for details.
- Methods on objects are invoked on the object using the `.` operator

```
String empty = "";
int len = empty.length(); // len is 0
```

- Look up the methods `length`, `indexOf`, `substring`, and `compareTo`, and `trim`
- Because `String` s are objects, beware of null references:

```
String boom = null;
int aPosInBoom = boom.indexOf("a");
```

Play with Strings.java

Closing Thoughts

Every powerful language has three mechanisms for combining simple ideas to form more complex ideas:

- primitive expressions, which represent the simplest entities the language is concerned with,
  - Values are the atoms of programs
- means of combination, by which compound elements are built from simpler ones, and
  - Programs combine and manipulate values
- means of abstraction, by which compound elements can be named and manipulated as units.
  - Variables are the simplest form of abstraction - naming values

Georgia Tech