Arithmetic Operators
Type Conversion
Constants

Lecture 3

Some slides borrowed from Kurt Eiselt, Tamara Munzner, and Steve Wolfman. Some learning goals from Beth Simon.

Readings

Your textbook is *Big Java* (3rd Ed).
This Week’s Reading: Ch 2.1-2.5, Ch 4.1-4.2.
Next Week: Ch 2.6-2.10, Finish Ch 4

Labs and Tutorials

Labs and tutorials start this week!

Lab #2 will be up soon. It has a pre-lab!

You MUST be enrolled in a lab section.

If the lab sections you need are full, go to ICICS Room 201 and see a Computer Science undergrad advisor!

Midterms – Save the Dates!

- Midterm #1 is 5:30-6:30pm on February 10 (Tuesday) in Woodward IRC 2
- Midterm #2 is 6-7pm on March 11 (Wednesday) in Woodward IRC 2

You must notify me well in advance (I will specify deadlines as the dates approach) if you have an unavoidable conflict with a midterm.

PeerWise Study

- PeerWise is a system designed to facilitate students helping each other learn.
- This term’s 111 sections are being studied to evaluate how effective PeerWise really is.
- You (Section 206) have been randomly selected as the “treatment group”.
  - You will get access to the PeerWise system. (Details later.)
  - You will be required (3% of course grade) to complete some activities on PeerWise.
- You have the right to opt out.
  - You must email me by noon, January 21, to opt out. Email me your full name, student ID number, and say that you opt out.
  - If you opt out, you will be denied access to PeerWise, you will not be required to do the activities, and you will be marked as if you were in Section 205.

Learning Goals

By the end of the next several lectures you will be able to...

- Apply with basic competence simple programming constructs such as sequential execution, variable typing and declaration, naming, algebraic operations, operation precedence.
- Create programs which translate explicit English problem statements (an algorithm) into short series of sequential Java instructions.
Learning Goals

By the end of class today you will be able to:

- Write Java expressions for mathematical computations, using different operators, with correct precedence and associativity.
- Explain how the same operator symbol can produce different results on different types.
- Describe when Java will automatically convert from one primitive type to another.
- Force Java to make a type conversion.
- Declare constants in your program, and explain why you would do so.

Last Time...

- We learned about the primitive data types.
- We learned how to write assignment statements and trace execution through several statements.
- Now, let’s look at the computation you can do with primitive data types…

Review: Data Types

- For every variable, we have to declare a data type
- Java language provides eight primitive data types
  - i.e. simple, fundamental
- For more complicated things, can use data types
  - created by others provided to us through the Java libraries
  - that we invent
    - More soon - today, let’s stay with the primitives

Review: Assignment Statements

- Assignment statement assigns value to variable
- Assignment statement is
  - identifier
    - followed by assignment operator (=)
    - followed by expression
    - followed by semicolon (;)
    
    ```
    b = 3;
    c = 8;
    a = b + c;
    weekly_pay = pay_rate * hours_worked;
    ```

- Note that = is not a test for equality!

### Primitive Data Types: Numbers

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>1 byte</td>
<td>-128</td>
<td>127</td>
</tr>
<tr>
<td>short</td>
<td>2 bytes</td>
<td>-32,768</td>
<td>32,767</td>
</tr>
<tr>
<td>int</td>
<td>4 bytes</td>
<td>-2,147,483,648</td>
<td>2,147,483,647</td>
</tr>
<tr>
<td>float</td>
<td>4 bytes</td>
<td>approx -3.4E38 (7 sig.digits)</td>
<td>approx 3.4E38 (7 sig.digits)</td>
</tr>
<tr>
<td>double</td>
<td>8 bytes</td>
<td>approx -1.7E308 (15 sig. digits)</td>
<td>approx 1.7E308 (15 sig. digits)</td>
</tr>
</tbody>
</table>

- Six primitives for numbers
  - fixed size, so finite capacity
  - integer vs. floating point

- Primary primitives are `int` and `double`
  - Just worry about those for now
  - Don’t need to memorize exact limits, but know roughly what the limits are.
**Data Types: Int and Double**

- **int**
  - integer
  - 4 bytes, about -2 billion to 2 billion
- **double**
  - real number
  - (double-precision floating point)
  - 8 bytes, 15 sig figs, humongous range
- (Number systems briefly explained in Appendix K)

**Experiments with Overflows**

**Rekap: Assignment Statements**

- **Assignment statement** assigns value to variable
- Assignment statement is
  - identifier
  - followed by **assignment operator** (=)
  - followed by **expression**
  - followed by semicolon (;)

```plaintext
b = 3;
c = 8;
a = b + c;
weekly_pay = pay_rate * hours_worked;
```

- Note that = is not a test for equality!

**Expressions**

- **expression** is combination of
  - one or more operators and operands
  - operator examples: +, *, /, ...
  - operand examples: numbers, variables, ...
  - usually performs a calculation
    - don't have to be arithmetic but often are
  - examples
    - `salary + bonus`
    - `3.14 * diameter`

**Arithmetic Operators**

- **+** addition
  - Works on int, double, byte, short, long, ...
- **-** subtraction
  - Works on all numeric types, too
- **** multiplication
  - Didn't have times sign on keyboard
  - Works on all numeric types, too

**Expressions**

- **expression** is combination of
  - one or more operators and operands
  - operator examples: +, *, /, ...
  - operand examples: numbers, **variables**, ...
  - usually performs a calculation
    - don't have to be arithmetic but often are
  - examples
    - `3`
    - `7 + 2`
    - `7 + 2 * 5`
    - `(7 + 2) * 5`
More Arithmetic Operators

- / division
- Integer division on integer types!
- Example: 13 / 5 results in 2
- Just like before you learned fractions
- Normal division on double and float
- % remainder (aka “mod”)
- Only works on integer types
- Example: 13 % 5 results in 3

Operator Overloading

- Hmm… the same symbol / can do different things for ints and doubles:
  - 13/5 results in 2 (the type is int)
  - 13.0/5.0 results in 2.6 (the type is double)
- Similar for other operators, e.g., +
  - 13+5 is 18 (18 is an int)
  - 13.0+5.0 is 18.0 (18.0 is a double)
  - “13”+“5” is “135” (“135” is a String)

Operator Precedence

- What does this expression evaluate to?
  7 + 2 * 5

Operator Precedence

- What does this expression evaluate to?
  7 + 2 * 5
- Multiplication has higher operator precedence than addition (just like in algebra)

<table>
<thead>
<tr>
<th>precedence</th>
<th>operator</th>
<th>operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 higher</td>
<td>+ -</td>
<td>unary plus and minus</td>
</tr>
<tr>
<td>2</td>
<td>* / %</td>
<td>multiply, divide, remainder</td>
</tr>
<tr>
<td>3 lower</td>
<td>+ -</td>
<td>add, subtract</td>
</tr>
</tbody>
</table>

Associativity

- What about this?
  7 – 5 – 3

Operator Precedence

- What does this expression evaluate to?
  7 + 2 * 5
- Multiplication has higher operator precedence than addition (just like in algebra)

<table>
<thead>
<tr>
<th>precedence</th>
<th>operator</th>
<th>operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 higher</td>
<td>+ -</td>
<td>unary plus and minus</td>
</tr>
<tr>
<td>2</td>
<td>* / %</td>
<td>multiply, divide, remainder</td>
</tr>
<tr>
<td>3 lower</td>
<td>+ -</td>
<td>add, subtract</td>
</tr>
</tbody>
</table>
- Use parentheses to change precedence order or just clarify intent
  (7 + 2) * 5  7 + (2 * 5)
Associativity
- What about this?
  
  \[ 7 - 5 - 3 \]
  
  The result is -1
- (7 – 5) – 3, not 7 – (5 – 3)
- Arithmetic operators of same precedence are **left associative**
- Matters for some operators; doesn’t for others
- Use parentheses to be clear!
- (Operators and precedence in Appendix E)

An old mystery...

```java
public class Assoc {
    public static void main (String[] args) {
        int a;
        int b = 3;
        int c = 5;
        a = b + c;
        System.out.println("The answer is "+ a);
        System.out.println("The answer is "+ (b + c));
        System.out.println(b + c + " is the answer.");
        System.out.println(b + (c + " is the answer."));
    }
}
```

Shorthand: Variable Declaration and Assignment
- **Variable declaration** reserves a chunk of memory and gives it a name.
- **Assignment** sticks a value into the variable.
- Can do both in one statement:
  - `typeName variableName;`
  - `variableName = value;`
- `typeName variableName = value;` can declare and assign in one step

Converting Between Types
- Which of these should be legal?
  - `int shoes = 2; // yes`
  - `double socks = 1.75; // yes`
  - `double socks = 1; // yes`
  - `int shoes = 1.5; // no`
- Integers are subset of reals
  - but reals are not subset of integers
- Java automatically converts `int` to `double` when needed (or smaller size to larger size)

Converting Between Types
- Which of these are legal?
  - `int shoes = 2; // yes`
  - `double socks = 1.75; // yes`
  - `double socks = 1; // yes`
  - `int shoes = 1.5; // no`
- Integers are subset of reals
  - but reals are not subset of integers
- Java automatically converts `int` to `double` when needed (or smaller size to larger size)

Casting
- **Casting**: force Java to convert from one type to another, **even with information loss**
- Converting from real to integer
  - `int shoes = (int) 1.5;`
- Truncation: fractional part thrown away
  - `int shoes = (int) 1.75;`
  - `int shoes = (int) 1.25;`
- Rounding: must be done explicitly
  - `shoes = Math.round(1.99);`
Converting Between Types

```java
public class Feet
{
    public static void main (String[] args)
    {
        int shoes = 2;
        int socks = (int) 1.75;
        System.out.println("shoes = " + shoes + " socks = " + socks);
        int toes = Math.round(1.99);
        System.out.println("toes = " + toes);
    }
}
```

What's wrong? Let's try it...

---

**Primary primitives are int and double**
- three other integer types
- one other real type

---

Converting Between Types

```java
public class Feet2
{
    public static void main (String[] args)
    {
        int shoes = 2;
        int socks = (int) 1.75;
        System.out.println("shoes = " + shoes + " socks = " + socks);
        long toes = Math.round(1.99);
        System.out.println("toes = " + toes);
    }
}
```

Or Tell Java with a Type Cast

```java
public class Feet2
{
    public static void main (String[] args)
    {
        int shoes = 2;
        int socks = (int) 1.75;
        System.out.println("shoes = " + shoes + " socks = " + socks);
        long toes = Math.round(1.99);
        System.out.println("toes = " + toes);
    }
}
```

---

Questions?

---

**Motivation for Constants**

- Suppose we want to write a program to compute the distance in miles to various stars, for which we know the distance in light-years.
What Changes, What Doesn’t?

```java
// Vroom.java Author: Tamara
// Playing with constants
//**********************************************************************/
public class Vroom
{
    public static void main (String[] args)
    {
        double lightYears, milesAway;
        lightYears = 4.35; // to Alpha Centauri
        milesAway = lightYears * 186000 * 60 * 60 * 24 * 365;
        System.out.println("lightYears: " + lightYears + " milesAway " + milesAway);
        lightYears = 68; // to Aldebaran
        milesAway = lightYears * 186000 * 60 * 60 * 24 * 365;
        System.out.println("lightYears: " + lightYears + " milesAway " + milesAway);
    }
}
```

Constants

- Things that do not vary
  - unlike variables
  - will never change
- Syntax:
  - `final typeName variableName;`
  - `final typeName variableName = value;`
- Constant names in all upper case
  - Java convention, not compiler/syntax requirement

Programming With Constants

```java
public static void main (String[] args)
{
    double lightYears, milesAway;
    final int LIGHTSPEED = 186000;
    final int SECONDS_PER_YEAR = 60 * 60 * 24 * 365;
    lightYears = 4.35; // to Alpha Centauri
    milesAway = lightYears * LIGHTSPEED * SECONDS_PER_YEAR;
    System.out.println("lightYears: " + lightYears + " miles " + milesAway);
    lightYears = 68; // to Aldebaran
    milesAway = lightYears * LIGHTSPEED * SECONDS_PER_YEAR;
    System.out.println("lightYears: " + lightYears + " miles " + milesAway);
}
```

Avoiding “Magic Numbers”

- "magic numbers": numeric constants directly in code
  - almost always bad idea!
    - hard to understand code
    - hard to make changes
    - typos possible
  - use constants instead

Questions?

```java
public static void main (String[] args)
{
    double lightYears, milesAway;
    final int LIGHTSPEED = 186000;
    final int SECONDS_PER_YEAR = 60 * 60 * 24 * 365;
    final double ALPHACENT_DIST = 4.35; // to Alpha Centauri
    final double ALDEBARAN_DIST = 68; // to Aldebaran
    lightYears = ALPHACENT_DIST;
    milesAway = lightYears * LIGHTSPEED * SECONDS_PER_YEAR;
    System.out.println("lightYears: " + lightYears + " miles " + milesAway);
    lightYears = ALDEBARAN_DIST;
    milesAway = lightYears * LIGHTSPEED * SECONDS_PER_YEAR;
    System.out.println("lightYears: " + lightYears + " miles " + milesAway);
}
```