Recap: 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>approx -128</td>
<td>approx 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>

- 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.

Recap: 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 L)

Recap: 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

Recap: 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
Recap: 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)!

Recap: Operator Precedence

- What does this expression evaluate to?
  \[ 7 + 2 \times 5 \]
- 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
  \[(7 + 2) \times 5\]  \[7 + (2 \times 5)\]

Recap: 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 F)

Today’s Objectives

- Understand type conversions on Java numeric types
- Learn how and why to declare and use constants
- Understand syntax and semantics
- Learn basic principles of program development and debugging

Converting Between Types

- Which of these are legal?
  - int shoes = 2;
  - double socks = 1.75;
  - double socks = 1;
  - int shoes = 1.5;

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

- **Feet.java**  Author: Tamara
- What type of things can be put on feet?
- 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?

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</td>
<td>approx 1.7E308</td>
</tr>
</tbody>
</table>

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

Converting Between Types

- **Feet2.java**  Author: Tamara
- What type of things can be put on feet?
- public class Feet2
  - public static void main (String[] args)
    - int shoes = 2;
    - int socks = (int) 1.75;
    - long toes = Math.round(1.99);
    - System.out.println("toes = " + toes);
  
- Or Tell Java with a Type Cast

- **Feet2.java**  Author: Tamara
- What type of things can be put on feet?
- public class Feet2
  - public static void main (String[] args)
    - int shoes = 2;
    - int socks = (int) 1.75;
    - System.out.println("shoes = " + shoes + " socks = " + socks);
    - System.out.println("shoes = " + shoes + " socks = " + socks);
    - System.out.println("shoes = " + shoes + " socks = " + socks);
    - System.out.println("toes = " + toes);

- What Changes, What Doesn't?

- **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);

- Primitive Data Types: Numbers

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

- Converting Between Types

  - **Feet3.java**  Author: Tamara
  - What type of things can be put on feet?
  - public class Feet3
    - public static void main (String[] args)
      - int shoes = 2;
      - int socks = (int) 1.75;
      - System.out.println("shoes = " + shoes + " socks = " + socks);
      - System.out.println("shoes = " + shoes + " socks = " + socks);
      - System.out.println("shoes = " + shoes + " socks = " + socks);
      - System.out.println("toes = " + toes);

- Or Tell Java with a Type Cast

  - **Feet3.java**  Author: Tamara
  - What type of things can be put on feet?
  - public class Feet3
    - public static void main (String[] args)
      - int shoes = 2;
      - int socks = (int) 1.75;
      - System.out.println("shoes = " + shoes + " socks = " + socks);
      - System.out.println("shoes = " + shoes + " socks = " + socks);
      - System.out.println("shoes = " + shoes + " socks = " + socks);
      - System.out.println("toes = " + toes);
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

Change of Gears...

Program Development

- Use an editor to create your Java program
  - often called source code
  - code used interchangeably with program or instructions in the computer world
- Another program, a compiler or an interpreter, translates source code into target language or object code, which is often machine language
- Finally, your computer can execute object code
Syntax
- Rules to dictate how statements are constructed.
  - Example: open bracket needs matching close bracket
  - If program is not syntactically correct, cannot be translated by compiler
  - Different than humans dealing with natural languages like English. Consider statement with incorrect syntax (grammar)
    - for weeks, rained in Vancouver it hasn’t
    - we still have pretty good shot at figuring out meaning

Semantics
- What will happen when statement is executed
- Programming languages have well-defined semantics, no ambiguity
- Different than natural languages like English. Consider statement:
  - Mary counted on her computer.
- How could we interpret this?
- Programming languages cannot allow for such ambiguities or computer would not know which interpretation to execute

Errors / “Bugs”
- Computers follows our instructions exactly
- If program produces the wrong result it’s the programmer’s fault
  - unless the user inputs incorrect data
  - then cannot expect program to output correct results: “Garbage in, garbage out” (GIGO)
- Debugging: process of finding and correcting errors
  - Unfortunately can be very time consuming!

Origin of the Term “Bug”
- A page from Grace Murray Hopper’s 1945 lab notebook:

Errors
- Error at compile time (during translation)
  - you did not follow syntax rules that say how Java elements must be combined to form valid Java statements
- Error at run time (during execution)
  - Source code compiles
    - Syntactically (structurally) correct
  - But program tried something computers cannot do
    - like divide a number by zero.
  - Typically program will crash: halt prematurely
Logical error
- Source code compiles
- Object code runs
- But program may still produce incorrect results because logic of your program is incorrect
  - Typically hardest problems to find