COMP 110/L Lecture 4

Kyle Dewey

Outline

- New types: `long` and `double`
  - Reading in with `Scanner`
  - Performing operations on them
  - How they interact with each other and other types
- Exponentiation with `Math.pow()`

New Type: `long`

Revisit: `AddTwo.java`

-If we try this with a really big number (e.g., 9876543210), it will outright crash
-If we try it with two still pretty big numbers (e.g., 1234567890 and 1234567890), it will produce incorrect results, even getting a negative number out of two positive numbers

Fundamental Problem

- `int` stores integers in the following range: -2^{31} to $(2^{31} - 1)$
- Numbers out of this range won't work right

-This range is around +/- 2 billion.
-2 billion sounds like a lot, and it's big enough for most things, but there are 7 billion people on the planet

`long` for Bigger Integers

- `long` works like `int`, but its range is exponentially larger
    - -2^{63} to $(2^{63} - 1)$

Working with `long`

Declaring a `long` variable

```
long myLong;
```

Working with `long`

Declaring a `long` variable

```
long myLong;
```

Reading in a `long` with `Scanner`

```
Scanner in = new Scanner(System.in);
long myLong = in.nextLong();
```

-Instead of declaring an int variable, we can declare a long variable
-We can read in a long using nextLong(), as opposed to nextInt()

Example:
`LongAddTwo.java`

Specifying `long`

- By default, if you write a number, Java assumes it's an `int`
- If you follow it with an `l` (the letter ell), Java will treat it as a long

Specifying `long`

- By default, if you write a number, Java assumes it's an `int`
- If you follow it with an `l` (the letter ell), Java will treat it as a long

---

```
14  // int
```

Specifying `long`

- By default, if you write a number, Java assumes it's an `int`
- If you follow it with an `l` (the letter ell), Java will treat it as a long

```
14  // int
```

```
14l  // long (that's an ell)
```

Interactions with `long`

String concatenation works like it does with `int`

Interactions with `long`

String concatenation works like it does with `int`

```
“my string” + 14l
```

Interactions with `long`

String concatenation works like it does with `int`

```
“my string” + 14l
“my string14”
```

Interactions with `long`

String concatenation works like it does with `int`

```
“my string” + 14l
“my string14”
```

```
13l + “other string”
```

Interactions with `long`

String concatenation works like it does with `int`

```
“my string” + 14l
“my string14”
```

```
13l + “other string”
“13other string”
```

Interactions with `long`

Addition works like it does with `int`

Interactions with `long`

Addition works like it does with `int`

```
5l + 4l
```

Interactions with `long`

Addition works like it does with `int`

```
51 + 41
  91
```

Interactions Between `long` and `int`

Values *coerce* into `long`

-Intuition: long is bigger, so it wins

Interactions Between `long` and `int`

Values *coerce* into `long`

```
4l + 2
```

–Intuition: long is bigger, so it wins

Interactions Between `long` and `int`

Values *coerce* into `long`

```
4l + 2
6l
```

–Intuition: long is bigger, so it wins

Interactions Between `long` and `int`

Values *coerce* into `long`

```
4l + 2
6l
```

```
3 + 6l
```

–Intuition: long is bigger, so it wins

Interactions Between `long` and `int`

Values *coerce* into `long`

```
4l + 2
6l
```

```
3 + 6l
9l
```

–Intuition: long is bigger, so it wins

New Type: `double`

Revisit: `AddTwo.java`

-If we try to put in a floating-point value, it outright crashes
-We want support for floating-point values (these are really useful!)

`double` for Floating-Point

- `double` stores floating-point values
- `float` also stores floating-point values, but it’s half the size of `double`
    - Narrower range, less precise

Working with `double`

Declaring a `double` variable

```
double myDouble;
```

Working with `double`

Declaring a `double` variable

```
double myDouble;
```

Reading in a `double` with `Scanner`

```
Scanner in = new Scanner(System.in);
double myDouble = in.nextDouble();
```

Example:
`DoubleAddTwo.java`

Specifying `double`

If the number contains a decimal point,
Java treats it as a `double`

Specifying `double`

If the number contains a decimal point, Java treats it as a `double`

```
4.5  // double
```

Specifying `double`

If the number contains a decimal point, Java treats it as a `double`

```
4.5  // double
1.0  // double
```

Specifying `double`

If the number contains a decimal point, Java treats it as a `double`

```
4.5  // double
1.0  // double
0.2  // double
```

Interactions with `double`

String concatenation works like it does with `int`

Interactions with `double`

String concatenation works like it does with `int`

```
“my string” + 0.5
```

Interactions with `double`

String concatenation works like it does with `int`

```
“my string” + 0.5
 “my string0.5”
```

Interactions with `double`

String concatenation works like it does with `int`

```
"my string" + 0.5
```

```
"my string0.5"
```

```
0.2 + "other string"
```

Interactions with `double`

String concatenation works like it does with `int`

```
"my string" + 0.5
"my string0.5"
```

```
0.2 + "other string"
"0.2other string"
```

Interactions with `double`

Addition works like it does with `int`

Interactions with `double`

Addition works like it does with `int`

```
5.0 + 4.2
```

Interactions with `double`

Addition works like it does with `int`

```
5.0 + 4.2
9.2
```

Interactions Between `double` and `int`

Values *coerce* into `double`

Interactions Between `double` and `int`

Values *coerce* into `double`

```
0.5 + 2
```

Interactions Between `double` and `int`

Values *coerce* into `double`

```
0.5 + 2
2.5
```

Interactions Between `double` and `int`

Values *coerce* into `double`

```
0.5 + 2
2.5
```

```
3 + 0.75
```

Interactions Between `double` and `int`

Values *coerce* into `double`

```
0.5 + 2
2.5
```

```
3 + 0.75
3.75
```

Interactions Between `double` and `long`

Values *coerce* into `double`

Interactions Between `double` and `long`

Values *coerce* into `double`

```
0.5 + 4l
```

Interactions Between `double` and `long`

Values *coerce* into `double`

```
0.5 + 4l
4.5
```

Interactions Between `double` and `long`

Values *coerce* into `double`

```
0.5 + 4l
4.5
```

```
3l + 0.75
```

Interactions Between `double` and `long`

Values *coerce* into `double`

```
0.5 + 4l
4.5
```

```
3l + 0.75
3.75
```

Exponentiation with `Math.pow()`

Exponentiation

Use `Math.pow()` for exponentiation (something to the power of something else)

Exponentiation

Use `Math.pow()` for exponentiation
(something to the power of something else)

Wanted: 2^7

Exponentiation

Use `Math.pow()` for exponentiation
(something to the power of something else)

Wanted: 2^7

```
Math.pow(2, 7)
```

Exponentiation

Use `Math.pow()` for exponentiation
(something to the power of something else)

Wanted: 2^7

```
Math.pow(2, 7)
```

Wanted: $3.4^{5.6}$

Exponentiation

Use `Math.pow()` for exponentiation
(something to the power of something else)

Wanted: 2^7

```
Math.pow(2, 7)
```

Wanted: $3.4^{5.6}$

```
Math.pow(3.4, 5.6)
```

Example:
`Exponentiation.java`