DEMO

Adding.java

Totalling.java

Takeaways

- 0.1 cannot be represented exactly in binary so we just an approximated value (rounded-up as shown in the example), that leads to the unexpected result.
- And adding a very small quantity to a very large quantity can mean the smaller quantity falls off the end of the mantissa.
- But if we add small quantities to each other, this doesn’t happen. And if they accumulate into a larger quantity, they may not be lost when we finally add the big quantity in.

DEMO

ArrayTotal.java

Takeaways

- When adding floating point numbers, add the smallest first.
- More generally, try to avoid adding dissimilar quantities.
- Specific scenario: When adding a list of floating point numbers, **sort** them first.

DEMO

`LoopCounter.java`

Takeaways

- Don’t use floating point variables to control what is essentially a counted loop.
- Also, use fewer arithmetic operations where possible.
  - fewer operations means less error being accumulated
- Avoid checking equality between two numbers using “==”
  - don’t check this condition: `x == 0.207`
  - check this: `(x >= 0.207-0.0001) && (x <= 0.207+0.0001)`
  - `or check this: abs(x - 0.207) <= 0.0001`

DEMO

`Examine.java`

Takeaway

- What are all these extra digits??
- `4/5 = 1.10011001 10011001 10011001 1001100...(binary) x 2^(-1)`
- This gets rounded to `1.10011001 10011001 1001101(binary) x 2^(-1)`
- When we print, it gets converted back to decimal, which is:
  `0.800000011920928955078125000000`
- However, the best precision you have here is just

  `2^{-23}*2^{-1} ≈ 6e-8`
- Only the 7 blue digits are significant
- **Don’t print more precision in your output than you are holding.**