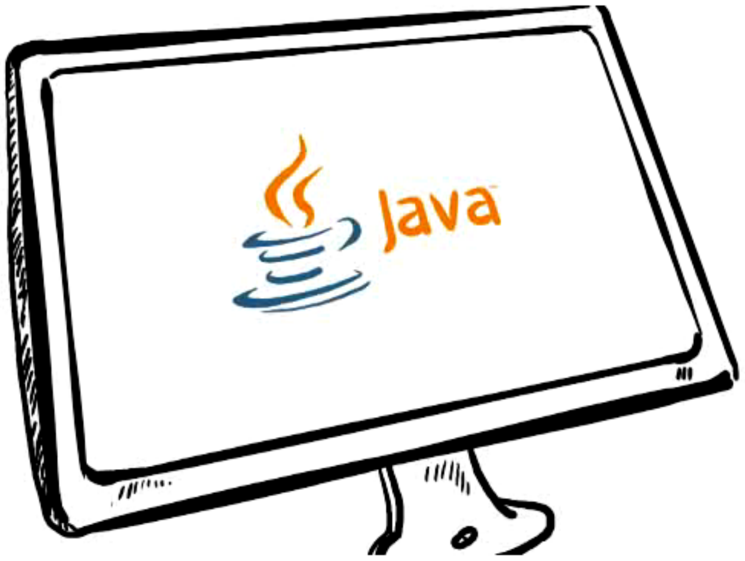
CMPS 200

Commenting and Documenting Your Programs



Spring 2016-17

Why comment?

- Undocumented programs are notoriously difficult to understand
- If you cannot understand a program, then you cannot:
  - ensure that the code works correctly
  - correct mistakes
  - add functionality

  That is, you cannot verify and maintain the code
- Analogy: a car that cannot be repaired the first time it breaks down. It therefore has no value!
- How do mechanics repair a car: they have a shop manual that describes the cars design, parts, interconnection of parts, etc

Why worry about program correctness?

- Type “software disasters” into Google!
- RISKS newsgroup

Why worry about program maintenance?

- Mainataining software costs more than developing it in the first place!
- Analogy: $20K car that costs $80K to run over its lifetime!

Why is software hard to get right?

- A program is a (long) sequence of instructions
- The program is intended to achieve a definite effect
- Transformational program: takes a single input, and produces a single output
- Program is intended to have some overall effect, i..e, to compute an output that is some function of the input
- This required effect is usually described by a specification
- The overall effect of a program is the cumulative result of the effects of each instruction, taken in sequence
- Program is correct: the cumulative effect of the sequence of instructions is the same as the specification
- Problem: it is difficult to keep track of the cumulative effect of the instructions

Analogy: directions to a location

- You are at location A, and wish to proceed to location B
- You are give a sequence of instructions:
  - Go forward 100 metres
  - Turn right, go 50 metres
  - Turn left, go ½ kilometre
  - etc
- Problem: given location A and a sequence of such instructions, how do you compute the final location?
- Answer: computing the final location is extremely difficult, unless you use a map!!!
- Likewise, computing the final effect of a sequence of program instructions is extremely difficult, unless you have a "mental map" of the program's execution
- The mental map is provided by the programs documentation
- Comments give you "mental checkpoints"

Documenting a program properly

- Given: a specification and a program
- The Problem: document the program with comments, so that you can check that the program is correct w.r.t. the specification
- A Solution. For each instruction:
  - Write a specification comment before the instruction which states what the instruction is supposed to achive
  - Write a description comment next to the instruction which describes the instruction
  - Write an effect (or summary) comment after the instruction which states what the instruction has actually done
  - Compare specification comment and effect comment. Use description comment to help make the comparison.

Documenting and reasoning in practice

- Want to use comments to help write the code in the first place
- Ideally, the program is then *correct by construction*
- Start with the specification
- Break it down into a sequence of small steps, and write a specification comment for each step
- Check the this sequence of steps actually implements the overall specification
- Write code that implements each specification comment, and check its correctness
- Use the method in a nested manner for nested constructs (conditionals, loops)
- When instructions are very simple, can treat several instructions together (avoid too many comments)