Clean Code
- Comments
- Functions

Comments
- A well placed comment can be great
- Too many comments can clutter up a program
- Outdated comments that convey misinformation can be disasterous
- A good comment can compensate for a failure to express ourselves in code.
- Comments all to often provide misinformation
- Truth can be found in one place, the code
- We should seek to explain ourselves in code

EX|

```
//Check to see if the employee is eligable for full benefits
If ( (employee.flags & HOURLY_FLAG) && (employee.age > 65) ) {}
```

  - Move the check into a specific function

```
If ( employee.ifEligibleForFullBenifits() ) {}
```

Good Comments
Legal Comments: Corporate statement in all code.
Informative Comments: Comments are needed to fully explain our code.
Explanation of Intent: Explain why you made a choice to code it that way.
Clarification: Sometimes we need comments to clarify the meaning of some inputs & outputs.
Warning of consequences

Functions
- Functions should be SMALL (smol boi):
- Self Test: Can you understand it all it does, in under 3 minutes?

Do One Thing
- Functions should only do one thing.
- A function should only do steps that are one level of abstraction below the stated name of the function.

Read Code From Top to Bottom
- Every function should be followed by those at the next level of abstraction.
- Switch statement ,
  By their nature do more than one thing
- They reside in low level classes.

Function Arguments:
- The ideal # of function is zero.
- Not always possible but never more than 3.
- Less arguments
    - Easier to read & understand
    - Easier to test.
- Output arguments are very hard to understand.

One Argument:
1. Asking a question about the argument.
    a. Boolean fileExists(“MyFile”);
2. Operating on that argument in some way.
    a. InputStream fileOpen(“MyFile”);

Avoid flag arguments if possible.
- Indicates that function is doing more than one thing.
- Indicates that we may need two functions.

Two Arguments:
1. Somethings come in pairs (x,y) coordinates
    a. Point p = newPoint(0,0);
2. Compare two things
    a. assertEquals(expected, actual);

Three Arguments:
1. Even harder to understand.
2. Easy to confuse order.
3. Sometimes needed
    a. assertEquals(expected, actual, delta);

More than Three Arguments:
1. Indicates that we should create a class for the common arguments.

Names for Functions:
EX) WriteField(Name)
- Another rule of thumb is to use keywords to clarify order.
- assertEquals(expected, actual);
- assertExpectedEqualsActual(expected, actual);

EX) lonLatToXY(lon,lat);

Have no side effects
Functions should only do what they say they are going to do.
-If they do hidden things we could unintended consequences that are difficult to tract down.

Closing Tips:

- Don’t use output arguments
- Functions should do <u>something</u> or <u>answer something</u> but not both.
- Use exceptions in functions instead of returning error codes. ***(What AJ does)***
- Try to extract Try/Catch to their own functions
- Look for code duplicate and factor out into functions.