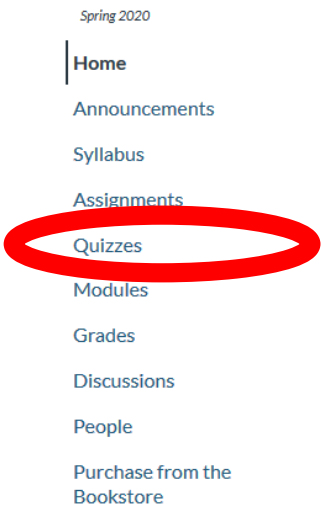

EECS 293:

Quiz Starting at 10:35am
Head to canvas and click on Quizzes
You can start as soon as the quiz is published on canvas
Lecture starting soon ...

Class Design

Reason: manage complexity
(warning: no single compelling measure of complexity)
- Create compelling abstraction
- Hide implementation details

Consequences:
- code more clearly self-documented
- code more easily seen to be correct
- function/ class names more informative

Other reasons:
- Limit effect of changes
- Central point of control
- Streamline
  - Parameter passing

Void foo(**arg1, arg2, … , arg6**, *arg 7, ... arg 10*) { … }

Void foo(arg16, arg710)
Class Arg16 { .. } Class arg710

- Return values

- Reusable code (software product line)
- Improving performance

Class names

- Classes are named as nouns (classes used to create Objects). E.g., TypeName, Set, Exception, Logger

Avoid verbs: class CheckType, but use TypeChecker

Class (Interface) Design

If a class seems to balloon and do too many unrelated Functions (not cohesive), move some of its action to other classes

E.g., private methods no more than 5-9

Public class Klass {

private foo1() {.. }
private foo2() …
private foo12()
private foo13()
…
private foo20()

Public bar() { foo12(); }
}

Class Helper1 { // package private
private foo1() …
foo12()
}

Class Helper2 {
Foo13() private foo20() }

Favor readability over write-ability

Avoid *semantic* violation of encapsulation

```
class Year {
    public static final int DAYS = 365;
    ...
}

class LeapYear {
    public static final int DAYS = ???
}
```