CS 2110: Interface Design

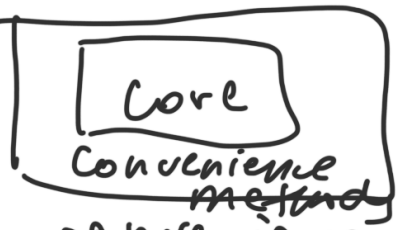
A2 Testing partners assigned — get in touch!

Last time: inheritance (ref: protected, abstract classes)

- How to divide into modules?
- what interfaces?

goals: separation of concerns — localize to modules
loose coupling
⇒ narrow interfaces

- expose just enough functionality to implement code efficiently.



- separate convenience methods into other modules
- easy to change implementation.

Documenting interfaces

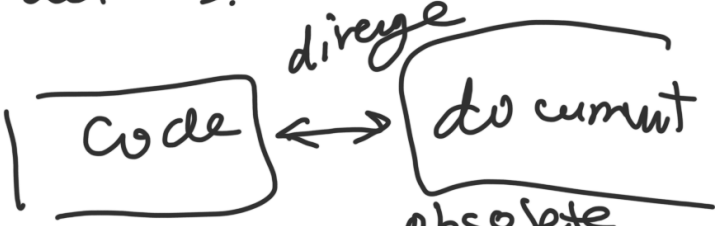
- Documentation = code (executed by humans)

  ⇒ know your audience

  ```
  x = x+1;   //add one to x   ← useless
  ```

  - attention is precious
  - say only what needs saying

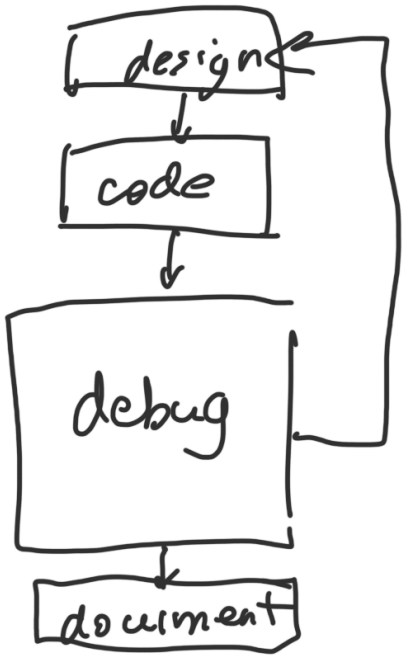
- Separate specs (for clients) from implementation notes (for implementers)
  - don't leak implementation details.
- Where to document?
  - comments in code
  - external document



code & documentation: tightly linked in both directions
pointers from code to document & vice versa.

when to document?

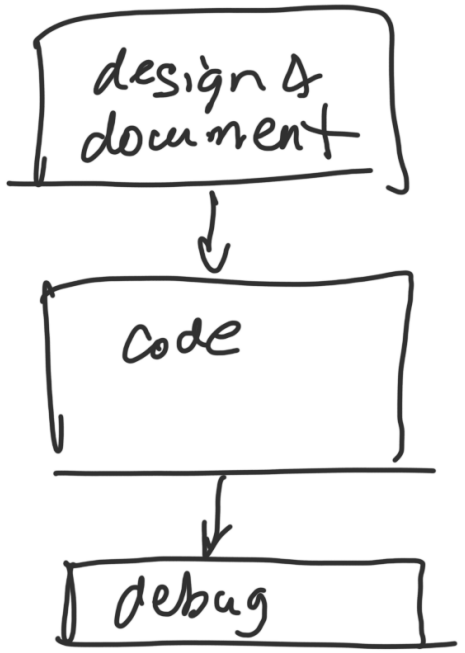
late



- design suffers
- debugging >50%

⇒ more design & documentation

early



more fun?

Design steps

Ex/ Polynomial

1. Class overview

```
/** An immutable polynomials. Ex. 2x^2 + 3x + 5 */
class Polynomial {
```

(2x^2 + 3x + 5 — notation)

2. Operations

- Creators - create objects of class
  - plus - add 2 polynomials, return sum.
  - zero()
  - monomial(int degree, int coefficient)

```
/** Returns a monomial of the
form mx^n where m is the coefficient
& n is the degree */
```

- Observers
  - report on state of object
  - no visible side effects

Spec clauses:
- returns/creates ↑ new object
- requires

- returns
- requires

- **Mutators**
  - primarily for side effects
  - only in mutable abstractions

  e.g. `Board.reset()`
  `Board.move()`

Spec:
- Requires
- Effects/ modifies
- (Returns)

Command/Query Separation
(Mutator / Observer)

every operation in one of 3 categories

```
interface Board {
  /** Returns: value of tile at row r, column c,
      or 0 if empty */
  int cell(int r, int c)
```

A: Creater
B: Observer ←
C: Mutator

Creators

- Constructor Creates:

```
/** Returns the zero polynomial */
Polynomial() {...}
```

- Factory method theZero=...

```
static Polynomial theZero = ...
public static Polynomial zero() { return theZero }
```

— looser coupling ⇒ can change implementation of creator, e.g. to use a subclass of Polynomial

Standard Observers

String toString()

"value:" to

— human-readable output

boolean equals(Object o)

Equality?

1. Leibniz equality
   "equal iff indistinguishable"
2. State equality
   "equal if same state"

- immutable abstractions
  (e.g. String, Polynomial, ...)
  ⇒ state equality = Leibniz equality
  ⇒ use state equality
- mutable abstractions
  Leibniz equality ⇒ Object.equals()
  same as ==

mutable objects may "stop being equal"

Standard antipatterns

- setters: methods for setting ivars
- setter + getter ⇒ no abstraction barrier.
- rep exposure – leaking mutable rep

```
class Board {
    int [7][] cells;
int[][] getCells() { return cells; }
}
```

client can change. (need to copy)