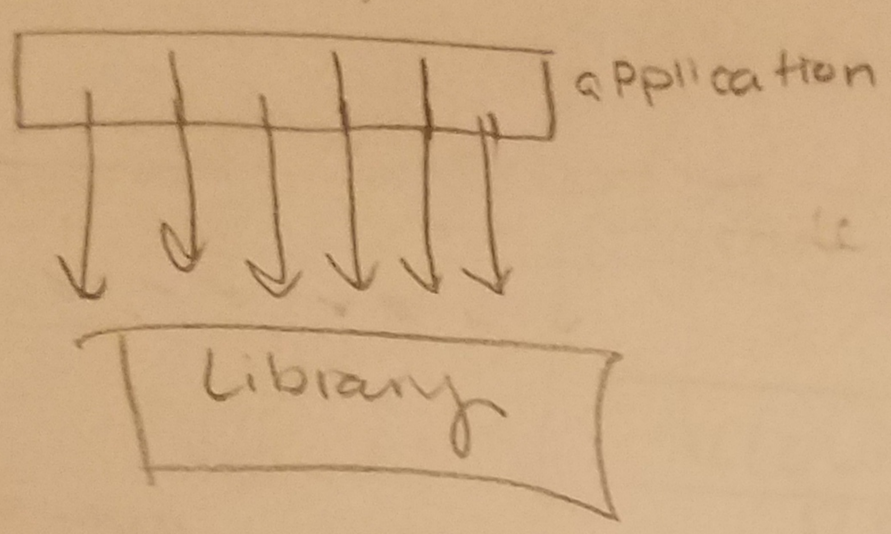
COMS 309 handout WK11 Fri 2018/03/23

OLD diagram

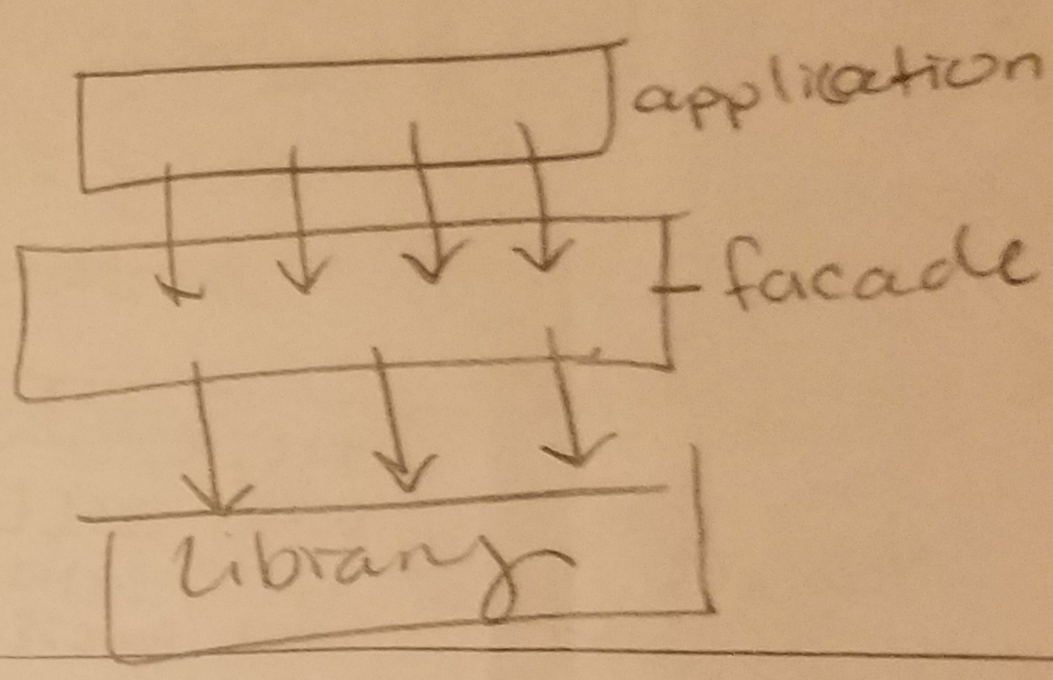
Draw

1. Facade pattern



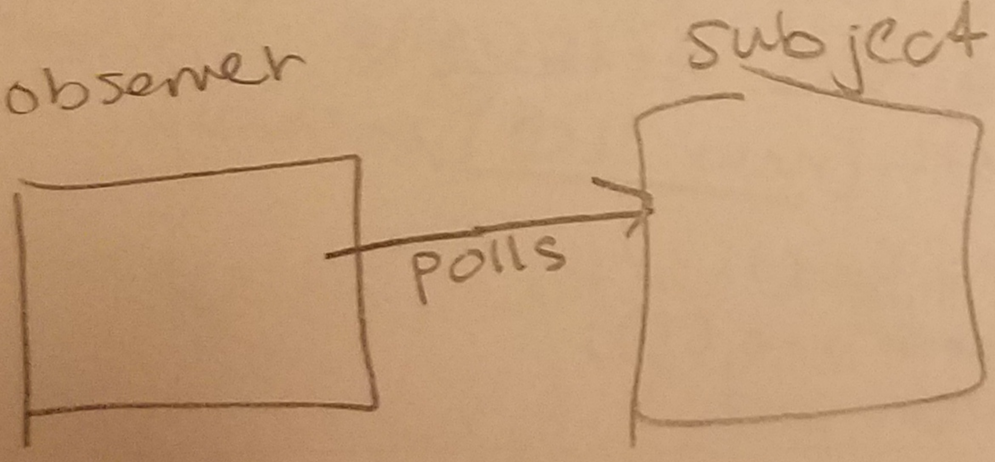
New diagram

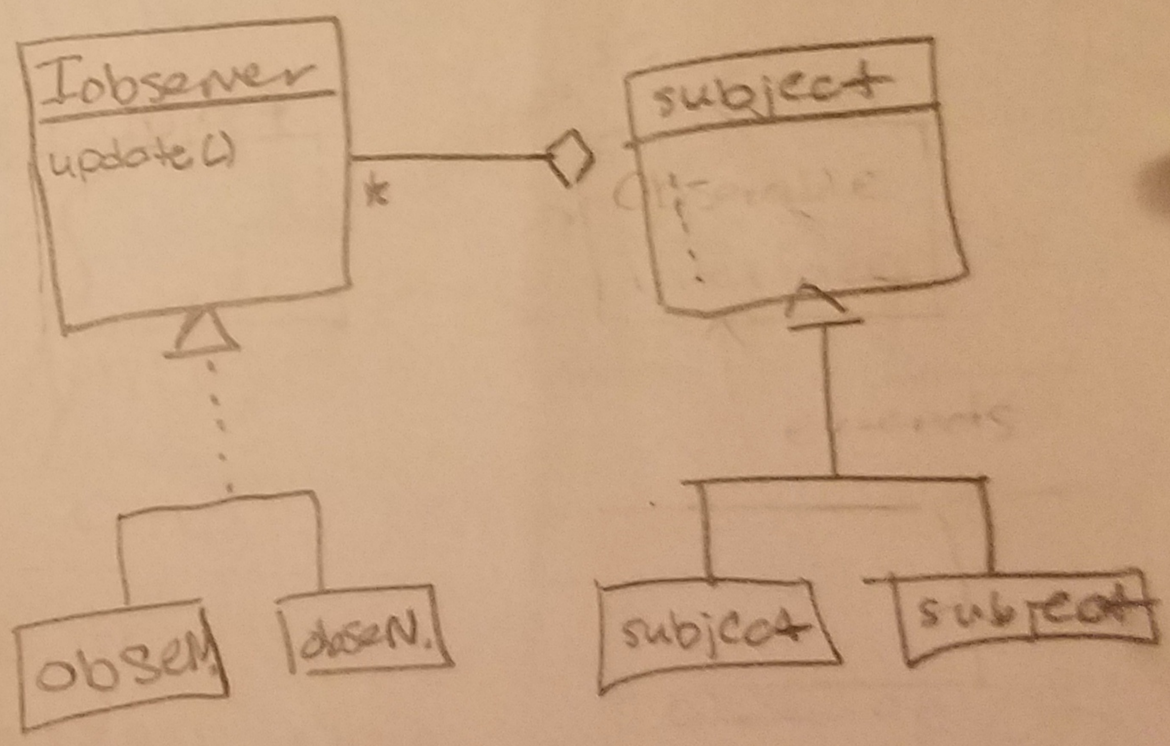
Draw



2. Observer pattern

observer





Q. See diagram on next page.
What is a major problem with HLC class

(a) for maintainers?
- hardcoding: if you change LLC you have to change HLC & recompile it every time

(b) for testers.
infoo(), you make a call to LLC obj, but its hard to stub. ⇒ hard to test b/c you have to change source code to run test

Application

```
main(){
  HLC h =
    new HLC();
  h.foo();
}
```

High Level Class

```
class HLC {
  void foo() {
    LLC l = new LLC();
    l.doSomething();
  }
  ILC l;
  void setLL(ILC l){
    this.l=l;}
  void foo(){ l.foo();}
}
```

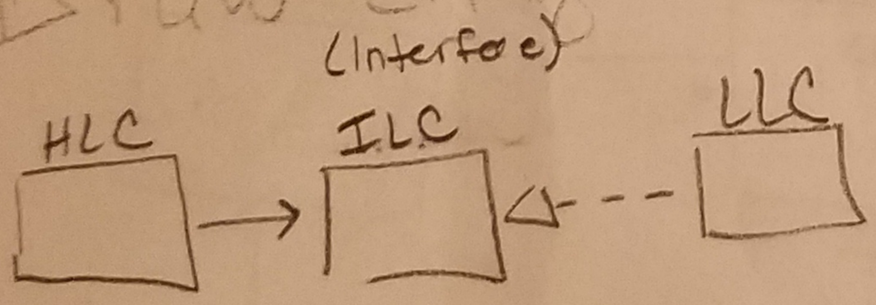
Low Level class

```
class LLC {
  void doSomething();
}
```

Independent of LLC ⇒ good

Dependency Injection (DI approach)?

Draw diagram of soln approach




```
to test new code:
public void testFoo(){
  h.setL(new LLC()) ← stub
  ?= h.foo();
  assertTrue(?);
}
```

use interface

). How does this help maintainers? testers?
now they can make objs to test. the HLC doesn't directly depend on LLC

How is this used in frameworks (like Spring)?

~~Deploy~~ Injection:

example of test 4:
- make interface & have class implement interface
- rewrite test class so there are no dependencies
make high level class implement the interface
Interface = new LLClass()