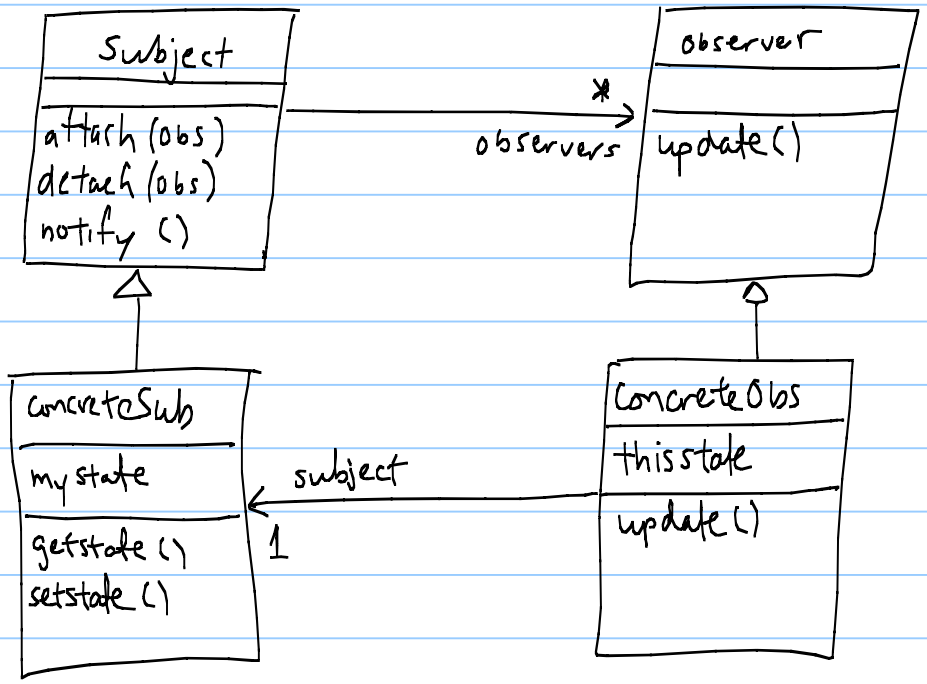
Comp 304 REEHAN SHAIKH

Note Title 3/24/2006



Implementation Issues / Variation

1) observer references where?
- Suppose # of sub is very large & # of obs. is very small. Since each obs has 1 subject, if we store the obs. refs in sub, we have a lot of wasted memory. Since all subjects will have a list of obs. some will be empty. Not good.
  in subject → memory ↑, speed ↑
- Some association class / lookup table / hash to store which subject has which observers
  in assoc. class → memory ↓ speed ↓ since we have to trace subjects in some has

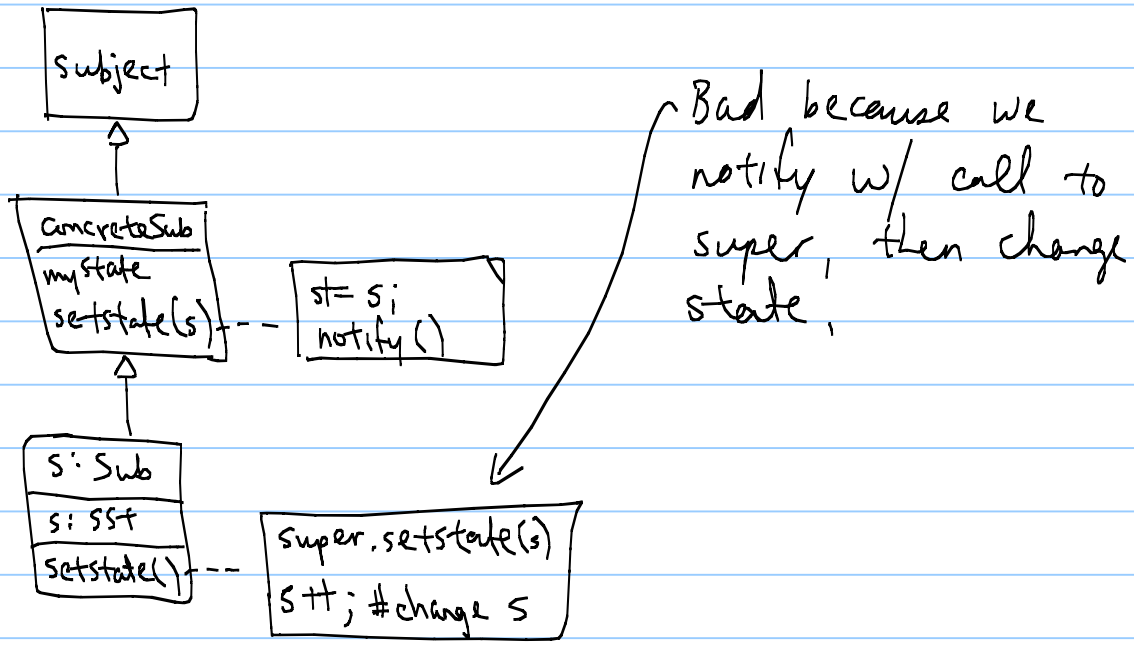
2) Observers have > 1 subject
   If so, when subject calls update(), it must send a reference to itself so the observer can getstate()

of that subject. Now, we don't need the arrow from concreteObs to concreteSub.

3) Who triggers update (send notify)?
SAFETY vs PERFORMANCE
↓ ↑
after every setstate() we do a notify() & updates
We allow many setstates() so we don't have a flood of notify's & updates

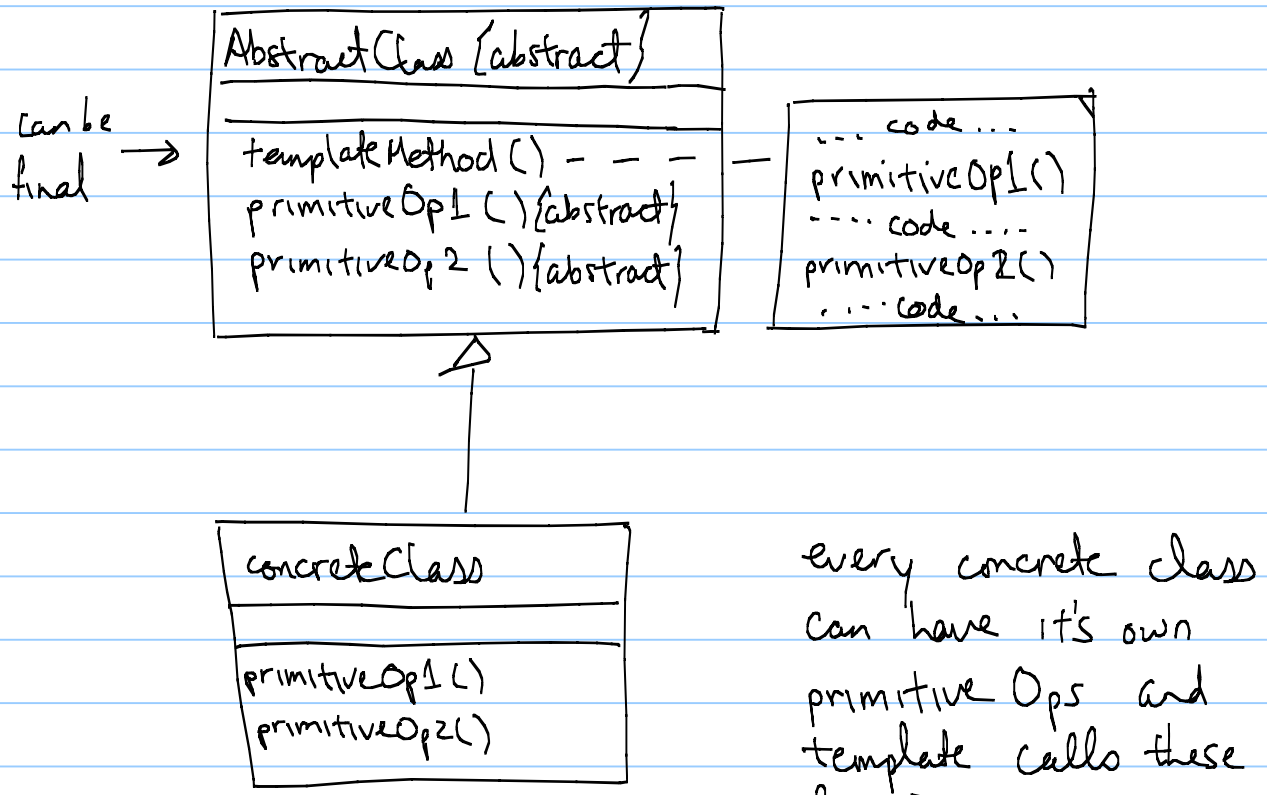
4) Delete Subject → do we delete Observers?
Depends on what we want. Subject should tell its observers it is dying! Then Obs's should do what they have to. Don't call getstate after!
Delete Observer → just detach from subject then die.

5) Subject state self-consistency before notify



TEMPLATE METHOD Pattern

uses inheritance to vary part of an algorithm (structure remains Fixed)



This will solve #5 above. TemplateMethod will have notify() as it's last line and it will call other methods (that can change state) but now, notify() will always be called last.

6) PUSH vs PULL
if update sends state, then Obs knows something about Subject, not a nice separation anymore.

	detail into sent	Obs. callback	IND.	Performance
PUSH	Y	N	Low	High
PULL	N	Y	High	Low

7) Register for specific interested

Observer can specify what part of the state it is interested in. So, the whole state doesn't need to be sent but now we have to keep track of what each observer wants.