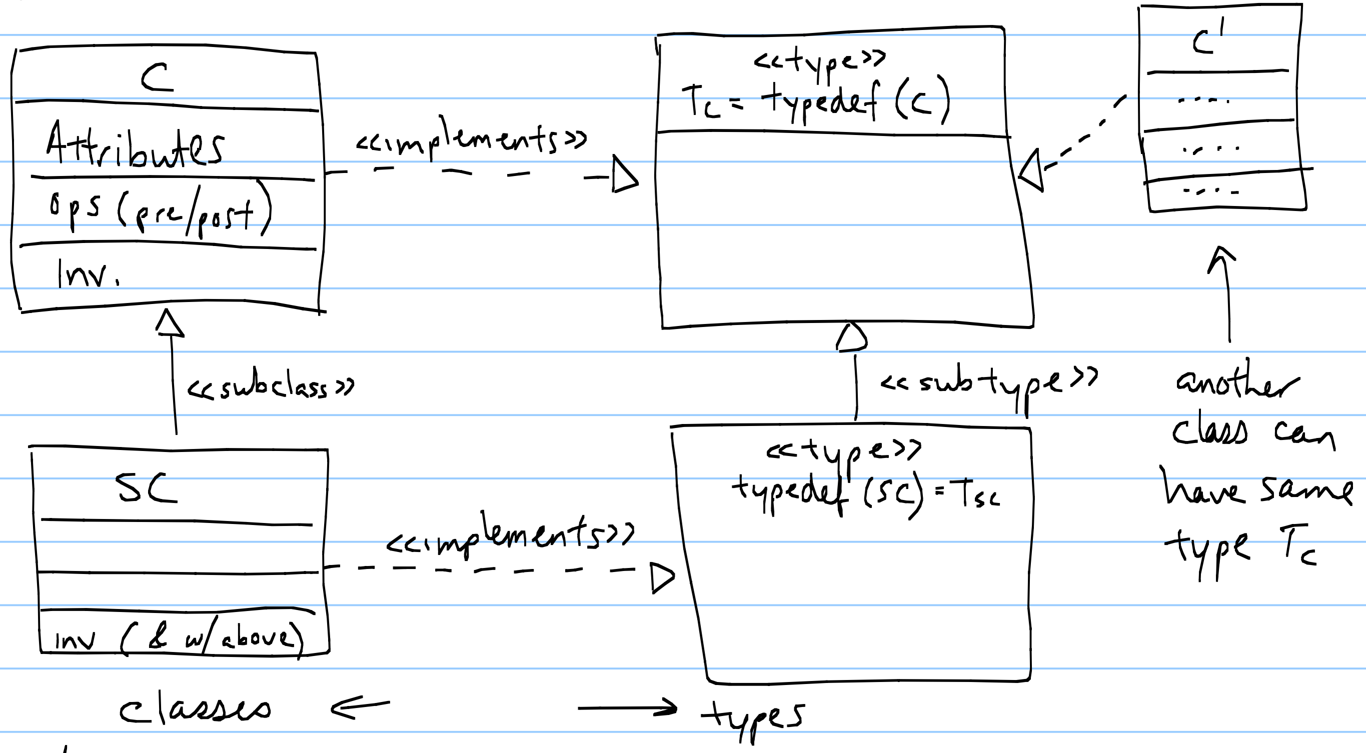
Comp 304 REEHAN SHAIKH

Note Title 4/7/2006

① Type Conformance (← closed behaviour ②)



Good OO Design: SUB-CLASSING ≡ SUBTYPING

Liskov substitutability Principle (LSP)

If Tsc is a subtype of Tc, Tsc must conform to Tc. An object of type Tsc can be provided in any context where an object of type Tc is expected.

Correctness is preserved when any accessor (not modifier) operations of the object is executed.

get / set

behaves in same way!

this is in #②, closed behaviour.

Statespace

1) Statespace of Tsc must have at least the same

dimensions as T_c. If there are more, then the subtype extends.

2) In the shared dimensions, the state space of T_{sc} must be $\subseteq$ state space of T_c (Projection)
Invariant of T_{sc} must be stronger than T_c.

Behaviour

1) T_{sc} must have at least the operations as T_c
2) The same operations must be identical (signature of operation, we don't care how it is implemented, the outside world just needs access to this method)
x same name, args, return
$T_{c.op} : T_{args} \rightarrow T_{ret}$

ie: string f (int i, float j) is type(f) : $\mathbb{Z} \times \mathbb{R} \rightarrow$ String
- now suppose we override f(float i, float j)
is this bad or good? good based on contravariance
- now we have string <10 chars f(float i, float j)
is also good because caller is getting something smaller than expected, so it won't matter.

-The pre-cond of $T_{sc}.op$ must be equal or weaker than the pre-cond of $T_c.op$. Type of arguments of $T_{sc}.op$ is a supertype of the type of arguments of $T_c.op$. This is contravariance.
- ″ post- ″ ″ ″ ″ ″ ″ ″ stronger
″ ″ ″ ″ ″ ″ ″ ″ return types ″
″ ″ ″ subtype ″ ″ ″ ″ return types ″ ″

this is co-variance

$$\begin{array}{cccccc} T_{c.op} & : & T_{args} & \longrightarrow & T_{ret} & \text{type of this} \\ \ddot{\vee}\,\text{sub} & & \underset{..}{\wedge}\,\text{super} & & \ddot{\vee}\,\text{sub} & \\ T_{sc.op} & : & T_{args} & \longrightarrow & T_{ret} & \text{this should be a} \\ & & \text{contra-} & & \text{co-} & \text{variance} \end{array}$$

ie: $[0, 10]$ $f(a1: [10, 20],\ a2: [100, 1000])$

$\downarrow$override

$f(a1: [0, 30],\ a2: [0, 2000])$

We still accept what the old one accepted. If we only accept less than 1st f, not good because function should still be at least as it was before, so accepting arguments should be equal or more in overridden f. but return type should be same or less because we are expecting something [0,10], so we can't give anything outside this range. It could be weaker than this range.