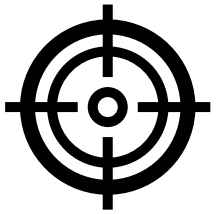
Design by Contract and Inheritance

Dr. Hoda Khalil

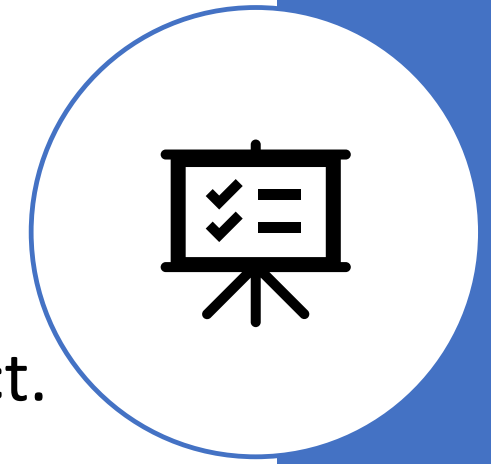
Objectives

- Understand Liskov Substitution Principle (LSP).
- Know and be able to apply the rules of LSP.



Outline

- Inheritance.
- Liskov Substitution Principle.
- Example.
- Substitution Rules.
- Liskov Substitution Principle and Design by Contract.
- Examples.
- Summary.

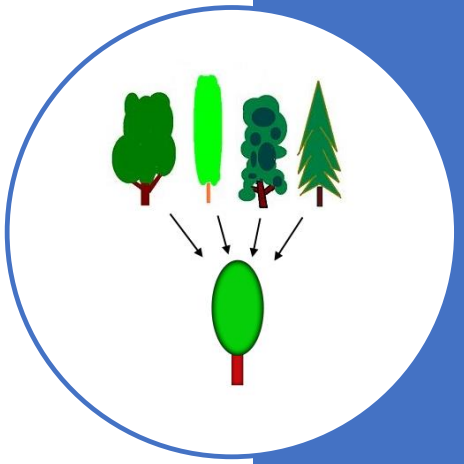


Inheritance (is_a)

- Inheritance: A reusability technique in which a child class inherits all the attributes and operations of a parent class (Bruegge and Dutoit 2014).

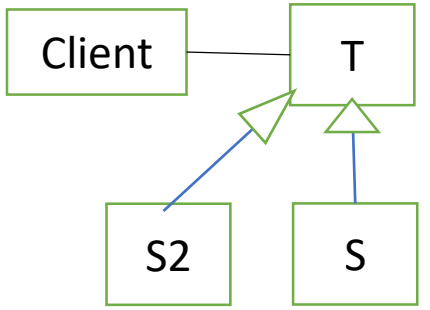
- Inheritance: A way of obtaining code without writing it. In particular, a subclass can inherit the implementation of its superclass methods(Liskov and Guttag 2001).

- Inheritance: A semantic notion by which the responsibilities (properties and constraints) of a subclass are considered to include the responsibilities of a superclass, in addition to its own, specifically declared responsibilities (IEEE Standard 2010) ✓



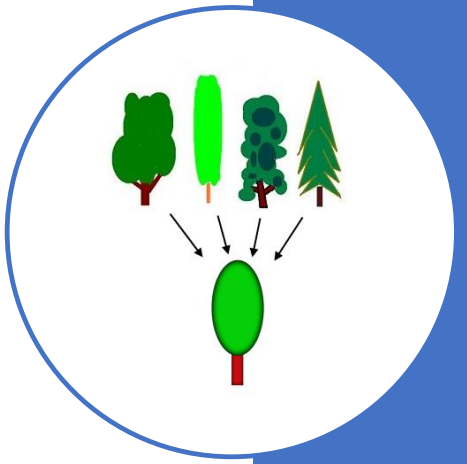
Liskov Substitution Principle (LSP)

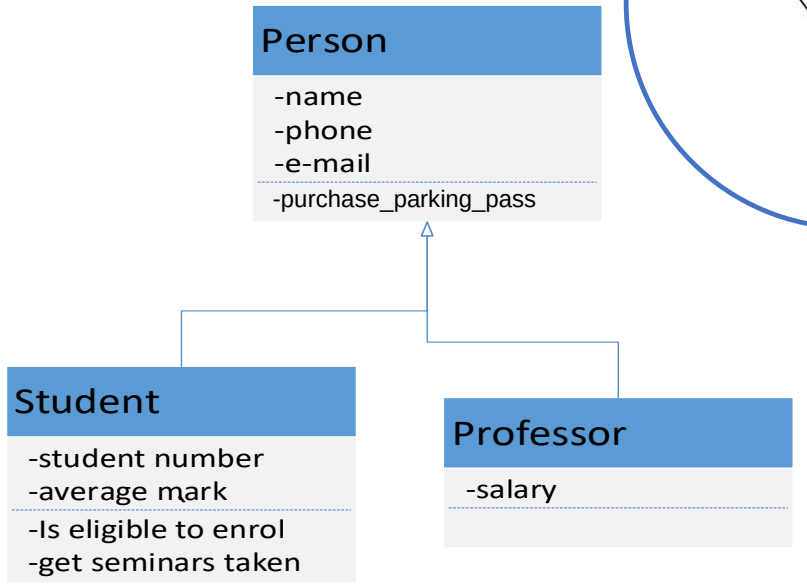
- The substitution principle provides abstraction by requiring that subtypes behave in accordance with the specification of their supertype (Liskov and Guttag 2001).
- The supertype behavior must be supported by the subtype (i.e. subtype objects can be substituted for supertype objects without affecting the behavior of the using code)
- In Object Oriented Design:
  - A method written in terms of a superclass T must be able to use instances of any subclass of T without knowing whether the instances are of subclass.
  - New subclasses of T can be added without modifying the methods of T (Extensibility).



Example

- A client calls the method purchase_parking_pass in an object `p` of type Person.
- Student and Professor are both subtypes of Person.
- An object `st` is an instance of subtype Student and an object `prof` is of subtype Professor.
- If `p` in the client code is replaced with `st` *or* `prof`, the client must not get affected.





Example: Violation of LSP

```
enum ShapeType {square, circle};


public class Shape
{
     private ShapeType myType;
     public Shape(ShapeType t){myType = t;}
     public void DrawShape(Shape s)
     {
          if( this.myType == ShapeType.square)
                    ((Square)this).Draw();
          else if(this.myType == ShapeType.circle)
                    ((Circle)this).Draw();
     }
}
```

Module cannot be extended without modification. Not extensible.
=>Violation of LSP.

Example adapted from the book: Agile Principles, Patterns, and Practices in C# (Martin and Martin 2006).

Example: Violation of LSP

```
public class Rectangle {
 protected double width;
 protected double height;
public void setWidth(double w) {
     width = w;}
public void setHeight(double h) {
     height = h;}
}
public class Square extends Rectangle {
public void setWidth(double w) {
     width = w; height = w;}
public void setHeight(double h) {
     width = h; height = h;}
}
```

If rectangle had a method draw, the caller will observe different behavior in case of substituting an object of type square (subtype) for an object of type rectangle (supertype).
=> Violation of LSP

Example adapted from the book: Agile Principles, Patterns, and Practices in C# (Martin and Martin 2006).

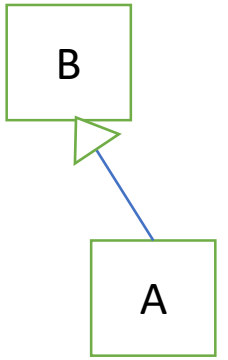
Substitution Rules

- Signature Rule
  - The subtype must have all the methods of the supertype.
  - The signatures of the subtype methods must be compatible with the signatures of the corresponding supertype methods.
- Method Rule
  - Calls for the subtype methods must behave like the corresponding supertype methods.
- Properties Rule
  - The subtype must preserve all properties that can be proved about supertype objects.

LSP and DbC: Preconditions Methods Rule

- An operation in a subclass that overrides an operation in a superclass must keep or weaken the preconditions.
- The precondition is weakened in the subclass operation => the subtype operation requires less from the caller.
- i.e. if class `A` overrides operation `op` in class `B`, and the preconditions of `A:op` is `preA` while the preconditions of `B:op` is `preB`, then

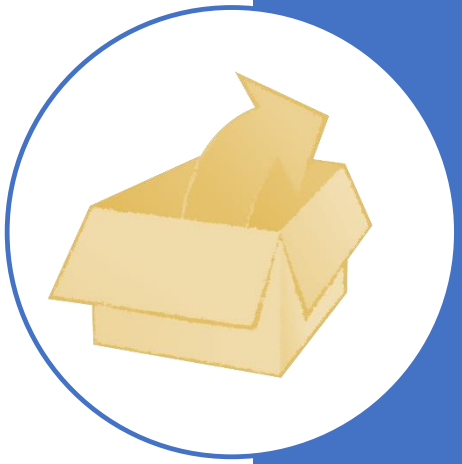
  `preB=> preA`



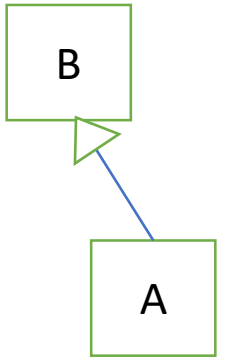
LSP and DbC: Postconditions Methods Rule

- An operation in a subclass that overrides an operation in a superclass must keep or strengthen the postconditions.
- The postconditions is strengthened in the subclass operation => the subtype operation provides more to the caller.
- i.e. if class `A` overrides operation `op` in class `B`, and the postconditions of `A:op` is `postA` while the preconditions of B:op is `postB`, then

  (`preB` and `postA`) => `postB`



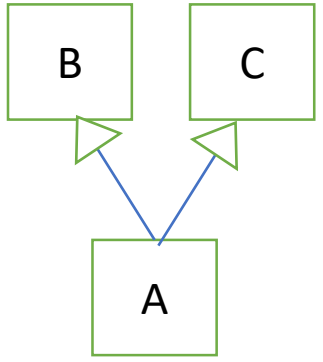
- If the precondition of `B:op` is satisfied and `B` is substituted for `A`, then the caller still expects the postcondition of `B:op` to be satisfied.



LSP and DbC: Invariants Properties Rule

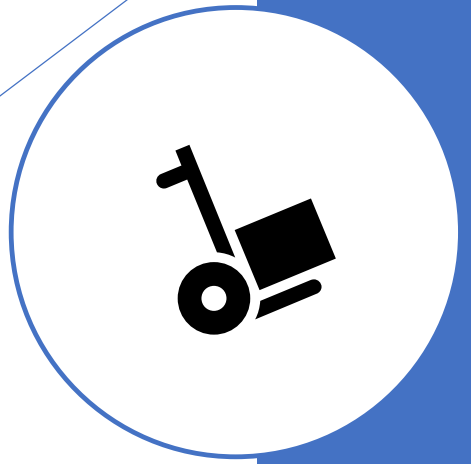
- A subtype (child) class must preserve the invariant of the supertype (parent)
- In the case of multiple inheritance, the invariant of the subclass must include all the invariants from the super classes (and-ed)
- i.e. if the invariants of classes `A`, `B` and `C` are `invA`, `invB` and `invC` respectively, and class `A` inherits from `B` and `C`, then

  `invA` => (`invB` and `invC`)



Example: Courier Company

Obligation on the client of the deliver service not to ask for the delivery of packages over 5 kg in weight
=> benefit for the supplier.



A courier company that delivers packages within a city.

Class COURIER

operation deliver (p: Package, d: Destination)

//Delivers package to destination.

Precondition: Weight of package p does not exceed 5 kg.

Postcondition: Package delivered within 3 working hours of when it was accepted.

Benefit for the client: Be sure that the package is delivered within 3 working hours.
=> Obligation on the supplier.

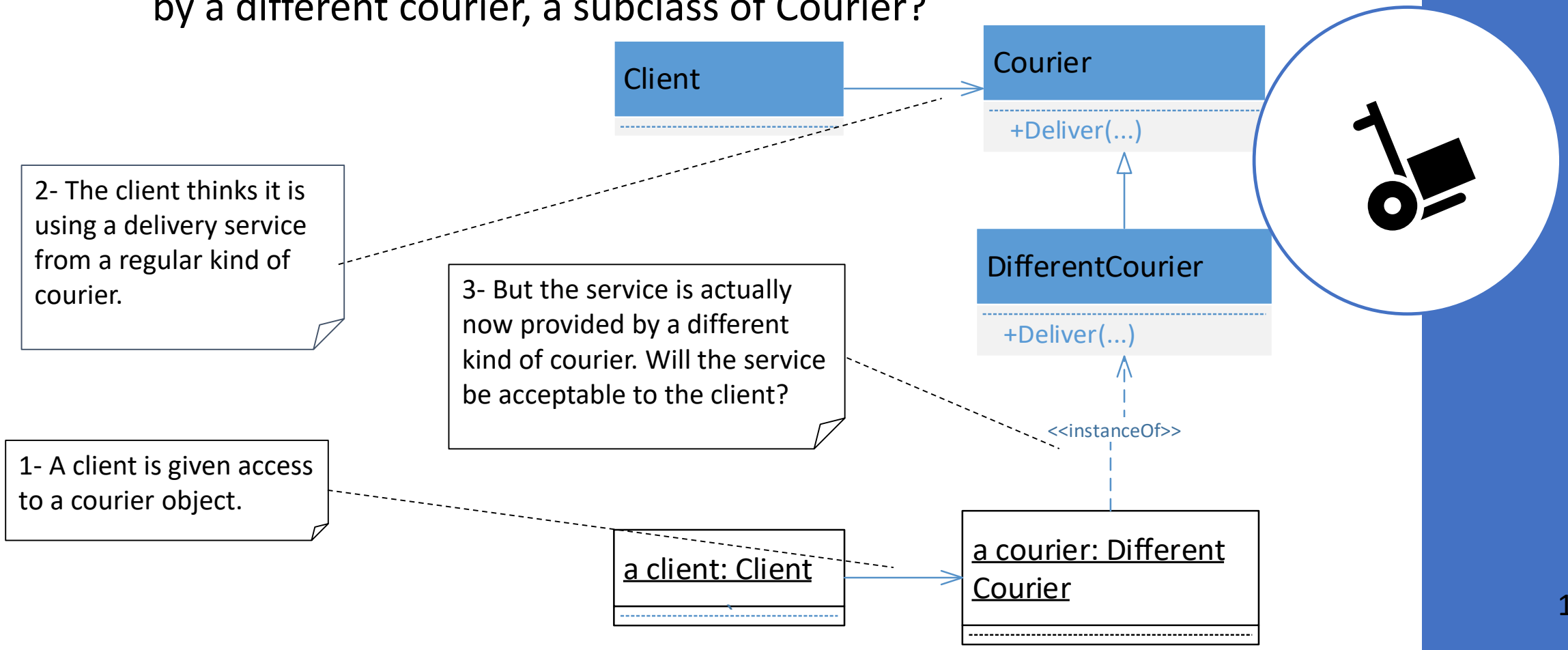
Example: Courier Service Deliver

	Obligations/ Responsibilities	Benefits/ Rights
Caller	Don’t ask for delivery of packages over 5 kg in weight.	Expects packages to be delivered within 3 working hours.
Callee	Deliver packages within 3 working hours.	Don’t have to cope with packages over 5 kg in weight.

Example: Courier Company

What if the preconditions and postconditions on the deliver service in the DifferentCourier class are not exactly the same as the Courier class?

What happens if the delivery service is actually provided by a different courier, a subclass of Courier?



Example: Courier Company

Redefine the Precondition for DifferentCourier::Deliver :

Case 1: e.g. Weight of package must not exceed 5 kg

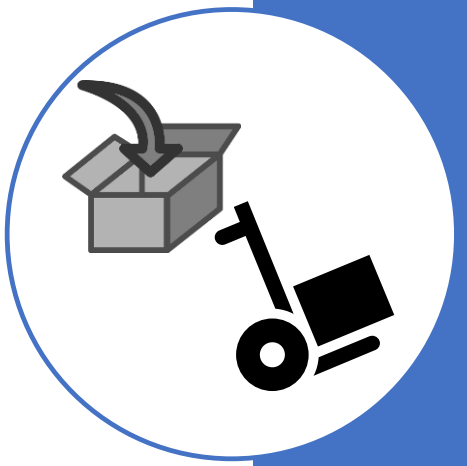
Same precondition ✓. Client satisfied ☺

Case 2: e.g. Weight of package must not exceed 8 kg

Less restrictive precondition ✓. Client satisfied ☺

Case 3: e.g. Weight of package must not exceed 3 kg

More restrictive precondition ✗. Client disappointed ☹



=> A feature in a subclass can leave the precondition on an inherited feature unchanged or it can redefine precondition to be a weaker one.

Example: Courier Company

Redefine the Postcondition for DifferentCourier::Deliver :

Case 1: e.g. Package delivered within 3 working hours.

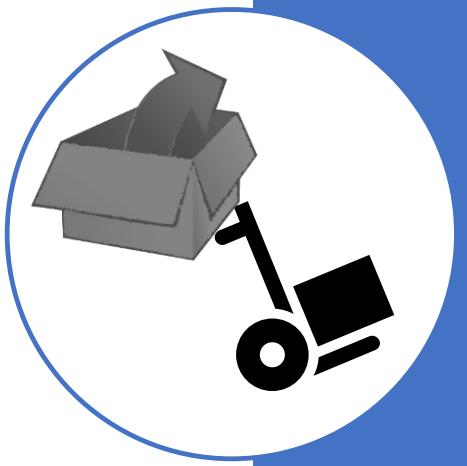
Same postcondition ✓. Client satisfied ☺

Case 2: e.g. Package delivered within 2 working hours.

Stronger constraint on the supplier ✓. Client satisfied ☺

Case 3: e.g. Package delivered within 5 hours.

Weaker constraint on the supplier ✘. Client disappointed ☹

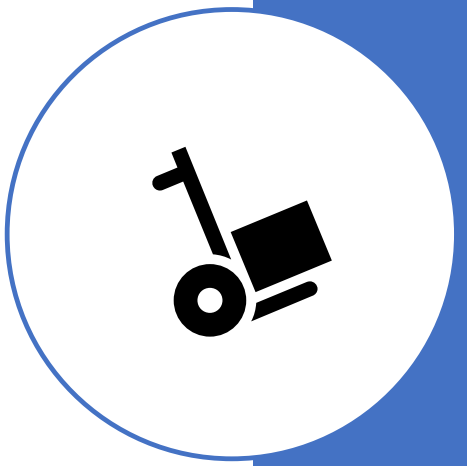


=> A feature in a subclass can leave the postcondition of an inherited feature unchanged or it can redefine postcondition to be a stronger one.

Example: Courier Company

Suppose that the courier service has an insurance policy to cover claims for damage to packages in transit.

Invariant: insurance_cover_in_dollars >=1000000



Redefine the Class Invariant for DifferentCourier:

e.g. insurance_cover_in_dollars >=2000000.

The new invariant for the DifferentCourier subclass should be

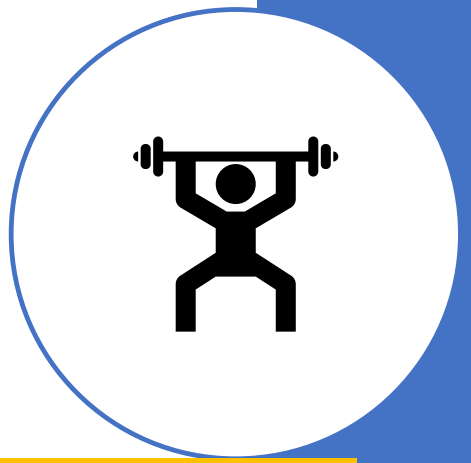
insurance_cover_in_dollars >=1000000 and insurance_cover_in_dollars >=2000000 ≡

insurance_cover_in_dollars >=2000000

=> A subclass inherits its superclass's invariant. It can add more clauses to the invariant, which are and-ed onto the inherited clause, thus strengthening the inherited invariant.

Exercise: Linked List implementation of Set

```
public class LinkedList {
/** Adds an element to the end of the list
 * Precondition:  element != null
 * postcondition: this.getLength() == old.getLength() + 1
 *       && this.contains(element) == true  */
public void addElement(Object element) {...}
}
class Set extends LinkedList {
/** Adds element, provided element is not already in the set
 * Precondition:  element != null && this.contains(element) == fa
 * postcondition: this.getLength() == old.getLength() + 1
 *       && this.contains(element) == true */
public void addElement(Object element) {...}
}
```



This strengthens the precondition ☹
Violetes LSP.

Example: OCL

context Person
inv: self.age >= 0 and

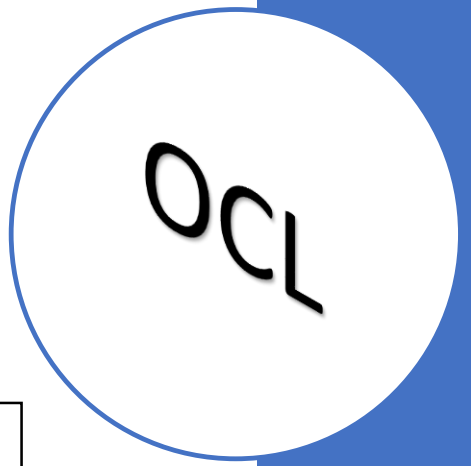
context Adult
inv: self.age>=18

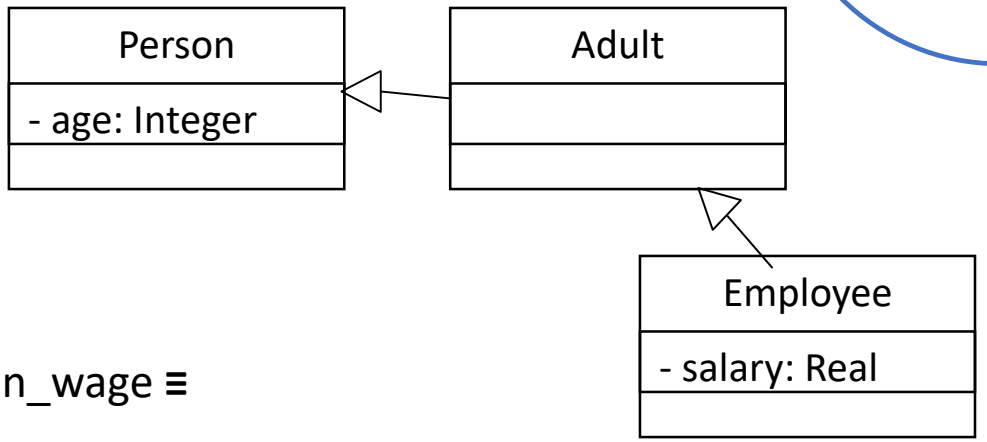
context Employee
inv: self.salary >= min_wage

Applying the Properties Rule:

context Adult
inv: self.age>=18 and self.age>= 0 ≡
self.age>=18

context Employee
inv: self.age>=18 and self.age>= 0 and self.salary>=min_wage ≡
self.age>=18 and self.salary>=min_wage





Example: OCL

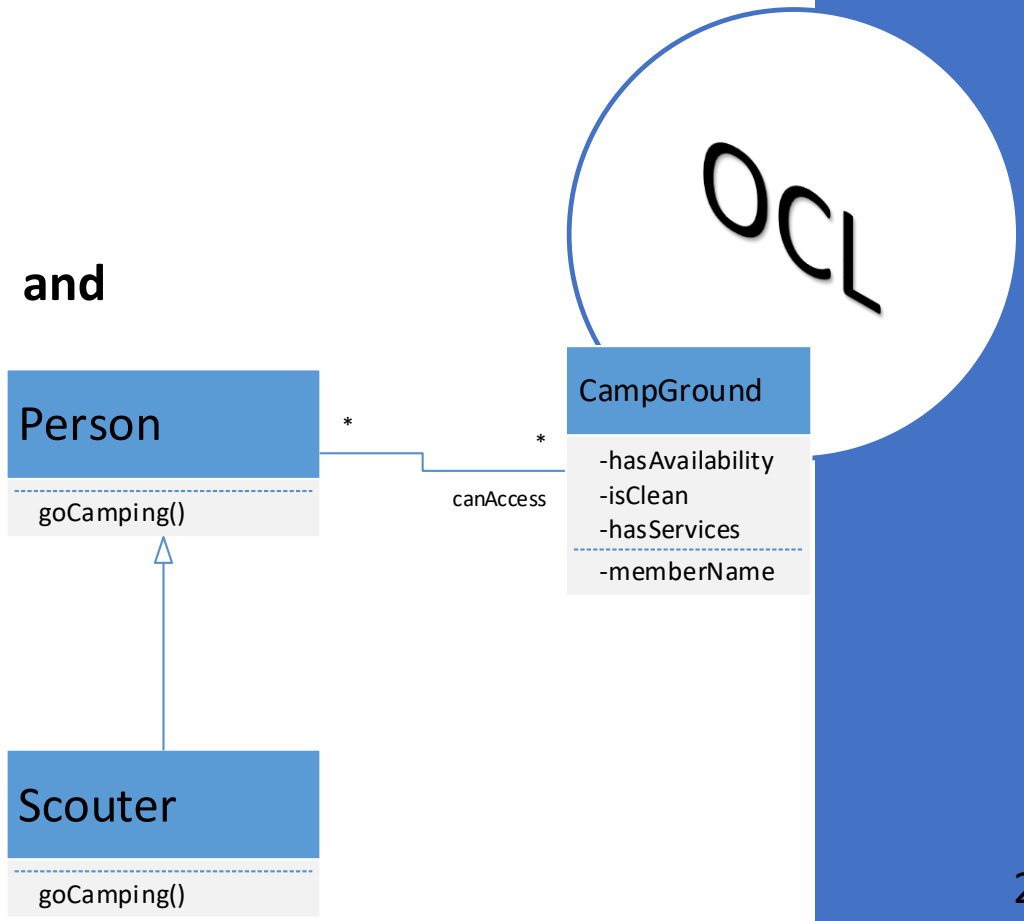
context Person::goCamping(campSite)

pre : self.canAccess->exists(g:CampGround I g = campSite **and** g.hasAvailability **and** g.hasServices and g.isClean) …(pre1)

context Scouter::goCamping(campSite)

pre : self.canAccess->exists(g:CampGround I g = campSite **and** g.hasAvailability)…(pre2)

➢ **Precondition is weakened (pre1 => pre2)**



Example: OCL

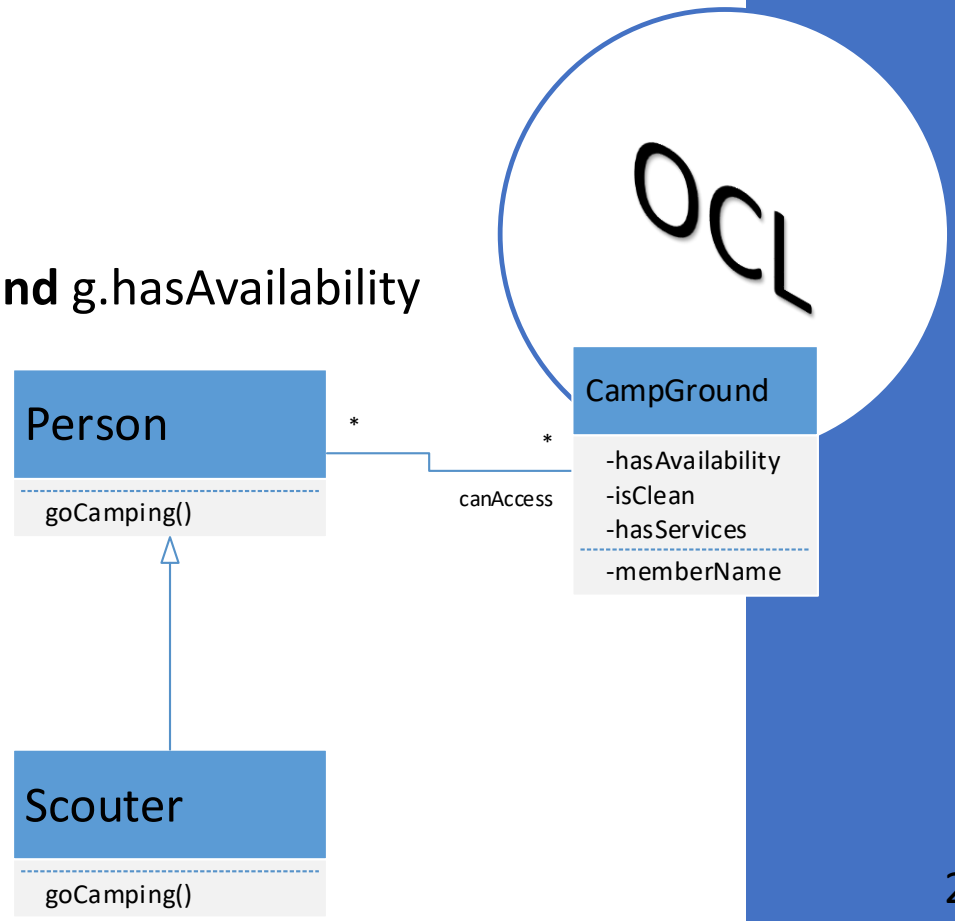
context Person::goCamping(campSite)

post : self.canAccess->exists(g:CampGround I g = campSite **and** g.has Availability)
…(post1)

context Scouter::goCamping(campSite)

post : self.canAccess->exists(g:CampGround I g = campSite **and** g.hasAvailability **and** g.isClean…(post2)

- **Postcondition is strengthened (pre1 and post2 => post1)**
- **The subtype method guarantees more than supertype Method.**



Summary

- Liskov Substitution Principle: The supertype behavior must be supported by the subtype (i.e. subtype objects can be substituted for supertype objects without affecting the behavior of the using code)
- If you are writing a subclass ad you inherit a feature from a superclass, you can redefine its contract
  - Keep or weaken the precondition (method rule).
  - Keep or strengthen the postcondition (method rule).
  - Preserve the invariant (properties rule).