Object-oriented design principles

Object-oriented software design

Dr. Balázs Simon
BME, IIT

LSP and DbC

LSP and DbC

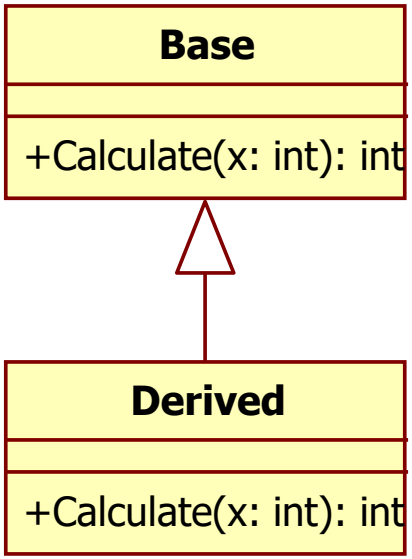
Is the LSP satisfied with the following conditions?

Base.Calculate pre:

```
0 <= x <= 100
```

Derived.Calculate pre:

```
20 <= x <= 50
```



LSP and DbC

Is the LSP satisfied with the following conditions? **Solution:** Violated

Explanation 1: Accepted range is smaller, pre-condition is strengthened.

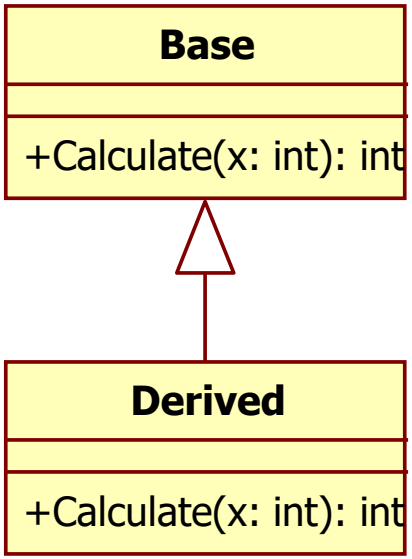
Explanation 2: A caller can safely pass values between 0 and 100 to a Base object. However, it cannot pass all these values to a Derived object through Base, and thus, Derived does not work within the limits of Base.

Base.Calculate pre:

```
0 <= x <= 100
```

Derived.Calculate pre:

```
20 <= x <= 50
```

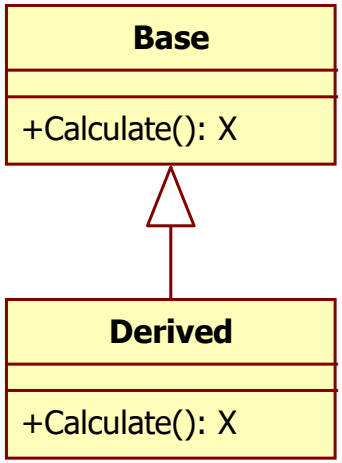


LSP and DbC

Is the LSP satisfied with the following conditions?

Base.Calculate post:
`X may be null`

Derived.Calculate post:
`X != null`



LSP and DbC

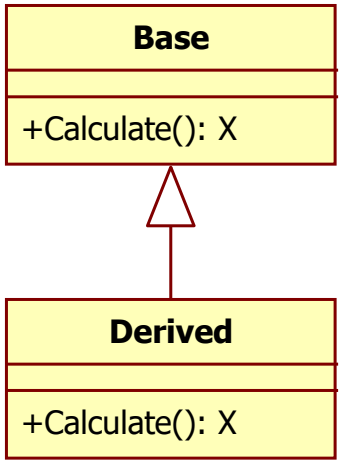
Is the LSP satisfied with the following conditions? **Solution:** Satisfied

Explanation 1: Returned range is smaller, post-condition is strengthened.

Explanation 2: A caller (client) object is prepared to take null and non-null values from Base. If Derived never returns null values through Base, it is still fine for the caller.

Base.Calculate post:
`X may be null`

Derived.Calculate post:
`X != null`



LSP and DbC

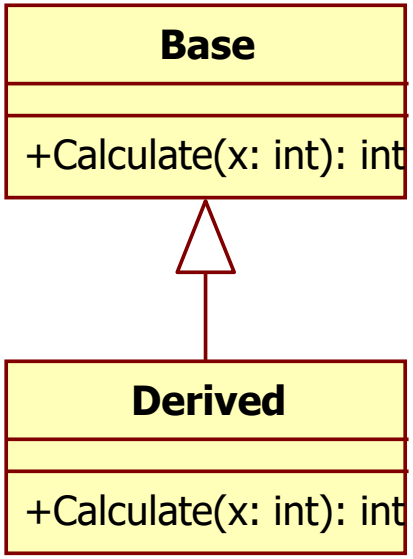
Is the LSP satisfied with the following conditions?

Base.Calculate pre:

```
0 <= x <= 100
```

Derived.Calculate pre:

```
0 <= x <= 1000
```



LSP and DbC

Is the LSP satisfied with the following conditions? **Solution:** Satisfied

Explanation 1: Accepted range is wider, pre-condition is weakened.

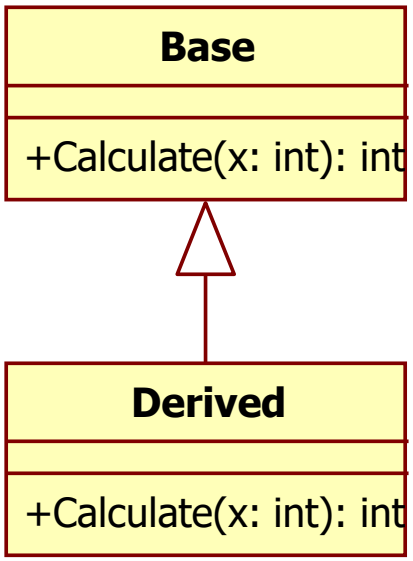
Explanation 2: A caller (client) object can safely pass values between 0 and 100 to Base, and also to Derived through Base, so Derived works between the limits of Base.

Base.Calculate pre:

```
0 <= x <= 100
```

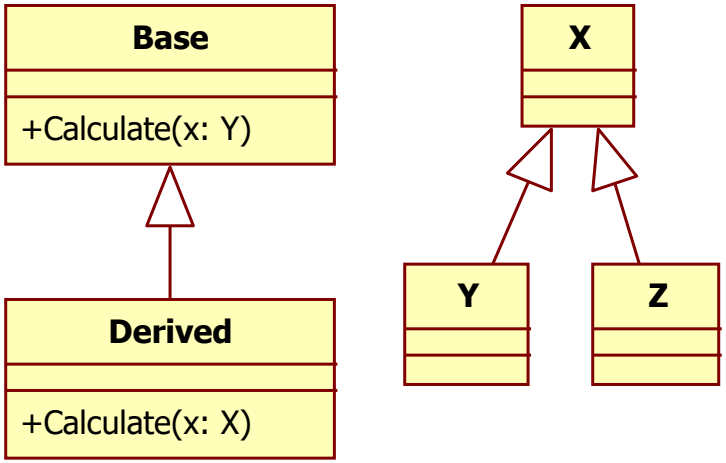
Derived.Calculate pre:

```
0 <= x <= 1000
```



LSP and DbC

Is the LSP satisfied with the following conditions?

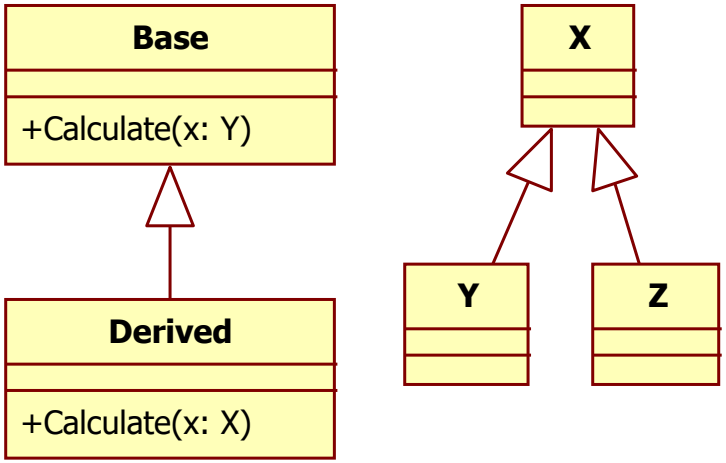


LSP and DbC

Is the LSP satisfied with the following conditions? **Solution:** Satisfied

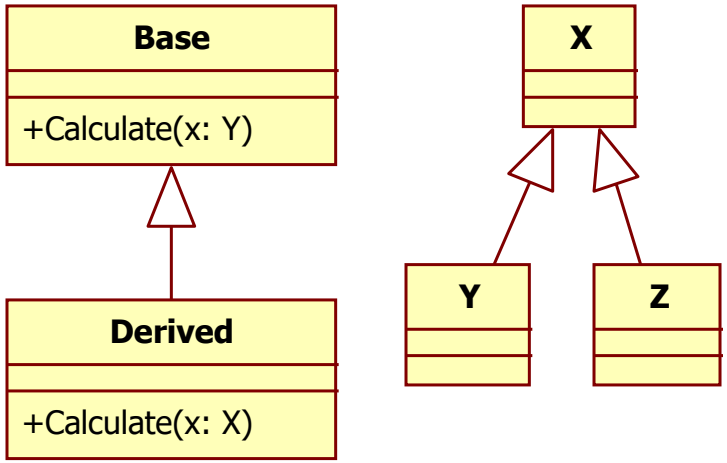
Explanation 1: Accepted range (X) is wider than in base (Y), pre-condition is weakened.

Explanation 2: A caller (client) object can safely pass Y and its descendants to Base, and also pass them to Derived through Base, so Derived works between the limits of Base.



This even has a name: contravariance

Contravariance

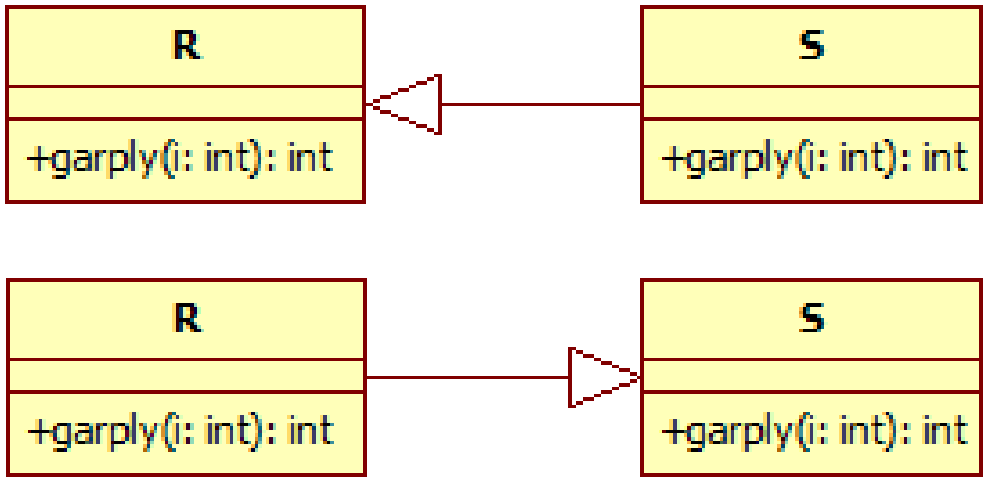


C# example:

```
// Contravariance.
// Assume that the following method is in the class:
// static void SetObject(object o) { }
Action<object> actObject = SetObject;
// An object that is instantiated with a less derived type argument
// is assigned to an object instantiated with a more derived type argument.
// Assignment compatibility is reversed.
Action<string> actString = actObject;
```

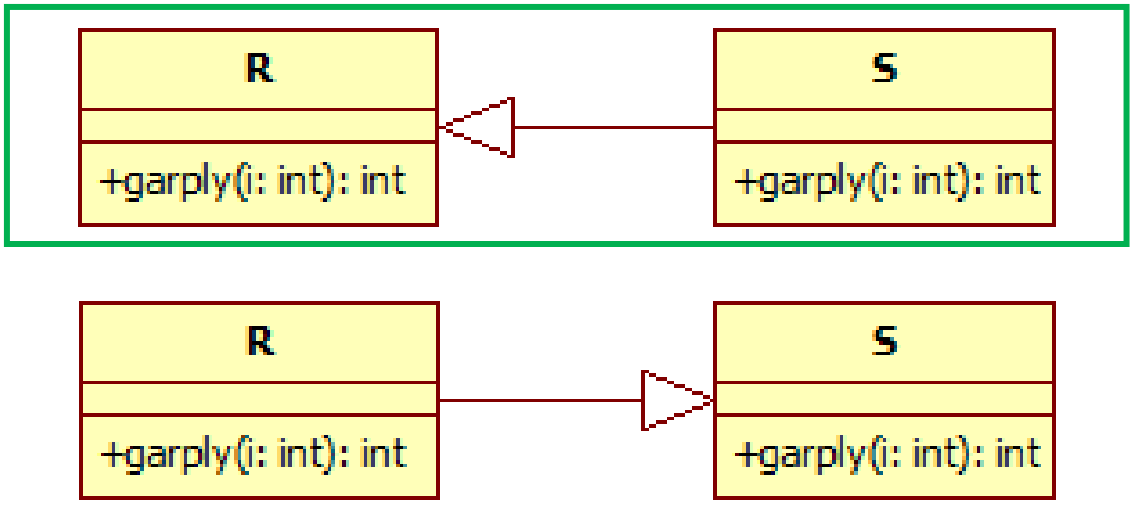
LSP and DbC

We know that between classes **R** and **S** there is inheritance, and they don't violate the Liskov principle. Draw the generalization arrow between them, if we know that the post-conditions of **S.garply()** are stronger than the post-conditions of **R.garply()**.



LSP and DbC

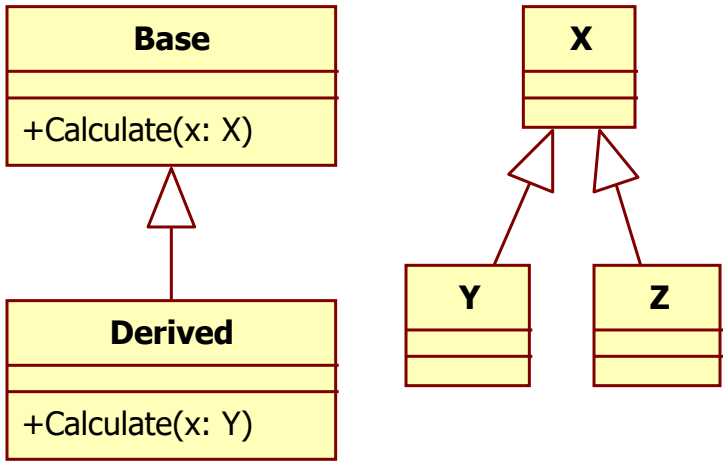
We know that between classes **R** and **S** there is inheritance, and they don't violate the Liskov principle. Draw the generalization arrow between them, if we know that the post-conditions of **S.garply()** are stronger than the post-conditions of **R.garply()**.



Solution: S is a descendant of R, since post-conditions can only be strengthened.

LSP and DbC

Is the LSP satisfied with the following conditions?

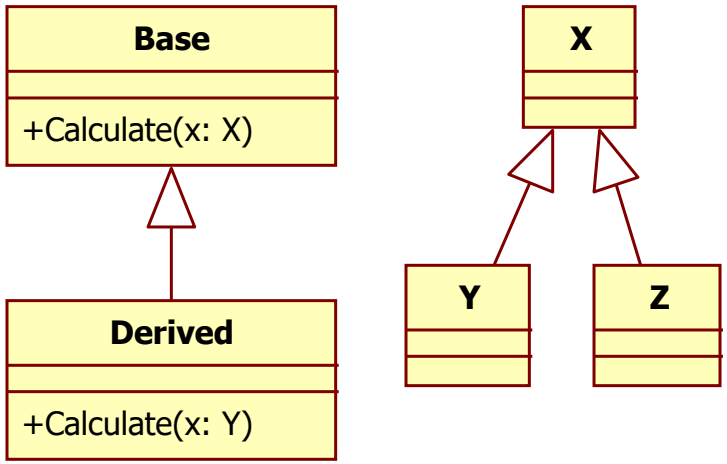


LSP and DbC

Is the LSP satisfied with the following conditions? **Solution:** Violated

Explanation 1: Accepted range (Y) is smaller than in base (X), pre-condition is weakened.

Explanation 2: A caller (client) object can safely pass X, Y or Z to Base, however, it cannot pass X or Z to Derived through Base.

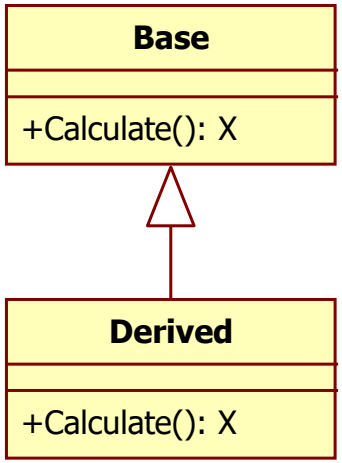


LSP and DbC

Is the LSP satisfied with the following conditions?

Base.Calculate post:
`X != null`

Derived.Calculate post:
`X may be null`



LSP and DbC

Is the LSP satisfied with the following conditions? **Solution:** Violated

Explanation 1: Returned range is wider than in base, post-condition is weakened.

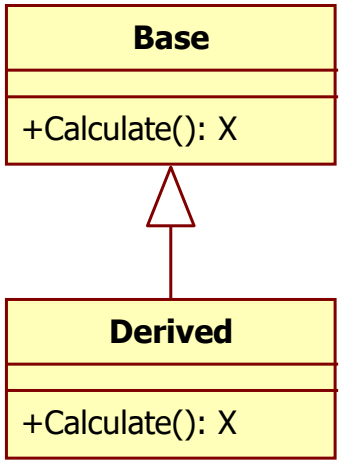
Explanation 2: A caller (client) object never expects null values from Base, however, Derived may return nulls through Base.

Base.Calculate post:

```
X != null
```

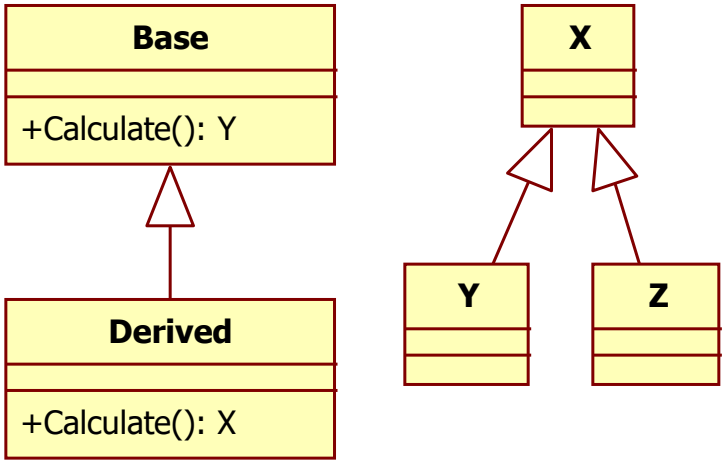
Derived.Calculate post:

```
X may be null
```



LSP and DbC

Is the LSP satisfied with the following conditions?

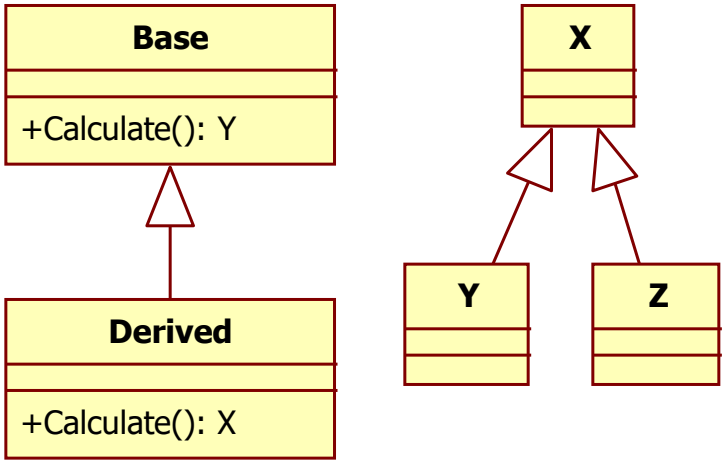


LSP and DbC

Is the LSP satisfied with the following conditions?

Solution: Violated

Explanation: Post-condition is weakened.



LSP and DbC

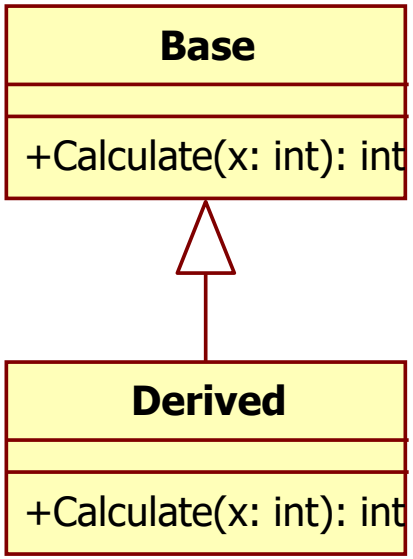
Is the LSP satisfied with the following conditions?

Base.Calculate post:

```
0 <= result <= 100
```

Derived.Calculate post:

```
0 <= result <= 1000
```



LSP and DbC

Is the LSP satisfied with the following conditions?

Solution: Violated

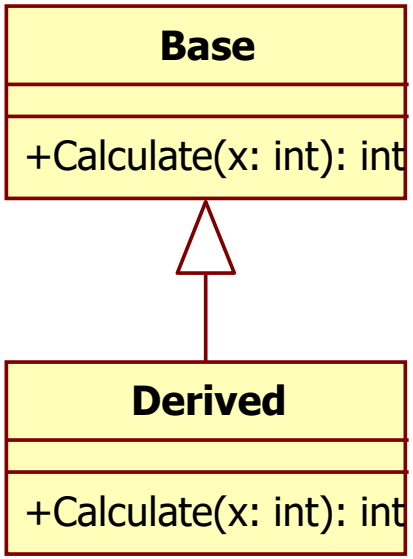
Explanation: Post-condition is weakened.

Base.Calculate post:

```
0 <= result <= 100
```

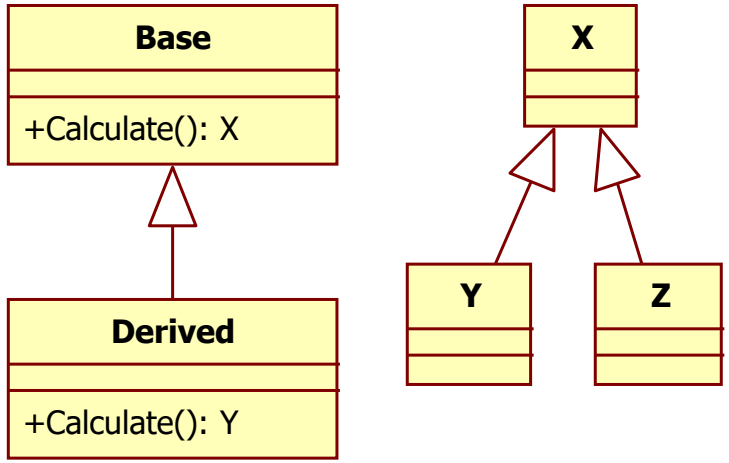
Derived.Calculate post:

```
0 <= result <= 1000
```



LSP and DbC

Is the LSP satisfied with the following conditions?

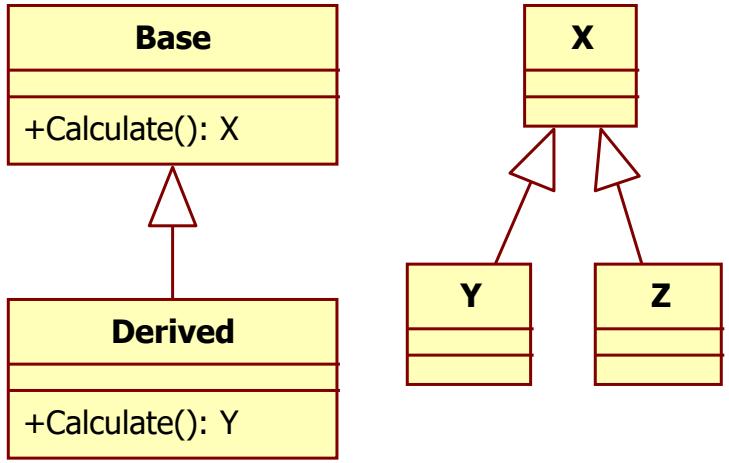


LSP and DbC

Is the LSP satisfied with the following conditions? **Solution:** Satisfied

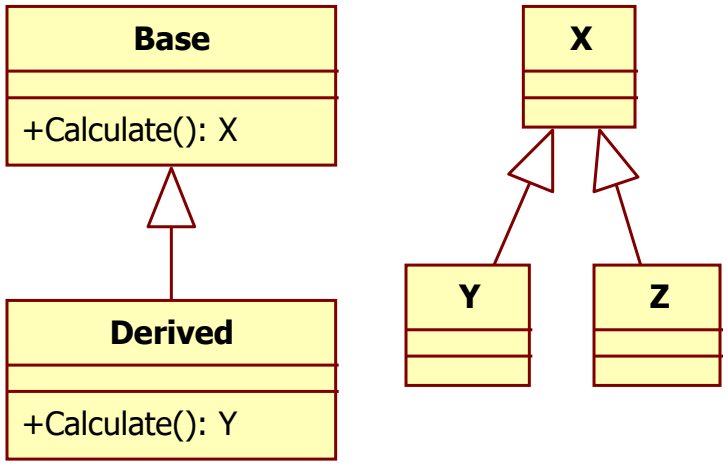
Explanation 1: Returned range (Y) is smaller than in base (X), post-condition is strenghtened.

Explanation 2: A caller (client) object expects X from Base. Derived called through Base, only returns Y values, which are still X values, and are perfectly fine for the client.



This even has a name: covariance

Covariance



C# example:

```
// Covariance.
IEnumerable<string> strings = new List<string>();
// An object that is instantiated with a more derived type argument
// is assigned to an object instantiated with a less derived type argument.
// Assignment compatibility is preserved.
IEnumerable<object> objects = strings;
```

LSP and DbC

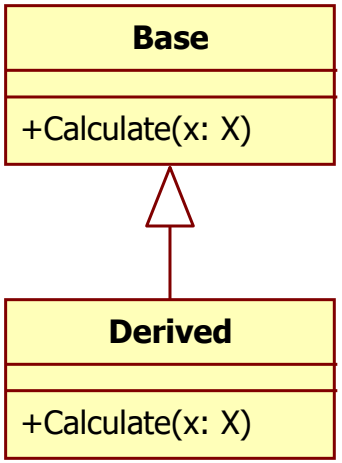
Is the LSP satisfied with the following conditions?

Base.Calculate pre:

```
x != null
```

Derived.Calculate pre:

```
x may be null
```



LSP and DbC

Is the LSP satisfied with the following conditions?

Solution: Satisfied

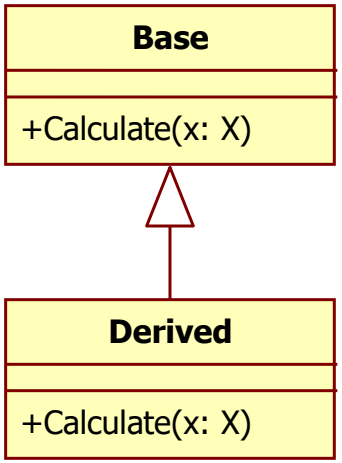
Explanation: Pre-condition is weakened.

Base.Calculate pre:

```
x != null
```

Derived.Calculate pre:

```
x may be null
```



LSP and DbC

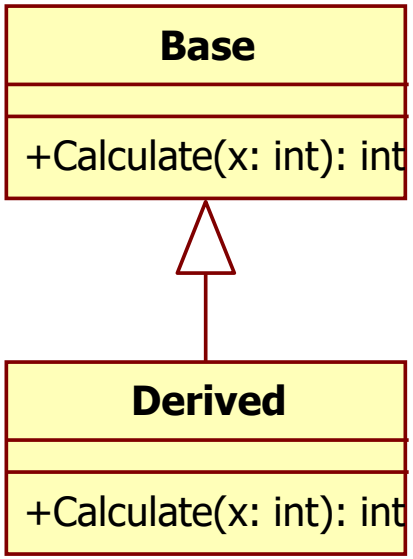
Is the LSP satisfied with the following conditions?

Base.Calculate post:

```
0 <= result <= 100
```

Derived.Calculate post:

```
20 <= result <= 50
```



LSP and DbC

Is the LSP satisfied with the following conditions?

Solution: Satisfied

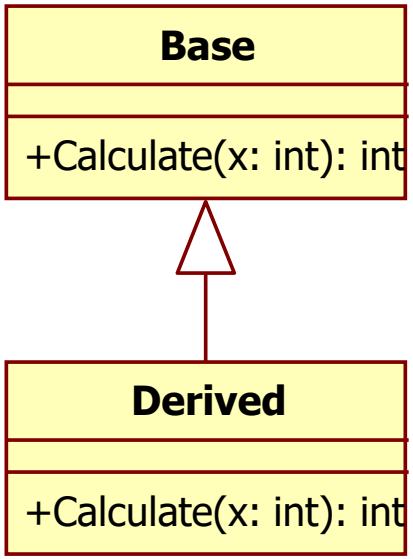
Explanation: Post-condition is strengthened.

Base.Calculate post:

```
0 <= result <= 100
```

Derived.Calculate post:

```
20 <= result <= 50
```

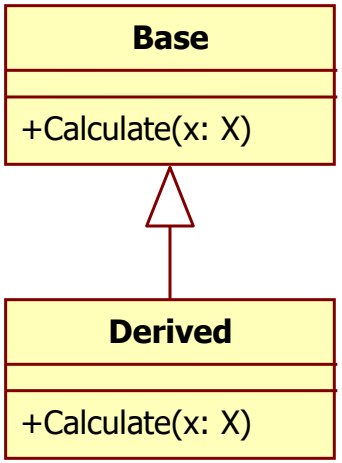


LSP and DbC

Is the LSP satisfied with the following conditions?

Base.Calculate pre:
`x may be null`

Derived.Calculate pre:
`x != null`



LSP and DbC

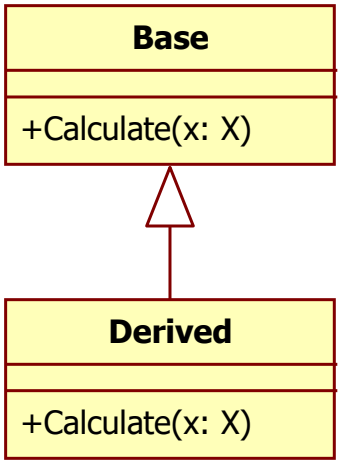
Is the LSP satisfied with the following conditions?

Solution: Violated

Explanation: Pre-condition is strengthened.

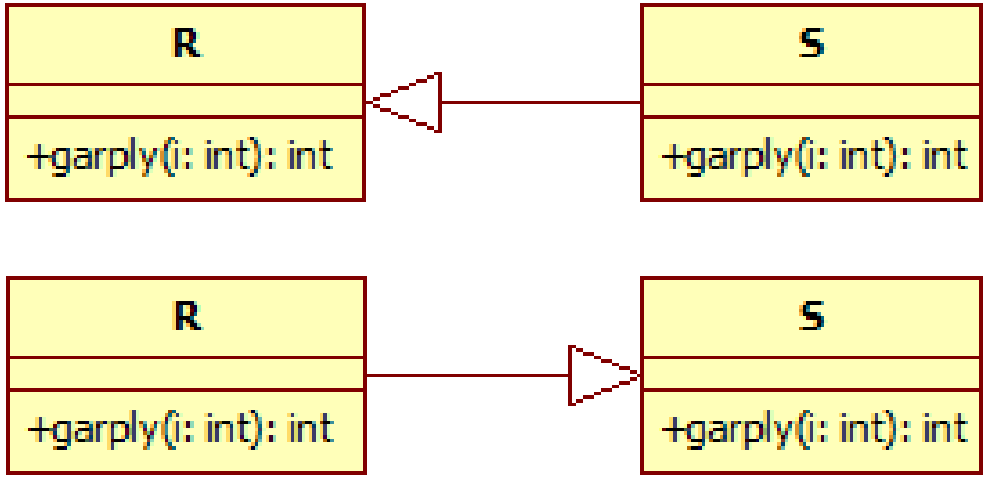
Base.Calculate pre:
`x may be null`

Derived.Calculate pre:
`x != null`



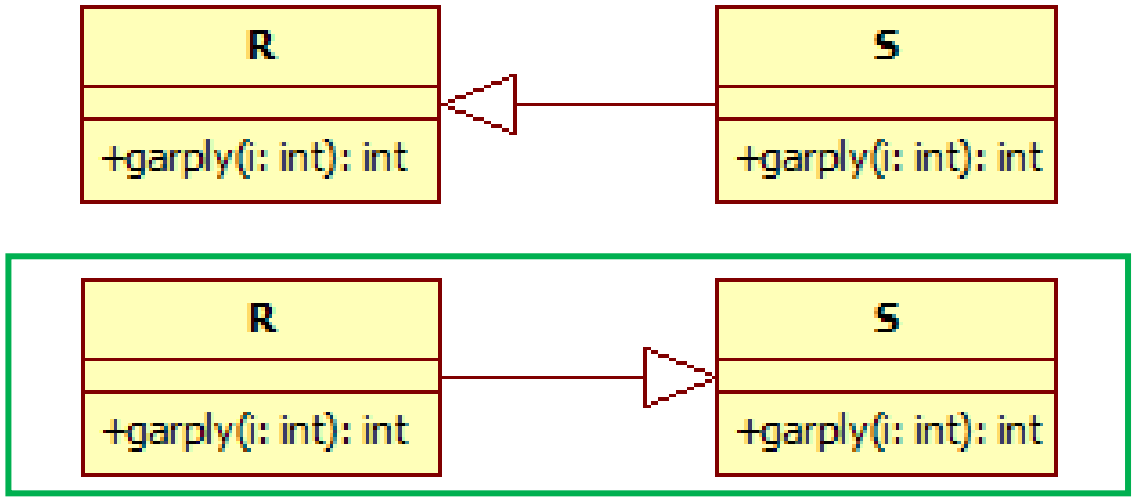
LSP and DbC

We know that between classes **R** and **S** there is inheritance, and they don't violate the Liskov principle. Draw the generalization arrow between them, if we know that the pre-conditions of **S.garply()** are stronger than the pre-conditions of **R.garply()**.



LSP and DbC

We know that between classes **R** and **S** there is inheritance, and they don't violate the Liskov principle. Draw the generalization arrow between them, if we know that the pre-conditions of **S.garply()** are stronger than the pre-conditions of **R.garply()**.

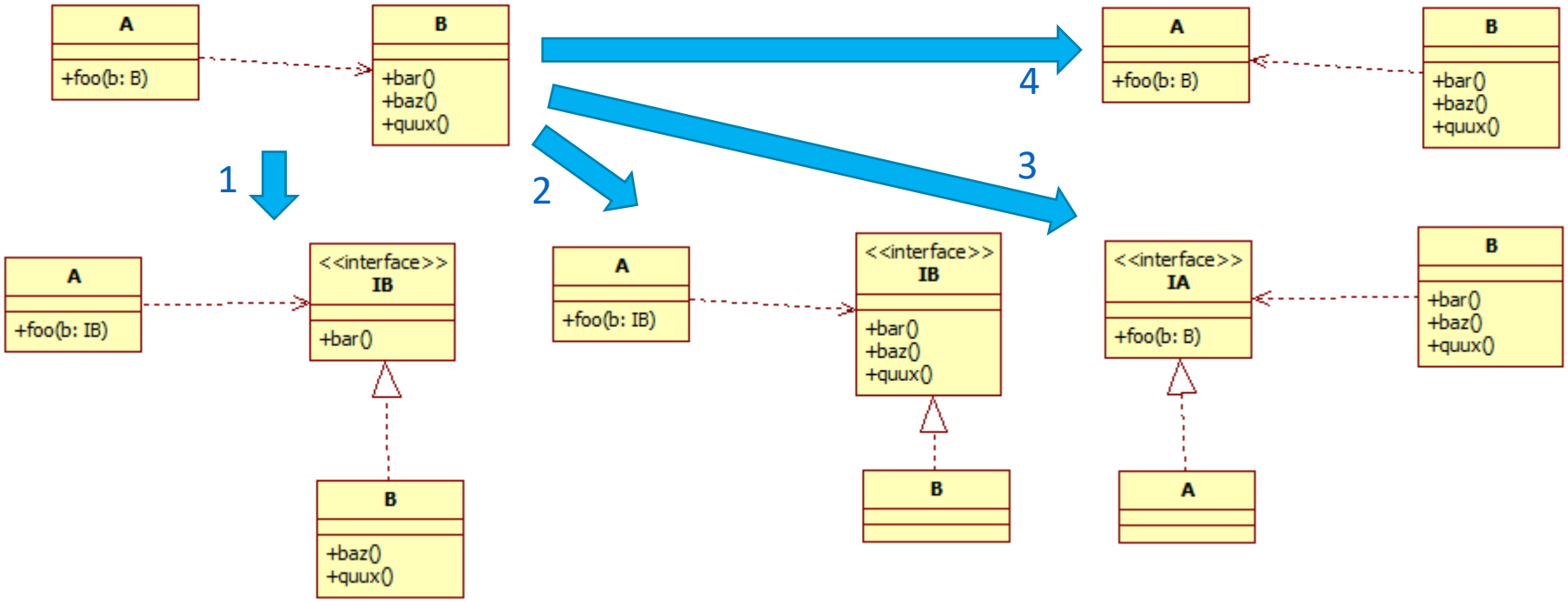


Solution: R is a descendant of S, since pre-conditions can only be weakened.

DIP and ISP

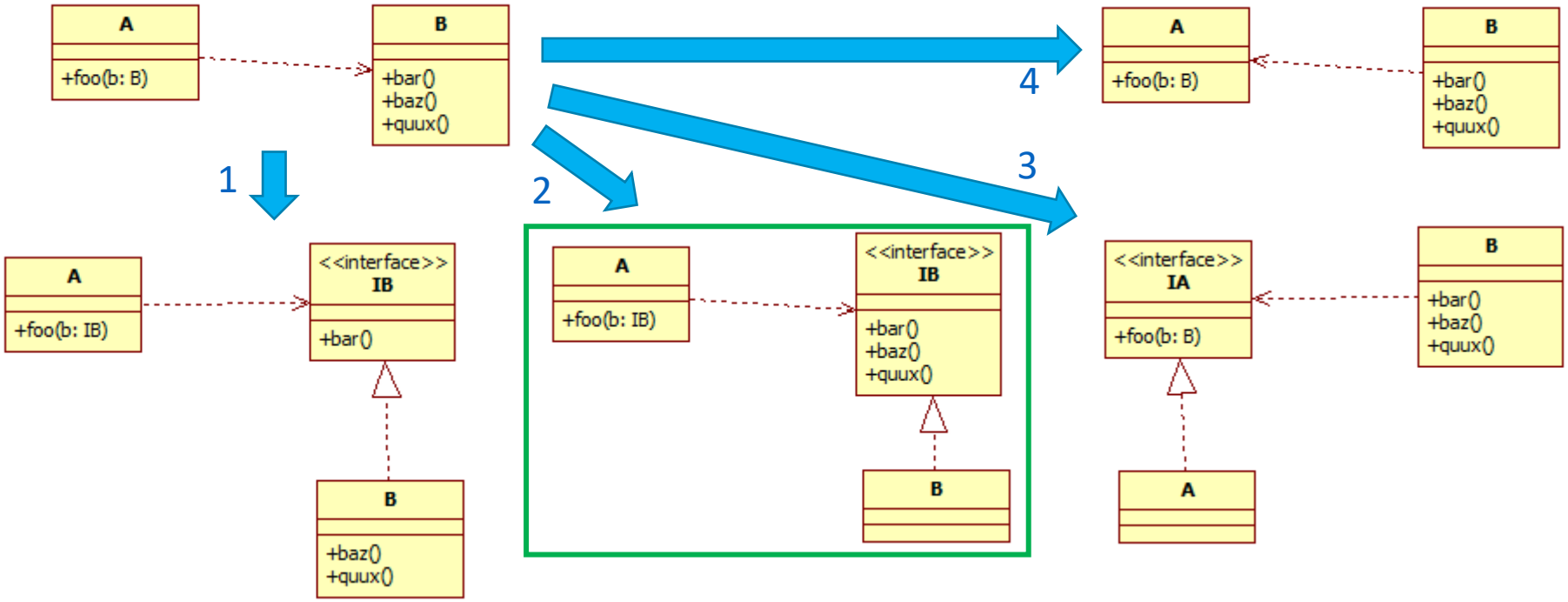
DIP

Redraw the following class diagram according to the DIP!



DIP

Redraw the following class diagram according to the DIP! **Solution:** 2



Explanation:

4 is clearly wrong, A still has to call B. Redrawing the arrow on the diagram won't help.

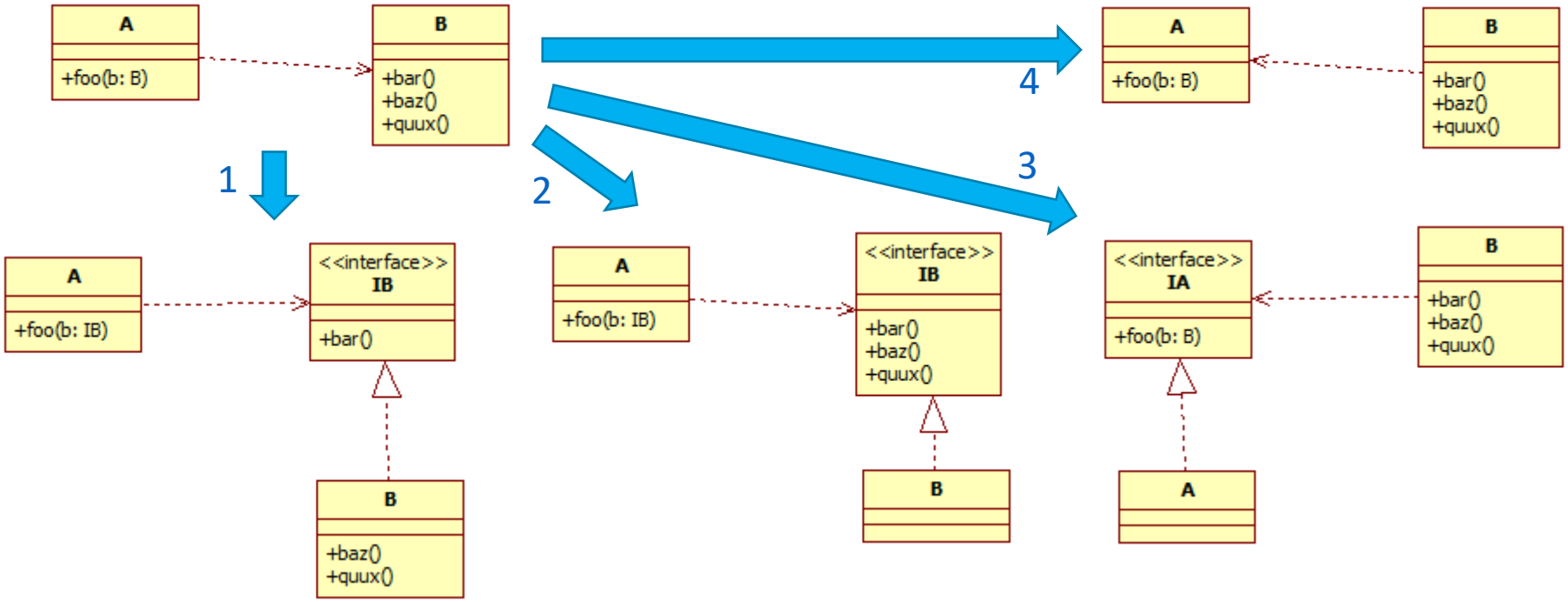
3 is also wrong, since A has to call B.

Either 1 or 2 is the correct one. Since we don't have more information, we don't know which methods are called from B by A, we assume all of them, hence the solution is 2.

ISP

Redraw the following class diagram according to the ISP,
if we know that the implementation of A is the following:

```
public class A { public void foo(B b) { b.bar(); }}
```

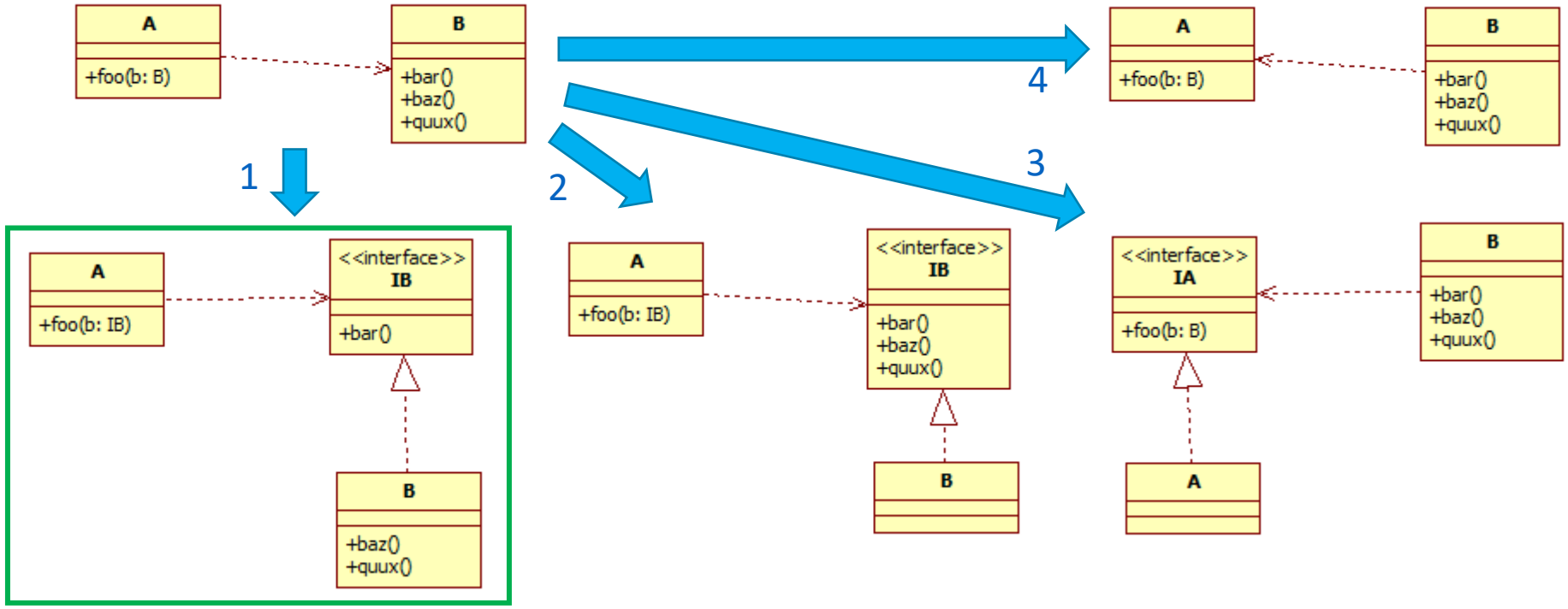


ISP

Redraw the following class diagram according to the ISP,
if we know that the implementation of A is the following:

```
public class A { public void foo(B b) { b.bar(); }}
```

Solution: 1



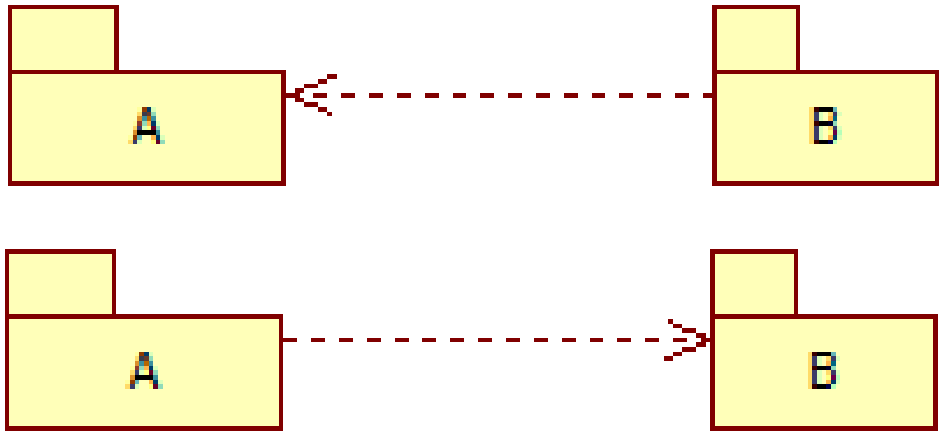
Explanation:

3 and 4 are clearly wrong. A calls B, and not the other way around.
Either 1 or 2 is the correct one. Since now we know exactly which methods are called from B by A, the solution is 2, since A should only depend on the methods it actually uses.
Note: DIP and ISP result in the exact same transformation, only the philosophy behind the principles are different.

SDP

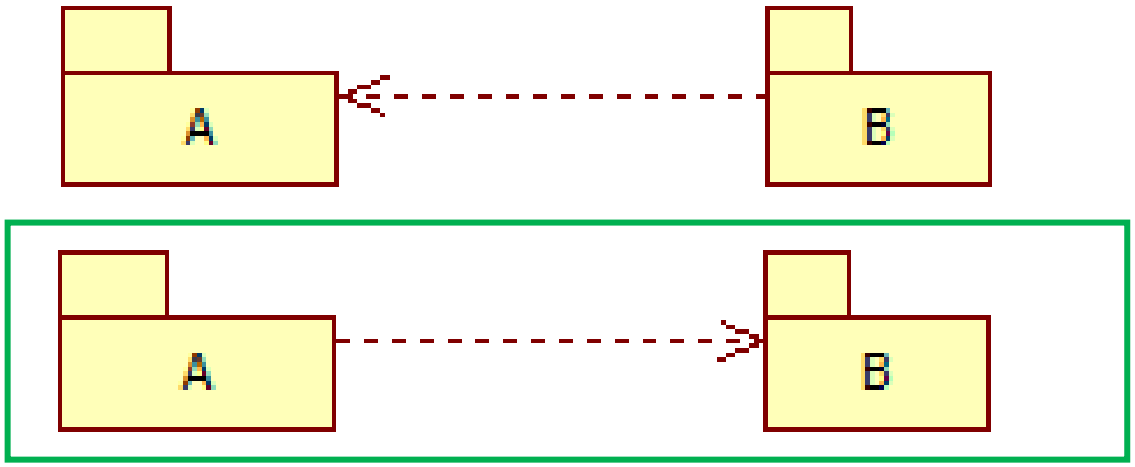
SDP

Package **A** changes frequently, while package **B** changes rarely.
Draw the dependency between them according to the SDP!



SDP

Package **A** changes frequently, while package **B** changes rarely.
Draw the dependency between them according to the SDP!



Solution: A depends on B, since B is the stable package.

True-false

True or false

Is the following statement true or false?

The goal of inheritance is the reuse of fields from the base class.

True or false

Is the following statement true or false?

Solution: False. The goal of inheritance is behavior reuse.

The goal of inheritance is the reuse of fields from the base class.

True or false

Is the following statement true or false?

If a class violates SRP, and it cannot be split into multiple classes due to its complexity, then the ISP can help to minimize dependencies.

True or false

Is the following statement true or false?

Solution: True

If a class violates SRP, and it cannot be split into multiple classes due to its complexity, then the ISP can help to minimize dependencies.

True or false

Is the following statement true or false?

A software is well designed if the stable packages depend on instable ones.

True or false

Is the following statement true or false?

Solution: False. Instable packages should depend on stable ones.

A software is well designed if the stable packages depend on instable ones.

True or false

Is the following statement true or false?

High cohesion and low coupling is good for software maintenance.

True or false

Is the following statement true or false?

Solution: True

High cohesion and low coupling is good for software maintenance.

True or false

Is the following statement true or false?

The goal of inheritance is the
reuse of behavior from the base class.

True or false

Is the following statement true or false?

Solution: True

The goal of inheritance is the
reuse of behavior from the base class.

True or false

Is the following statement true or false?

According to the YAGNI principle,
we do not need object-oriented design principles.

True or false

Is the following statement true or false?

Solution: False. YAGNI means that we should not design for changes that are very improbable, i.e., we should not overdesign the software.

According to the YAGNI principle,
we do not need object-oriented design principles.

True or false

Is the following statement true or false?

The software must always satisfy all the object-oriented design principles.

True or false

Is the following statement true or false?

Solution: False. Usually they cannot all be satisfied. We have to try to satisfy them, but if we have a good reason, we can break them. There are even design patterns that violate some of the SOLID principles. See later.

The software must always satisfy all the object-oriented design principles.

SOLID

SOLID

Which of the following design principles belong to the SOLID principles?

Liskov Substitution Principle
Interface Segregation Principle
Tell, Don't Ask
Dependency Inversion Principle
Acyclic Dependencies Principle
Stable Dependencies Principle
Single Choice Principle
Don't Repeat Yourself
Law of Demeter
Open/Closed Principle
Single Responsibility Principle
Stable Abstractions Principle

SOLID

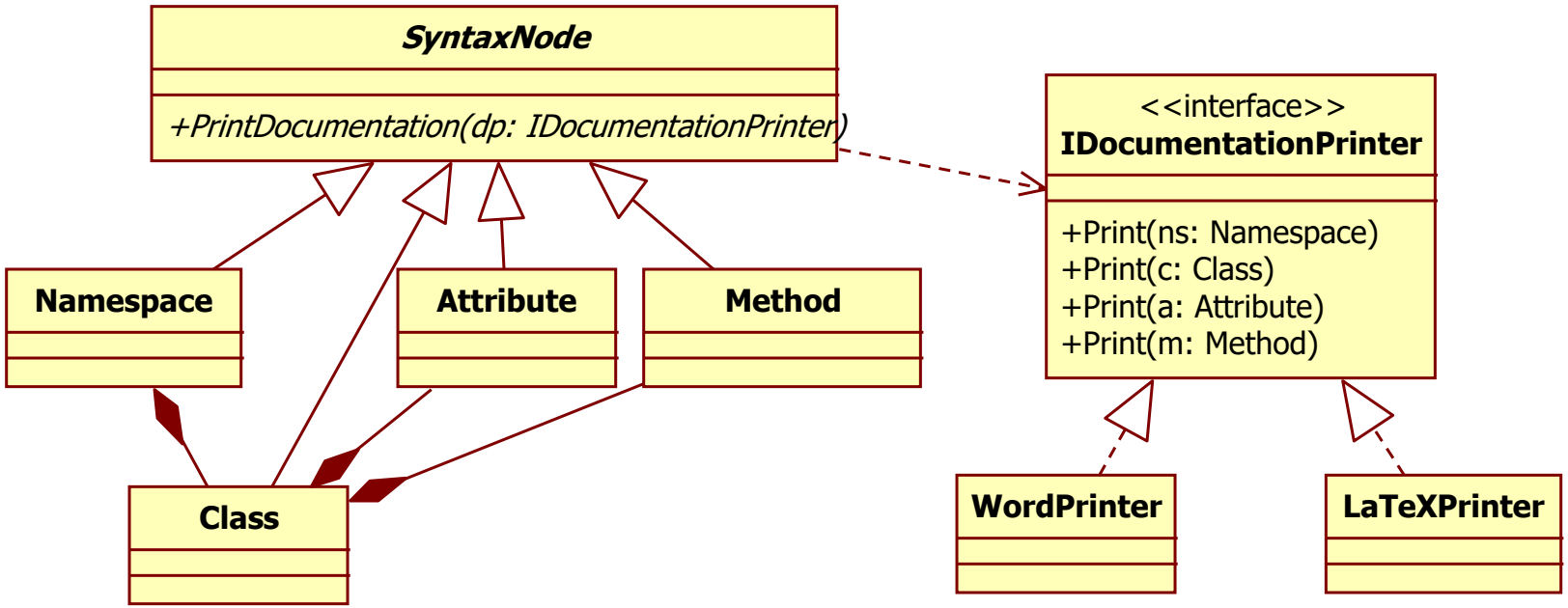
Which of the following design principles belong to the SOLID principles?

Solution:

Liskov Substitution Principle
Interface Segregation Principle
Tell, Don't Ask
Dependency Inversion Principle
Acyclic Dependencies Principle
Stable Dependencies Principle
Single Choice Principle
Don't Repeat Yourself
Law of Demeter
Open/Closed Principle
Single Responsibility Principle
Stable Abstractions Principle

Does the Visitor design pattern satisfy the ISP?

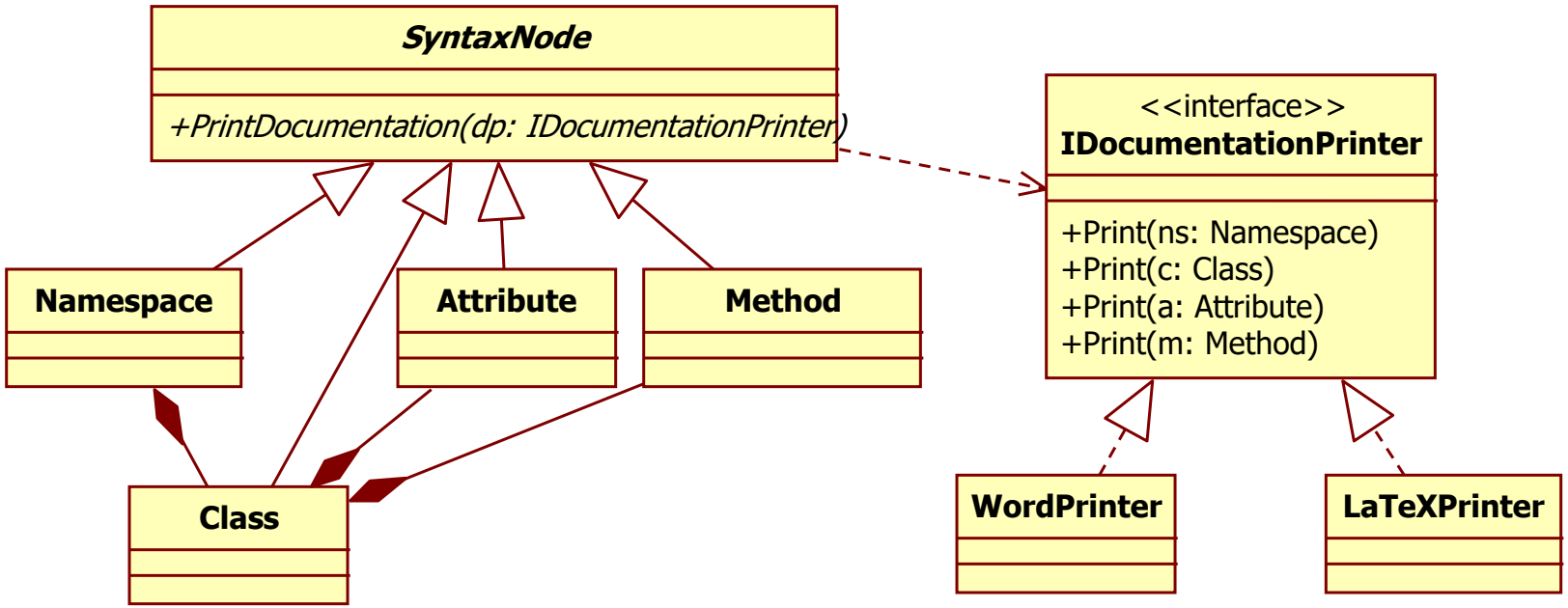
(For more explanation about the Visitor design pattern see: https://refactoring.guru/design-patterns/visitor)



SOLID

Does the Visitor design pattern satisfy the ISP?

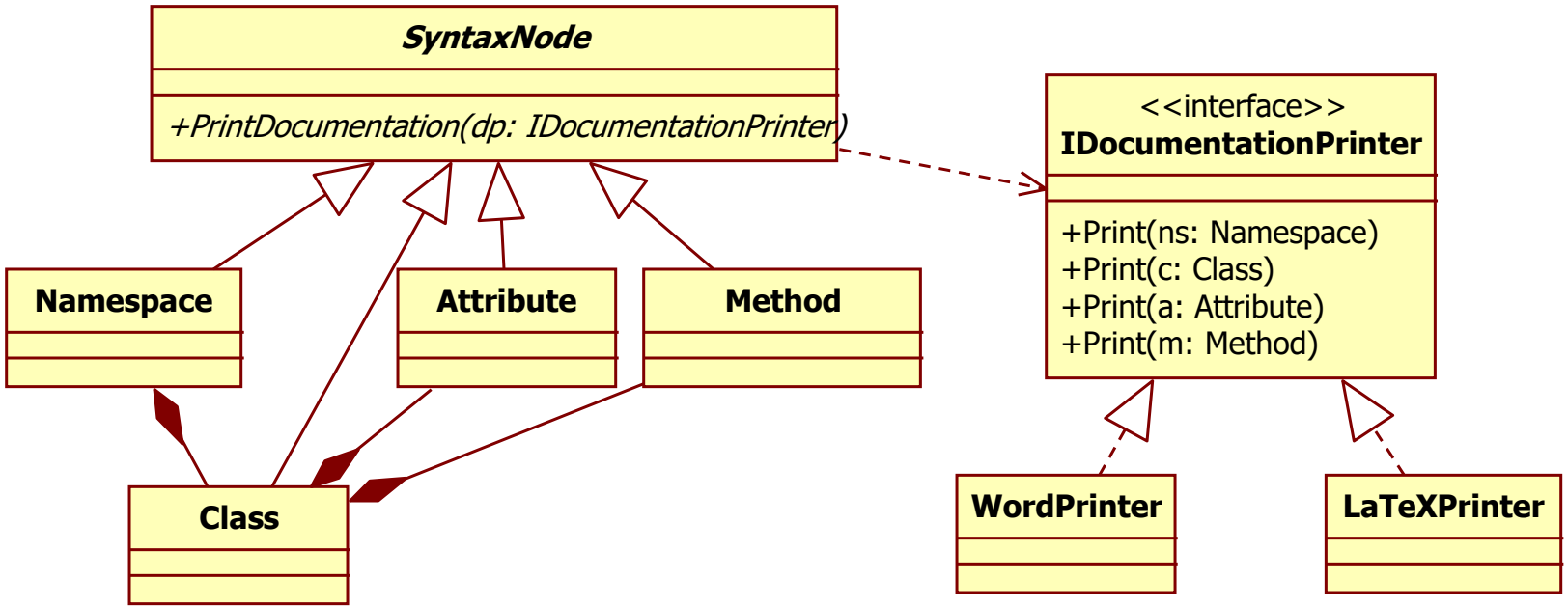
Solution: No. Namespace calls only Print(ns:Namespace), and not the other Print methods. Class calls only Print(c:Class), and not the others. etc.
But Namespace, Class, etc. depend on an interface (IDocumentation-Printer) that lists other Print methods, too.



SOLID

Does the Visitor design pattern satisfy the OCP?

(For more explanation about the Visitor design pattern see: https://refactoring.guru/design-patterns/visitor)



SOLID

Does the Visitor design pattern satisfy the OCP?

Solution: No. If a new descendant is added to the left side of the diagram, i.e., a new descendant to SyntaxNode, Namespace or Class, etc., then that new descendant's corresponding Print method must be added to the existing IDocumentationPrinter interface, and also to the classes that implement this interface. We have to change existing code, which is forbidden by OCP.