



SOLID Objects

Single Responsibility Principle



Single Responsibility Principle

Just because you *can* doesn't mean you *should*.

Single Responsibility Principle

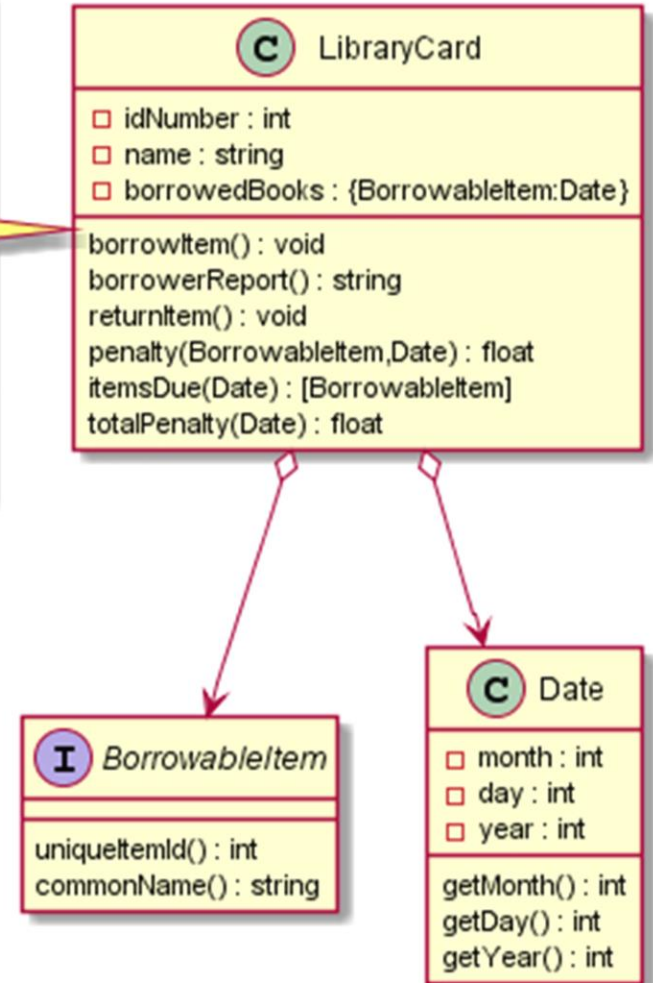
Objects should have cohesive and complete responsibilities.

It shouldn't be aware of knowledge it doesn't need and it shouldn't perform responsibilities that are irrelevant from it.

GOD Class (violation)

```
penalty(b:BorrowableItem, d:Date) -> float:
```

```
...  
day d = (d.getDay() +  
d.getMonth()*30 +  
d.getYear()* 360)  
...  
if b is a Book:  
    ...something book related  
else if b is a Periodical:  
    ...something periodical related  
...
```



Assigning the correct responsibilities

When you're introducing new behavior or information to a system, you should ask first: who should be responsible of this behavior or information?

- Who should be responsible of calculating the differences between dates? **Date should be responsible**, not LibraryCard.
- Who should be responsible of calculating the penalties of specific BorrowableItem realizations? **BorrowableItem realizations should be responsible**, not LibraryCard

Open/Close Principle

Classes should be open to extension and closed to modification.

Instead of changing the form and behavior of an existing class, you should extend the class

Open/Close Principle

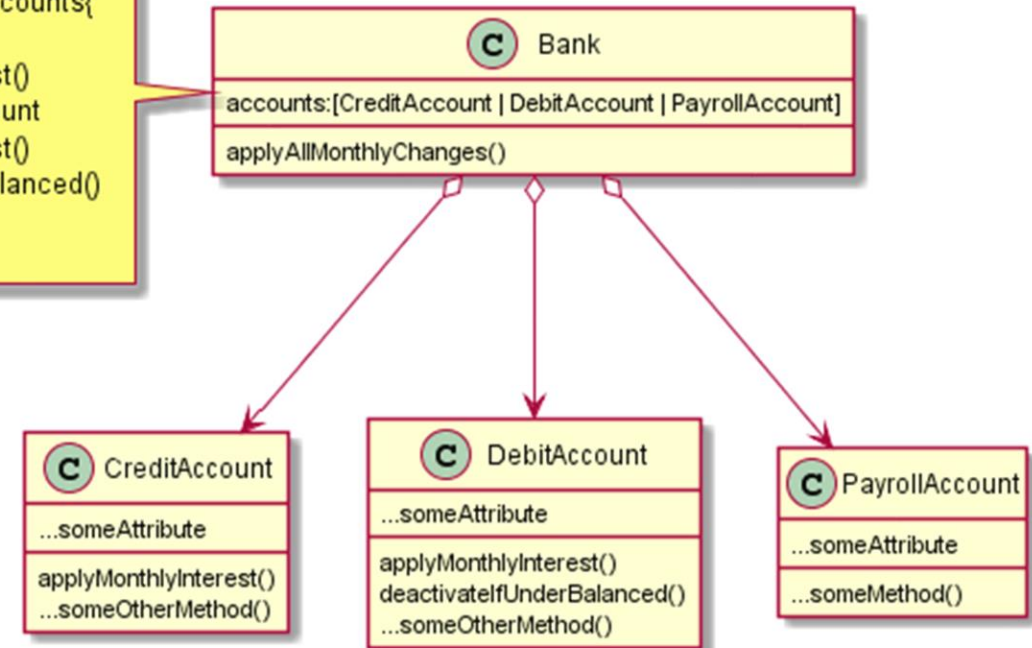


Open-Closed Principle

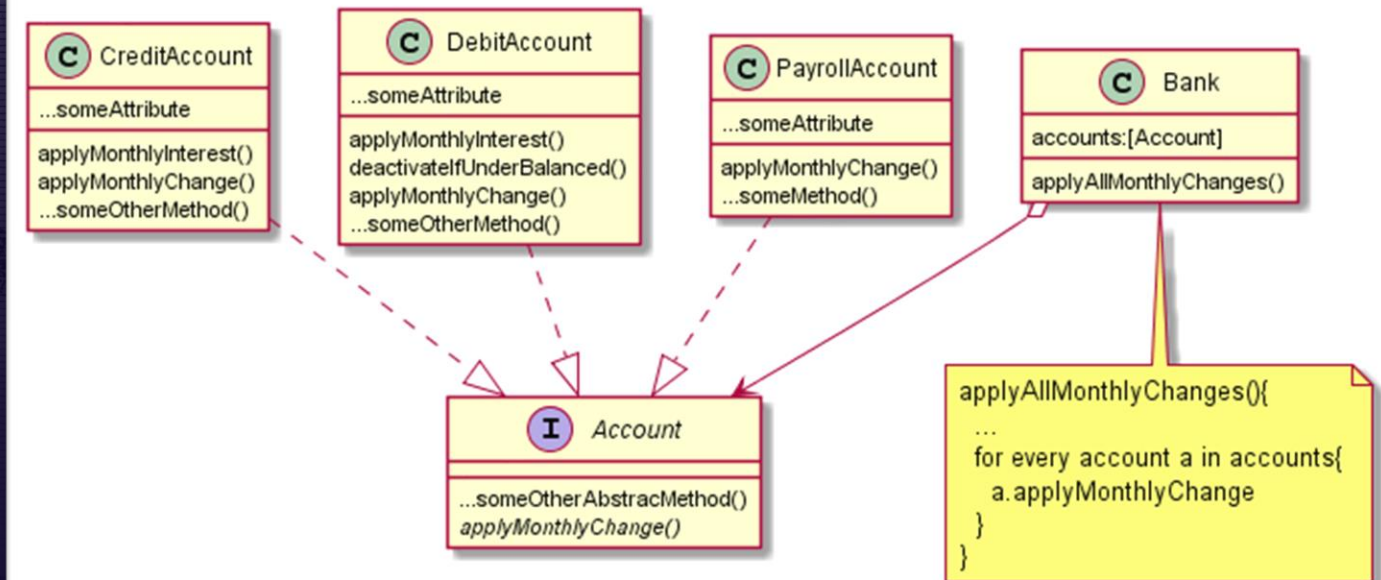
Open-chest surgery isn't needed when putting on a coat.

Open for Modification (violation)

```
applyAllMonthlyChanges(){  
  ...  
  for every account a in accounts{  
    if a is a CreditAccount  
      a.applyMonthlyInterest()  
    else if a is a DebitAccount  
      a.applyMonthlyInterest()  
      a.deactivateIfUnderBalanced()  
    }  
  }  
}
```



Open for Extension

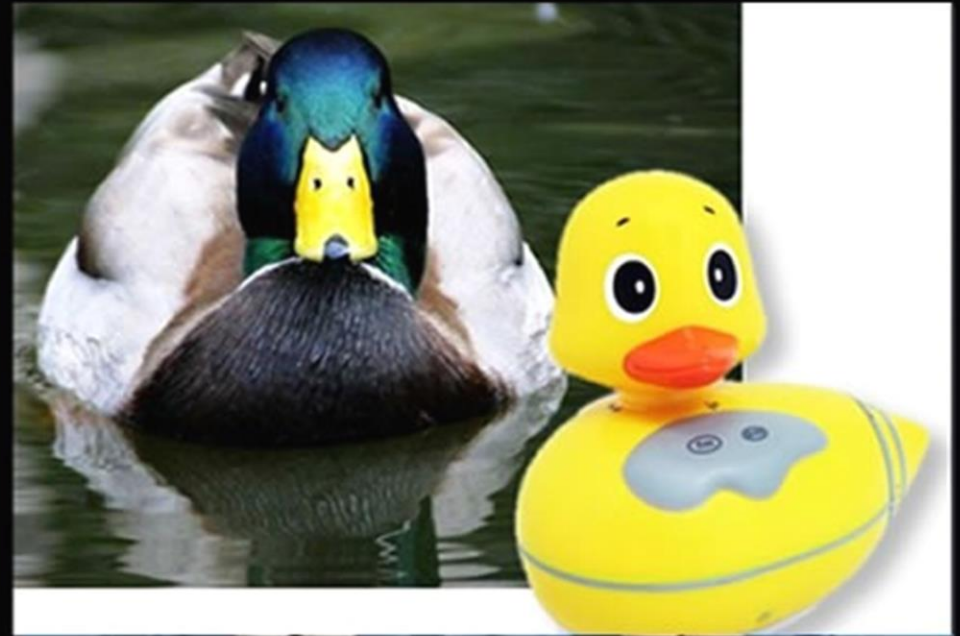




Liskov- Substitution Principle


Substituting objects by their subtypes/realizations should always work.

Liskov Substitution Principle



Liskov Substitution Principle

If it looks like a duck and quacks like a duck but needs batteries, you probably have the wrong abstraction.



*Which
relationship is
better?
Realization or
Specialization*

The background of the slide is a dark, atmospheric photograph of a forest. A large, leafy tree dominates the right side of the frame. In the lower-left background, a small, simple hut or cabin is visible, partially obscured by the trees. The overall lighting is dim, creating a moody and naturalistic setting.

Interface Segregation Principle

A client shouldn't be forced to implement methods that it doesn't use.

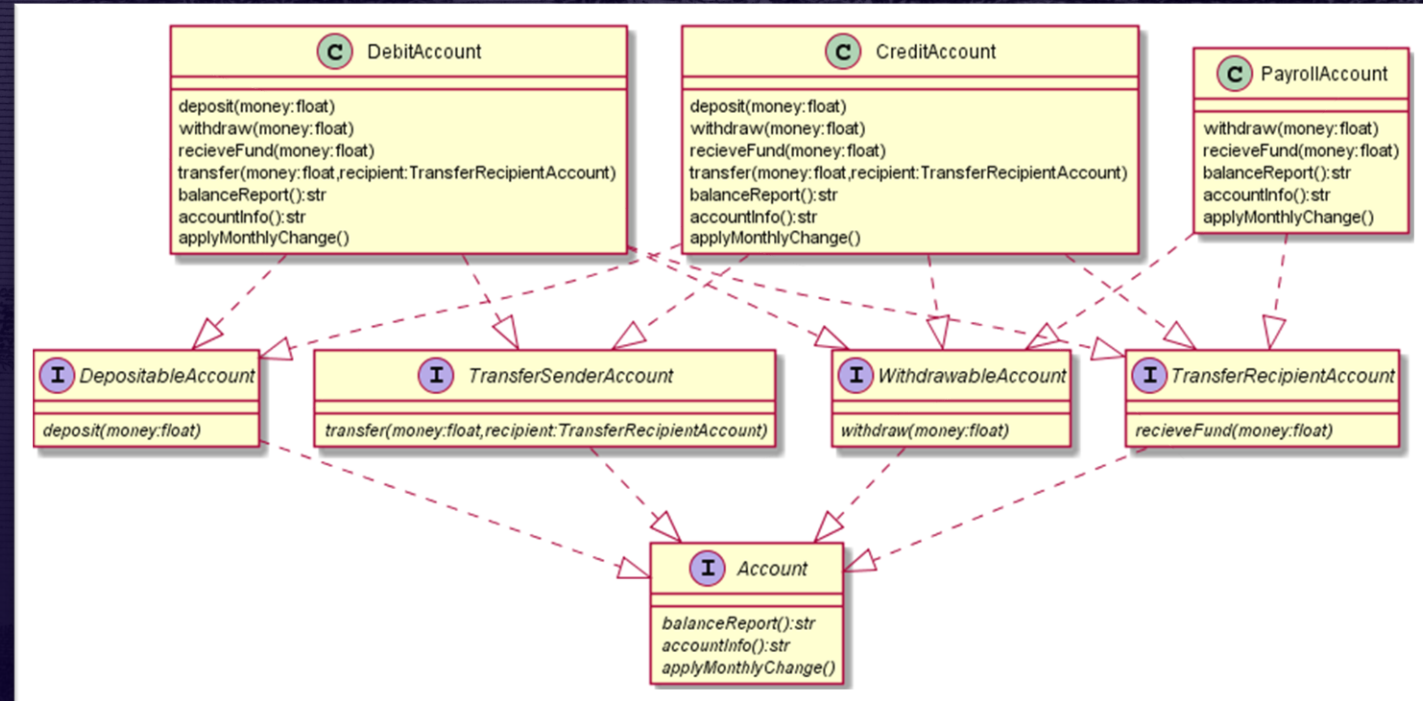
Interface Segregation Principle

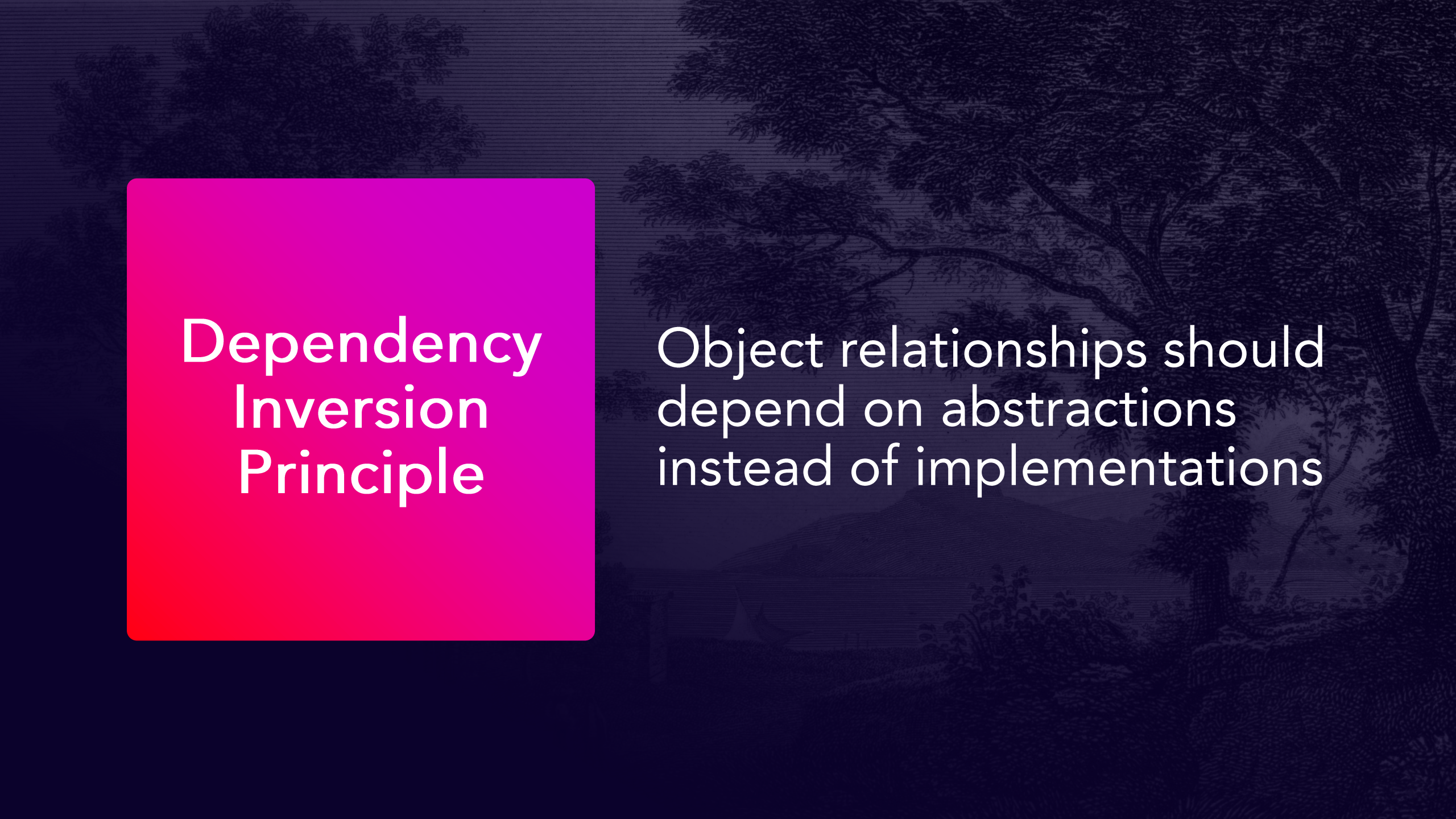


Interface Segregation Principle

You want me to plug this in *where*?

Role Interfaces



The background of the slide is a dark, atmospheric photograph of a forest. In the lower-left, a small, rustic hut with a thatched roof is nestled among trees. To the right, a waterfall is visible, cascading down a rocky ledge. The overall scene is dimly lit, with a blueish-purple tint, creating a serene and somewhat mysterious mood.

Dependency Inversion Principle

Object relationships should depend on abstractions instead of implementations

Dependency Inversion Principle



Dependency Inversion Principle

Would you solder a lamp directly
to the electrical wiring in a wall?

References

Motivational Pictures Reference

Bailey D. (2009) SOLID Development Principles – In Motivational Pictures. Retrieved from <https://lostechies.com/derickbailey/2009/02/11/solid-development-principles-in-motivational-pictures/>



SOLID Motivational Posters, by [Derick Bailey](#), is licensed under a [Creative Commons Attribution-Share Alike 3.0 United States License](#).

<http://creativecommons.org/licenses/by-sa/3.0/us/>