Evolving Code with the Open Closed Principle (OCP)

Dan Geabunea
PASSIONATE SOFTWARE DEVELOPER | BLOGGER

@romaniancoder www.romaniancoder.com

Overview

What is the Open Closed Principle (OCP)?

Demo: Add a new software feature without applying the OCP

OCP implementation strategies

Demo: Add a new software feature using OCP

Applying OCP for frameworks and APIs

Open Closed Principle

Classes, functions, and modules should be closed for modification, but open for extension.

OCP Definition Explained

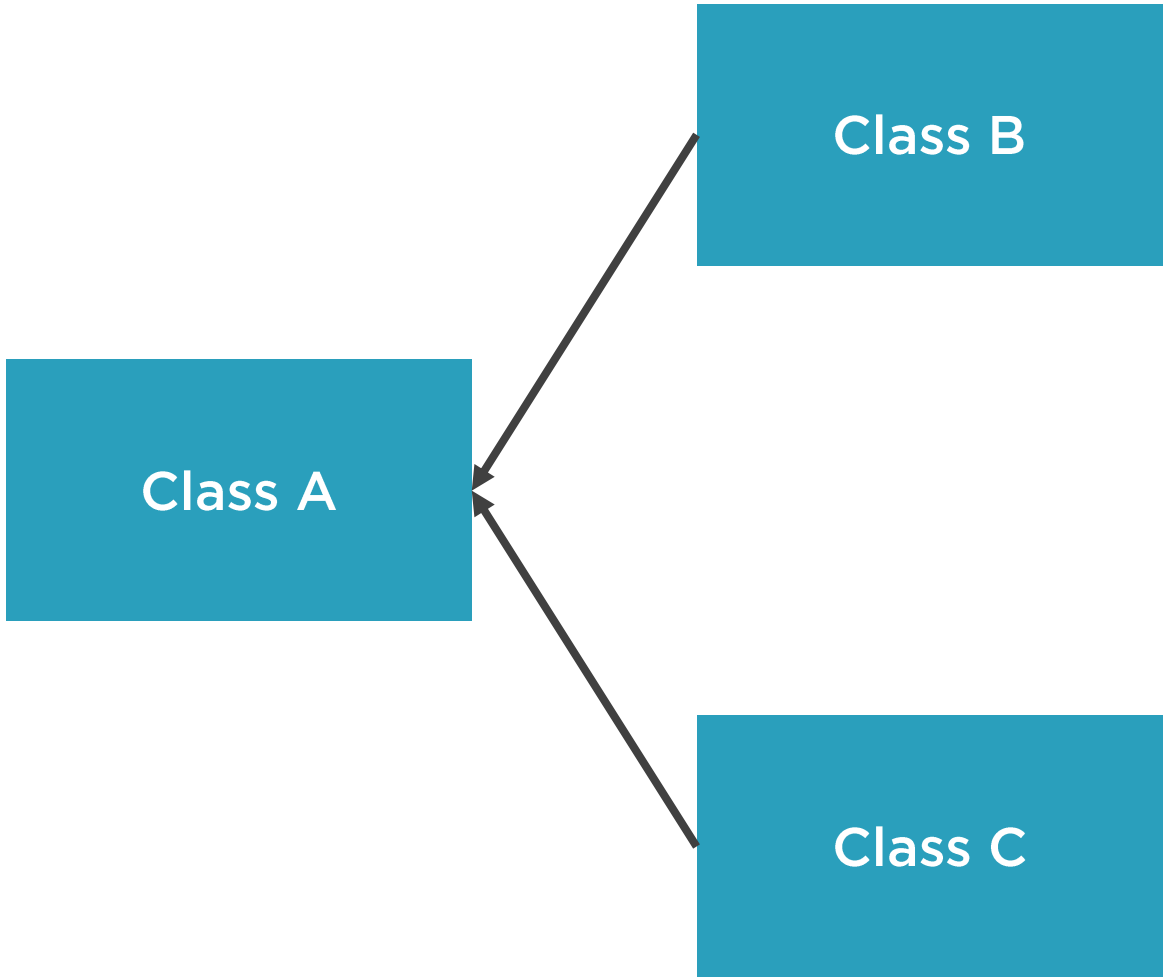
Closed for modification

Each new feature should not modify existing source code

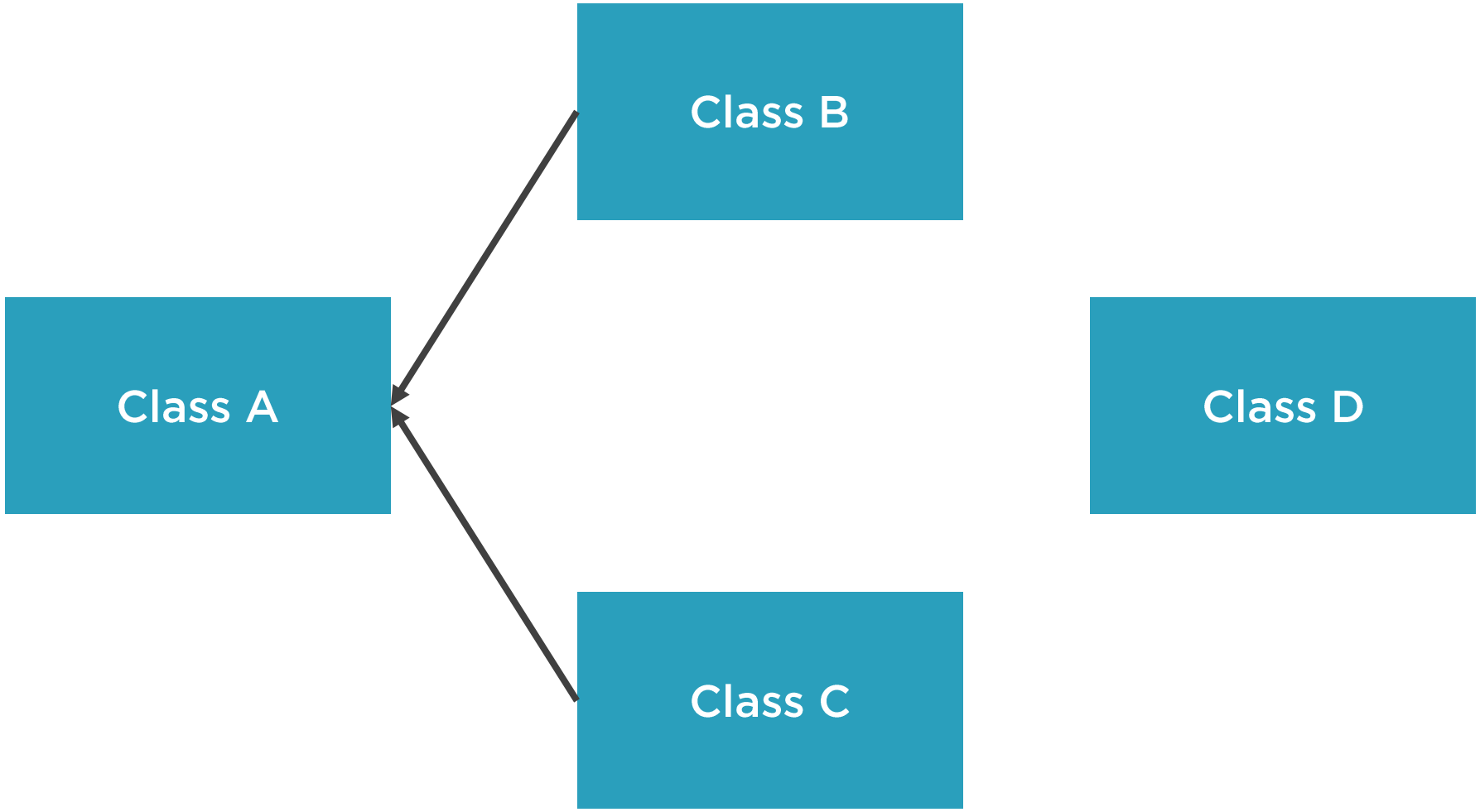
Open for extension

A component should be extendable to make it behave in new ways

Modifying Existing Code Is a Risk



Modifying Existing Code Is a Risk



Why Should You Apply the OCP?

New features can be added easily and with minimal cost

Minimizes the risk of regression bugs

Enforces decoupling by isolating changes in specific components, works along with the SRP

SOLID principles are most effective when applied together.

Demo

Add a new software feature without applying the Open Closed Principle

- Observe how easy it is to violate this principle
- Downside of adding features by modifying existing components

OCP Implementation Strategies

A Classic Example

```
public class BankAccount {
    . . .
    void transferMoney(double amount){
        // business logic for local transfer
     }
}
```

Inheritance

```
public class InternationalBankAccount extends BankAccount{
    . . .
    @Override
    void transferMoney(double amount){
        // business logic for international transfer
    }
}
```

Strategy Pattern

```
// Money Transfer Processor
public interface MoneyTransferProc {
    public void transferMoney(double amount);
}
```

Strategy Pattern

```
public class BankAccount implements MoneyTransferProc {
    public void transferMoney(double amount){…}
}

public class IntlBankAccount implements MoneyTransferProc {
        public void transferMoney(double amount){…}
}
```

Strategy Pattern

```
public class MoneyTransferProcessorFactory {
    public void MoneyTransferProc build(TransferType type){
        if(type == TransferType.Local){
            return new BankAccount();
        } else if(type == TransferType.Intl){
            return new IntlBankAccount();
        }
    }
}
```

Strategy Pattern

```
void processPayment(double amount, TransferType type){
    …
    MoneyTransferProc mtp = factory.build(type);
    mtp.transferMoney(amount);
}
```

Progressively Applying the OCP

Start small

Make changes inline

Bug fixes can be implemented this way

More changes

Consider Inheritance

Many changes / dynamic decision

Consider interfaces and design patterns like Strategy

Demo

Add a new software feature using the Open Closed Principle

- Apply the Strategy Pattern

Applying OCP for Frameworks and APIs

API

A contract/agreement between different software components on how they should work together.

A public framework or API is under your control.
But clients might use it in ways that you aren't aware of.

Exposed Framework / SDK

```
public class TaxCalculator {
    public double calculate(Employee e){
        // business logic
    }
}
```

Changes Can Break the Clients Implementations

```
public class TaxCalculator {
    public double calculate(Employee e, String currency){
        // business logic
    }
}
```

Make Framework/SDK Open for Modification

```
public interface AbstractTaxCalculator{
    public double calculate(Employee e);
}


// implemented by customer
class CustomerUSACalc implements AbstractTaxCalculator {
    public double calculate(Employee e){
        // business logic
    }
}
```

Best Practices for Changing APIs

Do not change existing public contracts: data classes, signatures

Expose abstractions to your customers and let them add new features on top of your framework

If a breaking change is inevitable, give your clients time to adapt

Summary

- Changing requirements are inevitable
- Inheritance and design patterns are effective ways to add features without modifying existing components
- OCP also applies to packages
- Sometimes it is not pragmatic to extend a component and we have to modify it
- Making a flexible design adds more complexity

OCP is all about changes.
Following this principle will lead to elegant designs that are easy and painless to extend.