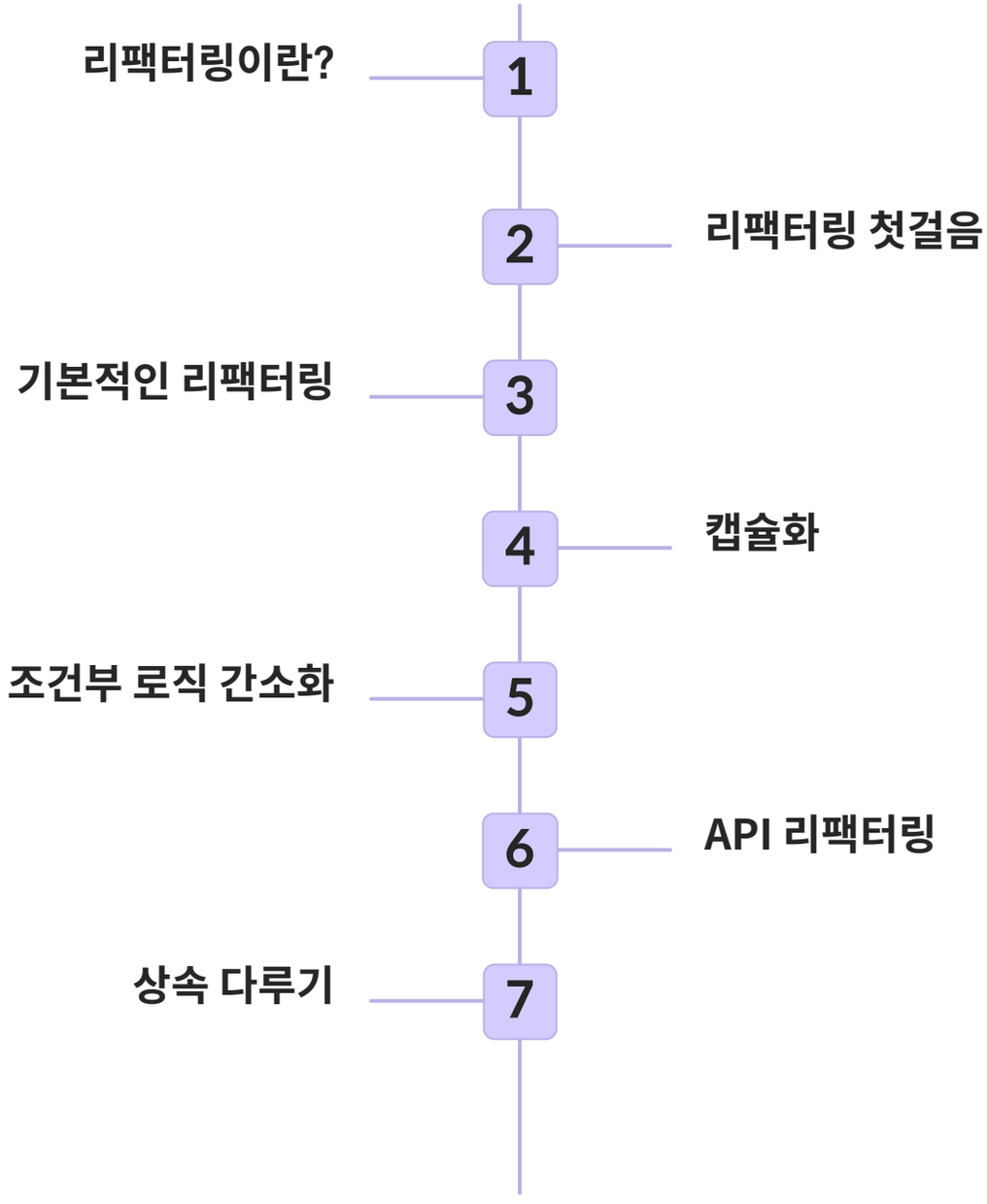
목차

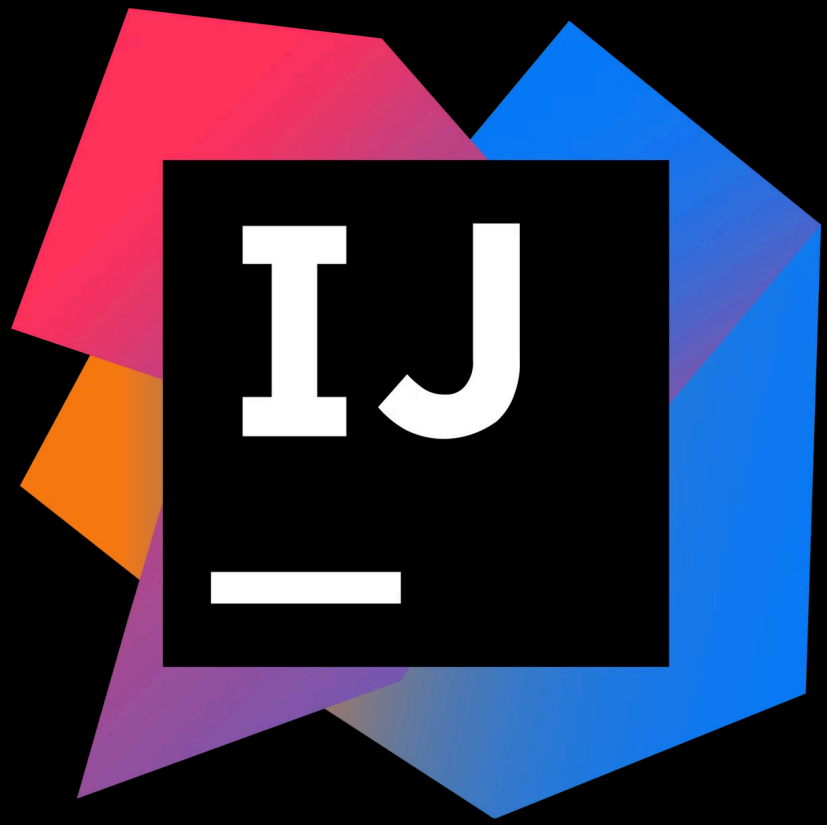

리팩터링이란?
1
2
리팩터링 첫걸음
기본적인 리팩터링
3
4
캡슐화
조건부 로직 간소화
5
6
API 리팩터링
상속 다루기
7

가장 중요한 것!

Intellij-IDEA 에서 제공하는

리팩터링 기능을

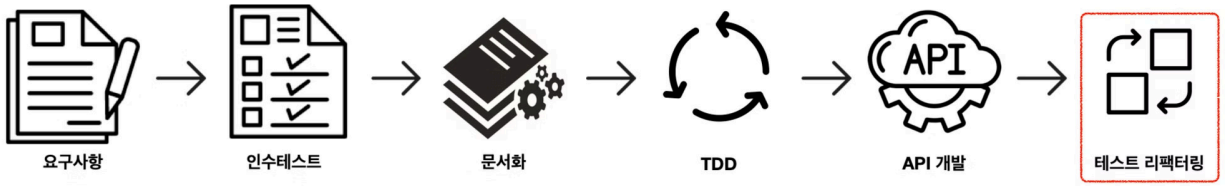
사용해 보시는 걸 추천 드립니다.



리팩터링이란?

> 소프트웨어의 외부 기능은 그대로 유지하면서,
>
> 내부 구조를 개선하는 작업

- ❌ 기능을 변경시키면 안됩니다.
- ✅ 테스트 코드가 필요합니다!



리팩터링, Why?

> 소프트웨어는 계속 변한다.

- 완벽한 코드는 없다.
- 코드는 팀원들과 함께 만들어 간다.

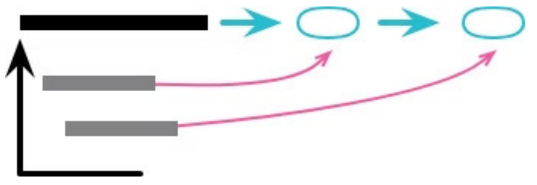
👉 리팩터링으로 꾸준히 개선해나가야 한다.

리팩터링 첫걸음

- **Replace Loop with Pipeline**
  - 반복문을 파이프라인으로 대체
- **Split Variable**
  - 변수 분리
- **Replace Magic Literal**
  - 매직 리터럴을 상수로 대체

Replace Loop with Pipeline

```java
public static List<String> twitterHandles(List<Author> authors, String company) {
    var result = new ArrayList<String> ();
    for (Author a : authors) {
        if (a.company.equals(company)) {
            var handle = a.twitterHandle;
            if (handle != null)
                result.add(handle);
        }
    }
    return result;
}
```



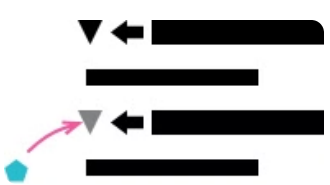
```java
public static List<String> twitterHandles(List<Author> authors, String company) {
    return authors.stream() Stream<Author>
        .filter(a -> Objects.equals(a.company, company))
        .map(a -> a.twitterHandle) Stream<String>
        .filter(Objects::nonNull)
        .toList();
}
```

Split Variable

- 어떤 변수가 여러번 재할당 되어도 적절한 경우
  - -for 문에서 i, j, k
  - 값을 축적시키는데 사용하는 변수 - Builder , result
- 어떤 의미를 가진 값인지 한 눈에 알아볼 수 있게 하자!

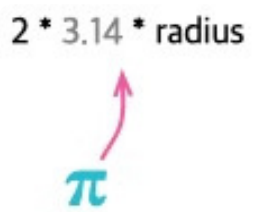
```
public void updateGeometry(double height, double width) {
    double temp = 2 * (height + width);
    perimeter = temp;

    temp = height * width;
    area = temp;
}
```

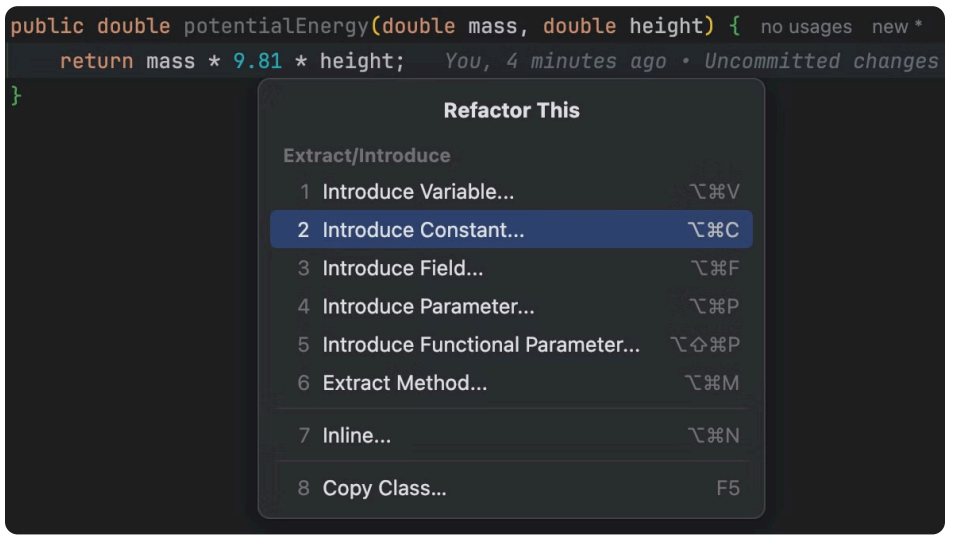
```
public void updateGeometry(double height, double width) {
    final double perimeter = 2 * (height + width);
    this.perimeter = perimeter;

    final double area = height * width;
    this.area = area;
}
```



Replace Magic Literal

```java
public double potentialEnergy(double mass, double height) {
    return mass * 9.81 * height;
}
```

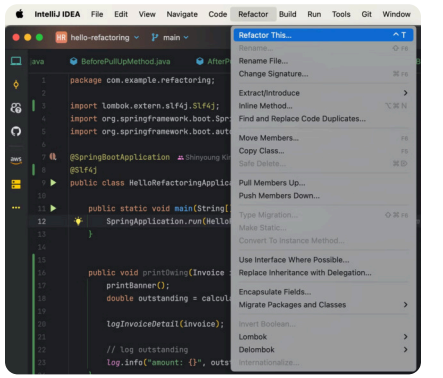





```java
public static final double GRAVITY_ACCELERATION = 9.81;  1 usage

public double potentialEnergy(double mass, double height) {  no usages
    return mass * GRAVITY_ACCELERATION * height;
}
```

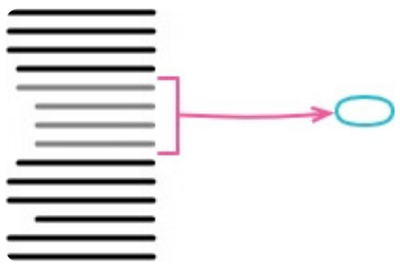
기본적인 리팩터링

- **Extract Method**: 중복 코드의 분리 및 함수로 추출
  - **Split Phase**: 단계 쪼개기
- **Inline Method**: 함수 호출을 함수 본문으로 대체
- **Extract Variable**: 복잡한 표현식을 변수로 추출
- **Inline Variable**: 불필요한 변수의 제거
  - **Replace Temp with Query**: 임시 변수를 질의 메소드로 대체
- **Rename Method/Variable**: 가독성을 높이기 위한 명확한 명명
  - **Change Function Declaration**: 함수 선언 변경
- **Introduce Parameter Object**: 여러 개의 매개변수를 객체로 추출
- **Combine Functions into Class**: 여러 함수를 클래스로 묶기
  - **Combine Functions into Transform**: 여러 함수를 변환 함수로 묶기



Extract Method

```java
public void printOwing(Invoice invoice) {  no usages  new *
    printBanner();
    double outstanding = calculateOutstanding(invoice);

    // log details
    log.info("name: {}", invoice.customer());
    log.info("address: {}", invoice.address());
    log.info("email: {}", invoice.email());

    // log outstanding
    log.info("amount: {}", outstanding);
}
```

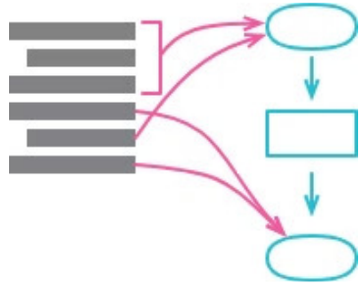


```java
public void printOwing(Invoice invoice) {  no usages  new *
    printBanner();
    double outstanding = calculateOutstanding(invoice);

    // log details
    log.info("name: {}", invoice.customer());
    log.info("address: {}", invoice.address());
    log.info("email: {}", invoice.email());

    // log outstanding
    log.info("amount: {}", outstanding);
}
```

```java
public void printOwing(Invoice invoice) {  no usages  new *
    printBanner();
    double outstanding = calculateOutstanding(invoice);

    // log details
    log.info("name: {}", invoice.customer());
    log.info("address: {}", invoice.address());
    log.info("email: {}", invoice.email());

    // log outstanding
    log.info("amount: {}", outstanding);
}
```

Split Phase

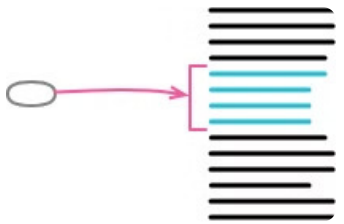
```java
public void orderProcess(String orderString) {  1 usage  new *
    String[] orderData = orderString.split( regex: "\s+");
    int productPrice = priceList.get(Integer.parseInt(orderData[0].split( regex: "-")[1]));
    int orderPrice = Integer.parseInt(orderData[1]) * productPrice;

    // ...
    log.info("Product Price: {}", productPrice);
    log.info("Order Price: {}", orderPrice);
}
```

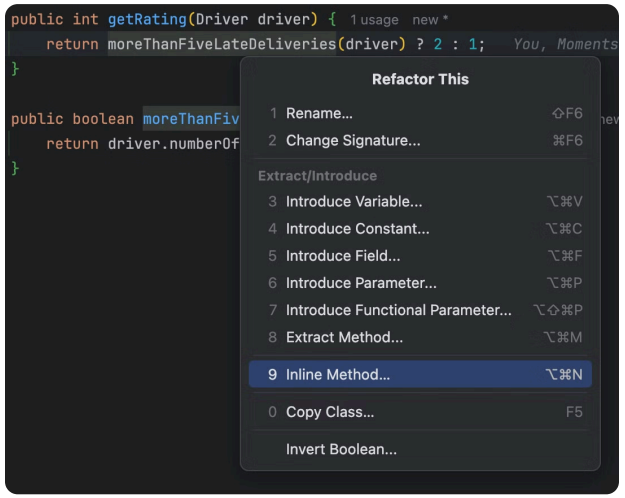


```java
private final List<Integer> priceList = List.of(100, 200, 300, 400, 500);  1 usage

public void orderProcess(String orderString) {  1 usage  new *
    Order order = Order.fromString(orderString);

    int productPrice = priceList.get(order.productId());
    int orderPrice = order.calculateOrderPrice(productPrice);

    // ...
    log.info("Product Price: {}", productPrice);
    log.info("Order Price: {}", orderPrice);
}

record Order(  4 usages  new *
    int productId,  1 usage
    int quantity  1 usage
) {
    public static final String REGEX = "\s+";  1 usage
    public static final String DELIMITER = "-";  1 usage

    public int calculateOrderPrice(int productPrice) {  1 usage  new *
        return quantity * productPrice;
    }

    public static Order fromString(String orderString) {  1 usage  new *
        String[] orderData = orderString.split(REGEX);

        int productId = Integer.parseInt(orderData[0].split(DELIMITER)[1]);
        int quantity = Integer.parseInt(orderData[1]);

        return new Order(
            productId,
            quantity
        );
    }
}
```

Inline Method

```
public int getRating(Driver driver) {  no usages  new *
    return moreThanFiveLateDeliveries(driver) ? 2 : 1;
}

public boolean moreThanFiveLateDeliveries(Driver driver) {
    return driver.numberOfLateDeliveries() > 5;
}
```






```
public int getRating(Driver driver) {  1 usage  new *
    return driver.numberOfLateDeliveries() > 5 ? 2 : 1;
}
```

Extract Variable

```
public double calculatePrice() {  no usages  new *
    // price is base price - discount + shipping cost
    return quantity * itemPrice -
        Math.max(0, quantity - minQuantityForDiscount) * itemPrice * discountRate +
        Math.max(minShippingCost, quantity * itemPrice * shippingCostRate);
}
```

```
public double calculatePrice() {  no usages  new *
    // price is base price - discount + shipping cost
    return quantity * itemPrice -    You, Moments ago • Uncommitted changes
        Mat                                                t) * itemPrice * discountRate +
        Mat                                                ce * shippingCostRate);
}
```

Refactor This

1	Rename...	⇧F6
	Extract/Introduce	
2	Introduce Variable...	⌥⌘V
3	Introduce Constant...	⌥⌘C
4	Introduce Field...	⌥⌘F
5	Introduce Parameter...	⌥⌘P
6	Introduce Functional Parameter...	⌥⇧⌘P
7	Introduce Functional Variable...	
8	Extract Method...	⌥⌘M
9	Inline Field...	⌥⌘N
0	Copy Class...	F5

```
public double calculatePrice() {  no usages  new *
    // price is base price - discount + shipping cost
    double basePrice = quantity * itemPrice;
    double discount = Math.max(0, quantity - minQuantityForDiscount) * itemPrice * discountRate;
    double shippingCost = Math.max(minShippingCost, quantity * itemPrice * shippingCostRate);

    return basePrice - discount + shippingCost;
}
```

Inline Variable

```
public boolean isMoreThanMinPrice() {  no usages
    double basePrice = baseOrder.price;

    return basePrice > MIN_PRICE;
}
```

```
public boolean isMoreThanMinPrice() {  no usages  new *
    double basePrice = baseOrder.price;    You, Moments ago
    return basePri
}
```

Refactor This

1 Rename... ⇧F6

Extract/Introduce

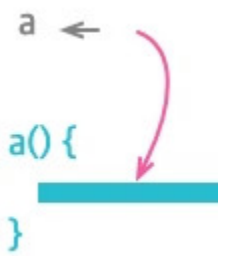
2 Introduce Constant... ⌥⌘C
3 Introduce Field... ⌥⌘F
4 Introduce Parameter... ⌥⌘P
5 Introduce Functional Parameter... ⌥⇧⌘P
6 Inline Variable ⌥⌘N
7 Copy Class... F5
8 Safe Delete... ⌘⌦
9 Type Migration... ⇧⌘F6

```
public boolean isMoreThanMinPrice() {  no usages
    return baseOrder.price > MIN_PRICE;
}
```

Replace Temp with Query

```
public double calculateFinalPrice() {  no usages  new *
    double basePrice = this.quantity * this.itemPrice;

    if (basePrice > 1000) {
        return basePrice * 0.95;
    } else {
        return basePrice * 0.98;
    }
}
```

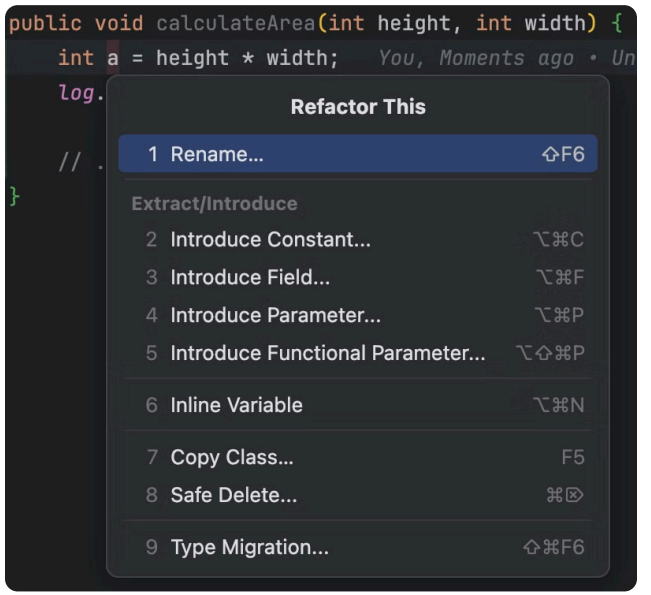
```
public double calculateFinalPrice() {  no usages  new *
    double basePrice = this.quantity * this.itemPrice;
          2.Inline Variable                1. Extract Method
    if (basePrice > 1000) {
        return basePrice * 0.95;
    } else {
        return basePrice * 0.98;
    }
}
```




```
public double calculateFinalPrice() {  no us
    if (basePrice() > 1000) {
        return basePrice() * 0.95;
    } else {
        return basePrice() * 0.98;
    }
}

private double basePrice() {  3 usages  new *
    return this.quantity * this.itemPrice;
}
```

Rename Method / Variable

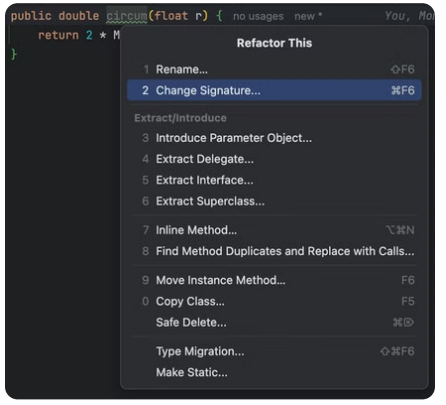
```java
public void calculateArea(int height, int width) {
    int a = height * width;
    log.info("Area: {}", a);

    // ...
}
```

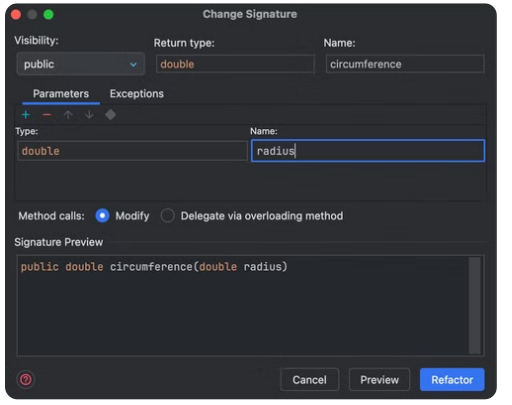


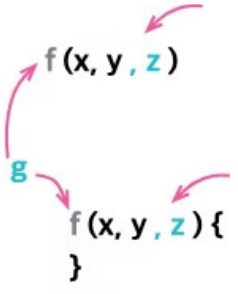

```java
public void calculateArea(int height, int width) {
    int area = height * width;
    log.info("Area: {}", area);

    // ...
}
```

Change Function Declaration

```
public double circum(float r) {
    return 2 * Math.PI * r;
}
```



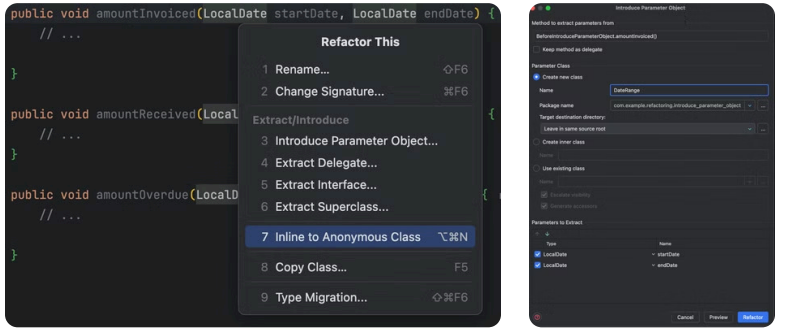


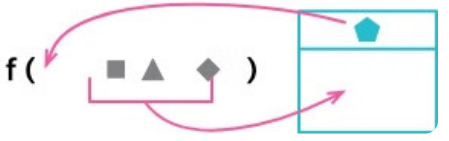



```
public double circumference(double radius) {
    return 2 * Math.PI * radius;
}
```

Introduce Parameter Object

```java
public void amountInvoiced(LocalDate startDate, LocalDate endDate) {
    // ...

}

public void amountReceived(LocalDate startDate, LocalDate endDate) {
    // ...
}

public void amountOverdue(LocalDate startDate, LocalDate endDate) {
    // ...

}
```

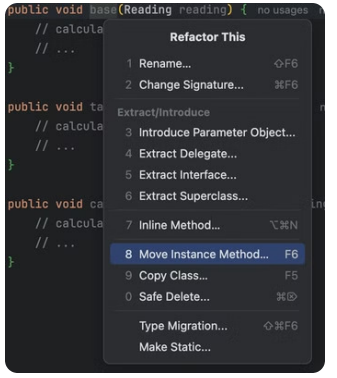


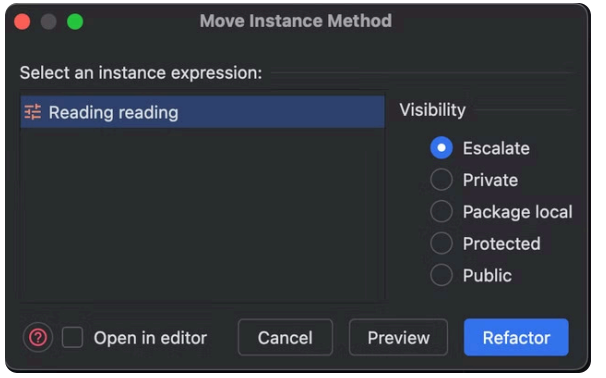


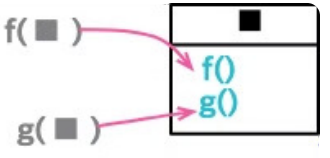

```java
public void amountInvoiced(DateRange dateRange) {
    // ...

}

public void amountReceived(DateRange dateRange) {
    // ...
}

public void amountOverdue(DateRange dateRange) {
    // ...

}
```

Combine Functions into Class

```
public void base(Reading reading) {  no usages  new *
    // calculate base charge
    // ...
}

public void taxableCharge(Reading reading) {  no usag
    // calculate taxable charge
    // ...
}

public void calculateBaseCharge(Reading reading) {
    // calculate base charge
    // ...
}
```

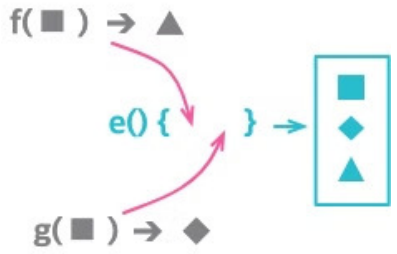







```
public class Reading {  no usages  new *
    public void base() {  no usages  new *
        // calculate base charge
        // ...
    }

    public void taxableCharge() {  no usag
        // calculate taxable charge
        // ...
    }

    public void calculateBaseCharge() {
        // calculate base charge
        // ...
    }
}
```

Combine Functions into Transform




```
function base(aReading) {...}
function taxableCharge(aReading) {...}
```

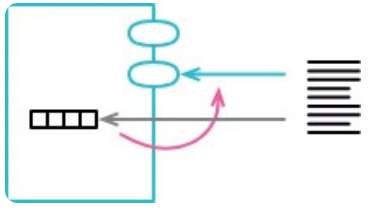
```
function enrichReading(argReading) {
  const aReading = _.cloneDeep(argReading);
  aReading.baseCharge = base(aReading);
  aReading.taxableCharge = taxableCharge(aReading);
  return aReading;
}
```

캡슐화

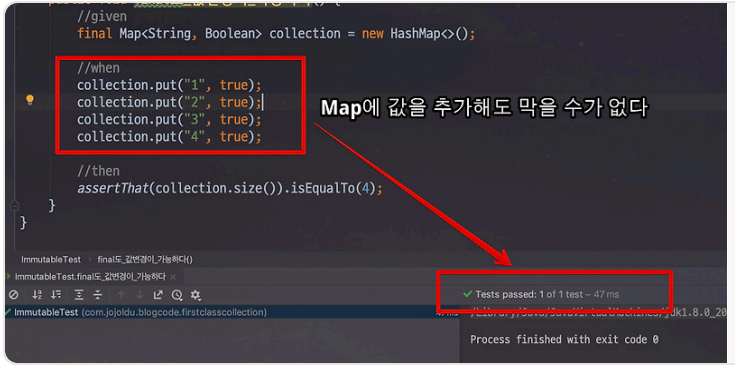
- **Encapsulate Collection**
  - 컬렉션 캡슐화
- **Replace Primitive with Object**
  - 기본 타입을 객체로 대체

Encapsulate Collection

```
List<Course> courses = student.getCourses();

courses.add(new Course());
courses.set(2, new Course());
courses.remove( index: 3);
```



```
class Courses {  3 usages  new *
    private List<Course> courses;  3 usages

    public Courses(List<Course> courses) {  no
        this.courses = courses;
    }

    public void addCourse(Course course) {  1 us
        courses.add(course);
    }

    public void removeCourse(Course course) {
        courses.remove(course);
    }
}
```



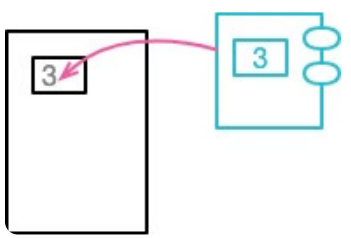
TISTORY

일급 컬렉션 (First Class Collection)의 소개와 써야할 이유

최근 클린코드 & TDD 강의의 리뷰어로 참가하면서 많은 분들이 공통적으로 어려워 하는 개념 한가지를 발견하게 되었습니다. 바로 일급 컬렉션인데요. 왜 객체지향적으로, ...

Replace Primitive with Object

```
orders.stream()
    .filter(o -> o.priority.equals("high") || o.priority.equals("rush"))
    .toList();
```



1. Extract Method
2. Convert Into Instance Method

```
class Order {  5 usages  new *
    private String priority;  2 usages

    private boolean isHighAndRush() {  1 usage  new *
        return this.priority.equals("high") || this.priority.equals("rush");
    }
}
```

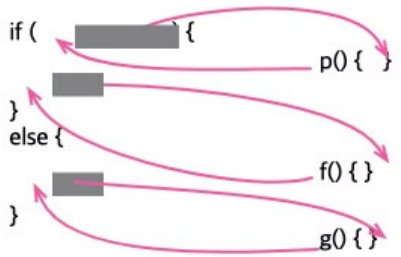
```
orders.stream()
    .filter(Order::isHighAndRush)
    .toList();
```

조건부 로직 간소화

- **Decompose Conditional**
  - 조건문을 보기 좋게 함수로 분리
- **Replace Conditional with Polymorphism**
  - 조건부 로직을 다형성으로 대체
- **Introduce Assertion**
  - 가정문을 추가하여 코드의 가독성 향상

```
ome != ""){
.sobrenome != ""){
is.dataNascimento != null){
this.cpf != "" && this.cpf.length == 11){
f(this.fone != "" && this.fone.length==11){
 if(this.telefone.length==0 || this.telefone.length==10){
   if(this.endereco != null && this.cep != ""){
     if(this.complemento_endereco != ""){
       this.endereco.complemento = this.complemento_endereco;
       if(this.senh != "" && this.csenha!=""){
         if(this.senh.length >=6 && this.senh.length <= 10){
           if(this.senh == this.csenha){
             if(this.maile != ""){
               this.router.navigate(['/resultado/'], {
                 state: {
                   nome: this.nome,
                   sobrenome: this.sobrenome,
                   datanascimento: this.dataNascimento,
                   sexo: this.sexo ,
                   celular: this.fone,
                   telefone: this.telefone,
                   email : this.maile,
                   senha: this.senh,
                   cpf: this.cpf,
                   cep: this.cep,
                   endereco: this.endereco
                 }
               });
             }else this.presentAlert("E-mail não foi informado");
           }else this.presentAlert("As senhas informadas não são iguais");
         }else this.presentAlert("A senha informada precisa ter entre 6 e 10 ca
       }else this.presentAlert("Dados de senhas não foram devidamente informados
     }else this.presentAlert("Por favor, informe o complemento do seu endereço"
   }else this.presentAlert("Por favor, informe um cep válido para ser consultad
 }else this.presentAlert("Número de telefone fixo inválido");
```

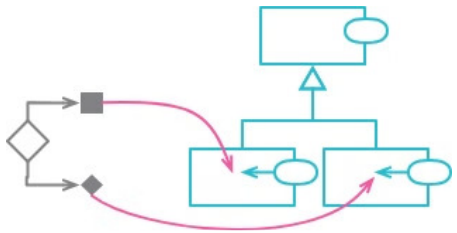
Decompose Conditional




```
if (!aDate.isBefore(plan.summerStart) && !aDate.isAfter(plan.summerEnd))
  charge = quantity * plan.summerRate;
else
  charge = quantity * plan.regularRate + plan.regularServiceCharge;
```

```
if (summer())
  charge = summerCharge();
else
  charge = regularCharge();
```

Replace Conditional with Polymorphism



```
switch (bird.type) {
  case 'EuropeanSwallow':
    return "average";
  case 'AfricanSwallow':
    return (bird.numberOfCoconuts > 2) ? "tired" : "average";
  case 'NorwegianBlueParrot':
    return (bird.voltage > 100) ? "scorched" : "beautiful";
  default:
    return "unknown";
```

```
class EuropeanSwallow {
  get plumage() {
    return "average";
  }
class AfricanSwallow {
  get plumage() {
     return (this.numberOfCoconuts > 2) ? "tired" : "average";
  }
class NorwegianBlueParrot {
  get plumage() {
     return (this.voltage > 100) ? "scorched" : "beautiful";
  }
```

Introduce Assertion


```
if (this.discountRate)
  base = base - (this.discountRate * base);
```

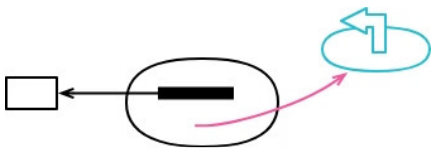
```
assert(this.discountRate >= 0);
if (this.discountRate)
  base = base - (this.discountRate * base);
```

- Java Guava Precondition 클래스
- Kotlin `require` , `check` 메서드
  - IllegalArgumentException
  - IllegalStateException

API 리팩터링

- **Separate Query from Modifier**: 질의 함수와 변경 함수 분리
- **Remove Flag Argument**: 플래그 인수 제거
- **Replace Parameter with Query**: 매개변수를 질의 함수로 대체
- **Replace Error Code with Exception**: 에러 코드를 예외로 대체

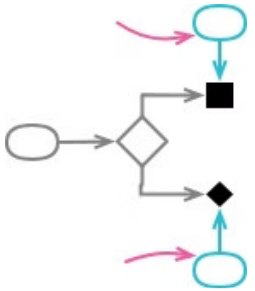
Separate Query from Modifier



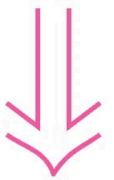
```
function getTotalOutstandingAndSendBill() {
  const result = customer.invoices.reduce((total, each) => each.amount + total, 0);
  sendBill();
  return result;
}
```

```
function totalOutstanding() {
  return customer.invoices.reduce((total, each) => each.amount + total, 0);
}
function sendBill() {
  emailGateway.send(formatBill(customer));
}
```

Remove Flag Argument



```
function setDimension(name, value) {
  if (name === "height") {
    this._height = value;
    return;
  }
  if (name === "width") {
    this._width = value;
    return;
  }
}
```



```
function setHeight(value) {this._height = value;}
function setWidth (value) {this._width = value;}
```

Replace Parameter with Query


```
availableVacation(anEmployee, anEmployee.grade);

function availableVacation(anEmployee, grade) {
  // calculate vacation...
```

```
availableVacation(anEmployee)

function availableVacation(anEmployee) {
  const grade = anEmployee.grade;
  // calculate vacation...
```

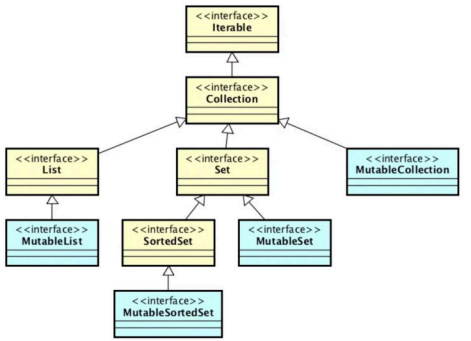
Replace Error Code with Exception


```
if (data)
  return new ShippingRules(data);
else
  return -23;
```

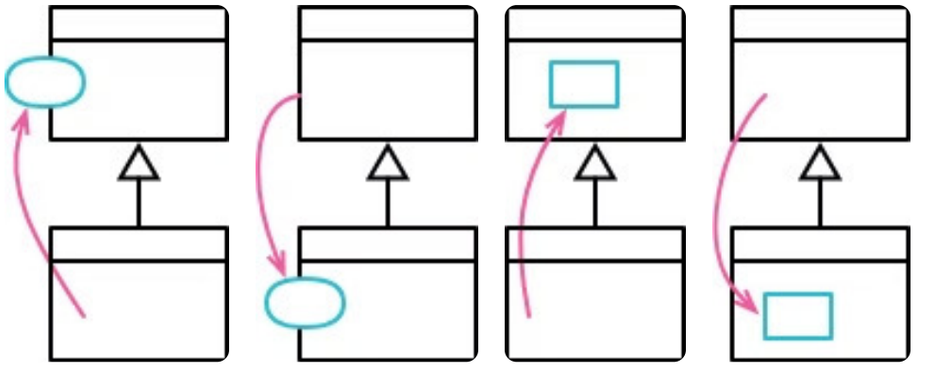
```
if (data)
  return new ShippingRules(data);
else
  throw new OrderProcessingError(-23);
```

상속 다루기

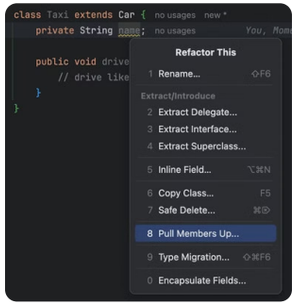
- **Pull Up Method**, **Pull Down Method**, **Pull Up Field**, **Pull Down Field**
  - 상속 관계에서 메소드와 필드를 상위 클래스로 끌어올리거나 내리기
  - **Pull Up Constructor Body**
    - 생성자 공통화
- **Replace Type Code with Subclasses**
  - 타입 코드를 하위 클래스로 대체
- **Replace Subclass with Delegate**, **Replace Superclass with Delegate**
  - 역할을 다른 객체에 위임

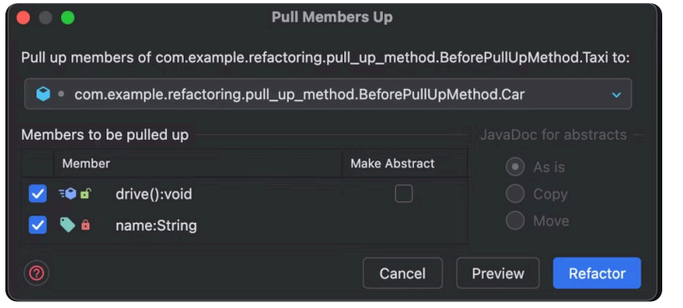


Pull Up/Down Method or Field



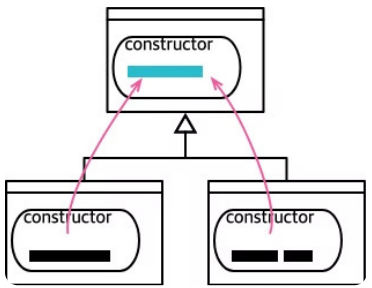
```
class Car {  1 usage  1 inheritor  new *

}

class Taxi extends Car {  no usages  new *
    private String name;  no usages

    public void drive() {  no usages  new *
        // drive like taxi
    }
}
```






```
class Car {  1 usage  1 inheritor  new *

    private String name;  no usages

    public void drive() {  no usages  new *
        // drive like taxi
    }
}

class Taxi extends Car {  no usages  new *

}
```

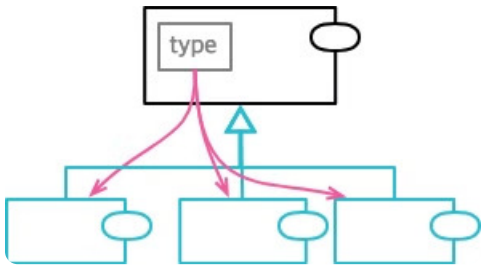
Pull Up Constructor Body




```
class Car {  1 usage   1 inheritor   new *
    protected String name;
}

class Taxi extends Car {  no usages   new *

    public Taxi(String name) {  no usages
        super();
        this.name = name;
    }
}
```

```
class Car {  1 usage   1 inheritor   new *
    protected String name;

    protected Car(String name) {  1 usage
    }
}

class Taxi extends Car {  no usages   new *

    public Taxi(String name) {  no usages
        super(name);
    }
}
```

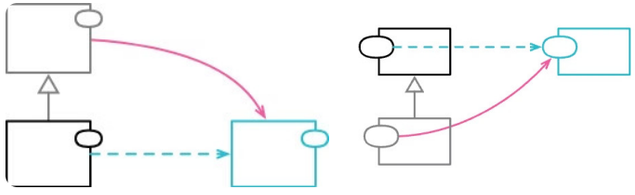
Replace Type Code with Subclasses




```
function createEmployee(name, type) {
  return new Employee(name, type);
}
```

```
function createEmployee(name, type) {
  switch (type) {
    case "engineer": return new Engineer(name);
    case "salesman": return new Salesman(name);
    case "manager":  return new Manager (name);
  }
```

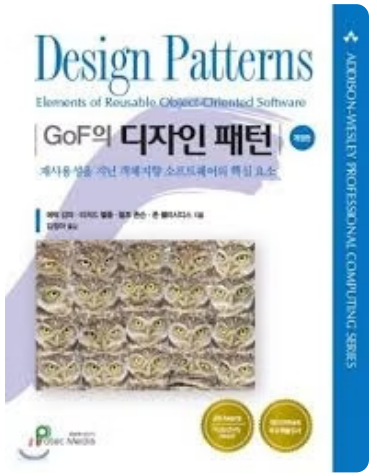
Replace Subclass/Superclass with Delegate

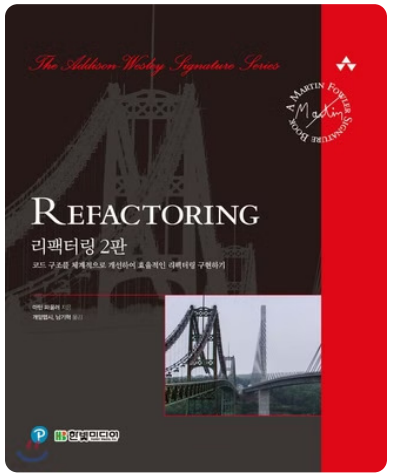


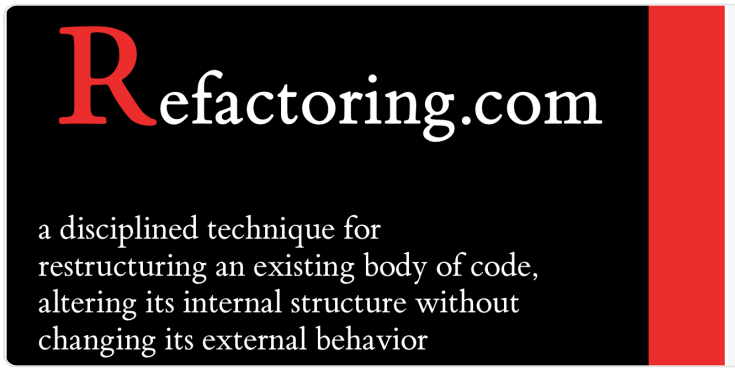
```
abstract class Car {  3 usages  2 inheritors  new *
    abstract void drive();  no usages  2 implementations
}

class Bus extends Car {  no usages  new *

    @Override  no usages  new *
    void drive() {
        // drive like bus
    }
}

class Taxi extends Car {  no usages  new *

    @Override  no usages  new *
    void drive() {
        // drive like taxi
    }
}
```

```
abstract class Car {  5 usages  2 inheritors  new *
    private final DrivingAlgorithm drivingAlgorithm;  2

    protected Car(DrivingAlgorithm drivingAlgorithm) {
        this.drivingAlgorithm = drivingAlgorithm;
    }

    public void drive() {  no usages  2 overrides  new *
        drivingAlgorithm.drive( car: this);
    }
}

class Bus extends Car {  no usages  new *

    public Bus(DrivingAlgorithm drivingAlgorithm) {  no
        super(new BusDrivingAlgorithm());
    }

    @Override  no usages  new *
    public void drive() {
        // drive like bus
    }
}

class Taxi extends Car {  no usages  new *

    public Taxi(DrivingAlgorithm drivingAlgorithm) {  no
        super(new TaxiDrivingAlgorithm());
    }

    @Override  no usages  new *
    public void drive() {
        // drive like taxi
    }
}
```

참고







refactoring.com

Refactoring Home Page

Introduction to the technique of refactoring and online catalog of refactorings