Refactoring for Deeper Understanding

NICOLE RAUCH
softwareentwicklung & entwicklungscoaching

The Plan for Today

- Only 1 hour
- Only 1 pattern

- Most important one for me right now

What is the problem?

```
public double calcDiscount( double p1, double p2 ){
    // ...
}
```

What is the problem?

```
public double calcDiscount( double p1, double p2 ){
    // ...
}
```

$\longrightarrow$ Bad parameter names

What about this one, then?

```
public double calcDiscount( double amount,
                            double discount ){
   // ...
}
```

What about this one, then?

```
public double calcDiscount( double amount,
                            double discount ){
    // ...
}
```

$\longrightarrow$ Everything is clear, right?

Maybe there are still some questions?

```
public double calcDiscount( double amount,
                            double discount ){
    // ...
}
```

Maybe there are still some questions?

```
public double calcDiscount( double amount,
                            double discount ){
  // ...
}
```

- Is the amount a monetary value? What about rounding?

Maybe there are still some questions?

```
public double calcDiscount( double amount,
                            double discount ){
   // ...
}
```

- Is the amount a monetary value? What about rounding?
- Should we really use `double` for monetary values?

Maybe there are still some questions?

```
public double calcDiscount( double amount,
                            double discount ){
   // ...
}
```

- Is the amount a monetary value? What about rounding?
- Should we really use `double` for monetary values?
- Is the discount a monetary amount that gets subtracted?

Maybe there are still some questions?

```
public double calcDiscount( double amount,
                            double discount ){
   // ...
}
```

- Is the amount a monetary value? What about rounding?
- Should we really use `double` for monetary values?
- Is the discount a monetary amount that gets subtracted?
- Or is the discount a percentage that is used to determine a fraction?

Maybe there are still some questions?

```
public double calcDiscount( double amount,
                            double discount ){
   // ...
}
```

- Is the amount a monetary value? What about rounding?
- Should we really use `double` for monetary values?
- Is the discount a monetary amount that gets subtracted?
- Or is the discount a percentage that is used to determine a fraction?
- Is a percentage of 5% passed as 5.0 or as 0.05?

Maybe there are still some questions?

```
public double calcDiscount( double amount,
                            double discount ){
   // ...
}
```

- Is the amount a monetary value? What about rounding?
- Should we really use `double` for monetary values?
- Is the discount a monetary amount that gets subtracted?
- Or is the discount a percentage that is used to determine a fraction?
- Is a percentage of 5% passed as 5.0 or as 0.05?
- Is the return value just the discount or the discounted amount?

Maybe there are still some questions?

```
public double calcDiscount( double amount,
                            double discount ){
   // ...
}
```

- Is the amount a monetary value? What about rounding?
- Should we really use `double` for monetary values?
- Is the discount a monetary amount that gets subtracted?
- Or is the discount a percentage that is used to determine a fraction?
- Is a percentage of 5% passed as 5.0 or as 0.05?
- Is the return value just the discount or the discounted amount?
- Is it easy to mess up the invocation by accidentally swapping parameters?

Maybe there are still some questions?

```
public double calcDiscount( double amount,
                            double discount ){
   // ...
}
```

- Is the amount a monetary value? What about rounding?
- Should we really use `double` for monetary values?
- Is the discount a monetary amount that gets subtracted?
- Or is the discount a percentage that is used to determine a fraction?
- Is a percentage of 5% passed as 5.0 or as 0.05?
- Is the return value just the discount or the discounted amount?
- Is it easy to mess up the invocation by accidentally swapping parameters?
- ...

How to improve this?

How to improve this?

Ask the type system for help!

How to improve this?

Ask the type system for help!

Object Calisthenics:

Programming exercises for better object-oriented code

Rule 3: Wrap All Primitives and Strings

Bad Types

```
public double calcDiscount( double amount,
                            double discount ){
   // ...
}
```

Better Types

```
public DiscountedAmount calcDiscount(
                            MonetaryAmount amount,
                            Percent discount ) {
    // ...
}
```

Why “DiscountedAmount” and not “AmountToBeSubtracted”?

- It’s visible whether the amount has already been discounted
- It’s robust because we cannot accidentally discount twice
- It describes stages in our process which leads to clear handovers

Let’s Check Our Questions

```
public DiscountedAmount calcDiscount(
                                    MonetaryAmount amount,
                                    Percent discount ) {
    // ...
}
```

Let’s Check Our Questions

```
public DiscountedAmount calcDiscount(
                          MonetaryAmount amount,
                          Percent discount ) {
   // ...
}
```

- Is the amount a monetary value? What about rounding?
- Should we really use `double` for monetary values?

$\Longrightarrow$ Encapsulated by MonetaryAmount, we know where to look / change

Let’s Check Our Questions

```
public DiscountedAmount calcDiscount(
                            MonetaryAmount amount,
                            Percent discount ) {
    // ...
}
```

- Is the discount a monetary amount that gets subtracted?
- Or is the discount a percentage that is used to divide the amount?

$\Longrightarrow$ Clear from the type name

Let’s Check Our Questions

```
public DiscountedAmount calcDiscount(
                        MonetaryAmount amount,
                        Percent discount ) {
   // ...
}
```

- Is a percentage of 5% passed as 5.0 or as 0.05?

$\Longrightarrow$ Still not obvious, but encapsulated by Percent, i.e. easy to find out

Let’s Check Our Questions

```
public DiscountedAmount calcDiscount(
                                MonetaryAmount amount,
                                Percent discount ) {
    // ...
}
```

- Is the return value just the discount or the discounted amount?

$\Longrightarrow$ Clear from the type name

Let’s Check Our Questions

```
public DiscountedAmount calcDiscount(
                        MonetaryAmount amount,
                        Percent discount ) {
  // ...
}
```

- Is it easy to mess up the invocation by accidentally swapping parameters?

$\Longrightarrow$ No, different types are not swappable

Refactoring - Demo

Recap: Refactoring Steps (1)

- Generate Parameter Object
  - With “old” Java: Generates class
  - With Java >= 16: Generates record
- Or write your own Parameter Object and introduce it
  - Caution with `.toString()`!
- Push the new type through step-by-step until everything is covered
- Lots of invocation sites? Add overloaded method that wraps encapsulation

“Code Magnets”

- Move code into the object that is closest to it
- Helps everybody find code faster
- Helps avoid redundant implementations because code was not found

Code Magnets in our example (1)

From `DiscountCalculator.java`

```
... percent.value() ...
```

```
... percent.value() / 100 ...
```

Code Magnets in our example (1)

From `DiscountCalculator.java`

```
... percent.value() ...
```

```
... percent.value() / 100 ...
```

Strong hint: The value is pulled out and manipulated. Let’s encapsulate it!

Code Magnets in our example (1)

From `DiscountCalculator.java`

```
... percent.value() ...

... percent.value() / 100 ...
```

Strong hint: The value is pulled out and manipulated. Let’s encapsulate it!

to `Percent.java`

```
public double asDecimal() { return percent / 100; }
public double asNominal() { return percent; }
```

(also improves our last open point)

Code Magnets in our example (2)

From `DiscountCalculator.java`

```
public static DiscountedAmount calcDiscount(
                              MonetaryAmount amount,
                              Percent discount ) { /* ... */ }
```

Code Magnets in our example (2)

From `DiscountCalculator.java`

```
public static DiscountedAmount calcDiscount(
                              MonetaryAmount amount,
                              Percent discount ) { /* ... */ }
```

Strong hint: The method is static. Let’s find a better place for it!

Code Magnets in our example (2)

From `DiscountCalculator.java`

```
public static DiscountedAmount calcDiscount(
                                 MonetaryAmount amount,
                                 Percent discount ) { /* ... */ }
```

Strong hint: The method is static. Let’s find a better place for it!

to `MonetaryAmount.java`

```
public DiscountedAmount applyDiscount(Percent discount) {
```

- **Reliable:** Whenever it gets created, it is correct (wrt business rules)
- **Encapsulated:** Easily modifiable in one location (no shotgun surgery)
- **Understandable:** You know where to look if something is unclear
- **Immutable:** Cannot accidentally be changed. Also, can be shared.
- **Testable:** Logic is isolated and easily testable.

- **Reliable:** Whenever it gets created, it is correct (wrt business rules)
- **Encapsulated:** Easily modifiable in one location (no shotgun surgery)
- **Understandable:** You know where to look if something is unclear
- **Immutable:** Cannot accidentally be changed. Also, can be shared.
- **Testable:** Logic is isolated and easily testable.

$\Longrightarrow$ Domain-Driven Design calls them **Value Objects**

Value Object in Domain-Driven Design

- Value
- Has no identity (describes “what”)
- Business wrapper around a technical datatype
- Forms a conceptual unit
- Can often be immutable
  - Then, sharing is possible (and highly desirable)
- Structure can be complex

Value Objects with multiple variables

```
public class MonetaryAmount {
    private final double amount;
    private final String currency;

  // ...
}
```

Value Objects with multiple variables

```
public class MonetaryAmount {
    private final double amount;
    private final String currency;

   // ...
}
```

Address with street, number, postal code, city, country...

Value Objects with multiple variables

```
public class MonetaryAmount {
    private final double amount;
    private final String currency;

   // ...
}
```

Address with street, number, postal code, city, country...

Encapsulates a business concept and reduces the “contact surface” to the surrounding code

How to find the words?

We need good names for types and variables

How to find the words?

We need good names for types and variables

Domain-Driven Design: **Ubiquitous Language**

- Focus on business, without technical stuff
- Uniqueness: Only one term per concept, only one concept per term
- Glossary captures all domain terms and explains them

This gives us the **deep understanding** we need to develop good software!

How to find the words?

We need good names for types and variables

Domain-Driven Design: **Ubiquitous Language**

- Focus on business, without technical stuff
- Uniqueness: Only one term per concept, only one concept per term
- Glossary captures all domain terms and explains them

This gives us the **deep understanding** we need to develop good software!

- Also lives in the team's language
- Only use domain terms in the code
- This implies: We need to cooperate with business people!

What else is possible?

What else is possible?

- On Percent creation, it is not clear what to pass (5 oder 0.05)
- Add dedicated builder methods

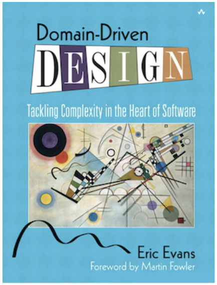
What else is possible?

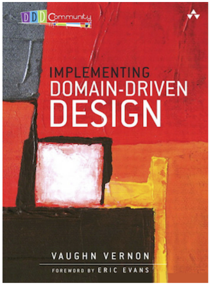
- On Percent creation, it is not clear what to pass (5 oder 0.05)
- Add dedicated builder methods

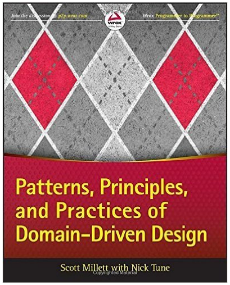
- Percentage calculation is hardcoded in the middle of somewhere
- Turn this into a Domain-Driven Design Policy

Resources: Domain-Driven Design in general

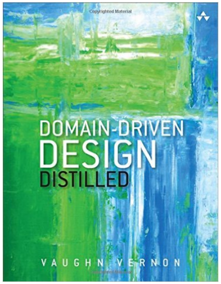
My recommendation:


Domain-Driven
DESIGN
Tackling Complexity in the Heart of Software
Eric Evans
Foreword by Martin Fowler

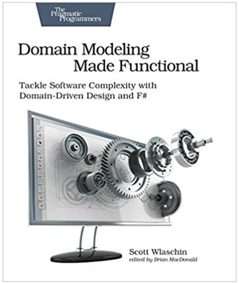



wrox
Patterns, Principles, and Practices of Domain-Driven Design
Scott Millett with Nick Tune

Resources: Domain-Driven Design in general



Vaughn Vernon
Domain-Driven
Design
kompakt
→ Aus dem Englischen übersetzt
von Carola Lilienthal und Henning Schwentner
dpunkt.verlag


The Pragmatic Programmers
Domain Modeling
Made Functional
Tackle Software Complexity with
Domain-Driven Design and F#
Scott Wlaschin
edited by Brian MacDonald

Resources: Various Links

- Object Calisthenics: `https://blog.avenuecode.com/object-calisthenics-principles-for-better-object-oriented-code`
- Java 14 Records: `https://docs.oracle.com/en/java/javase/14/language/records.html`
- Code Magnets: Talk “Power Use of Value Objects in DDD”: `https://www.infoq.com/presentations/Value-Objects-Dan-Bergh-Johnsson`

Want more?

Hexagonal Architecture - by Thomas Pierrain
In-depth session on Fri/Sat if interested

Thank You!

E-Mail	info@nicole-rauch.de
Twitter	@NicoleRauch
Web	http://www.nicole-rauch.de

NICOLE RAUCH
software development &
development coaching

Domain-Driven Design · Specification by Example
Software Craftsmanship
React & Redux · TypeScript
Functional Programming