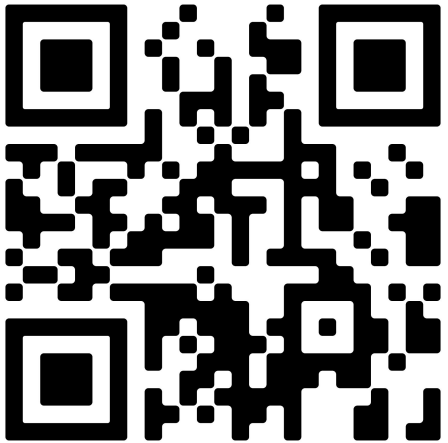
Не нужно ничего менять!

Как жить с иммутабильными объектами?

Антон Стеканов



Кто я

Разработчик со стажем 10+ лет и немного тимлид

В основном всякие бэкенды на Java,
но были разные языки и стеки: от GUI на Java и C#
до WEB UI от jQuery до React

Всегда старался и стараюсь работать над качеством, понятностью и поддерживаемостью кода

Интересуюсь функциональным программированием и различными подходами к разработке от Lisp с его макросами до Haskell и языков с зависимыми типами

Участник Московского клуба программистов

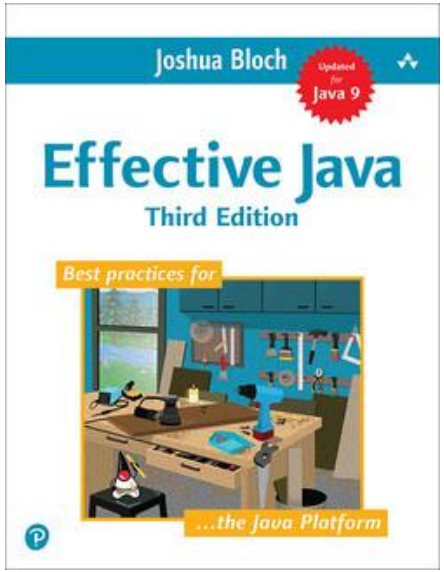
- Сайт: prog.msk.ru
- Чат в telegram: progmsk

Задавайте вопросы сразу

Авторитеты

Авторитеты

Joshua Bloch



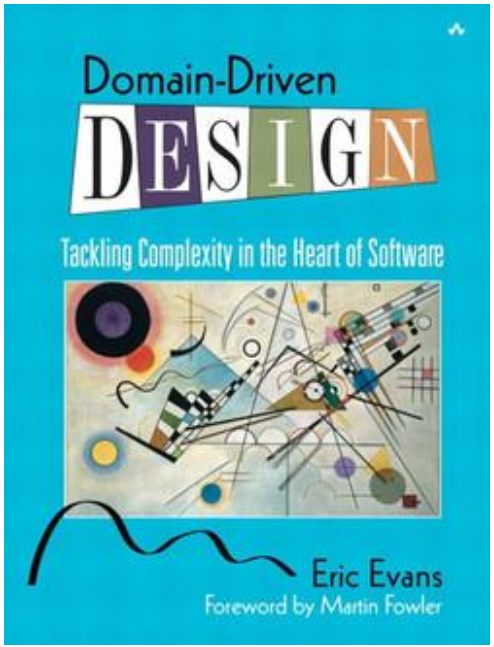
Effective Java, 3rd Edition
4 Classes and Interfaces
Item 17: Minimize mutability

В переводе:
4.3. Минимизируйте изменяемость

Авторитеты

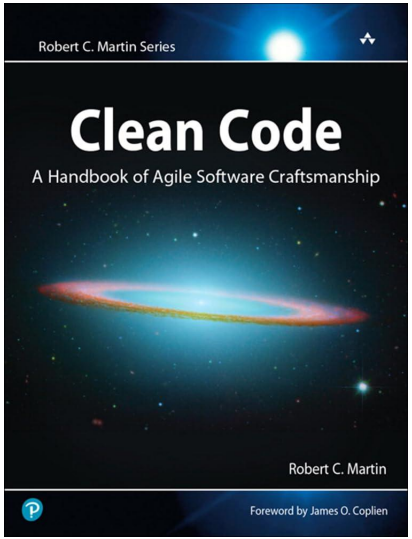
Eric Evans, Domain-Driven Design: Tackling Complexity in the Heart of Software,
Chapter 10, Side-effect-free-functions

Предметно-ориентированное проектирование (DDD): структуризация сложных программных систем
Глава 10, Функции без побочных эффектов



Авторитеты

Robert Martin



Clean Code: A Handbook of Agile Software Craftsmanship,
Chapter 3: Functions, Have No Side Effects

Чистый код. Создание анализ и рефакторинг,
Глава 3, Избавьтесь от побочных эффектов

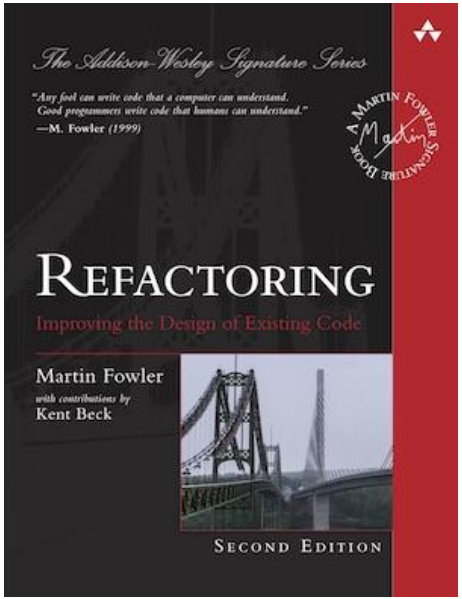
Авторитеты

Rich Hickey
автор Clojure

Simple Made Easy
“State is Never Simple”

Авторитеты

Martin Fowler



[Mutable Data] code smell

“Mutable data are harmful”

[Refactoring: Improving the Design of Existing Code]

Что плохого в изменяемости?

- Сложность понимания
- Непредсказуемость
- Проблемы с конкурентностью (см. JCP и Effective Java)
- Чистые функции
- Сложность тестирования
- Nullable поля

Liskov Substitution Principle (LSP)

```
public class Rectangle {

   int width, height;

   public Rectangle(int width, int height) {

       this.width = width;

       this.height = height;

   }

   public int getWidth() { return width; }

   public int getHeight() { return height; }

   public void setWidth(int width) { this.width = width; }

   public void setHeight(int height) { this.height = height; }

}
```

```
public class Square extends Rectangle {
   public Square(int size) {
       super(size, size);
   }
}
```

Liskov Substitution Principle (LSP)

```
public class Rectangle {

    int width, height;

    public Rectangle(int width, int height) {

        this.width = width;

        this.height = height;

    }

    public int getWidth() { return width; }

    public int getHeight() { return height; }

    public void setWidth(int width) { this.width = width; }

    public void setHeight(int height) { this.height = height; }

}
```

```
public class Square extends Rectangle {
    public Square(int size) {
        super(size, size);
    }
}

void example() {
    Square square = new Square(10);
    square.setHeight(5);
}
```

Возражения

- производительность
- неинтуитивно
- только для FP

Как жить?

- Возможно ли использовать в обычных языках иммутабельные объекты?
- Не будет ли это существенно сложнее, чем без них?
- Нужно ли для этого менять язык программирования?
- Стоит ли результат затраченных усилий?

Mutable POJO vs Records

```
public final class Point {
   private double x;
   private double y;

   public Point() {}

   public double getX() { return x; }
   public double getY() { return y; }

   public void setX(double x) {this.x = x;}
   public void setY(double y) {this.y = y;}
}
```

```
public record Point(double x, double y) {
}
```

Сеттеры не нужны

Mutable POJO vs Records

```
public final class Point {
    private double x;
    private double y;

    public Point() {}

    public double getX() { return x; }
    public double getY() { return y; }

    public void setX(double x) {this.x = x;}
    public void setY(double y) {this.y = y;}

    public void add(Point p) {
        x += p.getX();
        y += p.getY();
    }

    @Override
    public String toString() {
        return "Point{x=" + x + ", y=" + y + '}';
    }
}
```

```
public void add() {
    var p1 = new Point();
    p1.setX(1.0);
    p1.setY(2.0);

    var p2 = new Point();
    p2.setX(3.0);
    p2.setY(4.0);

    p1.add(p2);
    System.out.println(p1);
}
```

Point{x=4.0, y=6.0}

Стремитесь, чтобы недопустимые состояния были невыразимы

Mutable POJO vs Records

```
public final class Point {
    private double x;
    private double y;

    public Point(double x, double y) {
        this.x = x;
        this.y = y;
    }

    public double getX() { return x; }
    public double getY() { return y; }

    public void add(Point p) {
        x += p.getX();
        y += p.getY();
    }

    @Override
    public String toString() {
        return "Point{x=" + x + ", y=" + y + '}';
    }
}
```

```
public void add() {
    var p1 = new Point(1.0, 2.0);
    var p2 = new Point(3.0, 4.0);
    p1.add(p2);
    System.out.println(p1);
}
```

Передача параметров через конструкторы

Mutable POJO vs Records

```
public final class Point {
    private final double x;
    private final double y;

    public Point(double x, double y) {
        this.x = x;
        this.y = y;
    }

    public double getX() { return x; }
    public double getY() { return y; }

    public Point add(Point p) {
        return new Point(x + p.x, y + p.y);
    }

    @Override
    public String toString() {
        return "Point{x=" + x + ", y=" + y + '}';
    }
}
```

```
public void add() {
    var p1 = new Point(1.0, 2.0);
    var p2 = new Point(3.0, 4.0);
    var p3 = p1.add(p2);
    System.out.println(p3);
}
```

Неизменяемые данные

Mutable POJO vs Records

```
public record Point(double x, double y) {
    public Point add(Point p) {
        return new Point(x + p.x, y + p.y);
    }
}
```

```
public void add() {
    var p1 = new Point(1.0, 2.0);
    var p2 = new Point(3.0, 4.0);
    var p3 = p1.add(p2);
    System.out.println(p3);
}
```

Point{x=4.0, y=6.0}

А если много полей?

```java
public record Order(
        Long orderId,
        Long accountId,
        String accountCode,
        AccountType accountType,
        UUID clientId,
        Instant orderStartTime,
        Instant orderUpdateTime
        // , ...
        ) {}
```

```java
void example() {
    UUID clientId = UUID.fromString("a82aceea-6768-4adf-b33b-077641d114ae" )

    var newOrder = new Order(null, 3L, "A123",
            AccountType.BROKER, clientId,
            Instant.parse("2023-02-10T10:50:00Z" ),
            Instant.parse("2023-02-10T10:50:02Z" ) /* , ... */ );

    long id = save(newOrder);

    var savedOrder = new Order(id,
            newOrder.accountId(),
            newOrder.accountCode(), newOrder.accountType(),
            newOrder.clientId(),
            newOrder.orderStartTime(), newOrder.orderUpdateTime()
            /* , ... */ );
}
```

А если много полей? - Декомпозировать

```
public record Order(
        Long id,
        Account account,
        Instant startTime,
        Instant updateTime
        // , ...
        ) { }

public record Account(
        Long id,
        String code,
        AccountType type,
        UUID clientId) { }
```

```
void example() {
    UUID clientId = UUID.fromString("a82aceea-6768-4adf-b33b-077641d114ae" );

    Account account = new Account(3L, "A123", AccountType.BROKER, clientId);

    var newOrder = new Order(null, account,
            Instant.parse("2023-02-10T10:50:00Z"),
            Instant.parse("2023-02-10T10:50:02Z") /* , ... */ );

    long id = save(newOrder);

    var savedOrder = new Order(id, newOrder.account(),
            newOrder.startTime(), newOrder.updateTime()
            /* , ... */ );
}
```

Декомпозировать объекты полезно в любом случае

- Уменьшение когнитивной нагрузки (см. Магическое число семь плюс-минус два)
- Композиция предпочтительнее наследования реализации
  (см. Effective Java, Item 18: Favor composition over inheritance /
  4.4. Предпочитайте композицию наследованию)

А если много полей? - Декомпозировать

```
public record Order(
       Stored<Account> account,
       Instant startTime,
       Instant updateTime
       // , ...
       ) { }


public record Account(
       String code,
       AccountType type,
       UUID clientId) {}
```

```
void example() {
   UUID clientId = UUID.fromString(
      "a82aceea-6768-4adf-b33b-077641d114ae ");

   var account = new Stored<>(3L,
          new Account("A123", AccountType.BROKER, clientId));

   var newOrder = new Order(account,
          Instant.parse("2023-02-10T10:50:00Z"),
          Instant.parse("2023-02-10T10:50:02Z") /* , ... */ );

   long id = save(newOrder);

   var savedOrder = new Stored<>(id, newOrder);
}
```

```
public record Stored<T>(long id, T value) { }
```

А если много полей? - Билдеры

Effective Java, часть 2.2.

Когда всё-таки нужно менять

```
public record Order(
       Stored<Account> account,
       Instant startTime,
       Instant updateTime,
       int executedAmount
       // , ...
       ) { }
```

```
Stored<Order> execute(long orderId, int amount) {
   Stored<Order> storedOrder = getOrderById(orderId);
   Order order = storedOrder.value();
   Order newOrder = new Order(order.account(),
          order.startTime(), Instant.now(),
          order.executedAmount() + amount
          /* , ... */ );
   return update(new Stored<>(storedOrder.id(), newOrder));
}
```

```
public record Stored<T>(long id, T value) { }
```

Когда всё-таки нужно менять

```
public record Order(
        Stored<Account> account,
        Instant startTime,
        Instant updateTime,
        int executedAmount
        // , ...
) {
    public Order executePart(
            Instant now, int amount
    ) {
        return new Order(account,
                startTime, now,
                executedAmount + amount
                /* , ... */ );
    }
}

public record Stored<T>(long id, T value) { }
```

```
Stored<Order> execute(long orderId, int amount) {
    Stored<Order> storedOrder = getOrderById(orderId);
    Order order = storedOrder.value();
    Order newOrder = order.executePart(Instant.now(), amount);
    return update(new Stored<>(storedOrder.id(), newOrder));
}
```

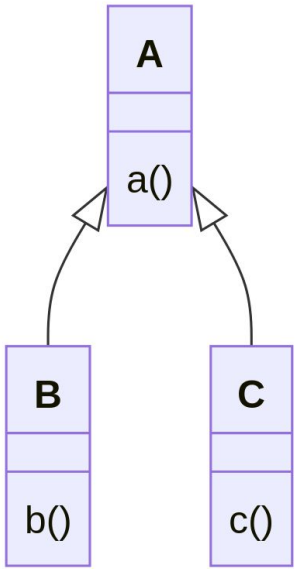
Когда всё-таки нужно менять

```
public record Order(
        Stored<Account> account,
        Instant startTime,
        Instant updateTime,
        int executedAmount
        // , ...
) {
    public Order executePart(
            Instant now, int amount
    ) {
        return new Order(account,
                startTime, now,
                executedAmount + amount
                /* , ... */ );
    }
}
```

```
Stored<Order> execute(long orderId, int amount) {
    Stored<Order> order = getOrderById(orderId);
    Stored<Order> newOrder = order.update(
            x -> x.executePart(Instant.now(), amount));
    return update(newOrder);
}
```

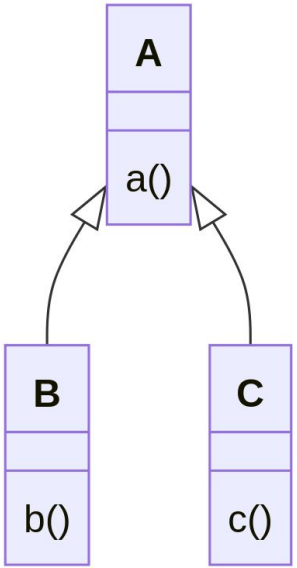
```
public record Stored<T>(long id, T value) {
    public Stored<T> update(Function<T, T> updateValue) {
        return new Stored<>(id, updateValue.apply(value));
    }
}
```

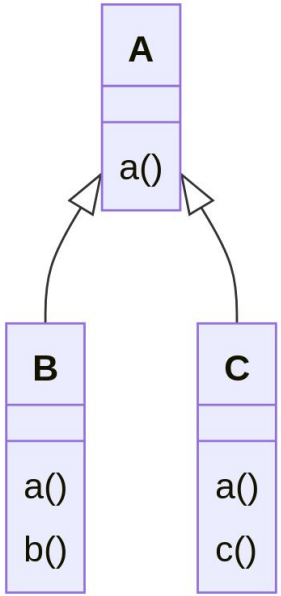
Как делать иерархии на рекордах?



А если очень хочется?

Тогда можно сделать иерархию на интерфейсах






```
interface A{
    int a();
}
record B (int a, int b) implements A {}
record C (int a, int c) implements A {}
```

Что делать, если без изменяемой структуры данных не обойтись?

- Копировать её и изолировать
- Устойчивые (persistent) структуры данных (см. vavr/clojure/scala)

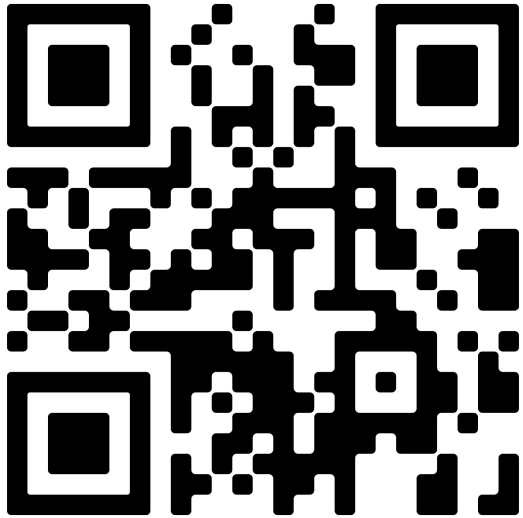
Производительность

- Преждевременная оптимизация - корень всех зол (Дональд Кнут, Эдсгер Дейкстра или Тони Хоар)
- Многие типичные шаблоны работы предполагают создание новых объектов без модификации старых (map + filter/reduce)
- Много короткоживущих объектов - это ок для современных GC (TBD пруфы)
- Устойчивые (persistent) структуры данных (Крис Окасаки)
- Мутабельные части можно изолировать

Вопросы

О чём поговорили

- Почему мутабельность это плохо?
  - Авторитеты (5 - 9)
- Почему боятся иммутабельности?
- Можно ли с этим жить?
- Примеры
  - Mutable POJO vs Records (15 - 19)
  - А если много полей? (20 - 22)
  - Когда всё-таки нужно менять (24 - 26)
- Производительность (30)



Telegram: anton0xf & progmsk
Email: anton0xf@gmail.com