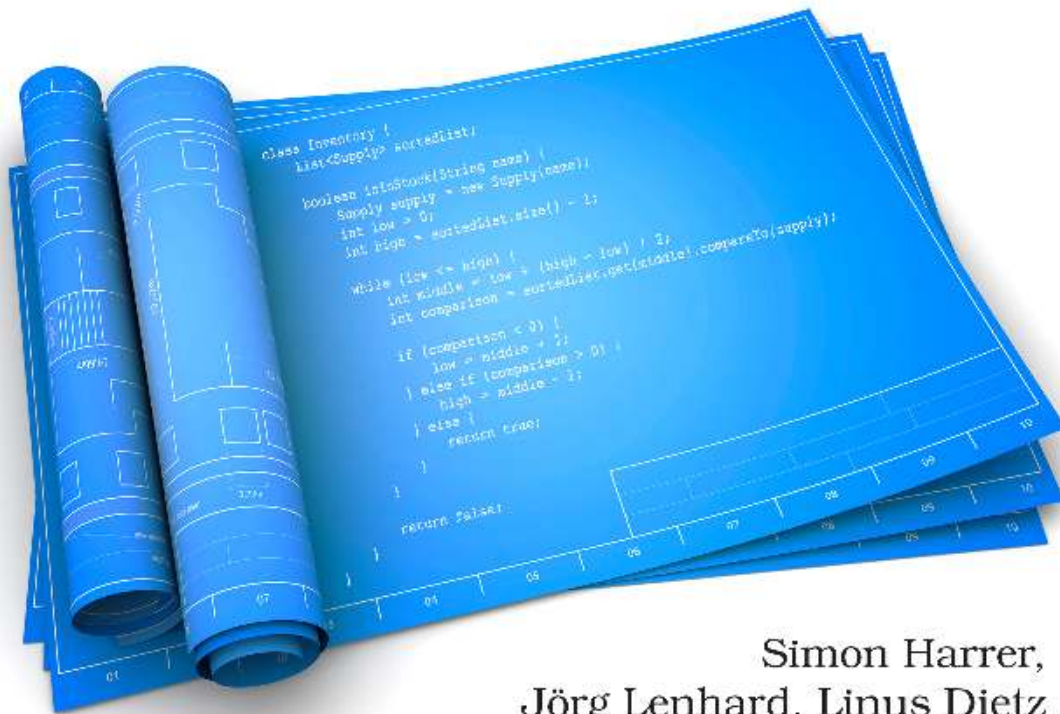


The
Pragmatic
Programmers

Java by Comparison

Become a Java Craftsman
in 70 Examples



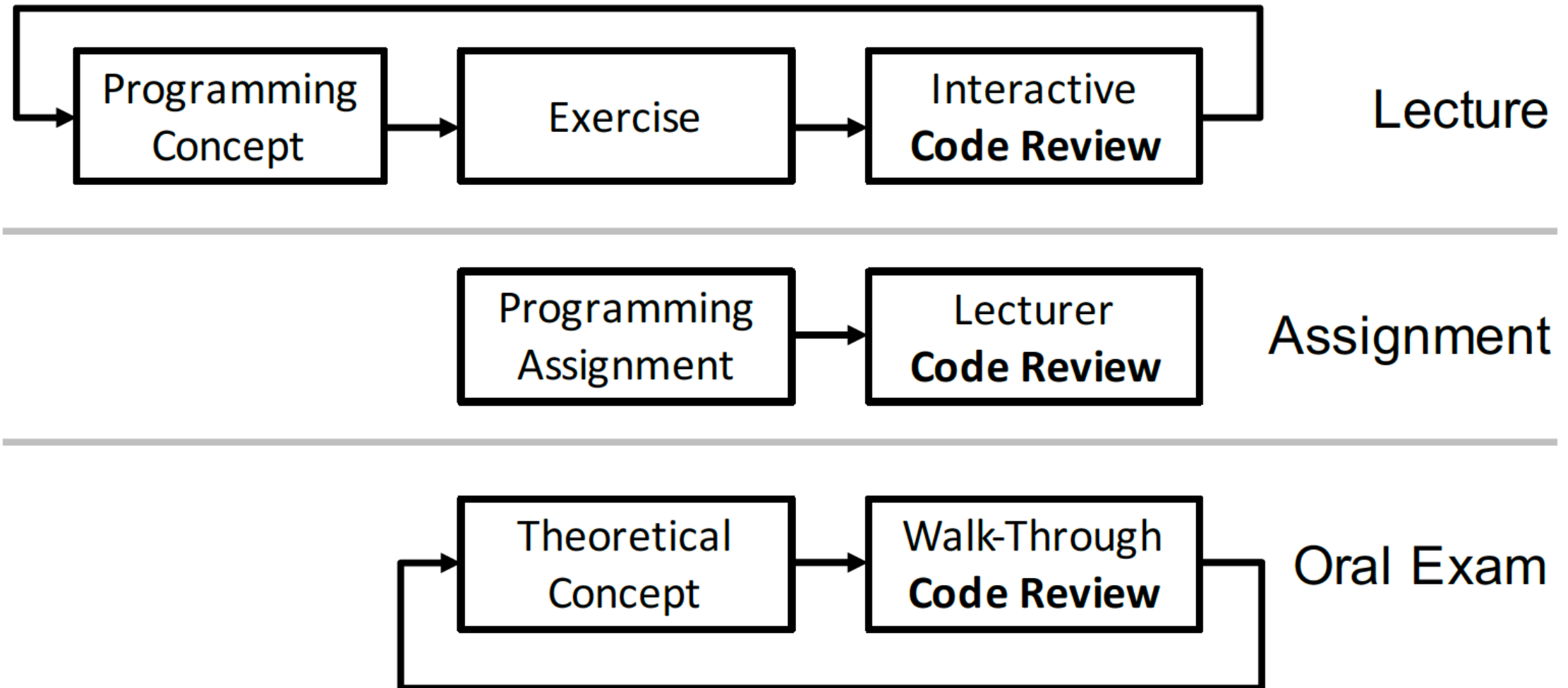
Simon Harrer,
Jörg Lenhard, Linus Dietz
*Foreword by Venkat Subramaniam
Edited by Andrea Stewart*

Clean Code in Java

Erfahrungen aus 6 Jahren
Java Lehre
an der Uni Bamberg



@simonharrer



Actionable name

Problematic
code

Problem
description

Avoid Negations

```
class Laboratory {
    Microscope microscope;

    Result analyze(Sample sample) {
        if (microscope.isInorganic(sample)) {
            return Result.INORGANIC;
        } else {
            return analyzeOrganic(sample);
        }
    }

    private Result analyzeOrganic(Sample sample) {
        if (!microscope.isHumanoid(sample)) {
            return Result.ALIEN;
        } else {
            return Result.HUMANOID;
        }
    }
}
```

Has anyone ever told you to “think positively”? Turns out, positive expressions are better in your code than negative ones because they’re often easier to grasp and they take up slightly less space.

The problem code shown here is another variant of the Laboratory component. It provides two methods that take a Sample and return a Result. There’s nothing obviously wrong with the code. It’ll fulfill its purpose, but it’s more complex than necessary.

Consider the conditions of the if statements: Both express a negating condition. The first one tests if a sample isInorganic(). The second one even uses the Boolean operator for negation: the exclamation mark ‘!’.

In most cases, you’ll find it easier to understand positive expressions when you read code. A negated expression adds one extra layer of indirection. Instead of a simple “X does apply,” you have to grasp an additional token: “X does *not* apply.”

This extra token is often unnecessary. Although it seems like a minor change, every tiny bit of simplification helps for more complex expressions (those that you’ll find in real code).

Remember: Everybody likes no negations.

Chapter 1: Start Cleaning Up • 4

Now, take a look at the simplified version:

```
class Laboratory {
    Microscope microscope;

    Result analyze(Sample sample) {
        if (microscope.isOrganic(sample)) {
            return analyzeOrganic(sample);
        } else {
            return Result.INORGANIC;
        }
    }

    private Result analyzeOrganic(Sample sample) {
        if (microscope.isHumanoid(sample)) {
            return Result.HUMANOID;
        } else {
            return Result.ALIEN;
        }
    }
}
```

We changed the code only slightly, but that already has an effect. Instead of isInorganic(), we’re calling isOrganic(), which has a positive wording instead of a negative one. That is, we have to switch the bodies of the if and else blocks.

When it comes to the second part, we’re calling the same method, isHumanoid(), but we eliminated the negation. This is a real simplification of the code. Again, it means that we have to switch the bodies of the if and else blocks.

All in all, these changes are quite simple and you might wonder: why should I bother? You should bother because it improves the understandability of the code and comes at virtually no cost. You don’t usually have to add further lines of code. All you have to do is re-sort existing parts of the code and you’ll see an improvement. It’s an opportunity that’s too good to let go!

Sometimes, however, you need the appropriate methods, like the isOrganic() instead of isInorganic() here. If the code you’re calling comes from a third-party library, you might not have the option to invoke different methods. But when you control it, don’t shy away from adding that method to the appropriate class—the few lines of code are worth their characters, because they make your code clearer in other places. In the long run you’ll end up with less code, since such methods reduce code duplication and can be reused in other parts of the program. From our experience, it’s best to get rid of the negative method completely—no need to maintain two similar methods.

Avoid Negations • 5

Solution
code

Solution
rationale

```
class Laboratory {  
    Microscope microscope;  
  
    Result analyze(Sample sample) {  
        if (microscope.isInorganic(sample) == true) {  
            return Result.INORGANIC;  
        } else {  
            return analyzeOrganic(sample);  
        }  
    }  
  
    private Result analyzeOrganic(Sample sample) {  
        if (microscope.isHumanoid(sample) == false) {  
            return Result.ALIEN;  
        } else {  
            return Result.HUMANOID;  
        }  
    }  
}
```

```
class Laboratory {  
    Microscope microscope;  
    Result analyze(Sample sample) {
```

```
class Laboratory {  
    Microscope microscope;  
  
    Result analyze(Sample sample) {  
        if (microscope.isInorganic(sample)) {  
            return Result.INORGANIC;  
        } else {  
            return analyzeOrganic(sample);  
        }  
    }  
  
    private Result analyzeOrganic(Sample sample) {  
        if (!microscope.isHumanoid(sample)) {  
            return Result.ALIEN;  
        } else {  
            return Result.HUMANOID;  
        }  
    }  
}
```

Avoid Unnecessary Comparisons

```
class Laboratory {  
    Microscope microscope;  
    Result analyze(Sample sample) {
```

```
class Laboratory {  
    Microscope microscope;  
    Result analyze(Sample sample) {  
        if (microscope.isOrganic(sample)) {  
            return analyzeOrganic(sample);  
        } else {  
            return Result.INORGANIC;  
        }  
    }  
  
    private Result analyzeOrganic(Sample sample) {  
        if (microscope.isHumanoid(sample)) {  
            return Result.HUMANOID;  
        } else {  
            return Result.ALIEN;  
        }  
    }  
}
```

Avoid Negations

```
class Astronaut {  
    String name;  
    int missions;  
  
    boolean isValid() {  
        if (missions < 0 || name == null || name.trim().isEmpty()) {  
            return false;  
        } else {  
            return true;  
        }  
    }  
}
```

```
class Astronaut {  
    String name;  
    int missions;  
  
    boolean isValid() {  
        if (missions < 0 || name == null || name.trim().isEmpty()) {  
            return false;  
        } else {  
            return true;  
        }  
    }  
}
```

```
class Astronaut {  
    String name;  
    int missions;  
  
    boolean isValid() {  
        return missions >= 0 && name != null && !name.trim().isEmpty();  
    }  
}
```

Return Boolean Expressions Directly

```
class BoardComputer {  
    CruiseControl cruiseControl;  
  
    void authorize(User user) {  
        Objects.requireNonNull(user);  
        if (user.isUnknown())  
            cruiseControl.logUnauthorizedAccessAttempt();  
        if (user.isAstronaut())  
            cruiseControl.grantAccess(user);  
        if (user.isCommander())  
            cruiseControl.grantAccess(user);  
            cruiseControl.grantAdminAccess(user);  
        }  
    }  
}
```

```
class BoardComputer {  
    CruiseControl cruiseControl;  
  
    void authorize(User user) {  
        Objects.requireNonNull(user);  
    }  
}
```

```
class BoardComputer {  
    CruiseControl cruiseControl;  
  
    void authorize(User user) {  
        Objects.requireNonNull(user);  
        if (user.isUnknown()) {  
            cruiseControl.logUnauthorizedAccessAttempt();  
        }  
        if (user.isAstronaut()) {  
            cruiseControl.grantAccess(user);  
        }  
        if (user.isCommander()) {  
            cruiseControl.grantAccess(user);  
        }  
        cruiseControl.grantAdminAccess(user); // SECURITY THREAT  
    }  
}
```

Always Use Braces

```
class MainSpaceShipManager {  
    AbstractRocketPropulsionEngine abstractRocketPropulsionEngine;  
    INavigationController navigationController;  
    boolean turboEnabledFlag;  
  
    void navigateSpaceShipTo(PlanetInfo planetInfo) {  
        RouteData data = navigationController.calculateRouteData(planetInfo);  
        LogHelper.logRouteData(data);  
        abstractRocketPropulsionEngine.invokeTask(data, turboEnabledFlag);  
    }  
}
```

```
class MainSpaceShipManager {
    AbstractRocketPropulsionEngine abstractRocketPropulsionEngine;
    INavigationController navigationController;
    boolean turboEnabledFlag;

    void navigateSpaceShipTo(PlanetInfo planetInfo) {
        RouteData data = navigationController.calculateRouteData(planetInfo);
        LogHelper.logRouteData(data);
        abstractRocketPropulsionEngine.invokeTask(data, turboEnabledFlag);
    }
}
```

```
class SpaceShip {
    Engine engine;
    Navigator navigator;
    boolean turboEnabled;

    void navigateTo(Planet destination) {
        Route route = navigator.calculateRouteTo(destination);
        Logger.log(route);
        engine.follow(route, turboEnabled);
    }
}
```

Avoid Meaningless Terms

```
class Supply {  
    /**  
     * The code universally identifies a supply.  
     *  
     * It follows a strict format, beginning with an S (for supply), followed  
     * by a five digit inventory number. Next comes a backslash that  
     * separates the country code from the preceding inventory number. This  
     * country code must be exactly two capital letters standing for one of  
     * the participating nations (US, EU, RU, CN). After that follows a dot  
     * and the actual name of the supply in lowercase letters.  
     */  
    static final Pattern CODE =  
        Pattern.compile("^S\\d{5}\\\\\\\\(US|EU|RU|CN)\\\\\\\\.[a-z]+$");  
}
```

```
class Supply {  
    /**  
     * The code universally identifies a supply.
```

```
class Supply {  
    /**  
     * The expression universally identifies a supply code.  
     *  
     * Format: "S<inventory-number>\<COUNTRY-CODE>.<name>"  
     *  
     * Valid examples: "S12345\US.pasta", "S08342\CN.wrench",  
     * "S88888\EU.laptop", "S12233\RU.brush"  
     *  
     * Invalid examples:  
     * "R12345\RU.fuel"           (Resource, not supply)  
     * "S1234\US.light"           (Need five digits)  
     * "S01234\AI.coconut"        (Wrong country code. Use US, EU, RU, or CN)  
     * " S88888\EU.laptop "      (Trailing whitespaces)  
    */  
    static final Pattern SUPPLY_CODE =  
        Pattern.compile("^S\\d{5}\\\\\\\\(US|EU|RU|CN)\\\\\\\\.[a-z]+$");  
}
```

Document Using Examples

```
class Mission {  
  
    Logbook logbook;  
    LocalDate start;  
  
    void update(String author, String message) {  
        LocalDate today = LocalDate.now();  
        String month = String.valueOf(today.getMonthValue());  
        String formattedMonth = month.length() < 2 ? "0" + month : month;  
        String entry = author.toUpperCase() + ": [" + formattedMonth + "-" +  
            today.getDayOfMonth() + "-" + today.getYear() + "](Day " +  
            (ChronoUnit.DAYS.between(start, today) + 1) + ")> " +  
            message + System.lineSeparator();  
        logbook.write(entry);  
    }  
}
```

```

class Mission {

    Logbook logbook;
    LocalDate start;

    void update(String author, String message) {
        LocalDate today = LocalDate.now();
        String month = String.valueOf(today.getMonthValue());
        String formattedMonth = month.length() < 2 ? "0" + month : month;
        String entry = author.toUpperCase() + ": [" + formattedMonth + "-" +

```

```

class Mission {

    Logbook logbook;
    LocalDate start;

    void update(String author, String message) {
        final LocalDate today = LocalDate.now();
        String entry = String.format("%S: [%tm-%<te-%<tY](Day %d)> %s%n",
            author, today,
            ChronoUnit.DAYS.between(start, today) + 1, message);
        logbook.write(entry);
    }
}

```

Favor Format Over Concatenation

```
class Hull {  
    int holes;  
}  
  
class HullRepairUnit {  
    void repairHole(Hull hull) {  
        if (isIntact(hull)) {  
            return;  
        }  
        hull.holes--;  
    }  
  
    boolean isIntact(Hull hull) {  
        return hull.holes == 0;  
    }  
}
```

```
class Hull {
    int holes;
}

class HullRepairUnit {
    void repairHole(Hull hull) {
        if (isIntact(hull)) {
            return;
        }
    }
}
```

```
class Hull {
    int holes;

    void repairHole() {
        if (isIntact()) {
            return;
        }
        holes--;
    }

    boolean isIntact() {
        return holes == 0;
    }
}
```

Combine State and Behaviour

```
class Inventory {  
    private final List<Supply> supplies;  
    Inventory(List<Supply> supplies) {  
        this.supplies = supplies;  
    }  
    List<Supply> getSupplies() {  
        return supplies;  
    }  
}
```

```
class Inventory {  
    private final List<Supply> supplies;  
    Inventory(List<Supply> supplies) {  
        this.supplies = supplies;  
    }  
    List<Supply> getSupplies() {  
        return supplies;  
    }  
}
```

```
class Inventory {  
    private final List<Supply> supplies;  
    Inventory(List<Supply> supplies) {  
        this.supplies = new ArrayList<>(supplies);  
    }  
    List<Supply> getSupplies() {  
        return Collections.unmodifiableList(supplies);  
    }  
}
```

Avoid Leaking References

```
class Inventory {  
    List<Supply> supplies = new ArrayList<>();  
    long countDifferentKinds() {  
        return supplies.stream()  
            .filter(supply -> !supply.isContaminated())  
            .map(supply -> supply.getName())  
            .distinct()  
            .count();  
    }  
}
```

```
class Inventory {  
    List<Supply> supplies = new ArrayList<>();  
    long countDifferentKinds() {  
        return supplies.stream()  
            .filter(supply -> !supply.isContaminated())  
            .map(supply -> supply.getName())  
            .distinct()  
            .count();  
    }  
}
```

```
class Inventory {  
    List<Supply> supplies = new ArrayList<>();  
    long countDifferentKinds() {  
        return supplies.stream()  
            .filter(Supply::isUncontaminated)  
            .map(Supply::getName)  
            .distinct()  
            .count();  
    }  
}
```

Favor Method References Over Lambdas

```
class NameTag {  
    final String name;  
  
    NameTag(String fullName) {  
        this.name = parse(fullName).toUpperCase();  
    }  
  
    String parse(String fullName) {  
        String[] components = fullName.split("[,| ]");  
        if (components == null || components.length < 2) {  
            return fullName;  
        }  
        if (fullName.contains(",")) {  
            return components[0];  
        } else {  
            return components[components.length - 1];  
        }  
    }  
}
```

```
class NameTag {  
    final String name;  
  
    NameTag(String fullName) {  
        this.name = parse(fullName).toUpperCase();  
    }  
  
    String parse(String fullName) {  
        String[] components = fullName.split("[,| ]");  
        if (components == null || components.length < 2) {  
            return fullName;  
        }  
        if (fullName.contains(",")) {  
            return components[0];  
        } else {  
            return components[components.length - 1];  
        }  
    }  
}
```

Name

Neil Alden Armstrong

Edwin Eugene Aldrin, Jr.

費俊龍

```

class NameTag {

    final String name;

    NameTag(String fullName) {
        this.name = parse(fullName).toUpperCase();
    }

    String parse(String fullName) {
        String[] components = fullName.split("[,| ]");
        if (components == null || components.length < 2) {
            return fullName;
        }
        if (fullName.contains(",")) {
            return components[0];
        } else {
            return components[components.length - 1];
        }
    }
}

```

<i>Name</i>	<i>Computed Tag</i>	<i>Expected Tag</i>	<i>Correct?</i>
Neil Alden Armstrong	ARMSTRONG	ARMSTRONG	✓
Edwin Eugene Aldrin, Jr.	EDWIN	ALDRIN	✗
費俊龍	費俊龍	費	✗

```

class NameTag {
    final String name;

    NameTag(String fullName) {
        this.name = parse(fullName).toUpperCase();
    }

    String parse(String fullName) {
        String[] components = fullName.split("[,| ]");
        if (components == null || components.length < 2) {
            return fullName;
        }
    }
}

```

```

class NameTag {
    final String name;

    NameTag(String name) {
        Objects.requireNonNull(name);
        this.name = name;
    }
}

```

<i>Name</i>	<i>Computed Tag</i>	<i>Expected Tag</i>	<i>Correct?</i>
Neil Alden Armstrong	ARMSTRONG	ARMSTRONG	✓
Edwin Eugene Aldrin, Jr.	EDWIN	ALDRIN	✗
費俊龍	費俊龍	費	✗

Wie bringt ihr Clean Code euren Azubis und Juniors bei?