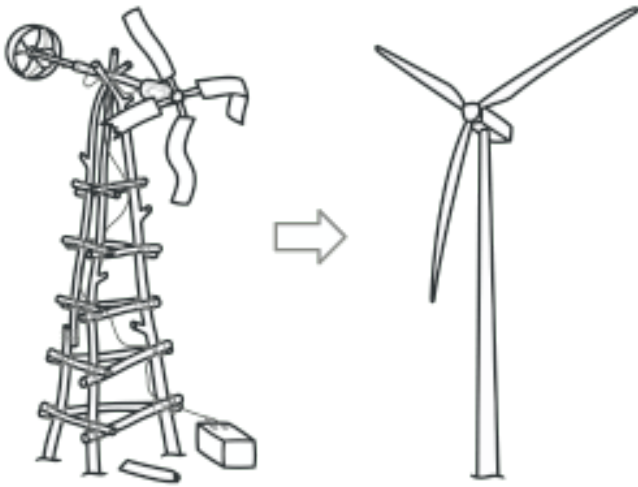


Refactoring

Introduction



Easy example (before)

```
public boolean max(int a, int b) {  
    if(a > b) {  
        return true;  
    } else if (a == b) {  
        return false;  
    } else {  
        return false;  
    }  
}
```

Easy example (after)

```
public boolean max(int a, int b) {  
    return a > b;  
}
```

Why ?

- Improve **understanding** of code
- Help find and fix **bugs**
- Accelerate the **speed** of development
- Improve **design**

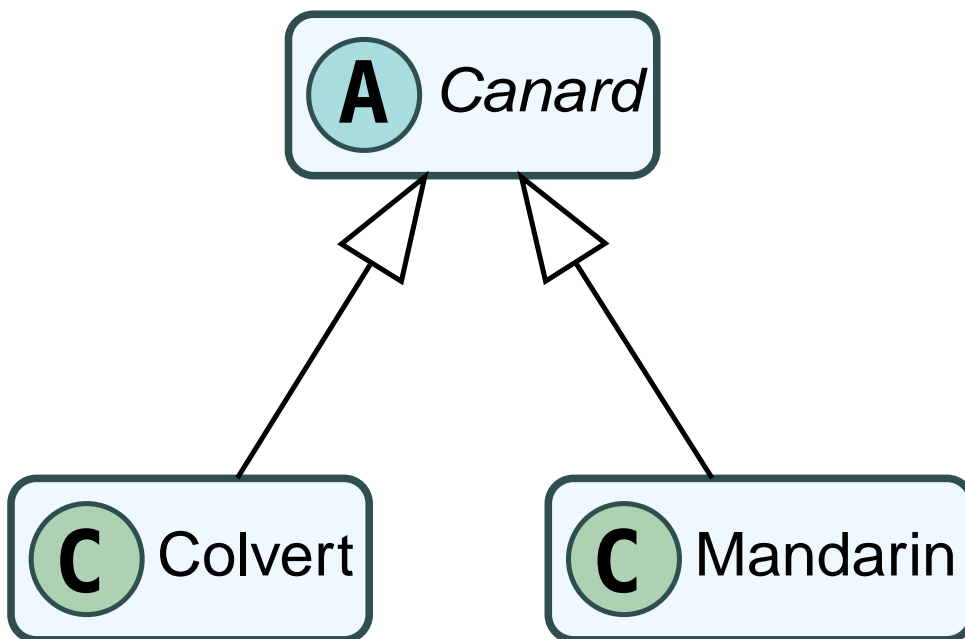
Code smells (basic)

- Large class
- Long methods
- Too many parameters
- Using a lot of primitive data types (*String*, *float*, ...)

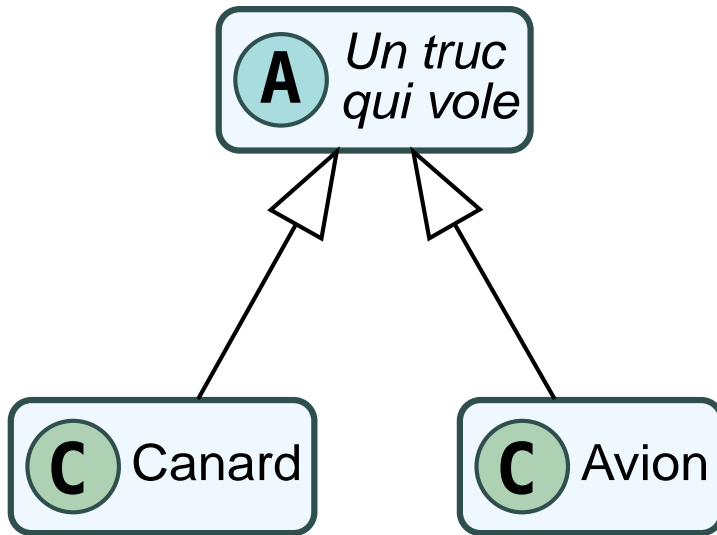
Code smells (OO)

- Switch case
- Temporary field (*null*)
- Parallel inheritance hierarchies
- Wrong use of inheritance

Use of inheritance



Wrong use of inheritance



Garbage Smells

- Too many comments
- Duplicated code
- Lazy class
- Unused code

Excessive connectivity

- External methods
- Dependency with another class' implementation details
- Long class call

Refactoring techniques

- Extract a class
- Extract a method
- Passing an entire object

Extract a class

```
class Human {  
    private String name;  
    private String age;  
    private String country;  
    private String city;  
    private String street;  
    private String house;  
    private String quarter;  
  
    public String getFullAddress() {  
        StringBuilder result = new StringBuilder();  
        return result  
            .append(country)  
            .append(", ")  
            .append(city)  
            .append(", ")  
            .append(street)  
            .append(", ")  
            .append(house)  
            .append(" ")  
            .append(quarter).toString();  
    }  
}
```

!

```
class Human {
    private String name;
    private String age;
    private Address address;

    private String getFullAddress() {
        return address.getFullAddress();
    }
}

class Address {
    private String country;
    private String city;
    private String street;
    private String house;
    private String quarter;

    public String getFullAddress() {
        StringBuilder result = new StringBuilder();
        return result
            .append(country)
            .append(", ")
            .append(city)
            .append(", ")
            .append(street)
            .append(", ")
            .append(house)
            .append(" ")
            .append(quarter).toString();
    }
}
```

Extract a method

```

public void calcQuadraticEq(double a, double b, double c) {
    double D = b * b - 4 * a * c;
    if (D > 0) {
        double x1, x2;
        x1 = (-b - Math.sqrt(D)) / (2 * a);
        x2 = (-b + Math.sqrt(D)) / (2 * a);
        System.out.println("x1 = " + x1 + ", x2 = " + x2);
    }
    else if (D == 0) {
        double x;
        x = -b / (2 * a);
        System.out.println("x = " + x);
    }
    else {
        System.out.println("Equation has no roots");
    }
}

```

!

```

public void calcQuadraticEq(double a, double b, double c) {
    double D = b * b - 4 * a * c;
    if (D > 0) {
        dGreaterThanZero(a, b, D);
    }
    else if (D == 0) {
        dEqualsZero(a, b);
    }
    else {
        dLessThanZero();
    }
}

```

Passing an entire object

```
public void employeeMethod(Employee employee) {
    // Some actions
    double yearlySalary = employee.getYearlySalary();
    double awards = employee.getAwards();
    double monthlySalary = getMonthlySalary(yearlySalary, awards);
    // Continue processing
}

public double getMonthlySalary(double yearlySalary, double awards) {
    return (yearlySalary + awards)/12;
}
```

!

```
public void employeeMethod(Employee employee) {
    // Some actions
    double monthlySalary = getMonthlySalary(employee);
    // Continue processing
}

public double getMonthlySalary(Employee employee) {
    return (employee.getYearlySalary() + employee.getAwards())/12;
}
```

Advice for the project

Issues to list the ideas

Filters

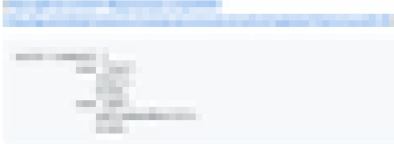
☐ **5 Open** ☒ 0 Closed

- ☐ **Apply the "functional" design pattern** **refactoring**
#6 opened 15 days ago by jmbruel 0 of 3
- ☐ **Make app...** **refactoring**
#5 opened 15 days ago by jmbruel 0 of 3
- ☐ **Refactoring...** **refactoring**
#4 opened 15 days ago by jmbruel 0 of 3
- ☐ **Refactor...** **refactoring**
#2 opened 20 days ago by jmbruel 0 of 3
- ☐ **Continuous Integration**
#1 opened 20 days ago by jmbruel 0 of 3

Describe / Explain each idea

jmbruel commented 15 days ago

Problem:
Comments are not aligned

Example:


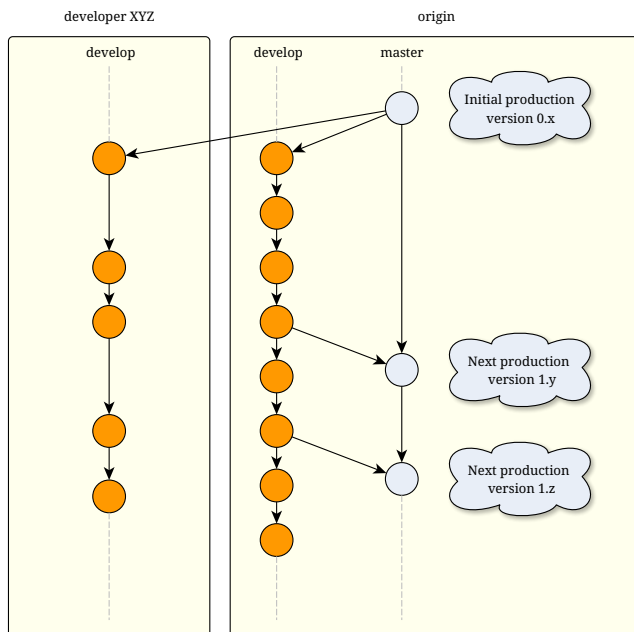
Argument:
Comments are not aligned

Proposed Solution:
Apply the "functional" design pattern
See also if it solves issue #4.

Check before merge:
(taken from <https://refactoring.guru/refactoring/how-to>)

- ☐ The code should become cleaner
- ☐ New functionality shouldn't be created during refactoring
- ☐ All existing tests must pass after refactoring.

1 refactoring = 1 branch



Professional README

[exempleProjet2020.pdf](#)

Ressources

- <https://refactoring.guru/fr/refactoring>
- <https://codegym.cc/groups/posts/196-how-refactoring-works-in-java>

Ready for a quiz?

make refa