

A decorative frame consisting of thin grey lines. A vertical line on the left and a horizontal line at the top intersect at a small circle in the top-left corner. Another horizontal line is positioned below the word 'Refactoring'. A vertical line on the right and a horizontal line at the bottom intersect at a small circle in the bottom-right corner.

Refactoring

*trimming your design
for agile development*

Agenda

- What is refactoring?
- Why is refactoring needed ?
- Support for Agile Development
- Getting started
- Discussion

What Is Refactoring ?

Refactoring:

Improving the design of existing code without changing the code's observable behavior.

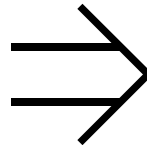
- Making structural changes
- Without behavioral changes

Replace Magic Number With Symbolic Constant

```
id = -1;  
for ( int i = 0; i < members.length; i++ ) {  
    if ( name.equals( persons[i].name() ) ) {  
        id = i;  
        break;  
    }  
}
```

// ...

```
if ( id != -1 ) {  
    displayAddress( persons[id] );  
} else {  
    displayUnknown();  
}
```



```
private static final int UNKNOWN = -1;
```

// ...

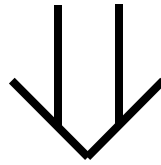
```
id = UNKNOWN;  
for ( int i = 0; i < members.length; i++ ) {  
    if ( name.equals( persons[i].name() ) ) {  
        id = i;  
        break;  
    }  
}
```

// ...

```
if ( id != UNKNOWN ) {  
    displayAddress( persons[id] );  
} else {  
    displayUnknown();  
}
```

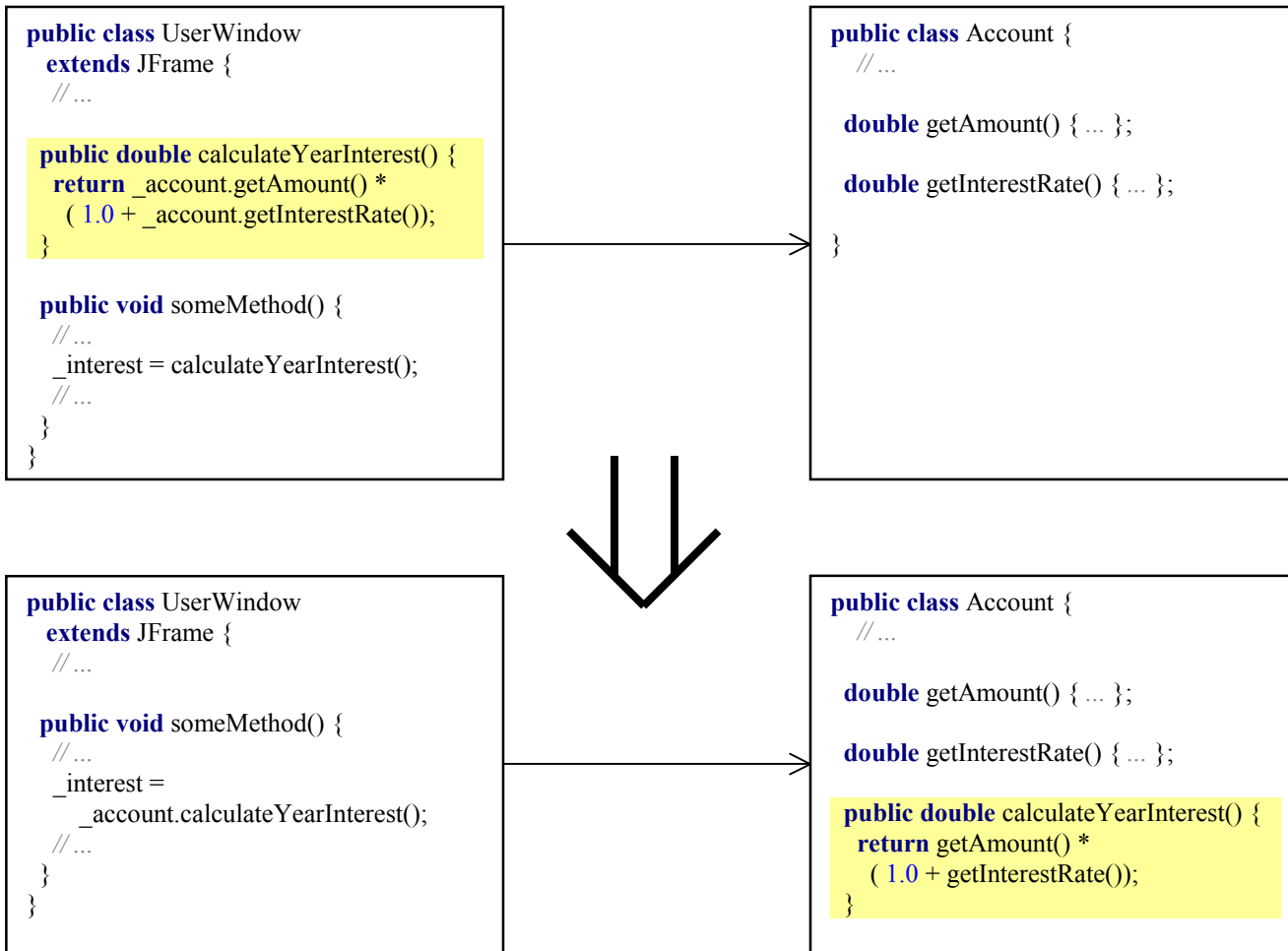
Inline Temp

```
public double dayInterest( double amount ) {  
    double interestRate = interestRate(amount);  
  
    return amount * interestRate / 365;  
}
```

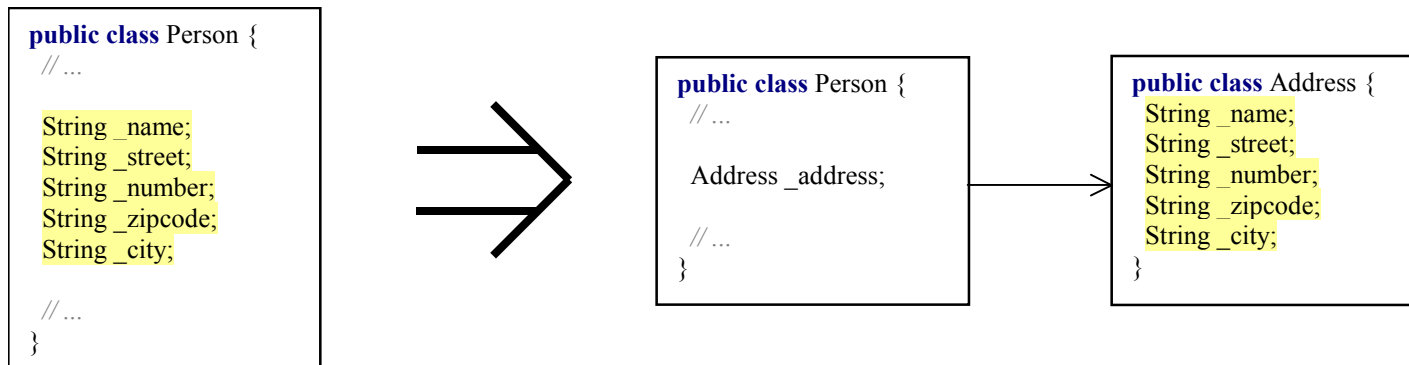


```
public double dayInterest( double amount ) {  
    return amount * interestRate(amount) / 365;  
}
```

Move Method

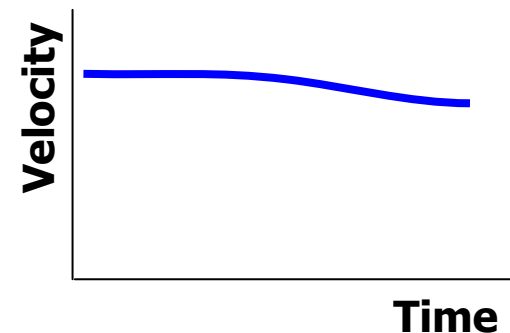
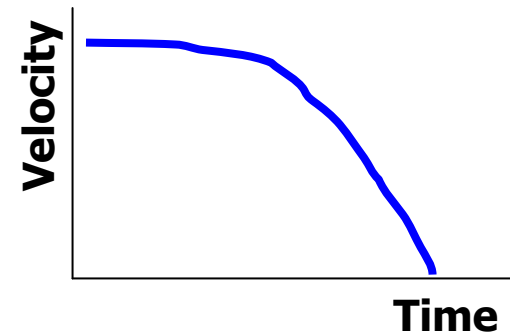


Extract Class



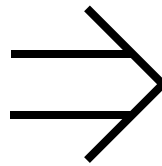
Why Is Refactoring Needed ?

- Code complexity tends to increase rapidly over time
 - Complexity slows down development
- Refactoring helps keeping complexity under control
 - Allowing development to proceed more rapidly



Code Complexity

```
private boolean isValid(String[] items) {  
    boolean flag = true;  
  
    for (int i = 0; i < items.length; i++) {  
        String item = items[i];  
        if (( item != null && item.length() != 0))  
            continue;  
  
        flag = false;  
        break;  
    }  
  
    return flag;  
}
```

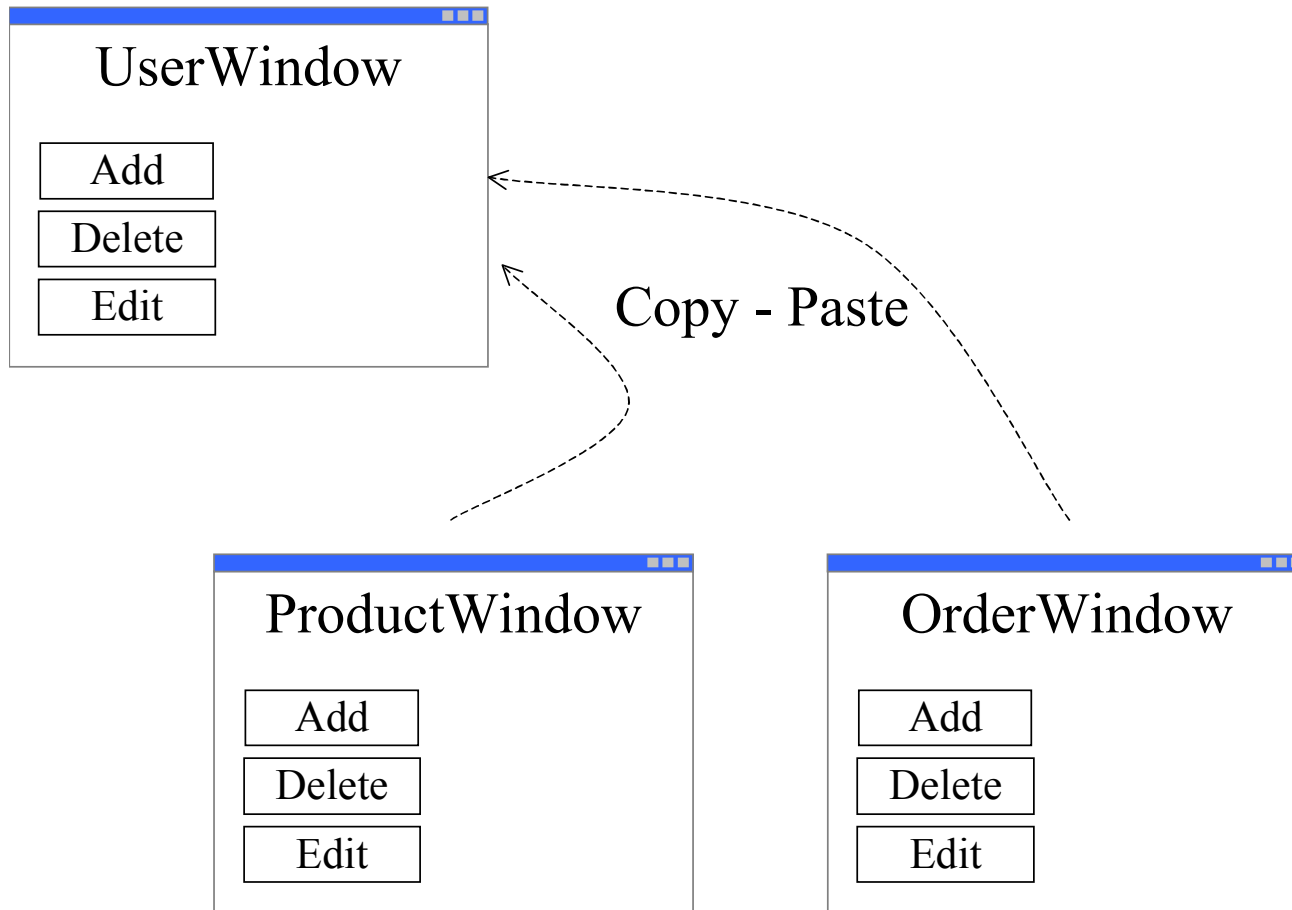


```
private boolean isValid(String[] items) {  
    for (int i = 0; i < items.length; i++) {  
        String item = items[i];  
        if ( item == null || item.length() == 0) {  
            return false;  
        }  
    }  
  
    return true;  
}
```

- Well-factored code:
 - Easier, Simpler, Cheaper

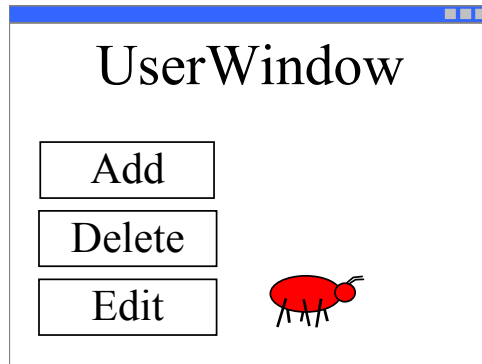
Duplicate Code

1 / 3

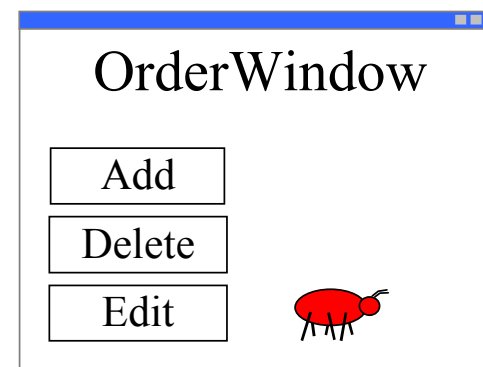
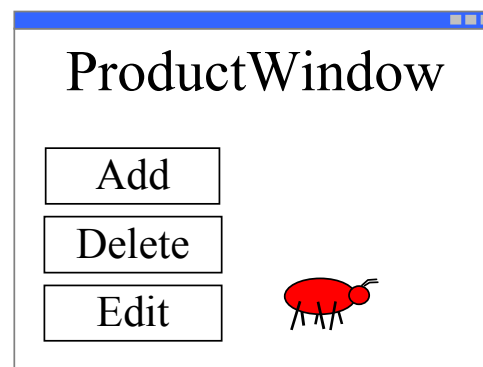


Duplicate Code

2 / 3

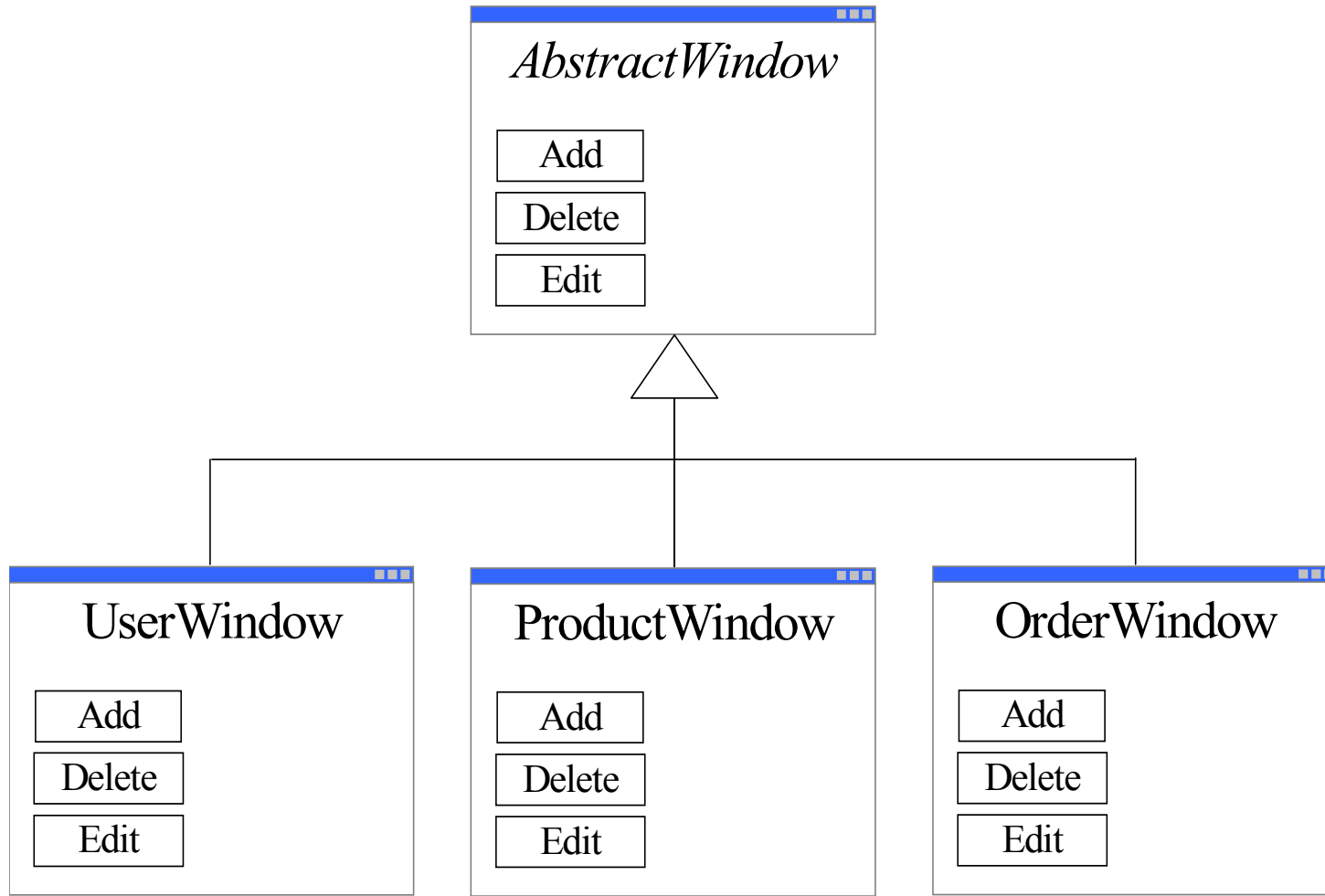


- Bugs copied as well
- Increased maintenance



Duplicate Code

3 / 3



Refactoring Detailed

- Code, once written, will be read and modified frequently
 - Refactoring is risky, it requires changes to working code.
- ⇒ Do it systematically, in small steps!

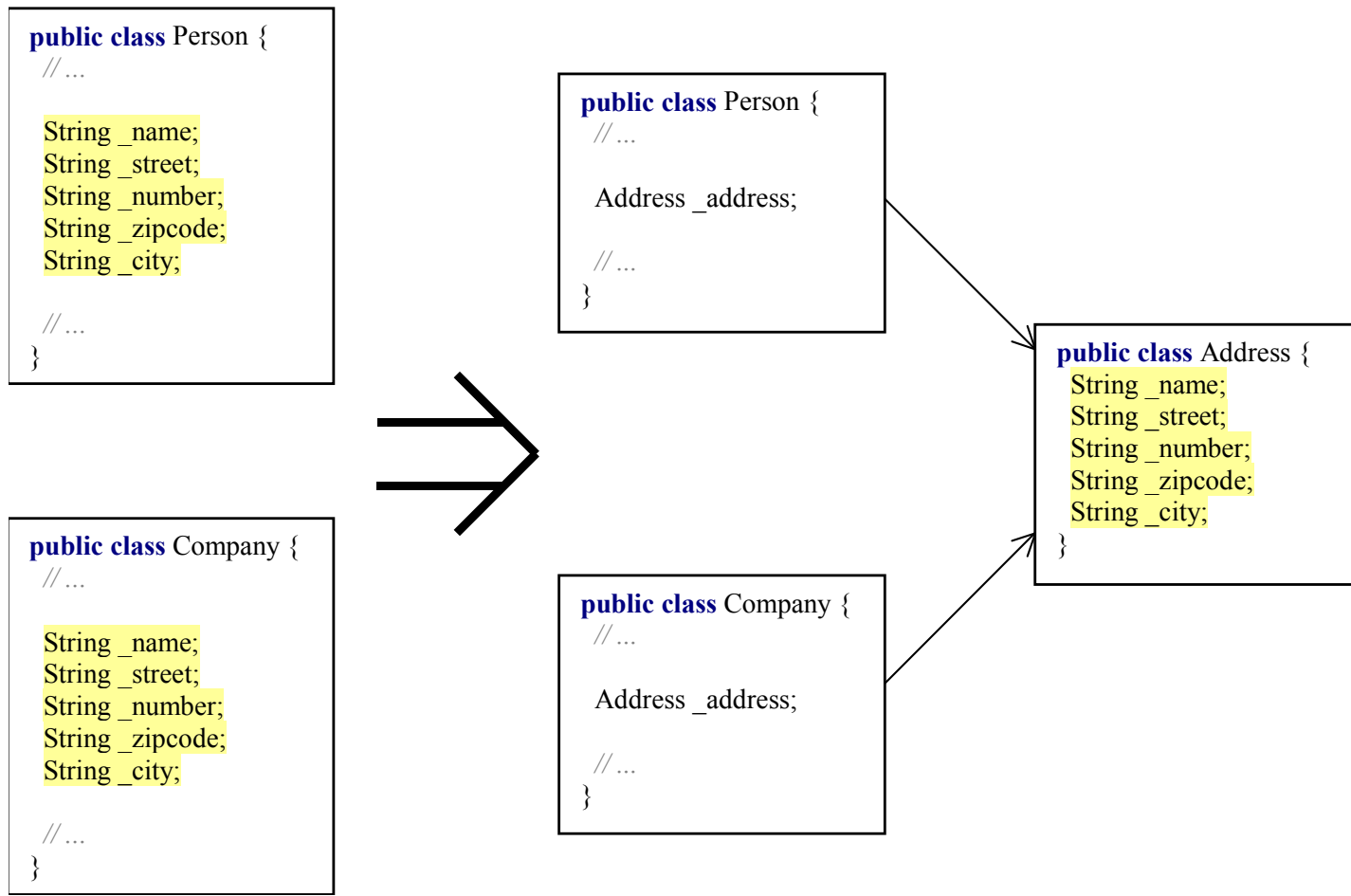
Covered By Unit Tests

- Refactoring without unit test is risky
 - We might introduce regressions
- Therefore,
 - Cover each class with unit tests
 - During a refactoring unit tests indicate if something is broken.

Towards A Simple Design

- The right design for the software at any given time is the one that:
 1. Runs all the tests
 2. Has no duplicate logic.
 3. States every intention important to the programmers
 4. Has the fewest possible classes and methods

Extract Class



Series Of Small Changes

- Refactoring implies working *incrementally*
- Making changes to the program in *small steps*
- In between run the unit tests *regularly*
- If you make a mistake it's easy to back out.

Small Steps

1 / 7

```
public double dayInterest( double amount ) {
    double interest = 0.0;
    if ( amount >= 250000.0 ) {
        interest = amount * creditInterestRate / 365;
        interest += amount * bonusInterestRate / 365;
    } else if ( amount >= 0.0 ) {
        interest = amount * creditInterestRate / 365;
    } else {
        interest = amount * debetInterestRate / 365;
    }
    return interest;
}
```

- Nearly identical code within branches
 - Start by making duplication more explicit

Small Steps

2 / 7

```
public double dayInterest( double amount ) {  
    double interest = 0.0;  
    if ( amount >= 250000.0 ) {  
        interest = amount * ( creditInterestRate + bonusInterestRate ) / 365;  
    } else if ( amount >= 0.0 ) {  
        interest = amount * creditInterestRate / 365;  
    } else {  
        interest = amount * debetInterestRate / 365;  
    }  
  
    return interest;  
}
```

- Interest formula can now be made identical
 - Separate similarities from differences

Small Steps

3 / 7

```
public double dayInterest( double amount ) {  
    double interest = 0.0;  
    if ( amount >= 250000.0 ) {  
        double interestRate = ( creditInterestRate + bonusInterestRate );  
        interest = amount * interestRate / 365;  
    } else if ( amount >= 0.0 ) {  
        double interestRate = creditInterestRate;  
        interest = amount * interestRate / 365;  
    } else {  
        double interestRate = debetInterestRate;  
        interest = amount * interestRate / 365;  
    }  
  
    return interest;  
}
```

- Interest formula is now identical
 - Move it out of the conditional

Small Steps

4 / 7

```
public double dayInterest( double amount ) {
    double interest = 0.0;
    double interestRate;
    if ( amount >= 250000.0 ) {
        interestRate = ( creditInterestRate + bonusInterestRate );
    } else if ( amount >= 0.0 ) {
        interestRate = creditInterestRate;
    } else {
        interestRate = debetInterestRate;
    }
    interest = amount * interestRate / 365;

    return interest;
}
```

- What's the use of the interest variable ?
 - Inline it and return the result directly

Small Steps

5 / 7

```
public double dayInterest( double amount ) {
    double interestRate;
    if ( amount >= 250000.0 ) {
        interestRate = ( creditInterestRate + bonusInterestRate );
    } else if ( amount >= 0.0 ) {
        interestRate = creditInterestRate;
    } else {
        interestRate = debetInterestRate;
    }

    return amount * interestRate / 365;
}
```

- Now what does the conditional logic mean ?
 - Extract a method with a name that communicates the code's intent

Small Steps

6 / 7

```
public double dayInterest( double amount ) {
    double interestRate = interestRate(amount);

    return amount * interestRate / 365;
}

private double interestRate(double amount) {
    double interestRate;
    if ( amount >= 250000.0 ) {
        interestRate = ( creditInterestRate + bonusInterestRate );
    } else if ( amount >= 0.0 ) {
        interestRate = creditInterestRate;
    } else {
        interestRate = debetInterestRate;
    }
    return interestRate;
}
```

- Inline interestRate in both methods

Small Steps

7 / 7

```
public double dayInterest( double amount ) {  
    return amount * interestRate(amount) / 365;  
}
```

```
private double interestRate(double amount) {  
    if ( amount >= 250000.0 ) {  
        return ( creditInterestRate + bonusInterestRate );  
    } else if ( amount >= 0.0 ) {  
        return creditInterestRate;  
    } else {  
        return debetInterestRate;  
    }  
}
```

- As a result of the refactoring we :
 - Eliminated the code duplication
 - Made the code more readable

Agile Software Development

Refactoring enables Agile Software Development by keeping the code

- Minimal
- Readable
- Tested

Agile Software Development

Keeping the code minimal means:

- Less code to maintain
- Changes are local to one place
- Team travels light

Agile Software Development

Keeping the code readable means:

- Faster and easier to comprehend
- Better understanding of the code
- Less mistakes and hence less bugs

Agile Software Development

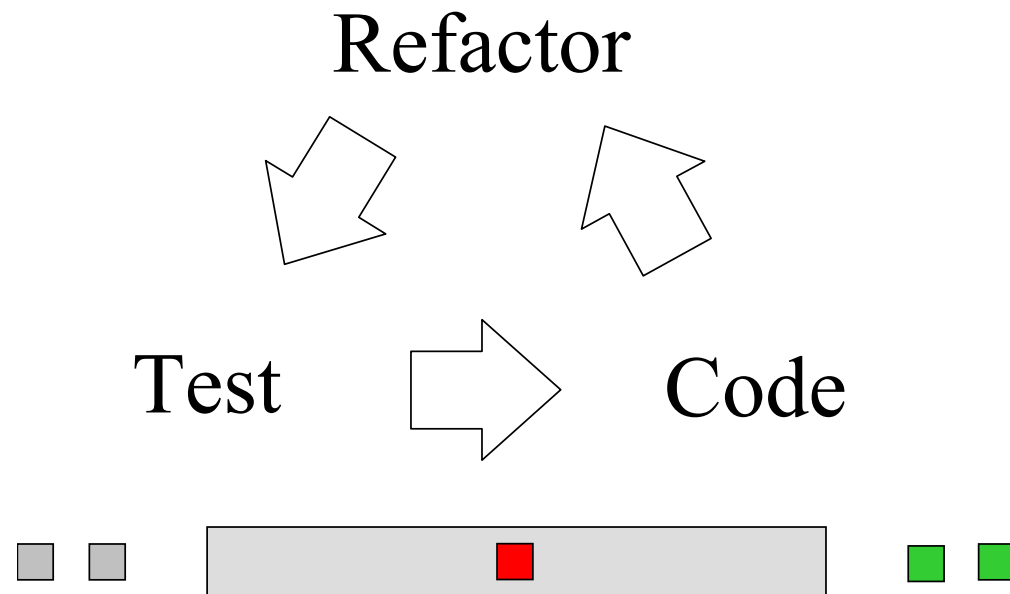
Keeping the code tested means:

- Rapid feedback,
 - Running a suite of automated unit tests
- Being confident,
 - When we need it most, stress situations
- Indicator of the projects health

Getting Started

- For new development
 - Why not start writing tests first ?
 - For an existing code base
 - Write test to lock existing behavior
- => Start the refactoring cycle

Writing The Tests First



- Red: Write a failing unit test
- Green: Write enough code to make it pass
- Refactor: Deal with design / code smells



Discussion

■ Questions ?

For More About Refactoring

Websites:

[<www.refactoring.com>](http://www.refactoring.com)

[<www.refactoring.be>](http://www.refactoring.be)

Contact us:

Sven.Gorts@refactoring.be

Hans.Keppens@refactoring.be

References

- Refactoring: *Improving The Design Of Existing Code*
(Martin Fowler)
- Refactoring Workbook
(William C. Wake)
- Refactoring To Patterns
(Joshua Kerievsky)