SOFTWARE TESTING

SOFTWARE REFACTORING

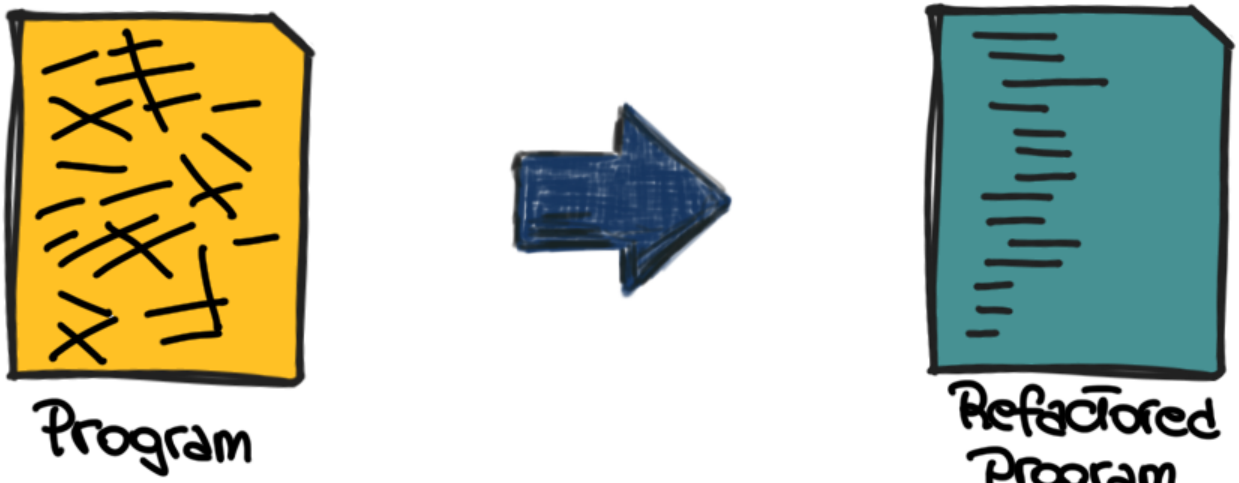
Mahdi Roozbahani

Slides are based on Alex Orso.

WHAT IS REFACTORING ?

WHAT IS REFACTORING ?

WHAT IS REFACTORING?



Goal: keep program readable, understandable, and maintainable

WHAT IS REFACTORING?

Goal: keep program readable, understandable, and maintainable

Key feature: behavior preserving.

BEHAVIOR PRESERVING

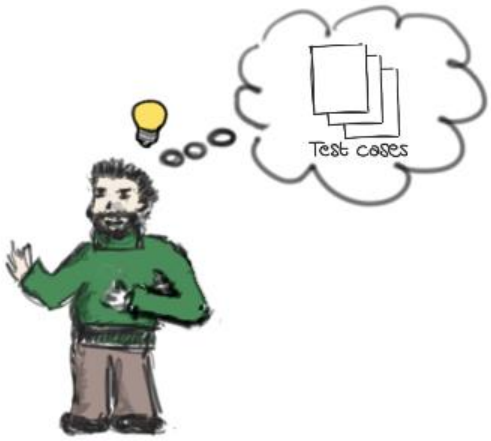
BEHAVIOR PRESERVING

How can we "guarantee" it?

BEHAVIOR PRESERVING

How can we "guarantee" it?

Test the code,
but beware:
no guarantees!



Why can't testing guarantee that a refactoring is behavior preserving?

Why can't testing guarantee that a refactoring is behavior preserving?

[ ] Because testing and refactoring are different activities

[ ] Because testing is inherently incomplete

[ ] Because testers are often inexperienced

Why can't testing guarantee that a refactoring is behavior preserving?

[ ] Because testing and refactoring are different activities

[ ] Because testing is inherently incomplete

[ ] Because testers are often inexperienced

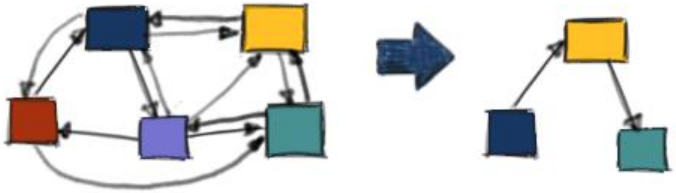
WHY REFACTORING?

WHY REFACTORING?

Requirements change

WHY REFACTORING?

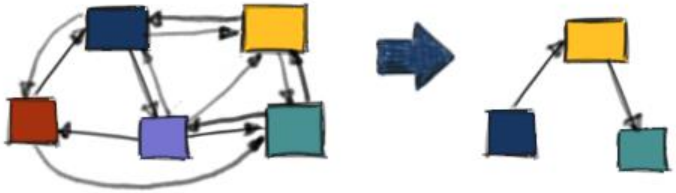
Requirements change



Design needs to be improved

WHY REFACTORING?

Requirements change



Design needs to be improved

Sloppiness / laziness

HOW YOU USED REFACTORING BEFORE?

HOW YOU USED REFACTORING BEFORE?

Even renaming a class is a refactoring!
(albeit a trivial one)

A LITTLE BIT OF HISTORY

A LITTLE BIT OF HISTORY

- Refactoring is something programmers have always done

A LITTLE BIT OF HISTORY

- Refactoring is something programmers have always done
- Especially important for object-oriented languages

A LITTLE BIT OF HISTORY

- Refactoring is something programmers have always done
- Especially important for object-oriented languages
- Opdyke's PhD Thesis (1990) discusses refactoring for SMALLTALK

A LITTLE BIT OF HISTORY

- Refactoring is something programmers have always done
- Especially important for object-oriented languages
- Opdyke's PhD Thesis (1990) discusses refactoring for SMALLTALK
- Increasingly popular due to agile development

FOWLER'S BOOK

Catalog of refectorings

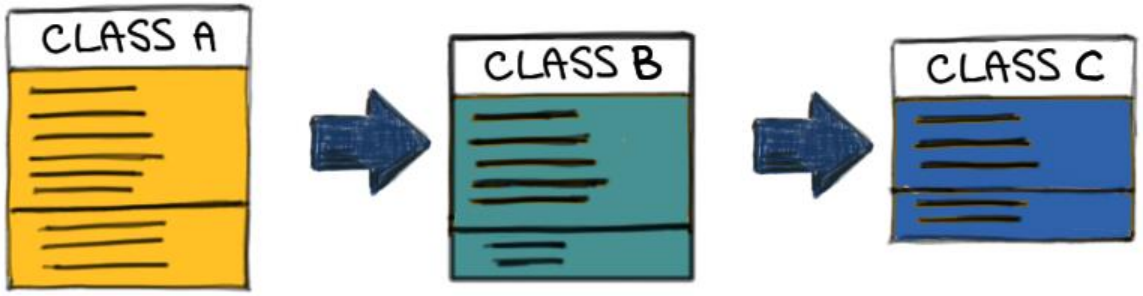
List of bad smells

Guidelines on when applying refactoring

Example of code before and after

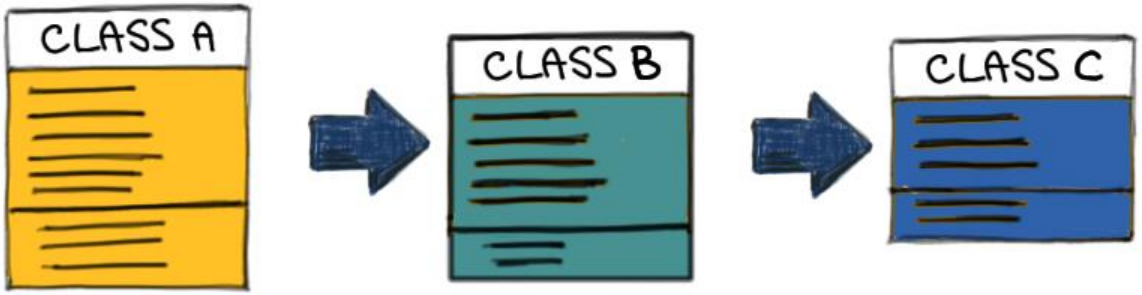
COMING UP NEXT

COMING UP NEXT



Some examples of refactoring

COMING UP NEXT



Some examples of refactoring

Some examples of "bad smells" in code

THERE ARE MANY REFACTORINGS

Add parameter
Change association
Reference to value
Value to reference
Collapse hierarchy
Consolidate conditionals
Procedures to objects
Decompose conditionals
Encapsulate collection

Encapsulate downcast
Encapsulate field
Extract method
Extact class
Inline class
Form template method
Hide delegate
Hide method
Inline temp
...

THERE ARE MANY REFACTORINGS

Add parameter
Change association
Reference to value
Value to reference
Collapse hierarchy
Consolidate conditionals
Procedures to objects
Decompose conditionals
Encapsulate collection

Encapsulate downcast
Encapsulate field
Extract method
Extact class
Inline class
Form template method
Hide delegate
Hide method
Inline temp
...

COLLAPSE HIERARCHY

A superclass and a subclass are too similar

⇒ merge them

COLLAPSE HIERARCHY

A superclass and a subclass are too similar

⇒ merge them

THERE ARE MANY REFACTORINGS

Add parameter
Change association
Reference to value
Value to reference
Collapse hierarchy
Consolidate conditionals
Procedures to objects
Decompose conditionals
Encapsulate collection

Encapsulate downcast
Encapsulate field
Extract method
Extact class
Inline class
Form template method
Hide delegate
Hide method
Inline temp
...

CONSOLIDATE CONDITIONAL EXPRESSION

Set of conditionals with the same results
=> combine and extract them

CONSOLIDATE CONDITIONAL EXPRESSION

Set of conditionals with the same results
=> combine and extract them

```
double disabilityAmount(){
    if (seniority<2)
        return 0;
    if (monthsDisabled>12)
        return 0;
    if (isPartTime)
        return 0;
    // compute disability amount
}
```

CONSOLIDATE CONDITIONAL EXPRESSION

Set of conditionals with the same results
=> combine and extract them

```
double disabilityAmount(){
    if (seniority<2)
        return 0;
    if (monthsDisabled>12)
        return 0;
    if (isPartTime)
        return 0;
    // compute disability amount
}
```

```
double disabilityAmount(){
    if (notEligibleForDisability())
        return 0;
    // compute disability amount
}
```

THERE ARE MANY REFACTORINGS

Add parameter
Change association
Reference to value
Value to reference
Collapse hierarchy
Consolidate conditionals
Procedures to objects
Decompose conditionals
Encapsulate collection

Encapsulate downcast
Encapsulate field
Extract method
Extact class
Inline class
Form template method
Hide delegate
Hide method
Inline temp
...

DECOMPOSE CONDITIONALS

A conditional statement is particularly complex
⇒ extract methods from conditions
modify THEN and ELSE part of the conditional

DECOMPOSE CONDITIONALS

A conditional statement is particularly complex
⇒ extract methods from conditions
modify THEN and ELSE part of the conditional

```
if (date.before (SUMMER_START) || date.after (SUMMER_END))
    charge = quantity * winterRate + winterServiceCharge;
else
    charge = quantity * summerRate;
```

DECOMPOSE CONDITIONALS

A conditional statement is particularly complex
⇒ extract methods from conditions
modify THEN and ELSE part of the conditional

```
if (date.before (SUMMER_START) || date.after (SUMMER_END))
    charge = quantity * winterRate + winterServiceCharge;
else
    charge = quantity * summerRate;
```

DECOMPOSE CONDITIONALS

A conditional statement is particularly complex
⇒ extract methods from conditions
modify THEN and ELSE part of the conditional

```
if (date.before (SUMMER_START)|| date.after (SUMMER_END))
    charge = quantity * winterRate + winterSeruiceCharge;
else
    charge = quantity * summerRate;
```

```
if (not Summer(date))
    charge = winterCharge(quantity)
else
    charge = summerCharge(quantity)
```

THERE ARE MANY REFACTORINGS

Add parameter
Change association
Reference to value
Value to reference
Collapse hierarchy
Consolidate conditionals
Procedures to objects
Decompose conditionals
Encapsulate collection

Encapsulate downcast
Encapsulate field
Extract method
Extact class
Inline class
Form template method
Hide delegate
Hide method
Inline temp
...

EXTRACT CLASS

A class is doing the work of two classes

⇒ create a new class and move there relevant fields/methods

EXTRACT CLASS

A class is doing the work of two classes

⇒ create a new class and move there relevant fields/methods

THERE ARE MANY REFACTORINGS

Add parameter
Change association
Reference to value
Value to reference
Collapse hierarchy
Consolidate conditionals
Procedures to objects
Decompose conditionals
Encapsulate collection

Encapsulate downcast
Encapsulate field
Extract method
Extact class
Inline class
Form template method
Hide delegate
Hide method
Inline temp
...

INLINE CLASS

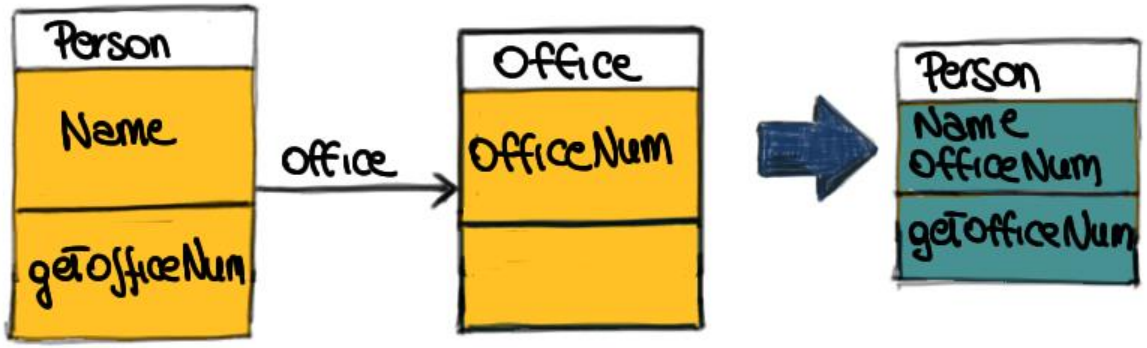
A class is not doing much

⇒ move its features into another class and delete this one

INLINE CLASS

A class is not doing much

⇒ move its features into another class and delete this one



THERE ARE MANY REFACTORINGS

Add parameter
Change association
Reference to value
Value to reference
Collapse hierarchy
Consolidate conditionals
Procedures to objects
Decompose conditionals
Encapsulate collection

Encapsulate downcast
Encapsulate field
Extract method
Extact class
Inline class
Form template method
Hide delegate
Hide method
Inline temp
...

EXTRACT METHOD

Cohesive code fragment in a large method

=> create a method using that code fragment

EXTRACT METHOD

Cohesive code fragment in a large method

=> create a method using that code fragment

```
void printOwing(){
...
   System.out.println("name:"+ name+
                      "address:"+address);
    ...
   System.out.println("amount owned "+
                      amount)
...
}
```

EXTRACT METHOD

Cohesive code fragment in a large method

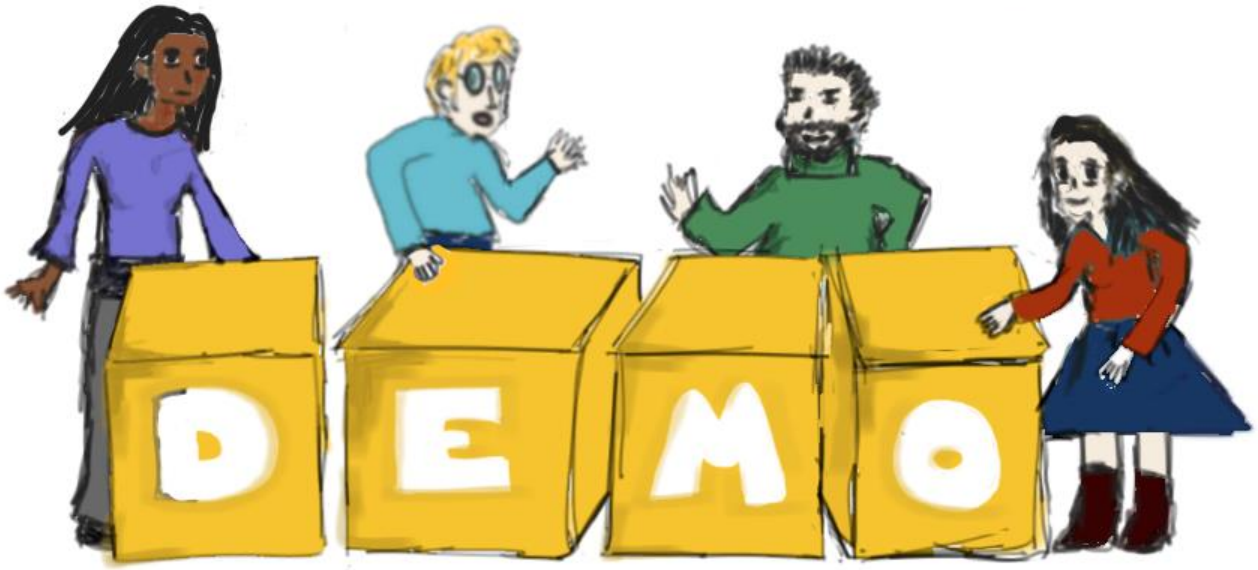
=> create a method using that code fragment

```
void printOwing(){
...
   System.out.println("name:"+ name+
                      "address:"+address);
    ...
   System.out.println("amount owned "+
                      amount)
...
}
```

```
void printOwing(){
...
   printDetails();
...
}
void printDetails(){
   System.out.println("name"+...);
   ...
   System.out.println("amount...);
}
```

HOW CAN WE ACTUALLY PERFORM REFACTORING ?

HOW CAN WE ACTUALLY PERFORM REFACTORING ?



When is it appropriate to apply refactoring "extract method"?

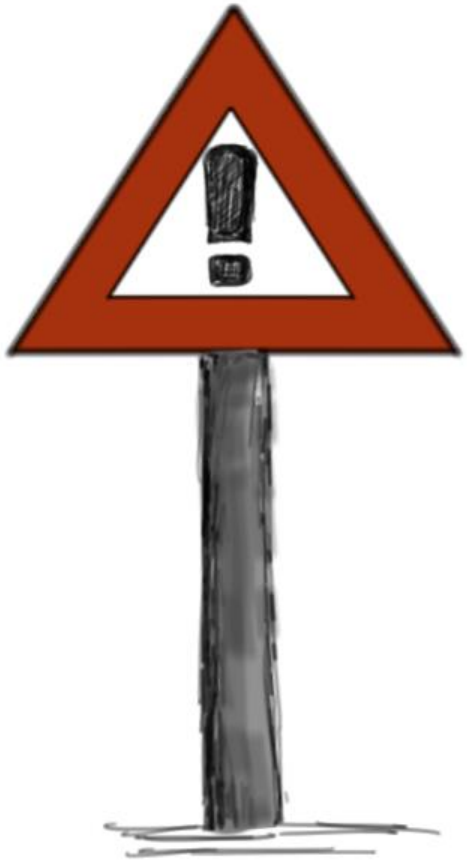
[ ] When there is duplicated code in two or more methods

[ ] When a class is too large

[ ] When the names of two classes are too similar

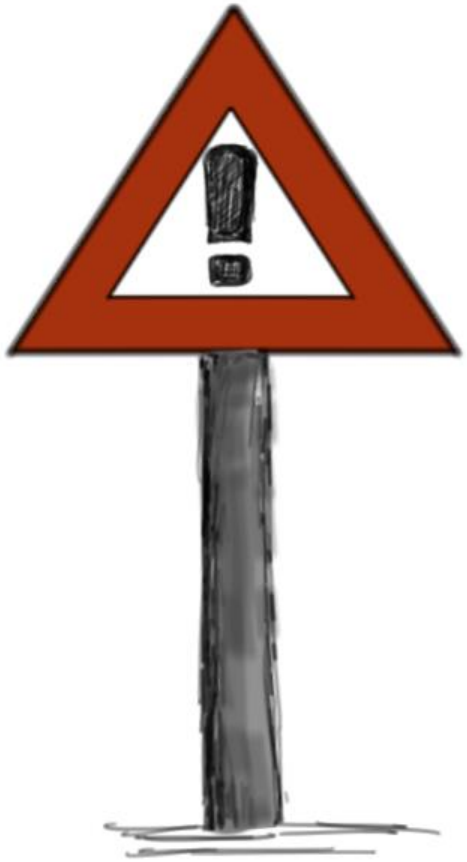
[ ] When a method is highly coupled with a class other than the one where it is defined

REFACTORING RISKS



Powerful tool, but ...

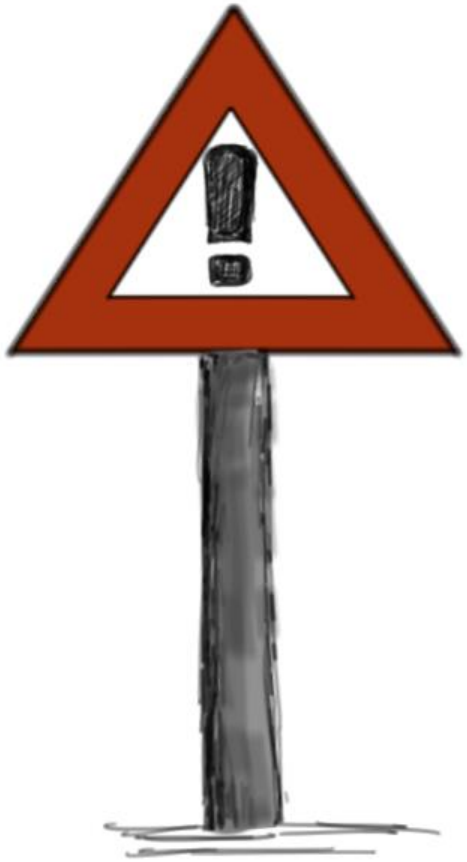
REFACTORING RISKS



Powerful tool, but ...

- May introduce subtle faults

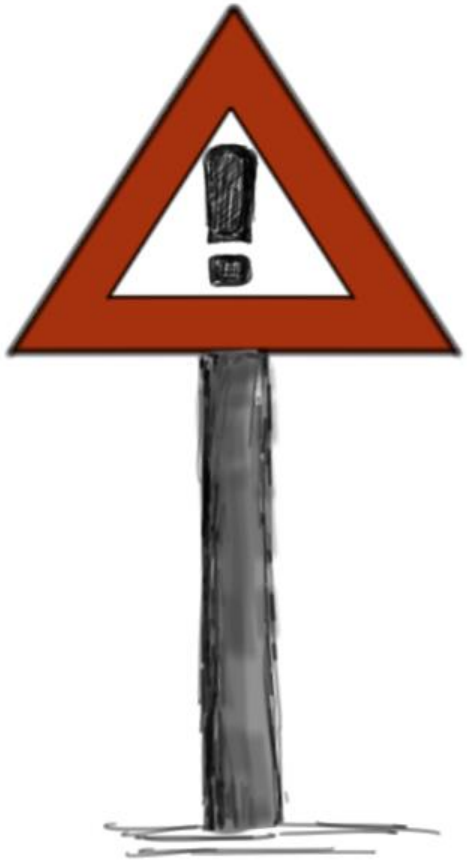
REFACTORING RISKS



Powerful tool, but ...

- May introduce subtle faults
- Should not be abused

REFACTORING RISKS



Powerful tool, but ...

- May introduce subtle faults
- Should not be abused
- Should be used carefully on systems in production

COST OF REFACTORING

COST OF REFACTORING

Manual work

COST OF REFACTORING

Manual work

Test development and maintenance

COST OF REFACTORING

Manual work

Test development and maintenance

Documentation maintenance

WHEN NOT TO REFACTOR?

WHEN NOT TO REFACTOR?

When code is broken

WHEN NOT TO REFACTOR?

When code is broken

When a deadline is close

WHEN NOT TO REFACTOR?

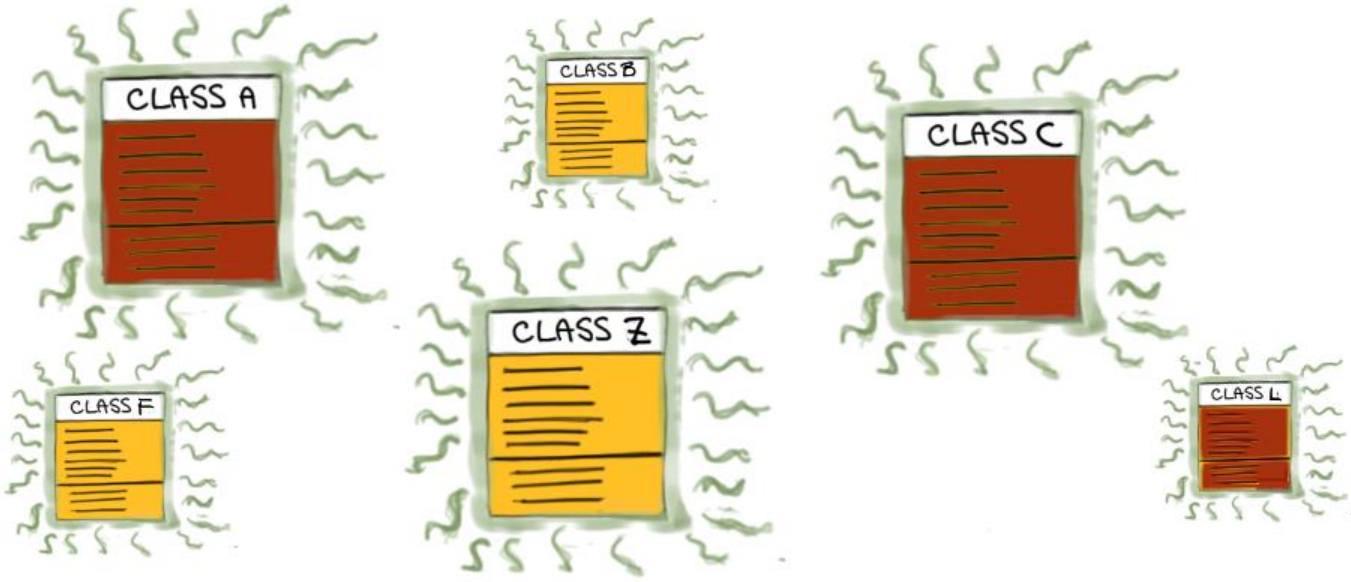
When code is broken

When a deadline is close

When there is no reason to!


BAD SMELLS

BAD SMELLS



A CATALOGUE OF BAD SMELLS

A CATALOGUE OF BAD SMELLS

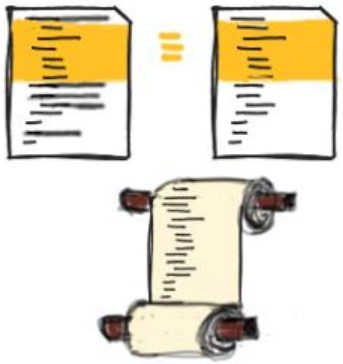
- Duplicated code
- Long method
- Large class
- Long parameter list
- Divergent change
- Shotgun surgery
- Feature envy
- Data clumps
- Primitive obsession
- Switch statements
- Parallel inheritance hierarchy
- Lazy class
- Speculative generality
- Temporary field
- Message chains
- Middle man
- Inappropriate intimacy
- Incomplete library class
- Data class
- Refused bequest

A FEW EXAMPLES

A FEW EXAMPLES

Duplicated code

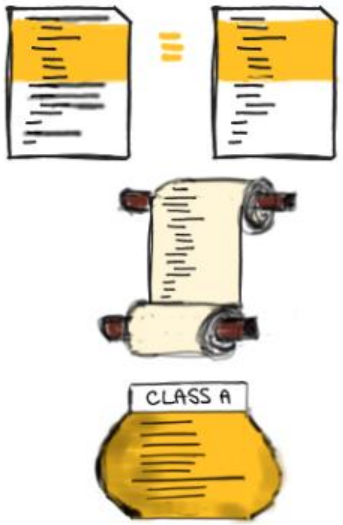
A FEW EXAMPLES



Duplicated code ➡ extract method

Long method

A FEW EXAMPLES



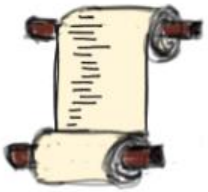
Duplicated code ➡ extract method

Long method ➡ extract method, decompose conditionals, ...

Large class

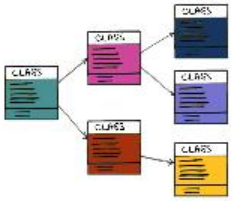
A FEW EXAMPLES

Duplicated code ➡ extract method



Long method ➡ extract method, decompose conditionals, ...

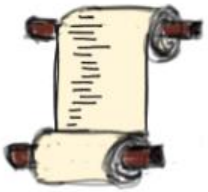
Large class ➡ extract class (or subclass)



Shotgun surgery ➡ move method / field, inline class, ...

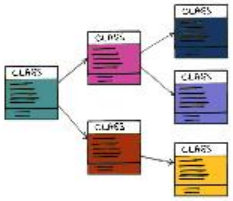
A FEW EXAMPLES

Duplicated code ➡ extract method



Long method ➡ extract method, decompose conditionals, ...

Large class ➡ extract class (or subclass)



Shotgun surgery ➡ move method / field, inline class, ...

Feature envy

Which of the following can be considered to be "bad smells" in the context of refactoring?

[ ] The program takes too long to execute

[ ] Method m() in class C is very long

[ ] Classes Cat and Dog are subclasses of class Animal

[ ] Every time we modify method m1(), we also need to modify method m2()