“Nie jest sztuką napisanie kodu zrozumiałego dla komputera. Dobrzy programiści tworzą kod zrozumiały dla ludzi.”

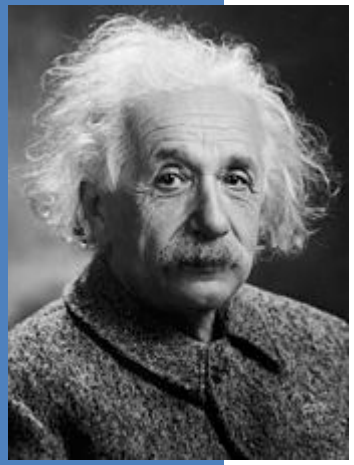
# REFAKTORYZACJA

*Przemysław Grzesiowski*
*2 marca 2018*

git clone https://github.com/infoshareacademy/jjdd3-materialy-refaktoring-java.git

“ *Everything should be made as simple as possible, but no simpler.*

*Einstein*

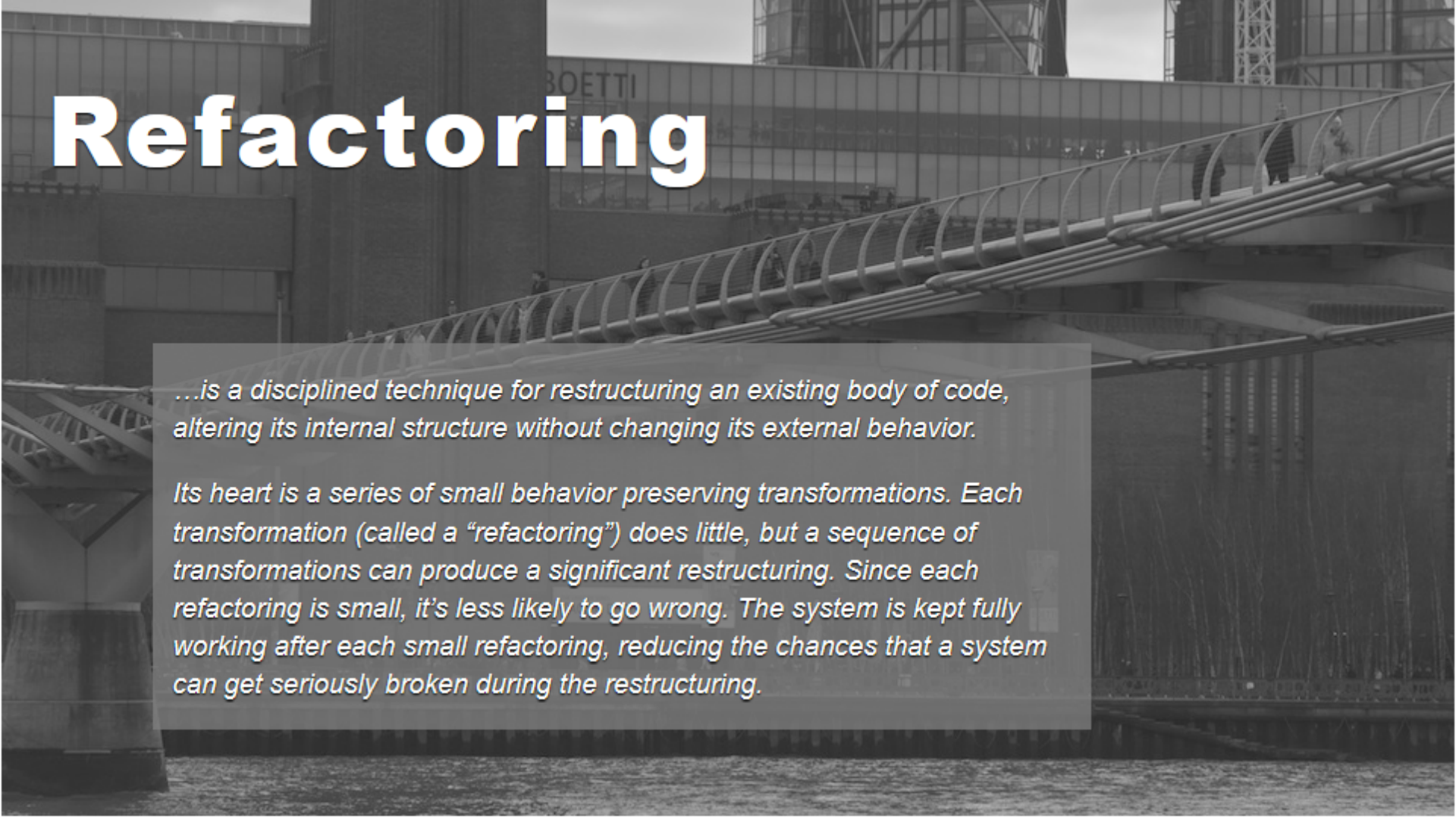


# DEFINICJA

Refaktoryzacja – co to takiego?

> ***Refactoring [noun]**: a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior*
>
> Martin Fowler

> ***Refactoring [verb]**: to restructure software by applying a series of refactorings without changing its observable behavior*
>
> Martin Fowler

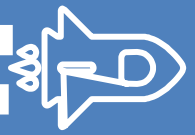


# Refactoring

*…is a disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior.*

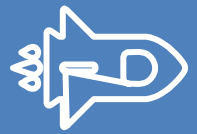
*Its heart is a series of small behavior preserving transformations. Each transformation (called a "refactoring") does little, but a sequence of transformations can produce a significant restructuring. Since each refactoring is small, it's less likely to go wrong. The system is kept fully working after each small refactoring, reducing the chances that a system can get seriously broken during the restructuring.*

# SAFE DELETE



Zacznijmy od usuwania - DeleteThisClassPlease.java

# EXTRACT



1. extract variable
    - 1.1. extract variable — method level
    - 1.2. extract constant — class level
    - 1.3. extract field — class level
2. extract method
3. extract parameter

# Extract Variable

You have a complicated expression.

***Put the result of the expression, or parts of the expression, in a temporary variable with a name that explains the purpose.***

```
if ( (platform.toUpperCase().indexOf("MAC") > -1) &&
     (browser.toUpperCase().indexOf("IE") > -1) &&
      wasInitialized() && resize > 0 )
{
  // do something
}
```

```
final boolean isMacOs     = platform.toUpperCase().indexOf("MAC") > -1;
final boolean isIEBrowser = browser.toUpperCase().indexOf("IE")  > -1;
final boolean wasResized  = resize > 0;

if (isMacOs && isIEBrowser && wasInitialized() && wasResized)
{
  // do something
}
```

**Inverse of** *Inline Temp*

**Naming:** In the books, this refactoring is called "Introduce Explaining Variable", but most tools and people now use the (better) name "extract variable"

# Extract Variable

You have a complicated expression.

***Put the result of the expression, or parts of the expression, in a temporary variable with a name that explains the purpose.***

```
if ( (platform.toUpperCase().indexOf("MAC") > -1) &&
     (browser.toUpperCase().indexOf("IE") > -1) &&
      wasInitialized() && resize > 0 )
{
  // do something
}
```

```
final boolean isMacOs     = platform.toUpperCase().indexOf("MAC") > -1;
final boolean isIEBrowser = browser.toUpperCase().indexOf("IE")  > -1;
final boolean wasResized  = resize > 0;

if (isMacOs && isIEBrowser && wasInitialized() && wasResized)
{
  // do something
}
```

**Inverse of** *Inline Temp*

**Naming:** In the books, this refactoring is called "Introduce Explaining Variable", but most tools and people now use the (better) name "extract variable"

## Benefits

- More readable code! Try to give the extracted variables good names that announce the variable's purpose loud and clear. More readability, fewer long-winded comments. Go for names like `customerTaxValue`, `cityUnemploymentRate`, `clientSalutationString`, etc.

# Extract Variable

You have a complicated expression.

***Put the result of the expression, or parts of the expression, in a temporary variable with a name that explains the purpose.***

```
if ( (platform.toUpperCase().indexOf("MAC") > -1) &&
     (browser.toUpperCase().indexOf("IE") > -1) &&
      wasInitialized() && resize > 0 )
{
  // do something
}
```

```
final boolean isMacOs     = platform.toUpperCase().indexOf("MAC") > -1;
final boolean isIEBrowser = browser.toUpperCase().indexOf("IE")  > -1;
final boolean wasResized  = resize > 0;

if (isMacOs && isIEBrowser && wasInitialized() && wasResized)
{
  // do something
}
```

**Inverse of** *Inline Temp*

**Naming:** In the books, this refactoring is called "Introduce Explaining Variable", but most tools and people now use the (better) name "extract variable"

## Benefits

- More readable code! Try to give the extracted variables good names that announce the variable's purpose loud and clear. More readability, fewer long-winded comments. Go for names like `customerTaxValue`, `cityUnemploymentRate`, `clientSalutationString`, etc.

## Drawbacks

- More variables are present in your code. But this is counterbalanced by the ease of reading your code.

# Extract Variable

You have a complicated expression.

***Put the result of the expression, or parts of the expression, in a temporary variable with a name that explains the purpose.***

```
if ( (platform.toUpperCase().indexOf("MAC") > -1) &&
     (browser.toUpperCase().indexOf("IE") > -1) &&
      wasInitialized() && resize > 0 )
{
  // do something
}
```

```
final boolean isMacOs     = platform.toUpperCase().indexOf("MAC") > -1;
final boolean isIEBrowser = browser.toUpperCase().indexOf("IE")  > -1;
final boolean wasResized  = resize > 0;

if (isMacOs && isIEBrowser && wasInitialized() && wasResized)
{
  // do something
}
```

**Inverse of** *Inline Temp*

**Naming:** In the books, this refactoring is called "Introduce Explaining Variable", but most tools and people now use the (better) name "extract variable"

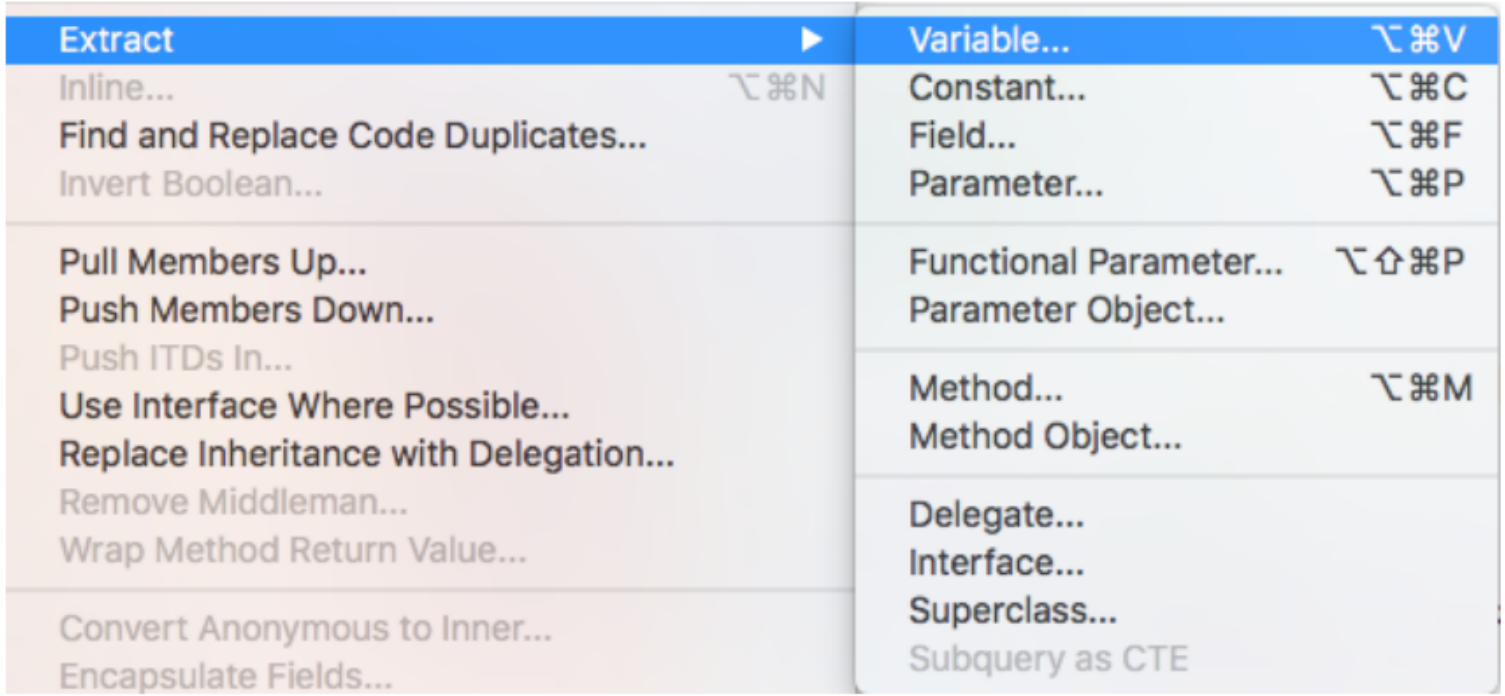
## Benefits

- More readable code! Try to give the extracted variables good names that announce the variable's purpose loud and clear. More readability, fewer long-winded comments. Go for names like `customerTaxValue`, `cityUnemploymentRate`, `clientSalutationString`, etc.

## Drawbacks

- More variables are present in your code. But this is counterbalanced by the ease of reading your code.

NAZYWAJ RZECZY PO IMIENIU !


Extract
Inline... ⌥⌘N
Find and Replace Code Duplicates...
Invert Boolean...
Pull Members Up...
Push Members Down...
Push ITDs In...
Use Interface Where Possible...
Replace Inheritance with Delegation...
Remove Middleman...
Wrap Method Return Value...
Convert Anonymous to Inner...
Encapsulate Fields...
Variable... ⌥⌘V
Constant... ⌥⌘C
Field... ⌥⌘F
Parameter... ⌥⌘P
Functional Parameter... ⌥⇧⌘P
Parameter Object...
Method... ⌥⌘M
Method Object...
Delegate...
Interface...
Superclass...
Subquery as CTE

# Ćwiczenie (3 in 1)

- Extract variable
- Extract constant
- Extract field

Dokonaj refaktoringu w klasie Okrag.java (ćwiczenie oznaczone jako “todo A”).
*Za każdym razem odpalaj testy, aby sprawdzić czy kod nadal działa tak jak należy.*

# Extract Method

You have a code fragment that can be grouped together.

***Turn the fragment into a method whose name explains the purpose of the method.***

```
void printOwing() {
  printBanner();

  //print details
  System.out.println ("name:  " + _name);
  System.out.println ("amount " + getOutstanding());
}
```

```
void printOwing() {
  printBanner();
  printDetails(getOutstanding());
}

void printDetails (double outstanding) {
  System.out.println ("name:  " + _name);
  System.out.println ("amount " + outstanding);
}
```

**Inverse of** *Inline Method*

# Extract Method

You have a code fragment that can be grouped together.

***Turn the fragment into a method whose name explains the purpose of the method.***

```
void printOwing() {
  printBanner();

  //print details
  System.out.println ("name:  " + _name);
  System.out.println ("amount " + getOutstanding());
}
```

```
void printOwing() {
  printBanner();
  printDetails(getOutstanding());
}

void printDetails (double outstanding) {
  System.out.println ("name:  " + _name);
  System.out.println ("amount " + outstanding);
}
```

**Inverse of** *Inline Method*

## Benefits

- More readable code! Be sure to give the new method a name that describes the method's purpose: `createOrder()`, `renderCustomerInfo()`, etc.

# Extract Method

You have a code fragment that can be grouped together.

***Turn the fragment into a method whose name explains the purpose of the method.***

```
void printOwing() {
  printBanner();

  //print details
  System.out.println ("name:  " + _name);
  System.out.println ("amount " + getOutstanding());
}
```

```
void printOwing() {
  printBanner();
  printDetails(getOutstanding());
}

void printDetails (double outstanding) {
  System.out.println ("name:  " + _name);
  System.out.println ("amount " + outstanding);
}
```

**Inverse of** *Inline Method*

## Benefits

- More readable code! Be sure to give the new method a name that describes the method's purpose: `createOrder()`, `renderCustomerInfo()`, etc.
- Less code duplication. Often the code that is found in a method can be reused in other places in your program. So you can replace duplicates with calls to your new method.

# Extract Method

You have a code fragment that can be grouped together.

***Turn the fragment into a method whose name explains the purpose of the method.***

```
void printOwing() {
  printBanner();

  //print details
  System.out.println ("name:  " + _name);
  System.out.println ("amount " + getOutstanding());
}
```

```
void printOwing() {
  printBanner();
  printDetails(getOutstanding());
}

void printDetails (double outstanding) {
  System.out.println ("name:  " + _name);
  System.out.println ("amount " + outstanding);
}
```

**Inverse of** *Inline Method*

## Benefits

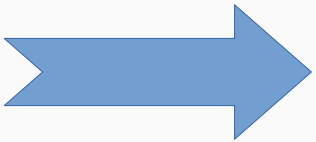
- More readable code! Be sure to give the new method a name that describes the method's purpose: `createOrder()`, `renderCustomerInfo()`, etc.
- Less code duplication. Often the code that is found in a method can be reused in other places in your program. So you can replace duplicates with calls to your new method.
- Isolates independent parts of code, meaning that errors are less likely (such as if the wrong variable is modified).

# Extract Method

infoShare academy

**Before**

```
public void method() {
   int a=1;
   int b=2;
   int c=a+b;
   int d=a+c;
```



**After**

```
public void method() {
    int a=1;
    int b=2;
    int c=add(a,b);
    int d=add(a,c);
}
    ...
private int add(int a, int b) {
    return a+b;
}
```

# Ćwiczenie
# Extract method

- Dokonaj refaktoringu:
  - TenisGame1.java.
  - ExtractMethod.java.

PAMIĘTAJ! *Za każdym razem odpalaj testy jednostkowe klasy którą refaktoryzujesz, aby sprawdzić czy kod nadal działa tak jak należy.*

# Extract parameter

**Before**

```
public class HelloWorldPrinter {
    public static void print() {
        System.out.println(generateText());
    }
    private static String generateText() {
        return "Hello, World!".toUpperCase();
    }
}
```

**After**

```
public class HelloWorldPrinter {
    public static void print() {
        System.out.println(generateText("Hello, World!"));
    }
    private static String generateText(String text) {
        return text.toUpperCase();
    }
}
```

# Ćwiczenie Extract Parameter

- Dokonaj refaktoringu w klasie ExtractParameter.java.

  PAMIĘTAJ! *Za każdym razem odpalaj testy jednostkowe klasy którą refaktoryzujesz, aby sprawdzić czy kod nadal działa tak jak należy.*

## Extract parameter

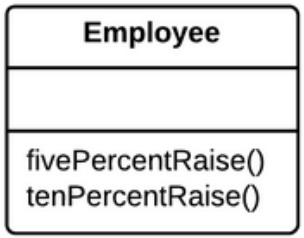
# Parameterize Method

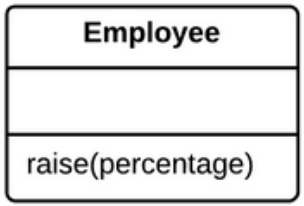
### Problem

Multiple methods perform similar actions that are different only in their internal values, numbers or operations.

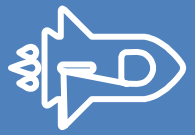
### Solution

Combine these methods by using a parameter that will pass the necessary special value.

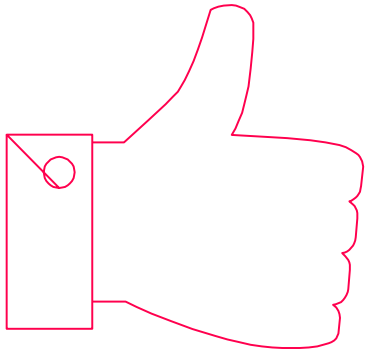


# EXTRACT



1. extract variable
    - 1.1. extract variable — method level
    - 1.2. extract constant — class level
    - 1.3. extract field — class level
2. extract method
3. extract parameter



# Trochę teorii się przyda...

Who loves refactoring?

Who loves refactoring?

developers

Who hates refactoring?

Who hates refactoring?

business

# Refactoring techniques

- **Scratch refactoring**
  Just change some part of the code and check what elements are involved in it and know system much better.

# Refactoring techniques

- **Scratch refactoring**
  Just change some part of the code and check what elements are involved in it and know system much better.

- **Large Scale Refactoring**
  A huge change in system, not really called refactoring, rather rewriting part of the system.

# Refactoring rules

- Relay on IDE where you can!
- Use manual refactoring when you have knowledge that IDE doesn't

# Refactoring rules

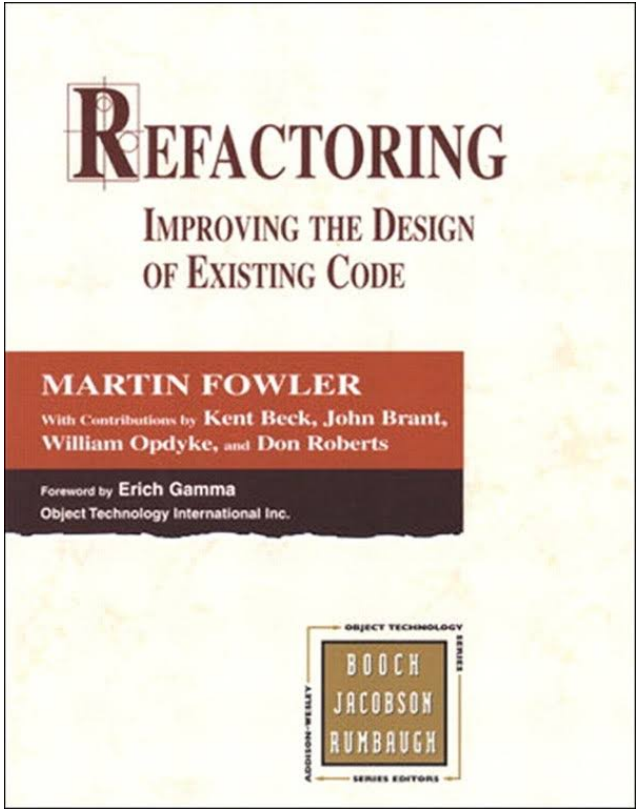
- Relay on IDE where you can!
- Use manual refactoring when you have knowledge that IDE doesn't
- Relay on unit tests to be sure that program works after changes

# Refactoring rules

- Relay on IDE where you can!
- Use manual refactoring when you have knowledge that IDE doesn't
- Relay on unit tests to be sure that program works after changes
- Make it a daily routine

What is **not refactoring**?

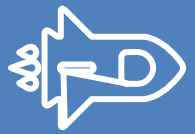
- Fixing bugs
- Optimalisation
- Tightening up error handling and adding defensive code
- Making the code more testable




Rok wydania - 1999

# Wracamy do przepastnego katalogu przekształceń ...

# INLINE



1. inline variable (opposite to extract variable)
2. inline method (opposite to extract method)
3. inline class

# Inline Method

A method's body is just as clear as its name.

***Put the method's body into the body of its callers and remove the method.***

```
int getRating() {
  return (moreThanFiveLateDeliveries()) ? 2 : 1;
}
boolean moreThanFiveLateDeliveries() {
  return _numberOfLateDeliveries > 5;
}
```

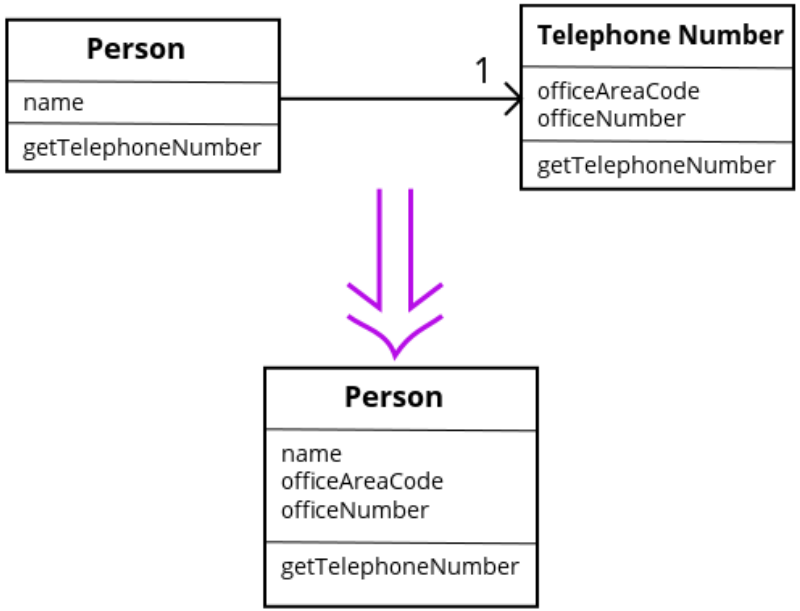
```
int getRating() {
  return (_numberOfLateDeliveries > 5) ? 2 : 1;
}
```

**inverse of** *Extract Method*

# Inline Class

A class isn't doing very much.

***Move all its features into another class and delete it.***



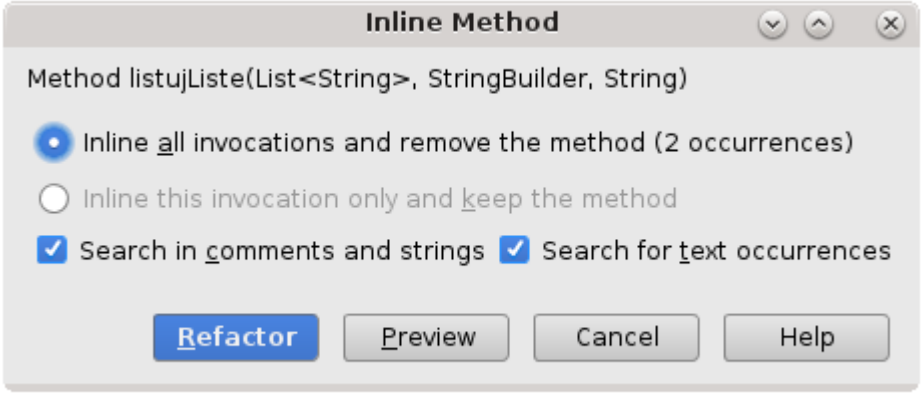

**Inverse of** *Extract Class*, *Extract Interface*

# Ćwiczenie Inline

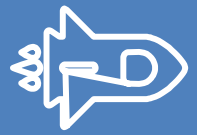
- Dokonaj refaktoringu:
  - TenisGame2.java
  - ExtractMethod2.java

PAMIĘTAJ! *Za każdym razem odpalaj testy jednostkowe klasy którą refaktoryzujesz, aby sprawdzić czy kod nadal działa tak jak należy.*

# Ćwiczenie Inline



# RENAME



rename variable
rename method
rename class
move package

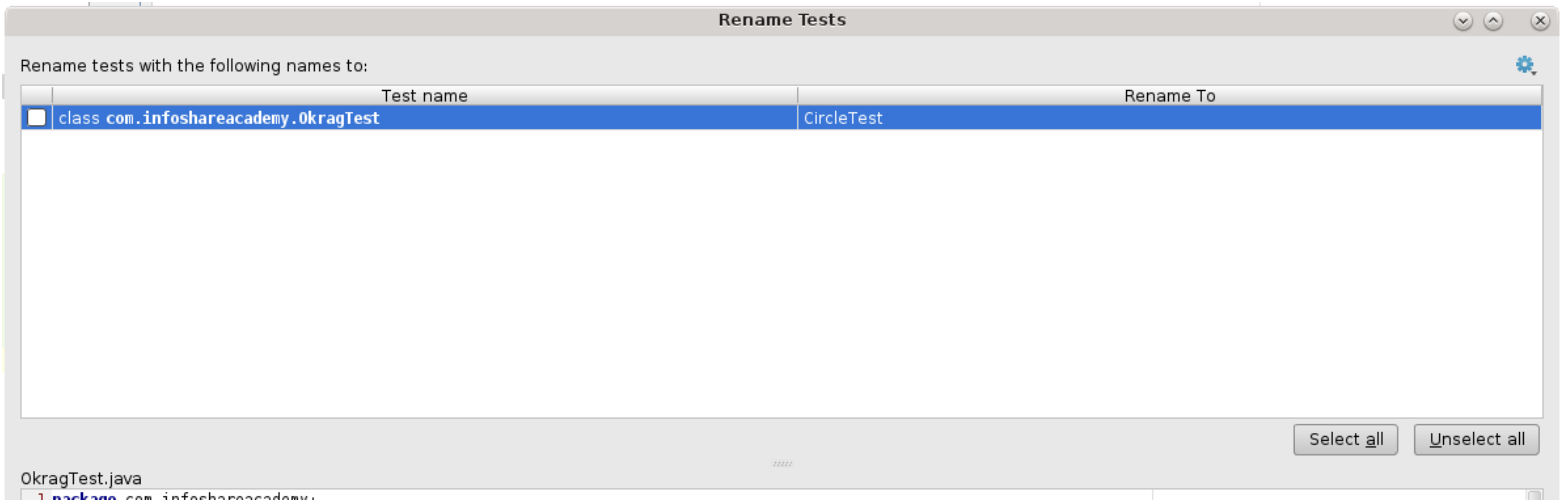
# Ćwiczenie rename

- rename variable
- rename method
- rename class
- move package

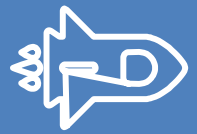
Dokonaj refaktoringu w klasie Okrag.java (todo B).
Za każdym razem odpalaj testy, aby sprawdzić czy kod nadal działa tak jak należy.

*Zwróć uwagę co się dzieje z getterami/setterami.*

# Rename Class (Okrag → Circle)



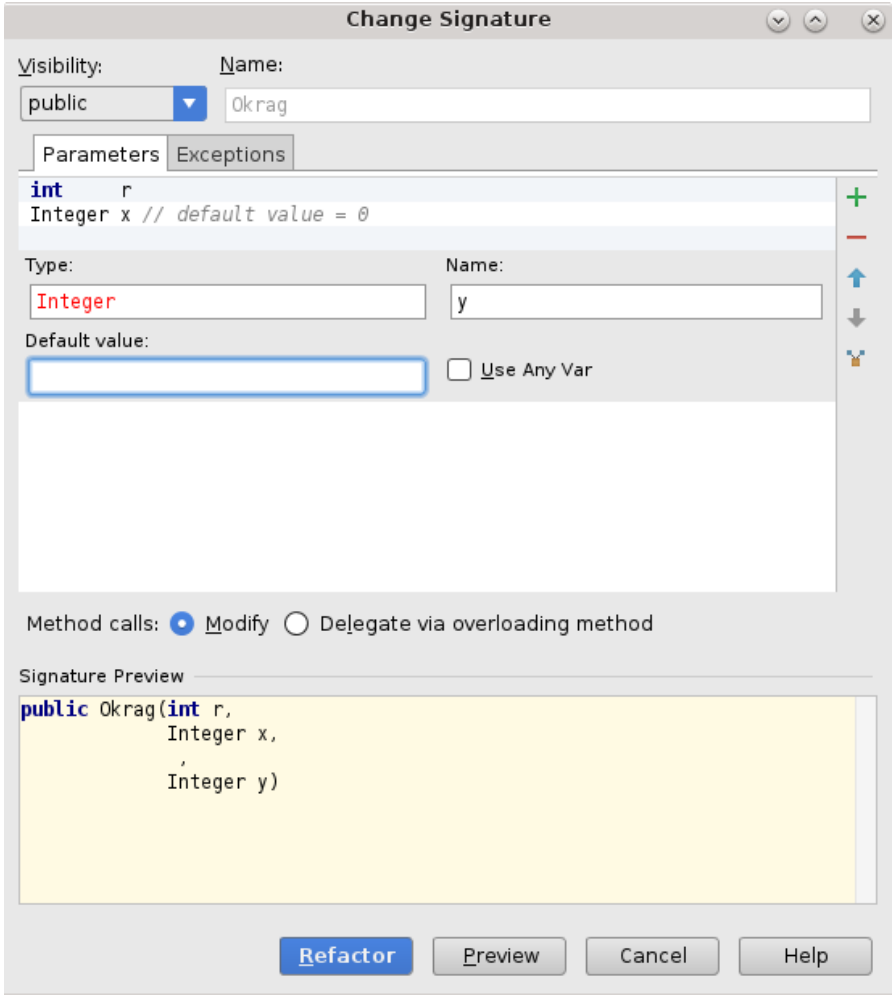
# ZMIANY



Change method signature
Make Method Static
Type Migration

# Change Signature

- change the method name.
- change the method return type.
- add new parameters (assign default values).
- remove existing parameters.
- reorder parameters.
- change parameter names and types.


Change Signature
Visibility:
Name:
public
Okrag
Parameters
Exceptions
int r
Integer x // default value = 0
Type:
Name:
Integer
y
Default value:
Use Any Var
Method calls:
Modify
Delegate via overloading method
Signature Preview
public Okrag(int r,
Integer x,
,
Integer y)
Refactor
Preview
Cancel
Help

# Ćwiczymy dalej – change method signature

Dokonaj refaktoringu w klasie Okrag.java (todo C).

*Za każdym razem odpalaj testy, aby sprawdzić czy kod nadal działa tak jak należy.*

# Type migration

**Before**

```
public class ResultContainer {
    private ArrayList<String> myResult;
    public String[] getResult() {
        return myResult.toArray(new String[myResult.size()]);
    }
}
```

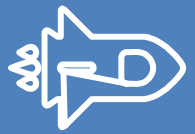
**After**

```
public class ResultContainer {
    private String[] myResult;
    public String[] getResult() {
        return myResult;
    }
}
```

# type migration

Wykonaj ćwiczenie opisane w TypeMigrationApp.java.

# DZIEDZICZENIE

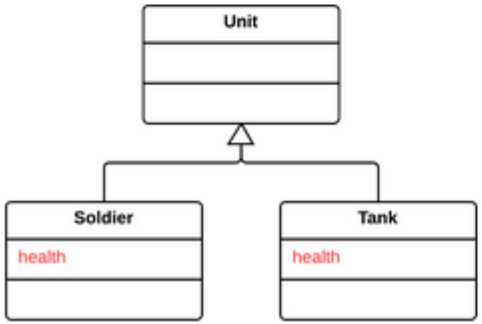


1. Pull members up
    - Pull method up
    - Pull field up
2. Push members down
    - Push method down
    - Push field down
3. Provide superclass (extract superclass)

# Pull Up Field

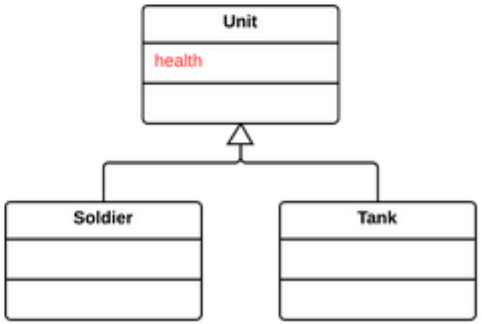
## Problem

Two classes have the same field.




## Solution

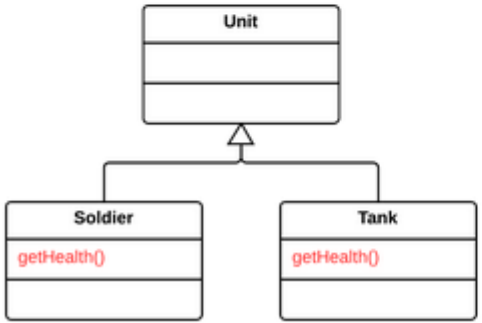
Remove the field from subclasses and move it to the superclass.



# Pull Up Method

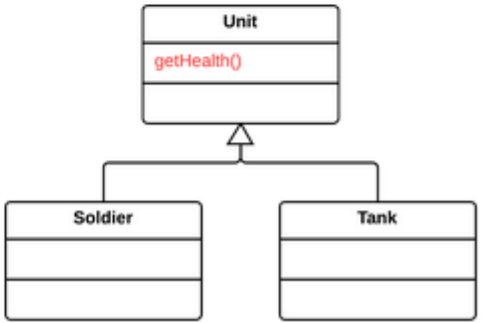
## Problem

Your subclasses have methods that perform similar work.




## Solution

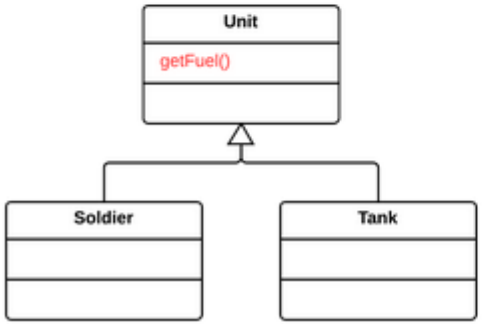
Make the methods identical and then move them to the relevant superclass.



# Push Down Method

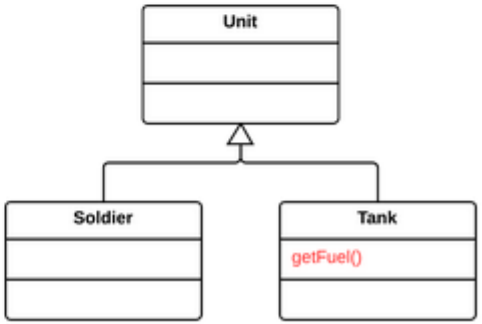
## Problem

Is behavior implemented in a superclass used by only one (or a few) subclasses?




## Solution

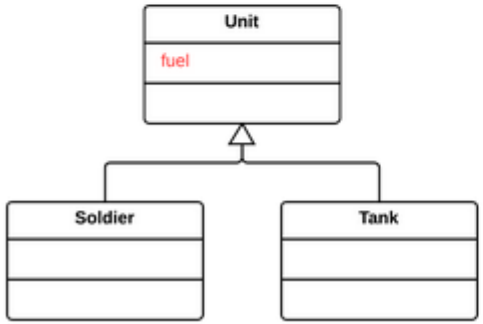
Move this behavior to the subclasses.



# Push Down Field

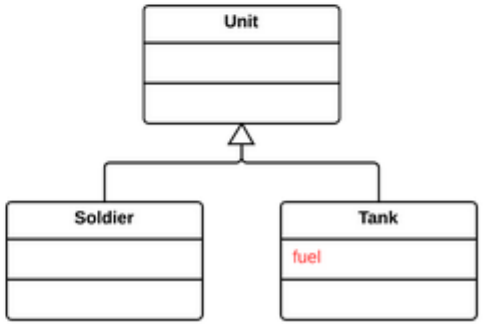
## Problem

Is a field used only in a few subclasses?




## Solution

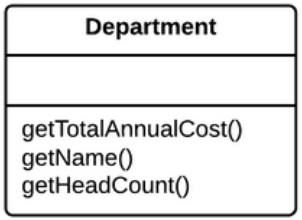
Move the field to these subclasses.

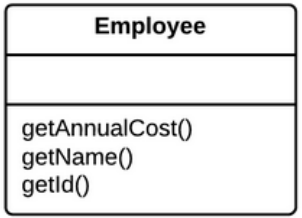


# Extract Superclass

## Problem

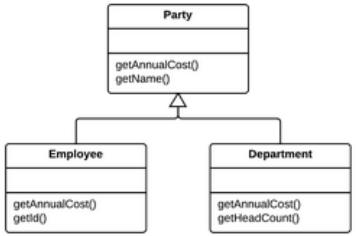
You have two classes with common fields and methods.




## Solution

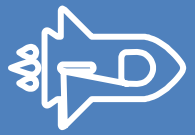
Create a shared superclass for them and move all the identical fields and methods to it.



# Pull up / push down

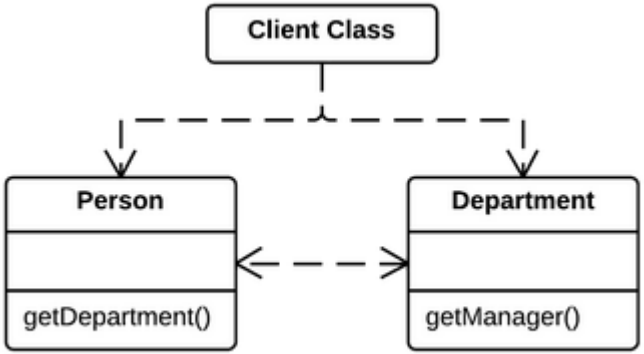
Wykonaj ćwiczenie opisane w klasie UpDownApp.java.

# DELEGATE



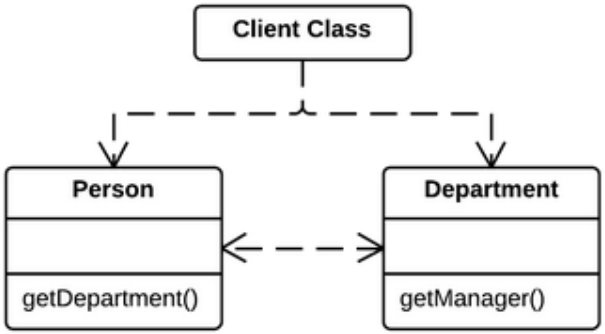
Hide Delegate
Remove Middleman

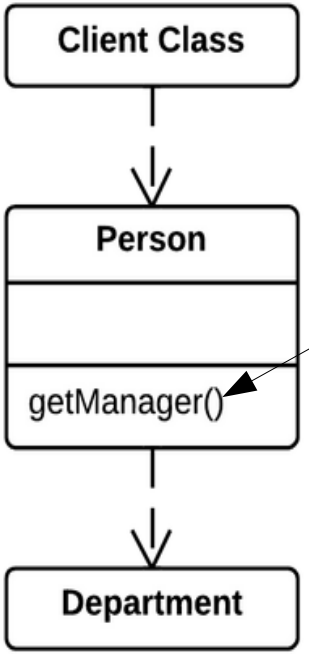
# Hide Delegate




String managerName = person.getDepartment().getManager();

# Hide Delegate




Stworzyliśmy całkowicie nową metodę (*getManager()* w klasie *Person)* która **deleguje** wywołanie do *Department*.

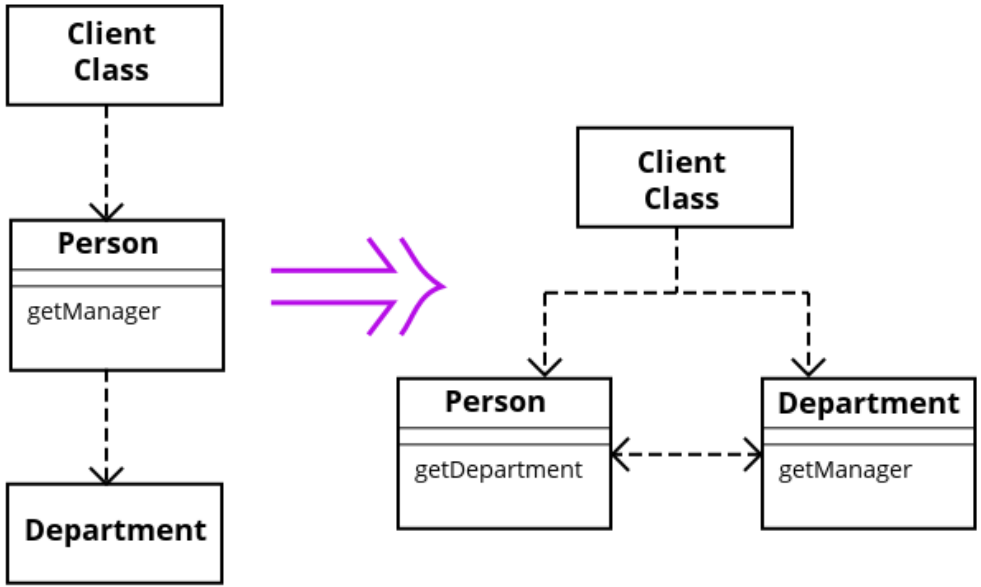
Now the client does not know about, or depend on class *Department*.

String managerName = person.getManager();

# Remove Middle Man

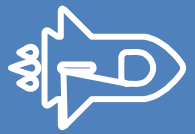
A class is doing too much simple delegation.

***Get the client to call the delegate directly.***




**Inverse of** *Hide Delegate*

# INNE

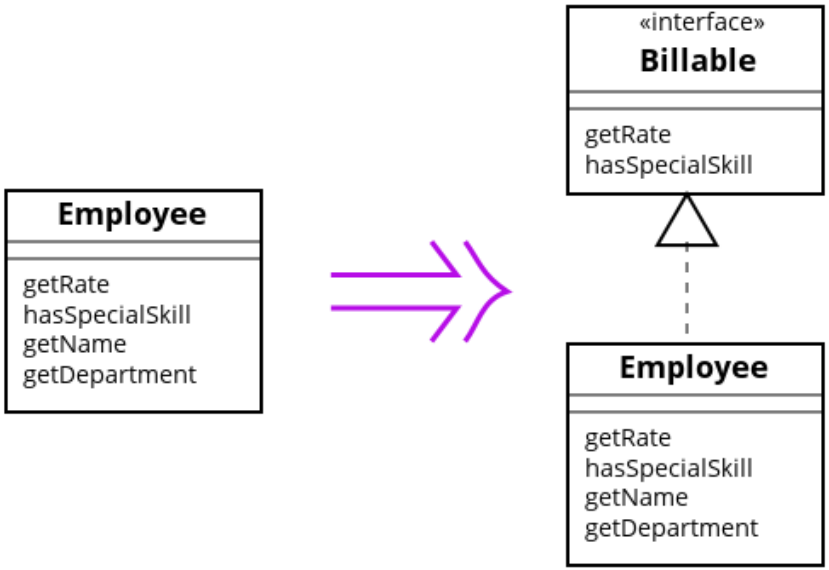


Extract interface
Encapsulate Field
Replace exception with test
Extract Class
Split temporary variable

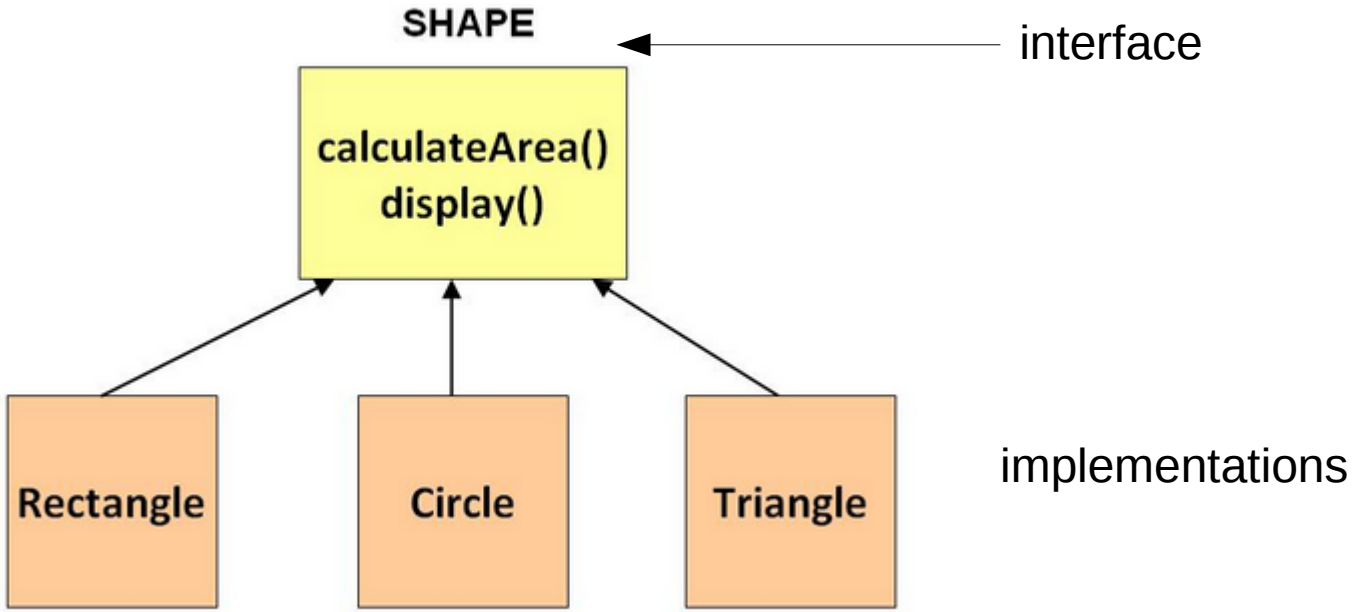
# Extract Interface

Several clients use the same subset of a class's interface, or two classes have part of their interfaces in common.

***Extract the subset into an interface.***



# Extract Interface



# Encapsulate Field

## Problem

You have a public field.

```
class Person {
  public String name;
}
```

## Solution

Make the field private and create access methods for it.

```
class Person {
  private String name;

  public String getName() {
    return name;
  }
  public void setName(String arg) {
    name = arg;
  }
}
```

## Why Refactor

One of the pillars of object-oriented programming is *Encapsulation*, the ability to conceal object data. Otherwise, all objects would be public and other objects could get and modify the data of your object without any checks and balances! Data is separated from the behaviors associated with this data, modularity of program sections is compromised, and maintenance becomes complicated.

# Encapsulate field/ extract interface - ćwiczenia

Wykonaj ćwiczenie opisane w klasie Okrag.java jako todo D.

# Replace Exception with Test

You are throwing an exception on a condition the caller could have checked first.

***Change the caller to make the test first.***

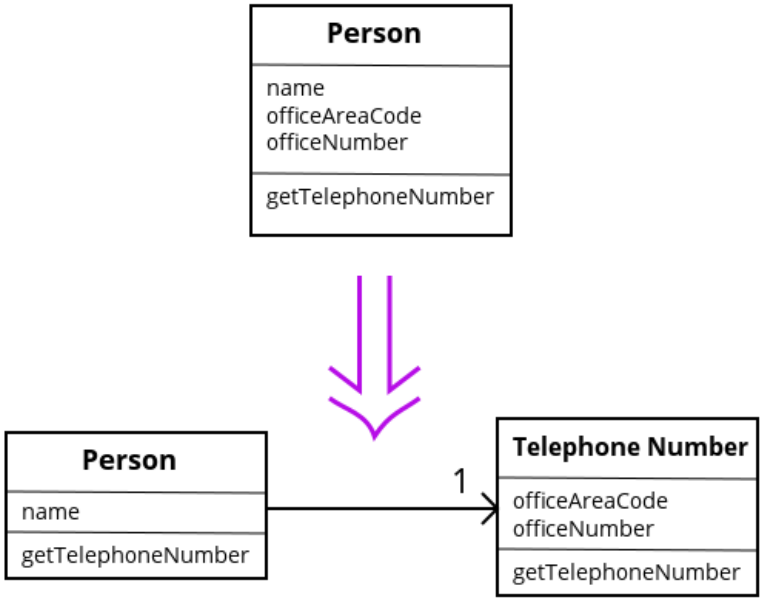
```
double getValueForPeriod (int periodNumber) {
  try {
    return _values[periodNumber];
  } catch (ArrayIndexOutOfBoundsException e) {
    return 0;
  }
}
```

```
double getValueForPeriod (int periodNumber) {
  if (periodNumber >= _values.length) return 0;
  return _values[periodNumber];
}
```

# Extract Class

You have one class doing work that should be done by two.

***Create a new class and move the relevant fields and methods from the old class into the new class.***




**Inverse of** *Inline Class*

# Split Temporary Variable



You have a temporary variable assigned to more than once, but is not a loop variable nor a collecting temporary variable.

***Make a separate temporary variable for each assignment.***

```
double temp = 2 * (_height + _width);
System.out.println (temp);
temp = _height * _width;
System.out.println (temp);
```

```
final double perimeter = 2 * (_height + _width);
System.out.println (perimeter);
final double area = _height * _width;
System.out.println (area);
```

# Split Temporary Variable

You have a temporary variable assigned to more than once, but is not a loop variable nor a collecting temporary variable.

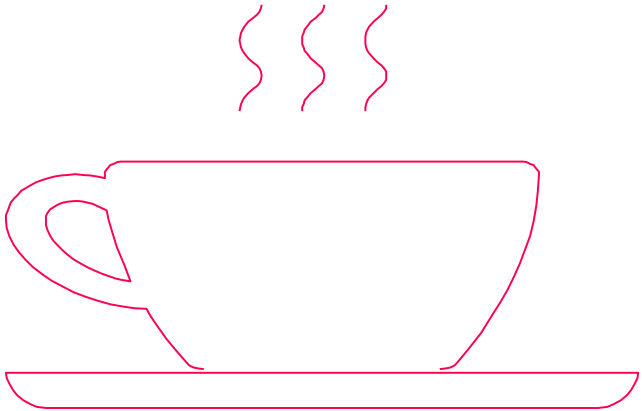
***Make a separate temporary variable for each assignment.***

```
double temp = 2 * (_height + _width);
System.out.println (temp);
temp = _height * _width;
System.out.println (temp);
```

```
final double perimeter = 2 * (_height + _width);
System.out.println (perimeter);
final double area = _height * _width;
System.out.println (area);
```

variable names are useful to document intermediate steps

# Koniec



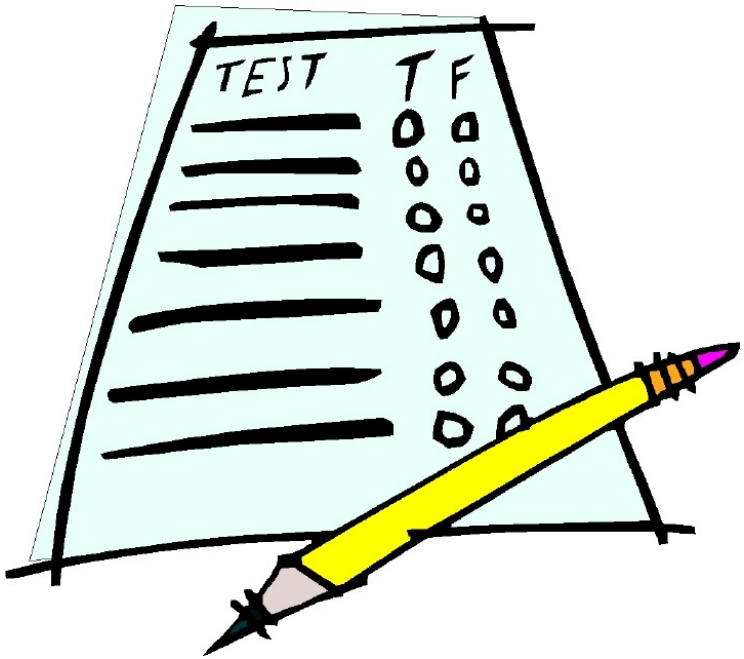
* to jeszcze nie koniec, pora na kilka porad ...

# Krok po kroczku

# Krok po kroczku

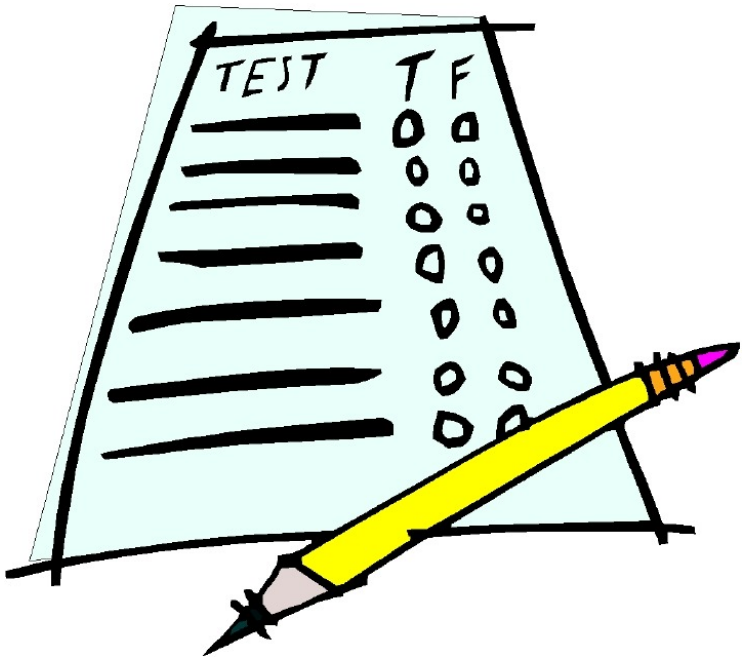
Proces refaktoryzacji polega na wprowadzaniu niewielkich zmian w kolejnych krokach. Dzięki temu, gdy się pomylisz, łatwo odnajdziesz błąd.

# Must have



# Must have

Przed przystąpieniem do refaktoryzacji upewnij się, że masz solidny pakiet testów. Testy muszą być samosprawdzalne.



## Java 8 – refaktoryzacja starego kodu

Java 8 Inspections in IntelliJ IDEA 2016.3:
https://www.youtube.com/watch?list=PLPZy-hmwOdEVEu_SlNs255dYZmriZekBW&v=KKmejHQ_jCA

## Literatura


1. refactoring.guru
2. refactoring.com
3. www.jetbrains.com/help/idea/refactoring-source-code.html
4. “Refaktoryzacja. Ulepszanie struktury istniejącego kodu” Martin Fowler, Kent Beck, John Brant, William Opdyke, Don Roberts

# Koniec