

# Refactoring kodu Java

## infoShare ACADEMY



# Witam

## Maciej Adamski

Senior Java Developer w Genesis Bluebird

# Refaktorинг kodu Java - Agenda

- Refaktoryzacja ?
- Kiedy refaktorować ?
- Narzędzia i techniki
- Zasady dobrego refactoringu
- Praktyka
- Project Lombok

# Refaktoryzacja co to jest ?

- Proces poprawiania konstrukcji istniejącego kodu bez widocznych zmian w jego zachowaniu, nie zmieniając funkcjonalności / logiki biznesowej
- Służy on utrzymywaniu wysokiej jakości organizacji systemu

# Refaktoryzacja

- Kto uwielbia refactoring ?      Programiści



# Refaktoryzacja

- Kto nienawidzi refactoringu ?      QA / Biznes



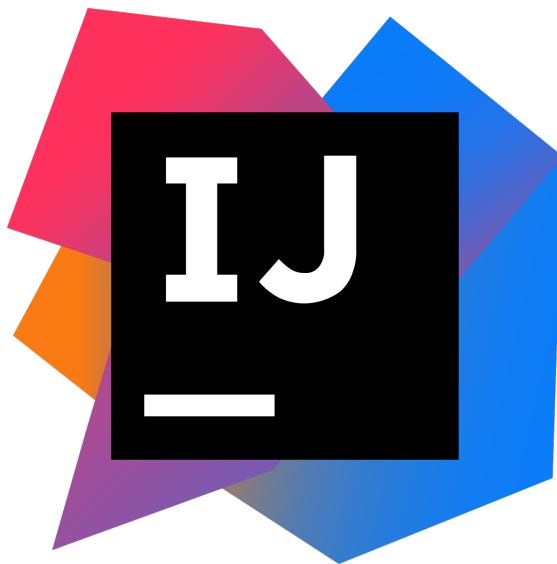
# Refaktoryzacja kiedy ?

- Refaktoryzacja powinna być **ciągłą** częścią procesu tworzenia kodu, w celu **ciąglej** poprawy jego jakości
  - Każdy programista powinien codziennie wykonywać czynności refaktoryzacyjne
  - Zawsze zostawiaj obóz czystszy, niż go zastałeś.
- 
- Jaki nakład pracy będzie potrzebny ?
  - Jak to się ma do „objętości” całego zadania ?
  - Czy na pewno niczego nie zepsuje ?

# Refaktoryzacja kiedy ?

- Kod jest ciężki do przeczytania przez innego programistę
- Posiada zduplikowaną logikę
- Posiada skomplikowaną logikę warunków
- Potrzebuje dodatkowego zachowania, a więc wymusza to na programiście zmianę działającego kodu
- Łamie zasadę otwarte – zamknięte czyli klasy / funkcję powinny być otwarte na rozszerzenie, ale zamknięte na modyfikację. Czyli jest trudny w utrzymaniu.
- Kent Beck + Robert Martin "Uncle Bob" + Joshua Bloch

# Narzędzia i techniki



<https://www.jetbrains.com/help/idea/refactoring-source-code.html>

# Narzędzia i techniki

- Co robić jak nie mam czasu na refaktoring ?
- //TODO
- //FIXME

```
public class Main {  
  
    //TODO Add some useful code here  
  
    /*TODO  
     * remove unnecessary variable  
     * add business logic  
     * */  
  
    //FIXME delete unnecessary code after integrate on proper API  
  
    //TODO  
    //implement business logic  
  
    //FIXME  
    //consider re-implement that method is hard to maintain  
  
    //FIXME Add more class files:  
    // - Player  
    // - NPC  
    // - Chicken  
}
```

# Narzędzia i techniki

TODO: Project Current File Scope Based

▼ Found 6 TODO items in 1 file

▼ com.company 6 items

▼ Main.java 6 items

- (5, 7) // *TODO Add some useful code here*
- (7, 7) /\* *TODO*
- (12, 7) // *FIXME delete unnecessary code after integrate on proper API*
- (14, 7) // *TODO*
- (17, 7) // *FIXME*
- (20, 7) // *FIXME Add more class files:*
  - // - *Player*
  - // - *NPC*
  - // - *Chicken*

# Narzędzia i techniki



Actually refactoring  
and fixing the code

// TODO Everywhere

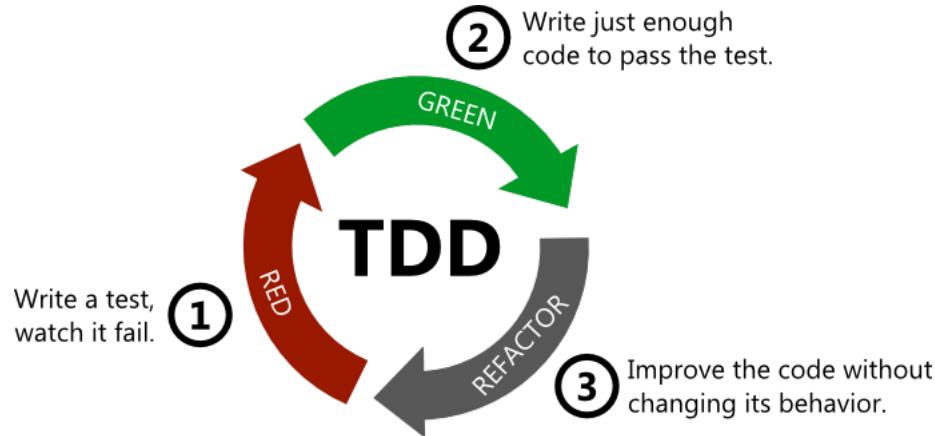
Some of my dear co-workers

# Narzędzia i techniki – dobre praktyki

- Scratch refactoring - zmiana tylko małego fragmentu kodu i sprawdzenie jakie elementy systemu zostały naruszone, dzięki temu poznajemy system dużo lepiej
- Large scale refactoring – wielka zmiana w systemie, wręcz „przepisanie” części systemu.

# Narzędzia i techniki – dobre praktyki

- TDD – test driven development lecz w odwróconej formie



# Narzędzia i techniki – dobre praktyki



FIX YOUR  
CODE TO  
PASS UNIT TESTS

FIX YOUR  
UNIT TESTS TO  
PASS YOUR CODE

# Narzędzia i techniki – co nie jest refactoringiem ?

- Naprawa bugów
- Optymalizacje
- Zawężenie obsługi błędów
- Sprawianie, że kod jest łatwiejszy do testowania

# Zasady dobrego refactoringu

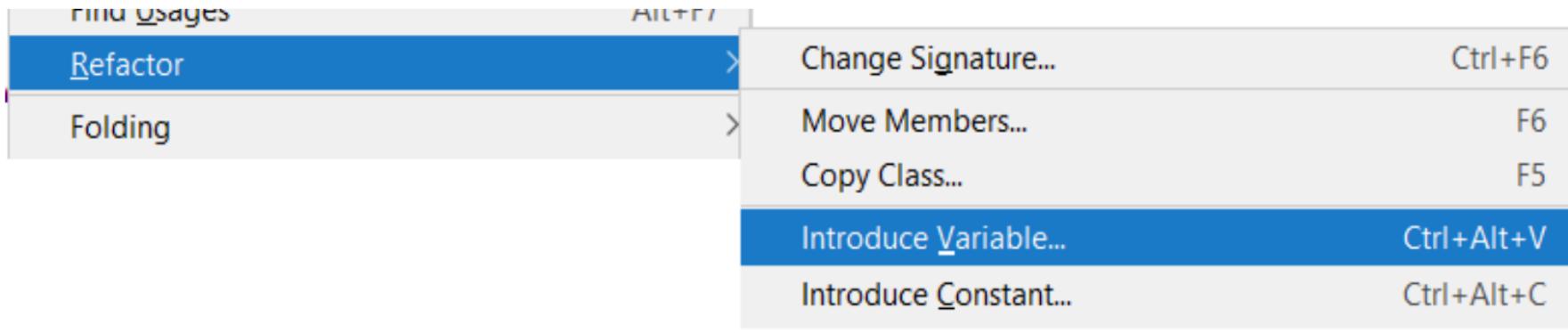
- Zaufaj swojemu IDE (gdzie tylko jest to możliwe)
- Rób to manualnie jeżeli IDE nie wie co poprawić
- Polegaj na testach unitowych, aby mieć pewność, że wszystko działa poprawnie po wprowadzeniu zmian
- Spraw aby to było Twoje rutynowe zajęcie każdego dnia pracy

# Praktyka

- Sklonuj repozytorium i zainportuj projekt Mavenowy do IntelliJ
- <https://github.com/infoshareacademy/jdz8-materialy-refactoring>

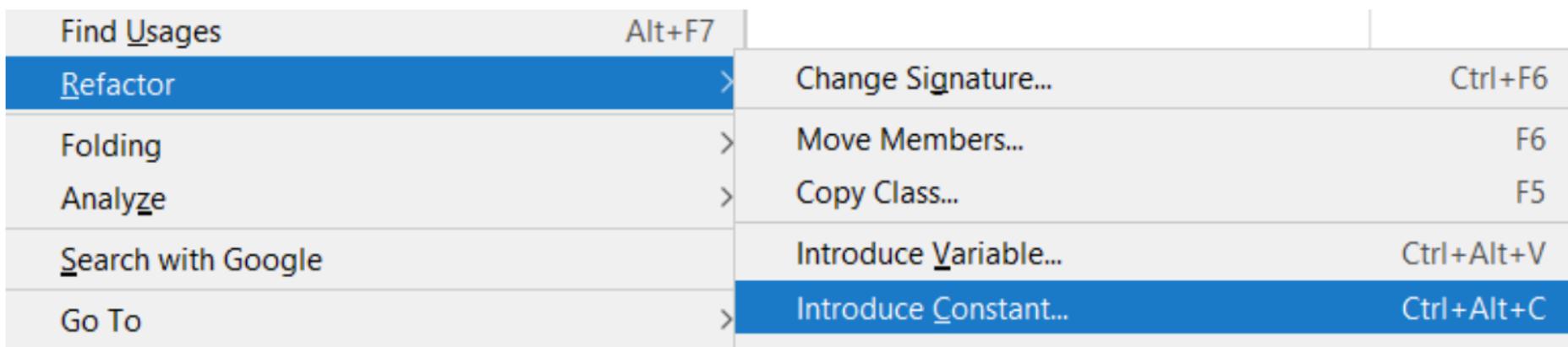
# Zadanie 1 – Extract Variable / Introduce Variable

- Otwórz klasę Zadanie1.class
- Za pomocą funkcji extract variable utwórz nową zmienną lokalną z drugaKlasa.jakasLiczba.intValue()
- Skrót klawiszowy w IntelliJ: CTRL + ALT + V



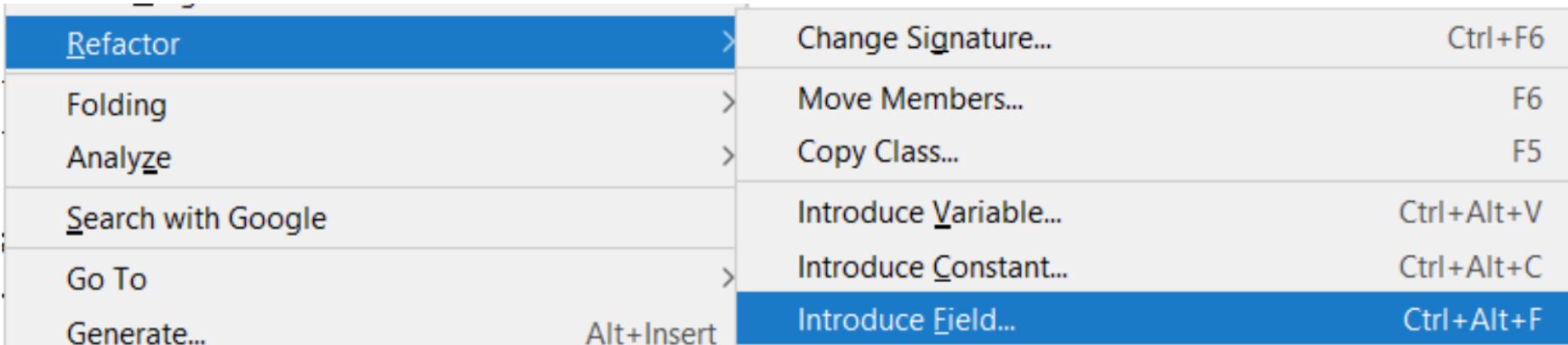
# Zadanie 2 - Extract Constant / Introduce Constant

- Otwórz klasę Zadanie2.class
- Wydziela zmienną “string” i tworzy z niej stałą (public static final)
- Skrót klawiszowy w IntelliJ: Ctrl + Alt + C



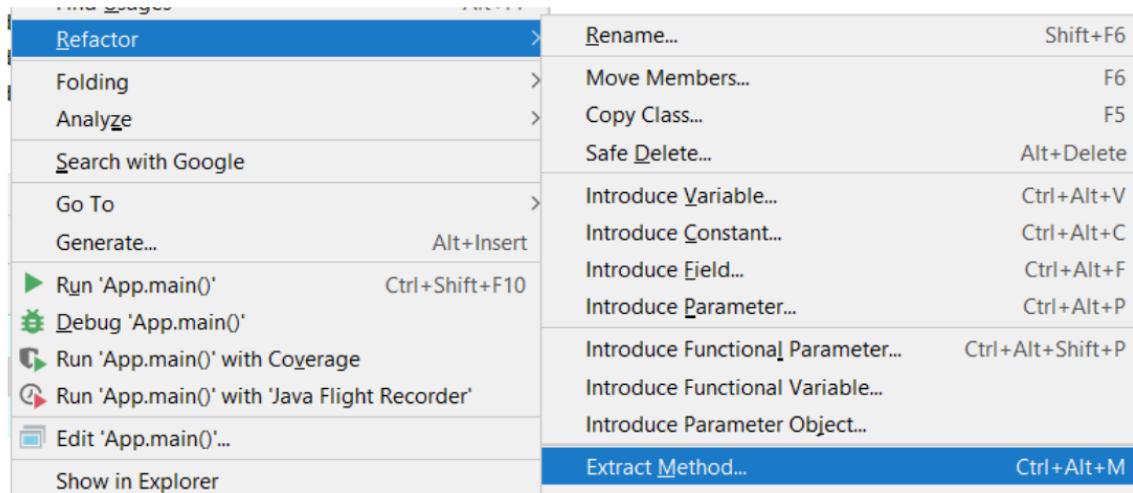
# Zadanie 3 – Extract Field / Introduce Field

- Otwórz klasę Zadanie3.class
- Za pomocą funkcji extract field utwórz nową pole w klasie z drugaKlasa.jakasLiczba.intValue()
- Skrót klawiszowy w IntelliJ: Ctrl + Alt + F



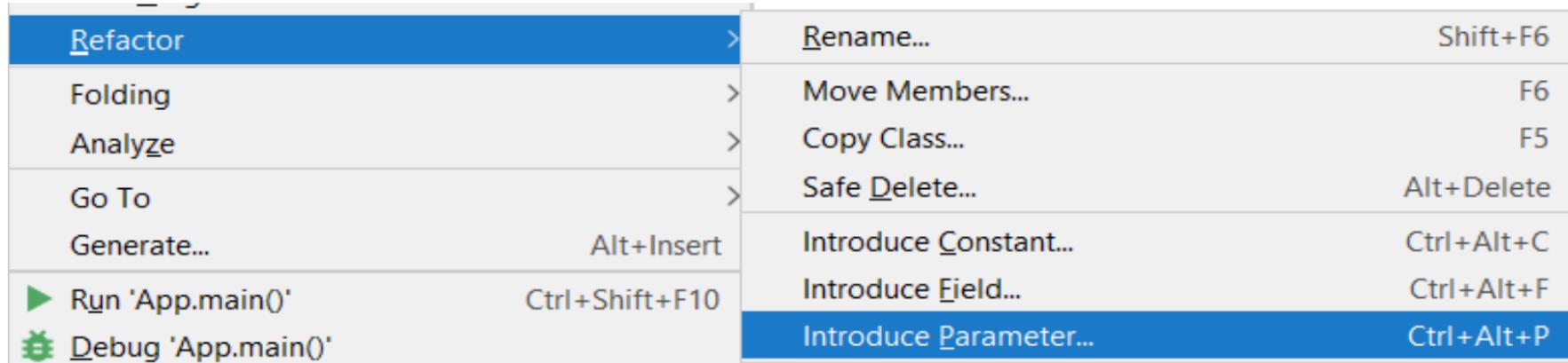
# Zadanie 4 – Extract Method

- Otwórz klasę Zadanie4.class
- Wydziel metodę dodawania
- Skrót klawiszowy w IntelliJ: Ctrl + Alt + M



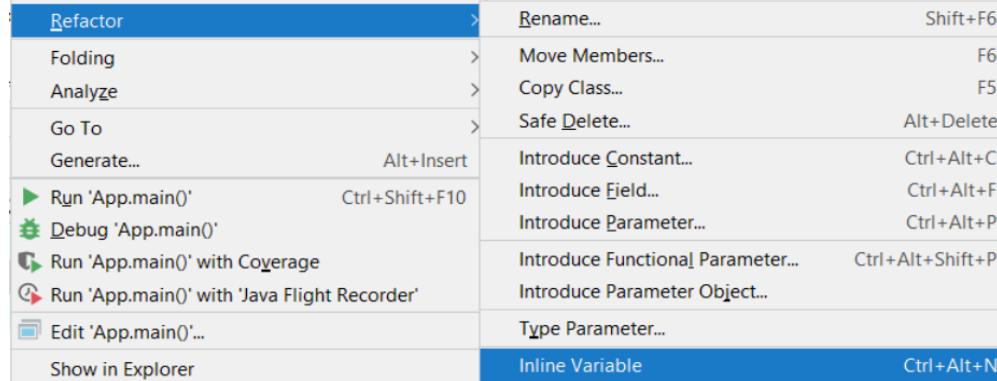
# Zadanie 5 – Extract Parameter /Introduce Parameter

- Otwórz klasę Zadanie5.class
- Wydziel lokalny tekst w metodzie generujTekst, aby zostało podane jako argument metody
- Skrót klawiszowy w IntelliJ: Ctrl + Alt + P



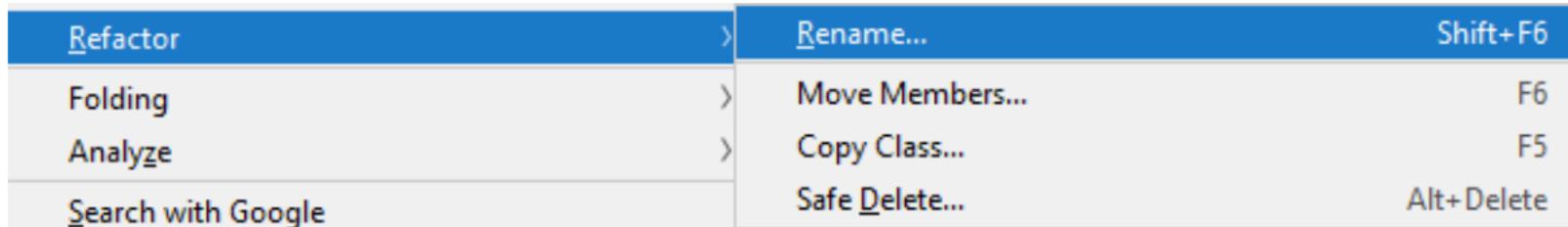
# Zadanie 6 – Inline Elements.

- Otwórz klasę Zadanie6.class
- Przeciwieństwo od Extract Variable. Zlikwiduj niepotrzebna zmienna “number” za pomocą funkcji inline elements.
- Skrót klawiszowy w IntelliJ: Ctrl + Alt + N



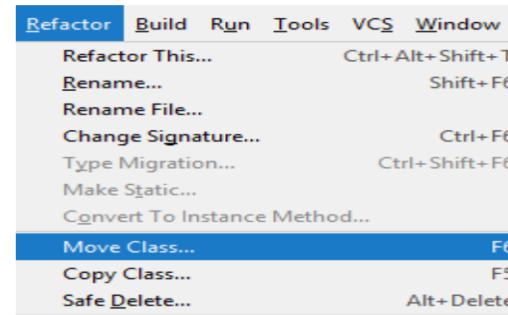
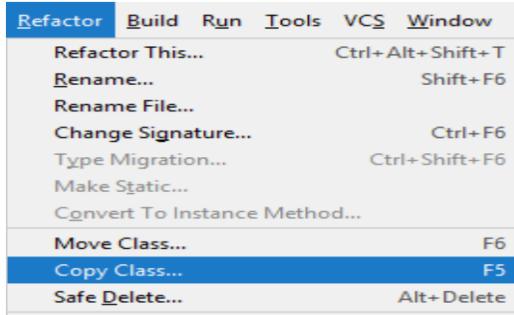
# Zadanie 7 - Rename

- Otwórz klasę Zadanie7.class
- Zmienia nazwę z uwzględnieniem wszystkich miejsc występowania.
- Będąc w klasie Zadanie7 klikając na pole zadanie7a zmień nazwę klasy na inną. Następnie będąc w klasie Zadanie7 zmień nazwę metody "metoda7a" na inną.
- Skrót klawiszowy w IntelliJ: Shift + F6



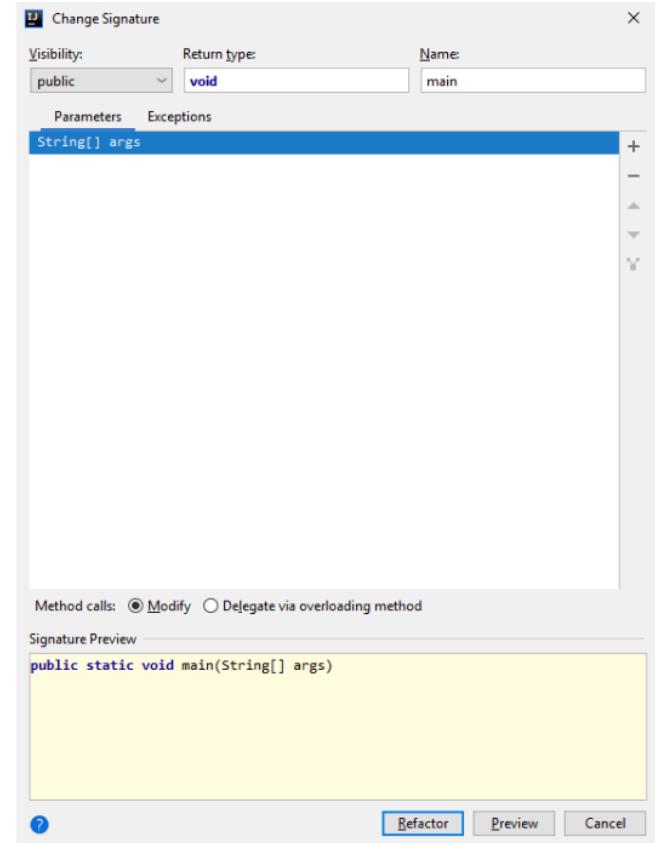
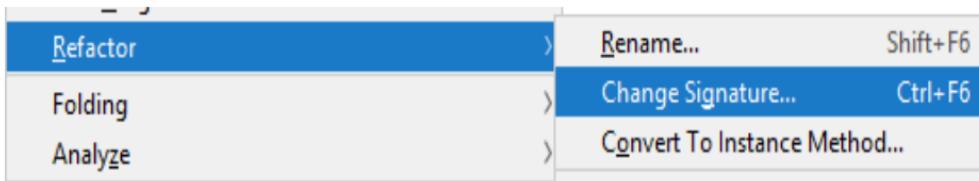
# Zadanie 8 – Copy/Move

- Otwórz klasę Zadanie8.class
- Kopiuje/przenosi z uwzględnieniem poprawy importów.
- Skopiuj klasę Zadanie8 używając funkcji copy class (Nazwij klasę Zadanie8a). Następnie z klasy Zadanie8a usuń metodę metoda8. Następnie skopiuj metodę metoda8 do klasy Zadanie8a używając metody move members (F6)
- Skrót klawiszowy w IntelliJ: F5/F6



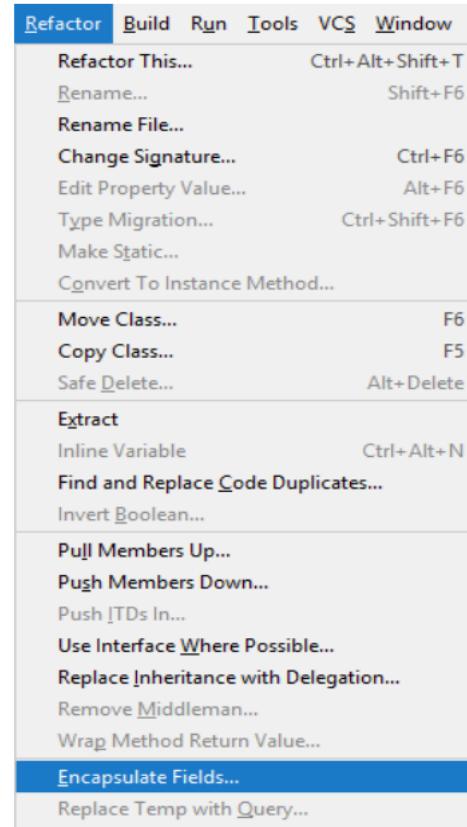
# Zadanie 9 – Change Signature

- Otwórz klasę Zadanie9.class
- Zmienia sygnaturę metody mojaMetoda.  
Zmienna wiek powinna być typu Integer  
natomiast imie typu String. Zmień również typ  
zwracany na String
- Skrót klawiszowy w IntelliJ: CTRL + F6



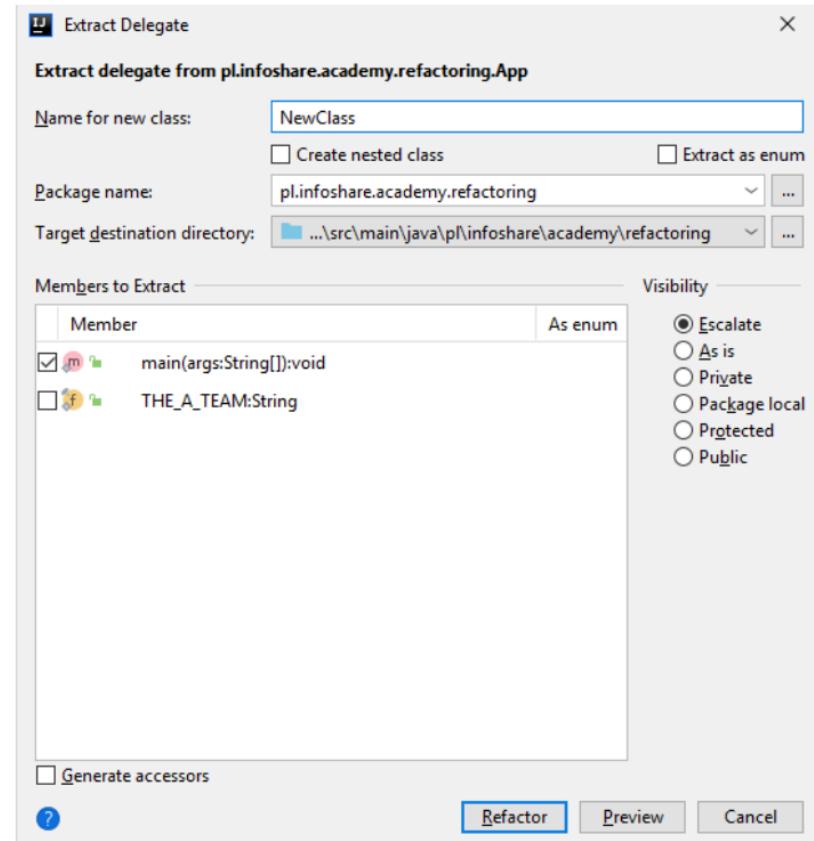
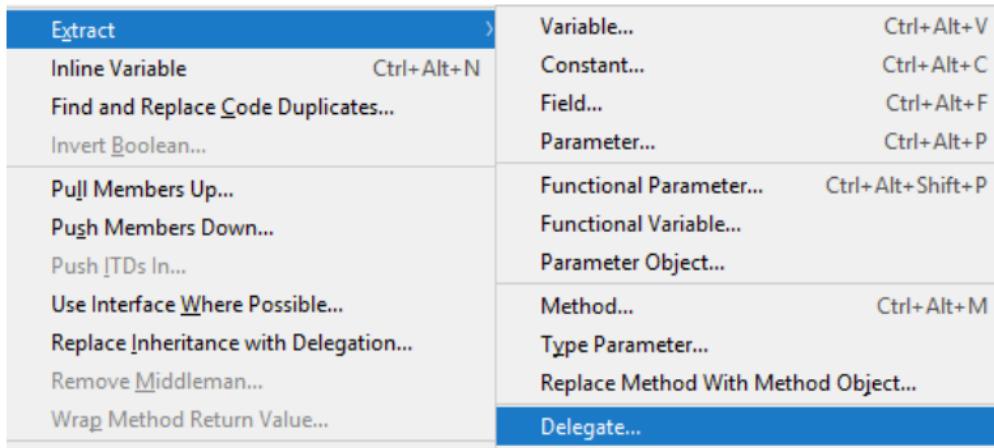
# Zadanie 10 – Encapsulate Fields.

- Otwórz klasę Zadanie10.class i Zadanie10a.class
- Enkapsuluje(hermetyzuje) pola. Hermetyzacja czyli ukrywanie pewnych danych składowych lub metod obiektów danej klasy.
- Za pomocą funkcji Encapsulate Fields hermetyzuj pole “mojString”. Zobacz co stało się w klasie Zadanie10a.



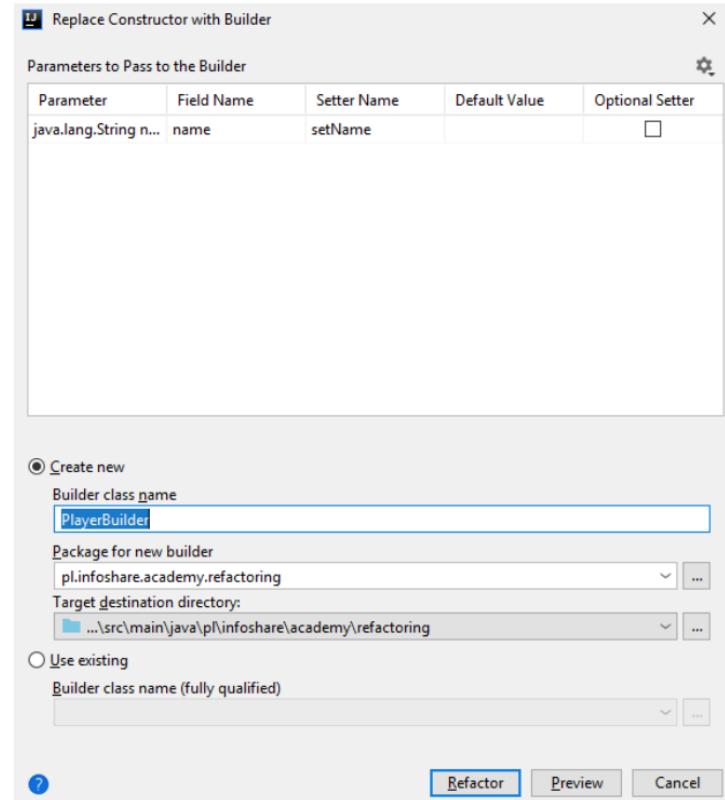
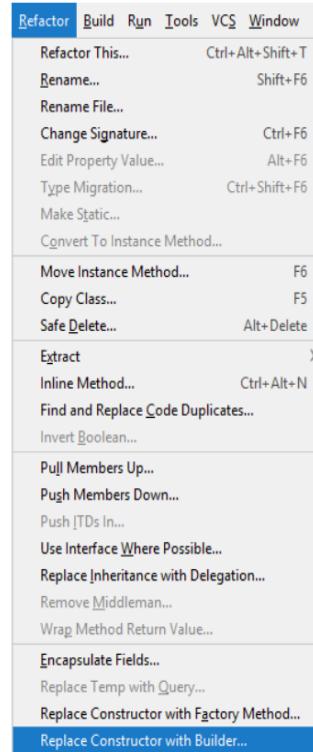
# Zadanie 11 – Extract Delegate.

- Otwórz klasę Zadanie11.class
- Wydziela członków klasy do nowej
- Wydziel pole mojString do nowej klasy



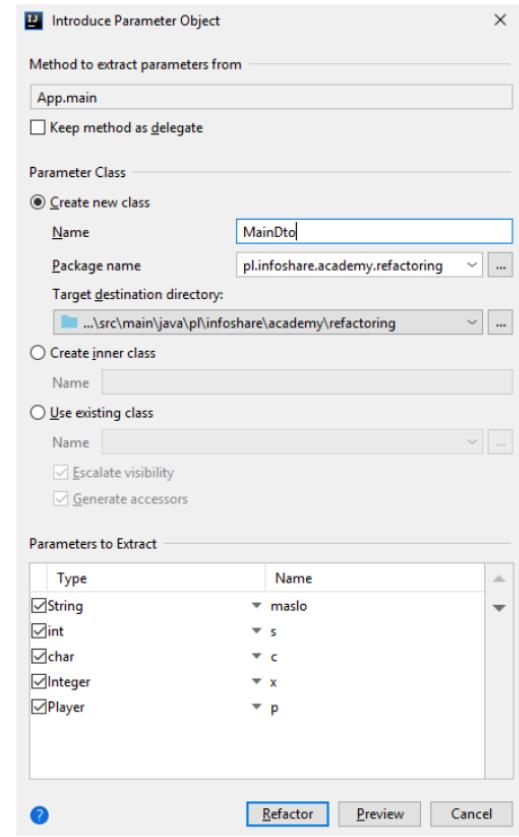
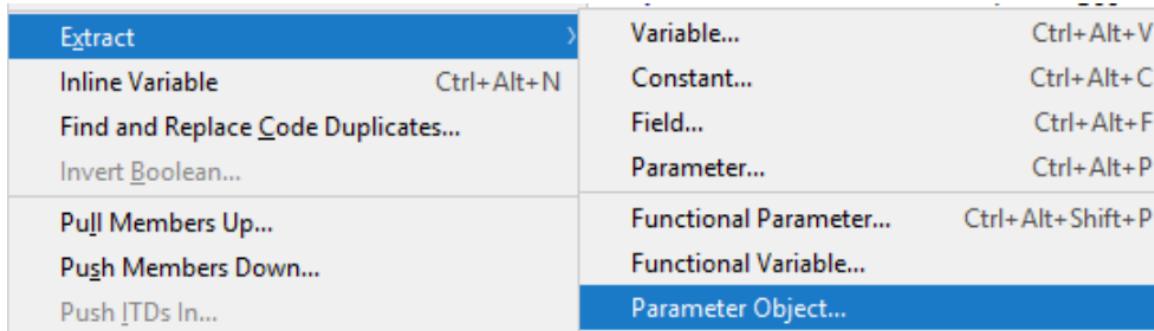
# Zadanie 12 – Replace Constructor with Builder.

- Otwórz klasę Zadanie12.class i Zadanie12a.class
- Zastępuje sposóbinicjalizacji klasy z użycia konstruktora na wzorzec projektowy Builder (budowniczego)
- Zmien konstruktor klasy Zadanie12 na wzorzec budowniczego zobacz co się stało w klasie Zadanie12a.class



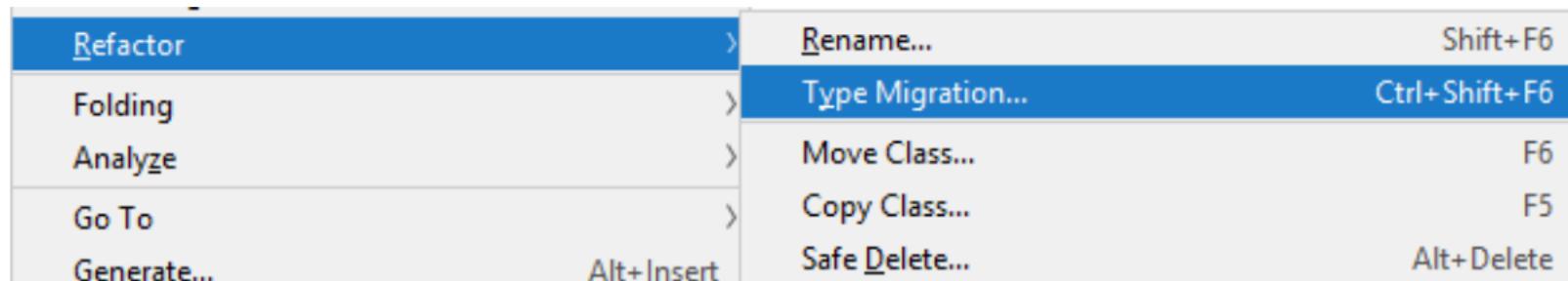
# Zadanie 13 – Parameter Object.

- Otwórz klasę Zadanie13.class
- Wydziela argumenty metody do obiektu, który służy wyłącznie do przekazywania argumentów (DTO – Data Transfer Object)



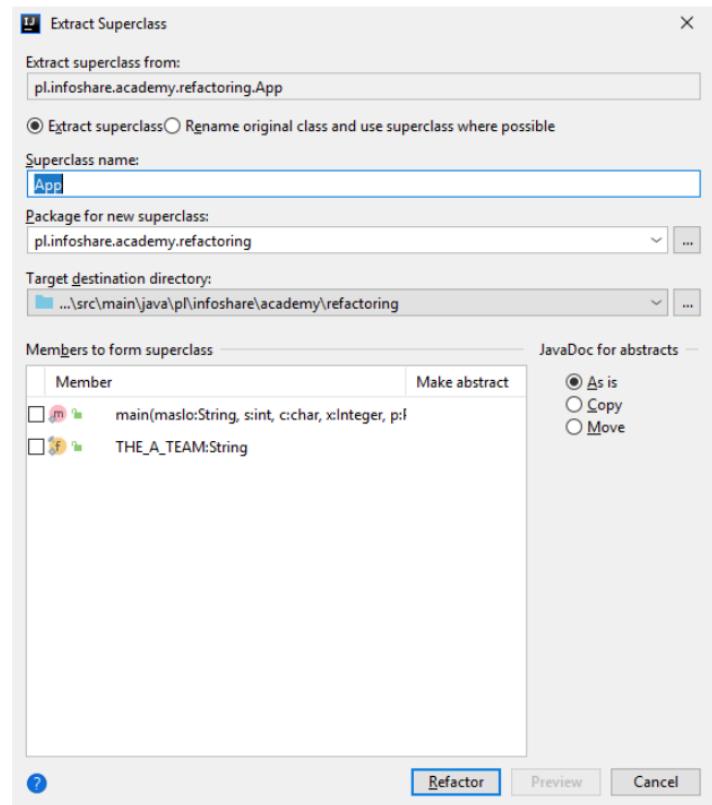
# Zadanie 14 – Type Migration.

- Otwórz klasę Zadanie14.class
- Zmienia / migruje typ danego pola
- Zmień parametr wejściowy w klasie Zadanie14a na Stringa
- Skrót klawiszowy w IntelliJ: Ctrl + Shift + F6



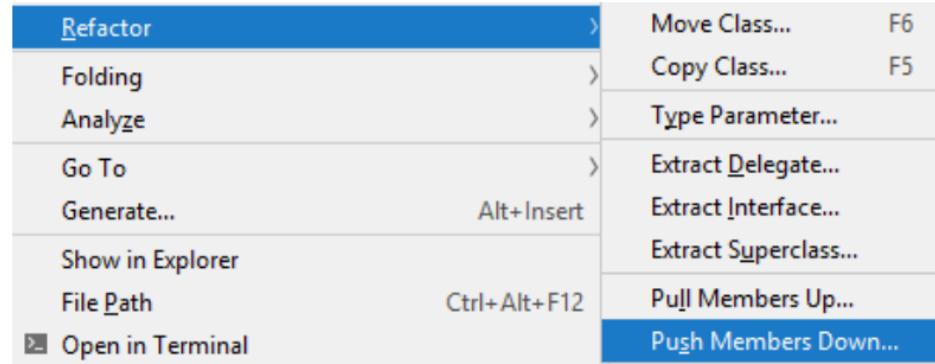
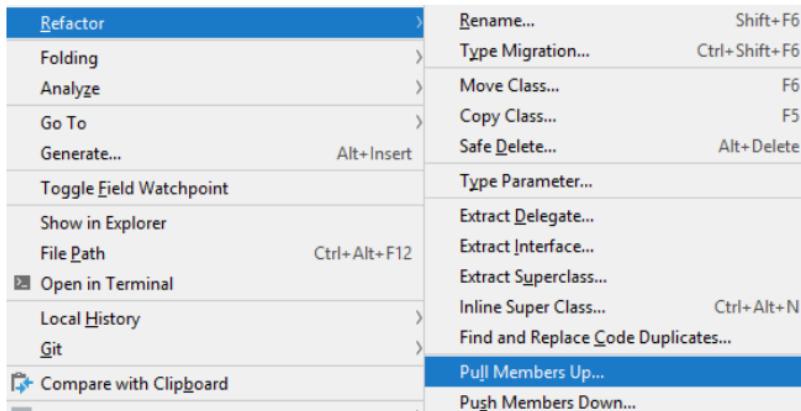
# Zadanie 15 - Extract Superclass.

- Otwórz klasę Zadanie15.class
- Dostarcza klasę nadziedną
- Wydziel pola oraz getter + setter do nowej klasy nadziednej



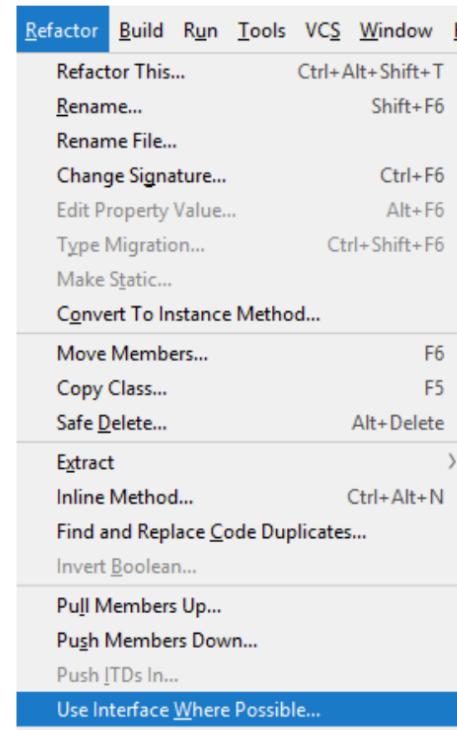
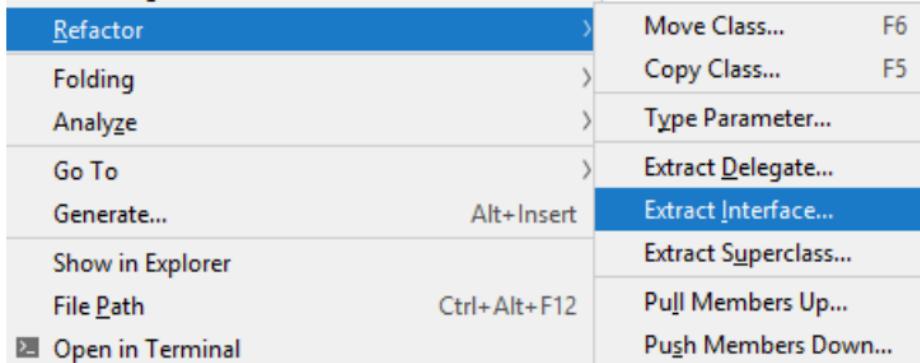
# Zadanie 16 – Pull Members Up / Push Members Down.

- Otwórz klasę Zadanie15.class
- Wydziela członków klasy do klasy nadzędnej / podrzędnej
- Wydziel metodę publicMethod do klasy nadzędnej utworzonej w zadaniu poprzednim



# Zadanie 17 - Extract Interface.

- Otwórz klasę Zadanie17.class
- Wydziela interfejs
- Wydziel metody publicMethod i secretMethod do interfejsu



# Project Lombok – brak boilerplate

- <https://projectlombok.org/>

## Lombok features

The [Lombok javadoc](#) is available, but we advise these pages.

### `val`

Finally! Hassle-free final local variables.

### `var`

Mutably! Hassle-free local variables.

### `@NonNull`

or: How I learned to stop worrying and love the NullPointerException.

### `@Cleanup`

Automatic resource management: Call your `close()` methods safely with no hassle.

### `@Getter/@Setter`

Never write `public int getFoo() {return foo;}` again.

### `@ToString`

No need to start a debugger to see your fields: Just let lombok generate a `toString` for you!

### `@EqualsAndHashCode`

Equality made easy: Generates `hashCode` and `equals` implementations from the fields of your object..

### `@NoArgsConstructor, @RequiredArgsConstructor and @AllArgsConstructor`

Constructors made to order: Generates constructors that take no arguments, one argument per final / non-nullfield, or one argument for every field.

### `@Data`

All together now: A shortcut for `@ToString`, `@EqualsAndHashCode`, `@Getter` on all fields, and `@Setter` on all non-final fields, and `@RequiredArgsConstructor`!

# Project Lombok – brak boilerplate

## @Value

Immutable classes made very easy.

## @Builder

... and Bob's your uncle: No-hassle fancy-pants APIs for object creation!

## @SneakyThrows

To boldly throw checked exceptions where no one has thrown them before!

## @Synchronized

`synchronized` done right: Don't expose your locks.

## @With

Immutable 'setters' - methods that create a clone but with one changed field.

## @Getter(lazy=true)

Laziness is a virtue!

## @Log

Captain's Log, stardate 24435.7: "What was that line again?"

## experimental

Head to the lab: The new stuff we're working on.

# Project Lombok – brak boilerplate

```

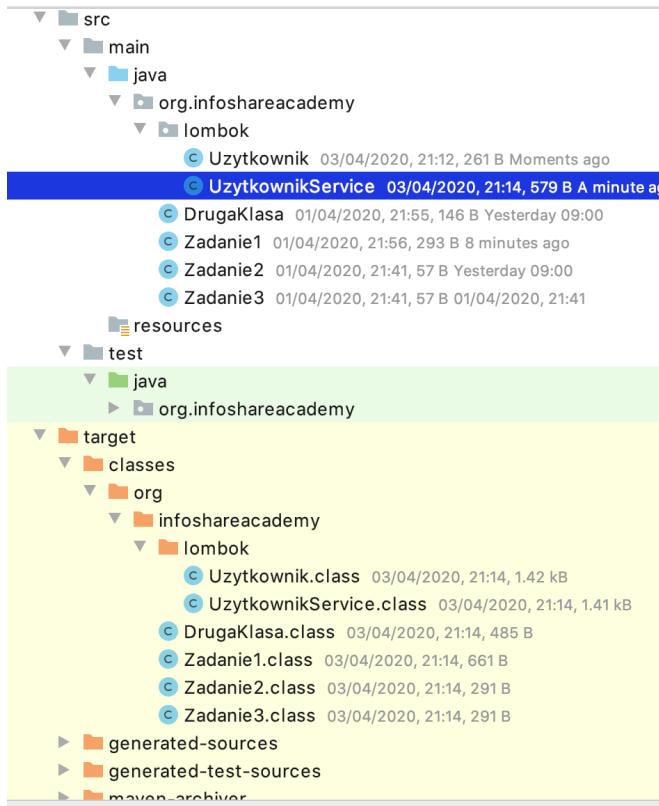
package org.infoshareacademy.lombok

import lombok.Builder;
import lombok.Getter;
import lombok.ToString;

@Getter
@Builder
@ToString

public class Uzytkownik {
    private final String imie;
    private final String nazwisko;
    private final Integer wiek;
}

```



# Project Lombok – brak boilerplate

```
public class Uzytkownik {  
    private final String imie;  
    private final String nazwisko;  
    private final Integer wiek;  
  
    Uzytkownik(String imie, String nazwisko, Integer wiek) {  
        this.imie = imie;  
        this.nazwisko = nazwisko;  
        this.wiek = wiek;  
    }  
  
    public static Uzytkownik.UzytkownikBuilder builder() { return new Uzytkownik.UzytkownikBuilder(); }  
  
    public String getImie() { return this.imie; }  
  
    public String getNazwisko() { return this.nazwisko; }  
  
    public Integer getWiek() { return this.wiek; }  
  
    public String toString() { return "Uzytkownik(imie=" + this.getImie() + ", nazwisko=" + this.getNazwisko() + ", wiek=" + this.getWiek() + ")"; }  
  
    public static class UzytkownikBuilder {  
        private String imie;  
        private String nazwisko;  
        private Integer wiek;
```

# Project Lombok – brak boilerplate

```
UzytkownikBuilder() {  
}  
  
public Uzytkownik.UzytkownikBuilder imie(String imie) {  
    this.imie = imie;  
    return this;  
}  
  
public Uzytkownik.UzytkownikBuilder nazwisko(String nazwisko) {  
    this.nazwisko = nazwisko;  
    return this;  
}  
  
public Uzytkownik.UzytkownikBuilder wiek(Integer wiek) {  
    this.wiek = wiek;  
    return this;  
}  
  
public Uzytkownik build() { return new Uzytkownik(this.imie, this.nazwisko, this.wiek); }  
  
public String toString() { return "Uzytkownik.UzytkownikBuilder(imie=" + this.imie + ", nazwisko=" + this.nazwisko + ", wiek=" + this.wiek + ")"; }  
}
```



# Dzieki

Kontakt:

<https://www.linkedin.com/in/maadamski/>

email: [maciejadamski0@gmail.com](mailto:maciejadamski0@gmail.com)

skype: maciejadamski0