

## Spis treści

GIT .....	4
JAVA .....	5
Polimorfizm na bazie klasy .....	5
Tworzenie podklasy ( dziedziczenie) .....	6
Klasy abstrakcyjne .....	7
Konwersja liczb ze stringa.....	9
Zmiana tablicy charów na string i string na tablice char.....	9
Formatowanie wyświetlanego tekstu (automatyczne formatowanie wyświetlanych liczb).....	10
Zmienne.....	10
Operatorzy: .....	11
Tworzenie tablicy .....	11
Tworzenie listy ( ArrayList) Lista jest indeksowana .....	11
Tworzenie listy HashSet ( Lista w której dane nie mogą się powtarzać).....	12
Tworzenie HashMap map<klucz,danepobierane(wartość)> .....	12
Switch case:.....	12
Przekształcenie zmiennej pobranej lub jakiejkolwiek.....	12
Pobieranie daty z urządzenia na którym uruchomiony jest program.....	13
IF .....	13
Tworzenie funkcji .....	13
Pętle for /while/do/for each.....	14
Pobieranie z konsoli.....	14
Enum .....	16
Formatowanie String .....	16
Kopiowanie plików od javy 7:.....	17
Wczytywanie pliku json.....	17
Varrags czyli skalowalna metoda.....	18
Interfejs funkcyjny .....	20
STREAM .....	22
Stream wielowątkowy.....	23
Combine stream.....	23
Kopiowanie .....	24
Rekurencja recursion .....	24
Łączenie dwóch funkcji Compose Function.....	25
Optional.....	26
Sortowanie Comparable .....	26
Zapisywanie danych do pliku.....	27
String replaceall.....	27
TESTOWANIE JUNIT 5.....	28

Asercje: .....	31
MATCHERY .....	31
Hamcrest .....	31
AssertJ.....	32
Globalne zmienne Junit5 (setup,tearDown).....	33
Testy Parametryzowane .....	33
Extension Model .....	35
Testy dynamiczne .....	36
Uruchamianie testów.....	37
Uruchamianie tylko wybranych testów (filtrowanie)@Tag.....	38
Zasady testów jednostkowych FIRST .....	38
Zasady testów jednostkowych CORRECT .....	39
Test Coverage – pokrycie testami kodu .....	39
Mockito 2 .....	40
Stuby .....	40
Mocki oraz implementacja mockito.....	42
Weryfikacja wywołań metod Metoda verify().metoda().....	43
Argument Matchery any() .....	44
Matchery z wyrażeniami lambda ( Filtrowanie obiektów z którymi wywoływana jest metoda) .....	44
Wyrzucanie wyjątków za pomocą given().willThrow() oraz ich obsługa assertThrows() .....	45
ArgumentCaptor – przechwytywanie przekazywanych obiektów .....	45
doAnswer(lambda).when(obiekt).metoda Przechwycenie oraz modyfikacja przekazanego obiektu....	46
Wywołanie prawdziwej metody na mocku given().wilcallrealmethod .....	47
Adnotacje oraz uproszczenie kodu @InjectMocks @Mock @ExtendWith @Captor @MockitoSettings .....	47
Zwiększa dokładność błędów .....	48
TDD .....	48
Wzorce projektowe .....	49
Observer .....	50
Singleton.....	51
Builder z klasa wewnętrzna.....	52
Builder Classic .....	53
Factory FABRIC.....	54
Factory Abstract.....	55
Adapter .....	56
Decorator .....	57
ChainOfResponsibility.....	57
TemplateMethod .....	58
Command .....	59

Memento .....	60
Strategy .....	61
Visitor.....	62
State .....	63
Maven .....	64
Scope .....	64
LifeCycles .....	64
Phases of Lifcycles Fazy działania maven .....	65
Goals and Plugins.....	66
Version .....	66
Properties - zmienne .....	66
Multimodule .....	66
<dependencyManagement> oraz <pluginManagement>.....	67
LOMBOCK.....	67
JPA /Hibernate/H2.....	68
Utworzenie tabeli .....	69
Insert do tabeli .....	69
Select.....	70
Update .....	70
Delete .....	70
OneToOne .....	71
OneToMany .....	71
Spring Data JPA.....	72

# GIT

git clone url  
git branch –delete name  
**git branch -d branch** ( usuwa branche bez konfliktów)  
**git branch -D branch** ( ignoruje wszystko i poprostu usuwa)

**git log opgit**  
**git log –oneline**  
**git init** - tworzenie repozytorium  
**ls** - pokazanie plików  
**ls -a** - pokazanie ukrytych plików/folderów  
**cd**- change directory  
**git status** - status repozytorium  
**touch xyz.txt** - tworzenie pliku  
**git add**- dodanie do staging area  
**git add -A** dodanie wszystkich z folderu git add .  
**git commit -m"Opis"** - opis comitu message  
**git commit -am** – opis oraz dodanie to staging"D  
**git branch** - sprawdzenie branchu  
**git checkout -b"nazwabranca"** - nowy branch - kopia aktualnego brouncha (feature branch poboczny)  
git switch -c nazwabranca - kopia brancha na nowy feateuyre  
**git checkout nazwabranca** - zmiana brancha  
**git checkout -** - poprzedni branch na którym byliśmy  
**git merge nazwabranca** - połączenie brancha  
**git pull –rebase** pobranie danych z gita  
**git push** - wypchanie do neta  
**git mv \*.html** Modyfikacjanazwy\*html  
**git branch delete branch** - usuwanie brancha  
**git rm nazwapliku.\*** - usuwanie konkretnego pliku  
**rm \*.txt** - usuwanie  
**git clean -n** - pokazuje co usunie z nie śledzonych plików z bierzącego folderu  
**git clean -dn** pokazuje co usunie z nie śledzonych plików ze wszystkich folderów w bierzacej lokalizacji  
**git clean -df** faktycznie usuwa te pliki

**git stash** - cofanie modyfikacji z jej zapisaniem na stach list  
**git stash list** - lista stach  
**git stash pop** - ustawianie modyfikacji z listy stach wraz z jej usunięciem z tej listy  
**git restore .** - powrót do oryginału  
git stash pop takes a stashed change, removes it from the “stash stack”, and applies it to your current working tree.  
git stash apply takes a stashed change and applies it to your current working tree (also leaving it on the “stash stack”).  
git stash branch creates a new branch from the same commit you were on when you stashed the changes, and applies the stashed changes to that new branch

**git remote add <name(own)> <url>**  
**git remote add origin URL** - dodanie respozytorium  
**git push -u origin nazwabranca** - wypchanie repozytorium na branch master  
**git push -f origin master**  
**With the -f tag you will override the remote branch code with your local repo code.**

git remote remove name  
**git remote oldname newname**

**git branch -M nazwanbrancha** - zmiana nazwy bieżącego brancha

**git push -all** - wypchanie na wszystkie branche  
**git pull** - aktualizacja projektu f  
git fetch ???  
**rm \*.txt** - usuwanie  
**git reset –hard** - cofanie do niezapisanych comitów  
**git checkout 3.txt** — cofanie zmian w konkretnym pliku  
edytor vi - **vi \*.txt**  
wychodzić z vi z zapisywaniem- **esc+ :wq**  
wychodzić z vi z bez zapisywania- **esc+ :q!**  
tryb edycji - **literka a**  
**git commit –amend** – zmiana nazwy commita

Łączenie commitów: **git rebase -i HEAD~N** (n - liczba ostatnich commitów)  
Łączenie commitów do numeru(hasha komita poprzedniego) - **git rebase -i (HASH)**  
Komendy wewnętrzne

Łączenie branchy poprzez rebase feature brancha  
Jesteśmy na feature  
Przejście na master : **git checkout master**  
Aktualizacja mastera: **git pull**  
Sprawdzenie zmian : **git log**  
Przejście na feature : **git checkout -**  
Łączenie commitów: **git rebase --i HEAD~N**  
Połączenie mastera z feature na feature: **git rebase master**  
(rozwiązywanie konfliktów)  
Sprawdzenie zmian : **git log**  
Przejście na master : **git checkout master**  
Łączenie mastera z feature na masterze: **git merge feature**  
Usuwanie feature brancha: **git branch -d branch**

---

branch - gałęzia  
commit - zmiany  
track - widzane przez git  
untracked - niewidziane przez git  
working area- staging area - repositorygit

Czyszczenie plików z gitignore

```
git rm -r --cached .
git add .
git commit -m 'Removed all files that are in the .gitignore'
git push origin master
```

## JAVA

### Polimorfizm na bazie klasy

Załóżmy, że tworzymy aplikację do zarządzania dokumentami. Na początek posiadamy cztery klasy. Jako bazę mamy nadklasę - **DocumentItem**, która dla uproszczenia posiada tylko jedną metodę - **getDescription**. Mamy też trzy konkretne klasy pochodne, które dostarczają własne implementacje metody **getDescription**. Mówimy, że metody te przesyłają metodę z klasy bazowej, co zresztą jest zgodne z treścią rozdziału **Przesyłanie metod**.

```
public class DocumentItem {
```

```

public String getDescription() {
    return "Class representing some document.";
}
}

public class WordDocument extends DocumentItem {

    public String getDescription {
        return "Class representing word document.";
    }
}

public class ExcelDocument extends DocumentItem {

    public String getDescription {
        return "Class representing calculations.";
    }
}

public class PdfDocument extends DocumentItem {

    public String getDescription {
        return "Class representing pdf document.";
    }
}

```

Tym samym klasa bazowa `DocumentItem` staje się bytem przyjmującym wiele form, w zależności od tego, której implementacji użyjemy. Podczas działania programu może ona występować zarówno w postaci obiektu klasy `WordDocument`, `ExcelDocument`, jak i `PdfDocument`. Zobaczmy w programie:

```

public class Start {

    public static void main(String[] args) {

        DocumentItem wordItem = new WordDocument();
        DocumentItem excelItem = new ExcelDocument();
        DocumentItem pdfItem = new PdfDocument();

        System.out.println(wordItem.getDescription());
        System.out.println(excelItem.getDescription());
        System.out.println(pdfItem.getDescription());
    }
}

```

## Tworzenie podklasy (dziedziczenie)

```

// podklasa nie ma dostępu do pól prywatnych nadklasy
public class nazwapodklasy extends nazwanadklasy
{
//odwołanie się do metody z nadklasy
super.nazwametody();
//metody w podklasie można nazywać tak samo i przesłaniają one metode z nadklasy
}

```

```

//konstruktor podklasy
public nazwapodklasy(String n, double s)
{
//super to odwołanie do konstruktora nadklasy
super(n,s);
}
nowy obiekt Nazwaklasy nazwaobiektu = new Nazwakonstruktora();
przesłanianie metody - gdy metoda w podklasie ma taka samą nazwę i sygnature typów
polimorfizm - możliwość odwoływanego się przez obiekty do wielu różnych typów
wiązanie dynamiczne - automatyczny dobór odpowiednich metod podczas pracy programu przez
maszynę wirtualną
sygnatura metody - lista typów parametrów metody
działania i metoda math
final - przed metodą oznacza metodę której nie można prześlonić
final - przed class oznacza że nie możemy zrobić podklasy oraz wszystkie metody classy są finalne
final - przed zmienną oznacza że nie możemy jej zmienić po utworzeniu obiektu
rzutowanie - proces konwersji pomiędzy dwoma typami
double x = 3.405
int y = (int) x
rzutowanie obiektu:
Nameclass1[] staff = new Nameclass1[1]
staff[0] = new Nameclass1();
Nameclass1 variable = (Nameclass2) staff[0];

```

Sprawdzenie czy dany obiekt nadklasy jest w ją podklasie  
`if(object instanceof class){}`

## Klasy abstrakcyjne

- mogą zawierać metody abstrakcyjne, czyli takie, które nie posiadają implementacji (ani nawet nawiasów klamrowych)
- może zawierać stałe (zmienne oznaczone jako public static final)
- mogą zawierać zwykłe metody, które niosą jakąś funkcjonalność, a klasy rozszerzające mogą je bez problemu dziedziczyć
- klasy rozszerzające klasę abstrakcyjną muszą stworzyć implementację dla metod oznaczonych jako abstrakcyjne w klasie abstrakcyjnej
- metod abstrakcyjnych nie można oznaczać jako statyczne (nie posiadają implementacji)
- podobnie jak w przypadku interfejsów nie da się tworzyć instancji klas abstrakcyjnych

Dobrze, mamy zatem stworzoną naszą pierwszą klasę abstrakcyjną, ale co nam to właściwie daje?

Po pierwsze, napotykając taką klasę w kodzie od razu wiemy, że jest ona tylko i wyłącznie bazą do tworzenia innych klas. Mówimy, że jest ona pewnego rodzaju projektem, który następnie powinien zostać odpowiednio zaimplementowany w klasach podległych (dziedziczących).

Do stworzenia takiego projektu wymagane jest jednak coś więcej niż tylko słowo `abstract` użyte przed nazwą klasy. W tym celu potrzebujemy stworzyć przynajmniej jedną metodę abstrakcyjną.

Metody abstrakcyjne w Javie charakteryzują się tym, że:

- Oznaczamy je modyfikatorem `abstract`.
- Nie posiadają swojego ciała, a jedynie deklaracje w postaci nazwy metody wraz z listą parametrów (albo bez parametrów).

I teraz możemy na przykład stworzyć wymaganie, aby każda klasa dziedzicząca z klasą `Item` oprócz nazwy zwracała jeszcze opis. Wprowadzamy więc do klasy `Item` metodę abstrakcyjną o nazwie `getDescription()`.

```

public abstract class Item {
    String name;
    public void setName(String name) {

```

```
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public abstract String getDescription();
}
```

Metoda ta nie posiada ciała. Jej przeznaczeniem jest wymuszenie na każdej klasie implementującej, aby dostarczyła ciało tej metody. Inaczej mówiąc, teraz do naszej klasy `DocumentItem` musimy dodać metodę `getDescription` wraz z jej ciałem:

```
public class DocumentItem extends Item {

    public String getDescription() {
        return "This is description for DocumentItem";
    }
}
```

W ten sposób dostarczyliśmy implementację metody abstrakcyjnej. Warto podkreślić, że w przypadku braku tej metody, klasa `DocumentItem` nie skompiluje się. Tak to działa w przypadku klas **nieabstrakcyjnych**. W przypadku gdy jedna klasa **abstrakcyjna** dziedziczy z drugiej klasy **abstrakcyjnej**, nie musimy implementować w niej metod abstrakcyjnych. Kompilacja się powiedzie.

Modyfikatory dostępu:

private - w obrębie danej klasy  
public - widoczność wszędzie nawet poza pakietem  
private - widoczność w pakiecie i wszystkich podklasach  
domyślnie - widoczność w obrębie pakietu

```
jshell> var result1 = intValue1 + intValue2;  
result1 ==> 98
```

```
jshell> var result2 = intValue1 - intValue2;  
result2 ==> 14
```

```
jshell> var result3 = intValue1 * intValue2;  
result3 ==> 2352
```

```
jshell> var result4 = intValue1 / intValue2;  
result4 ==> 1
```

```
jshell> double doubleValue = -3.99999;  
doubleValue ==> -3.99999
```

```
jshell> long rounded = Math.round(doubleValue)  
rounded ==> -4
```

```
jshell> double absValue = Math.abs(doubleValue);  
absValue ==> 3.99999
```

### Konwersja liczb ze stringa

```
string tekst = "true";  
boolean zmienna;  
zmienna = boolean.parseBoolean(tekst);
```

### Zmiana tablicy charów na string i string na tablice char

```
jshell> char[] chars = {'h', 'e', 'l', 'l', 'o'};  
chars ==> char[5] { 'h', 'e', 'l', 'l', 'o' }
```

```
jshell> String s = new String(chars)  
s ==> "hello"
```

```
jshell> var charArray = s.toCharArray()  
charArray ==> char[5] { 'h', 'e', 'l', 'l', 'o' }
```

## Formatowanie wyświetlanego tekstu (automatyczne formatowanie wyświetlanych liczb)

```
var doubleValue = 10_000_000.53;

var numF :NumberFormat = NumberFormat.getNumberInstance();
System.out.println("Number: " + numF.format(doubleValue));

var intF :NumberFormat = NumberFormat.getIntegerInstance();
System.out.println("Number: " + intF.format(doubleValue));

intF.setGroupingUsed(false);
System.out.println("Number: " + intF.format(doubleValue));

var locale = new Locale( language: "de", country: "DE");
var localeFormatter :NumberFormat = NumberFormat.getNumberInstance(locale);
System.out.println("Number: " + localeFormatter.format(doubleValue));

var currencyFormatter :NumberFormat = NumberFormat.getCurrencyInstance();
System.out.println(currencyFormatter.format(doubleValue));

var df = new DecimalFormat( pattern: "$00.00");
System.out.println(df.format( number: 1));
Number: 10 000 000,53
Number: 10 000 001
Number: 10000001
Number: 10.000.000,53
10 000 000,53 zł
$01,00
```

Formatowanie tekstu w stringu ( automatyczne dodawanie zmiennych)

```
var item = "Shirt";
var size = "M";
var price = 14.99;
var color = "Red";

var template = "Clothing item: %s, size %s, color %s, $%.2f";
var itemString = String.format(template,
    item, size, color, price);
System.out.println(itemString);
```

Psvm- main method

## Zmienne

byte - 1 bajt - zakres od -128 do 127

short - 2 bajty - zakres od -32 768 do 32 767

int - 4 bajty - zakres od -2 147 483 648 do 2 147 483 647

long - 8 bajtów - zakres od -2<sup>63</sup> do (2<sup>63</sup>-1) (posiadają przyrostek L, lub l)

float - 4 bajty - max ok 6-7 liczb po przecinku (posiadają przyrostek F, lub f)

double - 8 bajtów - max ok 15 cyfr po przecinku (posiadają przyrostek D, lub d)

int - stałe

boolean - true false // boolean zmieniona= true; // standardowo bez określenia false

double - zmieniono przecinkowe

char- znak w cudzysłowie  
string- string składa się tak naprawdę z kolejnych charów //string zmienna=""cos";

## Operatorzy:

>,<,==,>=,<=,!=  
int zmienna = 10;  
zmienna += -= \*= /= 5; (15,5,50,2)  
&& - AND  
|| - OR  
?= - Ternary (shorthand if-then) // krótki if

```
var message = (i == 1)           // boolean test
    ? "There is 1"             // result if true
    : "There are " + i;        // result if false
System.out.println(message);
```

## Tworzenie tablicy

typzmiennej[] nazwazmiennej = new typzmiennej[rozmiar];  
typzmiennej[] nazwa zmiennej = {atrybut1,aatrybut2}  
konwersja tablicy do stringa

```
Arrays.toString(firstArr))
```

```
Arrays.copyOf(table, length)
```

// odowłuje się do tej samej referencji

<https://docs.oracle.com/javase/9/docs/api/java/util/doc-files/coll-overview.html>

## Tworzenie listy ( ArrayList ) Lista jest indeksowana

```
List<typzmienej> nazwazmiennej = new ArrayList<>();
nazwazmiennej.add(); // dodawanie do listy
nazwazmiennej.remove(); // usuwanie danych z listy
nazwazmiennej.contains(); // czy zawiera
SORTOWANIE
Collections.sort(nazwazmiennej_listy);
```

## Tworzenie listy HashSet ( Lista w której dane nie mogą się powtarzać ).

Indeksowanie nie jest gwarantowane może być zmienne

```
Set<typzmiennnej> nazwazmiennej = new HashSet<>();  
metody jak do arraylisty
```

## Tworzenie HashMap map<klucz , dane pobierane (wartość) >

```
Map<typzmiennnej,typzmiennnej> nazwazmiennej = new HashMap<>();  
nazwazmiennej.put(); // dodawanie danych do mapy  
nazwazmiennej.get(); // Pobieranie danych wskazując klucz  
nazwazmiennej.isEmpty(); // czy lista jest pusta  
nazwazmiennej.clear(); // czyszczenie listy  
nazwazmiennej.containsKey(cos); // Czy w mapie istnieje klucz o który pytamy  
nazwazmiennej.containsValue(cos); // Czy w mapie istnieje wartość o którą pytamy
```

### **Wyświetlanie HashMap**

```
nazwazmiennej.keySet(); // wyświetla wszystkie klucze mapy  
Set<typzmiennejmapy_klucza> nazwazmiennejnowej = nazwazmienemapy.keySet(); // rozwinięcie komendy wyżej (  
nazwazmiennej.values(); // wyświetla wszystkie wartości mapy  
Collections<typzmiennejmapy_wartości> nazwazmiennejnowej = nazwazmienemapy.keySet(); // rozwinięcie komendy wyżej (  
tworzenie listy set z kluczy i wartości)  
zmiennamap.forEach((x, y) -> System.out.println());
```

Wyświetlanie mapy

```
var keys = zmiennamap.keySet();  
for (var key : keys){  
    var item = items.get(key);  
    displayDetails(item);  
}
```

## Switch case:

```
// nowa od java 12 , nie trzeba brak kończy od razu w jednym case  
switch(wybor) {  
    case 1 -> sdaasd;  
    case 2 -> sadasd;  
}
```

```
//STARO WERSJA , jak nie ma break wykonuje wszystko z wyborem i poniżej jego  
switch(wybor) {  
    case 1:  
        ///tekst  
        break;  
    default:  
        //tekst  
}
```

## Przekształcenie zmiennej pobranej lub jakiejkolwiek

```
zmienna.toLowerCase() //na małe litery  
zmienna.toUpperCase();
```

```
zmienna.charAt(MIEJSCE_ZNAKU); // zwraca literę z podanego miejsca w stringu/zmiennej  
zmienna.indexOf("Treść") // zwraca miejsce w którym rozpoczyna się wskazany tekst  
zmienna.substring(INDEX) // Wpisując miejsce w stringu otrzymujemy wszystko co jest za nim włącznie z nim  
zmienna.trim() // Usuwa wszystkie spacje ze stringa  
zmienna.getBytes(); // zwraca tablice bitów liter ze zmiennej  
zmienna.append("xyz!"); // Dodanie tekstu do istniejącej zmiennej  
zmienna.ignoreCase(); //ignorowanie wielkości liter  
zmienna.length() // ilość znaków w stringu
```

## Pobieranie daty z urządzenia na którym uruchomiony jest program

```
import java.time.LocalDateTime // trzeba załączyć klasę  
  
var zmienna(NOW) = LocalDateTime.now(); // Zwraca godzine  
var zmienna = NOW.getMonthValue();
```

## IF

```
if (zmienna < 5){  
    // cos tam  
} else if( ) {      // else if ile razy chce  
    //cos tam  
} else if (){  
    //cos tam  
} else {    // tylko raz  
}
```

### **Shortcut for IF**

```
Warunek ? what_if_true : what_if_false;
```

## Tworzenie funkcji

**Main :** public static void main(String[] args){}

**Ogólnie:** public static void zmienna("TUTAJ MOZNA DEKLAROWAĆ ZMIENNE"){}  
}

**Odwołanie do niej nazwafunkcji(parametr1, parametr2, parametr3);**

Funkcja musi coś zwracać, w przypadku "public static void" zwraca void. Możemy też zadeklarować że zwróci zmienna. np "public static double" aby przekazać do niej dane piszemy : return naszazmiennazprogramu;

Możemy przypisać wynik działania funkcji do innej zmiennej w naszym "MAIN" np  
double zmienna = nazwafunkcji(parametr1, parametr2, parametr3);

## Pętle for /while/do/for each

### wypisywanie rzeczy z tablicy

```
public static void main(String[] args) {  
  
    String[] months =  
        {"January", "February", "March",  
         "April", "May", "June",  
         "July", "August", "September",  
         "October", "November", "December"};  
  
    for (int i = 0; i < months.length; i++) { //pętla for  
        System.out.println(months[i]);  
    }  
  
    for (var month : months) { //pętla foreach  
        System.out.println(month);  
    }  
  
    var whileCounter = 0;           // pętla while  
    while (whileCounter < months.length) {  
        System.out.println(months[whileCounter]);  
        whileCounter++;  
    }  
  
    var doCounter = 0;             //pętla do while  
    do {  
        System.out.println(months[doCounter]);  
        doCounter++;  
    } while (doCounter < months.length);  
}
```

## Pobieranie z konsoli

```
import java.util.Scanner; // Scan input data from terminal  
  
Scanner namevar = new Scanner(System.in);  
namevar.inputnext(); // wait for input from terminal
```

Można też stworzyć zmienna od razu z pobieraniem danych  
`string zmienna=input.next();`

## KLASY

```
import java.util.ArrayList;
import java.util.List;

public class Party {           } //create class
    private List<String> guests = new ArrayList<>(); //create standart constructor

    public void addGuest(String name) {           //create metod add
        guests.add(name);
    }

    public List<String> getGuests() {           //create metod return
        return guests;
    }
} //metod need be public
```

Tworzenie nowego obiektu

```
Party nazwazmiennej = new Party(); //create object
new Party(); //shortcut
```

```
public class Triangle {
    double base;           //atrybuty
    double height;
    double sideLenOne;
    double sideLenTwo;
    double sideLenThree;

    public Triangle(double base, double height, //create own constructor
                   double sideLenOne, double sideLenTwo,
                   double sideLenThree) {
        this.base = base;
        this.height = height;           // this przypisuje zmienna do zmiennej w klasie
        this.sideLenOne = sideLenOne;
        this.sideLenTwo = sideLenTwo;
        this.sideLenThree = sideLenThree;
    }

    public double findArea() {           //metod for class
        return (this.base * this.height) / 2;
    }
}
```

Tworzenie nowego trójkąta plik main.class

```
public class Main {

    public static void main(String[] args) {
        Triangle triangleA=new Triangle(15,8,15,8,17)
            //obliczenie pola trójkąta za pomocą funkcji w triangle.class
        double triangleA = triangleA.findArea();
        System.out.println(triangleA);
    }
}
```

## Enum

```
new class ===> enum  
public enum size {  
    Big, Large, Medium  
}  
size.big    (String)
```

Wywołanie zwraca to samo co mamy określone w enum

```
public enum size {  
    B("Big"), L("large"), M("Medium");  
    private String description;  
    size(String description) {  
        this.description = description;  
    }  
  
    @Override  
    public String toString() {  
        return description;  
    }  
}
```

size.b

Wywołanie zwraca to co określone w description

## Formatowanie String

```
String format(Your %s %s order will cost %s,  
    item.getSize(),  
    item.getTyper(),  
    item.totalPrice());
```

Exception try catch

```
try {  
    //code  
}catch (Exception e){  
    sout  
}catch (otherexception e){  
    sout  
}
```

Non- Static method for class must be appeal for the object, but static don't need

## Kopiowanie plików od javy 7:

**try{}catch{}**

**try(tutaj automatycznie zamyka pliki/źródła)**

```
public static void main(String[] args) {
    String sourceFile = "pliki/dokument.txt";
    String targetFile = "pliki/targer.txt";

    try (FileReader oldFile = new FileReader(sourceFile);
        BufferedReader newBuffer = new BufferedReader(oldFile);
        FileWriter newFile = new FileWriter(targetFile);)
    {
        while(true){
            String newline = newBuffer.readLine();
            if(newline == null){
                break;
            }
            newFile.write(newline + "\n");
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

}

Nowa wersja kopiowania

```
public static void main(String[] args) {
    // write your code here

    Path sourceFile = Paths.get("files", "lorem ipsum.txt");
    Path targetFile = Paths.get("files", "target.txt");

    try {
        Files.copy(sourceFile, targetFile, StandardCopyOption.REPLACE_EXISTING);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

## Wczytywanie pliku json

```
public static void main(String[] args) throws IOException {
    // write your code here
    String fileName = "files/data.json";

    Gson gson = new Gson();
    try (FileReader fileReader = new FileReader(fileName);
        JsonReader reader = new JsonReader(fileReader))
    {
        Flower[] data = gson.fromJson(reader, Flower[].class);
        for (Flower flower :
```

```

        data) {
    System.out.println(flower);
}
}

}

```

## Metody generyczne

```

public class GenericMethods {

    static Character[] charArray = {'h', 'e', 'l', 'l', 'o'};
    static Integer[] intArray = {1, 2, 3, 4, 5};
    static Boolean[] boolArray = {true, false, true};

    public static <T> List<T> arrayToList(T[] array, List<T> list) {
        for (T object : array) {
            list.add(object);
        }
        return list;
    }

    public static void main(String[] args) {
        List<Character> charList = arrayToList(charArray, new ArrayList<>());
        List<Boolean> boolList = arrayToList(boolArray, new ArrayList<>());
        List<Integer> intList = arrayToList(intArray, new ArrayList<>());
        System.out.println(intList.get(0));
    }
}

```

## Varrags czyli skalowalna metoda.

Trzy kropki po typie zmiennej będzie automatycznie tworzyć tablice z podanych danych

```

public class Varargs {

    public static void main(String[] args) {
        String item1 = "Apples";
        String item2 = "Oranges";
        String item3 = "Pears";
        printShoppingList(item1, item2, item3);
        printShoppingList("Bread", "Milk", "Eggs", "Bananas");
    }
}

```

```

    private static void printShoppingList(String... items) {
        System.out.println("SHOPPING LIST");
        for (int i = 0; i < items.length; i++) {
            System.out.println(i + 1 + ": " + items[i]);
        }
        System.out.println();
    }
}

```

**Dzikie karty - Metody które są elastycznie pod kątem obiektów które rozszerzają dane klasy**

```

public static void main(String[] args) {

    // List of buildings
    List<Building> buildings = new ArrayList();
    buildings.add(new Building());
    buildings.add(new Building());
    printBuildings(buildings);

    // List of offices
    List<Office> offices = new ArrayList();
    offices.add(new Office());
    offices.add(new Office());
    printBuildings(offices);

    // List of houses
    List<House> houses = new ArrayList();
    houses.add(new House());
    houses.add(new House());
    printBuildings(houses);

    addHouseToList(houses);
    addHouseToList(buildings);
}

}

```

**Używamy "<? extends Building>" gdy chcemy używać metody do klas rozszerzających klase bulding**

```

static void printBuildings(List<? extends Building> buildings) {
    for(int i = 0; i < buildings.size(); i++) {
        System.out.println(buildings.get(i).toString() + " " + (i + 1));
    }
    System.out.println();
}

```

**Używamy "<? super House>" gdy chcemy używać metody do nadklasty**

```

static void addHouseToList(List<? super House> buildings) {
    buildings.add(new House());
    System.out.println();
}

```

```

}

```

## Interfejs funkcyjny

Posiada tylko jedną metodę abstrakcyjną nie licząc sie te zaimplementowane w klasie object  
rodzaje interfejsów funkcyjnych

*Function<T, R>* zawiera metodę *apply*, która przyjmuje instancję klasy T  
zwracającą instancję klasy R,

*Consumer<T>* zawiera metodę *accept*, która przyjmuje instancję klasy T,  
*Predicate<T>* zawiera metodę *test*, która przyjmuje instancję klasy T i  
zwraca flagę. Interfejs ten może posłużyć do zastąpienia interfejsu

*Checker*,

*Supplier<T>* zawiera metodę *get*, która nie przyjmuje żadnych  
parametrów i zwraca instancję klasy T,

*UnaryOperator<T>* jest specyficznym przypadkiem interfejsu *Function*. W  
tym przypadku typ argumentu i typ zwracany są te same.

*BinaryOperator<T>* przyjmuje i zwraca ten sam typ argumentu

1

```
public class main {  
  
    protected static class MyMath{  
        public static Integer triple(Integer x){  
            return x*3;  
        }  
    }  
    public static void main(String[] args) {  
// <przyjmuje,zwraca>  
        Function<Integer, Integer> newfunction = MyMath::triple;  
        System.out.println(newfunction.apply(5));  
        Function<Integer, Boolean> newtest = x -> x < 0 ? true : false;  
        System.out.println(newtest.apply(6));  
  
    }  
}
```

### Multi functional interface

```
public static void main(String[] args) {  
    BiFunction<Integer, Integer, Boolean> test = (x,y) -> x+y < 100 ? true : false;
```

```
BiFunction<Integer, Integer, Integer> test2 = (x, y) -> x+y;
System.out.println(test.apply(5, 100));
System.out.println(test2.apply(5, 5));
}
```

---

```
public interface trifunction<T, U, V, R> {
    R apply(T t, U u, V v);
}
public interface nonfunction <R>{
    R apply();
}

public static void main(String[] args) {
    trifunction<Integer, Integer, Integer, Integer> test = (x, y, z) -> x*y+z;
    System.out.println(test.apply(5, 6, 1));
    nonfunction<String> test2 = () -> "Siema nic nie przekazujemy";
    System.out.println(test2.apply());
}
```

## 2

```
@FunctionalInterface // Dzieki temu zapisowi intelij sprawdza czy to interfejs
funkcyjny
public interface GreetingMessage {

    public abstract void greet(String name);
}

public class Main {
    public static void main(String[] args) {
        GreetingMessage gm = new GreetingMessage() {
            @Override
                //nalezy nadpisac metode z intefesju
            public void greet(String name) {
                System.out.println("Hello " + name);
            }
        };
        gm.greet("Bethan");
    }
}
```

## Wyrażenia lambda i referencja metody

```
public static void main(String[] args) {
    //nowy obiekt
    Square s = new Square(4);
    //referencja metody
    Shapes zmienna3 = Square::calculateArea;
    //wyrażenie lambda
    Shapes zmienna2 = (Square zmienna) ->{
        return zmienna.calculateArea();
    };
    System.out.println("Area: " + zmienna2.getArea(s));
    System.out.println("Area: " + zmienna3.getArea(s));
}
```

# STREAM

**zmiennatypulist.stream()**

**.map()** // w nawiasach dajemy odwołanie do np do interfejsu **typu function** w którym coś wykonujemy na obiektach lub metody getter setter itp a później je zwracamy zmodyfikowane (nie te same !) można zwrócić cokolwiek np same litery liczbe itp

**.filter((word) -> word.length() > 5)** .filter() // w nawiasach dajemy odwołanie do np do interfejsu **typu predicate** w którym przedstawiamy nasze warunki (lambda)i zwracamy true lub false

**.reduce()** // w nawiasach dajemy odwołanie do np do interfejsu **typu bifunction** który np dodaje w swojej funkcji obiekty z listy np acc(ma swój akumulator) + b(przekazany obiekt z listy).

zwraca to na końcu jeden obiekt tego samego typu co wejściowy

**.sorted((x,y) -> x.compareTo(y))** // Zwraca posortowany stream

**.sorted()**

**.collect(Collectors.function())**

**.collect(Collectors.joining(" "))** // dodaje wszystkie obiekty z listy oddzielając je podanym delimitерem  
zwraca obiekt a nie liste

**.collect(Collectors.counting())** // zwraca ilość obiektów w stremie

**.collect(Collectors.groupingBy(** // zwraca Map'e w której kluczem jest

**(word) -> word.length()** // nasz podany warunek/funkcja a

**)** // a wartością lista obiektów tego typu co stream

```

.collect(Collectors.partitioningBy( // zwraca Map'ę w której kluczem jest boolean
    (word) -> word.length() > 5 // true jeśli spełnia warunek jeśli nie false
)
    // a wartością lista obiektów tego typu co stream zgadzająca się z warunkiem lub nie
.collect(Collectors.toMap())
.foreach(System.out::println
.limit
.mapToInt(x -> x)
.min()
.max()
.summarizeStatistics();

```

```

ArrayList<Book> books = populateLibrary();
books.stream().filter(book -> {
    return book.getAuthor().startsWith("J");
}).filter(book -> {
    return book.getTitle().startsWith("E");
}).forEach(System.out::println);

System.out.println("");

```

```

// to samo tylko wolniej
for (Book e: books)
    if(e.getAuthor().startsWith("J") && e.getTitle().startsWith("E"))
        System.out.println(e);
}
}

```

## Stream wielowątkowy

```

ArrayList<Book> books = populateLibrary();
books.parallelStream().filter(book -> {
    return book.getAuthor().startsWith("J");
}).filter(book -> {
    return book.getTitle().startsWith("E");
}).forEach(System.out::println);

```

## Combine stream

```

public static void main(String[] args) {
    Employee[] employeesArr = {
        new Employee("John", 34, "developer", 80000f),
        new Employee("Hannah", 24, "developer", 95000f),
        new Employee("Bart", 50, "sales executive", 100000f),
    }
}

```

```

        new Employee("Sophie", 49, "construction worker", 40000f),
        new Employee("Darren", 38, "writer", 50000f),
        new Employee("Nancy", 29, "developer", 75000f),
    } ;
List<Employee> employees = new ArrayList<>(Arrays.asList(employeesArr)) ;

Map<String,Float> averagemap = employees
    .stream()
    .collect(Collectors.groupingBy( x -> x.jobTitle))
    .entrySet()
    .stream()
    .collect(Collectors.toMap(
        x -> x.getKey(),
        x -> x.getValue()
            .stream()
            .map(y -> y.salary)
            .reduce( 0f, (acc,u) -> acc+u) / x.getValue().size()
    )) ;

System.out.println(averagemap);
}

```

## Kopiowanie

```

Path source = Paths.get("C:\\\\Users\\\\Bethan Palmer\\\\Desktop\\\\example.txt");
Path dest = Paths.get("C:\\\\Users\\\\Bethan Palmer\\\\Desktop\\\\new.txt");
try {
    Files.copy(source, dest, REPLACE_EXISTING); // replace_existing is optional
} catch (IOException ex) {
    ex.printStackTrace();
}

```

## Rekurencja recursion

```

public class Chapter4Video3 {
    static void countDown(Integer x) {
        if (x < 0) {
            System.out.println("Done!");
            return;
        }
        System.out.println(x);
        countDown(x - 1);
    }

    static void countUp(Integer x, Integer end) {

```

```

        if (x > end) {
            System.out.println("Done!");
            return;
        }
        System.out.println(x);
        countUp(x + 1, end); // woła sama siebie
    }

    public static void main(String[] args) {
        countUp(0, 10);
    }
}

```

// działa jak pętla w funkcji

## Łączanie dwóch funkcji Compose Function

```

Function<Integer, Integer> timestwo = x -> x*2
Function< Integer, Integer> minusone = x -? x-1
                                // to drugie robi sie pierwsze
Function <Integer,Integer> combine = minusone.compose(timestwo);

                                // to pierwsze robi sie pierwsze
Function <Integer,Integer> combine = minusone.andThen(timestwo);

```

```

Function<Employee, String> getName = employee -> employee.name;
Function<String, String> reverse = str -> new
StringBuilder(str).reverse().toString();
Function<String, String> uppercase = str -> str.toUpperCase();

Function<Employee, String> getReversedUppercasedName =
getName.andThen(reverse).andThen(uppercase);

```

```

List<String> results = employees
    .stream()
    .map(getReversedUppercasedName)
    .collect(Collectors.toList());
System.out.println(results);

```

## Optional

**Jest od javy 8 daje możliwość opakowania zmiennej w interfejs który może chronić przed wyrzuceniem błęd null exception**

```
Optional<Book> book =...
```

przykładowe metody do optional to:

```

book.ifPresent(System.out::println) // jeżeli nie null

book.ifPresentOrElse( System.out::println, () ->
System.out.println("book doesn't exist"));
    //jeżeli nie ma wyrzuć błąd
Book book1 = book.orElseThrow( () -> new RuntimeException("book doesn't exist"));
    //jeżeli nie ma podaj to co w lambda
book.orElseGet( () -> new Book("Spring w akcji", "9782123226803"));

```

## Sortowanie Comparable

*Returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.*

*Throws `ClassCastException` if the specified object's type prevents it from being compared to this object.*

*x.compareTo(y)*

*obiekty muszą implementować interfejs comparable implements comparable<class>*  
*powinny nadpisywać metodę compareTo*

*@Override*

```

public int compareTo(Class x){
return this.zmienna.compareTo(x.zmienna);
}

```

*Comparable:*

```

List<String> strings = Arrays.asList("Foo", "Bar", "Andy", "Shop");
Collections.max(strings); //Shop
Collections.min(strings); //Andy

```

```
Collections.sort(words); //Collection for binary search needs to be sorted
// [Andy, Bar, Foo, Shop]
Collections.binarySearch(words, "Foo"); //2
można użyć też do automatycznego sortowania treelist
Set<String> alphabeticalWordList = new TreeSet<>();
```

Comparator:

```
private static final Comparator<PhoneNumber> COMPARATOR =
    comparingInt((PhoneNumber pn) -> pn.areaCode)
        .thenComparingInt(pn -> pn.prefix)
        .thenComparingInt(pn -> pn.lineNum);

public int compareTo(PhoneNumber pn) {
    return COMPARATOR.compare(this, pn);
}
```

Sortowanie stream:

```
ranking.stream()
.sorted(Comparator.comparing(RankingEntry::getScore))
//...
```

```
ranking.stream()
.sorted(Comparator.comparing(RankingEntry::getScore).reversed())
//...
```

```
public class Order {
    public static String order(String words) {
        return Arrays.stream(words.split(" "))
            .sorted(Comparator.comparing(s -> Integer.valueOf(s.replaceAll("\\D", ""))))
            .reduce((a, b) -> a + " " + b).get();
    }
}
```

## Zapisywanie danych do pliku

```
void createFile() throws FileNotFoundException {
    File file = new File("orderBackup.txt");
    FileOutputStream fileOutputStream = new FileOutputStream(file);
    OutputStreamWriter outputStreamWriter = new OutputStreamWriter(fileOutputStream);
    writer = new BufferedWriter(outputStreamWriter); // do tego dodajemy naszego
stirnga
}
//dopisywanie danych do pliku
void backupOrder(Order order) throws IOException {
    writer.append(order.toString()); // dodawanie danych do pliku
}

//zamykanie pliku
void closeFile() throws IOException {
    writer.close();
}
```

## String replaceall

[https://regexone.com/lesson/repeating\\_characters?](https://regexone.com/lesson/repeating_characters?)

```

hello.replaceAll("[^\\d]", "");
[^0-9]abc... Letters
123... Digits
\d Any Digit
\D Any Non-digit character
. Any Character
\.
Period
[abc] Only a, b, or c
[^abc] Not a, b, nor c
[a-z] Characters a to z
[0-9] Numbers 0 to 9
\w Any Alphanumeric character
\W Any Non-alphanumeric character
{m} m Repetitions
{m,n} m to n Repetitions
* Zero or more repetitions
+ One or more repetitions
? Optional character
\s Any Whitespace
\S Any Non-whitespace character
^...$ Starts and ends
(...) Capture Group
(a(bc)) Capture Sub-group
(.*) Capture all
(abc|def) Matches abc or def

```

## TESTOWANIE JUNIT 5

### pom.xml

#### Kalibracja junit

- junit.jupiter muszą mieć te same wersje
- zależności dodajemy w dependency
- \${zmienna} - odwołanie do zmiennej
- <zmienna></zmiennna> - tworzenie zmiennej w properties

```

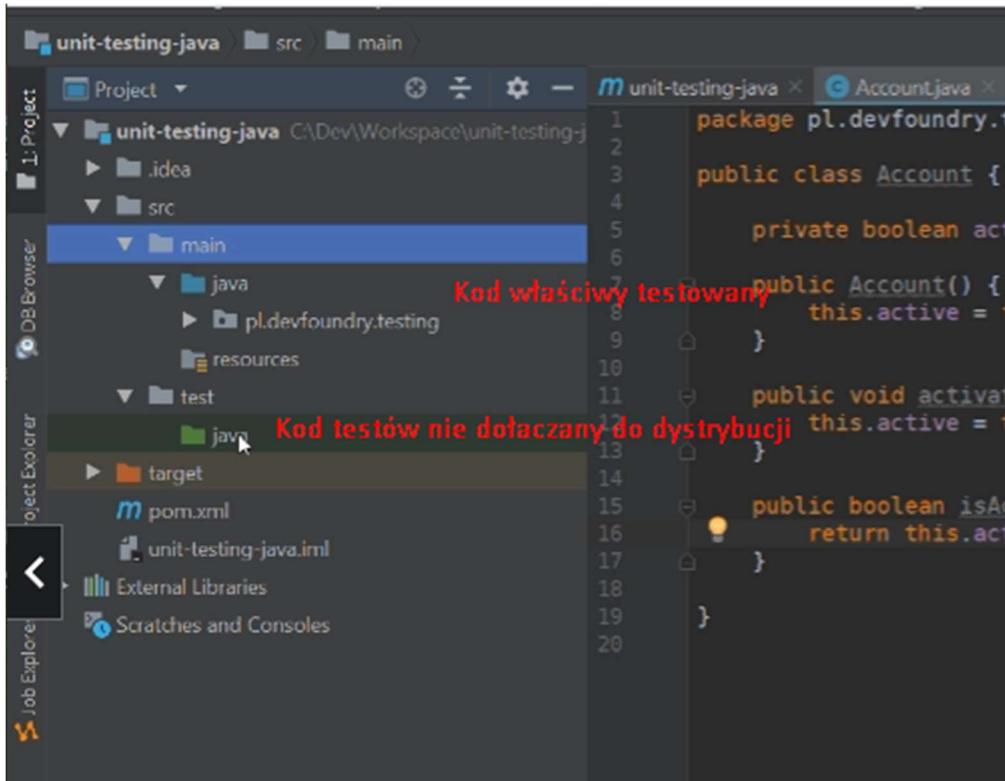
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <configuration>
                <source>11</source>
                <target>11</target>
            </configuration>
        </plugin>
        <plugin>

```

```
<artifactId>maven-surefire-plugin</artifactId>
  <version>2.22.0</version>
</plugin>
</plugins>
</build>
//dodanie pluginów testujących
<dependencies>
  <dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-engine</artifactId>
    <scope>test</scope>
    <version>${junit.version}</version>
  </dependency>
  <dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-api</artifactId>
    <scope>test</scope>
    <version>${junit.version}</version>
  </dependency>
  <dependency>
    <groupId>org.junit.platform</groupId>
    <artifactId>junit-platform-launcher</artifactId>
    <scope>test</scope>
    <version>1.0.0</version>
  </dependency>
  <dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter</artifactId>
    <version>RELEASE</version>
    <scope>test</scope>
  </dependency>
</dependencies>
<properties>
  <junit.version>5.0.0</junit.version>
  <maven.compiler.source>11</maven.compiler.source>
  <maven.compiler.target>11</maven.compiler.target>
</properties>

</project>
</project>
```

## Maven/testowanie



**Tworzenie klasy testowej** - tworzymy ją w folderze test/java/package nazwa klasy powinna byc to NazwaklasytestowanejTest.java

Można klikając przy klasie ctrl + shift + t

**Prosty test: ( jedna metoda jeden test) (nazwy testów tworzymy opisując czynnosci które robimy)**

```
package pl.rafał.testing;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertFalse;
class AccountTest {

    @Test
    void newAccountShouldNotBeActive() {
        //given
        Account newAccount = new Account();
        //then
        assertFalse(newAccount.isActive());
    }
}
```

```
@Test
void newAccountShouldBeActiveAfterActivation() {
    //given
    Account newAccount = new Account();
    assertFalse(newAccount.isActive());
    //when
    newAccount.activate();
    //then
    assertTrue(newAccount.isActive());
}
```

## Asercje:

```
assertFalse(newAccount.isActive()); // sprawdzanie czy false
assertTrue(newAccount.isActive()); // sprawdzanie czy true
assertEquals(SchouldBe, Actual); //powinno być
assertSame(obiekt, obiekt); //porównanie referencji obiektów
assertNotSame(obiekt, obiekt); //porównanie referencji obiektów
assertEquals(meal1, meal2); // do porównania obiektów w klasie trzeba
wygenerować metody equals i hashCode za pomocą generatora
assertNull(zmienna); // czy null
assertNotNull(zmienna); // czy nie null
```

### Test czy występuje wyjątek

```
assertThrows(TypWyjątu.class, () -> klasa.metoda());
```

### Test na timeout czasowy

```
assertTimeout(Duration.ofMillis(100), () -> klasa.metoda());
```

### Kilka aseracji

```
assertAll(
    () -> assertThat(cart.getOrders(), notNullValue())
    // () ->
    // () ->
);

```

### Warunkowa aseracja

```
assumingThat(address != null, //warunek
    () -> assertTrue(account.isActive()) //asseracja
);
```

## MATCHERY

## Hamcrest

Dodajemy dependencje do pom.xml z maven.org org.hamcrest

```
assertThat(newAccount.isActive(), equalTo(false));
assertThat(newAccount.isActive(), is(false));
assertThat(address, nullValue());
```

```
assertThat(address, notNullValue());
assertThat(meal1, not(sameInstance(meal2))); //referencje
```

<http://hamcrest.org/JavaHamcrest/javadoc/2.2/>

## Kolekcje

```
assertThat(order.getMeals(), empty());           //czy pusta
assertThat(order.getMeals().size(), equalTo(0));   //czy pusta
assertThat(order.getMeals(), hasSize(0));          //czy pusta
assertThat(order.getMeals(), emptyCollectionOf(Meal.class)); //czy pusta
```

```
assertThat(order.getMeals(), hasSize(1));
assertThat(order.getMeals(), contains(obiekt));
assertThat(order.getMeals(), hasItem(obiekt));
```

## Sprawdzanie poprawnej kolejności

```
assertThat(order.getMeals(), contains(meal1, meal2));
```

## Bez kolejności

```
assertThat(order.getMeals(), containsInAnyOrder(meal1, meal2));
```

## Czy listy są takie same:

```
assertThat(meals1, is(meals2));
```

## Kilka asercji do jednego assertThat wystarczy jedna spełniona anyOf()

```
assertThat(cart.getOrders(), anyOf(
    notNullValue(),
    hasSize(1),
    is(not(empty())),
    is(not(emptyCollectionOf(Order.class)))
));
```

## Kilka asercji do jednego assertThat wszystkie muszą być spełnione allOf()

```
assertThat(cart.getOrders(), allOf(
    notNullValue(),
    hasSize(1),
    is(not(empty())),
    is(not(emptyCollectionOf(Order.class)))
));
```

## Z junit

```
assertAll(
    () -> assertThat(cart.getOrders(), notNullValue())
    // () ->
    // () ->
);
```

# AssertJ

## Dodajemy dependencje do pom.xml z maven.org org.assertj-core

```
assertThat(newAccount.isActive()).isFalse();
assertThat(newAccount.isActive()).isTrue();
assertThat(address).isNull();
assertThat(discountedPrice).isEqualTo(25);
assertThat(meal1).isSame(meal2); //referencje
assertThat(meal1).isEqualTo(meal2); //rownosc obiektow
```

<http://joel-costigliola.github.io/assertj/core-8/api/index.html>

## Globalne zmienne Junit5 (setup, tearDown)

```
@Disabled // wyłączenie testu  
@DisplayName(„”) // wyświetlanie nazwy podczas uruchamiania testu  
@RepeatedTest(5) // powtórzenie testu ilość razy
```

```
private Order order;  
@BeforeEach // przed każdym testem setup  
void initializeOrder(){  
    order = new Order();  
}  
  
@AfterEach // po każdym tescie tearDown  
void cleanUp(){  
    order.cancel();  
}
```

**@BeforeAll oraz @AfterAll musi być zawsze static i mieć obiekty static**

```
private static OrderBackup orderBackup;  
  
@BeforeAll // przed pierwszym  
static void setup() throws FileNotFoundException {  
    orderBackup = new OrderBackup();  
    orderBackup.createFile();  
}  
  
@AfterAll // na koniec  
static void tearDown() throws IOException {  
    orderBackup.closeFile();  
}
```

## Testy Parametryzowane

Należy dodać dependency junit-jupiter-params

```
@ParameterizedTest  
@ValueSource(ints = {5, 10, 15}) // Tu podajemy zmienne do testu  
void mealPricesShouldBeLowerThan20(int price){  
    assertThat(price, lessThan(20));  
}
```

Źródłem może być klasa enum

```
@ParameterizedTest  
@EnumSource(OrderStatus.class)  
void allOrderStatusShouldBeShorterThan15Characters(OrderStatus status){  
    assertThat(status.toString().length(), lessThan(15));  
}
```

## Źródłem może być metoda / stream

```
@ParameterizedTest
@MethodSource("createMealsWithNameAndPrice")
void burgerShouldHaveCorrectNameAndPrice(String name, int price) {
    assertThat(name, containsString("burger"));
    assertThat(price, greaterThanOrEqualTo(10));
}

private static Stream<Arguments> createMealsWithNameAndPrice() {
    return Stream.of(
        Arguments.of("Hamburger", 10),
        Arguments.of("Cheesburger", 12)
    );
}
```

```
@ParameterizedTest
@MethodSource("createCakeNames")
void cakeNamesShouldEndWithCake(String name) {
    assertThat(name, notNullValue());
    assertThat(name, endsWith("cake"));
}

private static Stream<String> createCakeNames() {
    List<String> cakeNames = Arrays.asList("Cheesecake", "Fruitcake", "Cupcake");
    return cakeNames.stream();
}
```

## Wczytywanie z pliku .csv

```
@ParameterizedTest
@CsvSource({ "Fabryczna,10", "Armia,15", "'Romka,Atomka',16" })
void givenAddressesShouldNotBeEmptyAndHavePropperNames(String street, String number) {
    assertThat(street, notNullValue());
    assertThat(street, containsString("a"));
    assertThat(number, notNullValue());
    assertThat(number.length(), lessThan(8));
}

@ParameterizedTest
@CsvFileSource(resources = "/addresses.csv")
void givenAddressesFromCsvFileShouldNotBeEmptyAndHavePropperNames(String street, String number) {
    assertThat(street, notNullValue());
    assertThat(street, containsString("a"));
    assertThat(number, notNullValue());
    assertThat(number.length(), lessThan(8));
}
```

## Extension Model

Kolejność wykonywania

```
BeforeAllCallback (1)
  @BeforeAll (2)
    BeforeEachCallback (3)
      @BeforeEach (4)
        BeforeTestExecutionCallback (5)
          @Test (6)
            TestExecutionExceptionHandler (7)
          AfterTestExecutionCallback (8)
        @AfterEach (9)
      AfterEachCallback (10)
    @AfterAll (11)
AfterAllCallback (12)
```

Lifecycle Callbacks (@ExtendWith(Extension))

User code: methods of the test class

Aby dodać extensionmodel trzeba zrobić dodatkową klasę implementującą interfejs/sy

```
import org.junit.jupiter.api.extension.AfterEachCallback;
import org.junit.jupiter.api.extension.BeforeEachCallback;
import org.junit.jupiter.api.extension.ExtensionContext;

public class BeforeAfterExtension implements BeforeEachCallback, AfterEachCallback {
    //generate
    @Override
    public void afterEach(ExtensionContext extensionContext) throws Exception {
        System.out.println("Inside before each extension"); //przykład nasza metoda
    }

    @Override
    public void beforeEach(ExtensionContext extensionContext) throws Exception {
        System.out.println("Inside after each extension"); //przykład nasza metoda
    }
}
```

Oraz w klasach testowych dodać przed klasą lub przed konkretnym testem

```
@ExtendWith(BeforeAfterExtension.class)
```

## Przechwytywanie wyjątków

Klasa przechwytyjąca

```
import org.junit.jupiter.api.extension.ExtensionContext;
import org.junit.jupiter.api.extension.TestExecutionExceptionHandler;

import java.util.logging.Logger;

public class IAEceptionIgnoreExtension implements TestExecutionExceptionHandler {

    private static final Logger LOGGER =
Logger.getLogger(IAEceptionIgnoreExtension.class.getName()); // inicjujemy logger
    @Override
    public void handleTestExecutionException(ExtensionContext extensionContext,
Throwable throwable) throws Throwable {
        if(throwable instanceof IllegalArgumentException){
            LOGGER.info("Just ignored IllegalArgumentException!");
        }else{
            throw throwable;
        }
    }
}
```

przed testem który może wyrzucić błąd dodać dopisek

```
@ExtendWith(IAEceptionIgnoreExtension.class)
```

## Testy dynamiczne

```
@TestFactory
Collection<DynamicTest> dynamicTestCollection() {
    return Arrays.asList(
        dynamicTest("Dynamic test 1", () -> assertThat(5, lessThan(6))),
        dynamicTest("Dynamic test 2", () -> assertEquals(4, 2*2))
    );
}

@TestFactory
Collection<DynamicTest> calculateMealPrices() {
    //dodajemy posiłki
    order.addMealToOrder(new Meal(10, 5, "Bulka"));
    order.addMealToOrder(new Meal(15, 2, "zupa"));
    order.addMealToOrder(new Meal(20, 1, "chleb"));
    Collection<DynamicTest> dynamicTests = new ArrayList<>();
    //petla wielkosci zamówienia
    for (int i = 0; i < order.getMeals().size(); i++) {
        int price = order.getMeals().get(i).getPrice();
        int quantity = order.getMeals().get(i).getQuantity();
        //executable z junit.jupiter.api.function
        Executable executable = () -> assertThat(calculatePrice(price, quantity),
lessThan(100));

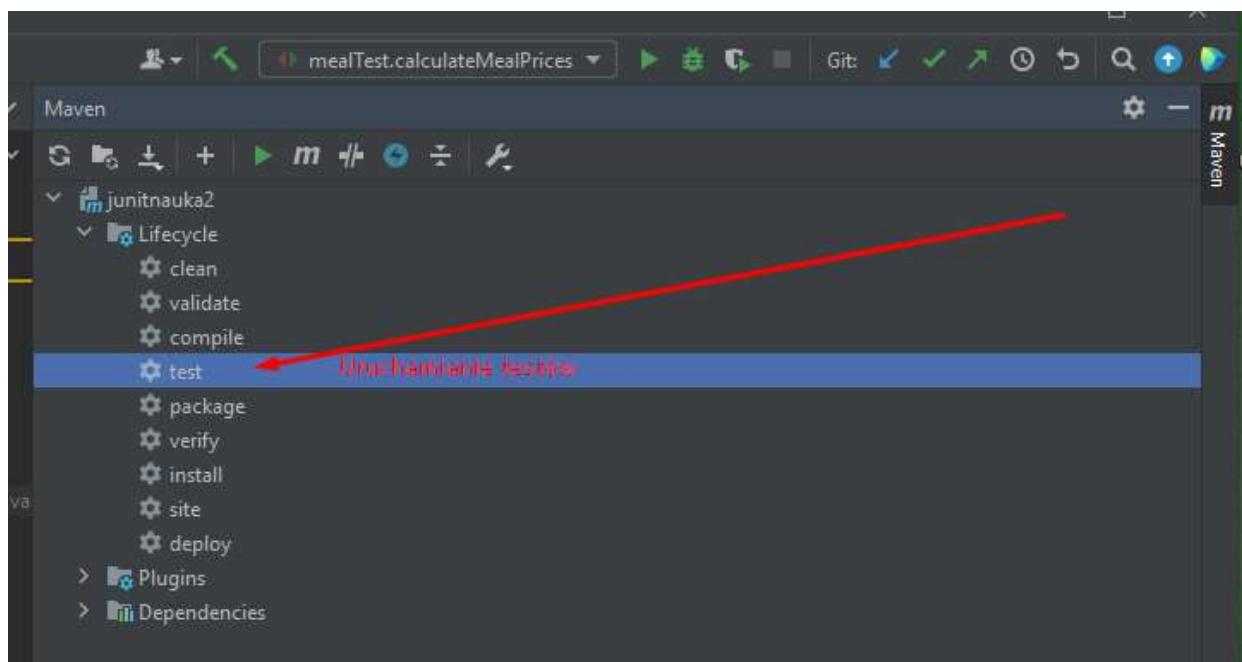
        String name = "Test name:" + i;
        //Test dynamiczny składa się z nazwy i działania
        DynamicTest dynamicTest = DynamicTest.dynamicTest(name, executable);
        dynamicTests.add(dynamicTest);
    }
    return dynamicTests;
}

private int calculatePrice(int price, int quantity) {
    return price*quantity;
}
```

## Uruchamianie testów

Konsola i – mvn test

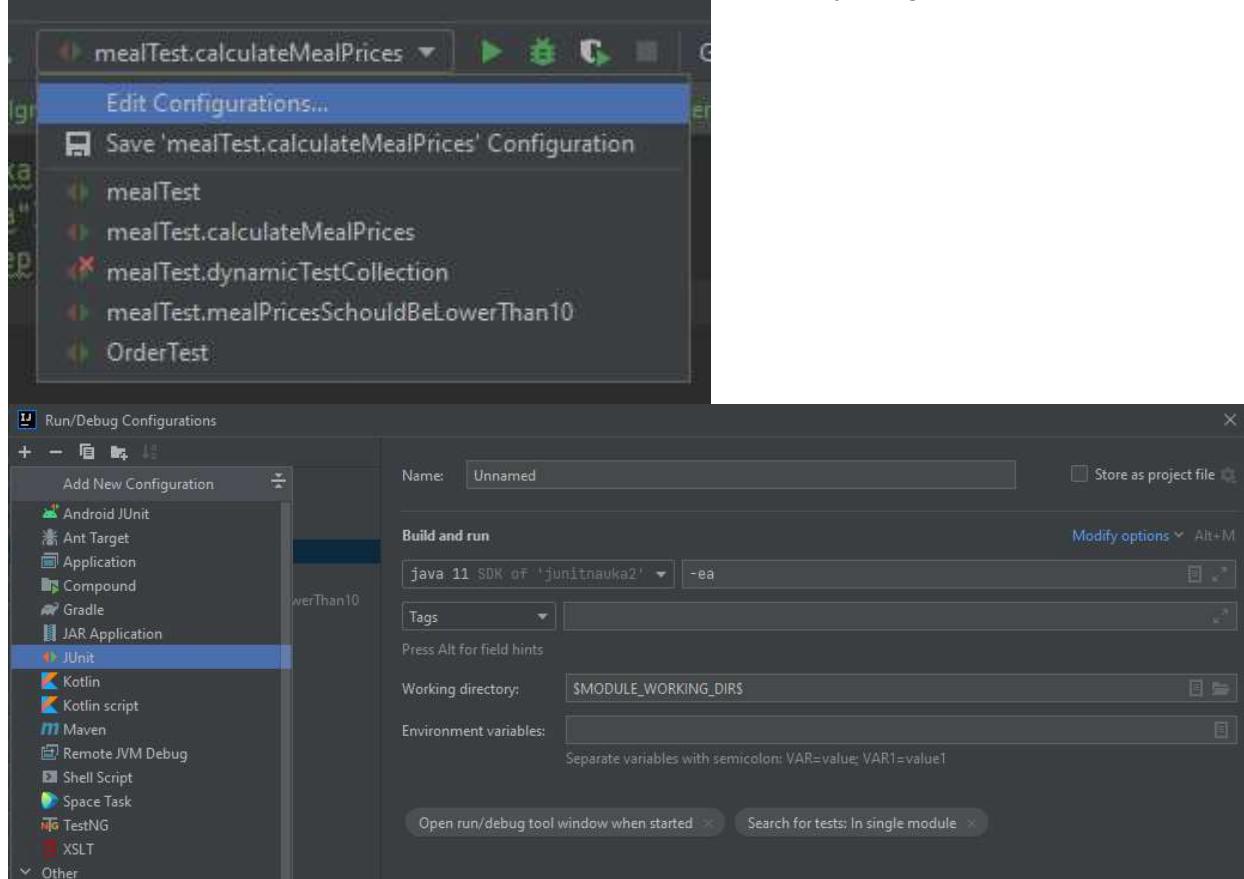
Lub



## Uruchamianie tylko wybranych testów (filtrowanie) @Tag

@Tag(“”) // można uruchamiać testy lub metody tylko z tagiem

Trzeba edytować konfiguracje i dodać uruchamianie tylko z danym tagiem



## Zasady testów jednostkowych FIRST

- **Fast** – szybkie testy po kilka ms
- **Isolated** – testowanie małych metod, odizolowanie od innych testów
- **Repeatable** – powtarzalność nie może być losowości
- **Self-validating** – automatyzacja- uruchamianie testów powinno być automatyczne
- **Timely** - Tworzenie testów podczas tworzenia lub zmiany danej funkcjonalności

## Zasady testów jednostkowych CORRECT

- **Conformance** – zgodność z wymaganiami biznesowymi ( sprawdzanie poprawności wprowadzanych danych)

```
public void setEmail(String email) {  
  
    if(email.matches( regex "[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\\.[A-Za-z]{2,6}$" )) {  
        this.email = email;  
    } else {  
        throw new IllegalArgumentException("Wrong email format");  
    }  
  
}  
  
@Test  
void invalidEmailShouldThrowException() {  
  
    //given  
    Account account = new Account();  
  
    //when  
    //then  
    assertThrows(IllegalArgumentException.class, () -> account.setEmail("wrongEmail"));  
}
```

- **Ordering** - np. metoda oczekuje wartości posortowanej – należy sprawdzić co zrobi gdy dostanie nie posegregowaną
- **Range** - testowanie przekraczania wartości zmiennych lub naszych nadanych maksymalnych minimalnych
- **Reference**- //given //when //then
- **Existence**- co się stanie gdy prześlemy do metody coś pustego/null
- **Cardinality** – Sprawdzanie wielkości list np po czyszczeniu
- **Time**- Sprawdzenie kolejności wykonywania metod np zapisywanie do pliku bez jego utworzenia – sprawdzanie wielowątkowości

## Test Coverage – pokrycie testami kodu

Plugin jacoco nie chce mi działać

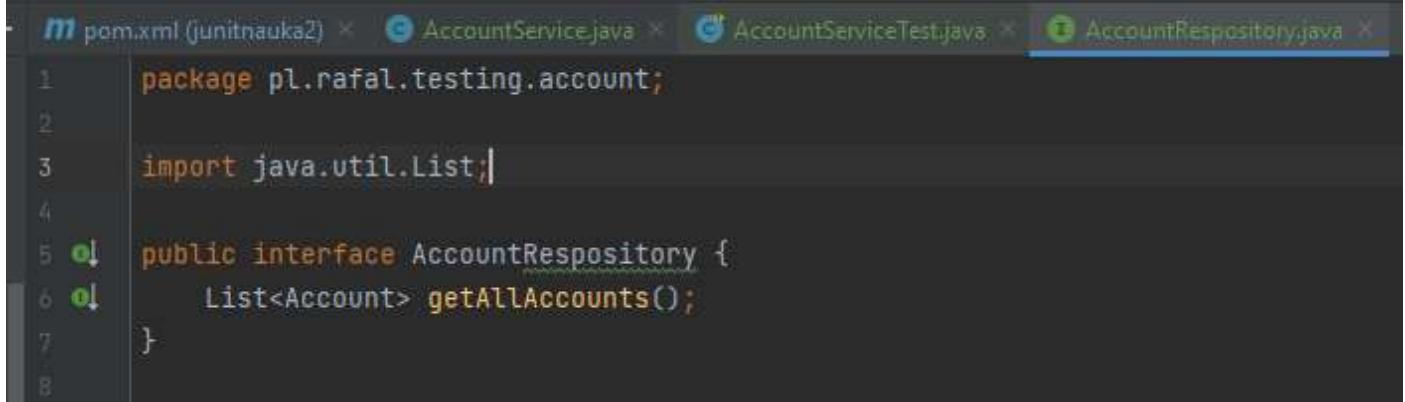
Jakins

## Mockito 2

### Stuby

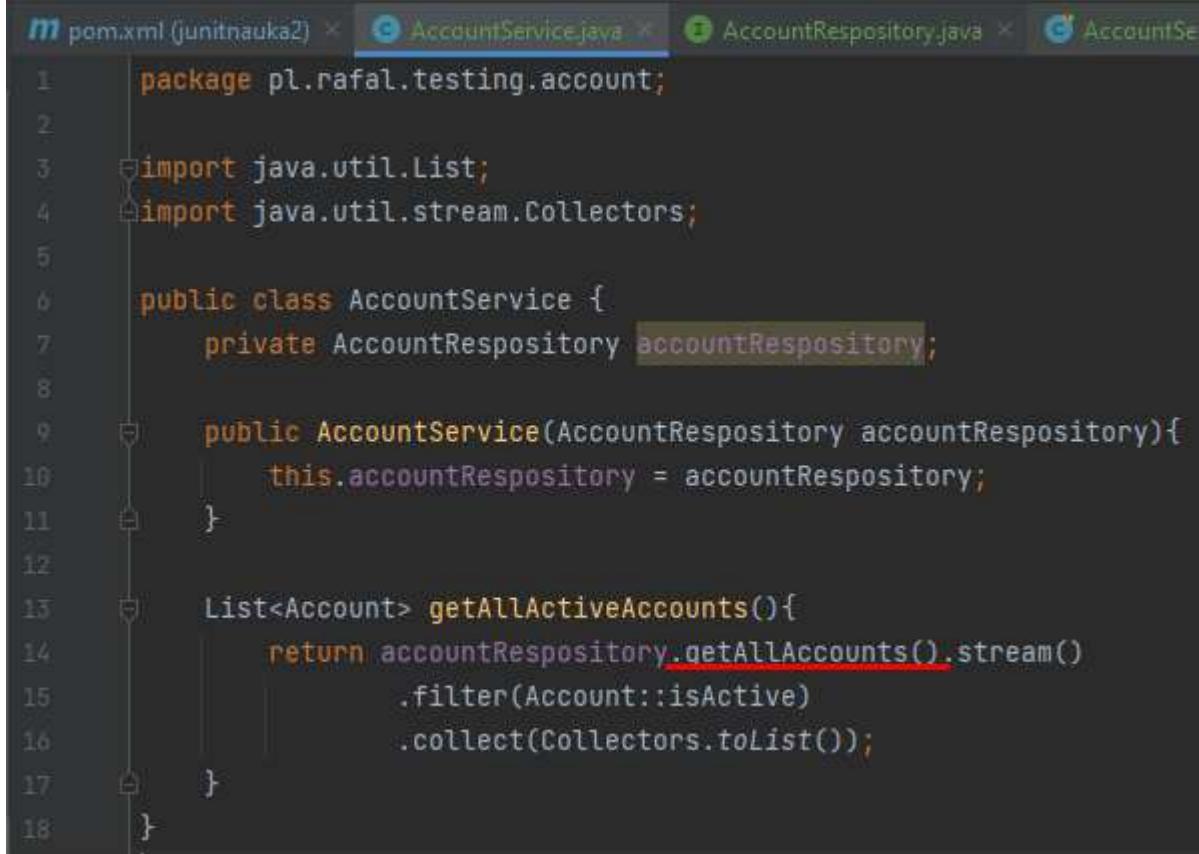
Implementujemy metode w interfejsie które powinna zwracać obiekty w repo

untRepository



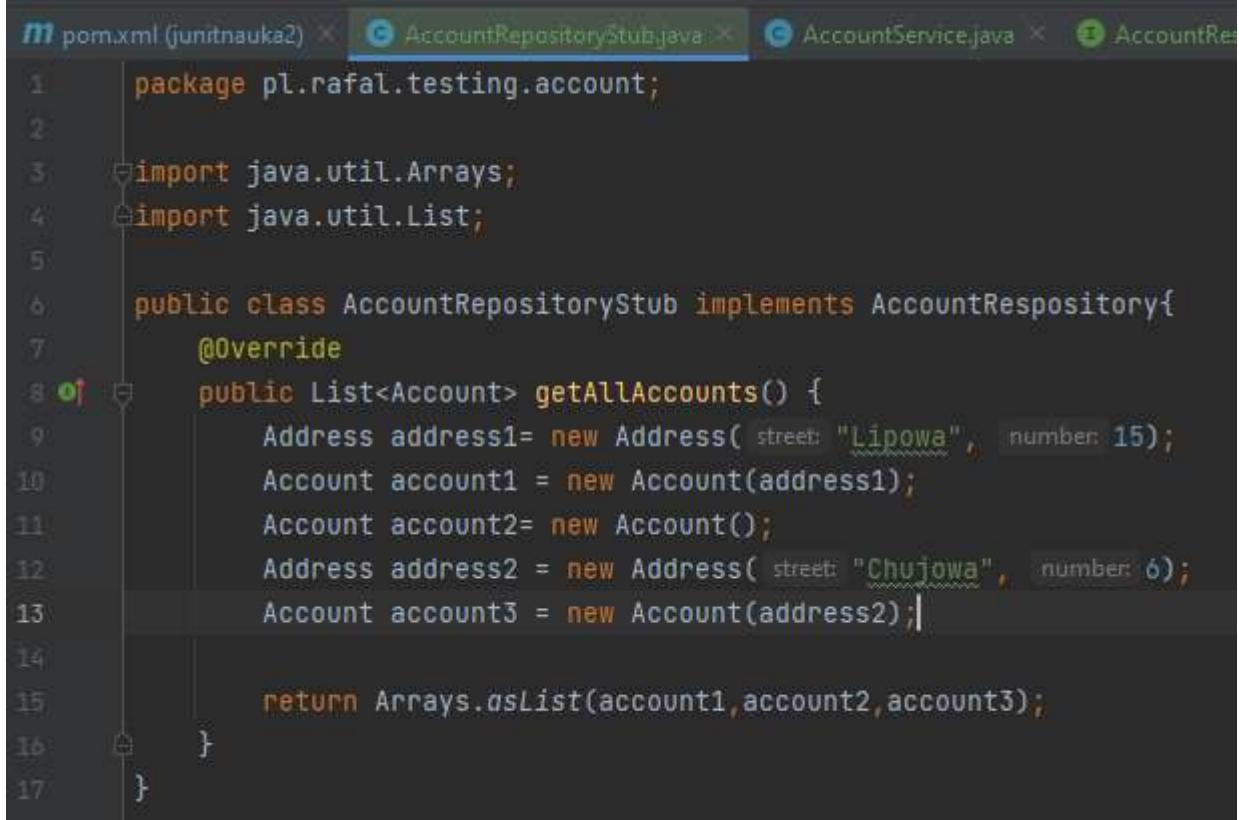
```
1 package pl.rafał.testing.account;
2
3 import java.util.List;
4
5 public interface AccountRespository {
6     List<Account> getAllAccounts();
7 }
8
```

Serwis który będzie zajmował się pobieraniem danych z repo



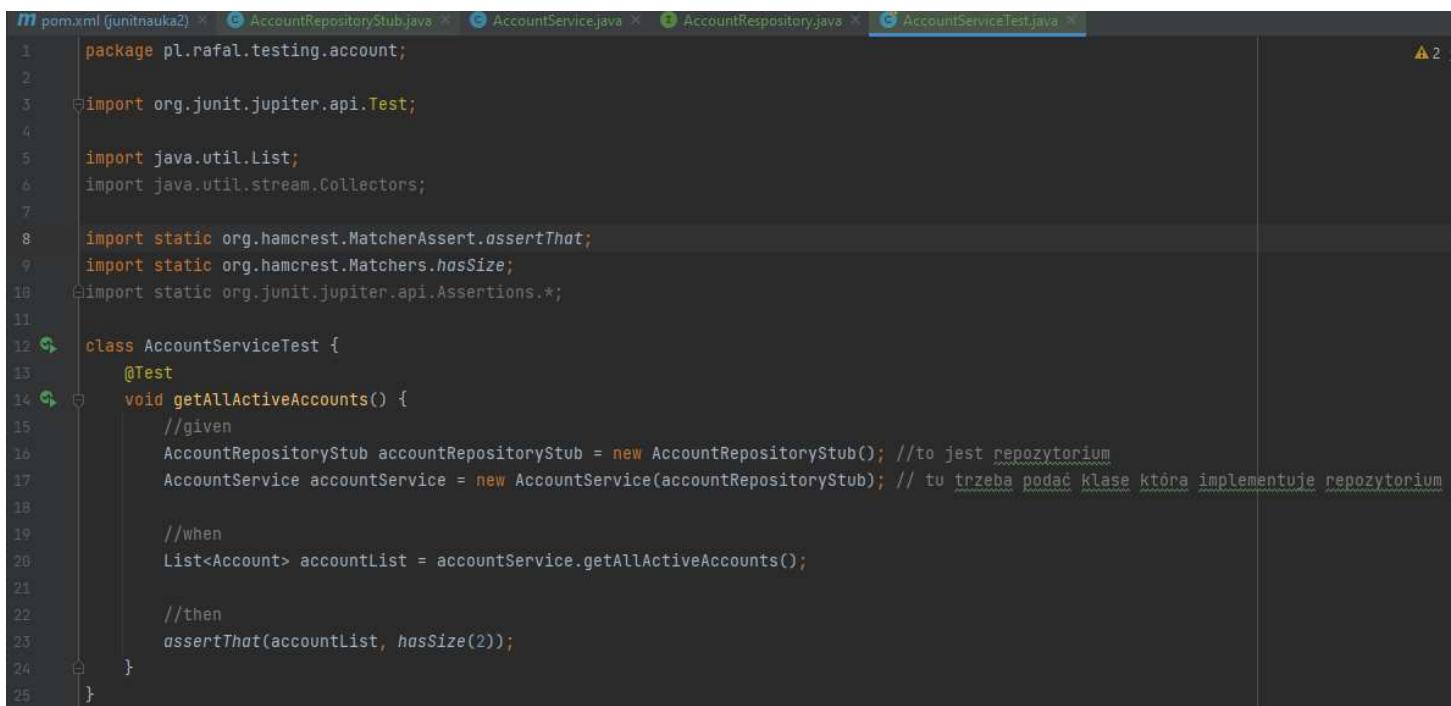
```
1 package pl.rafał.testing.account;
2
3 import java.util.List;
4 import java.util.stream.Collectors;
5
6 public class AccountService {
7     private AccountRespository accountRespository;
8
9     public AccountService(AccountRespository accountRespository){
10         this.accountRespository = accountRespository;
11     }
12
13     List<Account> getAllActiveAccounts(){
14         return accountRespository.getAllAccounts().stream()
15             .filter(Account::isActive)
16             .collect(Collectors.toList());
17     }
18 }
```

Klasa repo czyli Stub implementuje interfejs



```
1 package pl.rafał.testing.account;
2
3 import java.util.Arrays;
4 import java.util.List;
5
6 public class AccountRepositoryStub implements AccountRespository{
7     @Override
8     public List<Account> getAllAccounts() {
9         Address address1= new Address( street: "Lipowa", number: 15);
10        Account account1 = new Account(address1);
11        Account account2= new Account();
12        Address address2 = new Address( street: "Chujowa", number: 6);
13        Account account3 = new Account(address2);
14
15        return Arrays.asList(account1,account2,account3);
16    }
17}
```

Klasa testowa



```
1 package pl.rafał.testing.account;
2
3 import org.junit.jupiter.api.Test;
4
5 import java.util.List;
6 import java.util.stream.Collectors;
7
8 import static org.hamcrest.MatcherAssert.assertThat;
9 import static org.hamcrest.Matchers.hasSize;
10 import static org.junit.jupiter.api.Assertions.*;
11
12 class AccountServiceTest {
13     @Test
14     void getAllActiveAccounts() {
15         //given
16         AccountRepositoryStub accountRepositoryStub = new AccountRepositoryStub(); //to jest repozytorium
17         AccountService accountService = new AccountService(accountRepositoryStub); // tu trzeba podać klasę która implementuje repozytorium
18
19         //when
20         List<Account> accountList = accountService.getAllActiveAccounts();
21
22         //then
23         assertThat(accountList, hasSize(2));
24     }
25 }
```

## Mocki oraz implementacja Mockito

Dodajemy dependencje Mockito, mocki to imitacje obiektów. Gdy nie zrobimy konstrukcji when().thenReturn() zwrócony mock będzie to zero lub np. false przyjazny nie powodujący nullpointerexception

**when().thenReturn()** – można zamienić na **given().willReturn()** ze względu na mylące nazwy z BDD

```
pom.xml (junitnauka2) × AccountService.java × AccountRepository.java × AccountRepositoryStub.java × AccountServiceMockTest.java ×
10 import static org.mockito.Mockito.when;
11
12 public class AccountServiceMockTest {
13     @Test
14     void getAllActiveAccounts() {
15         //given
16         List<Account> accounts = prepareData();Lista pomocnicza
17         AccountRespository accountRepository = mock(AccountRespository.class); mock() - tworzy nowy mock z interfejsu AccountRespository
18         AccountService accountService = new AccountService(accountRepository); // tu trzeba podać klasę która implementuje repozytorium
19         when(accountRepository.getAllAccounts()).thenReturn(accounts); Funkcja when().thenReturn() zwraca dany obiekt gdy zostanie wykonana jakaś metoda na danym obiekcie
20         //when
21         List<Account> accountList = accountService.getAllActiveAccounts();
22
23         //then
24         assertThat(accountList, hasSize(2));
25     }
26     @ i private List<Account> prepareData(){ Metoda pomocnicza który generuje dane do listy
27         Address address1= new Address( street: "Lipowa", number: 15);
28         Account account1 = new Account(address1);
29         Account account2= new Account();
30         Address address2 = new Address( street: "Chujowa", number: 6);
31         Account account3 = new Account(address2);
32
33         return Arrays.asList(account1,account2,account3);
34     }
35 }
```

Kilka wywołań po kolejno przy każdym wywołaniu zmiana wartości zwracanej

```
@Test
void canHandleCartShouldReturnMultipleValues() {

    //given
    Order order = new Order();
    Cart cart = new Cart();
    cart.addOrderToCart(order);
    CartHandler cartHandler = mock(CartHandler.class);

    given(cartHandler.canHandleCart(cart)).willReturn(true, false, false, true);

    //then
    assertThat(cartHandler.canHandleCart(cart), equalTo(true));
    assertThat(cartHandler.canHandleCart(cart), equalTo(false));
    assertThat(cartHandler.canHandleCart(cart), equalTo(false));
    assertThat(cartHandler.canHandleCart(cart), equalTo(true));

}
```

## **Weryfikacja wywołań metod Metoda verify().metoda()**

Można zastąpić then().should().metoda()

```
class CartServiceTest {  
  
    @Test  
    void processCartShouldSendToPrepare(){  
        //given  
        Order order = new Order();  
        Cart cart = new Cart();  
        cart.addOrderToCart(order);  
  
        CartHandler cartHandler = mock(CartHandler.class); // Tworzy mock interfejsu Cart Handler  
        CartService cartService = new CartService(cartHandler); //Przekazuje mock do cart service  
  
        given(cartHandler.canHandCart(cart)).willReturn(true); // Wykonuje metodę interfejsu i zwraca true  
        //when  
        Cart resultCart = cartService.processCart(cart); // wykonuje metody klasy cartservice na mocku  
  
        //then  
        verify(cartHandler).sendToPrepare(cart); // sprawdza czy na mocku została wywołana metoda z metody processCart  
    }  
}
```

```
package pl.rafał.testing.cart;  
  
import pl.rafał.testing.order.OrderStatus;  
  
public class CartService {  
  
    private CartHandler cartHandler;  
  
    public CartService(CartHandler cartHandler) { this.cartHandler = cartHandler; }  
  
    Cart processCart(Cart cart){  
        if(cartHandler.canHandCart(cart)){  
            cartHandler.sendToPrepare(cart);  
            cart.getOrders().forEach(order ->  
                order.changeOrderStatus(OrderStatus.PREPARING));  
            return cart;  
        }else {  
            cart.getOrders().forEach(order ->  
                order.changeOrderStatus(OrderStatus.REJECTED));  
            return cart;  
        }  
    }  
}
```

Dodatkowe metody verify()...

```
verify(cartHandler, times(wantedNumberOfInvocations, 1)).sendToPrepare(cart);
verify(cartHandler, atLeastOnce()).sendToPrepare(cart);
```

Metoda **verify** na obiekcie **InOrder** sprawdza też kolejność wykonywanych metod

```
InOrder inOrder = inOrder(cartHandler);
inOrder.verify(cartHandler).canHandleCart(cart);
inOrder.verify(cartHandler).sendToPrepare(cart);
```

Metoda **verify()** sprawdzająca czy dana metoda **nie została nigdy wykonana**.

```
verify(obejkt, never()).metoda();
then(obejkt).should(never()).metoda();
```

```
//then
verify(cartHandler, never()).sendToPrepare(cart);
then(cartHandler).should(never()).sendToPrepare(cart);
expectThat(resultCart.cartOrders().size(), equalTo(1));
```

## Argument Matchery **any()**

UWAGA ! ZABRONIONE JEST MIESZANIE MATCHERÓW Z PRAWDZIWYMI WARTOŚCIAMI

//Zadziała przy jakimkolwiek obiekcie

```
given(cartHandler.canHandleCart(any())).willReturn(false);
```

//Zadziała tylko przy obiekcie wskazanego typu

```
given(cartHandler.canHandleCart(any(Cart.class))).willReturn(false);
```

Można też wyznać typy proste.

```
anyString()
anyInt()
```

## Matchery z wyrażeniami lambda (*Filtrowanie obiektów z którymi wywoływana jest metoda*)

np. to zadziała tylko na obiekty które mają liste zamówień większą niż 0.

```
given(cartHandler.canHandleCart(argThat(x -> x.getOrders().size() > 0))).willReturn(true);
// Gdy zostanie wykonana metoda canHandleCart( posiadająca conajmniej liste z zamówieniami większą od 0) zwraca true
```

## Wyrzucanie wyjątków za pomocą given().willThrow() oraz ich obsługa assertThrows()

```
@Test
void processCartShouldThrowException(){
    //given
    Order order = new Order();
    Cart cart = new Cart();
    cart.addOrderToCart(order);

    CartHandler cartHandler = mock(CartHandler.class); // Tworzy mock interfejsu Cart Handler
    CartService cartService = new CartService(cartHandler); //Przekazuje mock do cart service

    given(cartHandler.canHandleCart(cart)).willThrow(IllegalStateException.class);

    //when
    //then
    assertThrows(IllegalStateException.class, () -> cartService.processCart(cart));
    //wywołuje metodę processCart za pomocą lambdy która wywoła metodę canHandleCart która wyciągnie nasz podany wyjątek
}
```

## ArgumentCaptor – przechwytywanie przekazywanych obiektów

```
@Test
void processCartShouldSendToPrepareWithArgumentCaptor(){
    //given
    Order order = new Order();
    Cart cart = new Cart();
    cart.addOrderToCart(order);

    CartHandler cartHandler = mock(CartHandler.class); // Tworzy mock interfejsu Cart Handler
    CartService cartService = new CartService(cartHandler); //Przekazuje mock do cart service

    ArgumentCaptor<Cart> argumentCaptor = ArgumentCaptor.forClass(Cart.class);
    // wywołanie argumentCaptor łączy obiekt który podajemy do metody za pomocą given

    given(cartHandler.canHandleCart(cart)).willReturn(true);

    //when
    Cart resultCart = cartService.processCart(cart); // wykonuje metody klasy cartService na mocku

    //then
    then(cartHandler).should().sendToPrepare(argumentCaptor.capture());
    // sprawdza czy na mocku została wywołana metoda z metodą processCart oraz przechwytuje obiekt do argumentCaptor
    assertThat(argumentCaptor.getValue().getOrders().size(), equalTo(1));
    //assertja sprawdza czy obiekt który został przechwycony posiada listę wielkości 1
}
```

```
ArgumentCaptor<Cart> argumentCaptor = ArgumentCaptor.forClass(Cart.class);
```

Można zastąpić

```
@Captor
private ArgumentCaptor<Cart> argumentCaptor;
```

## `doAnswer(lambda).when(obiekt).metoda` Przechwycenie oraz modyfikacja przekazanego obiektu

```
@Test
void shouldAnswerWhenProcessCart() {
    //given
    Order order = new Order();
    Cart cart = new Cart();
    cart.addOrderToCart(order);

    CartHandler cartHandler = mock(CartHandler.class);
    CartService cartService = new CartService(cartHandler);

    doAnswer(invocationOnMock -> { Wydanie lambda modyfikujace przechwycony obiekt z
        Cart argumentCart = invocationOnMock.getArgument(0);
        argumentCart.clearCart();
        return true;
    }).when(cartHandler).canHandleCart(cart); Gdy zostanie wywolana metoda canHandleCart

    when(cartHandler.canHandleCart(cart)).then(i -> {
        Cart argumentCart = i.getArgument(0);
        argumentCart.clearCart();
        return true;
    });

    //when Tutaj wywolana zostanie metoda canHandleCart
    Cart resultCart = cartService.processCart(cart);

    //then
    then(cartHandler).should().sendToPrepare(cart);
    assertThat(resultCart.getOrders().size(), equalTo(0));
}
}
```

```
willAnswer(invocationOnMock -> {
    Cart argumentCart = invocationOnMock.getArgument(0);
    argumentCart.clearCart(); BDD
    return true;
}).given(cartHandler).canHandleCart(cart);

given(cartHandler.canHandleCart(cart)).will(i -> {
    Cart argumentCart = i.getArgument(0);
    argumentCart.clearCart(); BDD
    return true;
}); |
```

## **Wywołanie prawdziwej metody na mocku given().willCallRealMethod()**

```
@Test
void testMealSumPrice(){
    //given
    Meal meal = mock(Meal.class);

    given(meal.getPrice()).willReturn(15);
    given(meal.getQuantity()).willReturn(3);
    given(meal.sumPrice()).willCallRealMethod();
    // użycie prawdziwej metody na mocku. Bez tego zwracane będzie w przypadku typu int 0
    //when
    int result = meal.sumPrice();

    //then
    assertThat(result, equalTo(45));
}
```

## **Adnotacje oraz uproszczenie kodu @InjectMocks @Mock @ExtendWith @Captor @MockitoSettings**

```
@ExtendWith(MockitoExtension.class)
class CartServiceTest {

    @InjectMocks
    private CartService cartService; // Przekazuje mock do cart service
    @Mock
    private CartHandler cartHandler; // Tworzy mock interfejsu Cart Handler

    @Test
    void processCartShouldSendToPrepare(){
        //given
        Order order = new Order();
        Cart cart = new Cart();
        cart.addOrderToCart(order);

        CartHandler cartHandler = mock(CartHandler.class); // Tworzy mock interfejsu Cart Handler
        CartService cartService = new CartService(cartHandler); // Przekazuje mock do cart service
    }
}
```

@InjectMocks i @Mock zastępuje implementacje obiektów mockowych oraz ich tworzenie

```
ArgumentCaptor<Cart> argumentCaptor = ArgumentCaptor.forClass(Cart.class);
```

Można zastąpić

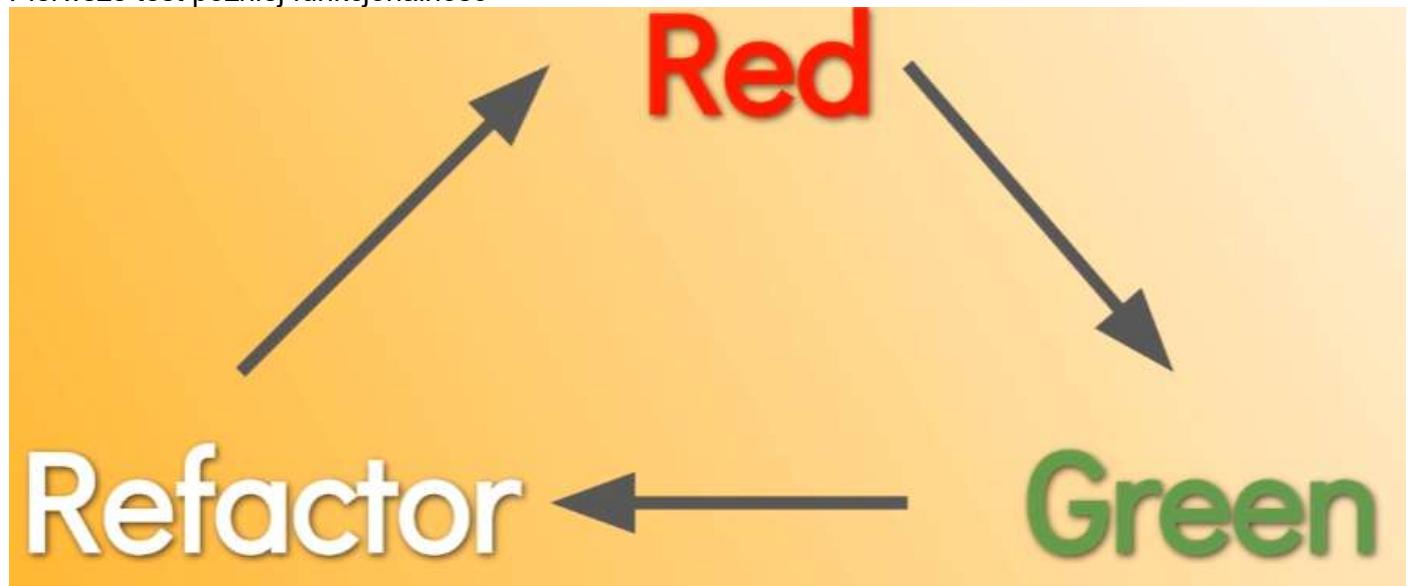
```
@Captor  
private ArgumentCaptor<Cart> argumentCaptor;
```

Zwiększa dokładność błędów

```
@MockitoSettings(strictness = Strictness.STRICT_STUBS)
```

## TDD

Pierwsze test później funkcjonalność



Tworzymy test który nie przechodzi → Tworzymy funkcjonalności by test przechodził → Refactor ->  
Wracamy do początku

## Wzorce projektowe

### Wzorce kreacyjne:

- Builder
- Factory
- Singleton

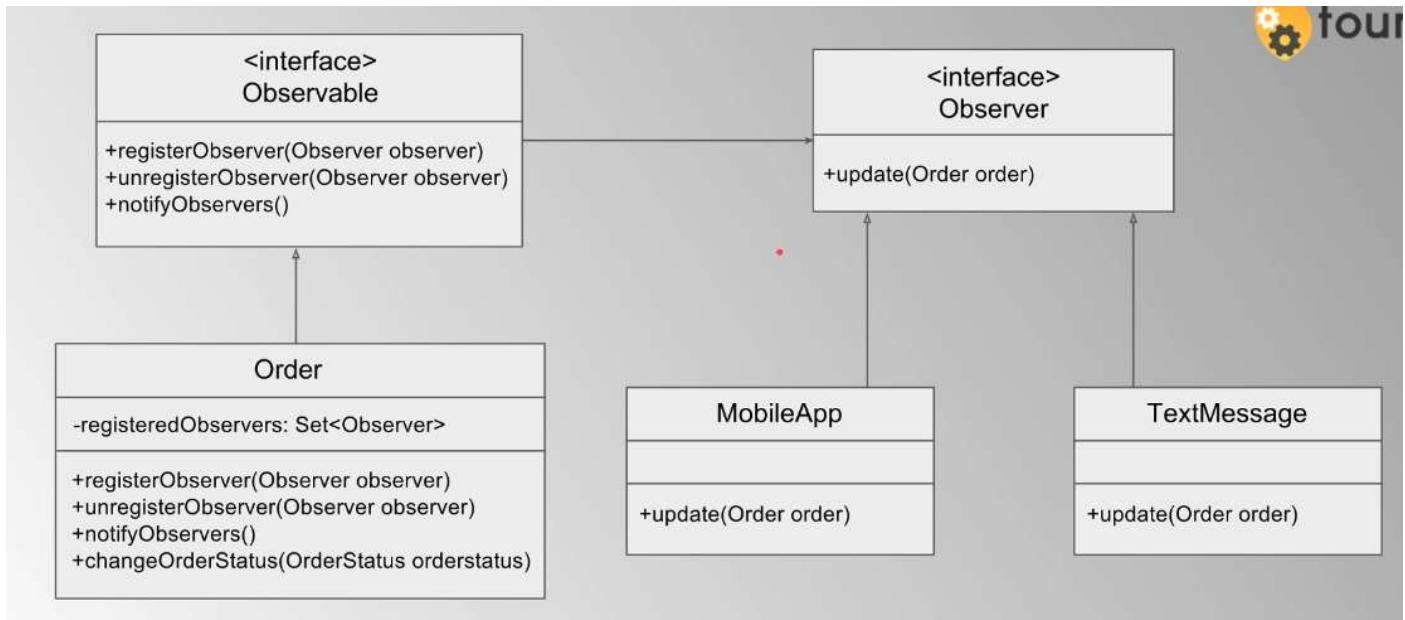
### Wzorce strukturalne:

- Adapter
- Decorator
- Flyweight

### Wzorce behawioralne:

- Chain of responsibility
- Command
- Memento
- Template method
- Strategy
- Observer

## Observer



Obserwatorzy muszą implementować interfejs observer z metodą update.

Obiekty order implementują interfejs observable z metodami registerObserver unregisterobserver oraz wywoływanie obserwatorów notifyObservers. W obiekcie order tworzymy liste obserwatorów.

```
public class Order implements Observable{
    private Long orderNumber;
    private OrderStatus orderStatus;
    //list of observer
    private Set<Observer> registeredObservers = new HashSet<>();
    //constructor
    public Order(Long orderNumber, OrderStatus orderStatus) {...}
    //implements method of observable
    @Override
    public void registerObserver(Observer observer) {
        registeredObservers.add(observer);
    }

    @Override
    public void unregisterObserver(Observer observer) {
        registeredObservers.remove(observer);
    }

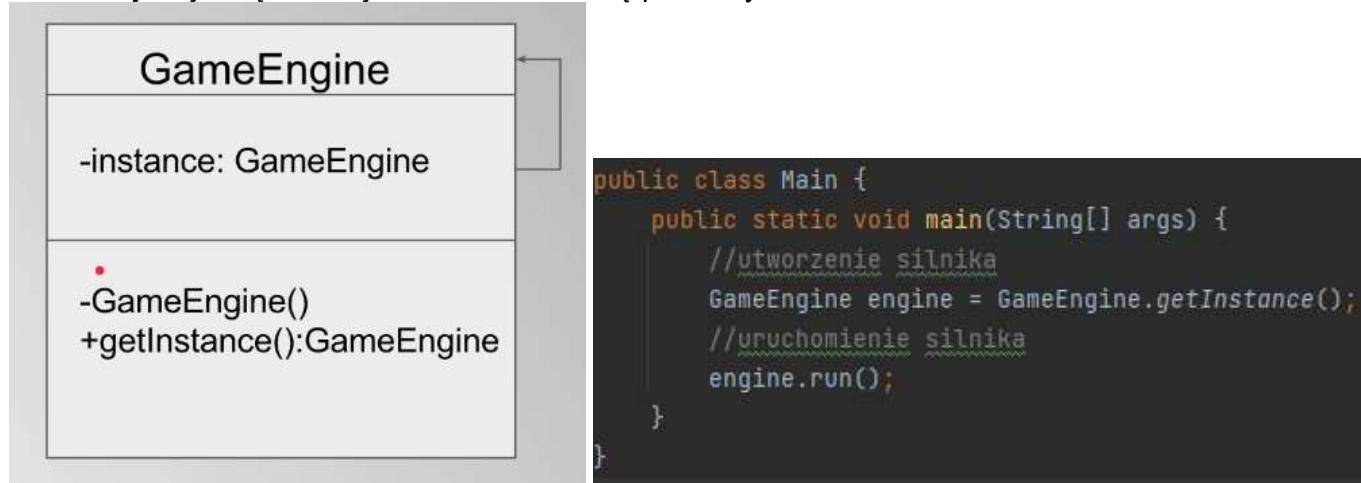
    @Override
    public void notificationObserver() {
        for (Observer e: registeredObservers) {
            e.update(this);
        }
    }

    public void changeOrderStatus(OrderStatus orderStatus){
        setOrderStatus(orderStatus);
        notificationObserver();
    }

    public interface Observable {
        void registerObserver(Observer observer);
        void unregisterObserver(Observer observer); public interface Observer {
            void update(Order order);
        }
    }
}
```

## Singleton

Posiada tylko jedną instancję która nie może się powtoryć



```
public class GameEngine {
    private int hp = 100;
    private String characterName = "";
    //pole statyczne z dostępem bez konieczności tworzenia obiektu
    private static GameEngine instance;
    //prywatny konstruktor
    private GameEngine() {
    }

    public void run(){
        while(true){
            //czekamy na input gracza
            //zmieniamy stan gry
            //renderujemy grafike
        }
    }
    //metoda tworząca silnik oraz zabezpieczająca przed jego wielokrotnym utworzeniem
    public static GameEngine getInstance(){
        if(instance == null){
            //chronienie wielowatkowości
            synchronized (GameEngine.class){
                if(instance == null) instance = new GameEngine();
            }
        }
        return instance;
    }
}
```

## Builder z klasą wewnętrzna

House
-walls: String
-floors: String
-roof: String
-House(HouseBuilder houseBuilder): House
+getWalls(): String
+getFloors(): String
+getRoof(): String
+static HouseBuilder
-walls: String
-floors: String
-roof: String
+buildWalls(String walls): HouseBuilder
+buildFloors(String floors): HouseBuilder
+buildRoof(String roof): HouseBuilder
+build(): House

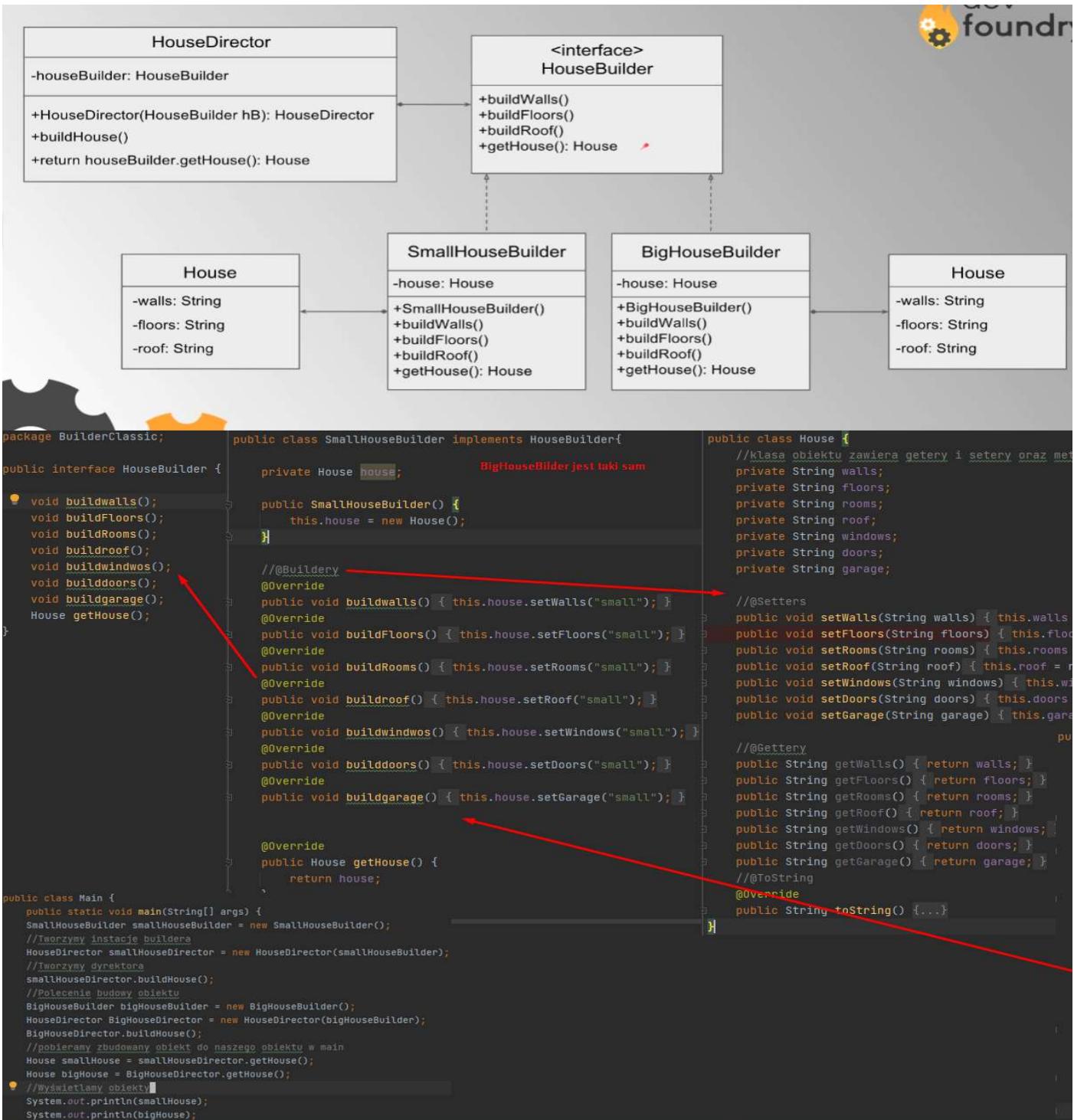
```
public class House {  
    private String walls;  
    private String floors;  
    private String rooms;  
    private String roof;  
    private String windows;  
    private String doors;  
    private String garage;  
  
    //konstruktor prywatny dla klasy housebuilder  
    //pobierajacy dane pol z housebuilder  
    private House(HouseBuilder houseBuilder) {  
        this.walls = houseBuilder.walls;  
        this.floors = houseBuilder.floors;  
        this.rooms = houseBuilder.rooms;  
        this.roof = houseBuilder.roof;  
        this.windows = houseBuilder.windows;  
        this.doors = houseBuilder.doors;  
        this.garage = houseBuilder.garage;  
    }  
    //@Gettery  
    //@ToString
```

```
public static class HouseBuilder {  
    private String walls;  
    private String floors;  
    private String rooms;  
    private String roof;  
    private String windows;  
    private String doors;  
    private String garage;  
  
    //@Buildery metody budujace "obiekt" zwracajace obiekt  
    // dzieki czemu mozemy wielokrotnie sie odwoywac  
    public HouseBuilder buildWalls(String walls){  
        this.walls = walls;  
        return this;  
    }  
    public HouseBuilder buildFloors(String floors){...}  
    public HouseBuilder buildRooms(String rooms){...}  
    public HouseBuilder buildRoof(String roof){...}  
    public HouseBuilder buildWindows(String windows){...}  
    public HouseBuilder buildDoors(String doors){...}  
    public HouseBuilder buildGarage(String garage){...}  
    //Metoda zakonczajaca proces budowy obiektu wywoluje  
    //konstruktor prywatny i przekazuje do niego obiekt "HouseBuilder"  
    public House build(){  
        return new House( houseBuilder: this);  
    }  
}
```

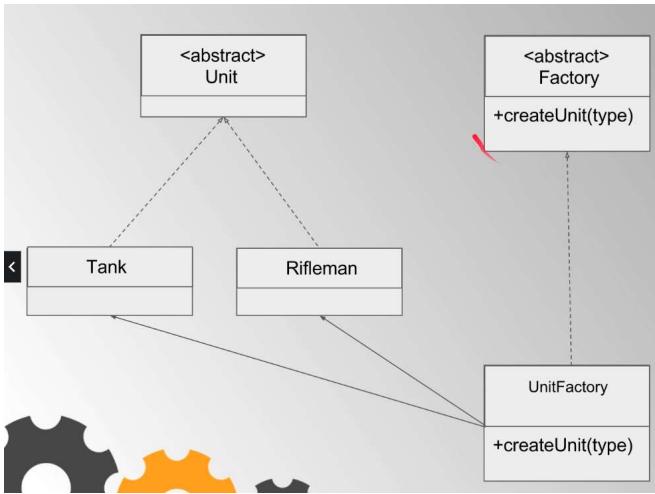
Przykład tworzenia nowego obiektu

```
//  
//  
House house1 = new House("walls", "floors", "rooms", "windows", "doors", "garage");  
House house2 = new House("")  
  
House house = new House.HouseBuilder()  
    .buildWalls("walls")  
    .buildFloors("floors")  
    .buildRoof("roof")  
    .buildRooms("rooms")  
    .build();
```

# Builder Classic



# Factory FABRIC



```
package FactoryPackage.FabricMethod.Units;

abstract public class Factory {
    abstract public Unit createUnit(UnitType type);
}

package FactoryPackage.FabricMethod.Units;

public class UnitFactory extends Factory{

    @Override
    public Unit createUnit(UnitType unitType) {

        switch (unitType){
            case Tank:
                return new Tank( hp: 200, exp: 0, damage: 20);
            case Rifleman:
                return new Rifleman( hp: 100, exp: 0, damage: 10);
            default:
                throw new UnsupportedOperationException("No such type");
        }
    }
}
```

```
package FactoryPackage.FabricMethod.Units;

public abstract class Unit {

    private int hp;
    private int exp;
    private int damage;

    protected Unit(int hp, int exp, int damage) {
        this.hp = hp;
        this.exp = exp;
        this.damage = damage;
    }

    // @Getter
    public int getHp() { return hp; }
    public int getExp() { return exp; }
    public int getDamage() { return damage; }
}

package FactoryPackage.FabricMethod.Units;

public class Rifleman extends Unit{

    Rifleman(int hp, int exp, int damage) { super(hp, exp, damage); }

}

package FactoryPackage.FabricMethod.Units;

public class Tank extends Unit{

    Tank(int hp, int exp, int damage) { super(hp, exp, damage); }

}
```

```
package FactoryPackage.AbstractMethod;

import FactoryPackage.FabricMethod.Units.*;

public class Main {
    public static void main(String[] args) {
        Factory factory = new UnitFactory();

        // Nie używamy konstruktora tylko wywołujemy metodę createUnit z UnitFactory.class
        // zaimplementowaną w klasie abstrakcyjnej factory
        Unit tank = factory.createUnit(UnitType.Tank);
        Unit rifleman = factory.createUnit(UnitType.Rifleman);

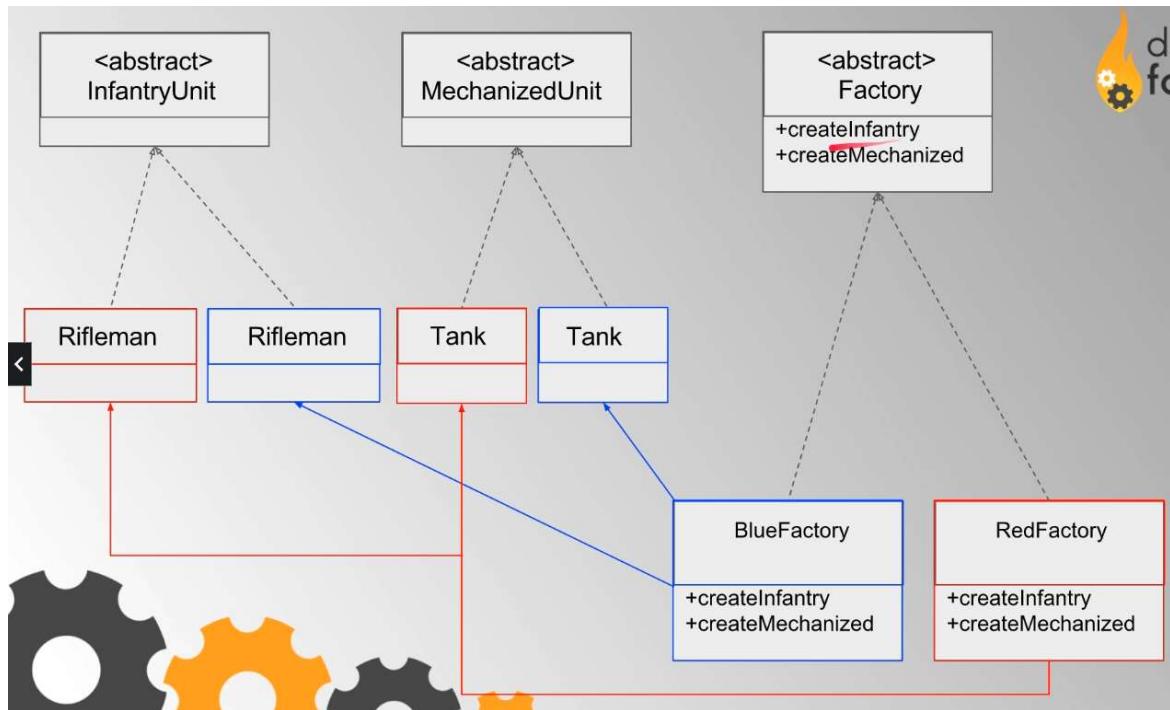
        // Tak nie można chronione jest to przez paczke units. Klasa
        // Unit.class ma konstruktor protected (czyli tylko w obrębie klas rozszerzających)
        // Klasa jednostek ma widoczność domyślną czyli w obrębie pakietu dzialeki czemu
        // UnitFactory.class może używać ich konstruktor.
        // Unit xd = new Rifleman(100, 5, 2);

    }
}

package FactoryPackage.FabricMethod.Units;

public enum UnitType {
    Tank, Rifleman
}
```

# Factory Abstract



```

package FactoryPackage.AbstractMethod.Factory;

abstract public class Factory {
    abstract public InfantryUnit createUnit(InfantryUnitType unitType);
    abstract public MechanizedUnit createUnit(MechanizedUnitType unitType);
}

package FactoryPackage.AbstractMethod.Factory;

public class BlueFactory extends Factory{
    private Flag flag = Flag.BLUE;

    @Override
    public InfantryUnit createUnit(InfantryUnitType unitType) {
        switch (unitType){
            case Sniper:
                return new Sniper( hp: 100, exp: 0, damage: 100, flag);
            case Rifleman:
                return new Rifleman( hp: 100, exp: 0, damage: 10, flag);
            default:
                throw new UnsupportedOperationException("No such type");
        }
    }

    @Override
    public MechanizedUnit createUnit(MechanizedUnitType unitType) {
        switch (unitType){
            case TANK:
                return new Tank( hp: 200, exp: 0, damage: 20, flag);
            case Car:
                return new Car( hp: 150, exp: 0, damage: 20, flag);
            default:
                throw new UnsupportedOperationException("No such type");
        }
    }
}

package FactoryPackage.AbstractMethod.Factory;

import FactoryPackage.AbstractMethod.Factory.*;
import FactoryPackage.AbstractMethod.Factory.InfantryUnitType;

public class Main {
    public static void main(String[] args) {
        Factory blueFactory = new BlueFactory();
        Factory redFactory = new RedFactory();

        Unit newUnit1 = blueFactory.createUnit(InfantryUnitType.Rifleman);
        Unit newUnit2 = redFactory.createUnit(InfantryUnitType.Rifleman);

        System.out.println("Siema tutaj " + newUnit1);
        System.out.println("Siema tutaj " + newUnit2);
        //Sniper sniper = new Sniper(50,0,5, Flag.RED);
    }
}

package FactoryPackage.AbstractMethod.Factory;

public enum InfantryUnitType {
    Rifleman, Sniper
}

package FactoryPackage.AbstractMethod.Factory;

public abstract class Unit {
}

package FactoryPackage.AbstractMethod.Factory;

public abstract class InfantryUnit extends Unit {
}

package FactoryPackage.AbstractMethod.Factory;

public abstract class MechanizedUnit extends Unit {
}

package FactoryPackage.AbstractMethod.Factory;

public class Sniper extends InfantryUnit {
    Sniper(int hp, int exp, int damage, Flag flag) { super(hp, exp, damage, flag); }

    protected InfantryUnit(int hp, int exp, int damage, Flag flag) {
        this.hp = hp;
        this.exp = exp;
        this.damage = damage;
        this.flag = flag;
    }

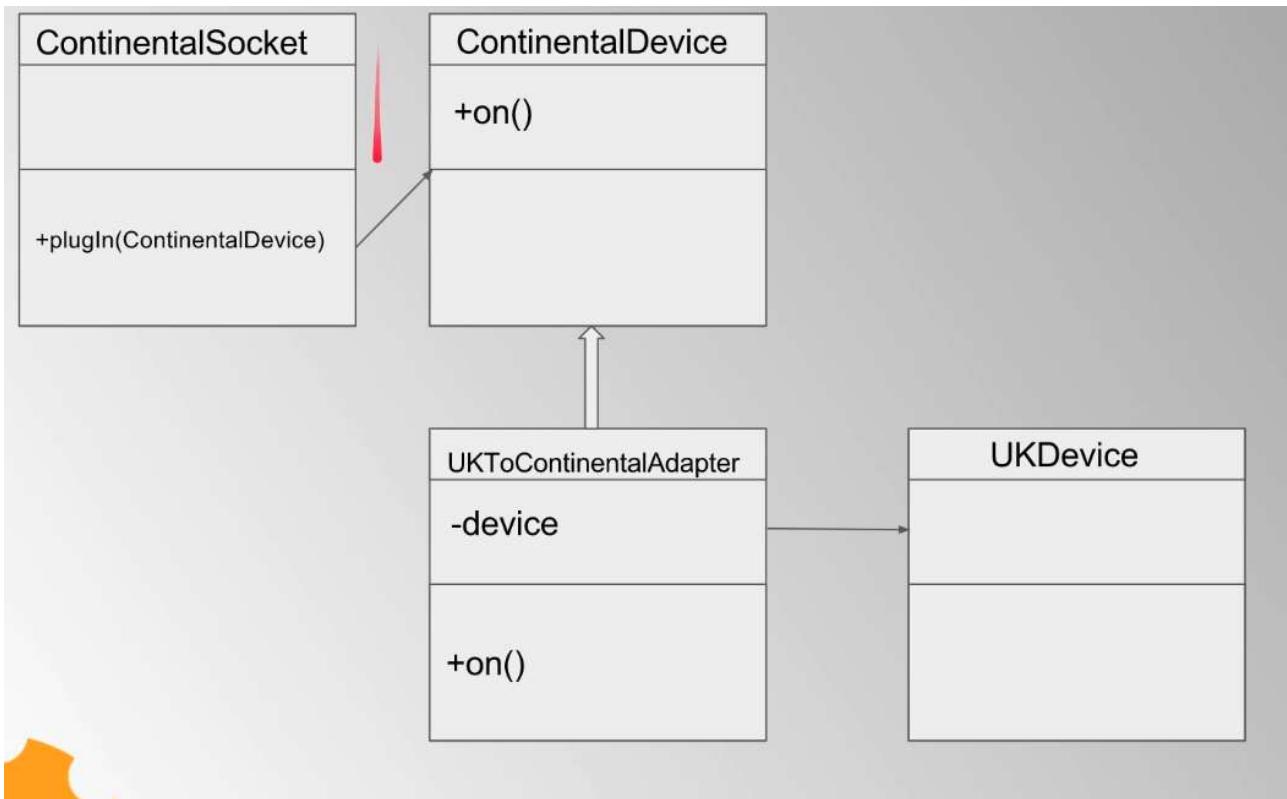
    //Getters
    public int getHp() { return hp; }
    public int getExp() { return exp; }
    public int getDamage() { return damage; }

    @Override
    public String toString() {
        return "InfantryUnit{" +
            "hp=" + hp +
            ", exp=" + exp +
            ", damage=" + damage +
            ", flag=" + flag +
            '}';
    }
}

package FactoryPackage.AbstractMethod.Factory;

public abstract class MechanizedUnit extends Unit {
}
  
```

# Adapter



```

package AdapterPattern;

import AdapterPattern.Adapters.*;

public class Main {
    public static void main(String[] args) {

        UEDevice radio = () -> System.out.println("SIEMA GRA UE");
        UESocekt gniazdkoUE = new UESocekt();
        gniazdkoUE.plugin(radio);

        UKDevice radiozangli = () -> System.out.println("Gra po angielsku");
        UKSocekt gniazdkoUK = new UKSocekt();
        gniazdkoUK.plugin(radiozangli);

        UKtoUE adapter = new UKtoUE(radiozangli);
        gniazdkoUE.plugin(adapter);
    }
}

package AdapterPattern.Adapters;

public class UKtoUE implements UEDevice{

    UKDevice deive;

    public UKtoUE(UKDevice device){
        this.deive = device;
    }

    @Override
    public void powerOn() {
        deive.powerOn();
    }
}
  
```

```

package AdapterPattern.Adapters;

public class UKSocekt {

    public void plugin(UKDevice device){
        device.powerOn();
    }
}

package AdapterPattern.Adapters;

public class UESocekt {

    public void plugin(UEDevice device){
        device.powerOn();
    }
}

package AdapterPattern.Adapters;

public interface UEDevice {
    void powerOn();
}

package AdapterPattern.Adapters;

public interface UKDevice {
    void powerOn();
}
  
```

## Decorator

```
//Klasa abstrakcyjna nie da się zrobić "obiektu typu Meal"
abstract public class Meal {
    public void prepareMeal() {
        System.out.println("Przygotowuję danie.");
    }
}

public class PotatoMeal extends Meal{
    @Override
    public void prepareMeal() {
        System.out.println("Przygotowuję danie na bazie ziemniaków.");
    }
}

public class RiceMeal extends Meal{
    @Override
    public void prepareMeal() {
        System.out.println("Przygotowuję danie na bazie ryżu.");
    }
}

//Klasa abstrakcyjna nie da się zrobić obiektu typu Mealdecorator
abstract public class MealDecorator extends Meal{
    //Posiada własne pole meal
    Meal meal;

    //tworząc decorator przekazujemy obiekt typu meal
    public MealDecorator(Meal meal) {
        this.meal = meal;
    }

    @Override
    public void prepareMeal() {
        this.meal.prepareMeal();
    }
}

public class Main {
    public static void main(String[] args) {
        Meal ryzzkunczakiem = new ChickenMealDecorator(new RiceMeal());
        ryzzkunczakiem.prepareMeal();
    }
}

public class ChickenMealDecorator extends MealDecorator{
    //Posiada własny konstruktor który przyjmuje obiekty typu meal (te same)
    public ChickenMealDecorator(Meal meal) {
        super(meal);
    }

    @Override
    public void prepareMeal() {
        //wywołuje poprzedni prepareMeal(Komunikat)
        super.prepareMeal();
        addChicken();
    }

    private void addChicken(){
        System.out.println("Do dania dodaje kurczaka");
    }
}

public class ShrimpMealDecorator extends MealDecorator{
    //Posiada własny konstruktor który przyjmuje obiekty typu meal (te same)
    public ShrimpMealDecorator(Meal meal) {
        super(meal);
    }

    @Override
    public void prepareMeal() {
        //wywołuje poprzedni prepareMeal(Komunikat)
        super.prepareMeal();
        addShrimp();
    }

    private void addShrimp(){
        System.out.println("Do dania dodaje krewetki");
    }
}
```

## ChainOfResponsibility

```
abstract public class Officer {
    //Ustawienie Odpowiedzialności
    private Officer superiorOfficer;

    abstract public void processMessage(Message message);
    abstract public String getName();

    public Officer getSuperiorOfficer() { return superiorOfficer; }
    //Ustawienie Odpowiedzialności
    public void setSuperiorOfficer(Officer superiorOfficer) { this.superiorOfficer = superiorOfficer; }

    public class Sergeant extends Officer{
        private static final int CODE = 111;
        private static final String NAME = "Sierżant";

        @Override
        public void processMessage(Message message) {
            if(message.getOfficerRank().equals(OfficerRank.SERGEANT)
                && message.getCODE() == CODE){
                System.out.println(NAME + " Otrzymał wiadomość : " + message.getContent());
            } else {
                System.out.println("Wiadomość została przekazana do " + getSuperiorOfficer().getName());
                getSuperiorOfficer().processMessage(message);
            }
        }

        public String getName() { return NAME; }
    }
}

public class General extends Officer{
    private static final int CODE = 333;
    private static final String NAME = "General";

    @Override
    public void processMessage(Message message) {
        if(message.getOfficerRank().equals(OfficerRank.GENERAL)
            && message.getCODE() == CODE){
            System.out.println(NAME + " Otrzymał wiadomość : " + message.getContent());
        } else {
            System.out.println("Wiadomość nie do odszyfrowania");
        }
    }

    public String getName() { return NAME; }
}

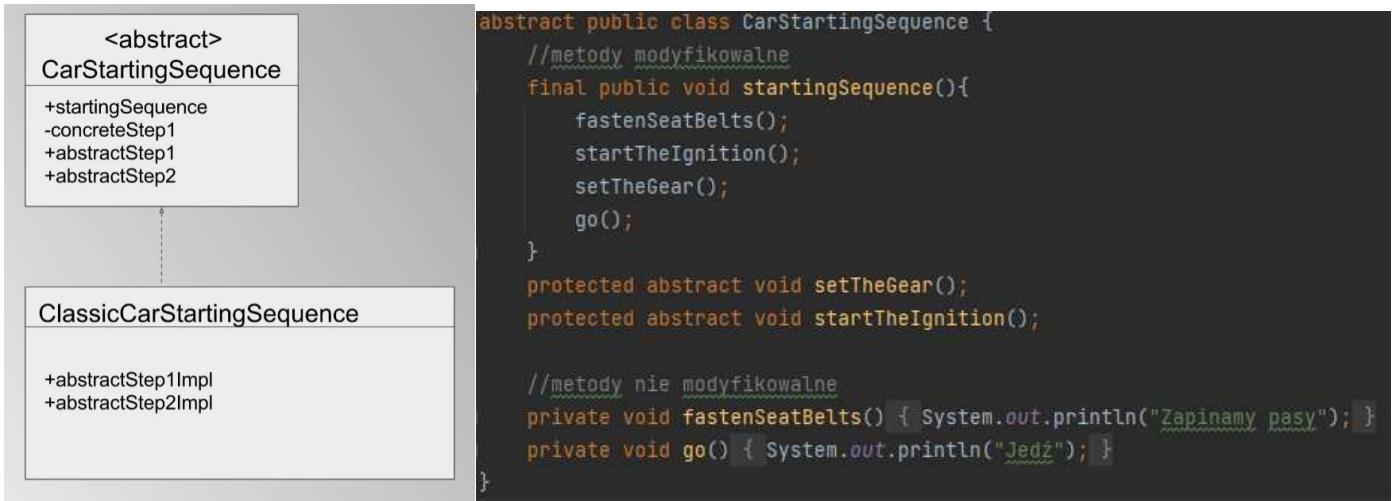
public class Message {
    private String content;
    private int CODE;
    private OfficerRank officerRank;

    public Message(String content, int CODE, OfficerRank officerRank) {
        this.content = content;
        this.CODE = CODE;
        this.officerRank = officerRank;
    }

    public String getContent() { return content; }
    public int getCODE() { return CODE; }
    public OfficerRank getOfficerRank() { return officerRank; }
}

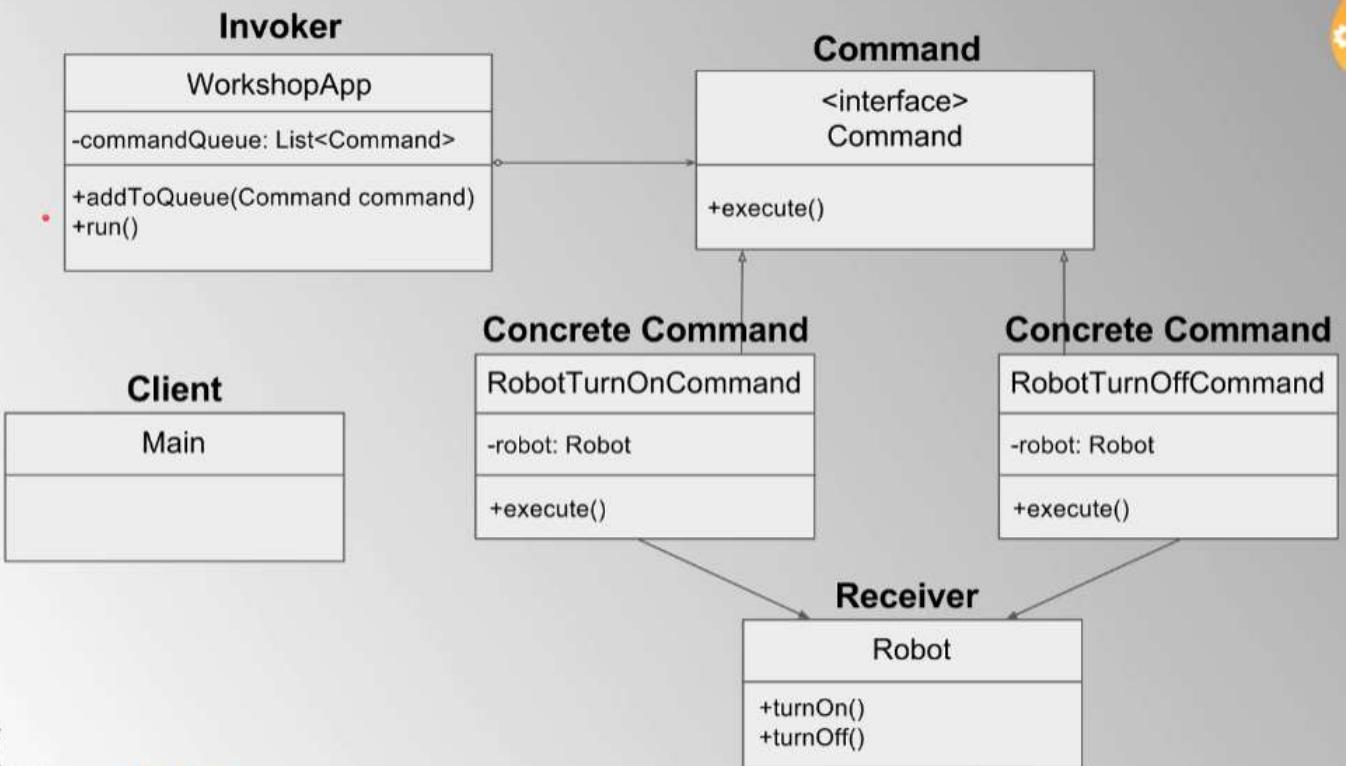
public class Main {
    public static void main(String[] args) {
        //Utworzenie wiadomości
        Message message = new Message(content: "No siema", CODE: 333, OfficerRank.GENERAL);
        //Tworzenie Oficerów
        Officer sergeant = new Sergeant();
        Officer capitain = new Capitain();
        Officer general = new General();
        //Ustawianie Odpowiedzialności
        sergeant.setSuperiorOfficer(capitain);
        capitain.setSuperiorOfficer(general);
        //Przekazanie wiadomości
        sergeant.processMessage(message);
    }
}
```

## TemplateMethod



```
public class Main {  
    public static void main(String[] args) {  
        CarStartingSequence newclassiccar = new StartTheClassicCar();  
        CarStartingSequence newautomat = new StartTheAutomatClassicCar();  
        newclassiccar.startingSequence();  
        newautomat.startingSequence();  
    }  
}  
  
public class StartTheAutomatClassicCar extends StartTheClassicCar{  
    @Override  
    protected void setTheGear() { System.out.println("Automatyczne wybieranie biegu"); }  
}  
  
public class StartTheClassicCar extends CarStartingSequence{  
    @Override  
    protected void setTheGear() { System.out.println("Wybranie biegu"); }  
    @Override  
    protected void startTheIgnition() { System.out.println("Przekrecenie kluczyka"); }  
}
```

## Command



```

public class Main {
    public static void main(String[] args) {
        Robot robot = new Robot();
        CoffeMaker coffeMaker = new CoffeMaker();
        WorkshopApp workshopApp = new WorkshopApp();

        workshopApp.addToQueue(new RobotTurnOn(robot));
        workshopApp.addToQueue(new RobotCut(robot));
        workshopApp.addToQueue(new RobotDrill(robot));
        workshopApp.addToQueue(new RobotTurnOff(robot));
        workshopApp.addToQueue(new CoffeMakerOn(coffeMaker));
        workshopApp.addToQueue(new CoffeMakerOff(coffeMaker));
        workshopApp.run();
    }
}

public class WorkshopApp {
    private static List<Command> CommandQueue = new ArrayList<>();

    public void addToQueue(Command command){
        CommandQueue.add(command);
    }

    public void run(){
        if(CommandQueue.isEmpty()) System.out.println("Kolejka jest pusta");
        for (Command command: CommandQueue) {
            command.execute();
        }
        CommandQueue.clear();
    }
}
  
```

```

public class RobotCut implements Command {
    private Robot robot;

    public RobotCut(Robot robot) { this.robot = robot; }

    @Override
    public void execute() { robot.cut(); }
}

public class Robot {
    void on() { System.out.println("Robot Właczony"); }

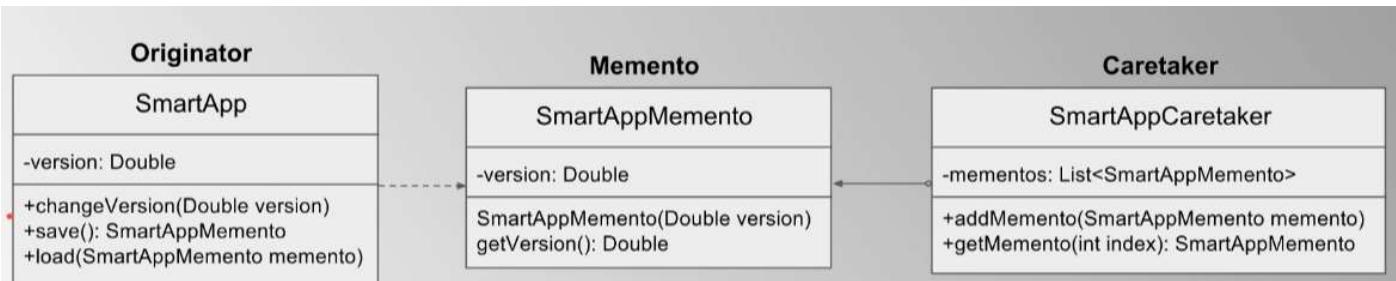
    void cut() { System.out.println("Robot Tnie"); }

    void drill() { System.out.println("Robot Wierci"); }

    void off() { System.out.println("Robot wyłącz się"); }
}
  
```

The code shows the implementation of the Command pattern. The `RobotCut` class implements the `Command` interface, which defines the `execute()` method. The `Robot` class contains methods for turning on, cutting, drilling, and turning off the robot. The `RobotCut` class overrides the `execute()` method to call the `cut()` method on the `Robot` object. The `Robot` class is highlighted with a red box, and the `Command` interface and its `execute()` method are highlighted with a red box and a green arrow pointing to it.

# Memento



```

public class Main {
    public static void main(String[] args) {
        SmartAppCareTaker smartAppCareTaker = new SmartAppCareTaker();
        SmartApp smartApp = new SmartApp();
        smartApp.changeVersion(1.0);
        smartAppCareTaker.addMemento(smartApp.save());

        smartApp.changeVersion(1.1);
        smartApp.changeVersion(1.2);

        smartApp.load(smartAppCareTaker.getMemento( index: 0));
    }
}

public class SmartAppCareTaker {
    private List<SmartAppMemento> mementos = new ArrayList<>();

    public void addMemento(SmartAppMemento smartAppMemento){
        mementos.add(smartAppMemento);
        System.out.println("Zapisana wersja to : " + smartAppMemento.getVersion() + " znajduje sie pod indeksem : " + (mementos.size()-1));
    }

    public SmartAppMemento getMemento(int index){
        System.out.println("Odczytano wersje " + mementos.get(index).getVersion());
        return mementos.get(index);
    }
}

public class SmartApp {
    private Double version;

    public void changeVersion(double version){
        this.version = version;
        System.out.println("Nowa wersja to : " + this.version);
    }

    public SmartAppMemento save(){
        return new SmartAppMemento(this.version);
    }

    public void load(SmartAppMemento smartAppMemento){
        this.version = smartAppMemento.getVersion();
        System.out.println("Wczytano wersje " + smartAppMemento.getVersion());
    }
}

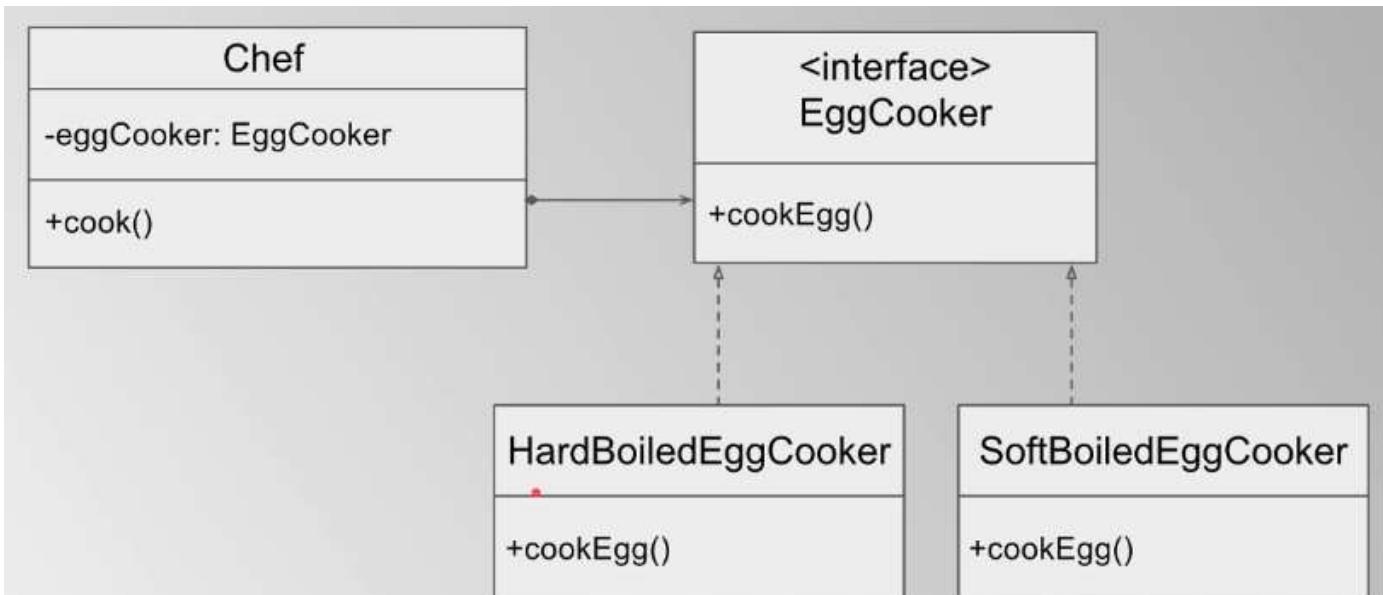
class SmartAppMemento {
    private Double version;

    SmartAppMemento(Double version) { this.version = version; }

    Double getVersion() { return version; }
}

```

## Strategy



```
public interface EggCooker {
    void cookEgg();
}

public class HardBoiledEggCooker implements EggCooker {
    @Override
    public void cookEgg() {
        System.out.print("Gotuje jajka na twardo");
    }
}

public class SoftBoiledEggCooker implements EggCooker {
    @Override
    public void cookEgg() {
        System.out.print("Gotuje jajka na mięko");
    }
}

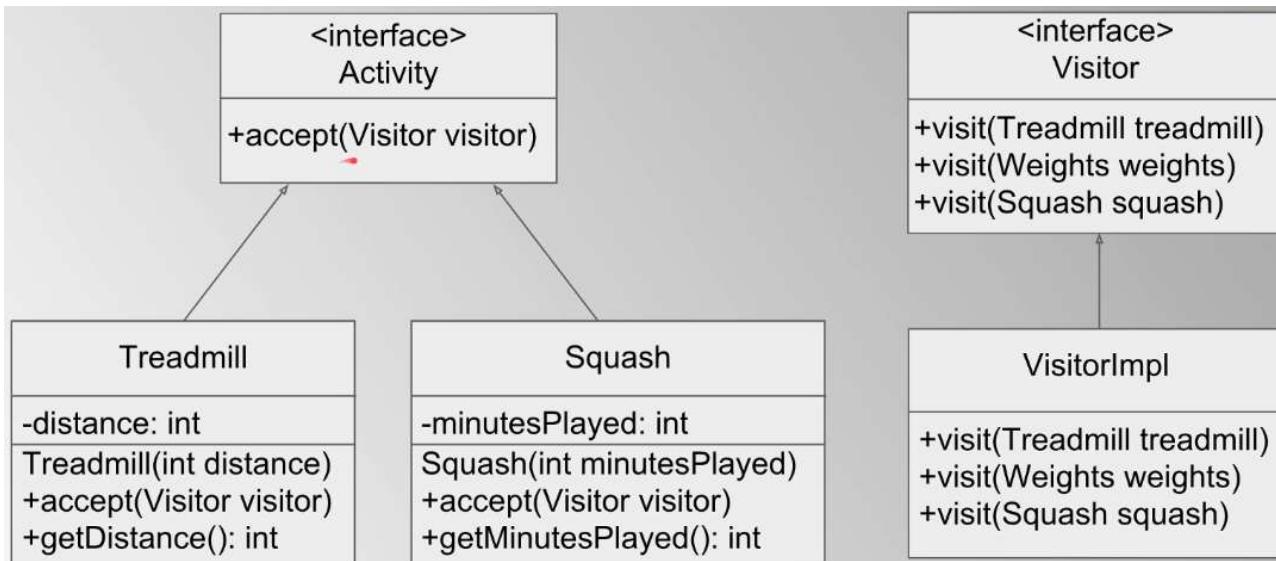
public class Cheff {
    private String name;

    public Cheff(String name) { this.name = name; }

    public void cook(EggCooker eggCooker){
        System.out.print(name + " ");
        eggCooker.cookEgg();
    }
}

public class Main {
    public static void main(String[] args) {
        Cheff cheff = new Cheff( name: "Małcin");
        cheff.cook(new HardBoiledEggCooker());
    }
}
```

## Visitor



```

public interface Visitor {
    void visit(Treadmill trademill);
    void visit(Squash squash);
}

public class VisitorImplement implements Visitor{
    @Override
    public void visit(Treadmill trademill) {
        System.out.println("Spalone Klaorie to : " + trademill.getDistance());
    }

    @Override
    public void visit(Squash squash) {
        System.out.println("Spalone Klaorie to : " + squash.getMinutesPlayed());
    }
}

public interface Activity {
    void accept(Visitor visitor);
}

public class Treadmill implements Activity{
    private int Distance;

    public Treadmill(int distance) {
        Distance = distance;
    }

    @Override
    public void accept(Visitor visitor) {
        visitor.visit( trademill: this);
    }

    public int getDistance() {
        return Distance;
    }
}

public class Main {
    public static void main(String[] args) {
        Visitor visitor = new VisitorImplement();
        Treadmill trademill = new Treadmill( distance: 50);
        trademill.accept(visitor);
    }
}
  
```

# State

```
interface State {
    void insertTheCoin(CoffeMachine coffeMachine);
    void pushTheButton(CoffeMachine coffeMachine);
    void takeTheCup(CoffeMachine coffeMachine);
    void returnTheCoin(CoffeMachine coffeMachine);
}

class NoCoin implements State{
    @Override
    public void insertTheCoin(CoffeMachine coffeMachine) {
        System.out.println("Wrzucono Monete");
        coffeMachine.setState(new CoinInserted());
    }

    @Override
    public void pushTheButton(CoffeMachine coffeMachine) {
        System.out.println("Wrzuć monetę");
    }

    @Override
    public void takeTheCup(CoffeMachine coffeMachine) {
        System.out.println("Wrzuć monetę");
    }

    @Override
    public void returnTheCoin(CoffeMachine coffeMachine) {
        System.out.println("Wrzuć monetę");
    }
}

class CoinInserted implements State{
    @Override
    public void insertTheCoin(CoffeMachine coffeMachine) {
        System.out.println("Moneta już jest");
    }

    @Override
    public void pushTheButton(CoffeMachine coffeMachine) {
        System.out.println("Nalewam kawę");
        coffeMachine.setState(new CupFull());
    }

    @Override
    public void takeTheCup(CoffeMachine coffeMachine) {
        System.out.println("Wcisnij przycisk");
    }

    @Override
    public void returnTheCoin(CoffeMachine coffeMachine) {
        System.out.println("Zwrócono gruby hajś");
        coffeMachine.setState(new NoCoin());
    }
}

class CupFull implements State{
    @Override
    public void insertTheCoin(CoffeMachine coffeMachine) {
        System.out.println("Zabierz kubek");
    }

    @Override
    public void pushTheButton(CoffeMachine coffeMachine) {
        System.out.println("Zabierz Kubek");
    }

    @Override
    public void takeTheCup(CoffeMachine coffeMachine) {
        System.out.println("Kubek zabrano");
        coffeMachine.setState(new NoCoin());
    }

    @Override
    public void returnTheCoin(CoffeMachine coffeMachine) {
        System.out.println("Teraz to za późno");
    }
}

public class Main {
    public static void main(String[] args) {
        CoffeMachine coffeMachine = new CoffeMachine();

        coffeMachine.insertTheCoin();
        coffeMachine.returnTheCoin();
        coffeMachine.insertTheCoin();
        coffeMachine.pushTheButton();
        coffeMachine.takeTheCup();
        coffeMachine.pushTheButton();
    }
}

public class CoffeMachine {
    private State state;

    public CoffeMachine() { this.state = new NoCoin(); }

    public void insertTheCoin() { state.insertTheCoin(coffeMachine: this); }
    public void pushTheButton() { state.pushTheButton(coffeMachine: this); }
    public void takeTheCup() { state.takeTheCup(coffeMachine: this); }
    public void returnTheCoin() { state.returnTheCoin(coffeMachine: this); }

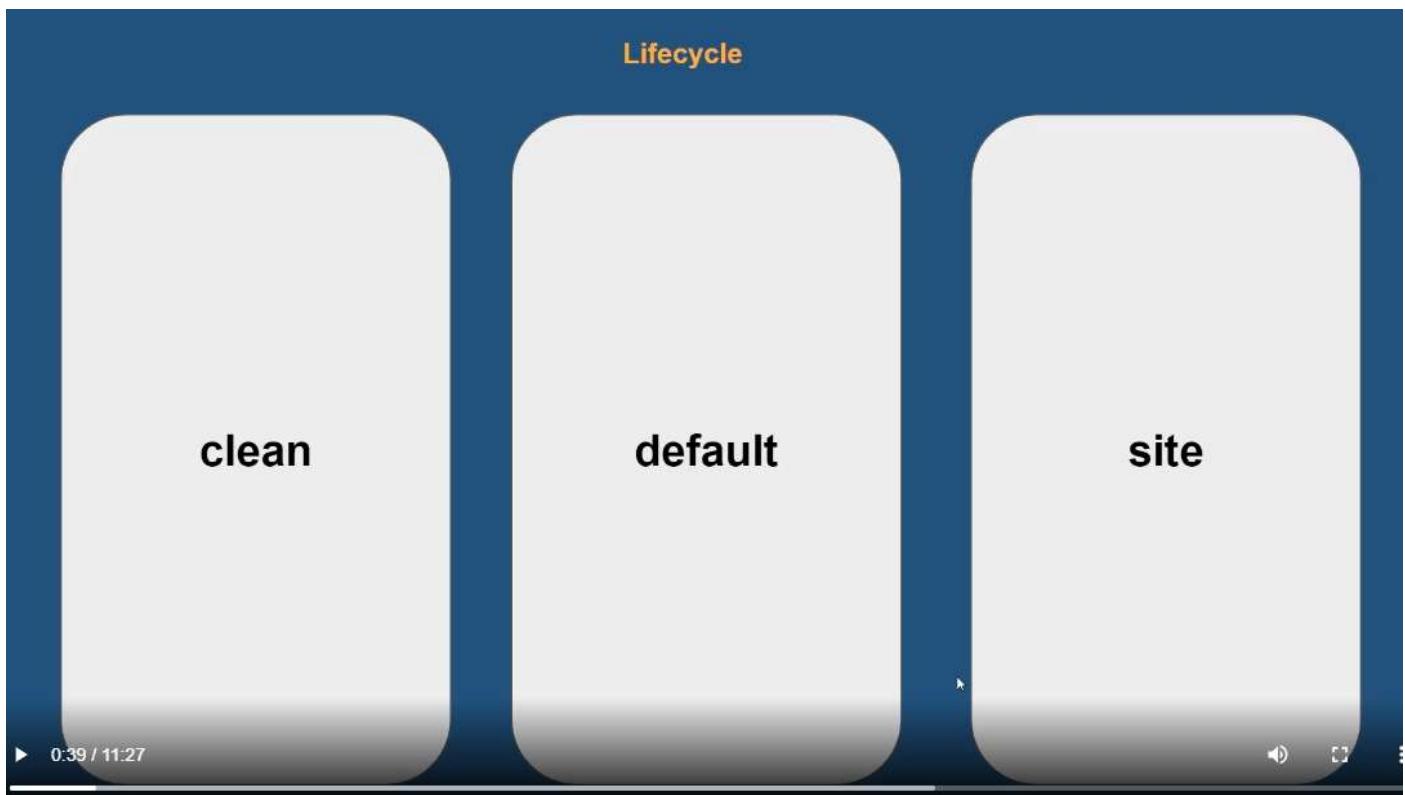
    void setState(State state) {
        this.state = state;
    }
}
```

# Maven

## Scope

- `compile` – wartość domyślna, zależność wymagana podczas kompilacji, testów, oraz wykonywania programu,
- `test` – zależność potrzebna tylko w fazie testów – taką wartość ustawiamy dla np. JUnit, ponieważ JUnit potrzebujemy w naszych aplikacjach tylko, gdy je testujemy – w wersji produkcyjnej naszego kodu JUnit nie jest potrzebne:
- `runtime` – zależność potrzebna dopiero na etapie działania programu – nie jest wymagana ani podczas kompilacji, ani testów – przykładem może być np. sterownik do obsługi bazy danych takiej jak Oracle.

## LifeCycles

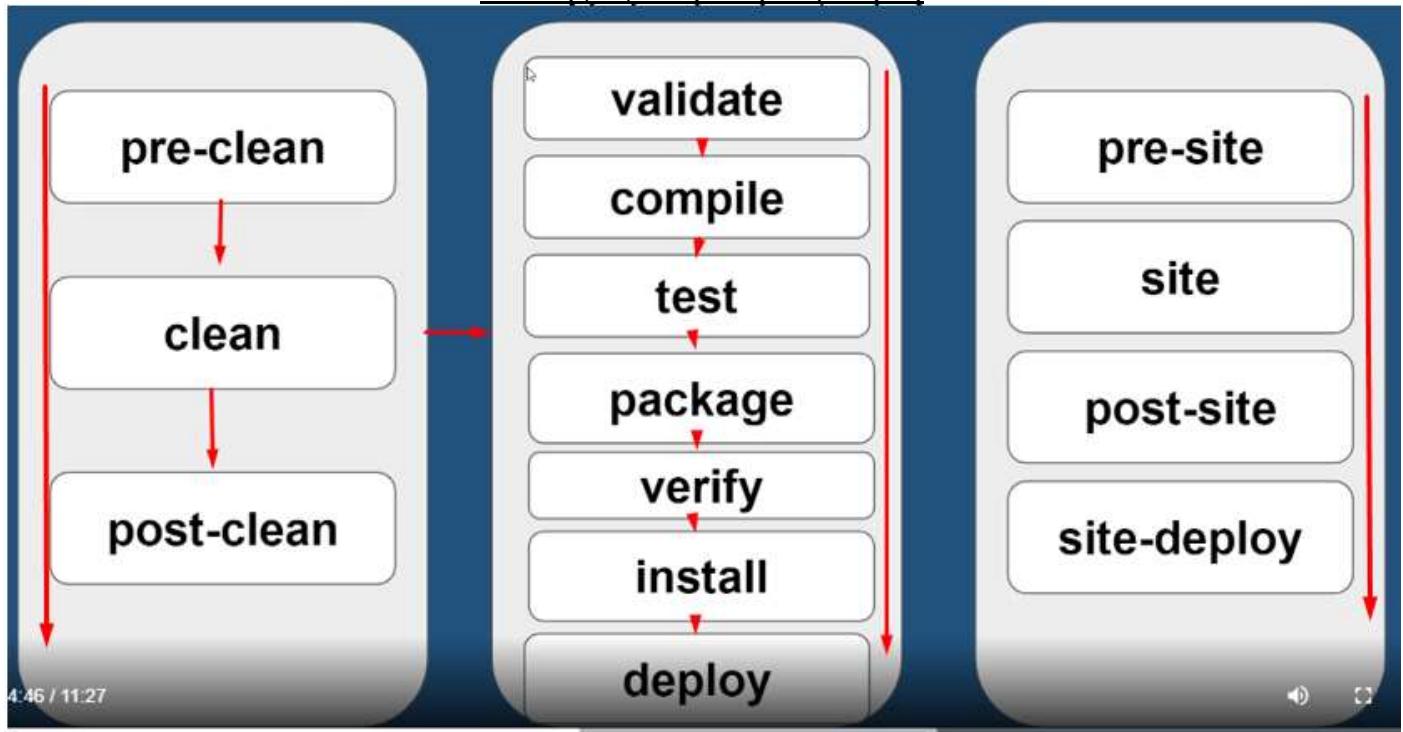


- `default` - cykl życia od walidacji projektu, przez jego budowę i instalację w lokalnym repozytorium, aż do wysłania go na zdalne repozytorium
- `clean` - cykl życia począwszy od kroku poprzedzającego usuwanie katalogu target, przez jego faktyczne usunięcie, aż do wykonania kroku "po wyczyszczeniu"
- `site` - cykl życia zaczynający się od kroku poprzedzającego stworzenie strony, przez jej realne stworzenie, aż do wysłania gotowej strony na repozytorium

site-deploy- strona z bazą .....

## Phases of Lifecycles Fazy działania maven

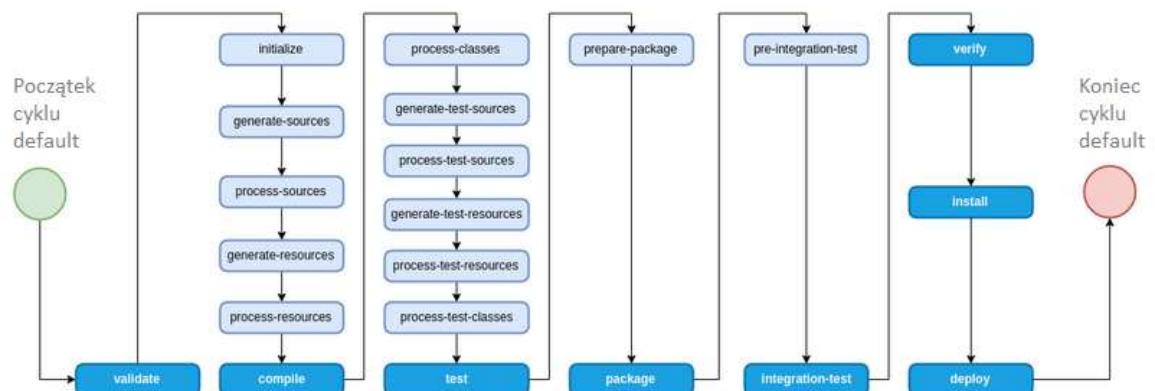
Składają się z cykli życia powyżej



### **Default**

- **Validate** – Sprawdza czy są wszystkie potrzebne biblioteki
- **Compile** – Kompilacja kodu
- **Test** – uruchamianie testów jednostkowych
- **Package** – Tworzenie pliku .jar
- **Verify** – Włączenie testów integracyjnych jeżeli istnieją
- **Install** – Wysłanie projektu (paczki jar) do repozytorium lokalnego **username/.m2**
- **Deploy** – Wysłanie projektu (paczki jar) do repozytorium zdalnego „firmowego”

✓ default - 21 faz



Komendy możemy mieszać tzn można użyć dwóch na raz np. mvn clean test.  
Z zachowaniem ich kolejności.

Wywołanie komendy niżej wywoła wszystkie jej powyższe. Np. Wywołanie mvn test, wywoła validate,compile,test

## Goals and Plugins

**Goal** jest to zadanie które może posiadać jakiś plugin

**Komendy :**

- **mvn pluginName:goalName** – wywołanie konkretnego Goal
- **Mvn pluginName:help** – pomoc powinna być w każdym pluiginie wraz ze spisem Goals

## Version

```
<version>1.0.0</version>
```

- **Pierwsza liczba** – Nowa wersja aplikacji nie jest kompatybilna wstępnie
- **Druga liczba** – Zmiany ulepszenia dodatkowe funkcjonalności nie psujące kompatybilności wstępnej
- **Trzecia liczba** – Poprawki bugów BugFix drobne

**Snapshot** – Wersje wciąż rozwijane **WorkInProgres**

Milestone m1-

RiskCandidate RC

## Properties - zmienne

- **\${zmienna}** - odwołanie do zmiennej
- **<zmienna></zmiennna>** - tworzenie zmiennej w properties

## Multimodule

Pom.xml tworzymy w folderze w którym robimy projekt

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <!-- parent pom -->
  <groupId>pl.chuj</groupId>
  <artifactId>multimodule</artifactId>
  <packaging>pom</packaging>
  <version>1.0</version>
</project>
```

Każdy moduł generujemy komenda

**mvn archetype:generate -DgroupId=pl.chuj -DartifactId=nazwamodulu**

## <dependencyManagement> oraz <pluginManagement>

ustawienia pom dla dzieci parent multimodule'om

```

<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.hibernate</groupId>
      <artifactId>hibernate-core</artifactId>
      <version>${hibernate.version}</version>
    </dependency>
    <dependency>
      <groupId>org.hamcrest</groupId>
      <artifactId>hamcrest</artifactId>
      <version>${hamcrest.version}</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.junit.jupiter</groupId>
      <artifactId>junit-jupiter</artifactId>
      <version>${junit.version}</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.mockito</groupId>
      <artifactId>mockito-junit-jupiter</artifactId>
      <version>${mockito.version}</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</dependencyManagement>

```

```

<build>
  <pluginManagement>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-surefire-plugin</artifactId>
        <version>2.22.2</version>
        <configuration>
          <argLine>
            --illegal-access=permit
          </argLine>
        </configuration>
      </plugin>
    </plugins>
  </pluginManagement>
</build>

```

## LOMBOK

<https://projectlombok.org/setup/maven>

<b>val</b>	Finally! Hassle-free final local variables.
<b>var</b>	Mutably! Hassle-free local variables.
<b>@NonNull</b>	or: How I learned to stop worrying and love the NullPointerException.
<b>@Cleanup</b>	Automatic resource management: Call your <code>close()</code> methods safely with no hassle.
<b>@Getter/@Setter</b>	Never write <code>public int getFoo() {return foo;}</code> again.
<b>@ToString</b>	No need to start a debugger to see your fields: Just let lombok generate a <code>toString</code> for you!
<b>@EqualsAndHashCode</b>	Equality made easy: Generates <code>hashCode</code> and <code>equals</code> implementations from the fields of your object..
<b>@NoArgsConstructor, @RequiredArgsConstructor and @AllArgsConstructor</b>	Constructors made to order: Generates constructors that take no arguments, one argument per final / non-nullfield, or one argument for every field.
<b>@Data</b>	All together now: A shortcut for <code>@ToString</code> , <code>@EqualsAndHashCode</code> , <code>@Getter</code> on all fields, and <code>@Setter</code> on all non-final fields, and <code>@RequiredArgsConstructor</code> !
<b>@Value</b>	Immutable classes made very easy.
<b>@Builder</b>	... and Bob's your uncle: No-hassle fancy-pants APIs for object creation!
<b>@SneakyThrows</b>	To boldly throw checked exceptions where no one has thrown them before!
<b>@Synchronized</b>	<code>synchronized</code> done right: Don't expose your locks.
<b>@With</b>	Immutable 'setters' - methods that create a clone but with one changed field.
<b>@Getter(lazy=true)</b>	Laziness is a virtue!
<b>@Log</b>	Captain's Log, stardate 24435.7: "What was that line again?"
<b>experimental</b>	Head to the lab: The new stuff we're working on.

# JPA /Hibernate/H2

**JPA – (JAVA PERSISTENCE API)** Nie wykonuje żadnej operacji jest to jedynie zbiór interfejsów które wymagają zaimplementowania

**HIBERNATE** należy do ORM co oznacza **Object/Relational Mapping** czyli mapowanie obiektów powiązanych z relacjami. **Hibernate implementuje interfejsy JPA.**

Aby uruchomić bazę H2 należy utworzyć plik persistence.xml w katalogu resources/ META-INF. W repo tworzymy entitymanagerfactory oraz entity manager

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence_2_0.xsd"
version="2.0">

    <persistence-unit name="persistence-test" transaction-type="RESOURCE LOCAL">

        <provider>org.hibernate.ejb.HibernatePersistence</provider>
        <properties>
            <property name="connection.driver_class" value="org.h2.Driver"/>
            <property name="hibernate.connection.url" value="jdbc:h2:./db/repository"/>
            <property name="hibernate.dialect" value="org.hibernate.dialect.H2Dialect"/>
            <property name="hibernate.hbm2ddl.auto" value="create-drop"/>
            <property name="hibernate.show_sql" value="true"/>
            <property name="hibernate.use_sql_comments" value="true"/>
            <property name="hibernate.format_sql" value="true"/>
        </properties>

    </persistence-unit>
</persistence>
```

```
<dependencies>          Maven Dependencies
    <dependency>
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
        <version>1.4.200</version>
    </dependency>
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>5.6.0.Final</version>
    </dependency>
</dependencies>
```



```
//uruchomienie bazy danych oraz utworzenie menagera
private static EntityManagerFactory factory = Persistence.createEntityManagerFactory( persistenceUnitName: "MyDataBase");
private static EntityManager em = factory.createEntityManager();

public static void main(String[] args) {
    GuestRepo guestRepo = new GuestRepo(em);
    ReservationRepo reservationRepo = new ReservationRepo(em);
```

W reposytorium piszemy nasze zapytania do bazy

```
public class GuestRepo {                                public class ReservationRepo {
    EntityManager em;
    EntityManager em;
    public GuestRepo(EntityManager em) {
        this.em = em;
    }
}
|
```

## Utworzenie tabeli

```
package pl.korman.databaseapk;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity @Entity Naza tabeli brana z nazwy klasy
public class Guest {
    @Id Oznaczenie pola nizej jako pola klucza tabeli
    @GeneratedValue(strategy = GenerationType.AUTO) Ustawienia automatycznego generowania ID
    private long ID;

    private String Name;
    private int Age;

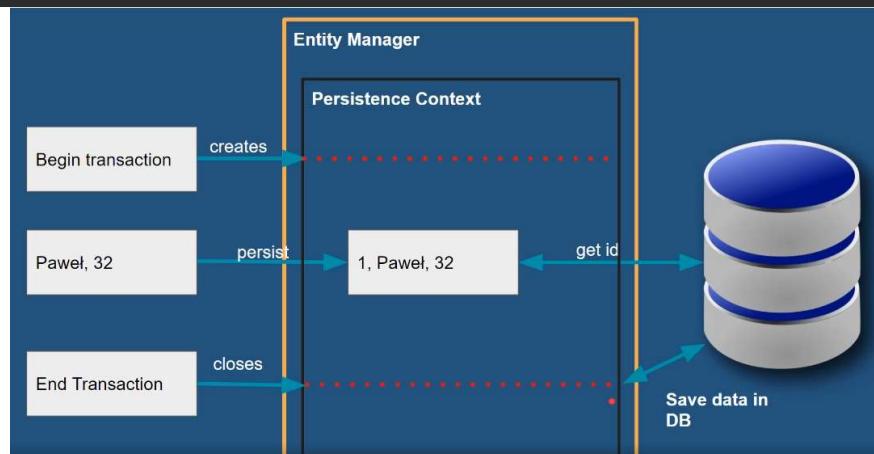
    public Guest(String name, int age) {
        Name = name;
        Age = age;
    }
}
```

## Insert do tabeli

```
public void createNewGuest(String name, int age) {
    System.out.println("---Opening New Transaction---");
    //Uruchomienie Transakcji z Baza
    EntityTransaction transaction = em.getTransaction();
    transaction.begin();

    //Utworzenie nowego obiektu oraz zapytania insert
    System.out.println("---Create---");
    Guest newguest = new Guest(name,age);
    em.persist(newguest);
    //pobranie id dziala już tutaj poniewaz em.persist() pobrało już z bazy id
    System.out.println("New Guest id " + newguest.getId());

    //Przekazanie insert do bazy danych oraz zamkniecie i zapisanie danych w bazie
    System.out.println("---Closing Transaction---");
    transaction.commit();
    System.out.println("---Closed and Saved---");
}
```



## Select

\*\*\*Wystarczy użyć samo em.find\*\*\*

```
public Guest selectGuestbyID(long ID){  
    em.clear();  
    // czyści presistence, w pierwszej kolejności guest szukany  
    // jest w presistence, jeżeli tu go nie ma szuka obiektu w bazie danych  
    // poprzez wykonanie crud select  
    return em.find(Guest.class, ID); // nazwa klasy + id  
}
```

## Update

Otwieramy transakcje--- aktualizujemy dane --- zapisujemy dane

```
public void updateAge(Guest guest, int newage) {  
    System.out.println("---Opening New Transaction---");  
    //Uruchomienie Transakcji z Baza  
    EntityTransaction transaction = em.getTransaction();  
    transaction.begin();  
    //ustawienie nowego parametru  
    guest.setAge(newage);  
    System.out.println("---Updating---");  
    transaction.commit();  
    System.out.println("---Updated---");  
}
```

## Delete

Otwieramy transakcje --- Usuwamy z presistence – Zapisujemy do Bazy

```
public void deleteGuest(Guest guest){  
    System.out.println("---Opening New Transaction---");  
    //Uruchomienie Transakcji z Baza  
    EntityTransaction transaction = em.getTransaction();  
    transaction.begin();  
  
    System.out.println("---Deleting and Update---");  
    //Utworzenie crud delete oraz usunięcie z presistence  
    em.remove(guest);  
    //Usunięcie z bazy danych  
    transaction.commit();  
    System.out.println("---Deleted and Updated---");  
}
```

## OneToOne

```

@Entity
public class Reservation {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private long ID;

    @OneToOne // Tutaj określamy relację wstrzykiwanej tabeli do nowej tabeli
    private Guest guest;

    public Reservation(Guest guest) {
        this.guest = guest;
    }
}

```

```

public void createReservation(Guest Guest){
    System.out.println("---Opening New Transaction---");
    //Uruchomienie Transakcji z Baza
    EntityTransaction transaction = em.getTransaction();
    transaction.begin();

    //Utworzenie nowej rezerwacji
    Reservation reservation = new Reservation(Guest);
    em.persist(reservation);

    System.out.println("---Closing Transaction---");
    transaction.commit();
    System.out.println("---Closed and Saved---");
}

```

## OneToMany

```

@Entity
public class Reservation {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private long ID;

    @OneToMany // Tutaj określamy relację wstrzykiwanej tabeli do nowej tabeli
    private List<Guest> guests;

    public Reservation(List<Guest> guestsList) {
        this.guests = guestsList;
    }
}

```

```

public void createReservation(List<Guest> Guests){
    System.out.println("---Opening New Transaction---");
    //Uruchomienie Transakcji z Baza
    EntityTransaction transaction = em.getTransaction();
    transaction.begin();

    //Utworzenie nowej rezerwacji
    Reservation reservation = new Reservation(Guests);
    em.persist(reservation);

    System.out.println("---Closing Transaction---");
    transaction.commit();
    System.out.println("---Closed and Saved---");
}

```

Hibernate:

```

create table Reservation (
    ID bigint not null,
    primary key (ID)
)

```

Hibernate:

```

create table Reservation_Guest (
    Reservation_ID bigint not null,
    guests_ID bigint not null
)

```

Hibernate:

```

alter table Reservation_Guest
    add constraint UK_9fe4lyw82mxL7uyhw93mhxn1h unique (guests_ID)

```

Hibernate:

```

alter table Reservation_Guest
    add constraint FKihdg9qwuk9rqpej1bt8vmnnk7
    foreign key (guests_ID)
    references Guest

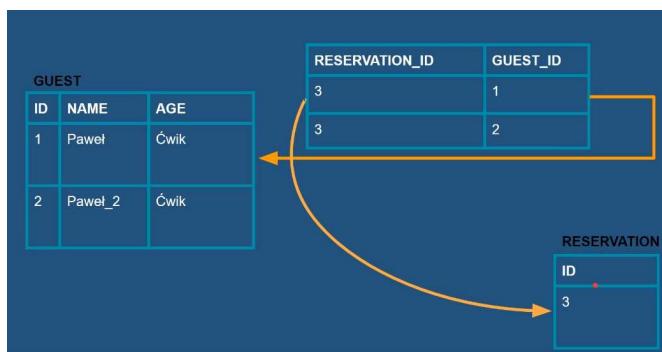
```

Hibernate:

```

alter table Reservation_Guest
    add constraint FKgsa1bdwud0sl4lvpvocyia48i
    foreign key (Reservation_ID)
    references Reservation

```



Hibernate tworzy tabelę pomocniczą do one to many

# Spring Data JPA

## ○ Spring Data :

### Spring Data

#### Spring Data JPA

Spring Data to **projekt Springa**, który zapewnia spójny model dostępu do danych. Ułatwia on korzystanie z **relacyjnych i nierelacyjnych** baz danych oraz **usług danych w chmurze**.

Projekt Spring Data składa się z wielu modułów, wśród których najbardziej popularnym jest właśnie Spring Data JPA, obejmujący zagadnienia z obszaru relacyjnych baz danych.

## ○ Spring Data JPA



Spring Data JPA to **moduł** projektu Spring Data, który dostarcza swoje interfejsy dostępu do interfejsów JPA.

Przykładem takiego interfejsu jest *CrudRepository*.