

JAVA - wprowadzenie

Część 1



**Fundusze
Europejskie**
Program Regionalny



URZĄD MARSZAŁKOWSKI
WOJEWÓDZTWA POMORSKIEGO

Unia Europejska
Europejski Fundusz Społeczny



Java - wprowadzenie

W Internecie znajdziemy ogromną ilość kursów i tutoriali dla osób początkujących, chcących nauczyć się programować w języku Java. Dlatego też niniejszy materiał nie jest obszernym kursem języka. Jest raczej tylko wprowadzeniem do nauki.

Znajdziemy tu:

- bardzo ogólne omówienie koncepcji i podstawowych elementów języka
- omówienie założeń testów jednostkowych na bazie frameworka JUnit
- zestaw ćwiczeń z podstawowych elementów z wykorzystaniem frameworka JUnit

Aby odróżnić poniższy kurs z całą resztą pozostałych, nie będziemy w ramach ćwiczeń pisać samodzielnych programów. Zamiast tego trzeba będzie wykorzystać technikę Test Driven Development - w ramach kursu jest dostarczony projekt z napisanymi testami modułowymi, natomiast wykonanie ćwiczeń polega na napisaniu funkcjonalności głównych klas w taki sposób, aby testy jednostkowe (modułowe) zaczęły przechodzić.

Dobrej zabawy!

Do uruchomienia projektu należy zaopatrzyć się w:

- Java JDK w wersji min. 8
- Środowisko programistyczne (zalecany JetBrains IntelliJ Idea, wystarczy wersja Community Edition)
- Dostęp do gitlab.com i dołączenie do projektu TrainingTA
- Opcjonalnie: git w wersji command line

W prezentacji omówiono pokrótce elementy języka wykorzystywane w zadaniach, jednakże dla pełnego zrozumienia konieczne będzie odszukanie dokładniejszego opisu tych elementów w Internecie.

Przyjmując założenie, że mamy sklonowane repozytorium projektu TrainingTA, możemy przełączyć się na branch `java_part1-ot`. Można to zrobić z poziomu IntelliJ Idea, poniżej sekwencja instrukcji do wykonania w command line:

1. sprawdzamy aktualny stan repozytorium

git status

jeżeli trzeba to:

git commit -am "dotychczasowa praca"

2. pobieramy aktualny stan zdalnego repozytorium

git fetch

3. przełączamy się na branch z zadaniami

git checkout java_part1-ot

4. tworzymy na tej podstawie własny branch

git checkout -b ImieNazwisko-java1

Uruchomienie (2)

Następnym krokiem będzie zaimportowanie projektu do IDE. W przypadku IntelliJ Idea wykonujemy:

1 Import project:



IntelliJ IDEA

Version 2019.1.3

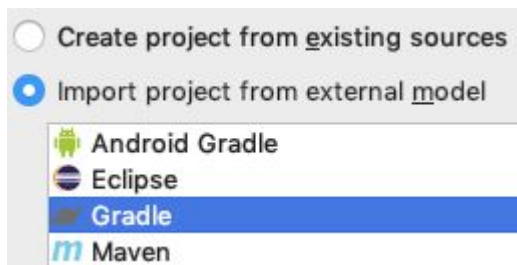
+ Create New Project

📁 Import Project

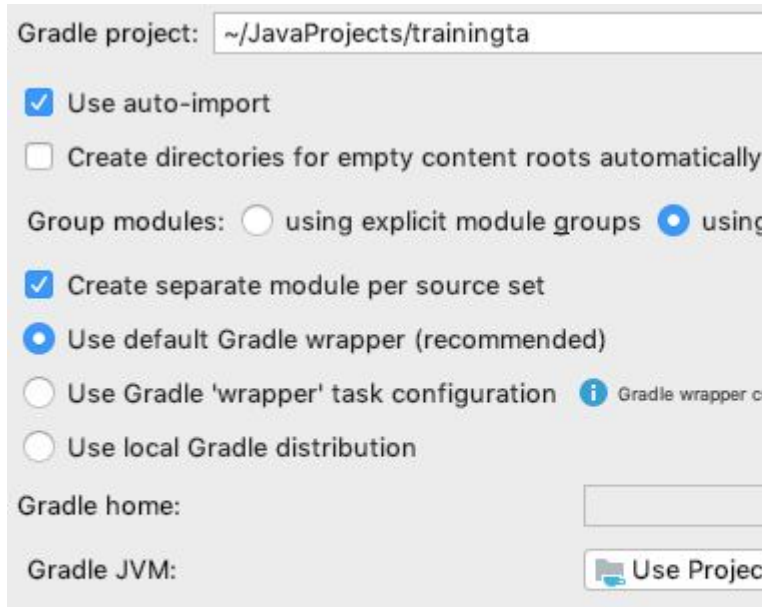
📁 Open

2 Wskazujemy katalog, do którego sklonowaliśmy projekt

3 Wybieramy model Gradle:



4 Zaznaczamy odpowiednie opcje (przydatny jest auto-import):



Uruchomienie (3)

Projekt w tym momencie powinien zostać zaimportowany i będzie gotowy do użycia.

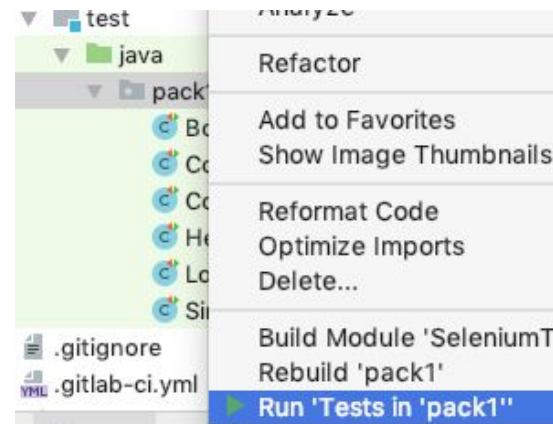
Pamiętajmy jednak, że w zależności od ustawień środowiska / wersji poszczególne kroki mogą wyglądać trochę inaczej.

Uruchamiać testy możemy na dwa sposoby:

1 Command line, za pomocą gradle:

gradlew test

2 Z poziomu IDE. Np. poprzez naciśnięcie prawego klawisza myszy na węźle `src/test/java/pack1` i wybraniu opcji **Run Tests in pack1**



Klasyka gatunku - Hello World!

Zadanie 1.1

Zwyczajowe pierwsze zadanie polega na wyświetleniu na ekranie napisu “Hello, World!”.

1. Otwórz klasę `src/main/java/pack1/HelloWorld`
2. Uzupełnij linię zawierającą instrukcję **return** `""` o odpowiedni tekst
3. Otwórz klasę `src/test/java/pack1/HelloWorldTest`
4. Wykonaj test przez naciśnięcie przycisku play obok testu lub z linii poleceń za pomocą komendy **gradlew test --tests HelloWorldTest**

Typy zmiennych

Typy proste

Zmienne w Javie zawsze mają określony typ danych, które mogą przechowywać.

Wyróżniamy 2 główne rodzaje typów zmiennych:

- typy proste (inaczej prymitywne)
- typy złożone - klasy

Język java zawiera następujące typy proste:

- Liczby całkowite: byte, int, long
- Liczby rzeczywiste: float, double
- Typ logiczny: boolean (może przyjmować tylko 2 wartości - true, false)
- Typ znakowy: char - pojedynczy znak

Typy proste - operacje arytmetyczne

Na typach prostych numerycznych możemy wykonywać operacje arytmetyczne.

Mamy do dyspozycji poniższe operatory:

$+$, $-$, $*$, $/$ - dodawanie, odejmowanie, mnożenie, dzielenie

$\%$ - reszta z dzielenia

Stopień precyzji przy operacjach matematycznych zależy od typu użytych zmiennych. Np. dla zmiennych typu `int` dzielenie będzie dawało wynik całkowity.

Natomiast dla zmiennych typu `double`, będzie liczbą rzeczywistą.

Operatory zwracające wartość logiczną

W programie możemy skonstruować równania i nierówności - zwrócą one wartość logiczną true, jeżeli równanie / nierówność jest spełniona oraz false w przeciwnym wypadku.

Np. `a + b == 9`

Możemy skorzystać z następujących operatorów:

`==`, `<`, `>`, `<=`, `>=` - odpowiednio porównanie czy równe, mniejsze, większe, mniejsze lub równe, większe lub równe.

Uwaga: `=` to operator przypisania, `==` operator porównania!

Samo równanie możemy również użyć w instrukcji return w postaci np.

```
return (a / 3 == 3);
```

Zadania - typy proste

Zadanie 2.1

Dodaj do siebie 2 liczby podane w parametrach. Zastąp instrukcję **return 0**; odpowiednią instrukcją **return ...** ; (skorzystaj z operatora +)

Zadanie 2.2 i 2.3

Analogicznie do zadania 2 wykonaj operację dzielenia. Nie przejmuj się dzieleniem przez 0 - nie musisz pisać specjalnej obsługi.

Zadanie 2.4

Sprawdź, czy podana w parametrze funkcji liczba jest parzysta.

Wykorzystaj operator % oraz operator porównania == - jego wynikiem jest właśnie wartość logiczna.

Co z dzieleniem przez 0

Analizując klasę SimpleTypesTests, zauważ ciekawą rzecz - przy dzieleniu liczb całkowitych jeden z testów próbuje podzielić przez 0, ale spodziewa się tzw. wyjątku (temat wyjątków będzie poruszony w drugiej części kursu). Natomiast przy dzieleniu liczb rzeczywistych tego wyjątku nie ma.

Dzieje się tak dlatego, że w przypadku liczb zmiennoprzecinkowych (rzeczywistych) nie mamy dokładnej ich reprezentacji komputerze a jedynie przybliżenie. Czyli liczba 0.0 - nie będzie równa dokładnie 0, a tylko bardzo zbliżona do tej wartości. Dzięki temu dzielenie przez 0.0 jest możliwe.



Typy złożone - klasy (1)

Java jest językiem obiektowym, co oznacza, że posługujemy się pojęciem obiektu i klasy.

Klasa jest definicją obiektu, **obiekt** jest konkretną instancją klasy.

Obiekt posiada pewne właściwości (cechy) potrafi też wykonywać określone operacje.

Klasę definiujemy za pomocą słowa kluczowego `class`.

Natomiast konkretny obiekt jest tworzony z użyciem operatora `new`:

```
Book book = new Book("Zbigniew Nienacki", "Pan Samochodzik i Templariusze");
```

```
public class Book
{
    private String author;
    private String title;

    public Book(String author, String title)
    {
        this.author = author;
        this.title = title;
    }

    public int getPagesCount()
    {
        return pagesCount;
    }

    public String getAuthor()
    {
        return author;
    }

    public String getTitle()
    {
        return String.format("%s\\", title);
    }
}
```

Typy złożone - klasy (2)

Zauważmy, że przykładowa klasa Book posiada 2 właściwości: author i title. Czyli każda książka będzie posiadała te cechy. Dodatkowo książka potrafi wykonać 2 czynności: getTitle i getAuthor, które odpowiednio zwracają tytuł i autora. Są to tzw. metody.

Klasa może mieć specjalną metodę, konstruktor, który jest używany tylko w połączeniu z operatorem new, tworzącym obiekt. Przygotowuje ona początkowe wartości obiektu, rezerwuje pamięć, itp. Jeżeli konstruktor nie jest jawnie zdefiniowany w kodzie źródłowym, kompilator automatycznie tworzy konstruktor domyślny, który nie przyjmuje parametrów.

W poniższym przykładzie tworzymy obiekt klasy Book, następnie korzystając z funkcji bibliotecznej dołączonej do Javy (System.out.println) wyświetlamy w konsoli linię zawierającą tytuł. Tytuł otrzymujemy poprzez wywołanie metody **getTitle()** na obiekcie **book**

```
Book book = new Book("Zbigniew Nienacki", "Pan Samochodzik i Templariusze");  
System.out.println(book.getTitle());
```

Klasa String

Z językiem Java otrzymujemy cały szereg przydatnych klas, które możemy wykorzystać do tworzenia własnych programów. Jedną z nich jest klasa String, która definiuje sposób przechowywania tekstu i niektóre operacje na nim.

Dla części klas nie jest niezbędne użycie operatora new, robi to za nas niejawnie kompilator. Dotyczy to także klasy String, którą możemy użyć w ten sposób:

```
String text = "Ala ma kota";
```

Mamy dostępnych cały szereg operacji, które mogą nam przekształcić tekst. Najlepiej się można o tym przekonać w IDE, jeżeli napiszemy nazwę zmiennej razem z kropką i poczekamy chwilę na podpowiedź.

```
String text = "Ala ma kota";  
text.  
m substrings(int beginIndex, int endIndex)  
m toCharArray()  
m toLowerCase()  
m toLowerCase(Locale locale)  
m toString()  
m toUpperCase()  
m toUpperCase(Locale locale)  
m trim()  
m subSequence(int beginIndex, int endIndex)  
m getClass()  
m notify()  
m notifyAll()  
^↓ and ^↑ will move caret down and up in the editor >>
```

Zadania - typy złożone

Zadanie 3.1

Zadeklaruj zmienną typu String i przypisz do niej zdanie, najlepiej wielokrotnie złożone ;) W instrukcji return zwróć tę zmienną, wykonując jednocześnie operację `.toUpperCase()`, co spowoduje zamianę znaków na wielkie litery.

Zadanie 3.2

Przygotowaliśmy dla Ciebie klasę Book, która znajduje się w pakiecie AdditionalClasses. Korzystając z podpowiedzi skonstruuj odpowiednio wartość zwracaną i zastosuj w instrukcji return.

Zadanie 3.3

Typy proste mają również swoje odpowiedniki w postaci klas - nazywa się to boxing. Czyli mamy np. typ prosty int i klasę Integer. Różnica jest taka, że obiekt klasy integer oprócz przechowywania wartości, potrafi również wykonać niektóre operacje na sobie. W zadaniu zwróć podaną wartość w postaci tekstu (czyli np. 2 jako "2"). Wskazówka: poszukaj metod rozpoczynających się od `.to...`

Instrukcje if ... else ...

Bardzo ważnymi instrukcjami sterującymi przebiegiem programu są instrukcje warunkowe.

if - jeżeli

```
if(warunek) {  
    wykonaj jakieś instrukcje    lub  
}  
  
if(warunek) {  
    wykonaj jakieś instrukcje  
}  
else {  
    w przeciwnym wypadku wykonaj te instrukcje  
}
```

W miejsce warunek podstawiamy wyrażenie, którego wynikiem jest wartość logiczna **true** lub **false**.

Operatory logiczne **and** i **or**

Bardzo często zdarza się, że warunek w instrukcji if jest w istocie złączeniem kilku warunków - np. jeżeli *spełnione jest a i b* lub *spełnione jest a i b*. Możemy wtedy posłużyć się operatorem logicznym.

Operator i (and) w Javie zapisujemy w postaci **&&**

Np. `if (a > b) && (a > c) { ... }`

Operator lub (or) w Javie zapisujemy w postaci **||**

Np. `if (a > b) || (a > c) { ... }`

Wyrażenie warunkowe

Jeżeli chcemy przypisać do zmiennej jedną wartość dla spełnionego warunku lub drugą dla niespełnionego, możemy posłużyć się tzw. wyrażeniem warunkowym. W istocie odpowiada ono instrukcji if ... else ...

Wyrażenie warunkowe przyjmuje postać:

Typ zmienna = warunek ? wartość_dla_true : wartość_dla_false;

```
int a = 2;  
int b = 5;  
int min = a <= b ? a : b;
```



```
int a = 2;  
int b = 5;  
int min;  
if (a <= b) {  
    min = a;  
} else {  
    min = b;  
}
```

Zadania - instrukcja if ... else ...

Zadanie 4.1

Napisz funkcję zwracającą większą liczbę z podanych wartości

Zadanie 4.2

Sprawdź, czy z podanych długości odcinków da się utworzyć trójkąt (suma długości dwóch boków trójkąta musi być większa od długości trzeciego boku).

Sprawdź, jaki to rodzaj trójkąta - równoboczny, równoramienny, dowolny.

Pamiętaj o operatorach logicznych `&&` i `||`.

Wskazówka: metoda może zawierać więcej niż jedną instrukcję `return`. Możesz to wykorzystać - np. jeżeli nie da się utworzyć trójkąta, nie ma sensu sprawdzać, czy jest równoboczny.

Tablice i listy

Pojedyncze zmienne czasami nie wystarczają, żeby przechować konieczne wartości w programie. Dlatego wprowadzono takie struktury danych jak np. tablice i listy. W uproszczeniu, tablica to ustawione po kolei wartości danego typu, np:

```
int tablica[3] = {1, 2, 3}
```



Musimy pamiętać, że tablice indeksujemy od 0, czyli, żeby uzyskać 1 element, należy odczytać go za pomocą:

```
int pierwszyElement = tablica[0] // pierwszyElement będzie miał wartość 1 w tym wypadku
```

Natomiast lista to są elementy połączone ze sobą, gdzie każdy element wskazuje następny

```
List<int> lista = new ArrayList<int>();
```



Kolejne elementy dokładamy za pomocą metody `.add` (np. `lista.add(4)`), natomiast pobieramy za pomocą metody `get` (np. `int 3 element = lista.get(2)`) również indeksując od 0.

Pętle for (1)

Jeżeli zachodzi potrzeba wykonania instrukcji wielokrotnie, to możemy użyć do tego celu pętli. Jednym z rodzajów pętli jest for, którą definiujemy następująco:

```
for (int zmienna = 0; zmienna < 5; zmienna++) { ... }
```

W nawiasie mamy 3 człony.

1. Stan początkowy - startowa wartość zmiennej
2. Warunek zakończenia pętli
3. Instrukcja sterująca (w tym wypadku zwiększająca wartość zmiennej o 1)

W powyższym przykładzie pętla wykona się 5 razy.

Pętle for (2)

Pętla for może występować też w postaci "dla każdego elementu" (for each). Stosujemy ją często do wykonania instrukcji dla każdego elementu ze zbioru (np. tablicy, listy).

Np.

```
int tablica[] = {1, 3, 5}
for (int element : tablica) {
    System.out.println(element);
}
```

Wyświetli po kolei wszystkie elementy z tablicy, czyli w tym wypadku 1, 3, 5

Zadania - pętle

Zadanie 5.1

Wypełnij tablicę wstawiając tam kolejne elementy od 2 do 6.
Weź pod uwagę, że tablicę indeksujemy od 0

Zadanie 5.2

Oblicz sumę elementów tablicy

Zadanie 5.3

Wyszukaj książki wskazanego autora. Mamy 2 listy: `foundBooks` - ją będziemy zwracać, oraz `booksInLibrary` - książki, które są w bibliotece. Przejdź za pomocą pętli `for` przez wszystkie książki w bibliotece, jeżeli dana książka pasuje do wskazanego autora, dodaj ją do listy zwracanej.

Możesz wykorzystać metodę:

`foundBooks.add(book)` ;



Zadania - czytnik książek

Zadanie 6.1

Mamy klasę reprezentującą czytnik książek. Przyjmuje ona w konstruktorze parametr - książkę, którą będziemy czytać. “Czytanie” polega na przesunięciu zakładki o zadaną ilość stron oraz zwróceniu numeru bieżącej książki.

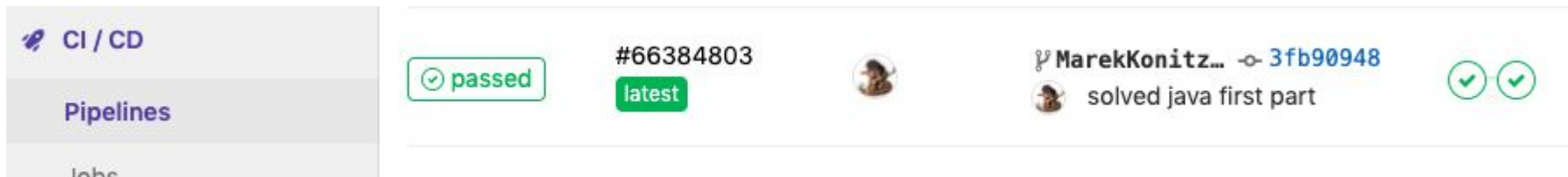
Zaimplementuj odpowiednio metodę **readSomePages()**

Push do GitLab

Po zakończeniu zadań (możemy również w trakcie) możemy umieścić rozwiązanie w gitlab.com

1. Pierwszy push:
git push --set-upstream origin ImieNazwisko-java1
2. Kolejny push:
git push

Spowoduje to automatyczny build w systemie CI i będziemy mogli zaobserwować rezultat. Jeżeli wszystko pójdzie dobrze:



The screenshot shows the GitLab CI/CD Pipelines interface. On the left, there is a sidebar with 'CI / CD' and 'Pipelines' sections. The main area displays a pipeline with ID '#66384803' and a status of 'passed' (indicated by a green checkmark icon). Below the status, it says 'latest'. To the right, there is a user profile picture and the name 'MarekKonitz...' followed by a commit hash '3fb90948'. Below this, it says 'solved java first part'. At the end of the pipeline, there are two green checkmark icons indicating success.

Jeżeli jakieś testy nie przejdą, dostaniemy maila ze stosowną informacją.