



# INVEST IN POMERANIA ACADEMY



Fundusze  
Europejskie  
Program Regionalny



Rzeczpospolita  
Polska



URZĄD MARSZAŁKOWSKI  
WOJEWÓDZTWA POMORSKIEGO

Unia Europejska  
Europejski Fundusz  
Rozwoju Regionalnego





# Podstawy JSE



Fundusze  
Europejskie  
Program Regionalny



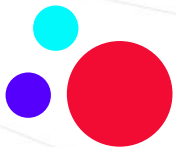
Rzeczpospolita  
Polska



URZĄD MARSZAŁKOWSKI  
WOJEWÓDZTWA POMORSKIEGO

Unia Europejska  
Europejski Fundusz  
Rozwoju Regionalnego





# HELLO

## Tomasz Lisowski

Software developer  
Scrum Master  
IT trainer





# Agenda

- wprowadzenie
- klasy
- obiekty
- metody
- typy danych
- instrukcje sterujące





infoShare  
ACADEMY

- język maszynowy – **bytecode**
- zapis binarny – 100001011001
- Java – obiektowy język programowania
- kompilacja
- zamian plików .java na .class
- JVM



# Wersjonowanie

Version	Release date	Example features
Java SE 11 <b>LTS</b>	September 2018	<ul style="list-style-type: none"><li>- HTTP Client API</li><li>- String API changes</li></ul>
Java SE 17 <b>LTS</b>	September 2021	<ul style="list-style-type: none"><li>- Pattern matching for switch</li><li>- Sealed classes</li></ul>
Java SE 18	March 2022	<ul style="list-style-type: none"><li>- Code snippet in Java API Documentation</li><li>- UTF-8 encoding by default</li></ul>
Java SE 19	September 2022	<ul style="list-style-type: none"><li>- Structured concurrency</li><li>- Foreign function (outside the JVM)</li></ul>
Java SE 20	March 2023	
Java SE 21 <b>LTS</b>	September 2023	

- **JVM – Java Virtual Machine**

“procesor” wykonujący skompilowany kod Javy i zarządzający pamięcią

- **JRE – Java Runtime Environment**

zawiera JVM oraz klasy niezbędne do uruchomienia programu Java

- **JDK – Java Development Kit**

zawiera JRE oraz narzędzia do implementacji i kompilacji



- podstawowy element składowy aplikacji
- typ danych
- konkretna definicja pewnego 'bytu' (instrukcja / przepis)
- zawiera pola i metody

```
public class Main {  
    public static void main(String[] args) {  
        System.out.println("infoShareAcademy - Java SE");  
    }  
}
```



właściwości / pola	funkcjonalności / metody
model	call()
ID	answer()
producer	sendSMS()
contact list	takePhoto()
OS version	



# Klasa - pole

- dana cecha naszej klasy
- może być dowolnego typu (klasy)
- może być ich wiele lub ani jednego

```
public class Car {  
    public String name;  
    public int maxSpeed;  
}
```

modyfikator dostępu → `public`

typ danych → `String`

nazwa (dowolna) → `maxSpeed`



# Klasa - metoda

- funkcjonalność klasy  
(tu logika - nie piszemy kodu poza metodami!)
- zwraca wybrany typ danych  
(lub nic - wtedy oznaczamy jako **void**)
- mogą przyjmować parametry

modyfikator  
dostępu

```
public void method1() {  
    System.out.println("Ta metoda nie zwraca nic!");  
}  
  
public int getNumberTwo() {  
    return 2; //ta metoda zwraca liczbę całkowitą 2  
}  
  
public int sum(int a, int b) {  
    return a + b; //ta metoda zwraca sumę dwóch parametrów  
}
```

typ  
zwracanych  
danych  
  
nazwa  
(dowolna)



# main()

- metoda main() to główna metoda, od której rozpoczyna się uruchamianie programu przez JVM

```
public class Main {  
    public static void main(String[] args) {  
        System.out.println("infoShareAcademy - Java SE");  
    }  
}
```



- instancja klasy
- konkretny, fizyczny byt na podstawie definicji
- klasa (definicja) opisuje obiekt (instancja)

```
public class Car {  
    public String name;  
    public int maxSpeed;  
}
```

```
Car myCar = new Car();
```

# **klasa vs obiekt**



# Konstruktor

```
public class Car {  
    public String name;  
    public int maxSpeed;  
  
    public Car() {  
        name = "default name";  
        maxSpeed = 150;  
    }  
}
```

- metoda tworząca obiekt
- konstruktor domyślny
- konstruktor parametrowy
- słowo kluczowe **this**

przypisanie do pola  
**name** (pola klasy)  
wartości z parametru

```
public class Car {  
    public String name;  
    public int maxSpeed;  
  
    public Car() {  
        name = "default name";  
        maxSpeed = 150;  
    }  
  
    public Car(String name) {  
        this.name = name;  
    }  
}
```



```
int number;  
String text;  
  
//domyślny konstruktor, nie trzeba go pisać jawnie  
Menu() {  
}  
  
//parametrowy konstruktor  
public Menu(int number, String text) {  
    this.number = number;  
    this.text = text;  
}
```



- stwórz klasę **MyClass**, która ma dwa pola:
    - liczbowe (int) o nazwie **number**
    - tekstowe (String) o nazwie **text**
  - w metodzie **main()** stwórz 2 obiekty typu **MyClass**
  - uzupełnij wszystkie pola wartościami
  - wypisz wartości pól na ekran
- \*wykorzystaj do tego metodę **System.out.println()**



## Ćwiczenie 1a

- stwórz klasę **Engine**, która ma dwa pola **Integer** (*power, capacity*)
  - w metodzie *main()* stwórz 2 obiekty typu **Engine**
  - uzupełnij wszystkie pola wartościami
  - wypisz wartości pól na ekran
- \*wykorzystaj do tego metodę **System.out.println()**



## Ćwiczenie 1b

- zmodyfikuj klasę **Car** – dodaj do niej pole typu **Engine**
- w metodzie *main()* stwórz obiekt typu Car
- uzupełnij wszystkie pola wartościami
- wypisz wynik analogicznie jak w zadaniu 1a



## Ćwiczenie 2a

- w klasie **Car** stwórz metodę, która wypisze na ekran nazwę obiektu (czyli wartość pola *name*)
- wywołaj metodę na obiekcie powstałym w zadaniu 1b

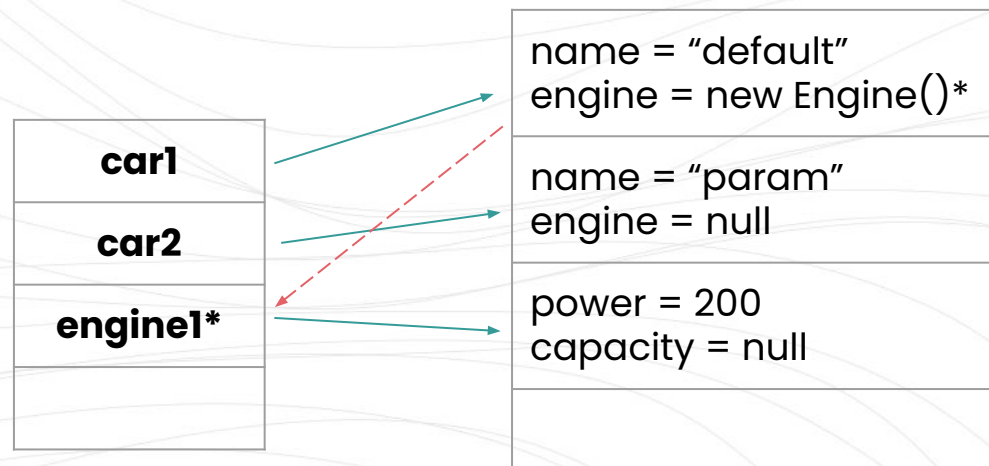


## Ćwiczenie 2b

- w klasie **Engine** stwórz 2 metody, które wypiszą na ekran moc oraz pojemność danego obiektu (pola *power/capacity*)
- w metodzie *main()* stwórz nowy obiekt typu **Engine**
- uzupełnij go wartościami
- wywołaj utworzone metody na powyższym obiekcie



```
Car car1 = new Car();  
Car car2 = new Car("param");
```





# Metody statyczne

- oznaczone słowem kluczowym ***static***
- można je wywołać bezpośrednio na klasie
- nie wymagają stworzenia instancji obiektu
- <https://www.javatpoint.com/static-keyword-in-java>

```
public class Menu {  
    public static void staticMethod() {  
        System.out.println("This is static method!");  
    }  
  
    public void nonStaticMethod() {  
        System.out.println("This is NON static method!");  
    }  
}
```

---

```
public static void main(String[] args) {  
    Menu menu = new Menu();  
    menu.nonStaticMethod();  
  
    Menu.staticMethod();  
}
```



## Ćwiczenie 3

- utwórz nową klasę **StaticExample**
- stwórz w niej dwie metody
- obydwie typu **void**
- jedna z nich niech będzie metodą statyczną
- każda z nich niech ma tylko jedną linię kodu:  
wypisanie na ekran informacji, czy jest statyczna, czy nie
- wywołaj obydwie metody w metodzie *main()*



# Zasięg widzenia pól

- pola klasy widoczne dla wszystkich jej metod
- ..lub nawet na zewnątrz
- pola w metodzie widoczne tylko dla niej
- tworzony obiekt wewnątrz metody “żyje” tylko w niej





# Zasięg widzenia pól

```
int number;  
String text;
```



pola klasy

```
public void method() {  
    int otherNumber;  
  
    number = 1;  
    otherNumber = 2;  
}
```



pole metody

```
public void otherMethod() {  
    number = 2;  
    otherNumber = 3;  
}
```



błąd - *otherNumber* nie jest  
widoczne w tej metodzie



# Modyfikatory dostępu

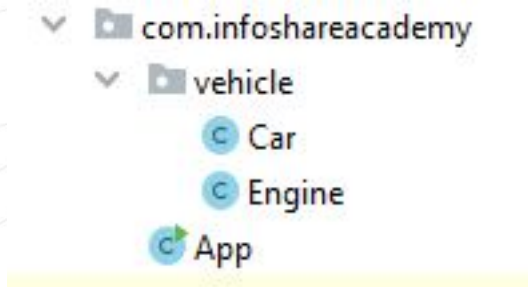
- słowa kluczowe, określające poziom dostępności
- **public** – dostęp do elementu dla wszystkich klas
- **protected** – dostęp tylko dla klas dziedziczących (lub z pakietu)
- **private** – brak widoczności elementów poza klasą
- *default* – dostęp pakietowy, nie istnieje takie słowo kluczowe
- package-private -> *publiczne w pakiecie, prywatne na zewnątrz*
- dobra praktyka – wszystkie pola prywatne

- klasy są pogrupowane w pakiety
- struktura hierarchiczna
- pakiety -> katalogi
- klasy -> pliki
- implementacja klasy znajduje się zawsze w określonym pakiecie
- informuje o tym słowo kluczowe ***package***

np. klasa znajduje się w pakiecie ***infoshareacademy***, który znajduje się w pakiecie ***com***

```
package com.infoshareacademy;
```

- klasy będące w tym samym pakiecie “znają się”
- klasy spoza pakietu należy zaimportować
- słowo kluczowe ***import***



```
package com.infoshareacademy;  
  
import com.infoshareacademy.vehicle.Car;  
import com.infoshareacademy.vehicle.Engine;
```





# getter i setter

- dostęp do wartości pól powinien odbywać się poprzez metody
- metody **set(Type value)** do ustawiania wartości
- metody **get()** do odczytania
- wszystkie pola powinny być oznaczone jako **private**
- getter i setter jako **public**
- metody dostępne są opcjonalne



```
private int number;  
  
public int getNumber() {  
    return number;  
}  
  
public void setNumber(int number) {  
    this.number = number;  
}
```



## Ćwiczenie 4a

- rozbuduj klasę **MyClass** o odpowiednie gettery i settery
- oznacz pola jako **private**
- ustaw wartości obiektów za pomocą setterów
- wyświetl ustawione wartości za pomocą getterów



## Ćwiczenie 4b

- rozbuduj klasy **Car** i **Engine** o odpowiednie gettery i settery
- wszystkie pola powinny mieć dostęp prywatny
- stwórz obiekt **Car** i **Engine** w metodzie *main()*
- ustaw wartości pól dla powyższych obiektów
- wyświetl ustawione powyżej wartości





# Typy danych

- wszystko jest obiektem
- typy reprezentują różne wartości, przechowywane w zmiennych
- np. tekstowe, liczbowe, logiczne
- różne formaty dat
- każda klasa jest typem danych





# Typy proste

- typy proste (primitive) nie są instancjami obiektów
- reprezentują podstawowe typy danych
- zawsze mają jakąś wartość
- nie możemy wykonywać na nich metod

```
int liczbaCalkowita;  
long duzaLiczbaCalkowita;  
double liczbaZmienniePrzecinkowa; //64bit  
float kolejnaLiczbaZmienniePrzecinkowa; //32bit  
boolean prawdaFalsz;  
char znak;
```

Data Type	Default Value	Default size
boolean	false	1 bit
char	'\u0000'	2 byte
byte	0	1 byte
short	0	2 byte
int	0	4 byte
long	0L	8 byte
float	0.0f	4 byte
double	0.0d	8 byte



# Typy obiektowe

- klasy – możemy tworzyć swoje typy obiektowe
- mogą mieć dowolne zachowania (metody)
- mogą nie mieć wartości -> NULL

```
Integer liczbaCalkowita;  
Long duzaLiczbaCalkowita;  
Double liczbaZmiennoPrzecinkowa;  
Float kolejnaLiczbaZmiennoPrzecinkowa;  
Boolean prawdaFalsz;  
String napis;
```



## Ćwiczenie 5

- stwórz nową klasę z dwoma polami
  - jedno typu **int**, drugie typu **Integer**
  - w metodzie *main()* stwórz nowy obiekt tej klasy
  - wypisz wartości obydwu pól
- 
- stwórz dwie zmienne liczbowe, typu **int** i **Integer**
  - porównaj metody, które możesz na nich wykonać (czyli co podpowiada IntelliJ, gdy napiszemy nazwę zmiennej i zrobimy "kropkę")

*zmiennaInt. ?*

*zmiennaInteger. ?*



- zmiana typu danych na inny
- np. dzielenie dwóch liczb całkowitych:  $3 / 2$
- rzutowanie poprzez "nawias" (dla typów prostych)
- rzutowanie za pomocą metody (dla obiektów)

```
int liczbaA = 10;  
int liczbaB = 3;  
//wynik dzielenia nie jest liczbą całkowitą  
  
int wynik = liczbaA / liczbaB;  
// zmienna wynik = 3
```



```
int liczbaA = 10;  
int liczbaB = 3;  
//wynik dzielenia nie jest liczbą całkowitą  
  
double wynikInt = liczbaA / liczbaB;  
// zmienna wynik = 3.0
```

```
int liczbaA = 10;  
int liczbaB = 3;  
  
double liczbaC = (double) liczbaA;  
//liczbaC = 10.0  
double liczbaD = (double) liczbaB;  
//liczbaD = 3.0  
  
double wynikInt = liczbaA / liczbaB;  
//wynikInt = 3.0  
double wynikDouble = liczbaC / liczbaD;  
//wynikDouble = ?
```

```
Double doubleA = a.doubleValue();
```

```
Double doubleB = b.value
```

m doubleValue()	double
m byteValue()	byte
m floatValue()	float
m intValue()	int
m longValue()	long
m shortValue()	short
Press Ctrl+. to choose the selected (or first) suggestion and insert a dot afterwards <a href="#">Next Tip</a>	
⋮	

```
Double doubleC = Double.parseDouble( s: "12.5");
```



# String

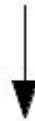
- obiektowy typ tekstowy
- **immutable** – nie można go zmienić, “zmiana” powoduje utworzenie nowej instancji
- posiada zestaw metod do operacji na tekście  
np. *compare()*, *concat()*, *split()*, *length()*, *replace()*, *substring()*

```
char[] chars = {'i', 'n', 'f', 'o', 'S', 'h', 'a', 'r', 'e'};  
String s = new String(chars);  
  
String s2 = "infoShare";
```



- **immutable** – modyfikacje wymagają przypisania

```
String s = "infoShare";  
s.concat("Academy");  
System.out.println(s);
```



infoShare

```
String s = "infoShare";  
s = s.concat("Academy");  
System.out.println(s);
```



infoShareAcademy





# Operatory

- działania, które można wykonywać na obiektach
- np. operacje matematyczne lub logiczne
- operatory porównania lub przypisania

Operator type	Example
Unary	<i>i++</i> , <i>i--</i> , <i>++i</i> , <i>--i</i> , <i>!</i>
Arithmetic	<i>*</i> , <i>/</i> , <i>%</i> , <i>+</i> , <i>-</i>
Relational	<i>&lt;</i> , <i>&gt;</i> , <i>&lt;=</i> , <i>&gt;=</i> , <i>instanceof</i> , <i>==</i> , <i>!=</i>
Logical	<i>&amp;&amp;</i> , <i>  </i>
Assignment	<i>=</i> , <i>+=</i> , <i>-=</i> , <i>*=</i> , <i>/=</i>



## Ćwiczenie 6

- utwórz kilka obiektów różnych typów
- nadaj im wartości, wykorzystując różne operatory  
np. *Integer i = 5 \* 10;*
- wykorzystaj też inne obiekty do utworzenia kolejnych  
np. *Integer i = 2;*  
*Integer j = i + 3;*

- każdy wyraz "oddzielamy" dużą literą
- **klasy** – rzeczowniki, zaczynamy dużą literą
- **metody** – czasowniki, zaczynamy małą literą
- **zmienne** – zaczynamy małą literą
- **stałe** – tylko duże litery, wyrazy "oddzielamy" znakiem "\_"

```
class MyClass {  
    Integer myVariable;  
    final Integer MY_CONSTANT = 2;  
  
    void myMethod() {  
        myVariable = MY_CONSTANT + 1;  
    }  
}
```



## Ćwiczenie 7a

- utwórz klasę **Student** z polem *name* typu **String**
- konstruktor powinien wymusić podanie tego pola
- utwórz w niej metodę *printName()*, która wypisze wartość pola *name*
- w metodzie *main()* stwórz kilka obiektów typu **Student** i wywołaj na nich powyższą metodę



## Ćwiczenie 7b

- w klasie **Student** stwórz nową metodę o nazwie *getExamDate()*, która zwraca typ **LocalDate**
- metoda powinna wylosować dwie liczby, *month* (1-12) oraz *day* (1-31)
- następnie stworzyć obiekt **LocalDate** dla obecnego roku oraz wylosowanego wcześniej miesiąca i dnia
- zwróć tak stworzony obiekt daty z tej metody
- w metodzie *main()* stwórz obiekt **Student**
- wypisz wylosowaną dla tego obiektu datę egzaminu



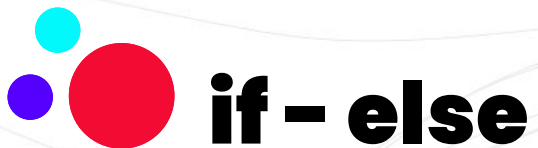




# if - else

- podstawowa operacja – instrukcja wyboru
- if = jeżeli
- *jeżeli warunek jest spełniony, to wykonaj instrukcje*

```
double wynik = liczbaA/liczbaB;  
  
if (wynik > 0) {  
    return "Liczba dodatnia";  
}
```



# if - else

- warunek *if* można łączyć z *else*
- *else* wykonywane, gdy pierwszy warunek jest niespełniony
- można zagnieżdżać oraz łączyć instrukcje *if - else*

```
if (wynik > 0) {  
    return "Liczba dodatnia";  
}  
else if (wynik == 0) {  
    return "Liczba 0";  
}  
else {  
    return "Liczba ujemna";  
}
```



## Ćwiczenie 8

- stwórz obiekt *car1* (*name* = "car1", *maxSpeed* = 100)
- stwórz obiekt *car2* (*name* = "car2", *maxSpeed* = 200)
- stwórz instrukcję warunkową, która wypisze nazwę pojazdu o większej wartości pola *maxSpeed*
  
- \*stwórz metodę, która obiektowi o większej wartości pola *maxSpeed* przypisze nową wartość pola *name* -> "faster car"
- \*wypisz wartość *name* obydwu obiektów



# Potrójny operator if

- jednoliniowa operacja zastępująca *if - else*
- *(warunek\_logiczny) ? pierwsze\_wyrazenie : drugie\_wyrazenie;*

```
int c = (a > b) ? a : b;
```

```
if (a > b) {  
    c = a;  
} else {  
    c = b;  
}
```



- “wielowarunkowy if”
- switch pobiera parametr i sprawdza dowolną liczbę warunków
- <https://stormit.pl/switch-case/>

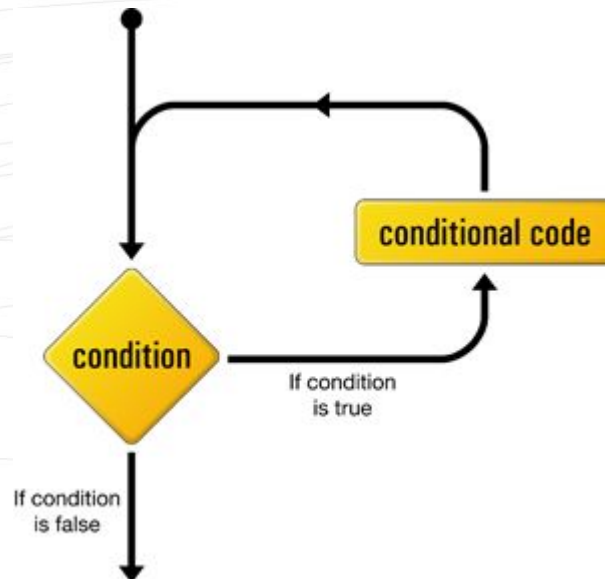
```
switch(liczba){  
    case 1:  
        jakieś_instrukcje_1;  
        break;  
    case 2:  
        jakieś_instrukcje_2;  
        break;  
    ...  
    default:  
        instrukcje, gdy nie znaleziono żadnego pasującego przypadku  
}
```



## Ćwiczenie 9

- stwórz metodę, która pobiera liczbę całkowitą
- wykorzystaj instrukcję *switch* do sprawdzenia, czy liczba z parametru jest parzysta
- stwórz analogiczną metodę z wykorzystaniem *if - else*
- która wersja jest lepsza?

- podstawowa operacja – cykliczne wykonanie danych instrukcji
- niewiadoma ilość wykonań
- ...lub ściśle określona
- można przerwać lub pominąć dany obieg pętli
- warunkiem wykonania pętli jest dana wartość logiczna (warunek)





# while

- wykorzystywana, gdy nie znamy ilości obiegów pętli
- ...ale znamy warunek jej zakończenia
- pętla while może wykonać się nieskończenie wiele razy
- albo wcale, gdy warunek już na starcie nie jest spełniony

```
int liczba = -5;  
while(liczba < 0) {  
    liczba++; //liczba = liczba + 1;  
}
```



## Ćwiczenie 10

- stwórz zmienną liczbową o wartości dodatniej
- stwórz pętlę while, która “kręci się” dopóki powyższy parametr jest większy od 0
- wewnątrz pętli wypisz wartość zmiennej, a następnie zmniejsz ją o 1





# do..while

- inna wersja pętli while
- pętla do..while zawsze wykona się co najmniej jeden raz
- warunek jest sprawdzany dopiero na zakończenie obiegu pętli

```
int liczba = 5;  
do {  
    liczba++; //liczba = liczba + 1;  
} while(liczba < 0);
```



## Ćwiczenie 11

- zmodyfikuj poprzednie zadanie:
  - odwróć warunek (pętla “kręci się” jeśli parametr jest mniejszy od 0)
  - zastosuj pętlę do..while

- zazwyczaj znamy liczbę iteracji w pętli
- 3 parametry
  - **wyrażenie początkowe** -> np. `int i = 0`
  - **warunek** -> np. `i < 5`
  - **modyfikator** -> np. `i++`

```
for (int i = 0; i < 5; i++) {  
    System.out.println("i: " + i);  
}
```



## Ćwiczenie 12

- stwórz zmienną liczbową o wartości 10
- stwórz pętlę for, która “kręci się” od 0
- warunek: licznik pętli mniejszy od utworzonej zmiennej
- z każdym obiegiem pętli zwiększ licznik o 1
- wypisz wartość licznika pętli w każdym obiegu



# break : continue

- instrukcje manipulujące działaniem pętli
- **break**
  - przerwanie pętli
- **continue**
  - pominięcie obecnej iteracji

```
int liczba = -5;
while(liczba < 0) {
    if (liczba == 2) {
        continue;
    }

    if (liczba == 3) {
        break;
    }

    liczba++; //liczba = liczba + 1;
}
```





## Ćwiczenie 13

- napisz metodę przyjmującą jeden parametr typu **int**
- w metodzie napisz pętlę iterującą od 0 do wartości tego parametru
- pomiń każdą liczbę parzystą
- przerwij pętlę, jeśli liczba jest podzielna bez reszty przez 11
- wypisz na konsolę pozostałe liczby

np. parametr = 14

wynik: 1, 3, 5, 7, 9

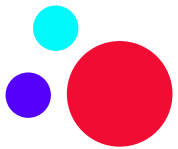


# Podstawy JSE – materiały dodatkowe

- <https://javastart.pl/baza-wiedzy/java-podstawy-jezyka>
- <https://www.tutorialspoint.com/java/>
- <https://docs.oracle.com/javase/tutorial/>

# Q&A

[infoShareAcademy.com](https://infoShareAcademy.com)



**Thanks!**



[tomasz.lisowski@proton.me](mailto:tomasz.lisowski@proton.me)



[www.infoshareacademy.com](http://www.infoshareacademy.com)





# INVEST IN POMERANIA ACADEMY



Fundusze  
Europejskie  
Program Regionalny



Rzeczpospolita  
Polska



URZĄD MARSZAŁKOWSKI  
WOJEWÓDZTWA POMORSKIEGO

Unia Europejska  
Europejski Fundusz  
Rozwoju Regionalnego

