



**Faculty
of Physics**

WARSAW UNIVERSITY OF TECHNOLOGY



Programowanie Obiektowe Java

Małgorzata Janik

Zakład Fizyki Jądrowej
malgorzata.janik@pw.edu.pl



Regulamin

Warunki zaliczenia
Wykład
Laboratoria
Projekt



Programowanie Obiektowe

- Strona internetowa przedmiotu

<http://java.fizyka.pw.edu.pl/>

- Wykłady

15 h, 2 h przez pierwszą połowę semestru

- Laboratoria

30 h, 2h co tydzień

- Projekt

realizowany w domu, trzy zajęcia projektowe sprawdzające postępy prac



Warunki zaliczenia

- **Projekt**
50 pkt
- **Laboratoria: zajęcia punktowane**
11 x 5 pkt = 55 pkt
- **Laboratoria: zajęcia weryfikujące wiedzę**
1 x 15 pkt = 15 pkt
- **Wykład**
+0.5 oceny w górę
- **Warunek zaliczenia:**
 - > 50% z projektu
 - > 50% punktów

Ilość punktów	Ocena
60 - 72	3
72.5 - 84	3.5
84.5 - 96	4
96.5 - 108	4.5
> 108	5



Wykład

- **Poniedziałek 16:15 – 18:00**
2 h, pierwsza połowa semestru
- **7 wykładów :**
obecność na minimum 6: +0.5 oceny w górę
- **Daty (możliwość zmiany)**
 - 19.02
 - 26.02
 - 5.03
 - 12.03
 - 19.03
 - (Wielkanoc)
 - 9.04
 - 16.04
 - Pod koniec semestru – 1h na wpisywanie ocen



Laboratoria

- przewidzianych jest 15 zajęć laboratoryjnych (w tym 11 punktowanych, 3 projektowe, 1 weryfikujące wiedzę)
- obecność jest obowiązkowa (możliwe są maksymalnie 2 nieobecności);
- spóźnienie na zajęcia powyżej 15 minut automatycznie jest odnotowane jako nieobecność;
- zajęcia trwają 90 minut, odbywają się bez przerwy;
- **Laboratoria: zajęcia punktowane (11 zajęć)**
- Zadania są dostępne na stronie java.fizyka.pw.edu.pl
- za każde zadanie można otrzymać 0-5 pkt
- w trakcie pisania programu wolno korzystać z napisanych przez siebie programów oraz zasobów Internetu (w szczególności dokumentacji Java); nie można korzystać z programów innych studentów ani komunikatorów
- napisany w trakcie trwania laboratorium program należy oddać na tych samych zajęciach
- za skończenie programu po zajęciach (w domu) i przedstawieniu go w kolejnym tygodniu można uzyskać dodatkowe punkty



Laboratoria

- **Zadanie weryfikujące wiedzę**
- celem zadania weryfikacyjnego jest zweryfikowanie indywidualnych umiejętności studenta z zakresu programowania w języku Java
- zadanie do napisania samodzielnie (bez pomocy prowadzącego)
- za zadanie weryfikacyjne można otrzymać 0-15 pkt
- w trakcie pisania programu wolno korzystać z napisanych przez siebie programów oraz zasobów Internetu (w szczególności dokumentacji Java)
- napisany w trakcie trwania laboratorium program należy oddać prowadzącemu na tych samych zajęciach (nie ma możliwości kończenia w domu)
- aby “zaliczyć” zadanie weryfikacyjne, należy zdobyć min. 6 pkt
- “zaliczenie” zadania weryfikującego nie jest obligatoryjne, jednakże
 - niezaliczenie zadania weryfikacyjnego świadczy o niesamodzielności studenta zatem automatycznie zeruje wszystkie punkty zebrane za kończenie zadań w domu
 - można uzyskać “zaliczenie” już w trakcie trwania zajęć weryfikacyjnych, jeśli pokaże się prowadzącemu odpowiednio działający fragment kodu



Laboratoria

- Wskazane jest aby zadania kończyć w trakcie zajęć. Możliwe jest również oddanie ostatecznej wersji programu na następnych zajęciach (ewentualnie wcześniej w trakcie konsultacji). Za skończenie programu po zajęciach możliwe będzie zdobycie dodatkowych 2.5 punktów ale tylko w przypadku przedstawienia w pełni działającego programu; poprawa polega na zademonstrowaniu działającego programu oraz dyskusji z prowadzącym
- → UWAGA: niezaliczenie zadania weryfikacyjnego świadczy o niesamodzielności studenta zatem automatycznie zeruje wszystkie punkty zebrane za kończenie zadań w domu.



Projekty

- **Celem wprowadzenia projektów jest:**
 - 1) Nauczenie pracy zespołowej nad projektami informatycznymi
 - 2) Nauczenie studentów samodzielnej pracy nad większym projektem
 - 3) Zapoznanie studentów z problematyką tworzenia gotowych projektów, taką tak:
 - a. Standardy kodu
 - b. Budowanie projektu
- **Szczegóły na temat projektów można znaleźć na stronie:**
 - <http://java.fizyka.pw.edu.pl/Projekty>
 - Na stronie można znaleźć również przykładowe tematy
 - Temat projektu każdy zespół ustala indywidualnie z prowadzącym laboratoria



Projekty

- **Realizowane zespołowo**
zespoły dwuosobowe
- **Zaliczenie czterech etapów kontrolnych:**
 - I. Specyfikacja (3 zajęcia), – 10% oceny
 - II. Prototype - User Interface (5 zajęcia), – 20% oceny [zajęcia projektowe]
 - III. Release Candidate (10 zajęcia), – 20% oceny [zajęcia projektowe]
 - IV. Final (15 zajęcia). – 50% oceny [zajęcia projektowe]
- **Zaliczenie polega na:**
 - I. Oddanie dokumentu specyfikacji prowadzącemu laboratoria
 - II – IV. Zajęcia projektowe
 - Prezentacja: (5-8 min na grupę) → pierwsza godzina zajęć
 - Omówienie kodu → druga godzina zajęć



Warunki zaliczenia

- **Projekt**
50 pkt
- **Laboratoria: zajęcia punktowane**
11 x 5 pkt = 55 pkt
- **Laboratoria: zajęcia weryfikujące wiedzę**
1 x 15 pkt = 15 pkt
- **Łącznie punktów:** 50 + 55 + 15 pkt = 120 pkt
- **Wykład**
+0.5 oceny w górę
- **Warunek zaliczenia:**
 - > 50% z projektu
 - > 50% punktów

Ilość punktów	Ocena
60 - 72	3
72.5 - 84	3.5
84.5 - 96	4
96.5 - 108	4.5
> 108	5

Pytania?



Powtarzanie przedmiotu

- **Projekt**

Jeśli projekt został zaliczony, ocena z projektu może zostać zamieniona na punkty.

5.0 = 100% = 50 pkt. 3.0 = 51% = 26 pkt.

W przeciwnym wypadku **obowiązkowe zaliczenie wszystkich etapów zgodnie z harmonogramem.**

- **Laboratoria: zajęcia punktowane**

~~11 x 5 pkt = 55 pkt~~

Ocena z laboratorium może zostać zamieniona na punkty. 100% = 55 pkt.

Czyli jeśli ktoś zaliczył laboratorium na 3, może dostać 51% punktów = 28 pkt.

W takim wypadku uczęszczanie na laboratoria nie jest obowiązkowe.

- **Laboratoria: zajęcia weryfikujące wiedzę**

1 x 15 pkt = 15 pkt – **obowiązkowe.**

Niezaliczenie tych zajęć: **-50% punktów z laboratorium.**

- **Łącznie punktów:** 50 + 55 + 15 pkt = 120 pkt

- **Wykład**

+0.5 oceny w górę

- **Warunek zaliczenia:**

- > 50% z projektu
- > 50% punktów

Pytania?




Ilość punktów	Ocena
60 - 72	3
72.5 - 84	3.5
84.5 - 96	4
96.5 - 108	4.5
> 108	5

Zaczynamy...!

Dlaczego Java?



Why Java?



[Wszystko](#) [Grafika](#) [Filmy](#) [Mapy](#) [Wiadomości](#) [Więcej](#) [Ustawienia](#) [Narzędzia](#)

Okolo 138 000 000 wyników (0,41 s)

[4 Reasons Why Java is Still #1 - Azul Systems, Inc.](#)
<https://www.azul.com/4-reasons-java-still-1/> ▼ [Tłumaczenie strony](#)
12 sty 2016 - Find out from world renowned **Java** Champion Simon Ritter about the ongoing success of **Java** and the four reasons why it is still number one.

[Why Java is the most popular programming language - The Server Side](#)
www.theserverside.com/.../Why-Java-is-the-most-popular-progra... ▼ [Tłumaczenie strony](#)
24 maj 2016 - Twenty-year-old **Java**, despite being long in the tooth, is still the most popular programming language for developing enterprise applications. The TIOBE index, which is one measure of the popularity of programming languages, shows that **Java** has been number one or number two for the past decade and ...

[Why Java Is Great - C2 Wiki](#)
wiki.c2.com/?WhyJavalsGreat ▼ [Tłumaczenie strony](#)
Simple grammar - **Java** has a very simple grammar familiar to anyone with experience in C and C++, which must be 99.9% of programmers. The BNF for **Java** has about 50 rules; that for C++, about 140. And C++ also has templates and a preprocessor in addition to the grammar. **Java** just got quite a bit more complex in 1.5 ...

[Why Learn Java - Best Programming Language](#)
www.bestprogramminglanguagefor.me/why-learn-java ▼ [Tłumaczenie strony](#)
Not to be confused with JavaScript, this general-purpose language was designed to be easier to use than C++, which was a notoriously complex language. 90% of the Fortune 500 companies have since used **Java** to develop desktop apps and website backend systems. **Java** is a highly portable language as it must be ...



TIOBE Index - Styczeń 2018

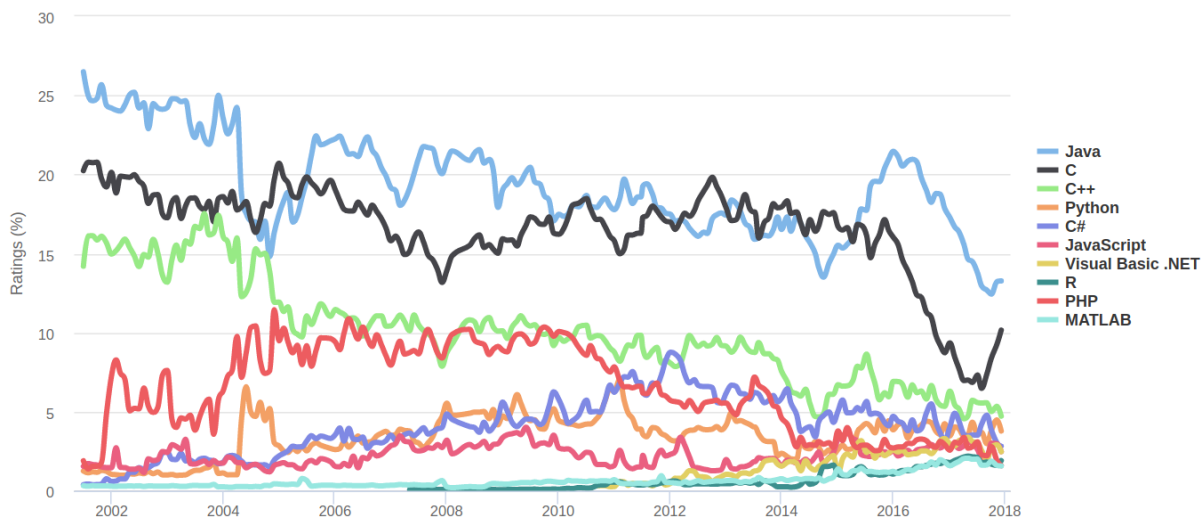
Najbardziej
popularny?

<https://www.tiobe.com/tiobe-index/>

Jan 2018	Jan 2017	Change	Programming Language	Ratings	Change
1	1		Java	14.215%	-3.06%
2	2		C	11.037%	+1.69%
3	3		C++	5.603%	-0.70%
4	5	▲	Python	4.678%	+1.21%
5	4	▼	C#	3.754%	-0.29%
6	7	▲	JavaScript	3.465%	+0.62%

TIOBE Programming Community Index

Source: www.tiobe.com



- Basically the calculation comes down to counting hits for the search query
+ "<language> programming"
- it's an attempt to measure the size of each language's community of developers.



Podobieństwo do C/C++

- Łatwa składnia
 - Java ma składnię znaną dla każdego kto miał do czynienia z C, C++ lub C#, czyli mniej więcej dla ~99.9% programistów.

Kod działający zarówno w C++ jak i w Javie

```
int foo=1, bar=3;
if (foo == 1) {
    foo=2;
}
else {
    foo=3;
}
for (int ii = 0; ii < bar; ii++){
    bar += 5 % 2;
}
```



Biblioteki

- Standardowe interfejsy programowania aplikacji (API)
 - Szeroki zestaw podstawowych bibliotek czeka, aby być wykorzystywany.
 - Są też na komputerach klienta, jeśli tylko ma zainstalowaną Javę.



Prosty w użyciu

- Szczegóły implementacji zrobi za Ciebie
 - Java jest językiem wysokiego poziomu
 - Została zaprojektowana, by być przyjazna początkującemu użytkownikowi: będzie się sama zabezpieczać przed dużą ilością głupich pomysłów programisty (pamiętasz Segmentation Fault z C++? Szansa, że na nie wpadniesz tutaj, są dużo mniejsze!).
 - Programujesz, nie przejmując się szczegółami przydzielania pamięci
 - **Garbage Collection** : nie przejmujesz się też zwalnianiem zaalokowanej pamięci – Java robi to za Ciebie → możesz zapomnieć o “delete”.



Przenośność

- “Write Once, Run Anywhere”
- *Maszyna Wirtualna*
 - Aplikacje Java zwykle kompilujemy, uzyskując programy które można uruchomić na **maszynie wirtualnej Java**, niezależnie od architektury komputera czy też systemu operacyjnego.
 - Musi się jedynie zgadzać zainstalowana wersja Javy.



Dlaczego Java?

Najbardziej popularny
Pożyczany na rynku
Podobny do C/C++
Prosty w użyciu
Standardowe API i biblioteki
Przenośny



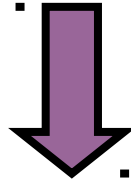
O co właściwie chodzi?
Co trzeba zainstalować?

Maszyna Wirtualna
JDK
Edycje Java
Jar



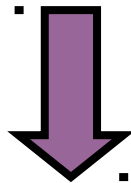
Maszyna Wirtualna

Kod źródłowy
pliki *.java



kompilacja

Kod bajtowy (bytecode)
pliki *.class



ładowanie

Wirtualna maszyna Java (**JVM**)

wykonywanie w środowisku Win/Unix/...



Maszyna Wirtualna

- Java Virtual Machine (JVM)
- Abstrakcyjna maszyna obliczeniowa, zdolna do wykonwania programu napisanego w Javie
 - (wykonywania kodu bajtowego Javy)
- Maszyna Wirtualna Javy nie jest nazwą konkretnego produktu.
 - Dostępna publicznie specyfikacja pozwala różnym producentom oprogramowania na tworzenie własnych maszyn wirtualnych pracujących pod kontrolą różnych środowisk i urządzeń.
- Jest zawarta w JRE: Java Runtime Environment



JRE i JDK

- **Java Runtime Environment (JRE)**
 - Pakiet zawierający wszystko, co potrzebne, by uruchomić program Java. Zawiera JVM (maszynę wirtualną) oraz implementację biblioteki Java Class.
 - Zawiera wyłącznie narzędzia niezbędne do uruchomienia aplikacji.
- **Java Development Kit (JDK)**
 - Oprócz tego, co JRE, zawiera również **narzędzia dla programistów**, pozwalające na tworzenie aplikacji na platformę JVM.



Java Development Kit (JDK)

JDK – narzędzia podstawowe

- **java** – interpreter
- **javac** – kompilator
- **apt** – annotation processing tool
- **javadoc**
- **appletviewer**
- **jar** – zarządca Java Archives (jars)
- **jdb** – debugger
- **javah** – narzędzie do tworzenia metod natywnych



JAR

- **JAR (Java Archive)** – archiwum Java, czyli plik zawierający skompresowane (ZIP) pliki klas i zasobów. Tworzy się je za pomocą narzędzia **jar** wchodzącego w skład JDK. Z klas znajdujących się w pliku .jar można korzystać, w tym uruchamiać aplikacje.
- Jeden plik (archiwum) zawierający wszystkie ważne dla programu pliki składowe.
- Plik JAR może zawierać plik manifestu, znajdujący się w META-INF/MANIFEST.MF, który opisuje jak używać pliku.
- Zwykle ma rozszerzenie .jar.



Java Development Kit (JDK)

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

ORACLE®

Menu



Sign In ▾

Country/Region ▾

Call

Oracle Technology Network / Java / Java SE / Downloads

Java™ SE Development Kit 8, Update 151

Java SE
Java EE
Java ME
Java SE Advanced & Suite
Java Embedded
Java DB
Web Tier
Java Card
Java TV
New to Java
Community
Java Magazine

Overview

Downloads

Documentation

Community

Technologies

Training

Java SDKs and Tools

Java SE
Java EE and Glassfish
Java ME
Java Card
NetBeans IDE
Java Mission Control

Java Resources

Java APIs
Technical Articles
Demos and Videos
Forums
Java Magazine
Developer Training
Tutorials
Java.com

Java SE Downloads



Java Platform (JDK) 9



NetBeans with JDK 8

Java Platform, Standard Edition

Java SE 9.0.4

Java SE 9.0.4 includes important bug fixes. Oracle strongly recommends that all Java SE 9 users upgrade to this release.

[Learn more](#)

- Installation Instructions
- Release Notes
- Oracle License
- Java SE Licensing Information User Manual
 - Includes Third Party Licenses
- Certified System Configurations
- Readme

JDK

DOWNLOAD

Server JRE

DOWNLOAD

JRE

DOWNLOAD

Edycje Java

- Java Standard Edition (Java SE)

Podstawowa platforma Java. Zawiera podstawowe biblioteki.

Ta wersja nas interesuje.

- Java Enterprise Edition (Java EE)

Rozszerzenie platformy podstawowej. Platforma przeznaczona m.in. dla systemów rozproszonych, tj. świadczących usługi dla wielu użytkowników.

- Java Micro Edition (Java ME)

jest to zbiór technologii i specyfikacji wykorzystywanych przez małe urządzenia, takie jak: telefony komórkowe i inne “małe” urządzenia. J2ME wykorzystuje niektóre komponenty J2SE, takie jak mniejsza maszyna wirtualna i odchudzone API.

- Inne (np. JavaFX)



Środowisko

Eclipse



IDE - Integrated development environment (zintegrowane środowisko programowania)

IDE - aplikacja (lub zespół aplikacji - środowisko) służących do tworzenia, modyfikowania, testowania i konserwacji oprogramowania.

Aplikacje będące **zintegrowanymi środowiskami programistycznymi** charakteryzują się tym, że udostępniają złożoną, wieloraką funkcjonalność obejmującą edycję kodu źródłowego, kompilowanie kodu źródłowego, tworzenie zasobów programu (tzn. formatek / ekranów / okien dialogowych, menu, raportów, elementów graficznych takich jak ikony, obrazy itp.), tworzenie baz danych, komponentów i innych.



IDE dla Javy

- Eclipse (<http://www.eclipse.org>)
- Netbeans (<http://netbeans.org/>),
- IntelliJ IDEA (<http://www.jetbrains.com/idea/>),
- Jcreator (<http://www.jcreator.com/>)
- ...

Na laboratoriach używany będzie Eclipse



Eclipse

<http://www.eclipse.org/downloads>



Google Custom Search

[GETTING STARTED](#) [MEMBERS](#) [PROJECTS](#) [MORE ▾](#)

Download Eclipse Technology
that is right for you

Tool Platforms



Get Eclipse **OXYGEN**

Install your favorite Eclipse packages.

[DOWNLOAD 64 BIT](#)

[Download Packages](#) | [Need Help?](#)



Eclipse Che

Eclipse Che is a developer
workspace server and cloud IDE.



A modern, open source software
development environment that
runs in the cloud.

Eclipse

http://www.eclipse.org/users

www.eclipse.org/users/



Google Custom Search

DOWNLOAD

GETTING STARTED MEMBERS PROJECTS MORE ▾

HOME / GETTING STARTED

1

Get Started

Download and install the Eclipse IDE.

DOWNLOAD ECLIPSE IDE

2

Extend Eclipse

Eclipse Marketplace is a great source of plug-ins and product that you can add to Eclipse.

- » Browse the **online catalog**
- » Use the Eclipse Marketplace Client from within Eclipse: [Help > Eclipse Marketplace...](#)

Popular Plugins:

- » **Subversive - SVN Team Provider**
- » **Eclipse Color Theme**
- » **Maven Integration for Eclipse**
- » **PyDev**

MARKETPLACE

3

Read Doc

Documentation is a great resource to get you started with the Eclipse IDE.

- » Getting Started with the **Eclipse IDE User Guide**
- » Getting Started with **Java development**
- » All online **Documentation**
- » What's new and noteworthy in **Eclipse Oxygen**
- » **Eclipse IDE Keybindings**

4

Get Help

There are many sources of help in the Eclipse community and ecosystem.

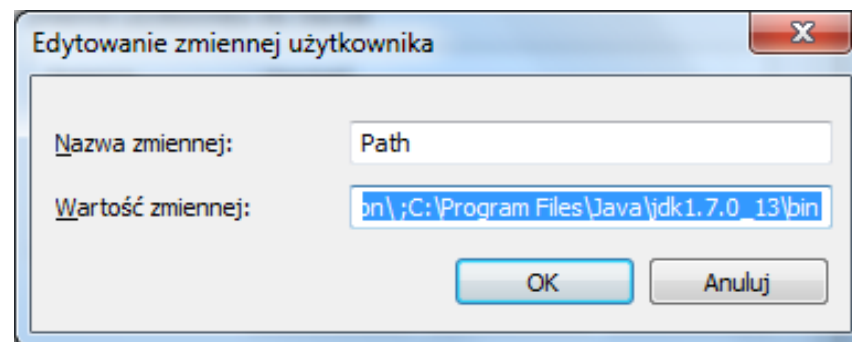
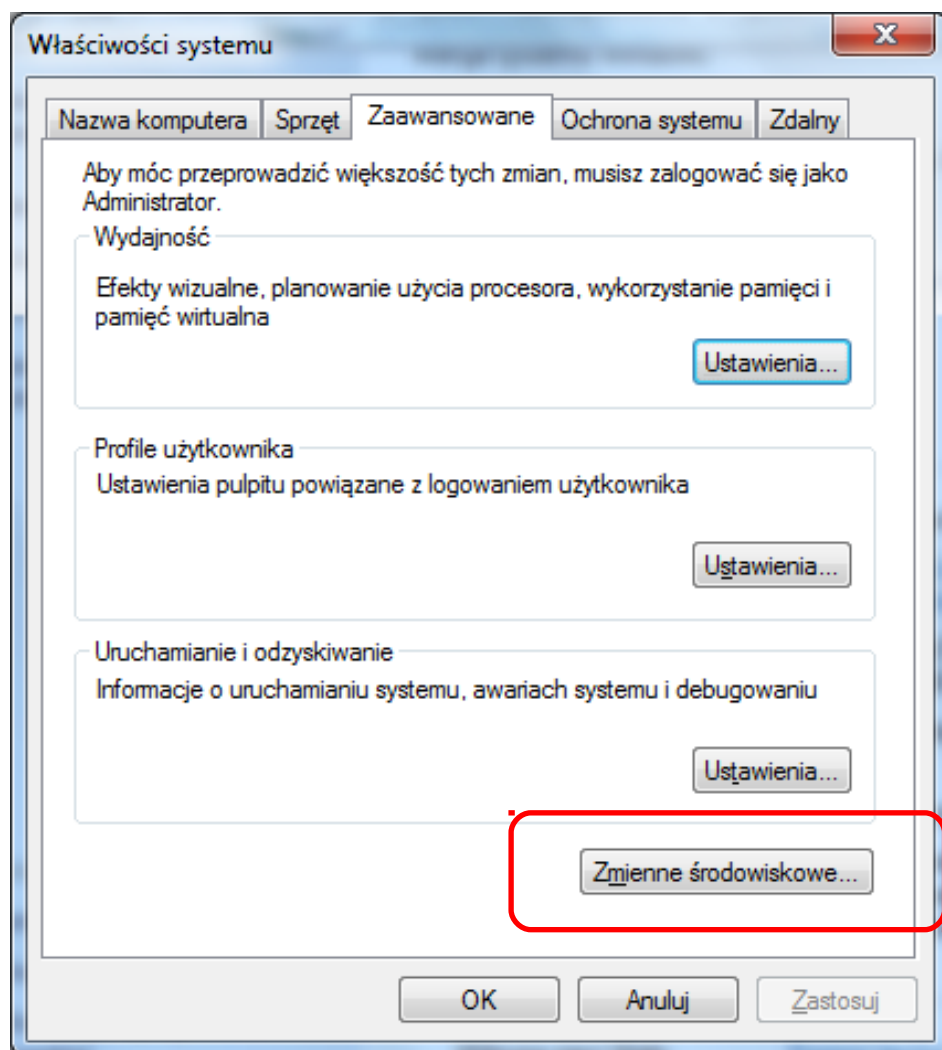
- » Ask questions on project **forums**; if you're not sure where to look, start with the **newcomer forum**
- » Open bug reports and feature requests in **Eclipse Bugzilla**
- » Connect live with project developers **via IRC**
- » Find companies that provide **training and consulting**
- » Investigate paid support options with **Long Term Support**.

Co ściągnąć?

- Java Development Kit (JDK) 8, Standard Edition (SE)
 - Najnowsza jest JDK 9, ale JDK 8 używane na laboratoriach powinno starczyć do wszystkiego co będziemy ćwiczyć
<http://www.oracle.com/technetwork/java/javase/downloads/index.html>
- Eclipse (<http://www.eclipse.org>)
 - Sugeruję ściągnąć najnowszy Eclipse Oxygen
- Na laboratoriach będziemy używać wersji:
 - JDK 1.8.0_151
 - Eclipse 3.8.1
 - archive.eclipse.org/eclipse/downloads



Uwaga: aby pod Windows możliwe było stosowanie komend JDK trzeba do zmiennej systemowej PATH dodać katalog \bin z JDK (np. C:\Program Files\Java\jdk1.7.0_13\bin)



Podstawy

Hello World [przykład]



Hello World

C++ / Java

Hello World: C++

```
#include <iostream>

using namespace std;

int main(int argc, char** argv)
{
    cout<<"Hello World!";
    return 0;
}
```

Hello World: Java

```
class Hello
{
    public static void main(String[ ] args)
    {
        System.out.println("Hello World!");
    }
}
```



Hello World

Hello World: Java

```
class Hello
{
    public static void main(String[ ] args)
    {
        System.out.println("Hello World!");
    }
}
```

Plik powinien nazywać się tak samo jak klasa z rozszerzeniem .java (Hello.java)

Kompilacja z linii poleceń: javac Hello.java

Uruchamianie z linii poleceń: java Hello



Podstawy

Struktura programu
Typy danych
Operatory
Instrukcje sterujące
Pętle



Struktura programu

```
package ... //deklaracja pakietu, opcjonalna ale zalecana
import ... // deklaracje importu
import ...
/** Komentarz dokumentacyjny */
// To jest klasa A
public class A
{
    ...
}
/* To jest
komentarz wielowierszowy
*/
class B
{
    public static void main(String[] args)
    {
        ...//Metoda main klasy startowej - od niej rozpoczyna się uruchamianie programu
    }
} //Koniec klasy B
```



Typy proste JAVA

Nazwa typu	Liczba bajtów	Dopuszczalne wartości	Znaczenie	Przykłady literalów
byte	1	-128 / +127	l. całkowita	1, 01, 0x01
short	2	-32768 / +32767	l. całkowita	128, 0xFF
int	4	-2147483648 / +2147483647	l. całkowita	32768, 0x1000
long	8	-9223372036854775808 / +9223372036854775807	l. całkowita	3l(L), 21474836
float	4	-3.xE-38 / (+3.xE+38)-1	l. rzeczywiste	3f, 3F, 3e(E)+10
double	8	-1.xE-30 / (+1.xE+30) -1	l. rzeczywiste	0.3, 0.3d(D) ...
char	2	0...65556	znaki Unicodu	'a', \u0013
boolean	1	true / false	wartości logiczne	true, false

Operatory

Operator	Znaczenie
+	Dodawanie
-	Odejmowanie
*	Mnożenie
/	Dzielenie
%	Reszta z dzielenia
\	Wynik dzielenia z resztą

Operator	Znaczenie
++	Inkrementacja - zwiększenie o 1
--	Dekrementacja - zmniejszenie o 1

Operator	Znaczenie
	Operacja \vee - LUB
&&	Operacja \wedge - I
!	Operacja \sim - negacja
>	większy niż
<	mniej niż
>=	większy równy
<=	mniej równy
!=	różny od

Instrukcje sterujące

analogiczne do C

```
if(warunek_logiczny){  
    //instrukcje wykonane jeżeli warunek jest PRAWDZIWY  
}
```

```
if(warunek_logiczny){  
    //instrukcje wykonane jeżeli warunek jest PRAWDZIWY  
}  
else{  
    //instrukcje wykonane jeżeli warunek jest FAŁSZYWY  
}
```



Instrukcje „if” można zagłębiać lub dokonywać wielokrotnego wyboru:

analogiczne do C

```
if(warunek_logiczny1){  
    if(warunek_logiczny2){  
        //instrukcje wykonane jeżeli warunek2 jest PRAWDZIWY  
    }  
}
```

```
if(warunek_logiczny1){  
    //instrukcje wykonane jeżeli warunek1 jest PRAWDZIWY  
}  
else if(warunek_logiczny2){  
    //instrukcje wykonane jeżeli warunek2 jest PRAWDZIWY  
}  
else{  
    //instrukcje wykonane jeżeli warunek1 i warunek 2 są FAŁSZYWE  
}
```



switch-case

```
switch ( key ) {  
    case value1:  
        // instrukcje dla key równego value1  
        break;  
    case value2:  
        // instrukcje dla key równego value2  
        break;  
    default:  
        break;  
}
```

Nadal analogiczne do C



switch-case

```
int i = 0;
switch ( i ) {
    case 0:
        System.out.println(0);
    case 1:
        System.out.println(1);
        break;
    default:
        System.out.println("default");
        break;
}
```

Brak „break” przy pierwszym warunku – wynikiem będzie wypisanie „0” i „1”

analogiczne do C



Pętla „for”

analogiczne do C

```
for(int i = 0; warunek; krok){  
    //instrukcja  
}
```

- Zmienną i nazywamy Indekssem Pętli
- Indeks może być dowolnym typem prostym poza boolean.
- Warunek może być dowolnym zdaniem logicznym, należy jednak zwrócić uwagę by nie była to tautologia. Otrzymamy wtedy pętlę nieskończoną
- Krok pętli może być dowolny jednak tak samo jak w przypadku warunku trzeba uważać na zapętlenie się programu.

```
int[] a = new int[] { 1, 2, 3 };  
for (int i = 0; i < a.length; i++) {  
    System.out.println(a[i]);  
}
```



do/while

Obie te konstrukcje są bardzo podobne do siebie. Główna różnica polega na tym iż w pętli **while** warunek jest sprawdzany przed wykonaniem instrukcji, a w pętli **do while** po wykonaniu instrukcji. Oznacza to, że pętla **do while** wykona się co najmniej jeden raz niezależnie od warunku.

```
while(warunekLogiczny){  
    //instrukcja  
}
```

```
do{  
    //instrukcja  
}while(warunekLogiczny)
```

...analogiczne do C



break/continue

Jeżeli chcemy przerwać wykonanie pętli jeżeli spełniony jest jakiś warunek to musimy użyć słowa **break**, a jeśli chcemy pominąć jakiś krok w pętli to musimy użyć słowa **continue**:

```
int i = 0;
while(true){
    if(i==5)
        break;
    System.out.println(i);
    i++;
}
```

```
for( int i = 0; i < 10; i++){
    if( i == 5 )
        continue;
    System.out.println(i);
}
```

...analogiczne do C



typ łańcuchowy (String)

Coś nowego!

- zaimplementowano jako obiekt
- automatyczne konwersje z innych typów i łączenie
- znaki łańcuchów są indeksowane od 0
- znak nieistniejący to -1

operatory: +, =, += mają specjalne znaczenie dla obiektów klasy String



typ łańcuchowy (String)

Coś nowego!

- zaimplementowano jako obiekt
- automatyczne konwersje z innych typów i łączenie
- znaki łańcuchów są indeksowane od 0
- znak nieistniejący to -1

operatory: +, =, += mają specjalne znaczenie dla obiektów klasy String

```
String s1;
```

```
String s2="Ala";
```

```
s1=s2+"ma Asa" // s1 będzie zawierać napis „Ala ma Asa”
```

Z obiektami typu String można konkatelować liczby towarzyszy temu niejawne wywołanie metody **toString()**

```
int k=1;
```

```
s1 = k+k+s2; // s1 będzie zawierać 2Ala
```

```
s1=s2+k+k; // s1 będzie zawierać Ala11
```

```
s1=s2+(k+k); // s1 będzie zawierać Ala2
```



Podstawy

String [przykład]

(perspektywy)



typ łańcuchowy (String)

Coś nowego!

- zaimplementowano jako obiekt
- automatyczne konwersje z innych typów i łączenie
- znaki łańcuchów są indeksowane od 0
- znak nieistniejący to -1

operatory: +, =, += mają specjalne znaczenie dla obiektów klasy String

```
String s1;
```

```
String s2="Ala";
```

```
s1=s2+"ma Asa" // s1 będzie zawierać napis „Ala ma Asa”
```

Z obiektami typu String można konkatelować liczby towarzyszy temu niejawne wywołanie metody **toString()**

```
int k=1;
```

```
s1 = k+k+s2; // s1 będzie zawierać 2Ala
```

```
s1=s2+k+k; // s1 będzie zawierać Ala11
```

```
s1=s2+(k+k); // s1 będzie zawierać Ala2
```



typ łańcuchowy (String)

//Przykład użycia obiektów String

```
public class StringPrzykład
```

```
{
```

```
    public static void main(String args[])
```

```
{
```

```
    String s1 = "Hello";
```

```
    String s2;
```

```
    s2 = s1;           //pełna kopia obiektu
```

```
    s1 = "World";
```

```
    System.out.println( "s1=" + s1 );    //s1=World
```

```
    System.out.println( "s2=" + s2 );    //s2=Hello
```

```
    s1 = 20+20+"";
```

```
    s2 = (20==30)+"";
```

```
    System.out.println( "s1=" + s1 );    //s1=40
```

```
    System.out.println( "s2=" + s2 );    //s2=false
```

```
    System.out.println( s1.charAt(1) );  //0
```

```
    System.out.println( s1.charAt(2) );
```

```
    // java.lang.StringIndexOutOfBoundsException:
```

```
    // String index out of range: 2
```

```
}
```

```
}
```



Podstawy Pakiety



Pakiety

- **Pakiet** – zbiór powiązanych klas i interfejsów, który pozwala na kontrolę dostępu i zapewnia hierarchiczny system nazewnictwa.

Przykładami są: javax.swing, java.lang, java.io. ...

Packages	
Package	Description
java.applet	Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.
java.awt	Contains all of the classes for creating user interfaces and for painting graphics and images.
java.awt.color	Provides classes for color spaces.
java.awt.datatransfer	Provides interfaces and classes for transferring data between and within applications.
java.awt.dnd	Drag and Drop is a direct manipulation gesture found in many Graphical User Interface systems that provides a mechanism to transfer information between two entities logically associated with presentation elements in the GUI.
java.awt.event	Provides interfaces and classes for dealing with different types of events fired by AWT components.
java.awt.font	Provides classes and interface relating to fonts.
java.awt.geom	Provides the Java 2D classes for defining and performing operations on objects related to two-dimensional geometry.
java.awt.im	Provides classes and interfaces for the input method framework.
java.awt.im.spi	Provides interfaces that enable the development of input methods that can be used with any Java runtime environment.

pełna lista standardowych pakietów w dokumentacji



Pakiety

- programy w Javie są złożone z zestawów pakietów, zawierających definicje klas i interfejsów
- typ zadeklarowany w pakiecie jest dostępny na zewnątrz pakietu tylko gdy został zadeklarowany jako publiczny (public)
- nazwa pakietu powinna być niepowtarzalna i jednocześnie określać jego charakter.
- zaleca się by określić kraj pochodzenia pakietu przez standardowe, dwuliterowe kody ISO, np..

`pl.edu.pw.fizyka.pojava.wykład`

`com.sun.examples`

- pakiet składa się z kompilowalnych plików, które automatycznie mają dostęp do wszystkich typów deklarowanych w pakiecie
- jeśli plik nie zawiera deklaracji pakietu to domyślnie należy do pakietu unnamed/default



Podstawy

Klasy i Dziedziczenie



Struktura klasy

```
class NazwaKlasy {  
    //deklaracje konstruktorów  
    NazwaKlasy(lista parametrów){  
        //treść/zawartość konstruktora  
    }  
    //deklaracje metod  
    Typ1 metoda1(lista-parametrów) {  
        //treść/zawartość metody1  
        return obiektTyp1;  
    }  
    ...  
    void metodaM(lista-parametrów) {  
        //treść/zawartość metodyM  
    }  
  
    //deklaracje pól  
    Typ pole1;  
    ...  
    Typ poleN;  
}
```



Struktura klasy

```
package pl.wykład1.klasapokazowa;

public class KlasaPrzykładowa {

    // deklaracje konstruktorów
    KlasaPrzykładowa() {
        x = 0;
        tekst = "domyslny";
    }

    // deklaracje metod
    public void setX(int aX) {
        x = aX;
    }

    public int getX() {
        return x;
    }

    public void wypisz() {
        System.out.println(x + " " + tekst);
    }

    // deklaracje pól
    int x;
    String tekst;
}
```

Jak stworzyć obiekt tej klasy?



Struktura klasy

```
package pl.wykład1.klasapokazowa;

public class KlasaPrzykładowa {

    // deklaracje konstruktorów
    KlasaPrzykładowa() {
        x = 0;
        tekst = "domyslny";
    }

    // deklaracje metod
    public void setX(int aX) {
        x = aX;
    }

    public int getX() {
        return x;
    }

    public void wypisz() {
        System.out.println(x + " " + tekst);
    }

    // deklaracje pól
    int x;
    String tekst;
}
```

Jak stworzyć obiekt tej klasy?

```
public static void main(String[] args) {
    KlasaPrzykładowa kp = new KlasaPrzykładowa();
    kp.wypisz();
}
```



Tworzenie obiektów

```
NazwaKlasy zmienna; //deklaracja zmiennej/referencji  
zmienna = new NazwaKlasy(argumenty_konstruktor);  
//tworzenie nowego obiektu przypisanego do referencji
```

Przykład: (tworzenie obiektu klasy Rectangle):

```
Rectangle prostokat;  
prostokat = new Rectangle(10,10,100,200);
```

Powyższy kod można uprościć, umieszczając deklarację zmiennej i tworzenie obiektu w jednej linii kodu:

```
NazwaKlasy zmienna = new NazwaKlasy(argumenty_konstruktor);
```

Przykład:

```
Rectangle prostokat = new Rectangle(10,10,100,200);
```



ZAPAMIĘTAJ!

Obiekty w Javie zawsze tworzone są przy pomocy operatora **new** po którym następuje konstruktor (czyli nazwa klasy, której obiekt jest tworzony oraz lista argumentów w nawiasach).

```
KlasaPrzykładowa kp = new KlasaPrzykładowa();
```

Operator **new** rezerwuje pamięć dla nowego obiektu oraz wywołuje odpowiedni konstruktor (klasa może mieć więcej niż jeden konstruktor, o tym który zostanie wywołany decyduje zgodność listy argumentów).

```
KlasaPrzykładowa kp2 = new KlasaPrzykładowa(5,"tekst2");
```

Operator **new** zwraca referencję do nowo utworzonego obiektu.



Dziedziczenie

```
package pl.wykład1.klasapokazowa;
```

```
public class KlasaPrzykładowa {
```

```
// deklaracje konstruktorów
```

```
KlasaPrzykładowa() {
```

```
    x = 0;
```

```
    tekst = "domyslny";
```

```
}
```

```
// deklaracje metod
```

```
public void setX(int aX) {
```

```
    x = aX;
```

```
}
```

```
public int getX() {
```

```
    return x;
```

```
}
```

```
public void wypisz() {
```

```
    System.out.println(x + " " + tekst);
```

```
}
```

```
// deklaracje pól
```

```
int x;
```

```
String tekst;
```

```
}
```

W jaki sposób zdefiniować klasę pochodną, dziedziczącą po już istniejącej?

```
public class KlasaPochodna extends KlasaPrzykładowa {
```

```
public void wypisz()
```

```
{
```

```
    System.out.println(x + " " + y + " " + tekst);
```

```
}
```

```
int y = 10;
```

```
}
```

```
public static void main(String[] args) {  
    //KlasaPrzykładowa kp = new KlasaPrzykładowa();  
    KlasaPochodna kp = new KlasaPochodna();  
    kp.wypisz();  
}
```


Dziedziczenie

```
package pl.wykład1.klasapokazowa;
```

```
public class KlasaPrzykładowa {
```

```
// deklaracje konstruktorów
```

```
KlasaPrzykładowa() {
```

```
    x = 0;
```

```
    tekst = "domyslny";
```

```
}
```

```
// deklaracje metod
```

```
public void setX(int aX) {
```

```
    x = aX;
```

```
}
```

```
public int getX() {
```

```
    return x;
```

```
}
```

```
public void wypisz() {
```

```
    System.out.println(x + " " + tekst);
```

```
}
```

```
// deklaracje pól
```

```
int x;
```

```
String tekst;
```

```
}
```

W jaki sposób zdefiniować klasę pochodną, dziedziczącą po już istniejącej?

```
public class KlasaPochodna extends KlasaPrzykładowa {
```

```
public void wypisz()
```

```
{
```

```
    System.out.println(x + " " + y + " " + tekst);
```

```
}
```

```
int y = 10;
```

```
}
```

```
public static void main(String[] args) {  
    //KlasaPrzykładowa kp = new KlasaPrzykładowa();  
    KlasaPochodna kp = new KlasaPochodna();  
    kp.wypisz();  
}
```

0 10 domyslny

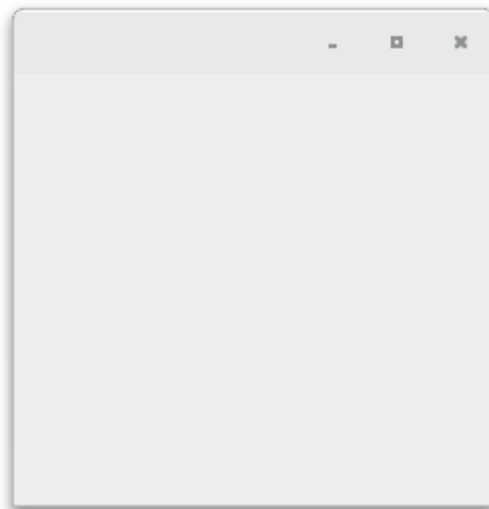
Okienka w Javie

Dziedziczenie po JFrame



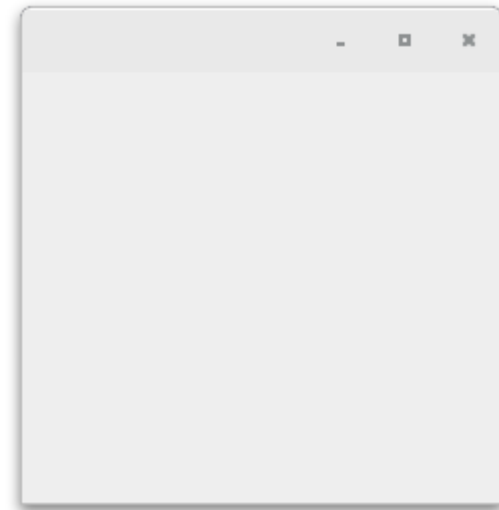
Frame

Własne okienko [przykład]



Dziedziczenie po JFrame

```
package pl.wyklad1.klasapokazowa;  
  
import javax.swing.JFrame;  
  
public class PierwszeOkienko extends JFrame {  
  
    public static void main(String[] args) {  
        PierwszeOkienko po = new PierwszeOkienko();  
        po.setSize(500,500);  
        po.setVisible(true);  
    }  
}
```



Struktura programu

```
package pl.mojastrona.mojpakiet;  
import javax.swing.*;
```

Określenie pakietu, do którego należą klasy zdefiniowane w tym pliku (*opcjonalne...*).

```
class MojeOkienko extends JFrame  
{
```

Zewnętrzne pakiety (lub pojedyncze klasy, interfejsy), z których korzystamy w naszym programie. „odpowiednik” dyrektywy `#include` w C/C++.

```
    public MojeOkienko()  
    {  
        ...  
    }  
}
```

Deklaracja klasy rozszerzającej inną klasę (dziedziczenie)

Konstruktor – taka sama nazwa jak klasa, może być kilka definicji konstruktorów dla jednej klasy, np.
`public MojeOkienko(int parametrPoczątkowy)`
{
}
}

```
class KlasaStartowa  
{
```

Druga klasa, w której deklarowane są referencje do obiektów innej klasy, oraz tworzony jest nowy obiekt operator „new” + wywołanie konstruktora

```
    MojeOkienko ob1 = new MojeOkienko();  
    public static void main(String[] args)  
    {  
        ...  
    }  
}
```

Metoda *main* klasy startowej – od niej rozpoczyna się uruchamianie programu

Trzy najpopularniejsze rodzaje programów:

- **aplikacja konsolowa – samodzielny program**
pracujący w trybie konsolowym/tekstowym systemu operacyjnego,
- **aplikacja graficzna – samodzielny program**
pracujący w trybie graficznym (okienkowym)
- **aplet – najczęściej nieduży program napisany w**
język Java i umieszczony na stronie HTML i
uruchamiany wraz z nią przez przeglądarkę
internetową, obsługującą język Java.
(deprecated – już nie stosowane)



Konwencja kodu

- Nazwa klasy powinna „coś znaczyć”. Pierwsza litera w nazwie klasy pisana jest wielka litera. Jeśli nazwa klasy składa się z kilku słów to pisze się je łącznie, każde słowo rozpoczynając z wielkiej litery:

```
class MojaKlasa
{
    ...
}
```



Konwencja kodu

Metody i pola oraz nazwy referencji pisze się analogicznie, poza tym że pierwsza litera jest mała (należy pamiętać, że język Java rozróżnia małe i wielkie litery):

```
class MojaKlasa{  
    int mojaZmienna;  
    void pobierzZmienna(int numerZmiennej){  
        ...  
    }  
}
```



Konwencja kodu

Instrukcja złożona to ciąg instrukcji pomiędzy nawiasami { ... }, nazywana blokiem instrukcji np.:

```
instrukcja_grupujaca //klasa, metoda, itp.  
{  
    ... //blok kodu  
} //koniec instrukcja_grupujaca
```

Instrukcja grupująca (np. klasa, metoda, instrukcja sterująca, np. pętla) rozpoczyna blok kodu. Jeśli zadeklarujemy w bloku zmienną to będzie ona widoczna tylko w tym bloku.

Po klamrze zamykającej nie trzeba stawiać średnika.



Konwencja kodu

Istnieją kilka wariantów stawiania nawiasów {}:

```
while(i) {  
}
```

```
while(i)  
{  
}
```

```
while(i)  
  {  
  }
```

Wszystkie te są poprawne. Warto jednak wybrać jeden i konsekwentnie się go trzymać. Warto również po otwarciu klamry od razu postawić klamrę zamykającą. Pozwoli to uniknąć wielu niepotrzebnych błędów.



Konwencja kodu

Warto stosować nadmiarowości i dodatkowe separatory celem poprawienia czytelności kodu np.:

```
int a = (c * 2) + (b / 3);
```

zamiast:

```
int a=c*2+b/3;
```



Konwencja kodu

```
instrukcja_grupujaca {  
    instrukcje bloku...  
} //koniec instrukcja_grupujaca
```

- Kod po klamrze otwierającej umieszcza się w nowej linii.
- Każda linia bloku jest przesunięta (wcięta) względem pierwszego znaku `instrukcja_grupujaca` o stałą liczbę znaków.
- Blok kończy się klamrą zamykającą w nowej linii na wysokości pierwszego znaku instrukcji grupującej.
- Blok warto kończyć komentarzem umożliwiającym identyfikację bloku zamykanego przez daną klamrę.



Konwencja kodu

Ważnym elementem czytelnej konstrukcji kodu jest stosowanie komentarzy liniowych:

```
int i = 5; //komentarz pojedynczej linii
```

lub blokowych:

```
/*
```

```
blok komentarza:
```

```
druga linia komentarza
```

```
...
```

```
*/
```

lub blokowych z dodatkowymi „gwiazdkami” *:

```
/*
```

```
* blok komentarza:
```

```
* druga linia komentarza
```

```
*/
```



Typ tablicowy

- Tablice w Javie są jednowymiarowe, ale mogą zawierać inne tablice.
- Przy definicji typu nie podaje się wielkości tablicy
- Każda tablica jest obiektem i ma zdefiniowane pole length o wartości równej liczbie elementów w tablicy
- Zadeklarowanie tablicy nie przydziela jej pamięci! Aby przydzielić pamięć należy użyć new
- Tablice są indeksowane wartościami typu int, stąd największa tablica ma 2¹⁴⁷⁴⁸³⁶⁴⁷ elementów.
 - Jeśli potrzebujesz większej tablicy, coś jest nie w porządku z Twoim kodem



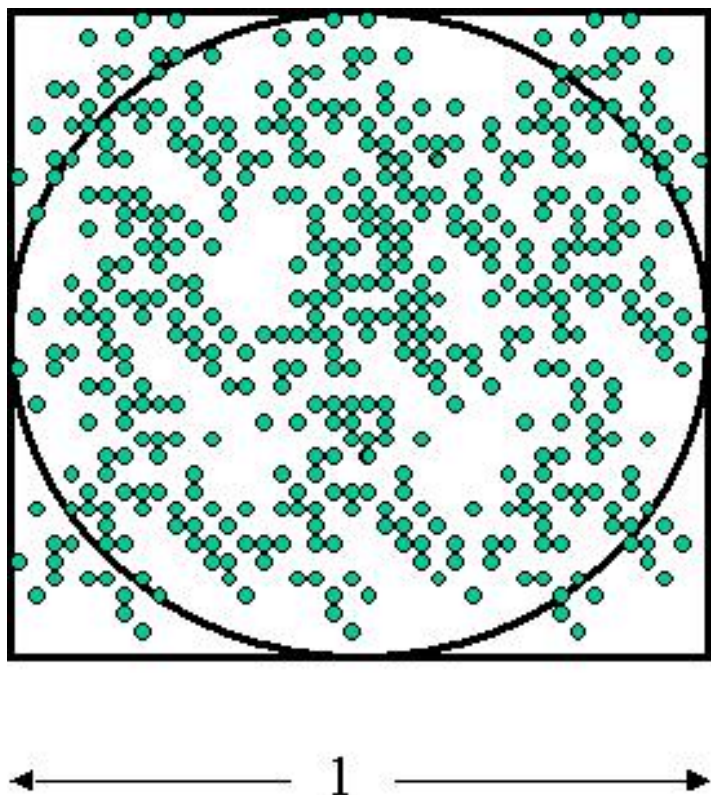
Typ tablicowy



```
//przykłady deklaracji i tworzenia tablic
int arr[];
int [] arr;           //to samo co wyżej
arr = new int[3];
int arr[] = { 0, 0, 0 }; //równoważne
int arr[][];
arr[][] = new int[3][]; //powstaje tablica wektorów
arr[0] = new int[5];    //pierwszy wektor
arr[1] = new int[5];
arr[2] = new int[5];    //ostatni wektor
arr[][] = new int[10][];
for (int i=0; i<10; i++)
    arr[i] = new int[i+1]; //tablica trójkątna
arr[][] = { {1}, {0, 1}, {0, 0, 1} };
arr[0] = null;
arr[1] = new int[10];
arr[2] = {10, 10};
```



Przykład – wykorzystanie metody Monte-Carlo do wyznaczenia liczby π



Pole kwadratu: l^2

Pole koła: $\pi \left(\frac{l}{2}\right)^2$

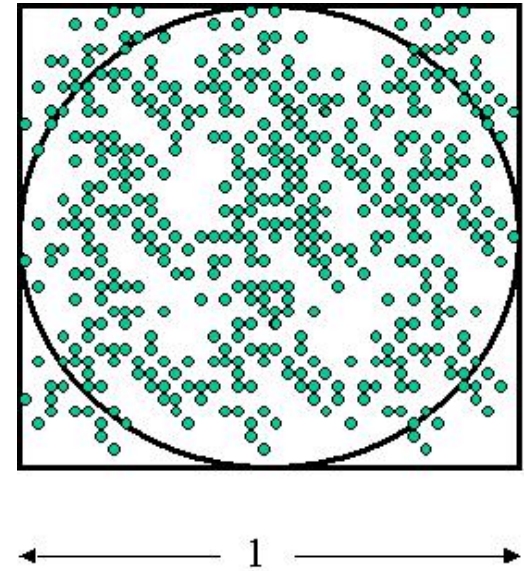
Losowanie N punktów, z czego M leży wewnątrz koła.

$$\frac{M}{N} = \frac{\pi}{4}$$



Przykład – wykorzystanie metody Monte-Carlo do wyznaczenia liczby π

```
class Monte_Carlo {  
    public static void main(String[] args) {  
        double r1, r2;  
        int i, m=0, n=10000;  
        for (i=1 ; i<=n ; i++){  
            r1=Math.random()-0.5;  
            r2=Math.random()-0.5;  
            if (r1 *r1 + r2 *r2 <0.25) m++;  
        }  
        System.out.println("PI oszacowane = "  
                             + 4.* (double) m/n);  
        System.out.println("PI z Math.PI = " + Math.PI);  
    }  
}
```



Literatura

- „Java – przewodnik dla początkujących”, Herbert Schildt
- „Thinking in Java” - Bruce Eckel
- „Java Podstawy” - Cay Horstmann, Gary Cornell
- „Java Receptury” - Ian F. Darwin
- „Java ćwiczenia praktyczne” - Marcin Lis
- „Java w zadaniach” - Steve Potts
- „Java po C++” - Jan Bielecki
- „Java 4 Swing” - Jan Bielecki
- Dokumentacja JAVA



Dokumentacja on-line (javadoc): <http://docs.oracle.com/javase/>



API Documentation

(Application Programming Interface -
interfejs programowania aplikacji)

The screenshot displays the Oracle Java SE Technical Documentation website for the Java Platform, Standard Edition 7. The browser window shows the URL `docs.oracle.com/javase/7/docs/api/index.html`. The page title is "Java™ Platform, Standard Edition 7 API Specification". Below the title, a description states: "This document is the API specification for the Java™ Platform, Standard Edition." and "See: Description".

The left sidebar contains a navigation menu with "All Classes" and "Packages". The "All Classes" list includes various abstract classes and interfaces such as `AbstractAction`, `AbstractAnnotationValueVisitor6`, `AbstractAnnotationValueVisitor7`, `AbstractBorder`, `AbstractButton`, `AbstractCellEditor`, `AbstractCollection`, `AbstractColorChooserPanel`, `AbstractDocument`, `AbstractDocument.AttributeContext`, `AbstractDocument.Content`, `AbstractDocument.ElementEdit`, `AbstractElementVisitor6`, `AbstractElementVisitor7`, `AbstractExecutorService`, `AbstractInterruptibleChannel`, `AbstractLayoutCache`, `AbstractLayoutCache.NodeDimensions`, `AbstractList`, `AbstractListModel`, `AbstractMap`, `AbstractMap.SimpleEntry`, `AbstractMap.SimpleImmutableEntry`, `AbstractMarshallerImpl`, `AbstractMethodError`, and `AbstractOwnableSynchronizer`.

The main content area features a "Packages" table with the following data:

Package	Description
<code>java.applet</code>	Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.
<code>java.awt</code>	Contains all of the classes for creating user interfaces and for painting graphics and images.
<code>java.awt.color</code>	Provides classes for color spaces.
<code>java.awt.datatransfer</code>	Provides interfaces and classes for transferring data between and within applications.
<code>java.awt.dnd</code>	Drag and Drop is a direct manipulation gesture found in many Graphical User Interface systems that provides a mechanism to transfer information between two entities logically associated with presentation elements in the GUI.
<code>java.awt.event</code>	Provides interfaces and classes for dealing with different types of events fired by AWT components.
<code>java.awt.font</code>	Provides classes and interface relating to fonts.
<code>java.awt.geom</code>	Provides the Java 2D classes for defining and performing operations on objects related to two-dimensional geometry.
<code>java.awt.im</code>	Provides classes and interfaces for the input method framework.
<code>java.awt.im.spi</code>	Provides interfaces that enable the development of input methods that can be used with any Java runtime environment.
<code>java.awt.image</code>	Provides classes for creating and modifying images.
<code>java.awt.image.renderable</code>	Provides classes and interfaces for producing rendering-independent images.
<code>java.awt.print</code>	Provides classes and interfaces for a general printing API.
<code>java.beans</code>	Contains classes related to developing <i>beans</i> – components based on the JavaBeans™ architecture.



Dziękuję za Uwagę!

**Do zobaczenia za tydzień.
Będę więcej opowiadać o projektach.**



Ciekawostki

Operator trójargumentowy ?



Zamiast bloku „if-else” można użyć operatora trójargumentowego ?

```
zmienna = warunek ? wartosc_jak_prawda : wartosc_jak_falsz;
```

Co jest równoważne:

```
if(warunek){  
    zmienna = wartosc_jak_prawda;  
}  
else{  
    zmienna = wartosc_jak_falsz;  
}
```

...analogiczne do C

