

# Moduł szkoleniowy Java

---

*poziom podstawowy 15 godzinny*

## Spis treści

1	Metryka dokumentu.....	4
2	Cel.....	5
3	Opis sposobu realizacji celów.....	5
4	Treści kształcenia.....	5
5	Opis założonych osiągnięć ucznia.....	5
6	Sposoby osiągania celów.....	5
7	Propozycje kryteriów oceny i metod sprawdzania osiągnięć ucznia .....	6
8	Lekcje.....	8
8.1	Lekcja 1 - Wstęp do Javy i środowiska pracy.....	8
8.1.1	Cel lekcji.....	8
8.1.2	Treść - slajdy z opisem .....	8
8.1.3	Opis założonych osiągnięć ucznia.....	29
8.2	Lekcja 2 - Typy podstawowe i instrukcje sterujące .....	29
8.2.1	Cel lekcji.....	29
8.2.2	Treść - slajdy z opisem .....	29
8.2.3	Opis założonych osiągnięć ucznia.....	38
8.3	Lekcja 3 - Wstęp do programowania obiektowego.....	39
8.3.1	Cel lekcji.....	39
8.3.2	Treść - slajdy z opisem .....	39
8.3.3	Opis założonych osiągnięć ucznia.....	48
8.4	Lekcja 4 – Interfejsy, klasy abstrakcyjne i dziedziczenie .....	48
8.4.1	Cel lekcji.....	48
8.4.2	Treść - slajdy z opisem .....	49
8.4.3	Opis założonych osiągnięć ucznia.....	56
8.5	Lekcja 5 - Polimorfizm, instrukcja „final”, klasy wewnętrzne i anonimowe.....	56
8.5.1	Cel lekcji.....	56
8.5.2	Treść - slajdy z opisem .....	57
8.5.3	Opis założonych osiągnięć ucznia.....	63
8.6	Lekcja 6 - Typ wyliczeniowy, JavaDocs .....	63
8.6.1	Cel lekcji.....	63
8.6.2	Treść - slajdy z opisem .....	63

8.6.3	Opis założonych osiągnięć ucznia .....	73
8.7	Lekcja 7 - Tablice jednowymiarowe i wielowymiarowe, kolekcje .....	73
8.7.1	Cel lekcji .....	73
8.7.2	Treść - slajdy z opisem .....	73
8.7.3	Opis założonych osiągnięć ucznia .....	82
8.8	Lekcja 8 - Operacje wejścia – wyjścia i wyjątki .....	82
8.8.1	Cel lekcji .....	82
8.8.2	Treść - slajdy z opisem .....	83
8.8.3	Opis założonych osiągnięć ucznia .....	94
8.9	Lekcja 9 - Graficzny interfejs użytkownika - Swing .....	94
8.9.1	Cel lekcji .....	94
8.9.2	Treść - slajdy z opisem .....	94
8.9.3	Opis założonych osiągnięć ucznia .....	104
8.10	Lekcja 10 - Przechwytywanie zdarzeń, tworzenie plików wykonywalnych .....	104
8.10.1	Cel lekcji .....	104
8.10.2	Treść - slajdy z opisem .....	105
8.10.3	Opis założonych osiągnięć ucznia .....	110

## 1 Metryka dokumentu

Szczeciński Park Naukowo Technologiczny			
Dokument	Moduł Szkoleniowy Java		
Krótki opis dokumentu	Moduł szkoleniowy kursu Java zawierający opis celów, treści poszczególnych lekcji, ćwiczeń oraz test końcowy sprawdzający wiedzę		
Data druku	12.02.2013	Liczba stron	110
Nazwa pliku	Java - Moduł szkoleniowy.docx	Status	roboczy

### Historia zmian

Nr wersji	Data	Opis	Działanie (*)	Autorzy
0.1	01.02.2013	Utworzenie nowego dokumentu	N	Aleksander Gralak, DailyGroup Sp. z o.o.
1.0	12.02.2013	Wersja gotowa do publikacji	Z	Aleksander Gralak, DailyGroup Sp. z o.o.

(\*) Działanie: N-Nowy, Z-Zmiana, W-Weryfikacja

### Lista dystrybucyjna

Imię i nazwisko / Rola	Organizacja

### Zgłoszono do odbioru

Imię i nazwisko		Data:		Podpis:	
Imię i nazwisko	Aleksander Gralak	Data:	12.02.2013	Podpis:	

## 2 Cel

Poznanie zastosowania i zalet języka programowania Java. Umiejętność wykorzystania podstawowych możliwości języka. Praktyczne wykorzystanie elementów środowiska pracy. Podstawowa umiejętność projektowania aplikacji zorientowanej obiektowo. Tworzenie wieloplatformowych aplikacji z interfejsem graficznym. Zwiększenie efektywności pracy poprzez umiejętność korzystania i tworzenia z dokumentacji.

## 3 Opis sposobu realizacji celów

10 półtoragodzinnych lekcji składających się z treści teoretycznych wraz z ćwiczeniami. Praktyczne uwagi dla osób początkujących dotyczące środowiska pracy Eclipse. Po zakończeniu całego cyklu - przeprowadzenie egzaminu sprawdzającego wiedzę.

## 4 Treści kształcenia

Treść kursu została podzielona na 10 bloków tematycznych – po jednym do każdej lekcji:

1. Wstęp do Javy i środowiska pracy,
2. Typy podstawowe i instrukcje sterujące,
3. Wstęp do programowania obiektowego ,
4. Interfejsy, klasy abstrakcyjne i dziedziczenie,
5. Polimorfizm, instrukcja „final”, klasy wewnętrzne i anonimowe,
6. Typ wyliczeniowy, JavaDocs ,
7. Tablice jednowymiarowe i wielowymiarowe, kolekcje,
8. Operacje wejścia – wyjścia i wyjątki ,
9. Graficzny interfejs użytkownika - Swing ,
10. Przechwytywanie zdarzeń, tworzenie plików wykonywalnych.

## 5 Opis założonych osiągnięć ucznia

Uczeń po odbyciu kursu pozna teoretyczne podstawy Języka Java, będzie umiał zaprojektować prostą aplikację zorientowaną obiektowo z interfejsem graficznym.

## 6 Sposoby osiągnięcia celów

Po odbyciu każdej lekcji uczeń powinien samodzielnie wykonać proste ćwiczenia w celu dokładnego zrozumienia omawianych mechanizmów. Programowanie w języku Java bez odpowiedniego środowiska jest bardzo trudne dlatego jednym z celów jest jego poznanie. Umożliwi to wykonywanie ćwiczeń w czasie efektywnym.

## 7 Propozycje kryteriów oceny i metod sprawdzania osiągnięć ucznia

Ocena końcowa powinna zależeć od wyniku testu sprawdzającego wiadomości teoretyczne (20 pytań zamkniętych pojedynczego wyboru - 20 punktów).

Propozycje kryteriów oceny:

- 0-9 punktów - ocena 1,
- 10-12 punktów - ocena 2,
- 13-15 punktów - ocena 3,
- 16-18 punktów - ocena 4,
- 29-20 punktów - ocena 5, Test końcowy sprawdzający wiedzę

20 zadań na 30 minutowy test sprawdzający wiedzę. Pogrubioną czcionką zaznaczono prawidłowe odpowiedzi.

1. Java kompiluje się do:
  - a. Kodu maszynowego
  - b. **Kodu bajtowego**
  - c. Pliku skryptowego
2. Kompilator do Javy znajduje się w pakiecie:
  - a. **JDK**
  - b. JRE
  - c. Eclipse
3. Aby przechować wartość 100 jakiego najmniejszego typu możemy użyć:
  - a. long
  - b. int
  - c. **short**
4. Która pętla wykona jeden krok przebiegu nawet w przypadku fałszywego warunku:
  - a. for
  - b. **do-while**
  - c. while
5. Słowo „new” tworzy:
  - a. nową klasę;
  - b. **nowy obiekt**
  - c. nowy typ
6. Czy kiedy w klasie występuje konstruktor „public MojaKlasa(int x)” mogą stworzyć obiekt za pomocą instrukcji „new MojaKlasa()”?
  - a. tak
  - b. nie
  - c. **tylko wtedy gdy istnieje drugi konstruktor „public MojaKlasa()”**
7. Która ze struktur nie posiada implementacji metod:
  - a. klasa

- b. **interfejs**
  - c. klasa niewłaściwa
8. Jeżeli chcemy, aby dana metoda była widoczna tylko w bieżącym pakiecie, to jakiego typu widoczności użyjemy?
- a. private
  - b. public
  - c. **nie podajemy typu, co oznacza użycie widoczności package (default)**
9. Które zdanie jest nieprawdziwe:
- a. Każdy obiekt typu Integer jest również obiektem typu Object
  - b. Mogę używać obiektu typu Integer tak, jak każdego innego obiektu typu Object
  - c. **Mogę stworzyć obiekt, który jest jednocześnie typu Integer i String.**
10. Zaznacz prawidłowe stwierdzenie
- a. **Mogę stworzyć instancję interfejsu, pod warunkiem, że zaimplementuję jego metody**
  - b. Jedyną możliwością stworzenia instancji interfejsu jest użycie słowa „implements”
  - c. Mogę stworzyć instancję interfejsu tak, jak zwykłej klasy.
11. Pisanie JavaDoc rozpoczynamy od znaku:
- a. **/\*\***
  - b. //
  - c. @
12. Podpowiadanie składni w środowisku Eclipse wywołujemy skrótem:
- a. „Ctrl” + „Shift” + „O”
  - b. **„Ctrl” + „Spacja”**
  - c. „F5”
13. Jeżeli chcemy udostępnić innym programistom wybór jednego z 5 słów, powinniśmy:
- a. **Skorzystać z typu wyliczeniowego - enum**
  - b. Umieścić słowa w tablicy
  - c. Poprosić o liczbę z przedziału od 1 do 5
14. Jeżeli chcemy wstawiać liczby w różne miejsca zbioru, powinniśmy skorzystać z:
- a. tablicy
  - b. typu wyliczeniowego
  - c. **Isty**
15. Odczytując plik używamy obiektu:
- a. System.in
  - b. OutputStream
  - c. **InputStream**
16. Aby stworzyć własny wyjątek należy dziedziczyć po klasie:
- a. **Exception**
  - b. Error
  - c. StackTrace
17. Aby umożliwić użytkownikowi wybór kilku opcji w interfejsie graficznym, skorzystamy z:
- a. **JComboBox**

- b. **JCheckBox**
  - c. JRadioButton
18. Aby stworzyć obiekt reagujący na zdarzenia myszy, należy zaimplementować interfejs:
- a. LayoutManager
  - b. MouseEvent
  - c. **MouseListener**
19. Zarządca rozkładu w interfejsie graficznym to w przypadku Swinga:
- a. **Layout**
  - b. Listener
  - c. JPanel
20. Plik wykonywalny JAR to:
- a. **Archiwum ZIP**
  - b. Plik z pojedynczym kodem bajtowym
  - c. Kod maszynowy

## 8 Lekcje

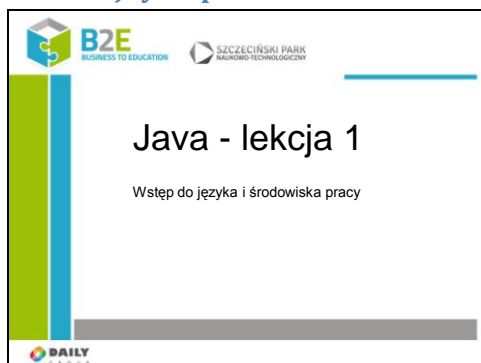
### 8.1 Lekcja 1 - Wstęp do Javy i środowiska pracy

#### 8.1.1 Cel lekcji

Celem lekcji jest poznanie czym jest język programowania Java, w jaki sposób działa i jakie są jego zalety. Wy tłumaczone jest jak poprawnie skonfigurować środowisko pracy i co jest potrzebne, aby móc uruchamiać zbudowane programy.

#### 8.1.2 Treść - slajdy z opisem

Slajd 1







Slajd 2

**B2E** **SZCZECIŃSKI PARK**  
BUSINESS TO EDUCATION NAUKOWO-TECHNOLOGICZNY

Główne koncepcje:

- Obiektość
- Niezależność od architektury
- Sieciowość i obsługa programowania rozproszonego
- Niezawodność i bezpieczeństwo

**Java**  
**Sun** microsystems  
**ORACLE**

**DAILY** Wstęp do języka i środowiska pracy

Java jest językiem kompilowanym do kodu bajtowego. Oznacza to, że aplikacje napisane w Javie i skompilowane nie mogą zostać uruchomione bezpośrednio przez system operacyjny. Niezbędna jest maszyna wirtualna.

Java została stworzona przez grupę roboczą pod kierunkiem Jamesa Goslinga z firmy Sun Microsystems, obecnym właścicielem tej technologii jest Oracle Corporation.

Jego podstawowe koncepcje zostały przejęte z języka Smalltalk (maszyna wirtualna, zarządzanie pamięcią) oraz z języka C++ (duża część składni i słów kluczowych).

### Główne koncepcje

Autorzy języka Java określili kilkanaście kluczowych koncepcji swojego języka. Najważniejsze z nich to:

#### Obiektość

W przeciwieństwie do proceduralno-obiektowego języka C++, Java jest silnie ukierunkowana na obiektość. O obiekcie można myśleć jako o samoistnej części programu, która może przyjmować określone stany i ma określone zachowania, które mogą zmieniać te stany bądź przysyłać dane do innych obiektów. Wyjątkiem od całkowitej obiektości są typy proste (inaczej: typy wbudowane, prymitywy).

#### Dziedziczenie

W Javie wszystkie obiekty są pochodną obiektu nadrzędnego (jego klasa nazywa się po prostu Object), z którego dziedziczą podstawowe zachowania i właściwości. Dzięki temu wszystkie mają wspólny podzbiór podstawowych możliwości, takich jak ich: identyfikacja, porównywanie, kopiowanie, niszczenie czy wsparcie dla programowania współbieżnego.

#### Niezależność od architektury

Tę właściwość Java ma dzięki temu, że kod źródłowy programów pisanych w Javie kompiluje się do kodu pośredniego (kodu

bajowego). Powstały kod jest niezależny od systemu operacyjnego i procesora, a wykonuje go tzw. wirtualna maszyna Javy, która (między innymi) tłumaczy kod uniwersalny na kod dostosowany do specyfiki konkretnego systemu operacyjnego i procesora. W tej chwili wirtualna maszyna Javy jest już dostępna dla większości systemów operacyjnych i procesorów. Jednak z uwagi na to, że kod pośredni jest interpretowany, taki program jest wolniejszy niż kompilowany do kodu maszynowego.

Sieciowość i obsługa programowania rozproszonego

Dzięki wykorzystaniu reguł obiektowości, Java nie widzi różnicy między danymi płynącymi z pliku lokalnego a danymi z pliku dostępnego przez HTTP czy FTP.

Niezawodność i bezpieczeństwo

W zamierzeniu Java miała zastąpić C++ – obiektowego następcę języka C. Jej projektanci zaczęli od rozpoznania cech języka C++, które są przyczyną największej liczby błędów programistycznych, by stworzyć język prosty w użyciu, bezpieczny i niezawodny. O ile po siedmiu odsłonach Javy jej prostota jest dyskusyjna, o tyle język faktycznie robi dużo, by utrudnić programiście popełnienie błędu.

Slajd 3



Przygotowanie środowiska pracy:

- [www.oracle.com](http://www.oracle.com)
- Java JRE (Java Runtime Environment)
- Java JDK (Java Development Kit)
- <http://www.eclipse.org/downloads/>
- Eclipse Classic

 Wstęp do języka i środowiska pracy

Aby móc uruchomić aplikację napisaną w Javie musimy mieć maszynę wirtualną. Znajduje się ona w pakiecie Java JRE (Java Runtime Environment). Można go łatwo wyszukać korzystając z wyszukiwarki Google lub pobrać bezpośrednio ze strony [www.oracle.com](http://www.oracle.com).

Aby móc uruchomić aplikację napisaną w Javie musimy mieć większy zestaw narzędzi o nazwie Java JDK (Java Development Kit). Znajduje się tam przede wszystkim kompilator. Cały pakiet dostępny jest również na stronie [www.oracle.com](http://www.oracle.com).

Należy przede wszystkim pamiętać, że Java stworzona jest na licencji GNU

Moduł szkoleniowy Java str. 10

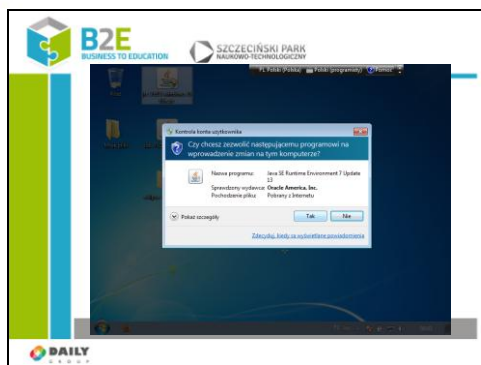
Człowiek - najlepsza inwestycja

Bardzo ważnym elementem jest również środowisko pracy. Stanowczo odradzam korzystanie z prostych edytorów tekstowych. Jednym z najpopularniejszych środowisk jest Eclipse. Można pobrać również środowisko NetBeans lub IntelliJ IDEA. Wszystkie trzy oferują podobne możliwości i są powszechnie wykorzystywane na całym świecie. Zalecam pobranie pakietu Eclipse Classic, ponieważ właśnie w nim będę pokazywał przykłady.

W pierwszej kolejności instalujemy pakiet Java JRE



Slajd 5



Slajd 6



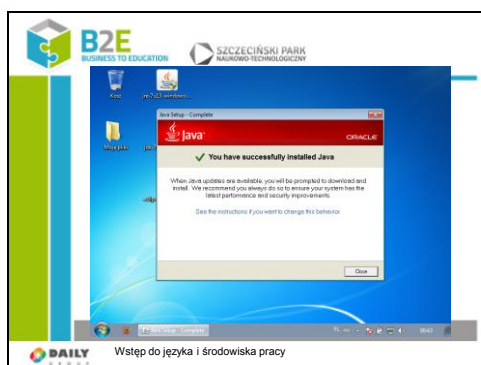
Przechodzimy kolejne kroki instalacji.

Slajd 7

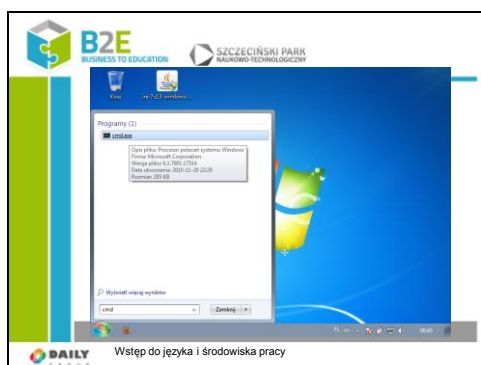




Slajd 8



Slajd 9

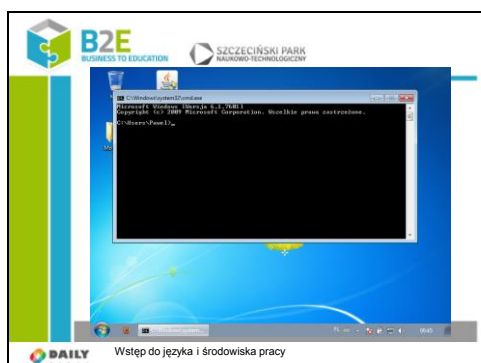


Po zakończeniu instalacji możemy korzystać z oprogramowania stworzonego w języku Java.

W dalszych slajdach pokazane będzie jak uruchomić aplikację. Dla zaprezentowania posiadam już gotowy plik wykonywalny. Proszę nie wykonywać tej części samodzielnie.

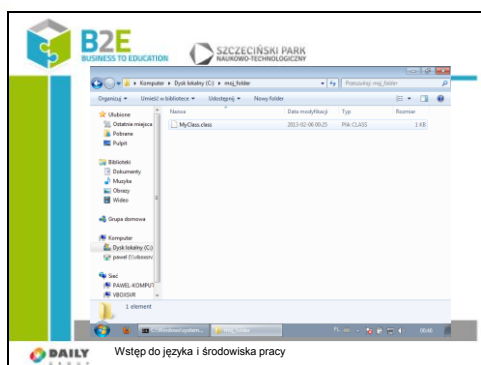
W tym celu należy uruchomić program cmd.exe.

Slajd 10



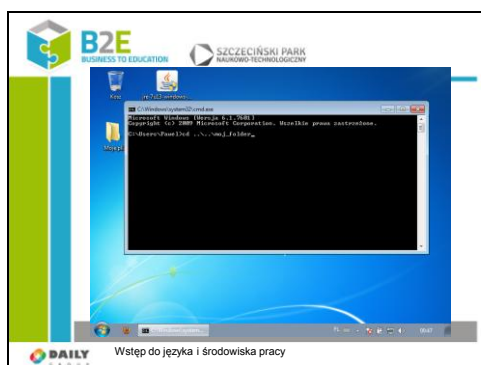


Slajd 11



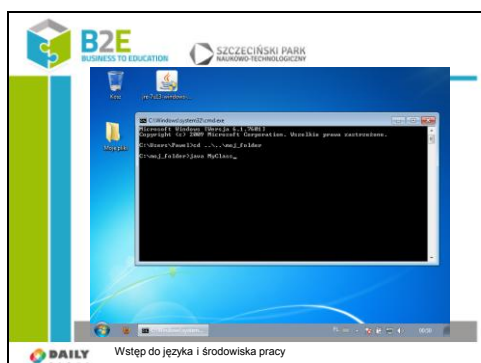
Pliki wykonywalne dla maszyny wirtualnej Javy posiadają rozszerzenie class.

Slajd 12



Przechodzimy do folderu w którym znajduje się nasza klasa.

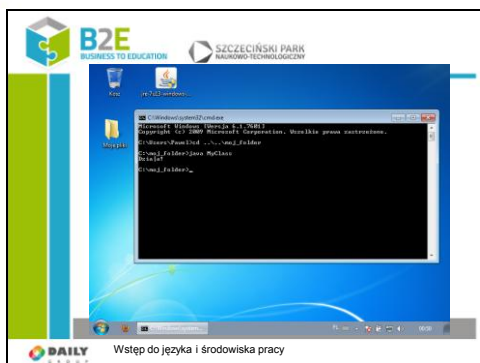
Slajd 13



Poleceniem java uruchamiamy maszynę podając jako parametr nazwę pliku bez rozszerzenia.

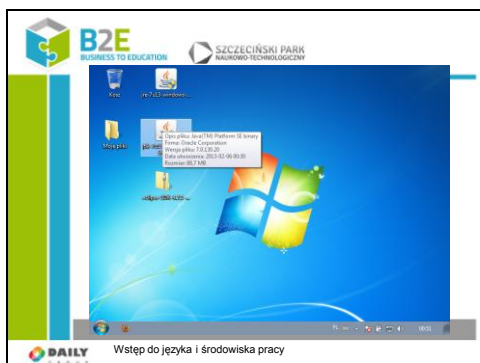


Slajd 14



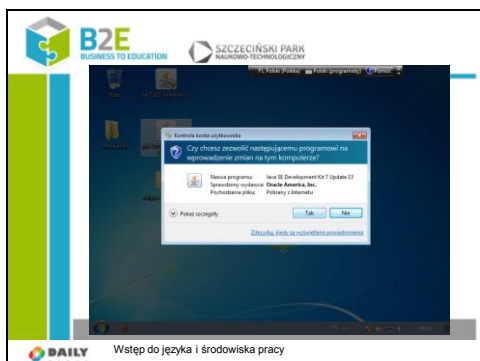
W wyniku otrzymaliśmy napis „Działa!”. Nie wyświetliła się jednak literka „ł”. Spowodowane jest to kodowaniem znaków cp1250 w programie cmd.exe. W Javie symbole zajmują 16 bitów, więc bez żadnych ograniczeń można stosować znaki regionalne.

Slajd 15



Aby móc zacząć programować w Javie potrzebujemy kompilatora z pakietu Java JDK.

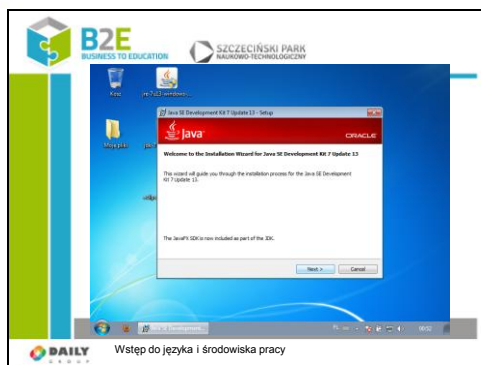
Slajd 16



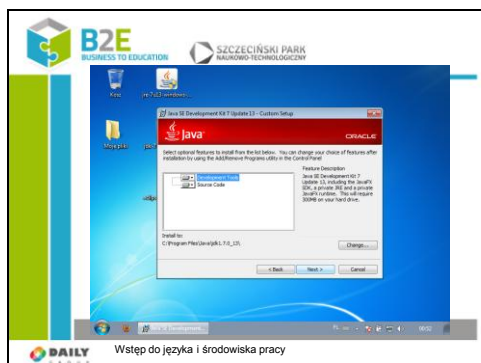
Kolejno wykonujemy polecenia instalatora.



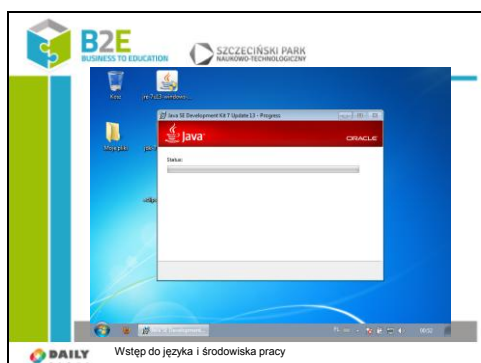
Slajd 17



Slajd 18



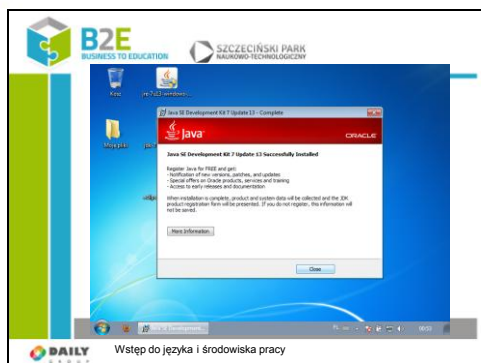
Slajd 19



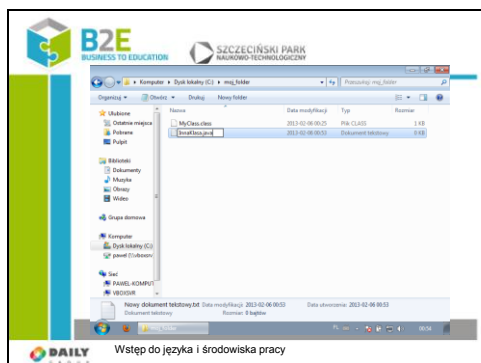




Slajd 20



Slajd 21

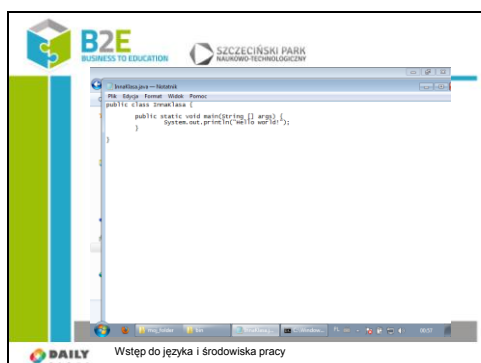


Po zakończonej instalacji Java JDK. Możemy napisać pierwszy program.

Możemy już wykonać pierwsze praktyczne ćwiczenie.

Stwórzmy plik o dowolnej nazwie, bez białych znaków i zaczynający się dużą literą, z rozszerzeniem \*.java.

Slajd 22



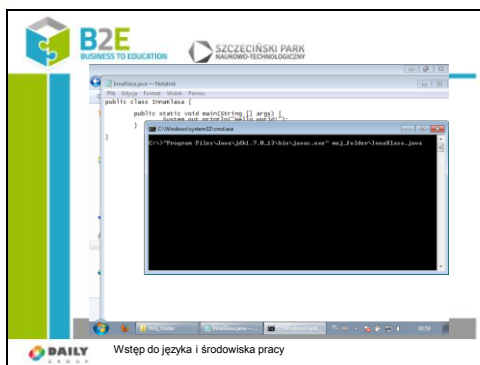
Do pliku wstawiamy ten fragment kodu.

Uwaga!

Jeżeli użyta została inna nazwa pliku niż „InnaKlasa” to należy ją wpisać w pierwszym wersie (bez rozszerzenia) zamiast wyrażenia „InnaKlasa”.



Slajd 23

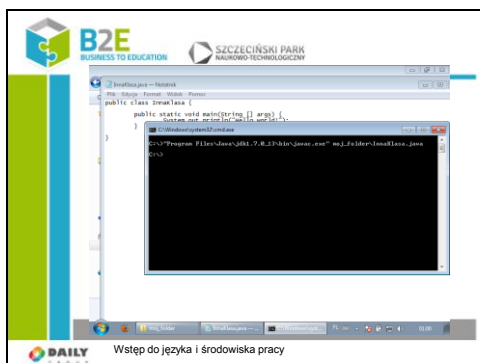


Kompilujemy nasz kod źródłowy widocznym poleceniem.

Pierwsza część jest ścieżką do kompilatora Javy – javac.exe.

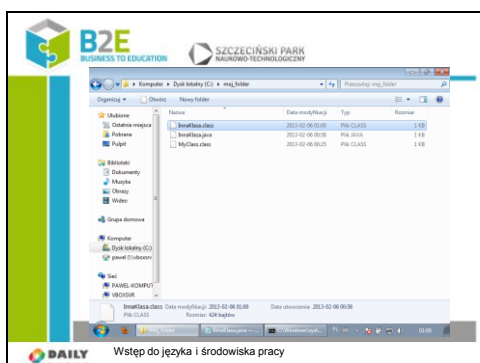
Argumentem przyjmowanym przez kompilator jest ścieżka do pliku z kodem źródłowym.

Slajd 24



Tak wygląda rezultat poprawnie wykonanego polecenia kompilacji (brak jakichkolwiek wiadomości i ponowne wyświetlenie znaku zachęty).

Slajd 25

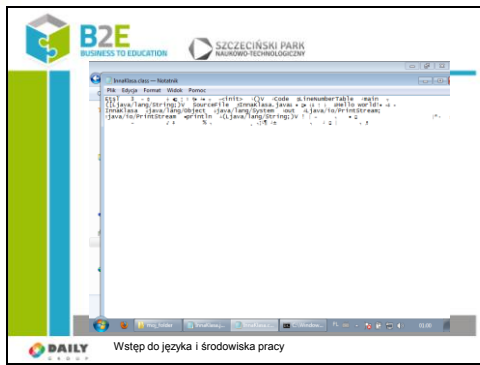


Jedyną widoczną zmianą jest pojawienie się pliku o takiej samej nazwie jaką wybraliśmy do pliku z rozszerzeniem „.java”.

Nowy plik posiada rozszerzenie „.class”. Zawiera on kod bajtowy przeznaczony do uruchamiania na każdej maszynie virtualnej.



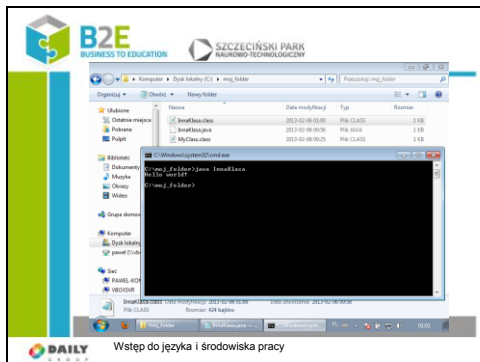
Slajd 26



Możemy wyświetlić zawartość nowego pliku w edytorze tekstowym.

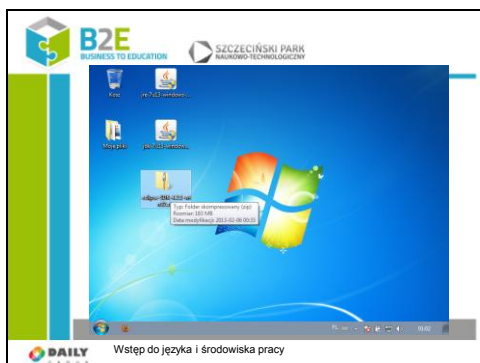
Jak widać nie jest to kod maszynowy. Wiele części można odczytać, np. „SourceFile” jest to plik źródłowy z którego powstał dany kod bajtowy, „java/lang/String” jest to adres obiektu, który przechowuje ciąg znaków.

Slajd 27



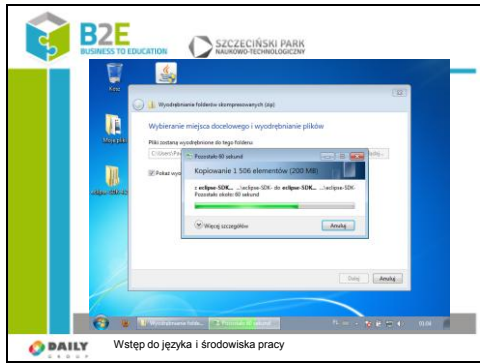
Omawiany plik możemy uruchomić poleceniem pokazanym w poprzednim przykładzie.

Slajd 28



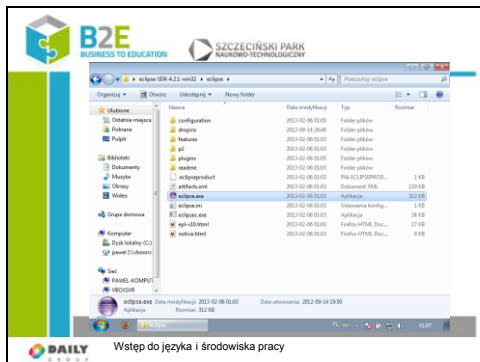
Pisanie kodu w prostym edytorze tekstowym jest trudne. Będziemy korzystać ze środowiska Eclipse w celu usprawnienia pracy.

Slajd 29



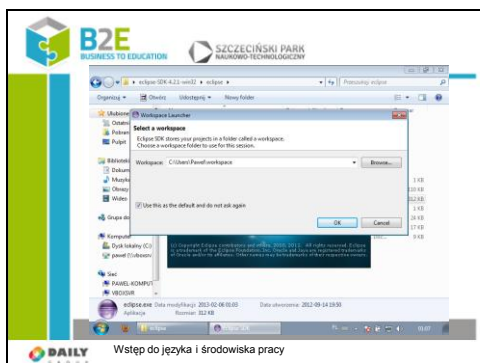
Wypakowujemy pobrane archiwum.

Slajd 30



Środowisko uruchamiamy za pomocą pliku „eclipse.exe”.

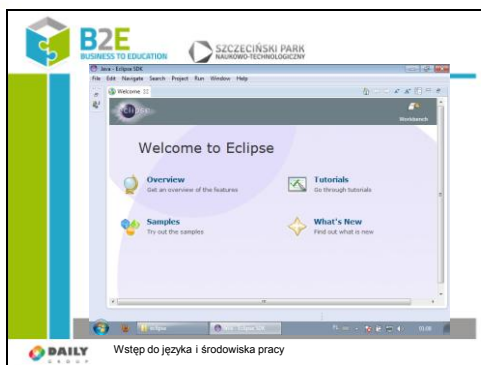
Slajd 31



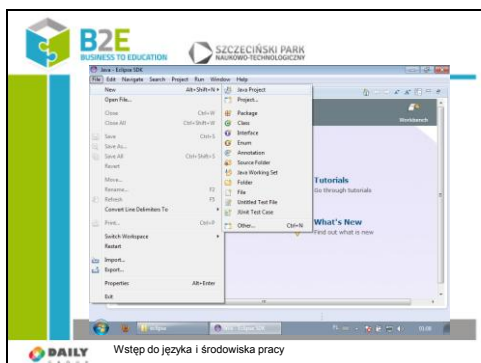
Przy pierwszym uruchomieniu trzeba wybrać folder pełniący funkcję „workspace”. Jest to lokalizacja w której znajdować się będą wszystkie nasze projekty.



Slajd 32



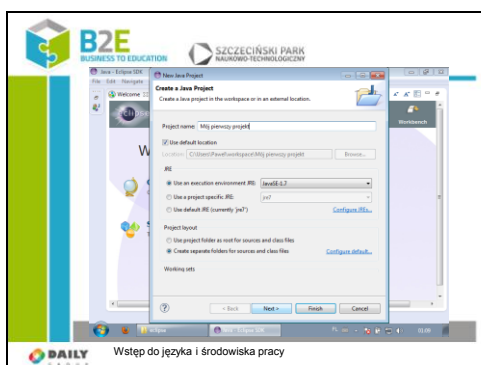
Slajd 33



Zanim przystąpimy do pisania kodu, musimy stworzyć projekt.

W tym celu wybieramy:  
File/New/Java Project

Slajd 34



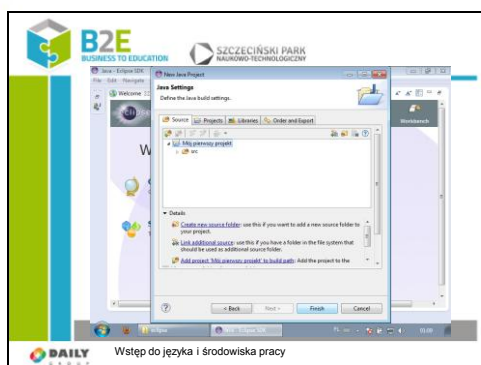
Nazwa projektu (Project name) może być dowolna.

Pozostałe parametry najlepiej pozostawić z ustawieniami domyślnymi.

Klikamy „Next”.

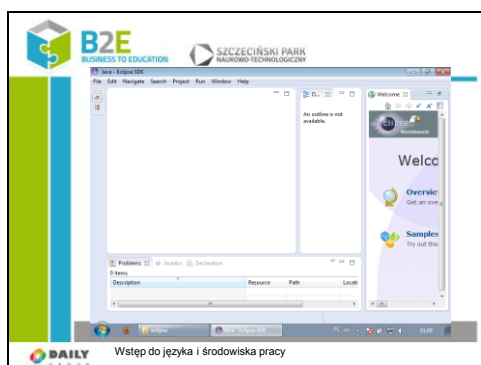


Slajd 35



Klikamy „Finish”.

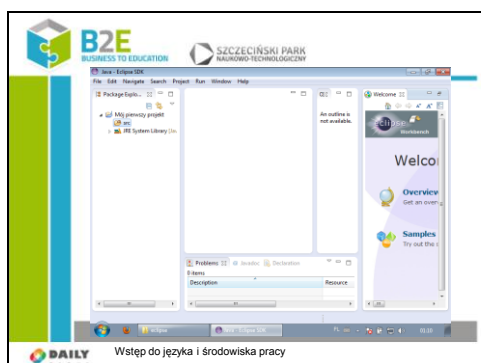
Slajd 36



Aby zobaczyć zawartość projektu klikamy na ikonę znajdującą się w lewym, górnym rogu (poniżej pozycji w menu o nazwie „File”).

Powinien wyświetlić się nam „Package Explorer”

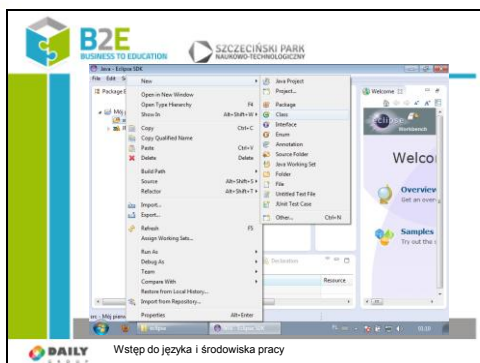
Slajd 37



Pliki kodu źródłowego przechowywane są w folderze „src”.

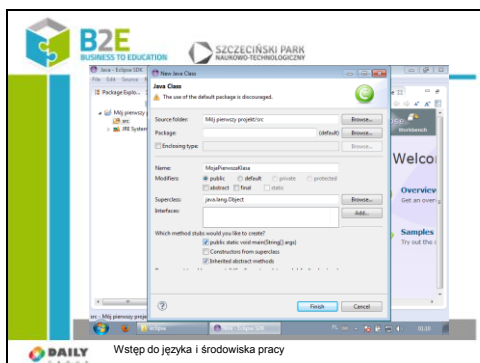
Nowy projekt nie posiada żadnych plików. W celu stworzenia nowego należy kliknąć na folderze „src” prawym przyciskiem myszy.

Slajd 38



Następnie wybieramy:  
New/Class.

Slajd 39

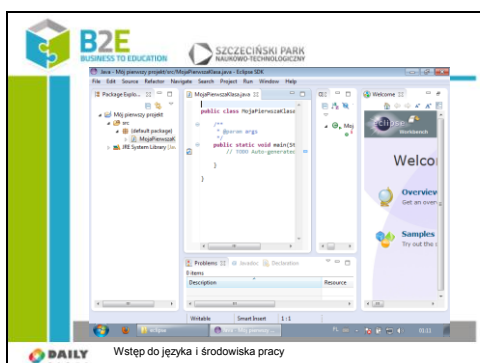


Podajmy nazwę klasy. Nie używamy spacji i zaczynamy dużą literą.

Zaznaczamy również opcję:  
„public static void main (String [] args)”.

Całość zatwierdzamy przyciskiem „Finish”.

Slajd 40

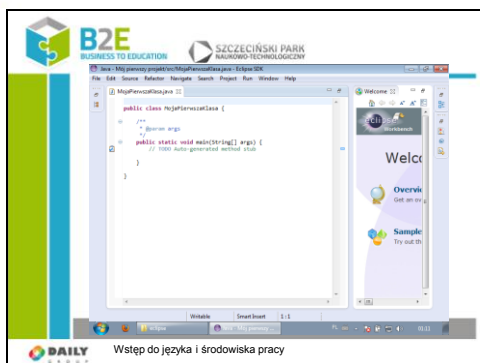


Na środku ekranu wyświetliła się zawartość pliku. Znajduje się on w folderze „src” w (defaultpackage)”.



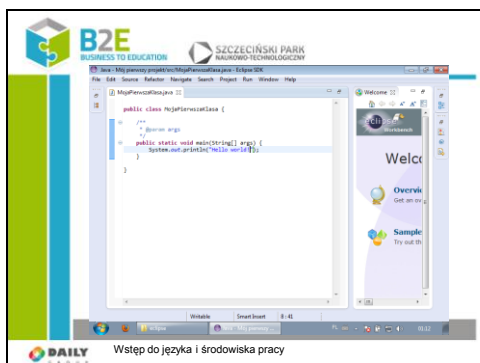


Slajd 41



Jeżeli chcemy wyświetlić plik na całej szerokości ekranu, klikamy podwójnie na zakładce z zawartością pliku.

Slajd 42

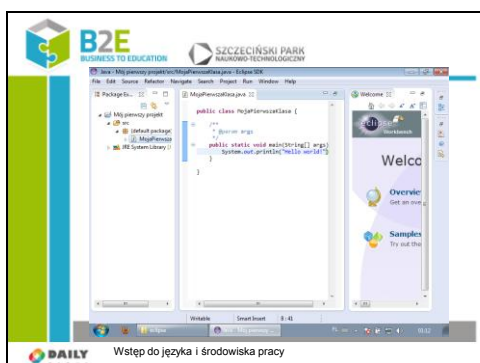


Wers zaczynający się na „//” zastępujemy wyrażeniem:

```
System.out.println("Hello world!");
```

Skrótem klawiszowym „Ctrl” + „S” zapisujemy zmiany.

Slajd 43



Ponownie klikamy podwójnie na nazwie karty w celu jej zmniejszenia.

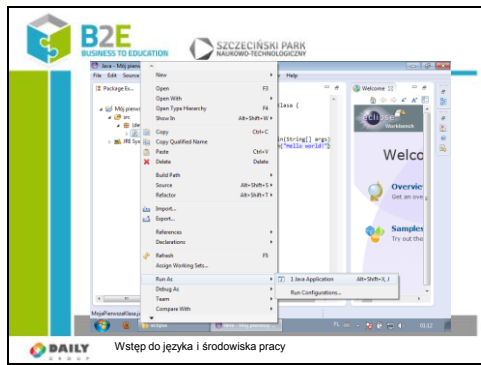
Dzięki temu ponownie widzimy strukturę plików w projekcie.

W celu uruchomienia pliku klikamy w obszarze „Package Explorer” na nim prawym klawiszem myszy.



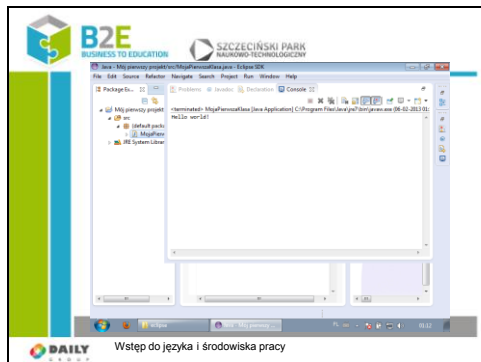


Slajd 44



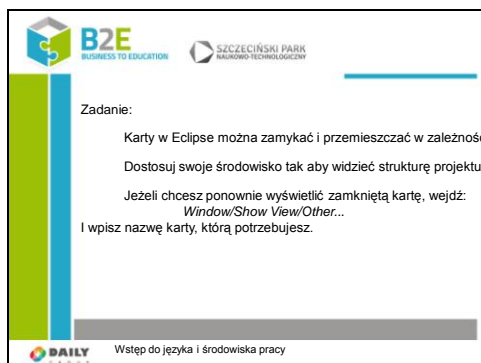
Wybieramy opcję:  
„Run As/1 Java Application”

Slajd 45



Jako rezultat operacji powinniśmy zobaczyć konsolę z widocznym napisem „Hello world!”.

Slajd 46



Zadanie:

Karty w Eclipse można zamykać i przemieszczać w zależności

Dostosuj swoje środowisko tak aby widzieć strukturę projektu.

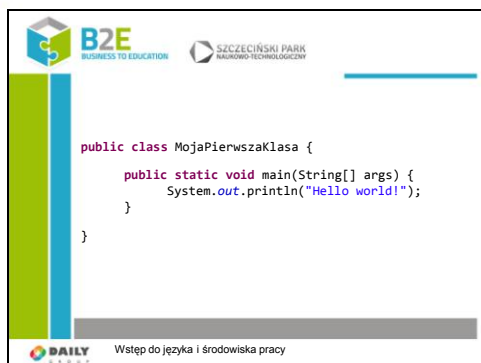
Jeżeli chcesz ponownie wyświetlić zamkniętą kartę, wejdź:

*Window/Show View/Other...*

I wpisz nazwę karty, którą potrzebujesz.



Slajd 47



Przeanalizujmy kod pierwszego programu.

Wszystko w Javie jest obiektem. Obiekt jest instancją danej klasy, czyli pewnym tworem przechowywanym w pamięci, który przechowuje dane i nie tylko. Pojęcie obiektu zostanie dokładnie wytłumaczone na lekcji 3.

W Javie w każdym pliku znajduje się jedna klasa. Później przedstawię jak definiować klasy wewnątrz innych klas. Jednak po procesie kompilacji zawsze klasa jest osobnym plikiem.

Bardzo ważne jest aby klasa miała taką samą nazwę jak plik, w którym się znajduje. Pomińcie tej zasady traktowane jest jako błąd!

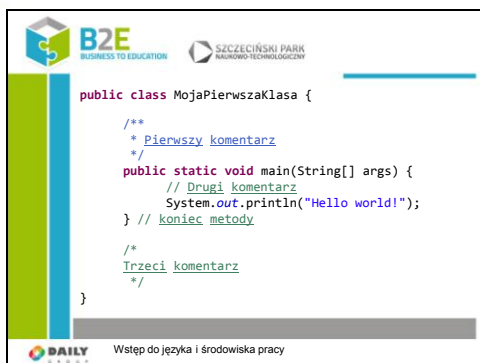
Wewnątrz klasy (pomiędzy nawiasami „{” i „}”) znajdują się pola i metody. W naszym przykładzie nie ma pól, jest tylko jedna metoda o nazwie „main”.

W późniejszych lekcjach zostaną również wytłumaczone znaczenia słów „public”, „static” i „void”. Na razie wszystkie polecenia będziemy pisać we wnętrzu metody „main”. Jest to szczególna metoda, ponieważ jest zawsze wywoływana przy próbie uruchomienia obiektu.

Obiekt przeznaczony do, bezpośredniego uruchomienia musi posiadać metodę: „public static void main(String [] args)”.



Slajd 48



W Javie, w prawie dowolnym miejscu, możemy pisać komentarze. Są one przeznaczone wyłącznie dla programistów.

Najczęściej spotykanymi komentarzami są te zaczynające się od symboli „/”. Możemy stosować je w tych samych liniach, w których są elementy kodu. Po wstawieniu symboli „/” kompilator pominie cały tekst, aż do końca linii.

Innym typem komentarzy są komentarze blokowe. Otwieramy je symbolem „/\*” i zamykamy „\*/”. Pomiędzy tymi znacznikami możemy wstawić dowolną liczbę wersów.

Dość szczególnym typem komentarzy blokowych są te zaczynające się symbolami „/\*\*”. Zostawia się je przed definicją klasy i jej metodami. Nazywane są „Javadoc”. Pełnią ważną funkcję w tworzeniu dokumentacji. W późniejszych lekcjach zostanie pokazane jak je tworzyć, oraz jak korzystać z tej dokumentacji.

Pisanie komentarzy jest bardzo ważne. W dużych projektach, nad którymi pracuje wiele osób, są one praktyczną formą zostawiania wiadomości. Jeżeli napiszemy fragment kodu, o bardzo skomplikowanej logice, warto go skomentować. W takim przypadku bardzo przydatne mogą okazać się nawet lakoniczne informacje o podstawowych zasadach działania kodu. Bardzo ułatwią one rozwijanie aplikacji osobą, które będą pracowały po nas.

Praca zespołowa nie jest jedynym powodem, dla którego pisze się komentarze. Jeżeli zechcemy po pewnym czasie powrócić do rozwijania swojej aplikacji, możemy nie pamiętać szczegółów dotyczących kodu. Oszczędzimy wiele czasu czytając komentarze, zamiast szczegółowo analizować kod.

Dla kompilatora bieżący kod jest identyczny jak ten na poprzednim slajdzie.



Slajd 49

```

System.out.println("Hello world!");

System.out.print("Hello");
System.out.println(" world!");

System.out.println(2+2);

Wynik:
Hello world!
Hello world!
4

```

Wstęp do języka i środowiska pracy

Metoda:

„System.out.println”

Wyświetla tekst podany w nawiasach (i w znakach „”) w konsoli i wstawia znak nowej linii.

W pakiecie „System.out” są również inne metody, np. „print” wstawia tekst bez znaku nowej linii.

Metoda „println” wyświetla nie tylko tekst. W Javie wszystkie obiekty mogą być wyświetlone w postaci tekstu. Będziemy wykorzystywać ją na najbliższych lekcjach do sprawdzania wyników przebiegu programów.

Jest to jedna z najpowszechniej używanych metod. W środowisku Eclipse możemy wywołać ją automatycznie wpisując „sysou” i następnie wciskając „Ctrl” + „spacja”.

Skrót klawiszowy „Ctrl” + „spacja” jest powszechnie wykorzystywany (nie tylko w Eclipse!) do podpowiadania elementów składni. Będziemy często go wykorzystywać.

Slajd 50

```

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

    System.out.println
    (2+2);
}

```

Wstęp do języka i środowiska pracy

W Javie każde polecenie musi kończyć się średnikiem. Nie oznacza to jednak, że średnik musi znajdować się na końcu każdej linii. Jeżeli nasze polecenia są zbyt długie (nie zaleca się stosować wersów o długości 200 znaków, są nieczytelne), możemy umieścić je w kilku wersach, pamiętając o zakończeniu całości średnikiem

Na slajdzie znajduje się poprawny kod programu, wykonujący to co zostało przedstawione na slajdzie poprzednim. Należy zauważyć jednak, że polecenia zapisane w taki sposób nie są czytelne dla ludzi.

Średnika nie wstawiamy, np. po nawiasach klamrowych.

### 8.1.3 Opis założonych osiągnięć ucznia

Po tej lekcji uczniowie poznają teoretyczną widzę na temat tego czym jest Java. Będą umieli rozpocząć pracę w Javie w odpowiednim środowisku.

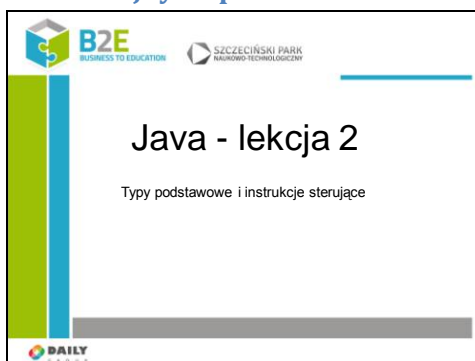
## 8.2 Lekcja 2 - Typy podstawowe i instrukcje sterujące

### 8.2.1 Cel lekcji

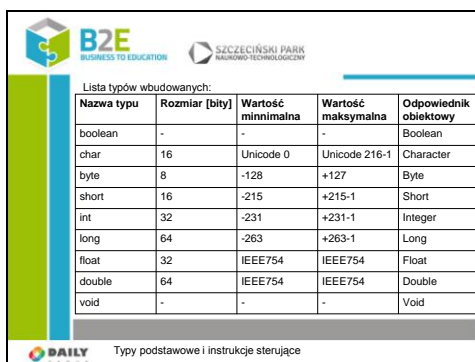
Celem lekcji jest poznanie podstawowych typów danych, wytłumaczenie jak bezpiecznie przechowywać dane w zmiennych. Poznanie pętli sterujących i zarządzanie przebiegiem wykonania programu za ich pomocą.

### 8.2.2 Treść - slajdy z opisem

Slajd 1



Slajd 2



Nazwa typu	Rozmiar [bity]	Wartość minimalna	Wartość maksymalna	Odpowiednik obiektowy
boolean	-	-	-	Boolean
char	16	Unicode 0	Unicode 216-1	Character
byte	8	-128	+127	Byte
short	16	-215	+215-1	Short
int	32	-231	+231-1	Integer
long	64	-263	+263-1	Long
float	32	IEEE754	IEEE754	Float
double	64	IEEE754	IEEE754	Double
void	-	-	-	Void

Typy wbudowane są szczególnym typem danych. Występują ponieważ traktowanie prostych danych jako obiekty nie jest szczególnie wydajne. Zmienne te przechowują proste wartości

Typ „boolean” służy do przechowywania wartości logicznej: prawda lub fałsz.

Typ „char” służy do przechowywania znaku (liter). Posiada 16 bitów, więc możemy swobodnie używać polskie znaki.

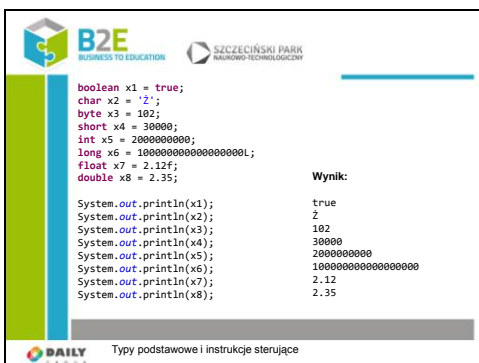
Typ „byte” służy do przechowywania bajtów danych.

Typy „short”, „int” i „long” służą do przechowywania liczb całkowitych. W zależności od zakresu na jakim chcemy operować wybieramy odpowiedni typ.

Typy „float” i „double” przechowują liczby zmiennoprzecinkowe zgodnie ze standardem IEEE754.

Typ „void” oznacza zerową informację. Nie można stworzyć zmiennej typu „void”, ale może on być zwracany przez metody.

Slajd 3



```
boolean x1 = true;
char x2 = '2';
byte x3 = 102;
short x4 = 30000;
int x5 = 2000000000;
long x6 = 1000000000000000000L;
float x7 = 2.12f;
double x8 = 2.35;

System.out.println(x1);
System.out.println(x2);
System.out.println(x3);
System.out.println(x4);
System.out.println(x5);
System.out.println(x6);
System.out.println(x7);
System.out.println(x8);
```

Wynik:

```
true
2
102
30000
2000000000
1000000000000000000
2.12
2.35
```

Typy podstawowe i instrukcje sterujące



Na slajdzie pokazane jest jak powinno się inicjalizować zmienne (nadawać wartości):

- boolean – używamy słów „true” lub „false”,
- char – podajemy pojedynczy znak w apostrofach,
- byte, short, int – liczba całkowita,
- long – liczba całkowita z literą „l” lub „L” na końcu,
- float – liczba ułamkowa z literą „f” lub „F” na końcu,
- double – liczba ułamkowa.

W celu wyświetlenia zainicjowanych wartości używamy wcześniej przedstawionej metody „println”.

Jak widać w wyniku nasze zmienne przechowują dokładnie takie wartości jak podaliśmy.


## Slajd 4

```
float f = 3.1415926535897932384626433f;
double d = 3.1415926535897932384626433f;
System.out.println("3.1415926535897932384626433");
System.out.println(f);
System.out.println(d);
```

**Wynik:**

```
3.1415926535897932384626433
3.1415927
3.1415927410125732
```



 Typy podstawowe i instrukcje sterujące

Dokonując operacji na typach zmiennoprzecinkowych należy szczególnie pamiętać o dokładności wykonywanych obliczeń.

Przykład pokazuje dokładność z jaką typy „float” i „double” przechowują liczbę Pi.

Nie oznacza to jednak, że w Javie nie można przeprowadzać obliczeń o wyższej precyzji. W standardowych bibliotekach Javy znajdują się klasy takie jak „BigInteger” i „BigDecimal”, które mogą wykorzystać do zapamiętania liczby całą dostępną pamięć RAM komputera. Są to jednak obiekty. Jak korzystać z obiektów zostanie wytłumaczone w następnej lekcji.

## Slajd 5





```
int a = 8, b = 2, c;
c = a + b;
System.out.println(c);
System.out.println(a+b);
System.out.println(a-b);
System.out.println(b-a);
System.out.println(a*b);
System.out.println(a/b);
System.out.println(c/a);
System.out.println(c%a);

a += a;
System.out.println(a);
a -= c;
System.out.println(a);
a *= a;
System.out.println(a);
a /= b;
System.out.println(a);
```

**Wynik:**

```
10
10
6
-6
16
4
1
2
16
6
36
18
```

 Typy podstawowe i instrukcje sterujące

Na typach wbudowanych możemy wykonywać operacje matematyczne korzystając z dostępnych operatorów.

Operator „%” służy do otrzymywania reszty z dzielenia, pozostałe działają zgodnie z intuicją.

Przypisanie sumy liczb a i b do c nastąpiło poprzez użycie operatora podstawiania („=”).



Operatory „+”, „-” i „\*” działają zgodnie z intuicją (suma, różnica, mnożenie).

Należy zwrócić szczególną uwagę na operator „/” dzielenia. W przypadku gdy dzielimy a przez b dostajemy poprawny wynik ( $8/2=4$ ). Kiedy jednak dzielimy c przez a ( $10/8=1,25$ ) otrzymaliśmy wynik 1 zamiast 1,25. Dzieje się tak, ponieważ typy te nie przechowują części ułamkowej. Nie następuje również zaokrąglenie wartości. Cyfry znajdujące się po przecinku są „odcinane”, więc zamiast np. 1,8 też dostaniemy 1.

Operatory użyte poniżej biorą argumenty z obu stron, przypisując wartość do zmiennej znajdującej się z lewej strony. Wykonują analogicznie operacje sumy, różnicy, mnożenia i dzielenia.




## Slajd 6

```
int i = 1;
// preinkrementacja
System.out.println(++i);
// postinkrementacja
System.out.println(i++);
// predekrementacja
System.out.println(--i);
// postdekrementacja
System.out.println(i--);
```

Wynik:

```
2
2
2
2
```

 Typy podstawowe i instrukcje sterujące

Operator preinkrementacji najpierw zwiększa zmienną o jedną jednostkę, następnie zwraca jej wartość. 1 zostało zwiększone, a następnie wyświetlone. Po operacji  $i=2$ .

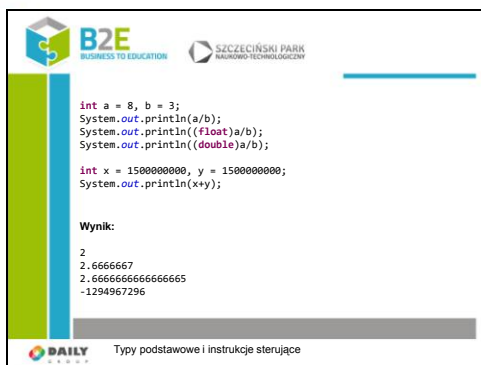
Operacja postinkrementacji najpierw zwraca wartość, a następnie ją zwiększa. 2 zostało najpierw wyświetlone, a następnie zwiększone. Po tej operacji  $i=3$ .

Operator predekrementacji najpierw zmniejsza zmienną o jedną jednostkę, następnie zwraca jej wartość. 3 zostało zmniejszone, a następnie wyświetlone. Po operacji  $i=2$ .

Operacja postdekrementacji najpierw zwraca wartość, a następnie ją zmniejsza. 2 zostało najpierw wyświetlone, a następnie zmniejszone. Po tej operacji  $i=1$ .



## Slajd 7



```
int a = 8, b = 3;
System.out.println(a/b);
System.out.println((float)a/b);
System.out.println((double)a/b);

int x = 1500000000, y = 1500000000;
System.out.println(x/y);
```

Wynik:

```
2
2.6666667
2.6666666666666665
-1294967296
```

DAILY Typy podstawowe i instrukcje sterujące

Fakt, że dla typów „int”, „short”, „long” wartości mniejsze od 1, są obcinane, nie oznacza, że nie możemy ich używać jeżeli w przyszłości może zająć potrzeba dokonania dokładniejszych obliczeń. Istnieje możliwość rzutowania jednych typów na drugie.

Na slajdzie mamy trzy możliwe wyniki dzielenia 8 przez 3 w zależności od dokładności wyniku jaki chcemy otrzymać.

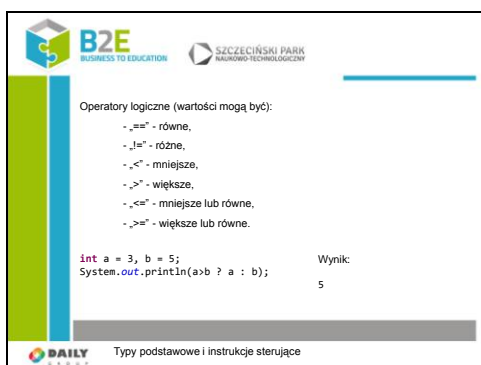
Poniżej mamy dość szczególny przypadek. Zastanówmy się, jak to jest możliwe, że dodając dwie liczby dodatnie, otrzymaliśmy ujemny wynik.

Proszę wyobrazić sobie, że typy dla liczb całkowitych mogą się „przekręcać”.

Dobłą analogią do tego zjawiska jest przykład licznika samochodowego. Jeżeli wartość na nim pokazywana będzie cały czas rosła, to przekroczy wartość maksymalną i ponownie przekroczy zero. Z omawianymi typami dziej się dokładnie tak samo.

Wartość maksymalna dla „int” to 2<sup>31</sup>-1 (około 2 mld). Jeżeli zechcemy zapisać 3 mld, to „licznik się przekręci” i do minimalnej wartości -2<sup>31</sup> (około -2 mld) dodana zostanie wartość będąca różnicą 3 mld – 2mld = 1 mld. Otrzymaliśmy w ten sposób wynik około -1 mld.

## Slajd 8



Operatory logiczne (wartości mogą być):

- „==” - równe,
- „!=” - różne,
- „<” - mniejsze,
- „>” - większe,
- „<=” - mniejsze lub równe,
- „>=” - większe lub równe.

```
int a = 3, b = 5;
System.out.println(a & b ? a : b);
```

Wynik:

```
5
```

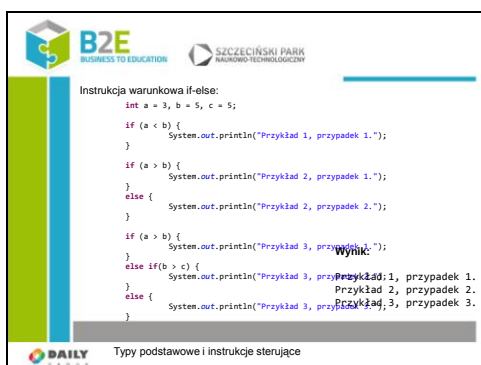
DAILY Typy podstawowe i instrukcje sterujące

Przedstawione operatory służą do porównywania wartości. Działają zgodnie z tym, czego używaliśmy na lekcjach matematyki.

Poniżej mamy przykład jedynego trójargumentowego operatora w Javie. Składa się z dwóch znaków: „?” i „:”. W pierwszej kolejności podajemy wartość logiczną. Wynikiem każdej operacji logicznej jest wartość logiczna. Możemy w tym miejscu równie dobrze użyć zmiennej typu boolean. Następnie, po znaku „?” podajemy polecenie, które ma zostać wykonane w przypadku prawdy. Po znaku „:” podajemy polecenie dla fałszywej wartości logicznej.

Wynikiem użycia operatora „?” w przykładzie pokazanym na slajdzie jest drukowanie w konsoli wartości większej (a lub b).

Slajd 9



```

Instrukcja warunkowa if-else:
int a = 3, b = 5, c = 5;
if (a < b) {
    System.out.println("Przykład 1, przypadek 1.");
}
if (a > b) {
    System.out.println("Przykład 2, przypadek 1.");
}
else {
    System.out.println("Przykład 2, przypadek 2.");
}
if (a > b) {
    System.out.println("Przykład 3, przypadek 1.");
}
else if (b > c) {
    System.out.println("Przykład 3, przypadek 1.");
}
Przykład 2, przypadek 2.
else {
    System.out.println("Przykład 3, przypadek 3.");
}
    
```

Wynik:

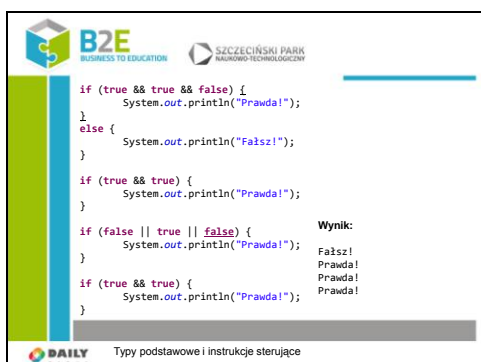
Przykład 2, przypadek 2.

Typy podstawowe i instrukcje sterujące

Na slajdzie przedstawione zostały różne warianty instrukcji warunkowej „if”. Korzystając z niej decydujemy, które polecenia mają być wykonane w zależności od zadanych kryteriów.

Instrukcja warunkowa „if” działa w następujący sposób. Warunek w nawiasie po słowie „if” musi być wyrażeniem logicznym (dawać wartość prawda lub fałsz). W przypadku prawdy wykonają się instrukcje zawarte w nawiasach klamrowych, za wyrażeniem logicznym. Jeżeli później występuje słowo „else”, to w przypadku prawdy wykonają się instrukcje występujące po słowie else.

Slajd 10



```

if (true && true && false) {
    System.out.println("Prawda!");
}
else {
    System.out.println("Fałsz!");
}
if (true && true) {
    System.out.println("Prawda!");
}
if (false || true || false) {
    System.out.println("Prawda!");
}
if (true && true) {
    System.out.println("Prawda!");
}
    
```

Wynik:

Fałsz!

Prawda!

Prawda!



Prawda!

Typy podstawowe i instrukcje sterujące

Dostępne mamy również inne operatory logiczne: koniunkcji („&&”) i alternatywy („||”).


Należy pamiętać, że kiedy użyjemy operatora „&&”, to kiedy przynajmniej jedna z wartości jest fałszywa, całe wyrażenie jest fałszywe. Dlatego wyrażenia sprawdzane są od lewej do prawej. Po napotkaniu pierwszej fałszywej wartości reszta nie jest już sprawdzana. Nie zachodzi taka potrzeba.

Slajd  
11

```
int a = 5, b = 7, c = 0, y = 0;
while (c < b) {
    y += a;
    c++;
}
System.out.println(y);
```

Wynik:  
35



 Typy podstawowe i instrukcje sterujące

Na slajdzie widzimy przykładową pętlę „while”. Wykonuje ona operację mnożenia a i b, zapisując wynik do c.

Po słowie „while” występuje wyrażenie logiczne, tak jak w instrukcji „if”. W przypadku prawdy instrukcje z nawiasów klamrowych zostaną wykonane. Następnie wyrażenie logiczne jest sprawdzane ponownie i w przypadku prawdy następuje kolejny krok pętli. Instrukcja „while” przestaje działać jeżeli warunek (wyrażenie logiczne w nawiasach) jest fałszywy.

Jeżeli źle skonstruujemy warunek logiczny, pętla może się nie zatrzymać! W takim przypadku proces w systemie operacyjnym, który odpowiada naszemu programowi, powinien być „zabity”.

Slajd  
12





**Ćwiczenie:**



Wykorzystując pętlę „while” napisz program obliczający 10 liczb ciągu Fibonacciego. Każda liczba ciągu powstaje poprzez dodanie dwóch poprzednich liczb, a d

Przykład:

```
Liczba 1 = 1
Liczba 2 = 1
Liczba 3 = 1 + 1 = 2
Liczba 4 = 1 + 2 = 3
```


 Typy podstawowe i instrukcje sterujące

Slajd  
13

```
int a = 1, b = 1;
do {
    a *= b++;
    System.out.println(a);
} while (a < 1000);
```

Wynik:  
1  
2  
6  
24  
120  
720  
5040

 Typy podstawowe i instrukcje sterujące

Pętla „do-while” działa tak samo jak pętla „while”, z tą różnicą, że zawsze występuje pierwszy przebieg pętli. Dopiero po nim sprawdzany jest warunek. W przypadku prawdy następuje kolejny przebieg.

Pętla widoczna na slajdzie wypisuje kolejne wartości silni, aż do momentu, kiedy wartość będzie większa niż 1000.

Slajd  
14






Ćwiczenie:

Wykorzystując pętlę „do-while” napisz program znajdujący najmniejszy dzielnik

Typy podstawowe i instrukcje sterujące

Slajd  
15

```
for (instrukcja początkowa ; warunek ; krok) {}

int a = 2, b = 9, c = 1;
for (int i=0; i<b; i++) {
    c *= a;
    System.out.println(c);
}

Wynik:
512
```

```
// to działa tak samo
int i=0;
for (; i<b; ) {
    i++;
    c *= a;
}

// to jest poprawne
for (;) {}
```

Typy podstawowe i instrukcje sterujące

Pętla „for” jest bardziej złożona od poprzednich przykładów. W nawiasie po słowie „for” nie występuje tylko warunek. Wyrażenie ma 3 części:

Pierwsza część, instrukcja początkowa, wykonywana tylko raz na samym początku. W naszym przypadku zadeklarowaliśmy i zainicjowaliśmy zmienną i.

Druga część, warunek, jest sprawdzana w każdym kroku. Musi to być wyrażenie logiczne. Dopóki wartość jest prawdziwa, dopóty wykonuje się pętla.



Trzecia część, to instrukcja, która wykonuje się po zakończeniu każdego kroku.

Przedstawiony przykład oblicza wartość liczby a podniesioną do potęgi b.

Każda z trzech instrukcji pętli „for” może być pusta. Z prawej strony slajdu pokazany jest przykład, który działa identycznie jak omawiana pętla. Jednak takie zapisywanie pętli jest mało czytelne dla ludzi.

Pętla „for”, przedstawiona w prawym dolnym rogu, również jest poprawna. Powoduje ona zawieszenie się wykonywanego programu, z powodu braku instrukcji warunkowej. Jest to przykład pętli nieskończonej.


Slajd  
16

```
for (int i=0, a=0 ; i<100 ; i++) {
    if (i>10) {
        break;
    }
    if (i%2 == 0) {
        continue;
    }
    a += i;
    System.out.println(a);
}
```

Wynik:

```
1
4
9
16
25
```

 Typy podstawowe i instrukcje sterujące

Przy operowaniu na pętlach możemy skorzystać z dwóch instrukcji: „break” i „continue”.



Przy napotkaniu słowa „continue”, wykonanie bieżącego kroku pętli zostaje zatrzymane i zaczyna się wykonywanie kroku następnego.

Po wystąpieniu słowa „break” następuje całkowite wyjście z pętli „for”.

Powyższy przykład sumuje wszystkie liczby nieparzyste od 1 do 10. Instrukcja warunkowa w pętli „for” w naszym przypadku nie ma żadnego znaczenia. Polecenie „break” przerywa działanie pętli kiedy tylko *i* będzie większe od 10.

Kiedy reszta z dzielenia *i* przez 2 będzie równa 0 (liczba parzysta), nie nastąpi operacja sumy i wypisania na ekran.


Slajd  
17

```
int a=2;
switch (a) {
    case 1:
        System.out.println("Wartość 1.");
        break;
    case 2:
        System.out.println("Wartość 2.");
    default:
        System.out.println("Inna wartość.");
}
```

Wynik:

```
Wartość 2.
Inna wartość.
```

 Typy podstawowe i instrukcje sterujące



Ostatnią pętlą jest pętla „switch”. Dzięki niej możemy decydować, które instrukcje wykonają się w danym przypadku.

Po słowie „switch” podajemy w nawiasie wartość typu „int”, „String” lub „Enum” (nie może to być wartość logiczna). Pojęcia „String” i „Enum” zostaną wytłumaczone w następnych lekcjach.

Następnie możemy zdefiniować dowolną ilość przypadków. Po słowie „case” podajemy wartość dla której dane instrukcje mają się wykonać. Nie otwieramy nawiasu klamrowego! Pamiętajmy, aby wstawić „break”, jeżeli nie chcemy, aby wykonały się instrukcje dla następnego przypadku. Po drugim „case” nie ma tego słowa, więc wykonały się również instrukcje dla przypadku „default”.


„default” jest przypadkiem szczególnym. Jeżeli wartość *a* nie występuje w żadnym z „case”, wówczas wykonają się instrukcje z tej części.

Slajd  
18

**Ćwiczenie:**

Korzystając z dwóch pętli „for” (jedna wewnątrz drugiej), wypisz w konsoli całe tablice. Do wyrównania tabeli możesz skorzystać z różnic w wykonaniu poleceń „printf”. Aby uzyskać łańcuch znaków efektu tabulacji użyj wyrażenia „\t”.

 Typy podstawowe i instrukcje sterujące

Slajd  
19






**Ćwiczenie:**

Korzystając z typu „float” i „double” dodaj 10 razy 0,1 do wartości 0. Sprawdź, czy wynik jest równy 1.0.

 Typy podstawowe i instrukcje sterujące

Slajd  
20


**Wnioski:**

Typy „float” i „double” zawsze obciążone są błędem obliczeniowym!

Nie stosuj porównań typu:

```
if (liczba == 3.14) {...}
```

Aby robić to właściwie należy najpierw dobrze poznać arytmetykę zmiennopozycyjną.

 Typy podstawowe i instrukcje sterujące

### 8.2.3 Opis założonych osiągnięć ucznia

Po tej lekcji uczniowie będą wiedzieli jak poprawnie przechowywać dane w zmiennych, oraz jak programować pętle do wykonywania odpowiednich operacji.

## 8.3 Lekcja 3 - Wstęp do programowania obiektowego

### 8.3.1 Cel lekcji

Celem lekcji jest wytłumaczenie na czym polega idea programowania obiektowego. Opis budowy klasy w Javie, konstruktorów i metod.

### 8.3.2 Treść - slajdy z opisem

Slajd 1



Slajd 2



Niemal każdy element świata rzeczywistego możemy potraktować jako obiekt, niezależnie od tego, czy jest niepodzielnym elementem (np. kamień), czy składa się z wielu innych części (np. długopis).

Jest to naturalny sposób postrzegania świata.





Slajd 3



O każdy obiekt ma typy właściwości:

Informacje:

- kolor długopisu,
- cena,
- prędkość maksymalna samochodu,
- masa;

Operacje:

- pisanie,
- zdolność do poruszania się,
- wydawanie dźwięków.

Zauważmy, że ciężko jest wyobrazić sobie temperaturę jako obiekt. Nie utożsamiamy obiektu z przedmiotem! Obiektem może być np. stan techniczny części samochodowej.

Myślę, że dobrym przykładem na to jest temperatura:

- ma swoją wartość,
- może maleć i wzrastać,
- nie może być niższa od zera absolutnego.

Jeżeli chcielibyśmy stworzyć model pomieszczenia, w którym się znajdujemy, obiektem powinno być przede wszystkim samo pomieszczenie, przedmioty znajdujące się w środku, jak i temperatura powietrza.

Byłby to model obiektowy. W jasny i czytelny sposób pozwoliłby zachować niezbędne informacje na temat stanu wszystkich przedmiotów, oraz przewidzieć możliwe zdarzenia. Jeżeli jest tablica i kreda, to możliwe, że któraś z osób coś na niej napisze. Jeżeli nie znalazłby się obiekt z operacją „pisz” (kreda lub długopis), to z obiektowego punktu widzenia niemożliwe jest, aby w tym pomieszczeniu zostało coś napisane.





Slajd 4



Na obiektach możemy dokonywać generalizacji i specjalizacji. Są to kluczowe zjawiska, jeżeli chcemy skorzystać z mechanizmu dziedziczenia. Pojęcie dziedziczenia zostanie wyjaśnione na poprzedniej lekcji.

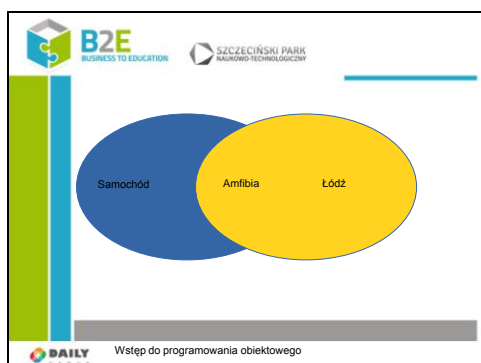
Samochód jest pojazdem spalinowym, jeżeli powiemy, że jest tylko pojazdem, wtedy następuje generalizacja.

Jeżeli powiemy, że samochód jest pojazdem, a ponadto jest pojazdem spalinowym, to następuje specjalizacja.

Zjawisko to jest bardzo przydatne. Wyobraźmy sobie, że piszemy grę, w której występuje ruch uliczny. Na skrzyżowaniu, na czerwonym świetle stoi grupa pojazdów (samochody, rowery, motocykle). Kiedy zapali się zielone światło, wywołamy na każdym z pojazdów metodę „jedź”. Nie interesuje nas wtedy jakiego typu jest poszczególny pojazd. Wszystkie one potrafią jeździć, zatrzymywać się i skręcać.

W przypadku, kiedy ta sama grupa pojazdów dojedzie do wjazdu na autostradę, musimy zatrzymać wszystkie pojazdy, które nie są spalinowe. Jazda rowerem po autostradzie jest zakazana.

Slajd 5



Pojedynczy obiekt może posiadać cechy, które są zbiorem cech, należących do dwóch różnych pojazdów.

Np. amfibia posiada cechy zarówno łodzi, jak i samochodu.



Slajd 6

Typy widoczności:

- public;
- package.

```
public class MojZnak {  
}
```

Wstęp do programowania obiektowego

Klasę w Javie zapisujemy podając typ widoczności, słowo kluczowe „class”, nazwę klasy i nawiasy klamrowe, w których znajduje się ciało klasy.

Omówimy dwa typy widoczności: „public” i „package”.

„public” oznacza, że nasza klasa będzie widoczna z każdego miejsca w kodzie.

Klasy w naszej aplikacji możemy trzymać w pakietach. Mechanizm działa tak, jak drzewo folderów w systemie operacyjnym. W Javie nie istnieje słowo kluczowe „package”. Jeżeli chcemy skorzystać z tego typu widoczności, to nie wpisujemy żadnego typu widoczności w definicji klasy. Wówczas nasza klasa będzie widoczna dla wszystkich klas znajdujących się w danym pakiecie.

Slajd 7

```
public class MojZnak {  
    private char znak;  
    public int liczba;  
}
```

```
public class MojaPierwszaKlasa {  
    public static void main(String[] args) {  
        MojZnak mojZnak = new MojZnak();  
        mojZnak.liczba = 2;  
    }  
}
```

Wstęp do programowania obiektowego

Jeżeli chcemy przechowywać w klasie pewne informacje, tworzymy pola. W Javie nie można przechowywać danych, które nie należą do żadnego obiektu!

Pola zawierają dodatkowo typ widoczności „private”. Oznacza to, że dane zmienna jest widoczna tylko w ciele danej klasy. Pola publiczne są widoczne w każdej klasie w aplikacji.

Na slajdzie w metodzie „main” po raz pierwszy świadomie tworzymy obiekt! Obiekt jest reprezentacją danej klasy (instancją) „powołaną do życia”. Przeanalizujmy jak odbywa się ten proces.

W pierwszej kolejności musimy stworzyć referencję. Referencja jest adresem na komórkę w pamięci, gdzie znajduje się dana klasa. Referencja posiada również typ obiektu na który wskazuje.

Piszemy:

TypReferencjinazwaReferencji;

W naszym przypadku jest to referencja o nazwie „mojZnak”, typu „MojZnak”.



Korzystając ze słowa kluczowego „new” alokujemy obszar pamięci operacyjnej komputera na przechowanie obiektu danego typu. Następnie znajduje się konstruktor danej klasy.

Należy pamiętać, że nazwy klas piszemy dużą literą, a nazwy pól z małej.0

Do pól danej klasy odwołujemy się pisząc referencję, wstawiając symbol kropki, następnie wstawiając nazwę pola (tak jak w przykładzie).

Zauważmy, że do pola „znak” nie możemy się odwołać (jest prywatne).

Slajd 8

 **B2E**  
BUSINESS TO EDUCATION  SZCZECIŃSKI PARK  
NAUKOWO-TECHNOLOGICZNY

```
public class MojZnak {  
    private char znak;  
    public int liczba;  
}  
  
public class MojaPierwszaKlasa {  
    public static void main(String[] args) {  
        MojZnak mojZnak = new MojZnak();  
        System.out.println(mojZnak.liczba);  
    }  
}
```





Wstęp do programowania obiektowego

W tym przykładzie chcemy wyświetlić pole „liczba” nie inicjalizując go. W tym przypadku otrzymamy 0, ale na ogół takie przypadki mogą być bardzo niebezpieczne i zagrażać stabilności naszego programu

Jeżeli tworzymy zaawansowane obiekty, musimy mieć kontrolę nad wartościami domyślnymi ich pól.

Slajd 9

 **B2E**  
BUSINESS TO EDUCATION  SZCZECIŃSKI PARK  
NAUKOWO-TECHNOLOGICZNY

```
public class MojZnak {  
    private char znak;  
    public int liczba;  
    public MojZnak() {  
        liczba = 7;  
    }  
}  
  
public class MojaPierwszaKlasa {  
    public static void main(String[] args) {  
        MojZnak mojZnak = new MojZnak();  
        System.out.println(mojZnak.liczba);  
    }  
}
```



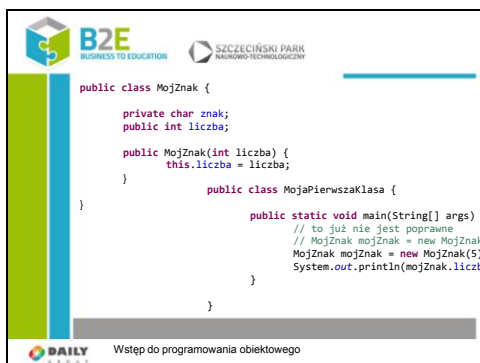
Wstęp do programowania obiektowego

Konstruktor jest metodą specjalną. Jeżeli nie zdefiniujemy własnego, to kompilator dołącza domyślny konstruktor bezargumentowy. Dlatego w poprzednich przykładach mogliśmy pisać „newMojZnak()”.

Tworzenie własnych konstruktorów rozwiązuje problem inicjalizowania pól.

W tym przypadku, po stworzeniu klasy pole „liczba” ma wartość 7.

## Slajd 10



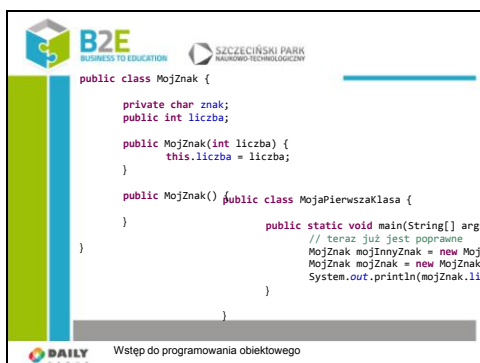
Na slajdzie widoczny jest przykład konstruktora parametrycznego. Tworząc obiekt, użytkownik musi podać wartość początkową dla pola „liczba”. Kompilator nie dołączył już konstruktora domyślnego. Jest to jedyny sposób w jaki możemy stworzyć obiekt.

Słowo kluczowe „this” jest referencją na obiekt, w którym aktualnie się znajdujemy. Polecenie „this.liczba = liczba” może wyglądać dziwnie, ale jest całkowicie bezpieczne. Powoduje przypisanie wartości argumentu do pola.

W środowisku Eclipse konstruktory możemy zbudować automatycznie korzystając z kreatora dostępnego w:

„Source/Generate Konstruktor using Fields...”

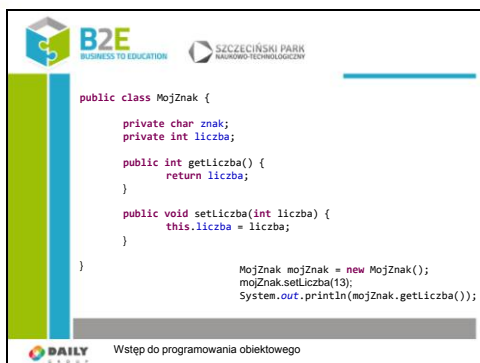
## Slajd 11



Możemy stworzyć dowolną liczbę konstruktorów. Muszą jednak różnić się listą argumentów, które przyjmują.



Slajd 12



Pola nie powinny mieć widoczności publicznej. Jest to zła praktyka programistyczna. Jak więc na nich operować z poziomu innych klas?

Do tego używa się metod. Metoda posiada typ widoczności, typ który zwracają, nazwę pisaną małą literą (jedynie metody, których nazwy piszemy dużą literą to konstruktory), oraz listę argumentów.

Szczególnym typem zwracany przez metodę jest „void”. Oznacza to, że metoda wykonuje jedynie pewne operacje, nie zwracając rezultatu. Tak działa metoda „setLiczba”.

W innych przypadkach, jeżeli zadeklarujemy zwracanie typu „int”, musimy w metodzie użyć słowa „return” wpisując za nim tę wartość. Tak działa metoda „getLiczba”.

Uwaga, słowo „return” kończy wykonywanie się metody. Instrukcje, które podamy po tym słowie, nie wykonają się.


Metody zaczynające się od fraz „set” i „get” są charakterystyczne. Służą właśnie do obsługi pól.

Nie ma potrzeby ich ręcznego pisania. Można skorzystać z kreatora w Eclipse:

„Source/Generate Getters and Setters...”


Metody „set” i „get” pełnią bardzo ważną rolę. Dają nam kontrolę nad wartościami, które użytkownik będzie chciał przypisać do pól. Np. jeżeli dane pole będzie przechowywało wartość temperatury, nie może mieć wartości mniejszej niż 273. Przy próbie zapisania wartości -400 metoda „set” powinna wpisać -273.

## Slajd 13



```
public class MojZnak {  
  
    public static String getNapis() {  
        return "Napis";  
    }  
  
    private static String getInnyNapis() {  
        return "InnyNapis";  
    }  
}
```

```
System.out.println(MojZnak.getNapis());  
// to jest nieprawidłowe  
// System.out.println(MojZnak.getInnyNapis());
```

 Wstęp do programowania obiektowego



Metody, tak samo jak pola, mogą być statyczne. Oznacza to, że są takie same dla wszystkich instancji danej klasy. Nie ma też potrzeby tworzenia klasy, jeżeli chcemy skorzystać z jej elementu statycznego.

W przykładzie mamy podane prawidłowe wywołanie metody statycznej. Zauważmy, że nie mamy żadnej referencji. Metody statyczne wywoływane są bezpośrednio dla nazwy klasy.

Dlatego właśnie aplikacje w Javie uruchamia się za pomocą metody „main”. Metoda ta jest dostępna zanim zostanie stworzona instancja danego obiektu.


Oczywiście nigdy nie wywołamy metody prywatnej z poziomu innego obiektu, tak jak pokazane to jest w komentowanym kodzie.

## Slajd 14



```
public class MojZnak {  
  
    private static int liczba;  
    private static char znak;  
  
    public static void setValues(int mojaLiczba) {  
        liczba = mojaLiczba;  
    }  
  
    public static void setValues(int mojaLiczba, char mojZnak) {  
        liczba = mojaLiczba;  
        znak = mojZnak;  
    }  
}
```

```
MojZnak.setValues(10);  
MojZnak.setValues(13, 'X');
```

 Wstęp do programowania obiektowego

W tym przypadku nie ma sensu tworzenie instancji danej klasy, wszystko jest statyczne. Nie utworzymy również dwóch różnych obiektów typu „MojZnak”. Możemy zapisać:

```
MojZnak z1 = newMojZnak();
```

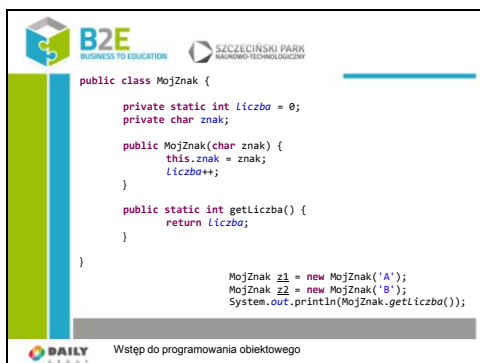
```
MojZnak z2 = newMojZnak();
```

W praktyce jest to bez sensu. I tak nie przechowamy w obiektach różnych wartości.

Oczywiście tak, jak w przypadku konstruktorów, może być kilka metod o tej samej nazwie, różniących się listą argumentów.



Slajd 15

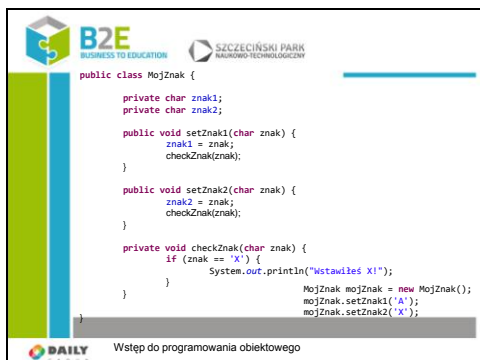


Jakie jest więc może być praktyczne zastosowanie pól statycznych? Możemy stworzyć klasę „StaleMatematyczne” i stworzyć publiczne, statyczne pole o nazwie pi i wartości „3,14”. We wszystkich klasach programu będziemy mogli korzystać z tej wartości wpisując „StaleMatematyczne.pi”.

Nieco ciekawszy jest przykład podany na slajdzie. Pole „liczba” jest statyczne, co oznacza, że jest wspólne dla wszystkich instancji danej klasy.

W każdym wywołaniu konstruktora następuje inkrementacja jego wartości. W prosty sposób możemy sprawdzić ile obiektów typu „MojZnak” zostało stworzonych.

Slajd 16



Z prywatnych metod korzystamy tylko w obrębie danej klasy. Można stosować je aby nie pisać dwa razy tego samego kodu.

W przykładzie, w każdej metodzie „set” następuje sprawdzenie, czy ustawiana wartość jest równa „X”.





Slajd 17





**Cwiczenie:**

Stwórz obiekt zawierający dwie liczby: a i b (dowolnego typu). Nie pozwól a  
Udostępnić użytkownikowi dwie metody: suma i iloczyn. Mają zwracać sumę  
Klasę nazwij „DwieLiczby”.



Wstęp do programowania obiektowego


Slajd 18



**Cwiczenie:**

Stwórz obiekt (o dowolnej nazwie) z dwoma metodami statycznymi. Metody  
Niech wynikiem metody suma będzie suma wartości, które zostały zwrócone  
Metoda iloczyn ma działać analogicznie.

Pamiętaj, że argumentem metody może być nie tylko typ wbudowany. Pon  
void metoda(Obiekt a) {...}



Wstęp do programowania obiektowego

### 8.3.3 Opis założonych osiągnięć ucznia

Po tej lekcji uczniowie będą umieli tworzyć proste obiekty w Javie, odpowiednio inicjalizować pola klas w trakcie ich tworzenia.

## 8.4 Lekcja 4 – Interfejsy, klasy abstrakcyjne i dziedziczenie

### 8.4.1 Cel lekcji

Celem lekcji jest wyjaśnienie pojęcia interfejsu i klasy abstrakcyjnej. Wy tłumaczone będzie jak zwiększyć efektywność i bezpieczeństwo pracy poprzez stosowanie dziedziczenia.

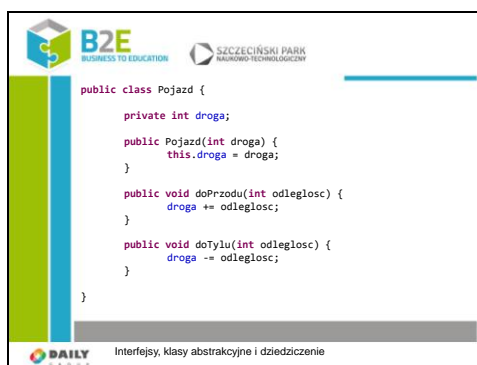


## 8.4.2 Treść - slajdy z opisem

### Slajd 1



### Slajd 2



Klasa pokazana na przykładzie implementuj w prosty sposób niektóre właściwości pojazdu.

Nowo stworzony pojazd ma licznik odległości ustawiony na zadaną wartość. Możemy poruszać się do przodu i do tyłu.

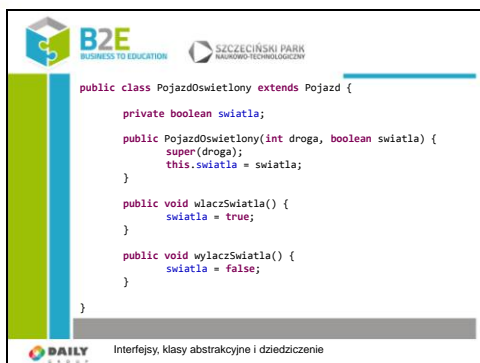
Może to być rower, motocykl, lub samochód.

Zastanówmy się co możemy zrobić, jeżeli chcemy stworzyć klasę „PojazdOswietlony”, który posiada te same właściwości co „Pojazd”. Ponadto chcemy, aby „PojazdOswietlony”, miał metody „włączSwiatla” i „wylączSwiatla”.

Pisanie całości nowej klasy, przepisując część funkcjonalności z pojazdu jest niepotrzebne. W Javie (w innych językach również) nie powinno się pisać dwa razy tego fragmentu kodu. Możemy korzystać z mechanizmu dziedziczenia.



Slajd 3

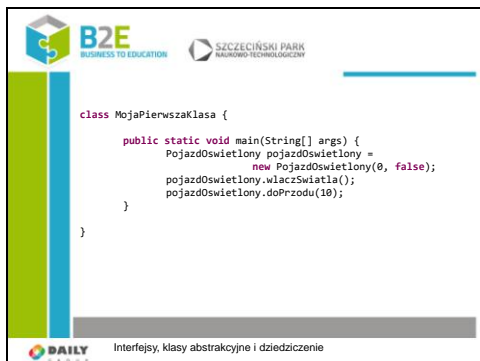


Możemy stworzyć nową klasę która posiada wszystkie właściwości klasy „Pojazd”.

Po nazwie klasy, używamy słowa „extends” i podajemy nazwę klasy, po której będziemy dziedziczyć.

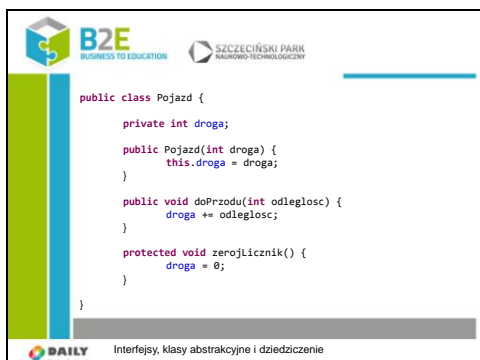
Klasa „Pojazd” posiadała tylko jeden konstruktor, z parametrem. Aby stworzyć obiekt „PojazdOswietlony” musimy najpierw wywołać konstruktor z klasy, którą dziedziczymy. Odpowiada za to instrukcja „super”. Jest to w pewnym sensie konstruktor klasy, z której dziedziczymy.

Slajd 4



Jak widać na slajdzie, na instancji obiektu „PojazdOswietlony” możemy wykonywać również metody z klasy „Pojazd”. W ten sposób uniknęliśmy ponownej implementacji niektórych funkcjonalności.



Slajd 5



Typ widoczności „protected” oznacza to samo co domyślny typ „package”. Metody i pola oznaczone w ten sposób widoczne są dla wszystkich klas znajdujących się w pakiecie.

Ponadto pola i metody mają jeszcze jedną właściwość. Są widoczne dla wszystkich klas, które dziedziczą z danej klasy.

## Slajd 6




```
public class PojazdOswietlony extends Pojazd {
    private boolean swiatla;

    public PojazdOswietlony(int droga, boolean swiatla) {
        super(droga);
        this.swiatla = swiatla;
    }

    public void wlaczSwiatla() {
        swiatla = true;
    }

    public void zeruj() {
        zerujLicznik();
    }
}
```



 Interfejsy, klasy abstrakcyjne i dziedziczenie

Możemy skorzystać metody „zerujLicznik” z dwóch powodów.

Po pierwsze, dziedziczymy po klasie „Pojzd”.


Po drugie, klasa „PojazdOswietlony” znajduje się w tym samym pakiecie.

## Slajd 7



```
class MojaPierwszaKlasa {
    public static void main(String[] args) {
        Pojazd pojazd = new Pojazd(0);
        pojazd.zerujLicznik();



        PojazdOswietlony pojazdOswietlony =
            new PojazdOswietlony(0, false);
        pojazdOswietlony.zerujLicznik();
        pojazdOswietlony.zeruj();
    }
}
```

 Interfejsy, klasy abstrakcyjne i dziedziczenie

Możemy skorzystać metod „zerujLicznik” tylko dlatego, że jesteśmy w tym samym pakiecie.


W innych pakietach dostępna będzie tylko metoda „zeruj”.

## Slajd 8



```
package pakiet1;



public class ObiektA {
    protected void pokazWiadomosc() {
        System.out.println("Wiadomość!");
    }
}
```

 Interfejsy, klasy abstrakcyjne i dziedziczenie

Aby lepiej pokazać działanie typu widoczności stworzone zostały dwa pakiety.

Klasa „ObiektA” znajduje się w osobnym pakiecie i zawiera metodę o właściwości „protected”.


## Slajd 9



```
package pakiet2;
import pakiet1.ObjektA;

public class ObjektB extends ObjektA {

    public void pokazOdziedziczonaWiadomosc() {
        pokazWiadomosc();
    }
}
```



 Interfejsy, klasy abstrakcyjne i dziedziczenie

Pierwszy wers oznacza, że bieżąca klasa znajduje się w pakiecie „pakiet2”.

„ObjektA” nie jest widoczny. Musi zostać zaimportowany z innego pakietu.

„ObjektB” dziedziczy z „ObjektA”, więc ma dostęp do jego metody chronionej („protected”).

## Slajd 10




```
package pakiet2;
import pakiet1.ObjektA;

public class Main {

    public static void main(String[] args) {
        ObjektA obiektA = new ObjektA();
        // nie ma odpowiedniej metody dla obiektA

        ObjektB obiektB = new ObjektB();
        obiektB.pokazOdziedziczonaWiadomosc();
    }
}
```



 Interfejsy, klasy abstrakcyjne i dziedziczenie

Znajdujemy się cały czas w pakiecie „pakiet2”.

Nie mamy dostępu do metody chronionej obiektu „ObjektA”.

Możemy ją jednak wywołać korzystając z „ObjektB”.

## Slajd 11



```
import java.awt.Dimension;


class MojaPierwszaKlasa {

    public static void main(String[] args) {
        Dimension dimension = new Dimension(2, 3);
        System.out.println(dimension.toString());

        System.out.println(new MojaPierwszaKlasa());
    }

    @Override
    public String toString() {
        return super.toString() + " - dziedziczę!";
    }
}
```

Wynik: java.awt.Dimension[width=2,height=3]  
MojaPierwszaKlasa@1e859c0 - dziedziczę!

 Interfejsy, klasy abstrakcyjne i dziedziczenie

Każda klasa w Javie dziedziczy po obiekcie „Object”, zawsze! „Object” posiada kilka metod i są one dostępne dla każdego stworzonego obiektu w Javie. Jedną z nich jest „toString”. Reprezentuje ona obiekt w postaci tekstu.



W metodzie „main” stworzony został obiekt typu „Dimension”. Obiekt ten przechowuje dwa podstawowe wymiary: wysokość i szerokość. Kiedy wywołamy na nim metodę „toString” otrzymamy tekstową reprezentację obiektu.

Klasa „MojPierwszaKlasa” zawiera już metodę „toString”. Chcę jednak aby zwracała wartość ustawioną przeze mnie. Wyrażenie „@Override” to adnotacja. Oznacza, że obiekt odziedziczył już taką metodę, ale chcę napisać ją we własny

sposób. Używając słowa „super” wywołuję metodę „toString” dla obiektu, po którym dziedziczyłem.

Metoda „println” domyślnie wywołuje metodę „toString” dla każdego obiektu, dlatego ostatnia instrukcja metody „main” działa w ten sposób.

Slajd 12


```
Dimension dim1 = new Dimension(2, 3);
Dimension dim2 = new Dimension(2, 3);

if (dim1 == dim2) {
    System.out.println("Tak dla 1");
}

if (dim1.equals(dim2)) {
    System.out.println("Tak dla 2");
}
```

Wynik:

Tak dla 2

 Interfejsy, klasy abstrakcyjne i dziedziczenie

Stworzone zostały dwa z pozoru takie same obiekty. Zastanówmy się jednak. Obiekty mają takie same dane, ale zostały utworzone w niezależny sposób.



W pierwszej instrukcji „if” nie sprawdzamy równości obiektów! Sprawdzamy równość referencji. Warunek byłby prawdziwy, jeżeli obie referencje wskazywałyby na ten sam obiekt, a tak nie jest.

Dla sprawdzenia równości obiektów wywołujemy na jednym z nich metodę „equals”. Metoda ta również jest dziedziczona po typie „Object”.

Jeżeli chcemy porównać dwie instancje stworzonej przez nas klasy, to pamiętajmy aby napisać dla niej własną wersję metody „equals”.




Slajd 13



Ćwiczenie:

Stwórz klasę „Animal” z metodami „eat” i „toString”. Niech „toString” zwraca Wywołaj „println” podając jako argument każdy z obiektów.

 Interfejsy, klasy abstrakcyjne i dziedziczenie

Slajd 14





W Javie możemy dziedziczyć tylko po jednej klasie!


Mam metody dla motorówki i dla samochodu. Jak stworzyć obiekt amfibia?

 Interfejsy, klasy abstrakcyjne i dziedziczenie

Slajd 15



```
public interface Boat {  
    public void floatOnWater();  
}  
  
public interface Car {  
    public void drive();  
}
```

 Interfejsy, klasy abstrakcyjne i dziedziczenie

W tym celu stosuje się interfejsy. Interfejs zawiera jedynie metody bez ciał. Tak jak widzimy na slajdzie. Dodatkowo możemy wstawić listę argumentów.

Interfejs przedstawia listę operacji, które może wykonać dany obiekt.



## Slajd 16

```
public class Amphibian implements Boat, Car {  
    @Override  
    public void drive() {System.out.println("Jadę!");}  
    @Override  
    public void floatOnWater() {System.out.println("Płynę!");}  
}  
  
class MojaPierwszaKlasa {  
    public static void main(String[] args) {  
        Amphibian amphibian = new Amphibian();  
        amphibian.drive();  
        amphibian.floatOnWater();  
    }  
}
```

 Interfejsy, klasy abstrakcyjne i dziedziczenie

Interfejsów się nie dziedziczy. Interfejsy implementujemy, tak jak jest to pokazane w pierwszym wersie.

Dopiero teraz możemy nadać ciałom metody.

## Slajd 17

```
class MojaPierwszaKlasa {  
    public static void main(String[] args) {  
        Amphibian amphibian = new Amphibian();  
        printCar(amphibian);  
    }  
    private static void printCar(Car car) {  
        System.out.println(car);  
    }  
}
```

 Interfejsy, klasy abstrakcyjne i dziedziczenie

Bieżący przykład pokazuje jaki jest sens nadawania interfejsów obiektom.

Metoda „printCar” przyjmuje jako argument obiekt typu „Car”. Obiekt „Amphibian” implementuje ten interfejs, dlatego wywołanie tej metody jest poprawne.

## Slajd 18

```
public abstract class Ptak {  
    private int waga;  
    public int getWaga() {  
        return waga;  
    }  
    public void setWaga(int waga) {  
        this.waga = waga;  
    }  
}
```

 Interfejsy, klasy abstrakcyjne i dziedziczenie

Aby stworzyć klasę abstrakcyjną należy użyć słowa „abstract” definiując klasę.

Przy jej implementacji obowiązują takie same zasady jak przy implementacji zwykłej klasy, z tą różnicą, że słowo „static” jest niedozwolone.

Slajd 19

```
public class Bocian extends Ptak {  
    }  
  
// niepoprawne wywołanie  
// Ptak ptak = new Ptak();  
Ptak ptak = new Bocian();  
Bocian bocian = new Bocian();  
Object object = new Bocian();  
ptak.setWaga(5);  
bocian.setWaga(5);  
// to już nie jest prawidłowe  
// object.setWaga(5);
```

Interfejsy, klasy abstrakcyjne i dziedziczenie

Dziedziczenie po klasie abstrakcyjnej odbywa się tak samo jak po zwykłej klasie. Nie można jednak stworzyć jej instancji, tak samo jak nie można stworzyć instancji interfejsu.

Na obiekt typu „Bocian” może wskazywać referencja typu: „Object”, „Bocian” lub „Ptak”. Należy pamiętać, że na referencji typu „Object” nie wywołamy metody „setWaga”.

Slajd 20

**Cwiczenie:**

Stwórz interfejs „WiFi” z metodą „connect”.  
Stwórz interfejs „Bluetooth” z metodą „connect”.  
Sprawdź czy możesz stworzyć interfejs „Wireless” z taką metodą. Może nie.

Stwórz klasę abstrakcyjną „Phone”, niech posiada pole z numerem telefonu.

Stwórz klasę „MobilePhone”, która dziedziczy po tych obiektach.

Interfejsy, klasy abstrakcyjne i dziedziczenie

### 8.4.3 Opis złożonych osiągnięć ucznia

Uczeń będzie umiał wykorzystać w praktyce interfejs i klasę abstrakcyjną, tworzyć nowe klasy wykorzystując wcześniej zaimplementowany kod.

## 8.5 Lekcja 5 - Polimorfizm, instrukcja „final”, klasy wewnętrzne i anonimowe

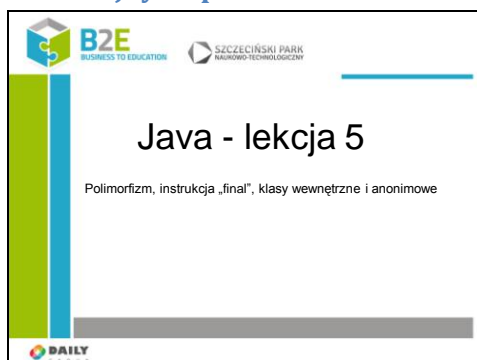
### 8.5.1 Cel lekcji

Celem lekcji jest wytłumaczenie pojęcia polimorfizmu, oraz jego praktycznego zastosowania w Javie. Pokazanie jak tworzyć klasy wewnętrzne, oraz efektywnie posługiwać się interfejsami.



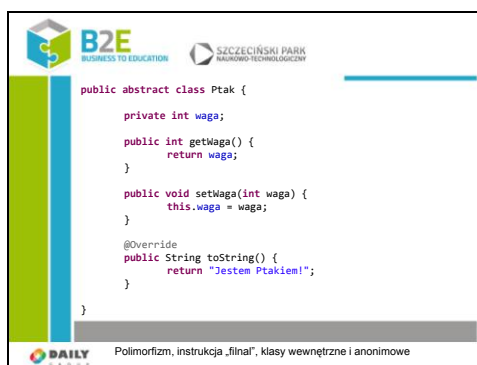
## 8.5.2 Treść - slajdy z opisem

### Slajd 1



Slide 1 content: Title "Java - lekcja 5", subtitle "Polimorfizm, instrukcja „final”, klasy wewnętrzne i anonimowe". Logos: B2E, Szczeciński Park Naukowo-Technologiczny, DAILY.

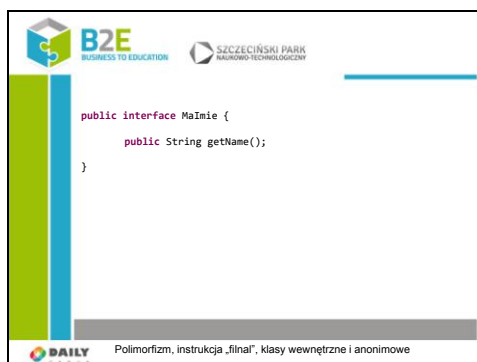
### Slajd 2



Slide 2 content: Java code for an abstract class `Ptak` with methods `getWaga()`, `setWaga()`, and `toString()`. Logos: B2E, Szczeciński Park Naukowo-Technologiczny, DAILY.

Mamy prostą klasę "Ptak" z własną implementacją metody "toString".



### Slajd 3




Slide 3 content: Java code for an interface `MaImie` with method `getName()`. Logos: B2E, Szczeciński Park Naukowo-Technologiczny, DAILY.

Istnieje również interfejs "MaImie" z metodą "getName".

## Slajd 4



```
public class Bocian extends Ptak implements MaImie {  
    private String imie = "Kajtek";  
  
    @Override  
    public String getName() {  
        return imie;  
    }  
  
    @Override  
    public String toString() {  
        return "Jestem bocianem o imieniu " + imie;  
    }  
}
```



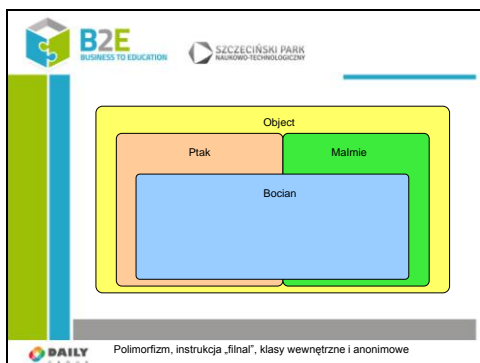
Polimorfizm, instrukcja „final”, klasy wewnętrzne i anonimowe

Stworzona jest klasa "Bocian", która rozszerza klasę "Ptak" i implementuje interfejs "Malmie".



Oczywiście w przypadku implementacji interfejsu, musimy zaimplementować wszystkie metody, które do tego interfejsu należą.

Bieżąca klasa ma własną wersję metody "toString".


## Slajd 5



## Slajd 6



```
class MojaPierwszaKlasa {  
    public static void main(String[] args) {  
        Bocian bocian = new Bocian();  
        Ptak ptak = (Ptak) bocian;  
        MaImie maImie = (MaImie) bocian;  
        Object object = (Object) bocian;  
  
        System.out.println(bocian);  
        System.out.println(ptak);  
        System.out.println(maImie);  
        System.out.println(object);  
  
        if (bocian == ptak) {  
            System.out.println("Zachodzi równość!");  
        }  
    }  
}
```



Polimorfizm, instrukcja „final”, klasy wewnętrzne i anonimowe

W wyniku działania programu dostaniemy cztery razy napis "Jestem bocianem o imieniu Kajtek". I napis "Zachodzi równość".

Jak widać możemy przypisać obiekt "bocian" do Referencji czterech różnych typów: "Bocian", "Ptak", "Malmie", "Object". Zjawisko to nazywamy plimorfizmem (inaczej wielopostaciowość).

Obiekt możemy traktować na różne sposoby w zależności od potrzeb.



Zastanówmy się dlaczego dla referencji typu "Ptak" nie pokazał się napis "Jestem ptakiem!"?

Odpowiedź może zasugerować nam porównanie. Pamiętajmy, że nie sprawdzamy równości obiektów, na które


wskazują referencje! Sprawdzamy, czy wskazują na ten sam obiekt. Wszystkie referencje w w metodzie "main" wskazują na obiekt stworzony przez konstruktor "Bocian()".

To, że metoda "toString" z klasy "Ptak" zwraca napis "Jestem ptakiem!", nie ma żadnego znaczenia. W klasie "Bocian" mamy zupełnie inną implementację tej metody. Klasa nie może posiadać dwóch metod o takiej samej nazwie i takiej samej liście argumentów.

Slajd 7

```
class MojaPierwszaKlasa {  
    public static void main(String[] args) {  
        Bocian bocian = new Bocian();  
  
        if (bocian instanceof Ptak) {  
            System.out.println("Bocian jest ptakiem!");  
        }  
    }  
}
```

 Polimorfizm, instrukcja „final”, klasy wewnętrzne i anonimowe

Aby móc rozpoznawać typ obiektu możemy skorzystać z operatora "instanceof". W powyższym przykładzie widzimy

Slajd 8

**Ćwiczenie:**

Stwórz dwie klasy:

Liczba, niech przechowuje dowolną liczbę i posiada metodę „podajWartosc”.

Napis, niech przechowuje dowolny tekst i posiada metodę „podajTekst”.

W dowolnym obiekcie, który posiada metodę „main” stwórz metodę „wyswietl”, która

 Polimorfizm, instrukcja „final”, klasy wewnętrzne i anonimowe



Slajd 9




Ćwiczenie:

Stwórz klasę „Temperatura” z metodą „podajTemperature”. Wykorzystaj klasę

Polimorfizm, instrukcja „final”, klasy wewnętrzne i anonimowe

Slajd 10



Ćwiczenie:

Stwórz taką klasę:

```
public class BocianBialy extends Bocian {  
      
}
```

Następnie zmodyfikuj klasę „Bocian” dodając słowo „final” jak w przykładzie

```
public final class Bocian extends Ptak implements MaImie {  
      
}
```

Jaki problem wystąpił w aplikacji?

Polimorfizm, instrukcja „final”, klasy wewnętrzne i anonimowe

Słowo "final" użyte w definicji klasy oznacza, że nie możemy po niej dziedziczyć.

Slajd 11



Ćwiczenie:



Dodaj „final” do metody „toString” w klasie „Bocian”. Czy „BocianBialy” może

Polimorfizm, instrukcja „final”, klasy wewnętrzne i anonimowe


Pisanie własnych wersji metod nazywamy przesłanianiem. Nie można przesłonić metody oznaczonej jako final. Jest to ostateczna implementacja.



Slajd 12



```
public class Drzewo {  
    private String gatunek = "Klon";  
    public class Lisc {  
        @Override  
        public String toString() {  
            return "Jestem liściem drzewa " + gatunek;  
        }  
    }  
    public Lisc getLisc() {  
        return new Lisc();  
    }  
}
```





Polimorfizm, instrukcja „final”, klasy wewnętrzne i anonimowe

Java umożliwia tworzenie klas wewnętrznych. Klasa wewnętrzna ma dostęp do elementów prywatnych klasy, w której się znajduje.


Klasa wewnętrzna może posiadać typy widoczności: prywatny, chroniony, publiczny i domyślny (pakietowy).

Slajd 13



```
class MojaPierwszaKlasa {  
    public static void main(String[] args) {  
        Drzewo drzewo = new Drzewo();  
        Drzewo.Lisc lisc = drzewo.new Lisc();  
        System.out.println(lisc);  
    }  
}
```

Wynik:  
Jestem liściem drzewa Klon





Polimorfizm, instrukcja „final”, klasy wewnętrzne i anonimowe

Tak wygląda prawidłowe wywołanie klasy wewnętrznej. Zwróćmy uwagę jak wygląda użycie operatora "new" dla obiektu "Lisc".


Dlaczego nie można stworzyć obiektu typu "Lisc" bez użycia obiektu "Drzewo"?

Obiekt typu "Lisc" może mieć dostęp do pól obiektu typu "Drzewo". Dlatego aby stworzyć "lisc" musimy stworzyć najpierw obiekt "drzewo".

Slajd 14



```
public class Miasto {  
    public class Ulica {  
        public class Dom {  
            @Override  
            public String toString() {  
                return Ulica.this.toString() + " Dom";  
            }  
        }  
        @Override  
        public String toString() {  
            return Miasto.this.toString() + " Ulica";  
        }  
    }  
    @Override  
    public String toString() {  
        return "Miasto";  
    }  
}
```



Polimorfizm, instrukcja „final”, klasy wewnętrzne i anonimowe

Widoczny przypadek jest nieco bardziej skomplikowany. Jak widać każda z klas może mieć własną metodę "toString".

Zwróćmy uwagę jak następuje wywołanie metody z klas zewnętrznych,





Slajd 15



```
Miasto miasto = new Miasto();
Miasto.Ulica ulica = miasto.new Ulica();
Miasto.Ulica.Dom dom = ulica.new Dom();
System.out.println(dom);
```

Wynik:



Miasto Ulica Dom

 Polimorfizm, instrukcja „final”, klasy wewnętrzne i anonimowe

Obiekty tworzymy dokładnie tak jak w poprzednim przykładzie.

Aby zbudować wynik wywołane zostały metody z wszystkich trzech obiektów.

Slajd 16




```
public abstract class Ptak {
    private int waga;

    public int getWaga() {
        return waga;
    }

    public void setWaga(int waga) {
        this.waga = waga;
    }



    @Override
    public String toString() {
        return "Jestem Ptakiem!";
    }
}
```

 Polimorfizm, instrukcja „final”, klasy wewnętrzne i anonimowe

Skorzystajmy z klasy abstrakcyjnej i interfejsu, których używaliśmy wcześniej.


W poprzedniej lekcji podane było, że nie możemy stworzyć instancji danej klasy abstrakcyjnej lub interfejsu. W praktyce nie możemy obejść tą zasadę tworząc klasę anonimową.

Slajd 17



```
class MojaPierwszaKlasa {
    public static void main(String[] args) {
        Ptak ptak = new Ptak() {
        };
        ptak.setWaga(10);

        MaImie maImie = new MaImie() {
            @Override
            public String getName() {
                return "Moje imie";
            }
        };
        System.out.println(maImie.getName());
    }
}
```

 Polimorfizm, instrukcja „final”, klasy wewnętrzne i anonimowe

"Ptak" jest klasą abstrakcyjną. W nawiasach klamrowych możemy dodać dodatkowe metody i pola do klasy. W ten sposób stworzony został obiekt anonimowy. Nie możemy odwołać się do klasy, której obiekt jest instancją. Klasa ta nie ma nazwy, ale wiemy, że zgodnie z polimorfizmem, jest typu "Ptak".

W przypadku interfejsu sprawa jest nieco bardziej skomplikowana. Nie ma on ciała swoich metod, musimy więc je zaimplementować.



Slajd 18



Slide 18 content: B2E logo, Szczeciński Park Naukowo-Technologiczny logo, and a code snippet for a Java exercise.

Cwiczenie:

Korzystając z klasy anonimowej stworzonej z interfejsu zaprojektuj mechanizm. Czas w postaci long (milisekundy) zwróci Ci wywołanie:

```
new java.util.Date().getTime()
```

Polimorfizm, instrukcja „final”, klasy wewnętrzne i anonimowe

### 8.5.3 Opis założonych osiągnięć ucznia

Po tej lekcji uczniowie będą wiedzieć jak wykorzystuje się w praktyce polimorfizm i interfejsy.

## 8.6 Lekcja 6 - Typ wyliczeniowy, JavaDocs

### 8.6.1 Cel lekcji

Celem lekcji jest zapoznanie uczniów z typem wyliczeniowym i prostą metodą dokumentowania kodu za pomocą JavaDocs.

### 8.6.2 Treść - slajdy z opisem

Slajd 1



Slide 1 content: B2E logo, Szczeciński Park Naukowo-Technologiczny logo, and the title 'Java - lekcja 6'.

Java - lekcja 6

Typ wyliczeniowy  
JavaDocs

Na lekcji poznamy co to jest typ wyliczeniowy, do czego znajduje zastosowanie i jak można go wykorzystać w programowaniu.

Poznamy narzędzia do dokumentowania kodu



Slajd 2

**JAVA**  
Dla początkujących

### Typ wyliczeniowy – co to jest?

- Enum type – Enumeration (typ wyliczeniowy)
- Specjalny typ danych który ma predefiniowane wartości
- Pory roku, Dni tygodnia, Dni miesiąca
- Rodzaje pojazdów
- Rodzaje filmów

Lekcja 6 i typ wyliczeniowy

Typ wyliczeniowym jest specjalny typ danych który może przyjmować jedną z zadeklarowanych stałych.

Przykładem może być pora roku. Załóżmy że chcemy aby zmienna `pora_roku` przyjmowała tylko ustalone przez nas wartości tj. wiosna, lato, jesien i zima.

Slajd 3

**JAVA**  
Dla początkujących

### Typ wyliczeniowy enum

```
public enum Pory_roku { WIOSNA, LATO,  
JESIEN, ZIMA }
```

Lekcja 8 - operacje wejścia wyjścia.

Przygotujmy nową uruchamialną klasę i zadeklarujmy następujący typ zmiennej: `Pory_roku` który będzie przyjmować następujące wartości

```
public enum Pory_roku { WIOSNA,  
LATO, JESIEN, ZIMA }
```

Slajd 4

**JAVA**  
Dla początkujących

### Typ wyliczeniowy enum

```
public enum Pory_roku { WIOSNA, LATO,  
JESIEN, ZIMA }
```

```
public static void main(String[] args) {  
    // TODO Auto-generated method stub  
    Pory_roku pory_roku = Pory_roku.JESIEN;
```

Lekcja 8 - operacje wejścia wyjścia.



Przygotujmy nową uruchamialną klasę i zadeklarujmy następujący typ zmiennej: `Pory_roku` który będzie przyjmować następujące wartości

```
public enum Pory_roku { WIOSNA,  
LATO, JESIEN, ZIMA }
```

W metodzie `main` zadeklaruj my sobie zmienną typu `Pory_roku` i przypiszmy jej wartość.

Zauważmy, że wpisując wartość możemy posługiwać się zadeklarowanym typem `Pory_roku`.

## Slajd 5

**JAVA**  
 Dla początkujących


### Typ wyliczeniowy enum

```

public enum Pory_roku { WIOSNA, LATO, ZESZEN, ZIMA }



public static void main(String[] args) {
    // TODO Auto-generated method stub
    Pory_roku pory_roku = Pory_roku.WIOSNA;
  }
  
```

✓ JESZEN : EnumTest.Pory\_roku - EnumTest.Pory\_roku  
 ✓ LATO : EnumTest.Pory\_roku - EnumTest.Pory\_roku  
 ✓ WIOSNA : EnumTest.Pory\_roku - EnumTest.Pory\_roku  
 ✓ ZIMA : EnumTest.Pory\_roku - EnumTest.Pory\_roku  
 ✗ valueOf(String arg0) : Pory\_roku - Pory\_roku  
 ✗ class : Class<EnumTest.Pory\_roku>


 Lekcja 8 - operacje wejścia wyjścia.

Zauważmy, że wpisując wartość możemy posługiwać się zadeklarowanym typem `Pory_roku`. Eclipse podpowiada nam wartości mechanizmem intellisense.


## Slajd 6

**JAVA**  
 Dla początkujących

### Typ wyliczeniowy enum - zadanie

- Proszę napisać klasę która na podstawie przekazanej konstruktorowi pory roku ustali czy lubisz porę roku.
- Metoda `Czy_Lubie()` niech wydrukuje / wyświetli informację czy zadana pora roku jest lubiana czy nie.


 Lekcja 8 - operacje wejścia wyjścia.

Proszę wykorzystać instrukcję `switch/case`. Dla uproszczenia klasę można zadeklarować jak zagnieżdżoną klasę programu zamiast nowego pliku dla klasy.



Inicjacja obiektu powinna następować wg przykładu:

```

EnumTest enumTest = new EnumTest();
Lubie_pory_roku w = enumTest.new
Lubie_pory_roku....
  
```



Slajd 7




JAVA  
Dla początkujących

### Typ wyliczeniowy enum - rozwiązanie

```
public class Lubie_pory_roku
{
    Pory_roku pory_roku;

    Lubie_pory_roku(Pory_roku p) {
        this.pory_roku = p;
    }

    public void Czy_Lubie(){
        System.out.print("Cześć czy lubię porę roku "+pory_roku.toString()+"
        switch (pory_roku) {
            case WIOSNA: System.out.println("Lubię wiosnę");break;
            case LATO: System.out.println("Bardzo lubię lato");break;
            case ZESZEN: System.out.println("Też tak sobie");break;
            default: System.out.println("był padł śnieg");break;
        }
    }
}
```



 Lekcja 8 - operacje wejścia wyjścia,

Klasa nazywa się `Lubie_pory_roku`  
Jej konstruktor zapamiętuje zadany przy tworzeniu obiektu klasy zmienną typu `Pora_roku` w polu klasy.

W metodzie `czy_lubie` zastosowano instrukcję `switch / case`  
Proszę zauważyć jak łatwo można wykorzystać typ wyliczeniowy do obsługi poszczególnych przypadków tego typu

Dodatkowo, proszę zwrócić uwagę na konwersję wartości nazw stałych w instrukcji wyświetlającej napis „Cześć czy lubię porę roku ”+`pory_roku.toString()`+“?”

Slajd 8




JAVA  
Dla początkujących

### Typ wyliczeniowy enum - rozwiązanie

```
public static void main(String[] args) {
    // TODO Auto-generated method stub

    EnumTest enumTest = new EnumTest();
    Lubie_pory_roku w = enumTest.new
    Lubie_pory_roku(Pory_roku.ZIMA);
    w.Czy_Lubie();
}
```

 Lekcja 8 - operacje wejścia wyjścia,

Wywołanie klasy zadeklarowanej jako klasę wewnętrzną można wykonać w zaprezentowany sposób.

Proszę zauważyć że konstruktor wołany jest ze stałą ustawianą typem `Pory_roku`.



Slajd 9

**JavaDocs – dokumentacja kodu**

- Czy dokumentacja jest potrzebna?
- Jak dokumentować?
- JavaDocs – narzędzie do dokumentowania
- Korzyści

Lekcja 6 i typ wyliczeniowy

Czy dokumentacja kodu jest potrzebna?  
Odp: Jest potrzebna. Wyobraźmy sobie że piszemy kod od dłuższego czasu. Powrót do napisanego kodu sprzed pół roku może okazać się trudny.

Krótki komentarz, opis zaraz przybliży nam do czego służy metoda, kto jaką wersję klasy i kiedy napisał

Jak dokumentować?

Oczywiście nie chodzi o pisanie wielostronicowych elaboratów. Opis ma być krótki, zrozumiały, zawierać tylko to co chcielibyśmy wiedzieć wykorzystując np. klasę. Najlepiej wg określonego szablonu

Narzędziem który wykorzystywany jest powszechnie jest JavaDocs. Używając środowiska Eclipse wykorzystujemy zintegrowane JavaDocs.

Slajd 10

**JavaDocs – dokumentacja kodu**

```
System.out.println("Kwadrat liczby to: "+a+".");
```

```
try {
```

```
    // zdefiniuj Buffer
```

```
    for(int i=0; i<10; i++)
```

```
    {
```

```
        System.out.println("Kwadrat liczby to: "+a+".");
```

```
    }
```

```
}
```

**void java.io.PrintStream.println(String x)**  
Prints a String and then terminate the line. This method behaves like `println()`.

**Parameters:**  
x The String to be printed.

@ Javadoc: ...

Lekcja 6 i typ wyliczeniowy

Wystarczy że najedziemy na metodę (tutaj `println()`) i otrzymujemy szybką informację jak dana metoda działa, jakie ma parametry

Dzięki temu nie musimy szukać informacji w dokumentacji.



Slajd 11

**JavaDocs – dokumentacja kodu**

```
EnumTest enumTest = new EnumTest();
Lubie_pory_roku w = enumTest.new Lubie_pory_roku(Pory_roku.LATOW.CzyLubie());
```

EnumTest.Lubie\_pory\_roku.Lubie\_pory\_roku.LATOW.CzyLubie()

Lekcja 6 i typ wyliczeniowy

A oto nasz ostatni kod (klasa Lubie\_pory\_roku). Po najechnaniu na nazwę klasy nie wyświetlają się żadne podręczne informacje.

Slajd 12

**JavaDocs – jak dokumentować**

- Komentarz
- Standard
- ```
/**
 * Funkcja obliczająca kwadrat liczby
 * @author karol.kowalski
 * @version 1.1
 */
```
- Predefiniowane tagi @

Lekcja 6 i typ wyliczeniowy

Dokumentacja kodu jest wykonywana w formie komentarza przed typem zmiennej, klasą, metodą.

Aby odróżnić dokumentację od zwykłego komentarza stosuje się znaczniki

```
/**
```

```
*/
```

Ustalono standard opisu i tagi

Slajd 13

**JavaDocs – przykład**

```
/**
 * Lista pór roku w naszej szerokości geograficznej
 *
 * @author karol.kowalski
 * @version 1.0
 * @see Pory_roku_ext
 */
public enum Pory_roku { WIOSNA, LATO, JESZEN, ZIMA }
```

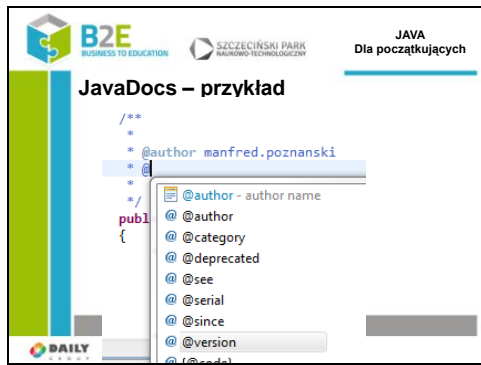
Lekcja 6 i typ wyliczeniowy

Proszę przyjrzeć się przykładowi który dokumentuje ostatnio zdefiniowany typ wyliczeniowy Pory\_roku.

Dzięki dołączonej dokumentacji każde użycie typu Pory\_roku powoduje podręczne wyświetlenie informacji.



Slajd 14



Narzędzie Eclipse wspiera przy pisaniu dokumentacji – lista dostępnych tagów

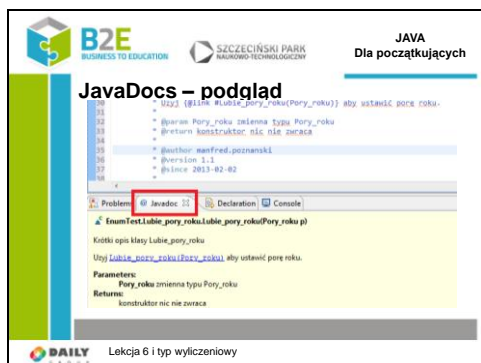
Slajd 15



Przyjęto że pierwszą linią dokumentującą będzie krótki opis co dany element wykonuje  
Następnie można dodać drugą linię (po znaczniku nowej linii <p>) zawierający opis metody i jej parametrów, oraz co zwraca metoda

Potem, zależnie od uznania tagi dot. autora, wersji daty wprowadzenia.

Slajd 16



Podgląd pisanej dokumentacji można śledzić na zakładce Javadoc



Slajd 17

**JavaDocs – przykład**

```
Pory_roku pory_roku; // lubienie albo n...
```

**EnumTest.Pory\_roku**

Lista pór roku w naszej szerokości geograficznej

**Version:**  
1.0

**Author:**  
karol.kowalski

**See Also:**  
[Pory\\_roku.est](#)

@

Press 'F2' for focus

Lekcja 6 i typ wyliczeniowy

Oto widok po najechnaniu na nazwę typu Pory\_roku

Slajd 18

**JavaDocs – zadanie**

- Proszę napisać dokumentację do ostatnio dodanej klasy `Lubie_pory_roku`
- Proszę opisać konstruktor
- Oraz metodę `Czy_lubie()`
- Proszę sprawdzić czy przy pisaniu oprogramowania wyświetla się dokumentacja

Lekcja 6 i typ wyliczeniowy

Oto widok po najechnaniu na nazwę typu Pory\_roku

Slajd 19

**JavaDocs – rozwiązanie zadania**

```
/**  
 * Klasa Lubie_pory_roku  
 * <p>  
 * Implementuje zaawansowany algorytm ekspercki do ustalania pref  
 roku użytkownika.</p>  
 * Konstruktor (@link #Lubie_pory_roku) ustawia wybraną porę roku  
 * Metoda (@link #Czy_lubie()) zwraca preferencje  
 *  
 * @author manfred.poznanski  
 * @version 1.1  
 * @since 2013-02-02  
 */  
public class Lubie_pory_roku
```

Lekcja 6 i typ wyliczeniowy

Oto przykładowa dokumentacja klasy Proszę zwrócić uwagę na linki do poszczególnych metod klasy



Slajd 20

**JavaDocs – rozwiązanie zadania**

```
/**
 * @author manfred.poznanski
 * @version 1.1
 * @since 2013-02-02
 */
public class Lubie_pory_roku
{
    Pory_roku pory_roku;

    /**
     * Krótki opis konstruktora Lubie_pory_roku
     * <p>
     * Użyj (@link #Lubie_pory_roku(Pory_roku)) aby ustawić pory_roku
     * @param Pory_roku zmienna typu Pory_roku
     * @return konstruktor nic nie zwraca
     */
    public Lubie_pory_roku(Pory_roku pory_roku) {
        this.pory_roku = pory_roku;
    }

    /**
     * Metoda wyświetlająca preferencje pór roku użytkownika
     * <p>
     * Metoda jest bezparametrowa (wykorzystuje pole klasy pory_roku)
     * @return przesyła na standardowe urządzenie wyjściowe (console)
     * z preferencjami
     */
    public void czyLubie() {
        // ...
    }
}
```

Lekcja 6 i typ wyliczeniowy

Oto przykładowa dokumentacja klasy  
Proszę zwrócić uwagę na linki do poszczególnych metod klasy

Slajd 21

**JavaDocs – rozwiązanie zadania**

```
/**
 * Krótki opis konstruktora Lubie_pory_roku
 * <p>
 * Użyj (@link #Lubie_pory_roku(Pory_roku)) aby ustawić pory_roku
 * @param Pory_roku zmienna typu Pory_roku
 * @return konstruktor nic nie zwraca
 */
public Lubie_pory_roku(Pory_roku p) {
    // ...
}
```

Lekcja 6 i typ wyliczeniowy

Slajd 22

**JavaDocs – rozwiązanie zadania**

```
/**
 * Metoda wyświetlająca preferencje pór roku użytkownika
 * <p>
 * Metoda jest bezparametrowa (wykorzystuje pole klasy pory_roku)
 * @return przesyła na standardowe urządzenie wyjściowe (console)
 * z preferencjami
 */
public void czyLubie() {
    // ...
}
```

Lekcja 6 i typ wyliczeniowy



Slajd 23

JavaDocs – rozwiązanie zadania

```
11/* * lista pór roku w naszej szerokości geograficznej!
12 public enum Pory_roku { WIOSNA, LATO, ZESZEN, ZIMA }
13
14/* * klasa lubie_pory_roku!
15 public class Lubie_pory_roku
16 {
17     Pory_roku pory_roku;
18
19/* * krótki opis konstruktora lubie_pory_roku!
20 lubie_pory_roku(Pory_roku p) {
21     this.pory_roku = p;
22
23/* * metoda udzielaająca preferencje pór roku użytkownika!
24 public void czy_lubie() {
25     System.out.println("Czyli czy lubię pór roku " + pory_roku.toString());
26     switch (pory_roku) {
27         case WIOSNA: System.out.println("Lubię wiosnę");break;
28         case LATO: System.out.println("Nawet lubię lato");break;
29         case ZESZEN: System.out.println("Nech tak sobie");break;
30         default: System.out.println("Zyle padal śnieg");break;
31     }
32 }
```

Lekcja 6 i typ wyliczeniowy

Dokumentacja nie musi zaśmieczać kodu – można ją zwinąć żeby nie przeszkadzała do jednej linii.

Slajd 24

JavaDocs – dokumentacja html

- Cały projekt wraz z dokumentacją można zapisać w HTML-u
- W tym celu należy wybrać opcję „Generate Javadoc”

Lekcja 6 i typ wyliczeniowy

Po uruchomieniu kreatora wybrać folder docelowy gdzie będzie zapisana strona z dokumentacją (Destination) i nacisnąć Finish

Slajd 25

JavaDocs – dokumentacja html

Enum EnumTestPory\_roku

```
enum EnumTestPory_roku {
    WIOSNA, LATO, ZESZEN, ZIMA;
}

public class EnumTestPory_roku {
    EnumTestPory_roku pory_roku;

    EnumTestPory_roku(EnumTestPory_roku pory_roku) {
        this.pory_roku = pory_roku;
    }

    public void czy_lubie() {
        System.out.println("Czyli czy lubię pór roku " + pory_roku.toString());
        switch (pory_roku) {
            case WIOSNA: System.out.println("Lubię wiosnę");break;
            case LATO: System.out.println("Nawet lubię lato");break;
            case ZESZEN: System.out.println("Nech tak sobie");break;
            default: System.out.println("Zyle padal śnieg");break;
        }
    }
}
```

Lekcja 6 i typ wyliczeniowy

Oto rezultat generowania. Automatycznie powstaje dokument HTML który przedstawia hierarchię klas, dla każdej klasy opis, listę metod, drzewo relacji i dokumentację którą wpisaliśmy



Slajd 26

**Pytania**

1. Co to jest typ wyliczeniowy?
2. Do czego stosujemy typ enum?
3. Jakie zalety w programowaniu ma typ enum?
4. Co to jest JavaDoc?
5. Jak należy dokumentować kod w JavaDoc?
6. Dlaczego dokumentowanie jest potrzebne?

Lekcja 6 i typ wyliczeniowy

### 8.6.3 Opis założonych osiągnięć ucznia

Po tej lekcji uczniowie będą umieli poznawać informacje na temat wykorzystywanych obiektów i ich metod z dokumentacji. W łatwy sposób operować typami wyliczeniowymi.

## 8.7 Lekcja 7 - Tablice jednowymiarowe i wielowymiarowe, kolekcje

### 8.7.1 Cel lekcji

Celem lekcji jest przedstawienie podstawowych możliwości przechowywania zbiorów obiektów w tablicach, lub kolekcjach. Wy tłumaczone będzie pojęcie listy.

### 8.7.2 Treść - slajdy z opisem

Slajd 1

**Java - lekcja 7**

Tablice jednowymiarowe i wielowymiarowe, kolekcje



Slajd 2

**B2E** **SZCZECIŃSKI PARK** **JAVA**  
BUSINESS TO EDUCATION NAUKOWO-TECHNOLOGICZNY Dla początkujących

**Co to jest tablica?**

- **Obiekt**
- **Określony typ elementów**
- **Długość**

**DAILY** Lekcja 7 – tablice, kolekcje

Tablicą nazywamy obiekt zawierający zestaw elementów określonego typu. Długość i typ tablicy są określone przy jej deklarowaniu

Struktura taka pozwala na zarządzanie w programie zbiorem elementów tego samego typu. Np. zestaw ostatnich 10 pomiarów temperatury.

Slajd 3

**B2E** **SZCZECIŃSKI PARK** **JAVA**  
BUSINESS TO EDUCATION NAUKOWO-TECHNOLOGICZNY Dla początkujących

**Deklaracja tablicy jednowymiarowej**

```
int[] tablica; // deklaracja - tablica będzie zawierać typu int
tablica = new int[8]; // inicjacja - długość tablicy jest 8

tablica[0] = 100; // pierwszy element tablicy
tablica[1] = 220;
tablica[2] = 250;
tablica[3] = 280;
tablica[4] = 300;
tablica[5] = 310;
tablica[6] = 400;
tablica[7] = 485; // ostatni element tablicy

System.out.println("Element tablicy o indeksie 3=" + tablica[3]);
```

**DAILY** Lekcja 7 – tablice, kolekcje

**int[] tablica;**

W pierwszej linii wykonywana jest deklaracja tablicy. Mówi ona że:

- tablica będzie zawierać typ całkowity (int),
- nazwa obiektu tablicy to ... tablica

**tablica = new int[8];**

Inicjowana jest tablica i ustawiana jest jej długość

Linie zawierające „tablica[1] = 220;” ustawiają kolejne wartości tablicy

Co robi ostatnia linia kodu?

Odp. Wyświetla napis „Element tablicy o indeksie 3=280”

Co zrobić aby wyświetlić pierwszą wartość tablicy?

Odp. Ustawić indeks tablica[0]



Slajd 4

**Tablica jednowymiarowa**

Jaki będzie efekt wywołania takiej linii dla kodu z poprzedniego slajdu?

```
System.out.println("Element tablicy o indeksie 8="+tablica[8]);
```

```
Exception in thread "main"  
java.lang.ArrayIndexOutOfBoundsException: 8  
at ArrayTest.main(ArrayTest.java:22)
```

Lekcja 7 – tablice, kolekcje

Zgłoszony zostanie błąd

Array Index Out Of Bounds Exception –  
Wyjątek: Indeks tablicy poza zakresem

Oznacza to, że nie wolno wychodzić poza zakres tablicy bo spowoduje to wyjątek i przerwanie działania programu.

Slajd 5

**Tablica jednowymiarowa - zadanie**

Proszę napisać program który:

- zadeklaruje tablicę 10 elementową typu int
- wypełni ją dowolnymi wartościami
- wyświetli wszystkie wartości w konsoli

Lekcja 7 – tablice, kolekcje

Sugestia: wykorzystać pętlę for

Slajd 6

**Rozwiązanie zadania**

```
int[] tablica; // deklaracja  
tablica = new int[10]; // inicjacja  
  
for (int i=0; i<10; i++) // iteruj od 0 do 9  
    tablica[i] = i*100;  
  
for (int i=0; i<10; i++) // iteruj od 0 do 9  
    System.out.println("Element tablicy o indeksie "+i+"="+tablica[i]);
```

Element tablicy o indeksie 0=0  
Element tablicy o indeksie 1=100  
Element tablicy o indeksie 2=200  
Element tablicy o indeksie 3=300  
Element tablicy o indeksie 4=400  
...

Lekcja 7 – tablice, kolekcje

Zastosowanie pętli for i zmiennej „i” do iterowania po elementach tablicy

Proszę zwrócić uwagę na:



Zmienna „i” jest wykorzystana do ustalenia wartości tablicy = i\*100;

Zmienna „i” jest wykorzystana do wyświetlenia indeksu przy wyświetlaniu wyników





Slajd 7



**JAVA**  
Dla początkujących

### Tablica jednowymiarowa - zadanie

Proszę zmodyfikować poprzedni program tak aby:

- wyświetlał sumę wartości w konsoli

**DAILY** Lekcja 7 – tablice, kolekcje

Sugestia: wykorzystać pętlę for dla wszystkich elementów tablicy  
**for (int x:tablica)**

Slajd 8



**JAVA**  
Dla początkujących

### Rozwiązanie zadania

```
int suma =0;
for (int x:tablica)
    suma+=x;

System.out.println("Suma elementów tablicy wynosi "+suma);
```

Suma elementów tablicy wynosi 4500

**DAILY** Lekcja 7 – tablice, kolekcje

Zastosowanie pętli for dla wszystkich elementów tablicy nie wymaga iteratora (zmiennej „i”).

Proszę zwrócić uwagę na formułę `suma+=x`;  
Co ona oznacza?  
Odp: `suma = suma + x`;

Slajd 9



**JAVA**  
Dla początkujących

### Tablica – inny sposób deklaracji

```
int[] tablica = {12,23,45,67,89,0,1,2,3,4};
```

Element tablicy o indeksie 0=12  
Element tablicy o indeksie 1=23  
Element tablicy o indeksie 2=45  
Element tablicy o indeksie 3=67  
Element tablicy o indeksie 4=89  
Element tablicy o indeksie 5=0  
Element tablicy o indeksie 6=1  
Element tablicy o indeksie 7=2  
Element tablicy o indeksie 8=3  
Element tablicy o indeksie 9=4  
Suma elementów tablicy wynosi 246

**DAILY** Lekcja 7 – tablice, kolekcje

Inny sposób deklarowania i inicjowania tablicy – przyda się tylko wówczas gdy wartości tablicy będą w programie stałe.



Slajd 10

**Łańcuch znaków - string**

```
String nazwisko = "Kowalski";  
System.out.println("Nazwisko =" + nazwisko);
```

Lekcja 7 – tablice, kolekcje

Łańcuch znaków String jest tablicą jednowymiarową zawierającą elementy typu znak (char).

Proszę sprawdźcie jak będzie działał powyższy program.

Odp. Jak się można spodziewać program wyświetli nazwisko „Kowalski”.

Slajd 11

**Rozmiar tablicy**

```
System.out.println("Liczba  
elementów tablicy wynosi " + tablica.length);  
System.out.println("Długość nazwiska =" + nazwisko.length());
```

Lekcja 7 – tablice, kolekcje

Typ String jest specyficznym typem tablicowym i posiada metody do wykonywania operacji na znakach. Metoda length() zwraca długość napisu zapamiętanego w zmiennej.

Typ tablicowy posiada licznik .length który zwraca liczbę elementów w tablicy

Slajd 12

**Tablica dwuwymiarowa**

W każdym elemencie tablicy x znajduje się jest kolejna tablica y

Rozmiar = 8

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0 |   |   |   |   |   |   |   |
| 1 |   |   |   |   |   |   |   |
| 2 |   |   |   |   |   |   |   |
| 3 |   |   |   |   |   |   |   |

Element o indeksie 6,0  
Element o indeksie 3,2

Lekcja 7 – tablice, kolekcje

Rozważmy tablicę dwuwymiarową 8x4  
Jak będzie wyglądała deklaracja takiej tablicy?



Slajd 13

**Deklaracja tablicy dwuwymiarowej**

```
int[][] tablica2;  
tablica2 = new int[8][4];  
  
int[][] tablica3;  
tablica3 = new int[8][4][3];
```

Lekcja 7 – tablice, kolekcje

Jak zadeklarować tablicę trójwymiarową 8x4x3?

Slajd 14

**Tablica dwuwymiarowa - zadanie**

Proszę program który:

- Zadeklaruje tablicę dwuwymiarową 8x4
- Ustawi jako wartość tablicy iloczyn indeksów np. dla indeksu 2,2 wartość będzie 4
- Wyświetli wszystkie elementy tablicy dwuwymiarowej
- Obliczy i wyświetli sumę wszystkich elementów tablicy

Lekcja 7 – tablice, kolekcje

Sugestia: wykorzystać 2x pętle for dla wszystkich elementów tablicy w wymiarze x i y (8,4)

Kto poda pierwszy wynik sumy?  
Odp: 168

Slajd 15

**Rozwiązanie zadania**

```
int[][] tablica2;  
tablica2 = new int[8][4];  
int suma2 = 0;  
  
for (int i=0;i<8;i++)  
    for (int j=0;j<4;j++)  
        tablica2[i][j] = i*j;  
  
for (int i=0;i<8;i++)  
    for (int j=0;j<4;j++)  
    {  
        suma2+=tablica2[j][i];  
        System.out.println("Element tablicy o indeksie "+i+  
            ", "+j+"="+tablica2[i][j]);  
    }  
    System.out.println("Suma elementów tablicy  
    dwuwymiarowej wynosi "+suma2);
```

Lekcja 7 – tablice, kolekcje

W rozwiązaniu zadania jest błąd.

Kto wskaże w którym miejscu, na czym błąd polega i co stałoby się przy uruchomieniu tego programu.

Należy bardzo uważać żeby nie pomylić kolejności iteratorów



Slajd 16

**JAVA**  
Dla początkujących

**Inne sposoby zarządzania tablicami**

Predefiniowane struktury:

- Collection
- LinkedList
- ArrayList
- HashMap

**DAILY** Lekcja 7 – tablice, kolekcje

Zarządzanie tablicami z przykładów jest na dłuższą metę dość uciążliwe.

Wady:

- Musimy pilnować zakresów,
- Musimy deklarować typy obiektów,
- Aby wyszukać dany element musimy napisać parę linii kodu,
- ...

W celu wyeliminowania tych ale również innych wad środowisko programistów przygotowało sobie specjalne narzędzia. Są to struktury pozwalające na łatwe zarządzanie danymi.

Slajd 17

**JAVA**  
Dla początkujących

**ArrayList**

Narzędzia do zarządzania listami danych:

- Collection
- LinkedList
- ArrayList
- HashMap

**DAILY** Lekcja 7 – tablice, kolekcje

Zarządzanie tablicami z przykładów jest na dłuższą metę dość uciążliwe.

Wady:

- Musimy pilnować zakresów,
- Musimy deklarować typy obiektów,
- Aby wyszukać dany element musimy napisać parę linii kodu,
- ...

W celu wyeliminowania tych ale również innych wad środowisko programistów przygotowało sobie specjalne narzędzia. Są to struktury pozwalające na łatwe zarządzanie danymi.



Slajd 18

**JAVA**  
Dla początkujących

### Przykład listy

```

import java.util.*; //potrzebna biblioteka narzędziowa
=
List l = new ArrayList(); // deklaracja i inicjacja listy
for (int i=0; i<10; i++)
    l.add(new Integer(i)); // dodawanie obiektów do listy
Iterator i = l.iterator(); // powołanie iteratora
while (i.hasNext()) // pętla po wszystkich elementach
    System.out.println("Element listy "+i.next());
// wyświetlenie kolejnych elementów
    
```

Lekcja 7 – tablice, kolekcje

Pierwsza uwaga do przykładu – należy wykorzystać bibliotekę narzędzi `java.util`. Dla uproszczenia dodamy wszystkie narzędzia w domenę `util` poprzez instrukcję

**import java.util.\*;** - gwiazdka oznacza że wszystkie biblioteki w danej bibliotece będą dołączone do programu.

Alternatywnie można dodać następujące linie:

```

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

Deklaracja listy o nazwie „l” jest prosta: **List l = new ArrayList();**

Warto zauważyć że nie deklarowaliśmy typu tablicy.

Następnie dodawane są **obiekty** do listy (10 elementów). Aby powołać obiekt stosujemy instrukcję `new Integer(<wartość>)` - nowy obiekt typu `Integer`.

Aby wyświetlić wszystkie elementy z listy należy powołać **iterator**. Jest to obiekt pozwalający na dostęp do kolejnych elementów listy.

Zastosowano pętlę `while` której sprawdza czy jest na iteratorze kolejny element. Jeśli tak instrukcja `i.next()` zwraca obiekt i jednocześnie przechodzi do kolejnego elementu listy.



Slajd 19

**JAVA**  
Dla początkujących

### Lista - zadanie

Proszę przerobić poprzedni program tak aby:

- Zarządzał łańcuchami znaków zamiast liczbami
- Dodanych zostało 20 elementów zamiast 10
- Wyświetlone zostały wszystkie elementy listy

Lekcja 7 – tablice, kolekcje

Sugestia: wykorzystać obiekt String analogicznie jak Integer

Slajd 20

**JAVA**  
Dla początkujących

### Rozwiązanie

```

for (int i=1;i<20;i++)
    l.add(new String("test "+i)); // dodawanie obiektów String do listy

for (int i=1;i<20;i++)
    l.add(new Integer(i+20)); // dodawanie obiektów Integer do listy
    
```

Element listy =test 19  
Element listy =21

Lekcja 7 – tablice, kolekcje

Wystarczy w jednym miejscu zmienić instrukcję powołującą nowe obiekty

Z new Integer(i) na np. new String(„tekst”+i)

Co warto zauważyć:

Przerobienie programu jest dużo prostsze niż w przypadku tablic [][]

Jedno miejsce

Nie musimy zajmować się kontrolą liczby elementów

Nic nie stoi na przeszkodzie aby lista zarządzała różnymi obiektami np. String i Integer

Mamy do dyspozycji całą paletę narzędzi do zarządzania listą – następny slajd

## Slajd 21

**JAVA**  
Dla początkujących

**Lista – dodatkowe funkcje**

...  
Lekcja 7 – tablice, kolekcje

Są wśród nich:

- Dodawanie i wstawianie obiektów (add), na pozycję, dodawanie całej kolekcji
- Czyszczenie listy
- Wyszukiwanie obiektów
- Pobieranie obiektów wg nr
- Usuwanie wg indeksów, wg obiektów, wszystkich
- Tworzenie podlist
- Ustawianie obiektu na pozycji
- Sprawdzanie rozmiaru
- Iterowanie

## Slajd 22

**JAVA**  
Dla początkujących

**Pytania**

1. Jak zadeklarować listę 1-dno wymiarową?
2. Jak wypełnić listę wartościami – zaproponuj krótki program?
3. Co się stanie przy przekroczeniu zakresu tablicy?
4. Jak sprawdzić długość tablicy?
5. Jak zadeklarować listę trójwymiarową?
6. Co to jest ArrayList i do czego służy?
7. Do czego służy obiekt Iterator?
8. W jakiej domenie znajdują się narzędzia do obsługi list?

...  
Lekcja 7 – tablice, kolekcje

### 8.7.3 Opis założonych osiągnięć ucznia

Po tej lekcji uczniowie będą umieli gromadzić obiekty w tablicach i kolekcjach. Posiadą również umiejętność przeglądanie tych struktur za pomocą pętli.

## 8.8 Lekcja 8 - Operacje wejścia – wyjścia i wyjątki

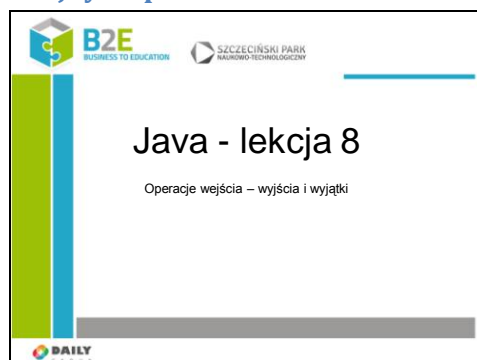
### 8.8.1 Cel lekcji

Celem lekcji jest prezentacja możliwości zabezpieczenia aplikacji przed błędami, oraz umiejętność wymiany danych z zasobami z poza programu (sieć, pliki lokalne).

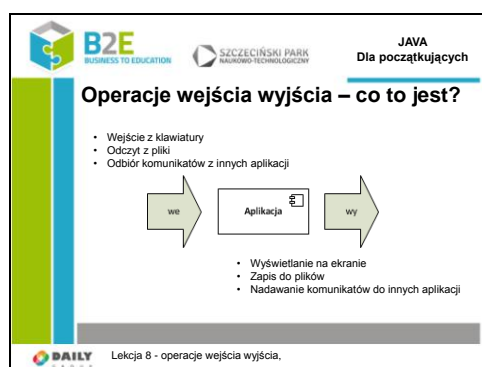


## 8.8.2 Treść - slajdy z opisem

### Slajd 1



### Slajd 2



Każdy program powinien mieć możliwość komunikowania się z użytkownikiem, systemem operacyjnym czy innymi aplikacjami.

W języku Java komunikacja ta jest realizowana za pomocą strumieni. Obsługa strumieni jest zaimplementowana w domenie **system.io** za pomocą obiektów out i in.

**import java.io.\***

Przykładem jest często wykorzystywana metoda print lub println wyświetlająca napis w konsoli `System.out.println("Zła liczba „");`



Slajd 3

Slide 3 content: Drukowanie na ekranie. System.out.println("Tekst Tekst Tekst");. Lektura 8 - operacje wejścia wyjścia.

Przykładem użycia strumienia wyjściowego out jest wyświetlenie tekstu w konsoli.

W powyższym przykładzie program wysła do strumienia zewnętrznego tekst.

Przykład wykorzystuje standardowy strumień out zdefiniowany w obiekcie System.

Slajd 4

Slide 4 content: Java code for reading input using BufferedReader and InputStreamReader. Lektura 8 - operacje wejścia wyjścia i wyjątki.

Operacje wejści-wyjścia w Javie polegają na obsłudze strumieni. Maszyna wirtualna udostępnia obiekty, które, np. odczytują znaki z klawiatury.

Przykład podany na slajdzie odczytuje cały wiersz wpisany do konsoli.

Pojawił się nowy element: bloki "try-catch". Jest to mechanizm obsługi błędów. W przypadku gdybyśmy utracili połączenie z urządzeniem z którego czytamy, program jest w stanie obsłużyć błąd i zareagować w odpowiedni sposób. Uchroni to naszą aplikację od nieprzewidzianego zachowania.

Slajd 5

Slide 5 content: Java code for printing escape characters. Lektura 8 - operacje wejścia wyjścia i wyjątki.

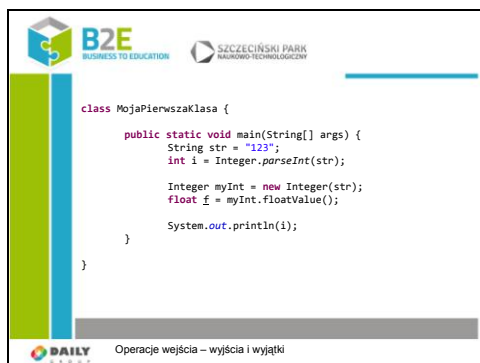
\ " - cudzysłów  
\n – znak nowej linii  
\ – znak \

\\ - znak \

\t - tabulator



Slajd 6



Każdemu typowi wbudowanemu odpowiada Klasa. W praktyce możemy stosować obiekty i typy wbudowane zamiennie. Nie ma to znaczenia dla obliczeń. Możemy również stosować operatory matematyczne, tak samo, jak dla typów wbudowanych.

Po przypisaniu nowego obiektu typu "Integer" do referencji "myInt" możemy sprawdzić jakie metody należą do tej klasy. W tym celu po wpisaniu nazwy i kropki wciskamy "Ctrl" + "Spacja". Środowisko Eclipse poda nam automatycznie listę metod. Możemy poruszać się po niej "góra-dół" za pomocą strzałek. Zobaczymy również opisy każdej z metod. Wykonując ćwiczenia sprawdzamy czym są obiekty, których używamy.

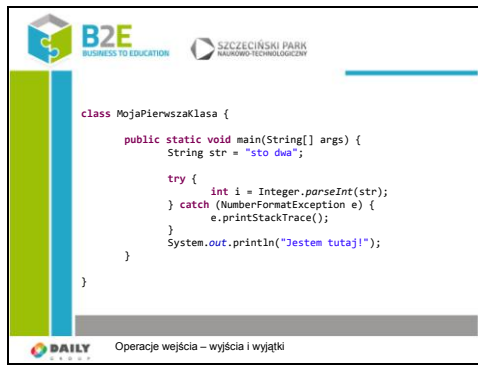
Slajd 7



M



Slajd 8



W przypadku próby stworzenia instancji klasy "Integer", korzystając z nieprawidłowego łańcucha znaków, program zakończy się niepowodzeniem.

Możemy tego uniknąć stosując blok "try-catch", tak jak na przykładzie. W bloku "try" wykonują się polecenia potencjalnie "niebezpieczne". Oczywiście łańcuch znaków "sto dwa" jest nieprawidłowy i przejdziemy do bloku "catch". Problem zostanie przechwycony i wykonają się polecenie z tego bloku. Na ekranie zobaczymy cały stos programu. W przyszłości, w dużych aplikacjach, będziemy mogli łatwo odszukać klasę, metodę, a nawet wers, w którym pojawił się błąd, wraz z opisem błędu.

Fakt, że w konsoli pokazał się raport o błędzie, nie wpływa na niestabilność programu. Napis "Jestem tutaj!" również będzie widoczny. Uniknęliśmy niespodziewanego przerwania programu.

Slajd 9



Ćwiczenie:  
Wykorzystując pierwszy przykład z bieżącej lekcji wczytaj z konsoli dwie liczby. Wykorzystaj fakt, że każda referencja może wskazywać na pusty obiekt „null”.

```
Integer i = null;
```

Następnie możesz wykonać porównanie, aby sprawdzić, czy nadal jest „null”.

```
i == null
```

## Slajd 10

```
public class ZeroParameterException extends Exception {  
    public ZeroParameterException(String message) {  
        super(message);  
    }  
}
```

Operacje wejścia – wyjścia i wyjątki

Stworzone przez nas klasy mogą powodować charakterystyczne tylko dla nich błędy. Warto, aby w takim przypadku rzucały wyjątek stworzony specjalnie dla nich.

Stworzenie takiego wyjątku jest bardzo proste. Wystarczy dziedziczyć po klasie "Exception" i zaimplementować przynajmniej jeden konstruktor.

## Slajd 11

```
class MojaPierwszaKlasa {  
    public static void main(String[] args) {  
        int a = 2, b = 0;  
        try {calculate(a, b);} catch (ZeroParameterException e) {  
            e.printStackTrace();  
        }  
    }  
    private static int calculate(int a, int b)  
    throws ZeroParameterException {  
        if (a == 0) {throw new ZeroParameterException(  
            "Argument a jest zerem!");}  
        if (b == 0) {throw new ZeroParameterException(  
            "Argument b jest zerem!");}  
        return a + b;  
    }  
}
```

Operacje wejścia – wyjścia i wyjątki

Jeżeli metoda może wyjątek wyrzucić, należy zadeklarować to za listą argumentów, tak jak na slajdzie.

Kiedy zajdzie niepożądana sytuacja (w tym przypadku jeden z argumentów będzie zerem), wyjątek powinien być wyrzucony. Użycie "throw" kończy wywołanie metody. Nie zwróci ona żadnej wartości.

## Slajd 12

JAVA  
Dla początkujących

**Jak pobrać dane z klawiatury?**

```
System.out.println("Podaj wartość argumentu: ");  
double a = 0;  
  
BufferedReader br=new BufferedReader(new  
    InputStreamReader(System.in));
```

Lekcja 8 - operacje wejścia wyjścia,

W celu odczytu danych z klawiatury należy przygotować urządzenie wejściowe.

Przykładowa deklaracja może wyglądać następująco:

```
BufferedReader br=new  
BufferedReader(new  
InputStreamReader(System.in));
```

Obiekt br typu BufferedReader będzie czytać wprost ze strumienia wejściowego System.in



Slajd 13



Slide 13 content: A presentation slide titled "JAVA Dla początkujących" with the subtitle "Jak pobrać dane z klawiatury?". It shows a code snippet for importing java.io.\* and creating a BufferedReader. The slide includes logos for B2E, Szczeciński Park Naukowo-Technologiczny, and DAILY Group. At the bottom, it says "Lekcja 8 - operacje wejścia wyjścia."

Aby wykorzystać klasę `BufferedReader` i `InputStreamReader` należy użyć bibliotek `java.io`. W tym celu na samym początku kodu należy dodać linię:

```
import java.io.*;
```

Slajd 14



Slide 14 content: A presentation slide titled "JAVA Dla początkujących" with the subtitle "Jak pobrać dane z klawiatury?". It shows a code snippet for reading a line of input using `br.readLine()` and printing it. The slide includes logos for B2E, Szczeciński Park Naukowo-Technologiczny, and DAILY Group. At the bottom, it says "Lekcja 8 - operacje wejścia wyjścia."

Jakiego typu są odczytane dane z klawiatury?

Odp: Są to znaki, w szczególności ciąg znaków czyli typ `String`.

Sprawdźmy czy działa – wprowadźmy ciąg znaków z klawiatury, po zatwierdzeniu enterem niech wyświetli się wprowadzona wartość.

Proszę dodajcie linię:

```
String b = "";  
b = br.readLine();
```

Eclipse automatycznie zwróci uwagę, że `br.readLine()`; należy otoczyć sekcją krytyczną.

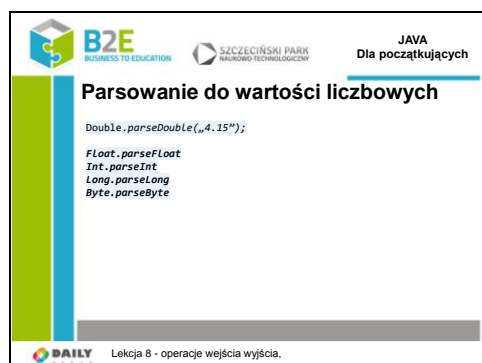
Wszystkim działa?

Pytanie: Co należy zrobić aby program obliczył potęgę wpisanej przez użytkownika liczby?

Odp:

- Skonwertować ją do typu liczbowego double
- Wykonać operację mnożenia

Slajd 15



Slide 15 content: The slide features logos for B2E, Szczeciński Park Naukowo-Technologiczny, and DAILY. The title is 'JAVA Dla początkujących'. The main heading is 'Parsowanie do wartości liczbowych'. Below it, the code `Double.parseDouble("4.15");` is shown. A list of parsing methods follows: `Float.parseFloat`, `Int.parseInt`, `Long.parseLong`, and `Byte.parseByte`. The footer includes the DAILY logo and the text 'Lekcja 8 - operacje wejścia wyjścia.'

Powyższa formuła dokonuje parsowania. Podobnie można zrobić dla innych typów

Uwaga!

Co zwróci formuła w przypadku gdy zadana wartość nie będzie nadawała się do konwersji na liczbę?

Oczywiście wygenerowany zostanie wyjątek. Dlatego należy instrukcję również otoczyć sekcją krytyczną try/catch

Pytanie:

Jak należy dokończyć nasz program?

- Parsowanie
- Obsługa błędu parsowania – sekcja catch
- Obliczenie  $a^a$
- Wyświetlenie





Slajd 16



JAVA  
Dla początkujących

### Jak pobrać dane z klawiatury?

```
double a = 0;
BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));
System.out.print("Podaj liczbę ");



try {
a = Double.parseDouble(br.readLine());
} catch (NumberFormatException | IOException e) {
// TODO Auto-generated catch block
e.printStackTrace();
System.out.println("Nieprawidłowa Liczba "+e.getMessage()+"!");
}

System.out.println("Kwadrat liczby to: "+a*a+");
```

 Lekcja 8 - operacje wejścia wyjścia,


Program powinien wyglądać mniej więcej tak:

Slajd 17



### Ćwiczenie:

Napisz program liczący pieniąstki trójmianu kwadratowego. W przypadku p

 Operacje wejścia – wyjścia i wyjątki





Slajd 18

```
import java.io.IOException;
import java.io.InputStream;
import java.net.MalformedURLException;
import java.net.URL;
import java.net.URLConnection;

class MojaPierwszaKlasa {
    public static void main(String[] args) {
        try {
            URL url = new URL("http://google.com");
            URLConnection con = url.openConnection();
            InputStream is = con.getInputStream();

            is.close();
        } catch (MalformedURLException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Operacje wejścia – wyjścia i wyjątki

Strumienie są bardzo uniwersalne. Java umożliwia w łatwy sposób obsługę sieci za ich pomocą.

Spójrz na wyjątki i spróbuj odgadnąć jakie problemy mogą wystąpić w trakcie połączenia z internetem.

Nieprawidłowy adres URL i błąd odczytu (zerwane połączenie).

Odpowiednie nazewnictwo czyni pracę programisty łatwiejszą.

Slajd 19

**Ćwiczenie:**

Wykorzystując kod z poprzedniego przykładu połącz się z dowolną stroną internetową. Sprawdź jakie metody możesz operować na obiekcie typu `InputStream`.

Operacje wejścia – wyjścia i wyjątki

Slajd 20

```
JAVA
Dla początkujących

Zapis do pliku

// zdefiniowanie strumienia wyjściowego
BufferedWriter out = new BufferedWriter(new FileWriter("plik_wy.txt"));

for(int i=0; i<15; i++)
{
    out.write("Linia "+i); // zapis linii tekstu
    out.newLine(); // znacznik nowej linii
}

out.close(); // zamknięcie strumienia
```

Lekcja 8 - operacje wejścia wyjścia,

Przykładem użycia strumienia wyjściowego `out` jest zapis tekstu do pliku.

Jak widać jest on znacznie bardziej skomplikowany niż przykład wyświetlania tekstu na ekranie.

Pierwszym etapem jest zdefiniowanie obiektu `out` powiązanego z plikiem „plik\_wy.txt”.



Następnie w 15 iteracjach zapisywane są kolejne linie do pliku – np. tekst („Linia 1”)

Ostatecznie następuje zamknięcie strumienia `out.close()`;


Uwaga:  
Aby skorzystać z klas `BufferedWriter`,

FileWriter należy zaimportować  
bibliotekę **import java.io.\***

Slajd 21

  **JAVA**  
Dla początkujących  
**Zadanie**  
Napisz program który:  

- zapisze do pliku o nazwie wyniki.txt
- wyniki mnożenia liczb zakresu 20 do 10

 Lekcja 8 - operacje wejścia wyjścia,

Zadanie nie powinno trwać dłużej niż 10 minut

Należy przypomnieć o lekcji związanej z sekcjami krytycznymi try/catch

Slajd 22

  **JAVA**  
Dla początkujących  
**Rozwiązanie**



```
try {  
    // zdefiniowanie strumienia wyjściowego  
    BufferedWriter out = new BufferedWriter(new FileWriter("wyniki.txt"));  
    for(int i=1;i<=20;i++)  
        for (int j=0;j<=10;j++)  
        {  
            out.write("wynik = "+i*j);// zapis wyników  
            out.newLine();// znacznik nowej linii  
        }  
    out.close();// zamknięcie strumienia  
} catch (IOException e) {  
    // TODO Auto-generated catch block  
    e.printStackTrace();  
}
```

 Lekcja 8 - operacje wejścia wyjścia,


Uwaga,  
System zgłosi potrzebę zastosowania sekcji krytycznej (try/catch) do obsługi ew. błędów IO



Slajd 23



```
class MojaPierwszaKlasa {
    public static void main(String[] args) {
        try {
            FileInputStream fstream =
                new FileInputStream("plik.txt");
            DataInputStream in = new DataInputStream(fstream);
            BufferedReader br =
                new BufferedReader(new InputStreamReader(
                    in));
            String strLine;
            while ((strLine = br.readLine()) != null) {
                System.out.println(strLine);
            }
            in.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```



Operacje wejścia – wyjścia i wyjątki

Podstawową operacją na strumieniach jest obsługa plików. Przykład pokazuje jak odczytać plik.

Domyślna lokalizacja pliku jest w folderze naszego projektu.

Slajd 24



Ćwiczenie:

Korzystając z klas „FileWriter” i „BufferedWriter” utwórz plik i zapisz w nim w



Operacje wejścia – wyjścia i wyjątki

Slajd 25



Ćwiczenie:

Korzystając z klas „FileWriter” i „BufferedWriter” utwórz plik i zapisz w nim w



Operacje wejścia – wyjścia i wyjątki



Slajd 26



### 8.8.3 Opis założonych osiągnięć ucznia

Po tej lekcji uczniowie będą skorzystać z połączenia sieciowego i plików lokalnych w komputerze. Ponadto będą umieli zabezpieczyć swój program przed potencjalnymi błędami.

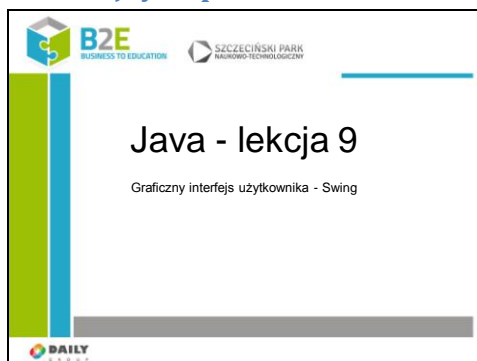
## 8.9 Lekcja 9 - Graficzny interfejs użytkownika - Swing

### 8.9.1 Cel lekcji

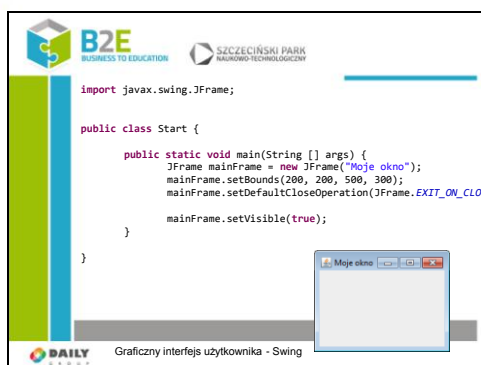
Celem lekcji jest pokazanie możliwości graficznego pakietu Swing. Wytłumaczenie jak poprawnie korzystać z tych elementów, oraz jak zarządzać ich rozkładem w oknie.

### 8.9.2 Treść - slajdy z opisem

Slajd 1



Slajd 2



Okno możemy stworzyć korzystając z klasy "JFrame".

Łańcuch znaków podany w konstruktorze jest tytułem okna.

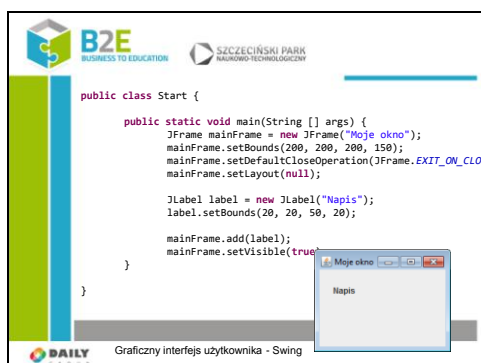
Metoda "setBounds" posiada 4 argumenty:

- odległość w poziomie od lewego, górnego rogu,
- odległość w pionie od lewego, górnego rogu,
- szerokość okna,
- wysokość okna.

Jeżeli nie ustawimy metody "setDefaultCloseOperation", to kliknięcie przycisku zamknięcia w prawym, górnym rogu nie spowoduje wyłączenia programu. Okno zniknie, ale aplikacja nadal będzie chodziła w tle!

Ostatnie polecenie powoduje wyświetlenie się okna.

Slajd 3



Do dyspozycji mamy wiele gotowych komponentów. Jeżeli chcemy wyświetlić tekst możemy skorzystać z klasy "JLabel".

Skorzystajmy z metody "setLayout" z argumentem "null". Będziemy wtedy mogli skorzystać z absolutnego pozycjonowania komponentów widoku. W późniejszych slajdach zostanie wytłumaczone, jak układać elementy korzystając z zarządców rozkładu (layoutów).

Nie zapomnijmy dodać obiektu "label" do okna.

Slajd 4

```

public class Start {
    public static void main(String [] args) {
        JFrame mainFrame = new JFrame("Moje okno");
        mainFrame.setBounds(200, 200, 200, 150);
        mainFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        mainFrame.setLayout(null);

        JTextField jTextField = new JTextField("Napis");
        jTextField.setBounds(20, 20, 100, 40);

        mainFrame.add(jTextField);
        mainFrame.setVisible(true);
    }
}

```

Graficzny interfejs użytkownika - Swing

Klasa "JTextField" umożliwia wyświetlenie pola tekstowego z możliwością edycji. Ma ono jednak ograniczenie. Tekst może mieć tylko jeden wers.

Slajd 5

```

public class Start {
    public static void main(String [] args) {
        JFrame mainFrame = new JFrame("Moje okno");
        mainFrame.setBounds(200, 200, 200, 150);
        mainFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        mainFrame.setLayout(null);

        JTextArea jTextArea = new JTextArea("Napis");
        jTextArea.setBounds(20, 20, 100, 40);

        mainFrame.add(jTextArea);
        mainFrame.setVisible(true);
    }
}

```

Graficzny interfejs użytkownika - Swing

"JTextArea" umożliwia wyświetlenie tekstu składającego się z wielu wersów. Zauważmy jednak, że tekst nie mieści się w polu.

Slajd 6

```

public class Start {
    public static void main(String [] args) {
        JFrame mainFrame = new JFrame("Moje okno");
        mainFrame.setBounds(200, 200, 200, 150);
        mainFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        mainFrame.setLayout(null);

        JTextArea jTextArea = new JTextArea("Napis");
        JScrollPane jScrollPane = new JScrollPane(jTextArea);
        jScrollPane.setBounds(10, 10, 170, 100);

        mainFrame.add(jScrollPane);
        mainFrame.setVisible(true);
    }
}

```

Graficzny interfejs użytkownika - Swing

Problem z widocznością tekstu możemy rozwiązać za pomocą "JScrollPane". Obiekt "JTextArea" może teraz zmieniać swój rozmiar dynamicznie. Przeglądanie ułatwiają paski przewijania.



Slajd 7

```

JRadioButton button1 = new JRadioButton("Czerwony");
button1.setBounds(10, 10, 100, 20);
JRadioButton button2 = new JRadioButton("Zielony");
button2.setBounds(10, 30, 100, 20);
JRadioButton button3 = new JRadioButton("Niebieski");
button3.setBounds(10, 50, 100, 20);
ButtonGroup colorButtonGroup = new ButtonGroup();
colorButtonGroup.add(button1);
colorButtonGroup.add(button2);
colorButtonGroup.add(button3);

mainFrame.add(button1);
mainFrame.add(button2);
mainFrame.add(button3);
    
```

Graficzny interfejs użytkownika - Swing

Obiekty "JRadioButton" umożliwiają wybór pojedynczej opcji. Zaznaczony może być tylko jeden element dlatego wszystkie muszą tworzyć grupę. Obiekt "ButtonGroup" kontroluje wszystkie komponenty "JRadioButton". Jeżeli nie użylibyśmy go, opisywana zasada nie zachodziłaby.

Slajd 8

```

String [] dane = {"Czerwony", "Zielony", "Niebieski"};
SpinnerModel model = new SpinnerListModel(dane);
JSpinner spinner = new JSpinner(model);
spinner.setBounds(10, 10, 100, 20);

mainFrame.add(spinner);
    
```

Graficzny interfejs użytkownika - Swing

Obiekt typu "JSpinner" również umożliwia nam wybór tylko jednej opcji, ale w nieco inny sposób. Zwróćmy uwagę na sposób, w jaki wstawiamy dane do obiektu. Korzystamy z modelu. Wynika to z zastosowania Wzorca projektowego MVC przez twórców Swingu. Wzorec ten rozdziela warstwę widoku od warstwy danych.

Slajd 9

```

JComboBox<String> jComboBox = new JComboBox<String>();
jComboBox.addItem("Czerwony");
jComboBox.addItem("Zielony");
jComboBox.addItem("Niebieski");
jComboBox.setBounds(10, 10, 100, 20);

mainFrame.add(jComboBox);
    
```

Graficzny interfejs użytkownika - Swing

"JComboBox", też pozwala wybrać tylko jedną opcję. Jest on typem generycznym. W nawiasach "<>" podajemy typ obiektu jaki będzie przechowywany w "JComboBox".



Slajd 10

```

JCheckBox jCheckBox1 = new JCheckBox("Zgadzasz się?");
jCheckBox1.setBounds(10, 10, 150, 20);
JCheckBox jCheckBox2 = new JCheckBox("Wysłać e-maila?");
jCheckBox2.setBounds(10, 30, 150, 20);
JCheckBox jCheckBox3 = new JCheckBox("Pokażać ponownie?");
jCheckBox3.setBounds(10, 50, 150, 20);

mainFrame.add(jCheckBox1);
mainFrame.add(jCheckBox2);
mainFrame.add(jCheckBox3);

```

Graficzny interfejs użytkownika - Swing

"JCheckBox" umożliwia wybór kilku opcji. Elementy te są całkowicie niezależne. Nie dodajemy ich do żadnej grupy, tak jak to było w przypadku "JRadioButton".

Slajd 11

```

String[] selections = { "kot", "pies", "ryba", "koń", "ptak", "wilk" };
JList<String> list = new JList<String>(selections);
JScrollPane jScrollPane = new JScrollPane(list);
jScrollPane.setBounds(10, 10, 100, 100);
mainFrame.add(jScrollPane);

```

Graficzny interfejs użytkownika - Swing

"JList" jest również typem generycznym. Pozwala on wyświetlić dane w bardziej czytelny sposób. Można ustawić możliwość zaznaczania pojedynczego lub wielokrotnego.

Slajd 12

```

String rowData[][] = { { "1", "Paweł" }, { "2", "Tomek" }, { "3", "Filip" }, { "4", "Łukasz" }, { "5", "Michał" } };
String columnNames[] = { "Lp.", "Imię" };
JTable table = new JTable(rowData, columnNames);
JScrollPane jScrollPane = new JScrollPane(table);
jScrollPane.setBounds(10, 10, 150, 100);
mainFrame.add(jScrollPane);

```

Graficzny interfejs użytkownika - Swing

"JTable" działa podobnie jak "JList", z tą różnicą, że umożliwia wyświetlenie dodatkowych kolumn.



Slajd 13

Ćwiczenie:  
Zbuduj formularz do wpisania adresu zamieszkania.

Imię:   
Nazwisko:   
Ulica:   
Numer domu:   
Numer Lokalu:   
Kod pocztowy:  -   
Miasto:

Graficzny interfejs użytkownika - Swing

Slajd 14

Ćwiczenie:  
Stwórz plik o zawartości:

Jan  
Kowalski  
Wiejska  
12  
4  
12-345  
Warszawa

Plik może znajdować się w dowolnej lokalizacji.

Graficzny interfejs użytkownika - Swing



Slajd 15

Ćwiczenie:  
Odczytaj dane z pliku i wprowadź do stworzonego formularza.  
Aby ułatwić sobie odszukiwanie pliku skorzystaj z gotowego okna dialogowego

```
JFileChooser chooser = new JFileChooser();  
chooser.showOpenDialog(null);  
File curFile = chooser.getSelectedFile();
```

Graficzny interfejs użytkownika - Swing

## Slajd 16




Ćwiczenie:

Spróbuj zmienić obraz okna z formularzem. Czy komponenty zawsze są widoczne?



Wywołaj metodę:

```
mainFrame.setResizable(false);
```


Sprawdź ponownie zachowanie okna.

 Graficzny interfejs użytkownika - Swing

## Slajd 17

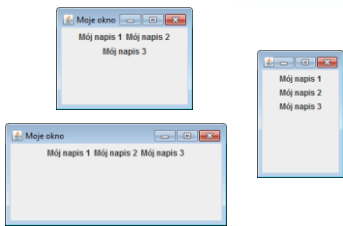





```
public class Start {  
    public static void main(String[] args) {  
        JFrame mainFrame = new JFrame("Moje okno");  
        mainFrame.setBounds(200, 200, 200, 150);  
        mainFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        mainFrame.setLayout(new FlowLayout());  
  
        JLabel label1 = new JLabel("Mój napis 1");  
        mainFrame.add(label1);  
        JLabel label2 = new JLabel("Mój napis 2");  
        mainFrame.add(label2);  
        JLabel label3 = new JLabel("Mój napis 3");  
        mainFrame.add(label3);  
  
        mainFrame.setVisible(true);  
    }  
}
```

 Graficzny interfejs użytkownika - Swing

We wcześniejszych przykładach do metody "setLayout" wstawialiśmy obiekt "null". Teraz skorzystamy z zarządcy rozkładu o nazwie "FlowLayout". Nie musimy już układać każdego elementu osobno. Co więcej, nawet gdy użyjemy metody "setBounds", nie uzyskamy oczekiwanego efektu. "FlowLayout" decyduje o położeniu komponentów.

## Slajd 18



 Graficzny interfejs użytkownika - Swing

Możemy uruchomić poprzedni przykład i zobaczyć jak "FlowLayout" zarządza widokiem.

Slajd 19

```

JLabel label1 = new JLabel("Mój napis 1");
mainFrame.add(label1);
JLabel label2 = new JLabel("Mój napis 2");
mainFrame.add(label2);
JLabel label3 = new JLabel("Mój napis 3");
mainFrame.add(label3);

JPanel jPanel = new JPanel();
jPanel.setBackground(Color.YELLOW);
JLabel label4 = new JLabel("Mój napis 4");
jPanel.add(label4);
JLabel label5 = new JLabel("Mój napis 5");
jPanel.add(label5);

mainFrame.add(jPanel);
mainFrame.setVisible(true);
    
```

Graficzny interfejs użytkownika - Swing

Korzystając z klasy "JPanel" możemy grupować elementy. "Mój napis 4" i "Mój napis 5" będą znajdować się wewnątrz panelu.

Slajd 20

Graficzny interfejs użytkownika - Swing

Możemy zaobserwować, że przemieszczany jest cały panel. Dzieje się tak, ponieważ "JPanel" ma własnego zarządcę rozkładu i ustawiony "FlowLayout" nie zarządza komponentami z żółtego pola.

Slajd 21

```

public class Start {
    public static void main(String [] args) {
        JFrame mainFrame = new JFrame("Moje okno");
        mainFrame.setBounds(200, 200, 200, 150);
        mainFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        mainFrame.setLayout(new BorderLayout());

        JLabel label1 = new JLabel("Mój napis 1");
        mainFrame.add(label1, BorderLayout.NORTH);
        JLabel label2 = new JLabel("Mój napis 2");
        mainFrame.add(label2, BorderLayout.WEST);
        JLabel label3 = new JLabel("Mój napis 3");
        mainFrame.add(label3, BorderLayout.EAST);
        JLabel label4 = new JLabel("Mój napis 4");
        mainFrame.add(label4, BorderLayout.SOUTH);
        JLabel label5 = new JLabel("Mój napis 5");
        mainFrame.add(label5, BorderLayout.CENTER);

        mainFrame.setVisible(true);
    }
}
    
```

Graficzny interfejs użytkownika - Swing

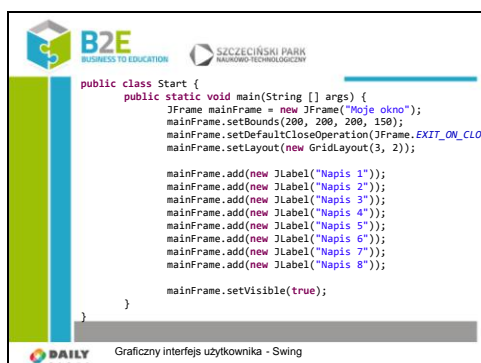
Kolejnym przykładem jest "BorderLayout". Jeżeli nie wywołamy metody "setLayout", jest on domyslnym zarządcą. Dodając każdy element podajemy dodatkowo jeden z parametrów: "NORTH", "SOUTH", "WEST", "EAST" i "CENTER".

Slajd 22



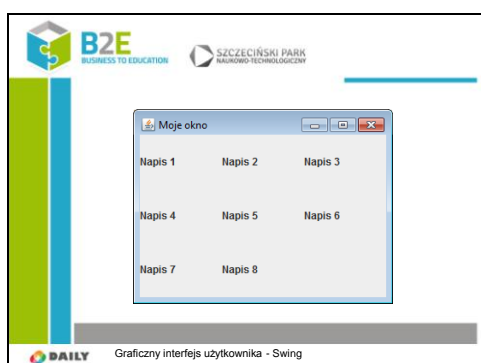
Zmieniając rozmiar okna możemy zaobserwować gdzie poszczególne obszary się znajdują.

Slajd 23



"GridLayout" układa komponenty podobnie jak tabela. Tworząc go należy podać liczbę kolumn i wiersów.

Slajd 24





Slajd 25

Ćwiczenie:



Popraw formularz tak, aby dostosowywał się do rozmiaru okna. Możesz określić minimalny rozmiar okna:

```
mainFrame.setMinimumSize(new Dimension(200, 150));
```


W ten sposób będziemy mieli pewność, że użytkownik nie zmniejszy okna.

 Graficzny interfejs użytkownika - Swing

Slajd 26



```
public class DiagonalLayout implements LayoutManager {
    @Override
    public void addLayoutComponent(String name, Component comp) {}
    @Override
    public void layoutContainer(Container parent) {
        Component[] tab = parent.getComponents();
        Dimension dim = parent.getSize();
        int h = dim.height / (tab.length + 6);
        int w = dim.width / (tab.length + 2);
        for (int i=0; i<tab.length; i++) {
            tab[i].setBounds(tab[i].getWidth(), tab[i].getHeight(),
                             w*(i+1), h*(i+1));
        }
    }
    @Override
    public Dimension minimumLayoutSize(Container parent) {
        return null;
    }
    @Override
    public Dimension preferredLayoutSize(Container parent) {
        return null;
    }
    @Override
    public void removeLayoutComponent(Component comp) {}
}
```

 Graficzny interfejs użytkownika - Swing

Swing umożliwia nam napisanie własnego zarządcy rozkładu. Wystarczy zaimplementować "LayoutManager". Zaimplementujemy przynajmniej metodę "layoutContainer". Odpowiada ona za ponowne rozmieszczenie komponentów po zmianie rozmiaru okna.

Powyższy przykład układa elementy wzdłuż przekątnej okna.


Slajd 27

```
public class Start {
    public static void main(String[] args) {
        JFrame mainFrame = new JFrame("Moje okno");
        mainFrame.setBounds(200, 200, 200, 150);
        mainFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        mainFrame.setLayout(new DiagonalLayout());

        mainFrame.add(new JLabel("Napis 1"));
        mainFrame.add(new JLabel("Napis 2"));
        mainFrame.add(new JLabel("Napis 3"));
        mainFrame.add(new JLabel("Napis 4"));
        mainFrame.add(new JLabel("Napis 5"));
        mainFrame.add(new JLabel("Napis 6"));
        mainFrame.add(new JLabel("Napis 7"));
        mainFrame.add(new JLabel("Napis 8"));

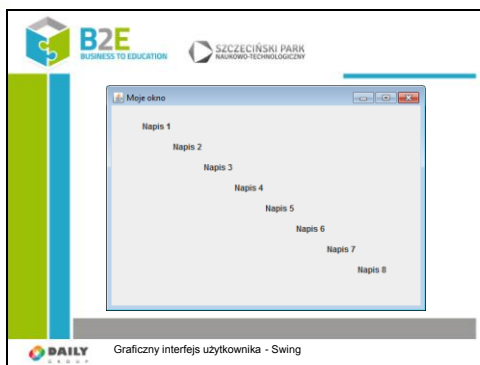
        mainFrame.setMinimumSize(new Dimension(200, 150));
        mainFrame.setVisible(true);
    }
}
```

 Graficzny interfejs użytkownika - Swing

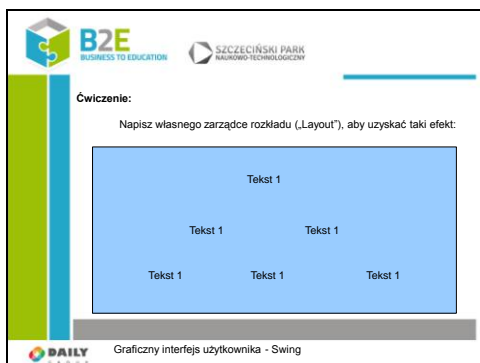




Slajd 28



Slajd 29



### 8.9.3 Opis złożonych osiągnięć ucznia

Po tej lekcji uczniowie będą potrafili stworzyć proste okno ze skonfigurowanymi komponentami. Obsługą zmiany rozmiaru okna.

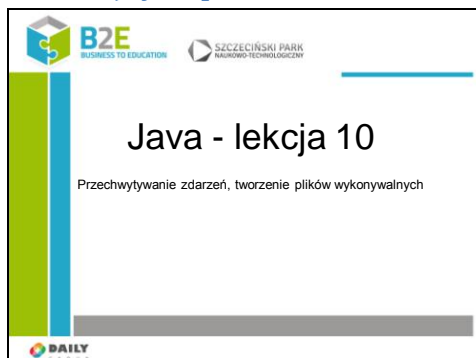
## 8.10 Lekcja 10 - Przechwytywanie zdarzeń, tworzenie plików wykonywalnych

### 8.10.1 Cel lekcji

Celem lekcji jest pokazanie jak prawidłowo obsługiwać zdarzenia takie jak: ruchy kursora, starowania klawiatura. Omówiona zostanie budowa plików wykonywalnych jar.

## 8.10.2 Treść - slajdy z opisem

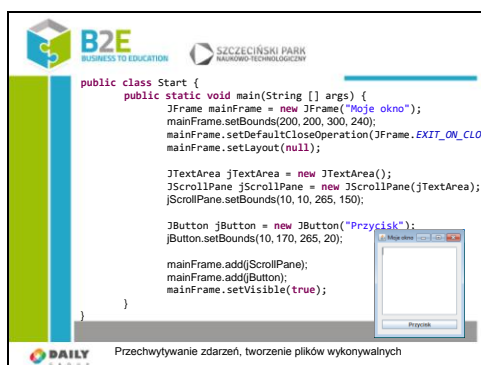
### Slajd 1



Java - lekcja 10

Przechwytywanie zdarzeń, tworzenie plików wykonywalnych

### Slajd 2



```
public class Start {
    public static void main(String [] args) {
        JFrame mainFrame = new JFrame("Moje okno");
        mainFrame.setBounds(200, 200, 300, 240);
        mainFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        mainFrame.setLayout(null);

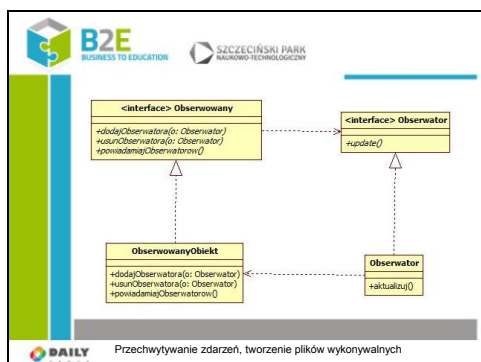
        JTextArea jTextArea = new JTextArea();
        JScrollPane jScrollPane = new JScrollPane(jTextArea);
        jScrollPane.setBounds(10, 10, 265, 150);

        JButton jButton = new JButton("Przycisk");
        jButton.setBounds(10, 170, 265, 20);

        mainFrame.add(jScrollPane);
        mainFrame.add(jButton);
        mainFrame.setVisible(true);
    }
}
```

Przechwytywanie zdarzeń, tworzenie plików wykonywalnych

### Slajd 3



W programowaniu obiektowym obiekty posiadają pewien stan, tj. zbiór aktualnych wartości pól obiektu, który w wyniku wykonywania na nich operacji może ulegać zmianie. Od bieżącego stanu mogą być zależne inne obiekty, dlatego musi istnieć możliwość ich powiadomienia o jego zmianie tak, aby mogły one się do niej dostosować. Możemy także żądać, aby inne obiekty były powiadamiane o tym, że inny obiekt próbuje wykonać konkretną czynność, np. ponownie nawiązywać utracone połączenie z bazą danych. Pragniemy zaimplementować ogólny mechanizm, który umożliwi nam osiągnięcie tych celów.

We wzorcu obserwator wyróżniamy dwa podstawowe typy obiektów:



obserwowany (ang. observable, subject) - obiekt, o którym chcemy uzyskiwać informacje,

obserwator (ang. observer, listener) - obiekty oczekujące na powiadomienie o zmianie stanu obiektu obserwowanego.

Kiedy stan obiektu obserwowanego się zmienia, wywołuje on metodę "powiadomObserwatorów()", która wysyła powiadomienia do wszystkich zarejestrowanych obserwatorów.

Obserwator jest stosowany w aplikacjach z graficznym interfejsem użytkownika.

Slajd 4

 **B2E**  
BUSINESS TO EDUCATION  **SZCZECIŃSKI PARK**  
NAUKOWO-TECHNOLOGICZNY

```
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import javax.swing.JTextArea;



public class MyListener implements MouseListener {
    private JTextArea jTextArea;
    public MyListener(JTextArea jTextArea) {
        this.jTextArea = jTextArea;
    }
    @Override
    public void mouseClicked(MouseEvent arg0) {
        jTextArea.setText(jTextArea.getText()
            + "\nnastąpiło kliknięcie");
    }
    @Override
    public void mouseEntered(MouseEvent arg0) {
        jTextArea.setText(jTextArea.getText()
            + "\nkursor wszedł w obszar przycisku");
    }
}
```



Przechwytywanie zdarzeń, tworzenie plików wykonywalnych




Slajd 5



```
@Override
public void mouseExited(MouseEvent arg0) {
    JTextArea.setText(jTextArea.getText()
        + "\nkursor wyszedł z obszaru przycisku");


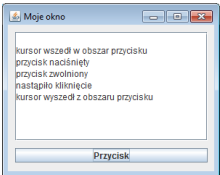


@Override
public void mousePressed(MouseEvent arg0) {
    JTextArea.setText(jTextArea.getText()
        + "\nprzycisk naćnięty");

@Override
public void mouseReleased(MouseEvent arg0) {
    JTextArea.setText(jTextArea.getText()
        + "\nprzycisk zwolniony");
}
```





Przechwytywanie zdarzeń, tworzenie plików wykonywalnych

Slajd 6




Przechwytywanie zdarzeń, tworzenie plików wykonywalnych

Slajd 7



**Ćwiczenie:**

Zaimplementuj interfejs „KeyListener” i dodaj go jako słuchacza do przycisku



Przechwytywanie zdarzeń, tworzenie plików wykonywalnych



Slajd 8

Ćwiczenie:

Stwórz okno z komponentem „JLabel”. Niech komponent porusza się po oknie. Jeżeli zmienisz pozycję elementu, wywołaj metodę „repaint”.

Przechwytywanie zdarzeń, tworzenie plików wykonywalnych

Slajd 9

```
public class Start {  
    public class MyInnerClass{  
    }  
  
    public static void main(String [] args) {  
        JFrame mainFrame = new JFrame("Moje okno");  
        mainFrame.setBounds(200, 200, 300, 240);  
        mainFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    }  
}
```

Przechwytywanie zdarzeń, tworzenie plików wykonywalnych

Do przykładu z początku lekcji dodaj klasę wewnętrzną i zapisz zmiany. Dla pewności uruchom program.

Slajd 10

Project Structure:

- .settings
- bin
- src
- .classpath
- .project

Files:

- MyKeyListener.java
- MyListener.java
- Start.java
- MyKeyListener.class
- MyListener.class
- StartMyInnerClass.class
- Start.class

Przechwytywanie zdarzeń, tworzenie plików wykonywalnych

W projekcie znajdują się trzy pliki źródłowe:

- MyKeyListener.java,
- MyListener.java,
- Start.java.

Wejdź do folderu workspace, następnie do folderu z tym projektem. W środku znajdują się dwa foldery "src" i "bin".

Sprawdź ich zawartość. W folderze "src" znajdują się pliki źródłowe i są trzy.

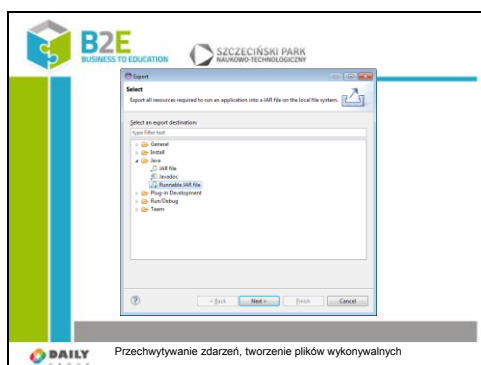
Dlaczego w folderze są cztery pliki?

Na jednej z pierwszych lekcji powiedziane było, że w jednym pliku znajdować się może tylko jedna klasa.

W plikach źródłowych mogą być klasy wewnętrzne, ale po procesie kompilacji każda klasa stanowi osobny plik.

W klasie "Start" jest klasa wewnętrzna "MyInnerClass". Po skompilowaniu jest osobnym plikiem. Po nazwie można łatwo rozpoznać, z której klasy została wyodrębniona.

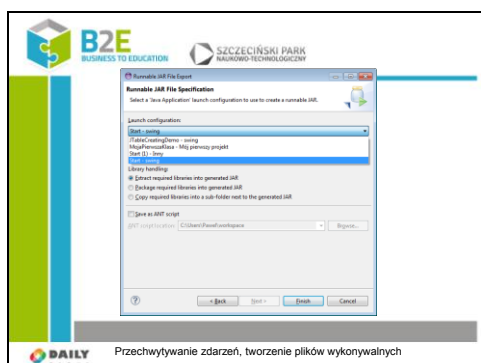
Slajd 11



W środowisku Eclipse kliknij prawym przyciskiem na projekt i użyj opcji eksport.

Wybierz opcję "Java/Runnable JAR file".

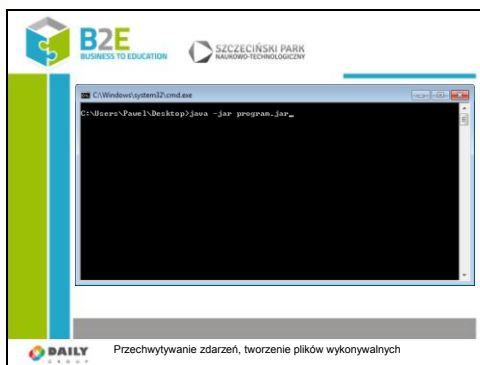
Slajd 12



Wybierz plik, który służy do uruchamiania aplikacji. Następnie podaj nazwę i lokalizację pliku.



Slajd 13



Uruchom konsolę i wywołaj polecenie tak, jak na slajdzie.

Powinno nastąpić prawidłowe uruchomienie programu.

Zmień rozszerzenie pliku z "jar" na "zip" i rozpakuj go.

Zauważ, że jest to zwykłe archiwum zip. Znajdują się w nim wszystkie pliki bajtkodu, należące do danego projektu.

W folderze "META-INF" znajduje się plik manifestu. Zobacz jego zawartość.

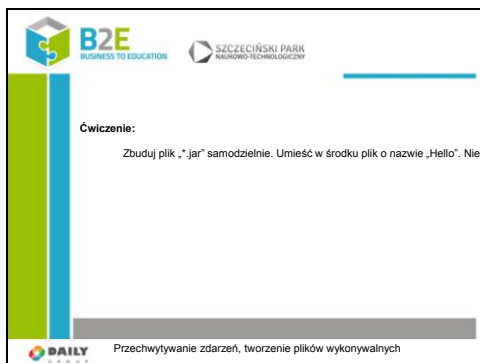
Manifest-Version: 1.0

Class-Path: .

Main-Class: Start

W ostatnim wersie jest nazwa klasy, z której wywoływany jest program.

Slajd 14



### 8.10.3 Opis założonych osiągnięć ucznia

Po tej lekcji uczniowie będą umieli obsłużyć zdarzenia systemowe związane z kursorem i klawiaturą. Będą rozumieć jak wygląda tworzenie plików wykonywalnych.