O'REILLY®

Wydanie III

Head First

Java

Rusz głową!

Przewodnik
po praktycznym
programowaniu w Javie

Kathy Sierra
Bert Bates
Trisha Gee

Helion

Tytuł oryginału: Head First Java: A Brain-Friendly Guide, 3rd Edition Tłumaczenie: Piotr Rajca
ISBN: 978-83-283-9984-6

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
https://helion.pl/user/opinie/javrg3
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice tel. 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *https://helion.pl* (księgarnia internetowa, katalog książek)

Printed in Poland.

Spis treści (podsumowanie)

	Wprowadzenie	xxiii
1	Przełamując zalew początkowych trudności. *Szybki skok na głęboką wodę*	1
2	Wycieczka do Obiektowa. *Klasy i obiekty*	27
3	Poznaj swoje zmienne. *Typy podstawowe i odwołania*	49
4	Jak działają obiekty. *Metody wykorzystują zmienne instancyjne*	71
5	Supermocne metody. *Pisanie programu*	95
6	Korzystanie z biblioteki Javy. *Poznaj Java API*	123
7	Wygodniejsze życie w Obiektowie. *Dziedziczenie i polimorfizm*	163
8	Poważny polimorfizm. *Interfejsy i klasy abstrakcyjne*	195
9	Życie i śmierć obiektu. *Konstruktory i odśmiecacz*	233
10	Liczby mają znaczenie. *Liczby oraz składowe statyczne*	271
11	Struktury danych. *Kolekcje i typy ogólne*	305
12	Co, a nie jak. *Wyrażenia lambda i strumienie*	365
13	Ryzykowne zachowanie. *Obsługa wyjątków*	417
14	Historia bardzo graficzna. *Tworzenie graficznego interfejsu użytkownika*	455
15	Popracuj nad Swingiem. *Stosowanie biblioteki Swing*	503
16	Zapisywanie obiektów (i tekstu). *Serializacja i operacje wejścia-wyjścia na plikach*	531
17	Nawiąż połączenie. *Zagadnienia sieciowe i wątki*	579
18	Rozwiązywanie problemów współbieżności. *Warunki współzawodnictwa i dane niemodyfikowalne*	631
A	Dodatek A. *Ostatnie doprawianie kodu*	665
B	Dodatek B. *Jedenaście najważniejszych tematów, które nie zmieściły się w głównej części książki*	675
	Skorowidz	693

Spis treści (z prawdziwego zdarzenia)

Wprowadzenie

Twój mózg myśli o Javie. W tym rozdziale *Ty* próbujesz się czegoś *dowiedzieć*, natomiast Twój *mózg* wyświadcza Ci uprzejmość i stara się, aby te informacje nie zostały *zapamiętane na długo*. Myśli sobie: „Lepiej zostawić miejsce na ważniejsze rzeczy, takie jak dzikie zwierzęta, których należy się wystrzegać, lub rozważania, czy jeżdżenie na snowboardzie w stroju Adama to dobry pomysł". Jak zatem można oszukać własny mózg i przekonać go, że od znajomości Javy zależy nasze życie?

Dla kogo jest przeznaczona ta książka?	xxiv
Wiemy, co sobie myślisz	xxv
Metapoznanie — myślenie o myśleniu	xxvii
Oto co zrobiliśmy	xxviii
Oto co możesz zrobić, aby zmusić swój mózg do posłuszeństwa	xxix
Czego potrzebujesz, aby skorzystać z tej książki?	xxx
Kilka ostatnich spraw, o których musisz wiedzieć	xxxi

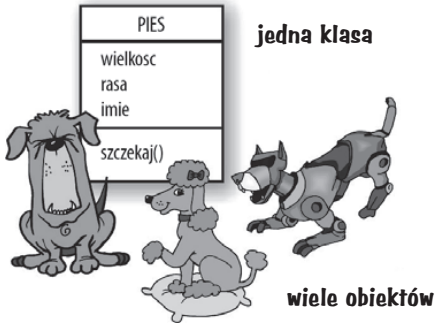
1 Przełamując zalew początkowych trudności

Java zabiera nas w nowe miejsca. Od momentu pojawienia się pierwszej, skromnej wersji o numerze 1.02 Java pociągała programistów ze względu na przyjazną składnię, cechy obiektowe, zarządzanie pamięcią, a przede wszystkim obietnicę przenośności. Wskoczymy od razu na głęboką wodę: napiszemy prosty program w Javie, skompilujemy go i uruchomimy. Pokażemy składnię Javy, instrukcje warunkowe, pętle i inne rzeczy, które sprawiają, że Java jest super. A więc do dzieła!

Jak działa Java?	2
Co będziesz robić w Javie?	3
Krótka historia Javy	4
Struktura kodu w Javie	7
Tworzenie klasy z metodą main()	9
Pętle i pętle, i...	13
Rozgałęzienia warunkowe	15
Tworzenie poważnej aplikacji biznesowej	16
Program krasomówczy	19
Ćwiczenia	20
Rozwiązania ćwiczeń	24

2 Wycieczka do Obiektowa

Mówiono mi, że będą obiekty. W rozdziale 1. cały tworzony kod był umieszczany w metodzie `main()`. Nie jest to poprawne rozwiązanie obiektowe. Teraz musimy więc zostawić świat programowania proceduralnego i zacząć tworzyć własne obiekty. Przyjrzymy się czynnikom, które sprawiają, że programowanie zorientowane obiektowo w języku Java jest takie fajne. Przedstawimy różnice pomiędzy klasą a obiektem. Przekonamy się, w jaki sposób obiekty mogą ułatwić nam życie.



Wojna o fotel	28
Tworzenie pierwszego obiektu	36
Tworzenie i testowanie obiektów Film	37
Szybko! Opuszczamy metodę main()!	38
Uruchamianie zgadywanki	40
Ćwiczenia	42
Rozwiązania ćwiczeń	46

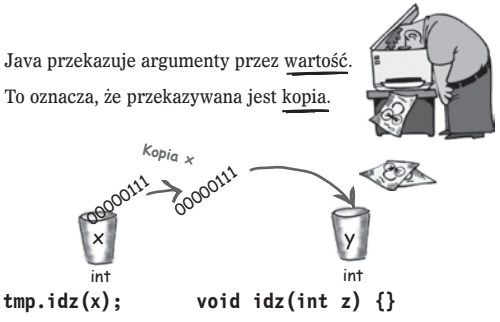
3 Poznaj swoje zmienne

Zmienne można podzielić na dwie kategorie: zmienne typów podstawowych oraz odwołania. Ale przecież **musi być coś bardziej interesującego** niż liczby całkowite, łańcuchy znaków i tablice. Co należy zrobić, gdybyśmy chcieli stworzyć obiekt `WłaścicielZwierzaka` ze zmienną instancyjną `Pies`? Albo obiekt `Samochód` ze zmienną instancyjną `Silnik`? W tym rozdziale wyjaśnimy tajemnice typów danych w Javie i przyjrzymy się, co można *zadeklarować* jako zmienną, co w takiej zmiennej można *zapisać* i do czego jej można *użyć*. W końcu zobaczymy także, jak *naprawdę* wygląda życie na automatycznie odśmiecanej stercie.

Deklarowanie zmiennej	50
„Proszę podwójną. Albo nie — całkowitą!”	51
Trzymaj się z daleka od tego słowa kluczowego!	53
Kontrolowanie obiektu Pies	54
Odwołanie do obiektu to jedynie inna wartość zmiennej	55
Życie na odśmiecanej stercie	57
Tablica jest jakby tacą z kubkami	59
Przykładowy obiekt Pies	62
Ćwiczenia	63
Rozwiązania ćwiczeń	68

4 Jak działają obiekty?

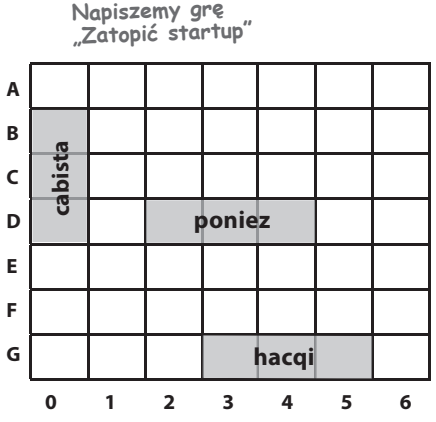
Stan wpływa na działanie, a działanie wpływa na stan. Wiemy już, że obiekty mają **stan** oraz **działanie**, które to aspekty obiektów są odpowiednio reprezentowane przez **zmienne instancyjne** oraz **metody**. Teraz zajmiemy się zagadnieniem, w jaki sposób stan oraz działanie obiektu są ze sobą *powiązane*. Zachowania obiektu korzystają z jego unikalnego stanu. Innymi słowy, ***metody wykorzystują wartości zmiennych instancyjnych***. Na przykład „Jeśli pies waży mniej niż 7 kilogramów, szczekaj piskliwie, w przeciwnym razie...” bądź też: „Powiększ wagę o 2 kilogramy”. ***A zatem pozmieniajmy stan.***



Pamiętaj! Klasa opisuje to, co obiekt wie, oraz to, co robi	72
Wielkość ma wpływ na sposób szczekania	73
Do metod można przekazywać informacje	74
Metoda może coś zwrócić	75
Do metody można przekazać więcej niż jedną informację	76
Ciekawe rozwiązania wykorzystujące parametry i wartości wynikowe	79
Hermetyzacja	80
Jak zachowują się obiekty w tablicy?	83
Deklarowanie i inicjalizacja zmiennych instancyjnych	84
Porównywanie zmiennych (typów podstawowych oraz odwołań)	86
Ćwiczenia	88
Rozwiązania ćwiczeń	93

5 Supermocne metody

Dodajmy naszym metodom nieco siły. Igraliśmy ze zmiennymi, zabawialiśmy się z kilkoma obiektami i napisaliśmy parę wierszy kodu. Ale potrzeba nam więcej narzędzi. Takich jak **operatory**. Oraz **pętle**. Może się nam przydać umiejętność **generowania liczb losowych**. Albo **zamieniania łańcucha znaków na liczbę**, ech... to by było świetne. A czy nie można by się nauczyć tego wszystkiego, *pisząc* jakiś program? Tak żeby zobaczyć, jak wygląda pisanie i testowanie normalnego programu od samego początku. **Na przykład jakąś grę**... taką jak gra w okręty.



Napiszmy grę przypominającą okręty, o nazwie „Zatopić startup"	96
Tworzenie klasy	99
Pisanie implementacji metod	101
Pisanie kodu testowego dla klasy ProstyStartup	102
Metoda sprawdz()	104
Kod przygotowawczy klasy ProstyStartupGra	108
Metoda main() gry	110
Zagrajmy	113
Trochę więcej o pętlach for	114
Rozszerzone pętle for	116
Rzutowanie wartości typów podstawowych	117
Ćwiczenia	118
Rozwiązania ćwiczeń	121

6 Korzystanie z biblioteki Javy

Java jest wyposażona w setki gotowych do użycia klas. Jeśli tylko potrafisz znaleźć w bibliotece Javy, nazywanej **Java API**, to, czego Ci potrzeba, to nie będziesz musiał ponownie wymyślać koła. *W końcu masz ciekawsze rzeczy do roboty.* Jeśli musisz napisać kod, to równie dobrze możesz napisać wyłącznie te jego fragmenty, którą są unikalne dla tworzonej aplikacji. Java API to cała masa klas, które tylko czekają, byś zaczął ich używać jako elementów konstrukcyjnych w swoich programach.

„Dobrze wiedzieć, że w pakiecie java.util istnieje klasa ArrayList. Ale jak mogłabym się o tym ***sama*** *dowiedzieć?"*

— Julia, lat 31, modelka

Ostatni rozdział zakończył się w dramatycznych okolicznościach — w programie znaleźliśmy błąd	124
Obudź się i przejrzyj bibliotekę	130
Niektóre możliwości klasy ArrayList	131
Porównanie klasy ArrayList ze zwyczajną tablicą	135
Napiszmy WŁAŚCIWĄ wersję gry „Zatopić startup"	138
Kod przygotowawczy właściwej klasy StartupGraMax	142
Ostateczna wersja klasy Startup	148
Wyrażenia logiczne o bardzo dużych możliwościach	151
Stosowanie biblioteki (Java API)	152
Ćwiczenia	160
Rozwiązania ćwiczeń	161

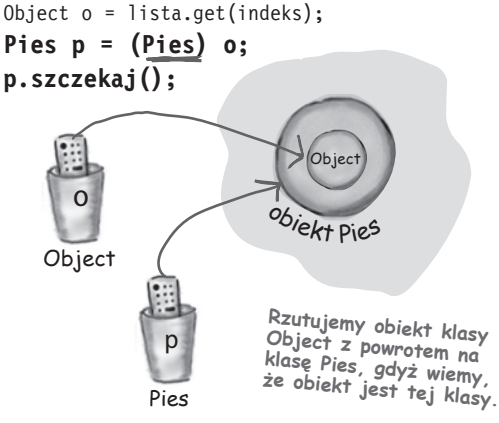
7 Wygodniejsze życie w Obiektowie

Planuj swoje programy, myśląc perspektywicznie. Co by było, gdybyś mógł tworzyć swój kod w Javie w taki sposób, by ktoś *inny* mógł go **łatwo** rozszerzać? I gdybyś mógł pisać kod bardzo elastyczny, pozwalający na wprowadzanie tych denerwujących zmian specyfikacji zgłaszanych w ostatniej chwili. Gdy zdobędziesz Plan Polimorfizmu, poznasz pięć kroków do lepszego projektowania klas, trzy sztuczki polimorficzne i osiem sposobów tworzenia elastycznego kodu, a jeśli zadzwonisz już teraz — otrzymasz dodatkową lekcję na temat czterech sposobów stosowania dziedziczenia.

Wojna o fotel raz jeszcze… 164
Zrozumienie dziedziczenia 166
Zaprojektujmy drzewo dziedziczenia dla programu symulacji zwierząt 168
Poszukiwanie dalszych możliwości zastosowania dziedziczenia 171
Relacje JEST oraz MA 175
Jak możesz określić, czy dobrze zaprojektowałeś hierarchię dziedziczenia? 177
Czy korzystanie z dziedziczenia przy projektowaniu klas jest „używaniem", czy „nadużywaniem"? 179
Jak dotrzymać kontraktu — reguły przesłaniania 188
Przeciążanie metody 189
Ćwiczenia 190
Rozwiązania ćwiczeń 193

8 Poważny polimorfizm

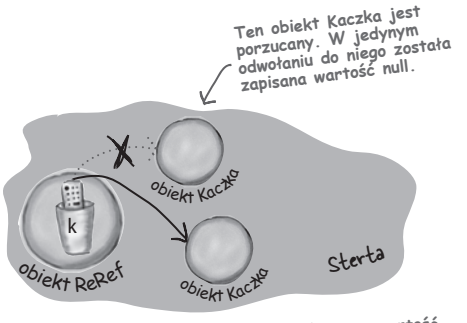
Dziedziczenie to jedynie początek. Aby w pełni wykorzystać możliwości, jakie daje polimorfizm, będziemy potrzebować interfejsów. Musimy zostawić proste dziedziczenie i pójść dalej — dotrzeć do poziomu elastyczności i możliwości rozbudowy kodu, jakie dają jedynie projektowanie i programowanie z wykorzystaniem specyfikacji interfejsów. Jednak czym jest interfejs? Otóż interfejs to całkowicie — w stu procentach — abstrakcyjna klasa. A co to jest abstrakcyjna klasa? To klasa, która nie daje możliwości tworzenia obiektów. A do czego taka klasa może nam się przydać? O tym dowiesz się z tego rozdziału.



Czy projektując przedstawioną obok hierarchię klas, o czymś zapomnieliśmy? 196
Kompilator nie pozwoli utworzyć obiektów klasy abstrakcyjnej 199
Abstrakcyjne kontra konkretne 200
MUSISZ zaimplementować wszystkie metody abstrakcyjne 202
Polimorfizm w działaniu 204
A co z obiektami innych klas? Dlaczego nie stworzyć listy, w której można by zapisać cokolwiek? 206
Kiedy Pies nie zachowuje się jak Pies 210
Rozważmy różne możliwości projektowe 217
Tworzenie i implementowanie interfejsu ZwierzakDomowy 223
Wywoływanie wersji metody zdefiniowanej w klasie bazowej 226
Ćwiczenia 228
Rozwiązania ćwiczeń 231

9 Życie i śmierć obiektu

Obiekty się rodzą i obiekty umierają. To Ty odpowiadasz za ich cykl życia. Ty decydujesz, kiedy obiekt należy *utworzyć*. Również Ty decydujesz, kiedy obiekt należy *porzucić*. **Odśmiecacz sterty** odzyskuje pamięć. W tym rozdziale opisujemy, w jaki sposób obiekty są tworzone, gdzie są przechowywane podczas swojego istnienia oraz jak można je efektywnie zachować lub porzucić. Oznacza to, że będziemy pisać o stercie, stosie, zasięgu, konstruktorach, konstruktorach bazowych, odwołaniach pustych oraz nadawaniu się do usunięcia.



W zmiennej instancyjnej „k" jest zapisywana wartość null, co można porównać z pilotem, który nie jest zaprogramowany do obsługi żadnego urządzenia. Dopóki zmienna ta nie zostanie ponownie zaprogramowana (czyli do momentu, gdy zapiszemy w niej jakiś obiekt), nie możemy jej nawet używać wraz z operatorem kropki.

Stos i sterta. Gdzie są przechowywane dane? 234
Metody są zapisywane na stosie 235
A co ze zmiennymi lokalnymi, które są obiektami? 236
Cud utworzenia obiektu 238
Tworzenie obiektu Kaczka 240
Czy kompilator zawsze tworzy dla nas konstruktor bezargumentowy? 244
Nanoprzegląd. Cztery rzeczy o konstruktorach, które należy zapamiętać 247
Znaczenie konstruktorów klasy bazowej w życiu obiektu 249
Czy dziecko może istnieć przed rodzicami? 252
A co ze zmiennymi referencyjnymi? 258
Nie podoba mi się kierunek, w jakim to wszystko zmierza. 259
Ćwiczenia 264
Rozwiązania ćwiczeń 268

10 Liczby mają znaczenie

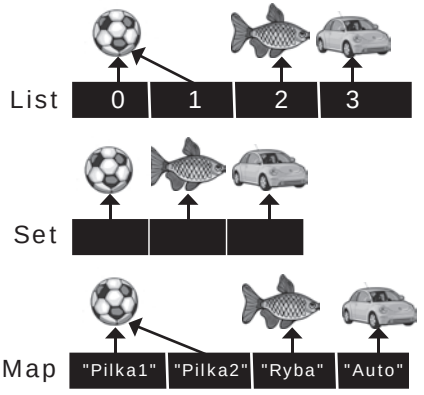
Wykonuj obliczenia matematyczne. Java API udostępnia metody pozwalające wyznaczyć wartość bezwzględną liczby, zaokrąglić ją albo znaleźć większą lub mniejszą z dwóch wartości. A co z formatowaniem? Moglibyśmy chcieć wyświetlać liczby z dokładnie dwoma liczbami dziesiętnymi lub z odstępami pomiędzy grupami cyfr. Oprócz tego może się pojawić konieczność wyświetlania dat. Albo zamiany liczb na łańcuchy znaków. Albo łańcucha na liczbę. Zaczniemy od wyjaśnienia, co dla zmiennej lub metody oznacza bycie *statyczną*.

Wszystkie obiekty tej samej klasy wspólnie **używają jednej kopii każdej ze składowych statycznych.**

Metody klasy Math — najlepsze z możliwych odpowiedników metod globalnych 272
Różnice pomiędzy metodami zwyczajnymi a statycznymi 273
Inicjalizacja zmiennych statycznych 279
Metody klasy Math 284
Reprezentowanie wartości typów podstawowych w formie obiektów 286
Automatyczna konwersja działa niemal wszędzie 288
A teraz w przeciwnym kierunku… zamiana liczby na łańcuch znaków 291
Formatowanie liczb 292
Specyfikator formatu 296
Ćwiczenia 302
Rozwiązania ćwiczeń 304

11 Struktury danych

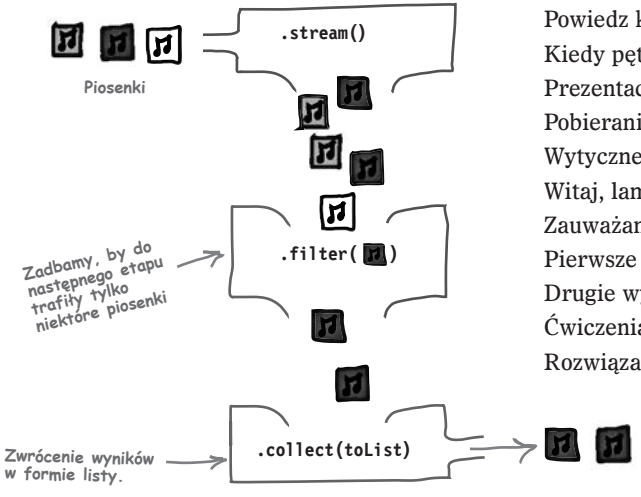
Przechowywanie w Javie nie nastręcza żadnych trudności. Dysponujesz wszystkimi narzędziami związanymi z korzystaniem z kolekcji oraz sortowaniem danych i nie musisz pisać żadnych własnych algorytmów. Biblioteka kolekcji Javy udostępnia struktury danych, które powinny zaspokoić praktycznie wszelkie Twoje potrzeby. Chciałbyś mieć listę, do której będziesz mógł bez problemów dodawać nowe elementy? Chcesz znaleźć coś na podstawie nazwy? Chcesz stworzyć listę, która będzie automatycznie eliminować powtarzające się elementy? Chcesz posortować listę współpracowników według liczby określającej, ile razy znienacka uderzyli Cię w plecy?



Poznajemy API pakietu java.util, List i Collections 310
Typy ogólne oznaczają większe bezpieczeństwo typów 320
Ponownie odwiedzimy metodę sort() 323
Nowa, poprawiona, porównywalna klasa Piosenka 326
Sortowanie z wykorzystaniem wyłącznie komparatorów 332
Zastosowanie wyrażeń lambda w kodzie aplikacji szafy grającej 338
Wykorzystanie kolekcji HashSet zamiast ArrayList 343
Co MUSIMY wiedzieć o klasie TreeSet... 349
Poznaliśmy już listy i zbiory, teraz poznamy mapy 351
W końcu wracamy do typów ogólnych 354
Rozwiązania ćwiczeń 360

12 Wyrażenia lambda i strumienie: co, a nie jak

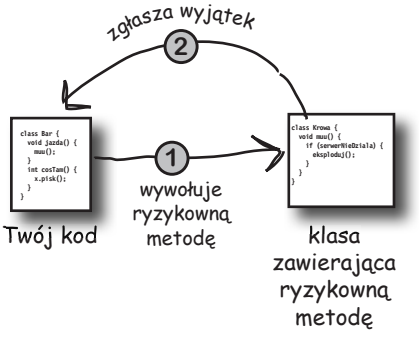
A co by było, gdybyś... nie musiał mówić komputerowi, jak coś zrobić? W tym rozdziale zajmiemy się *API strumieni*. Przekonasz się w nim także, jak przydatne mogą być wyrażenia lambda w połączeniu ze strumieniami, i dowiesz się, jak używać API strumieni do wyszukiwania i przekształcania danych w kolekcjach.



Powiedz komputerowi, CZEGO chcesz 366
Kiedy pętle for schodzą na złą drogę 368
Prezentacja API strumieni 371
Pobieranie wyników ze strumienia 374
Wytyczne dotyczące stosowania strumieni 380
Witaj, lambdo, nasz (niezbyt) stary przyjacielu 384
Zauważanie interfejsów funkcyjnych 392
Pierwsze wyzwanie Leona: znaleźć wszystkie rockowe utwory 396
Drugie wyzwanie Leona: lista wszystkich gatunków 400
Ćwiczenia 411
Rozwiązania ćwiczeń 413

13 Ryzykowne zachowanie

Różne rzeczy się zdarzają. Pliku nie ma tam, gdzie powinien być. Serwer został wyłączony. Niezależnie od tego, jak dobrym jesteś programistą, nie jesteś w stanie kontrolować *wszystkiego*. Pisząc „ryzykowną" metodę, będziesz potrzebował kodu, który „poradzi" sobie w sytuacji, gdy zdarzy się coś złego. Jednak skąd wiadomo, że metoda jest „ryzykowna"? I gdzie należy umieścić kod przeznaczony do *obsługi* tej ***wyjątkowej*** sytuacji? W *tym* rozdziale napiszemy odtwarzacz muzyki MIDI, korzystając z „ryzykownego" JavaSound API, dlatego lepiej będzie, jeśli dowiemy się, jak obsługiwać taki potencjalnie niebezpieczny kod.



Stwórzmy program MuzMachina	418
Przede wszystkim potrzebujemy sekwensera	420
Wyjątek jest obiektem… klasy Exception	424
Sterowanie przepływem w blokach try-catch	428
Czy wspominaliśmy, że metoda może zgłaszać więcej niż jeden wyjątek?	431
W przypadku użycia wielu bloków catch należy je uporządkować ze względu na zakres — od najmniejszego do największego	434
Zrezygnowanie z obsługi wyjątku (poprzez jego zadeklarowanie) jedynie odsuwa w czasie to, co nieuniknione	438
Kod od kuchni	441
Wersja 1. Twój pierwszy odtwarzacz muzyki	444
Wersja 2. Eksperymenty muzyczne z wykorzystaniem argumentów przekazywanych z wiersza poleceń	448
Ćwiczenia	450
Rozwiązania ćwiczeń	453

14 Historia bardzo graficzna

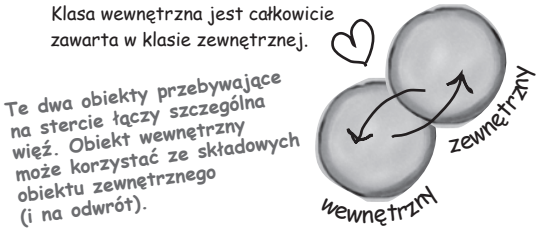
Pogódź się z faktem, że potrzebujesz graficznego interfejsu użytkownika. Nawet jeśli wierzysz, że całe życie spędzisz na pisaniu programów wykonywanych po stronie serwera, gdzie funkcję interfejsu użytkownika pełnią strony WWW, to i tak wcześniej czy później będziesz musiał napisać jakiś program narzędziowy i zapewne zechcesz wyposażyć go w graficzny interfejs użytkownika. Zagadnieniami związanymi z graficznym interfejsem użytkownika będziemy się zajmować przez dwa rozdziały, a w międzyczasie poznamy także kilka nowych możliwości języka Java, takich jak **obsługa zdarzeń** oraz **klasy wewnętrzne**. Wyświetlimy na ekranie przycisk, narysujemy coś w oknie, wyświetlimy obrazek JPEG, a nawet napiszemy prostą animację.

```
class MojaKlasaZewnetrzna {

   class MojaKlasaWewnetrzna {
      void doDziela() {
       }
   }

}
```

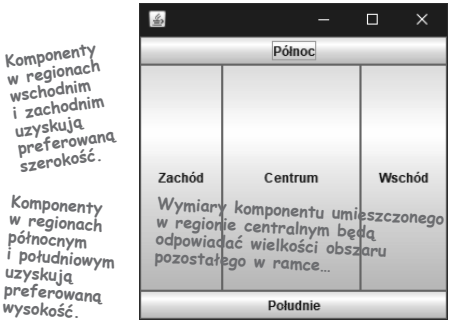
Klasa wewnętrzna jest całkowicie zawarta w klasie zewnętrznej.



Wszystko zaczyna się od okna	456
Przechwytywanie zdarzeń generowanych przez działania użytkownika	459
Odbiorcy, źródła i zdarzenia	463
Stwórz własny komponent umożliwiający rysowanie	466
Fajne operacje, jakie można realizować w metodzie paintComponent()	467
Układy GUI — wyświetlanie w ramce więcej niż jednego komponentu	472
Klasy wewnętrzne spieszą z pomocą!	478
Wyrażenia lambda spieszą z pomocą (ponownie)!	484
Wykorzystanie klas wewnętrznych do tworzenia animacji	486
Prostszy sposób tworzenia komunikatów-zdarzeń	492
Ćwiczenia	496
Rozwiązania ćwiczeń	501

15 Popracuj nad Swingiem

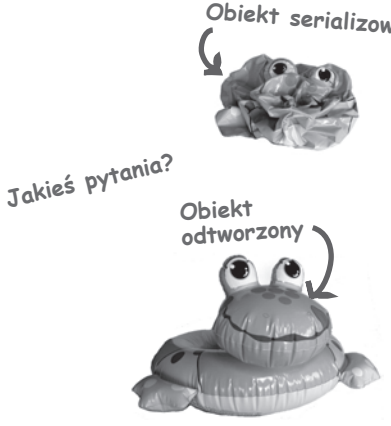
Swing jest łatwy. Chyba że naprawdę *zwracasz uwagę* na to, gdzie zostaną wyświetlone poszczególne elementy interfejsu użytkownika. Kod wykorzystujący bibliotekę Swing *wygląda na prosty*, jednak potem go kompilujesz, uruchamiasz, patrzysz na wyniki i myślisz: „Hej, *to* nie miało być w *tym* miejscu". Elementem, który sprawia, że tworzony *kod* jest *prosty*, a *kontrola* położenia elementów *trudna*, jest **menedżer układu**. Jednak przy niewielkim nakładzie pracy można nagiąć menedżer układu do swojej woli. W tym rozdziale „popracujemy nad naszym Swingiem" i dowiemy się także czegoś więcej o komponentach.



Komponenty biblioteki Swing	504
Menedżery układu	505
Wielka trójka menedżerów układu: BorderLayout, FlowLayout oraz BoxLayout	507
Zabawy z komponentami biblioteki Swing	517
Kod od kuchni	522
Tworzenie aplikacji MuzMachina	523
Ćwiczenia	528
Rozwiązania ćwiczeń	530

16 Zapisywanie obiektów (i tekstu)

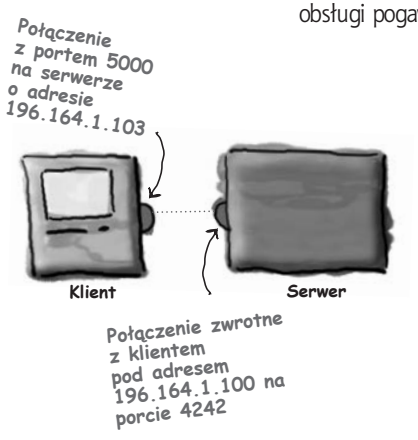
Obiekty można „pakować", a następnie odtwarzać. Obiekty mają swój stan i działanie. Działanie jest określane przez samą *klasę* obiektu, jednak stan określają poszczególne *obiekty*. Jeśli Twój program musi zapisywać stan, to *możesz to zrobić w złożony sposób* — odczytując stan każdego obiektu i pracowicie zapisując wartość każdej zmiennej instancyjnej w pliku w wybranym formacie. Możesz także zrobić to samo w **prosty obiektowy sposób** — po prostu „zapisać-zamrozić-wysuszyć-zachować" sam obiekt, a następnie go „odczytać-odmrozić-namoczyć-odtworzyć".



Zapisywanie serializowanego obiektu do pliku	534
Jeśli chcesz, aby klasa zapewniała możliwość serializacji, zaimplementuj interfejs Serializable	539
Deserializacja — odtwarzanie obiektów	543
Identyfikator wersji — wielki problem serializacji	548
Zapisywanie łańcucha znaków w pliku tekstowym	551
Odczyt zawartości pliku tekstowego	558
Kwiz — gra (zarys kodu)	559
Typy Path, Paths i Files (zabawy z katalogami)	565
Finalnie, bliższe spojrzenie na finally	566
Zapisywanie kompozycji	571
Ćwiczenia	574
Rozwiązania ćwiczeń	576

17 Nawiąż połączenie

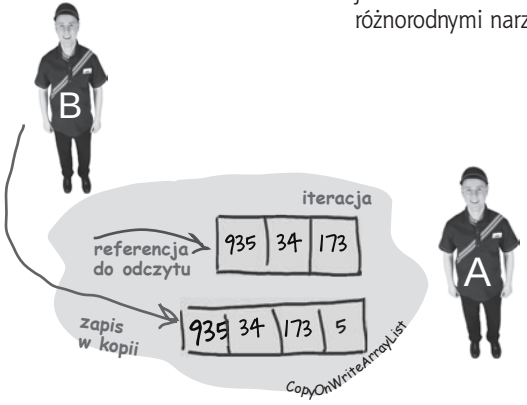
Nawiąż połączenie ze światem zewnętrznym. To całkiem łatwe. Wszystkie szczegóły niskiego poziomu związane z komunikacją sieciową są realizowane przez klasy należące do wbudowanej biblioteki Javy. Jedną z ogromnych zalet Javy jest to, że komunikacja sieciowa bardzo przypomina zwyczajną obsługę wejścia-wyjścia, a obie operacje mogą się różnić jedynie obiektami umieszczanymi na samym końcu łańcucha strumieni. W tym rozdziale utworzymy gniazdo klienta. Utworzymy także gniazdo serwera. Napiszemy również kod klienta oraz kod serwera. Zanim ten rozdział się skończy, będziesz dysponować w pełni funkcjonalnym, wielowątkowym klientem do obsługi pogawędek internetowych. Czy użyliśmy słowa *wielowątkowy*?



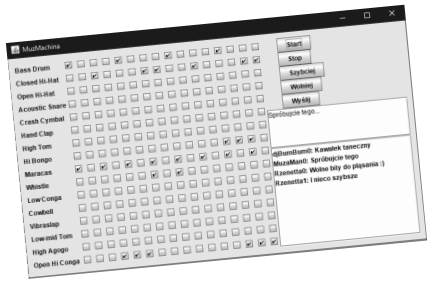
Nawiązywanie połączenia, wysyłanie i odbieranie danych	582
Program CodziennePoradyKlient	590
Tworzenie prostej aplikacji serwera	593
Możliwość stosowania wielu wątków w Javie zapewnia jedna klasa — Thread	602
Trzy stany nowego wątku	608
Usypianie wątku	614
Tworzenie i uruchamianie *dwóch* (lub większej liczby!) wątków	618
Koniec pracy w puli wątków	621
Nowa i poprawiona wersja programu ProstyKlientPogawedek	624
Ćwiczenia	626
Rozwiązania ćwiczeń	628

18 Rozwiązywanie problemów współbieżności

Jednoczesne robienie dwóch lub większej liczby rzeczy jest trudne. Pisanie kodu wielowątkowego nie przysparza problemów. Jednak pisanie kodu wielowątkowego, który będzie działał w oczekiwany sposób, może być znacznie trudniejsze. W tym ostatnim rozdziale książki pokażemy Ci kilka przykładowych problemów, które mogą występować w przypadku korzystania z dwóch lub większej liczby jednocześnie działających wątków. Przy okazji poznasz także wybrane z narzędzi dostępnych w pakiecie `java.util.concurrent`, ułatwiających pisanie poprawnie działającego kodu wielowątkowego. Dowiesz się tu, jak tworzyć obiekty niemodyfikowalne (czyli takie, których stan nie może się zmieniać), których można bezpiecznie używać w wielu jednocześnie działających wątkach. Pod koniec rozdziału będziesz dysponował wieloma różnorodnymi narzędziami do pisania kodu współbieżnego.



Problem Moniki i Roberta w formie kodu	634
Stosowanie blokady obiektu	639
Przerażający problem „utraconej modyfikacji”	642
Zadeklaruj metodę inkrementuj() jako metodę atomową. Synchronizuj ją!	644
Wzajemna blokada, mroczna strona wątków	646
Operacje CAS z użyciem zmiennych atomowych	648
Stosowanie niezmiennych obiektów	651
Więcej problemów ze współużytkowanymi danymi	654
Stosowanie struktur danych bezpiecznych pod względem wielowątkowym	656
Ćwiczenia	660
Rozwiązania ćwiczeń	662



A Dodatek A

Ostatnie doprawianie kodu. Kompletny kod klienta i serwera MuzMachiny z funkcją sieciowych pogawędek. Twoja szansa na zostanie gwiazdą rocka.

Ostateczna wersja programu MuzMachina 666
Ostateczna wersja serwera aplikacji MuzMachina 673

B Dodatek B

Jedenaście najważniejszych tematów, które nie zmieściły się w głównej części książki

Jeszcze nie możemy pozwolić Ci odejść. Wciąż mamy dla Ciebie parę ważnych informacji, ale to już *jest* koniec tej książki. I tym razem mówimy poważnie.

Nr 11. JShell (Java REPL) 676
Nr 10. Pakiety 677
Nr 9. Niezmienność w łańcuchach i klasach opakowujących 680
Nr 8. Poziomy i modyfikatory dostępu (czyli kto co widzi) 681
Nr 7. Zmienna lista argumentów 683
Nr 6. Adnotacje 684
Nr 5. Lambdy i mapy 685
Nr 4. Strumienie równoległe 687
Nr 3. Wyliczenia (nazywane także typami wyliczeniowymi) 696
Nr 2. Wnioskowanie typów zmiennych lokalnych (słowo kluczowe var) 690
Nr 1. Rekordy 691

S Skorowidz 693

Wycieczka do Obiektowa

Powiedziano mi, że będą obiekty. W rozdziale 1. cały tworzony kod był umieszczany w metodzie `main()`. Nie jest to poprawne rozwiązanie obiektowe. W rzeczywistości, z punktu widzenia programowania zorientowanego obiektowo, jest to rozwiązanie całkowicie niewłaściwe. Cóż, w programie krasomówczym wykorzystaliśmy kilka obiektów, takich jak tablice łańcuchów znaków (obiektów `String`), jednak nie stworzyliśmy żadnego własnego *typu obiektowego*. Teraz musimy więc zostawić świat programowania proceduralnego, usunąć to, co najważniejsze, z metody `main()` i zacząć tworzyć własne obiekty. Przyjrzymy się czynnikom, które sprawiają, że programowanie zorientowane obiektowo przy użyciu języka Java jest takie fajne. Przedstawimy różnice pomiędzy klasą a obiektem. Przekonamy się, w jaki sposób obiekty mogą ułatwić nam życie (a przynajmniej jego aspekty związane z programowaniem; nie będziemy bowiem w stanie sprawić, abyś zaczął się znać na modzie i wyrobił sobie dobry gust). Jedno ostrzeżenie: kiedy dotrzesz do Obiektowa, możesz już nigdy się z niego nie wydostać. Wyślij nam pocztówkę.

Wojna o fotel

(albo jak obiekty mogą zmienić Twoje życie)

Dawno temu w sklepie z oprogramowaniem dwóch programistów dostało tę samą specyfikację i kazano im „napisać, co trzeba”. Naprawdę denerwujący szef projektu zmusił obu programistów do rywalizowania ze sobą, obiecując im, że ten, który pierwszy odda kod, dostanie Superfotel™ oraz biurko z regulacją wysokości, które mają wszyscy programiści w Dolinie Krzemowej. Laura, programistka proceduralna, oraz Bronek, programista obiektowy, wiedzą, że wykonanie zadania będzie jak przysłowiowa kaszka z mlekiem.

Laura, siedząc w swoim boksie, pomyślała: „Jakie rzeczy ten program ma *robić*? Jakich ***procedur*** potrzebuje?”. Po czym sama sobie odpowiedziała: „Potrzebne mi będą procedury: **obroc** i **odtworzDzwiek**”. I zabrała się za pisanie odpowiednich procedur. No bo w końcu czym innym jest program, jeśli nie zbiorem stosownych procedur?

W międzyczasie Bronek poszedł do kawiarni na filiżankę kawy i zadał sobie pytanie: „Jakie są ***obiekty*** w tym programie… jacy są jego najważniejsi *bohaterowie*?”. Pierwszą odpowiedzią, jaka mu przyszła do głowy, była: **Figury**. Oczywiście w programie występują też inne obiekty, takie jak Użytkownik, Dźwięk czy też zdarzenie Kliknięcie, ale Bronek dysponuje już biblioteką obsługującą te obiekty, zatem może się skoncentrować na tworzeniu Figur. Przeczytaj dalszą treść rozdziału, aby się przekonać, jak Laura i Bronek tworzyli swoje programy, a przede wszystkim aby uzyskać odpowiedź na najbardziej palące pytanie: ***„Kto dostanie Superfotel i regulowane biurko?”***.

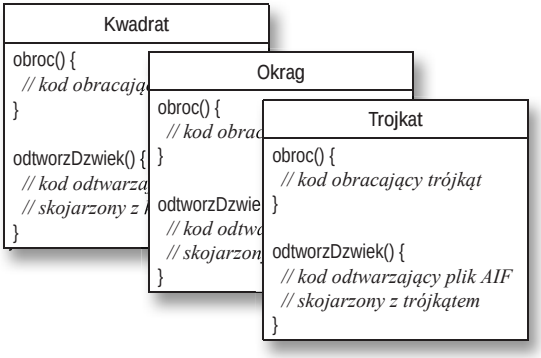
W boksie Laury

Jak miliardy razy wcześniej, Laura zabrała się do pisania **niezwykle ważnych procedur**. Błyskawicznie napisała procedury **obroc** oraz **odtworzDzwiek**:

```
obroc(numFigury) {
   // obrócenie figury o 360 stopni
}
odtworzDzwiek(numFigury) {
   // na podstawie numeru figury określ,
   // jaki plik AIF należy odtworzyć, i odtwórz go
}
```

W boksie Bronka przy filiżance kawy

Bronek stworzył ***klasy*** dla każdej z trzech figur.

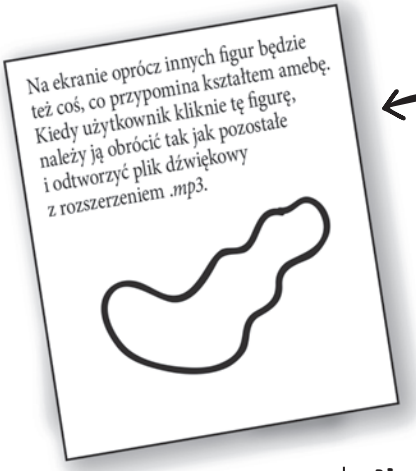


Laura myślała, że ma wygraną w garści.
Już niemal czuła miękką skórę Superfotela pod swoją...

Ale chwileczkę! Nastąpiła zmiana specyfikacji.

— W porządku. *Technicznie* rzecz biorąc, wygrała Laura — powiedział Szef — ale musimy dodać do programu jedną malutką rzecz. Dla takich doświadczonych programistów jak wy nie będzie to stanowiło żadnego problemu.

— *Gdyby tak dostawać dziesięć groszy za każdą zmianę specyfikacji...* — pomyślała Laura, zdając sobie doskonale sprawę z tego, że bezproblemowość zmian specyfikacji jest czystą fikcją. — *A jednak Bronek jest dziwnie spokojny. Co jest grane?* — Mimo to Laura wciąż trzymała się swojej opinii, że podejście obiektowe, choć ciekawe, jest wolne. Pomyślała też, że jeśli ktoś chciałby sprawić, by zmieniła zdanie, musiałby poddać ją bezlitosnym torturom.



← Oto co zostało dodane do specyfikacji

Z powrotem w boksie Laury

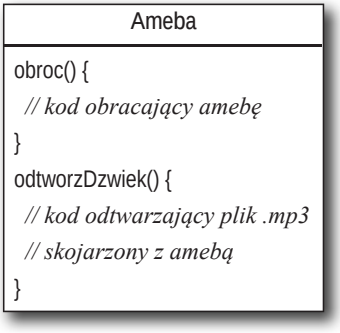
Procedura obracania nawet po zmianach będzie działać dobrze, podobnie jak kod służący do przejrzenia tablicy figur i dopasowania wartości numFigury do faktycznej figury. *Ale trzeba zmienić procedurę odtworzDzwiek.*

```
odtworzDzwiek(numFigury) {
  // jeśli kształt to nie ameba,
     // to na podstawie numeru figury określ,
     // jaki plik AIF należy odtworzyć, i odtwórz
     // go
  // w przeciwnym razie
     // odtwórz plik dźwiękowy .mp3 skojarzony
     // z amebą
}
```

Okazało się, że zmiany nie są takie straszne, ***niemniej konieczność modyfikowania już sprawdzonego kodu wywołała u Laury pewne uczucie niepokoju***. W końcu właśnie *ona* jak *nikt* inny powinna doskonale wiedzieć, że niezależnie od tego, co mówi szef projektu, ***specyfikacja zawsze się zmienia***.

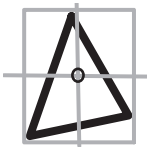
Na plaży na laptopie Bronka

Bronek uśmiechnął się, wychylił łyczek swojej Margarity i *napisał jedną nową klasę*. Czasami myślał sobie, że rzeczą, którą najbardziej kocha w programowaniu obiektowym, jest brak konieczności modyfikowania kodu, który już raz został przetestowany i udostępniony. „Elastyczność, rozszerzalność..." — mruczał pod nosem Bronek, przypominając sobie wszystkie zalety programowania obiektowego.



Laura wpadła do biura szefa tuż przed Bronkiem

(Aha! I tyle są warte te wszystkie obiektowe bzdury). Ale uśmiech na jej twarzy szybko zgasł, kiedy naprawdę denerwujący szef projektu powiedział: — Ależ nie! Ameba miała być obracana w zupełnie *inny* sposób…

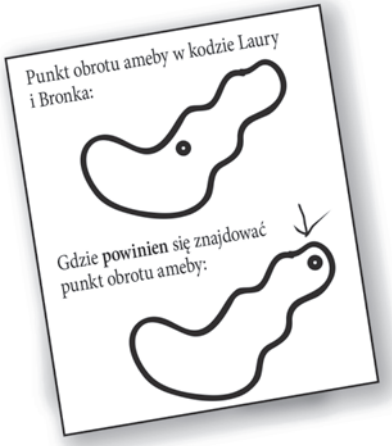


Okazuje się, że kod obracający figury napisany przez obu programistów działa w następujący sposób:

1) Określa prostokątny obszar, w jakim mieści się figura.
2) Wyznacza środek tego obszaru i obraca go wokół tego punktu.

Ale, jak się okazuje, kształt przypominający amebę miał być obracany wokół punktu znajdującego się na jego końcu, zupełnie tak samo jak wskazówka zegara.

— Jestem ugotowana — pomyślała Laura, wyobrażając sobie bulgoczący KociołekSzamana™. — Chociaż, niech pomyślę… Mogłabym przecież dodać do procedury obracającej figury jeszcze jedną instrukcję if-else i w niej zakodować na stałe punkt obrotu dla ameby. To prawdopodobnie nie zepsułoby całej procedury. — Ale wtedy cichutki głosik w jej głowie odezwał się: — *Wielki błąd. Czy jesteś absolutnie pewna, że specyfikacja znowu się nie zmieni?*



O tym beztrosko zapomniano napisać w specyfikacji

Z powrotem w boksie Laury

Laura doszła do wniosku, że lepszym rozwiązaniem będzie dodanie do procedury obracającej figury argumentów, które określają współrzędne punktu obrotu. ***Doprowadziło to do poważnych zmian w kodzie.*** Testowanie, rekompilacja, cała masa roboty, którą trzeba wykonać od nowa. To, co wcześniej działało, teraz przestało działać.

```
obroc(numFigury, xPt, yPt) {
  // jeśli figura to nie ameba,
     // wyznaczenie środka na podstawie
     // prostokąta opisanego
     // i obrócenie figury o 360 stopni,
  // w przeciwnym razie
     // wyznaczenie punktu obrotu z uwzględnieniem
     // podanego przesunięcia xPt, yPt
     // i obrócenie figury o 360 stopni
}
```

Na laptopie Bronka, gdzieś na widowni Festiwalu Kapel Wirtualnych

Nie tracąc nawet jednego taktu z prezentowanych utworów, Bronek zmodyfikował **metodę** obracającą, ale wyłącznie w klasie Ameba. ***W żaden sposób nie zmienił przetestowanych, skompilowanych i działających kodów*** stanowiących pozostałe części programu. Aby określić punkt obrotu figury przypominającej amebę, Bronek dodał **atrybuty**, które będą mieć wszystkie ameby. Zmodyfikował, przetestował i przesłał (oczywiście bezprzewodowo) zmodyfikowaną wersję programu w czasie trwania utworu *Serce metody*.

Ameba
int xPO; int yPO; obroc() {   *// kod obracający amebę*   *// wokół punktu xPO, yPO* } odtworzDzwiek() {   *// kod odtwarzający nowy plik .mp3*   *// skojarzony z amebą* }

Czyli to „obiektowiec" Bronek zdobył Superfotel i biurko, czy tak?

Nie tak szybko. Laura znalazła pewną wadę rozwiązania przedstawionego przez Bronka. A ponieważ uważała, że jeśli zdobędzie Superfotel i biurko, to będzie także pierwszą kandydatką do przyszłego awansu, musiała przystąpić do natarcia.

LAURA: W twoim programie powtarzają się te same fragmenty kodu! Procedura do obracania jest we wszystkich tych rzeczach… no, figurach.

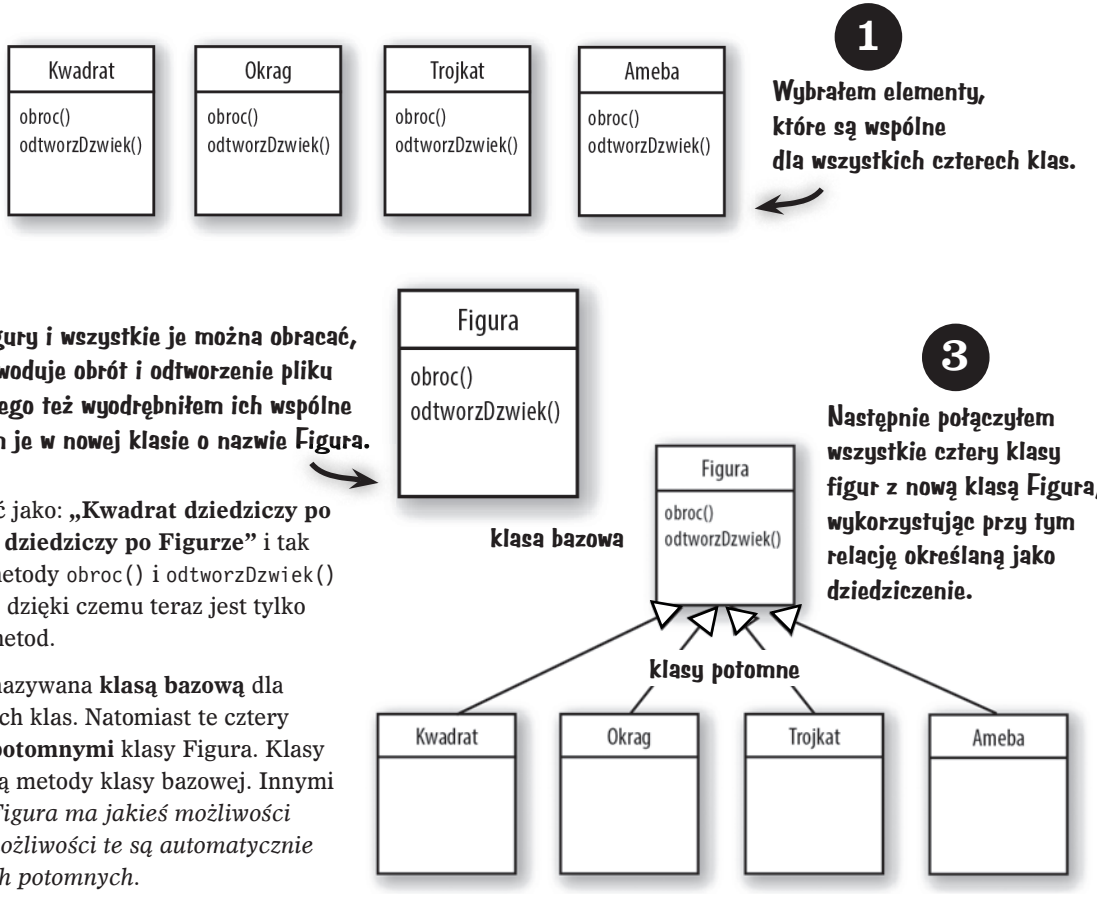
BRONEK: To jest ***metoda***, a nie *procedura*. A to nie są *rzeczy*, tylko ***klasy***.

LAURA: Nieważne. Idiotyczny projekt. Musisz mieć aż *cztery* różne „metody" do obracania. Czy taki projekt może być dobry?

BRONEK: O! Jak mniemam, nie widziałaś ostatecznej wersji. Pozwól, że pokażę ci coś, co w programowaniu obiektowym określamy jako **dziedziczenie**.

Oto czego naprawdę chciała Laura

(uważała, że fotel będzie pierwszym krokiem do awansu i większych pieniędzy)



Należy to rozumieć jako: **„Kwadrat dziedziczy po Figurze"**, **„Okrąg dziedziczy po Figurze"** i tak dalej. Usunąłem metody `obroc()` i `odtworzDzwiek()` z pozostałych klas, dzięki czemu teraz jest tylko jedna kopia tych metod.

Klasa `Figura` jest nazywana **klasą bazową** dla pozostałych czterech klas. Natomiast te cztery klasy są **klasami potomnymi** klasy Figura. Klasy potomne dziedziczą metody klasy bazowej. Innymi słowy, *jeśli klasa Figura ma jakieś możliwości funkcjonalne, to możliwości te są automatycznie dostępne w klasach potomnych.*

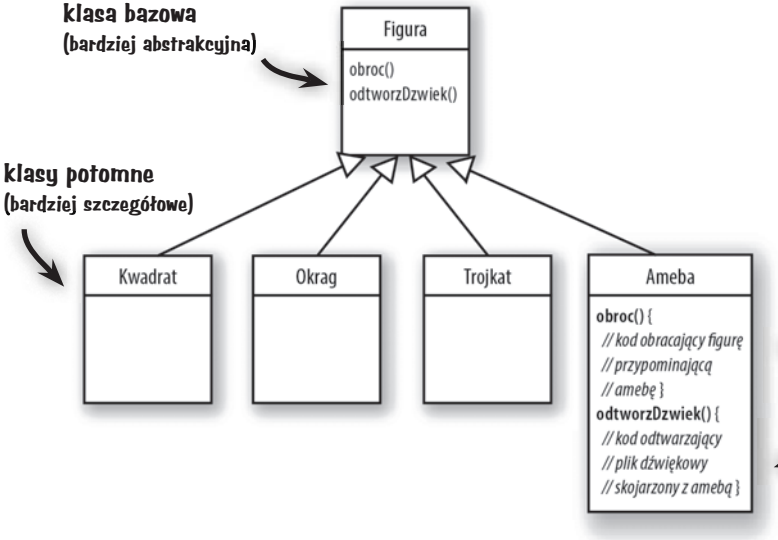
A co z metodą obroc() dla ameby?

LAURA: Ale na czym polegał cały problem — czy nie na tym, że figura przypominająca kształtem amebę miała mieć całkowicie inne procedury `obroc()` i `odtworzDzwiek()`?

BRONEK: Metody.

LAURA: Nieważne. W jaki sposób ameba może robić coś innego, jeśli dziedziczy możliwości funkcjonalne po klasie `Figura`?

BRONEK: To ostatni etap zadania. Klasa `Ameba` może **przesłonić** metody klasy `Figura`. Następnie, podczas wykonywania programu, kiedy każemy obrócić amebę, JVM będzie dokładnie wiedzieć, jaką metodę `obroc()` należy wykonać.



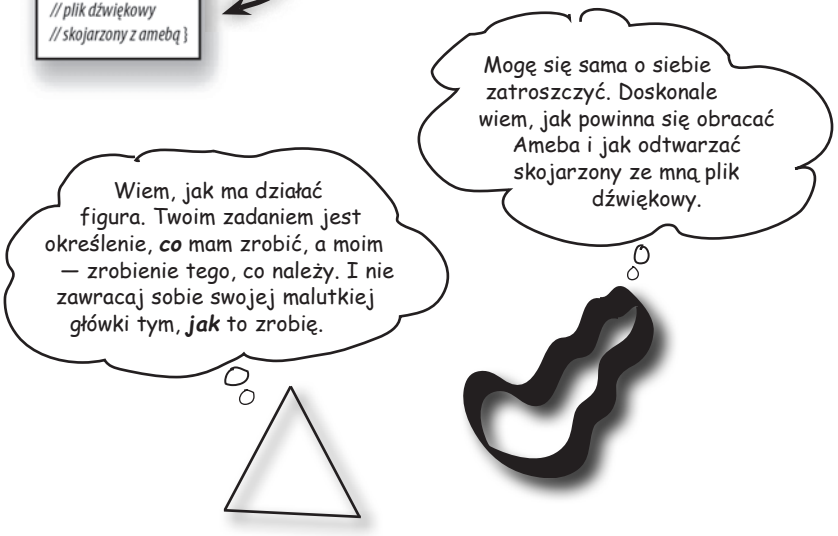
4

W klasie Ameba przesłaniam metody obroc() i odtworzDzwiek() bazowej klasy Figura.

Przesłanianie oznacza po prostu, że jeśli klasa potomna musi zmienić lub rozszerzyć działanie jednej z odziedziczonych metod, to ponownie ją definiuje.

LAURA: A w jaki sposób „każesz" obiektowi Ameba coś zrobić? Czy nie musisz wywołać jakiejś funkcji, o, przepraszam — *metody* — i wskazać jej, *którą* figurę obrócić?

BRONEK: I to właśnie jest najfajniejsza rzecz w programowaniu obiektowym. Kiedy trzeba obrócić na przykład trójkąt, kod programu wywołuje metodę `obroc()` *obiektu trójkąta*. Pozostała część programu tak naprawdę wcale nie interesuje się tym, w jaki sposób obraca się trójkąt. A kiedy trzeba dodać coś nowego do programu, tworzy się nową klasę dla nowego typu obiektu; dzięki temu ten **nowy obiekt będzie się zachowywać w sposób unikalny**.



Ta niepewność mnie zabije! Kto wygra Superfotel i biurko?

Ania z drugiego piętra.

(Bez czyjejkolwiek wiedzy szef projektu przekazał specyfikację programu *trzem* programistom. Ania skończyła go najszybciej, gdyż użyła wersji obiektowej, a przy tym z nikim się nie musiała spierać).

Co Ci się podoba w programowaniu obiektowym?

„Pomaga mi projektować programy w bardziej naturalny sposób. Obiekty wchodzące w skład programów mogą ewoluować".

— Jonasz, 27, projektant oprogramowania

„To, że w razie konieczności dodania nowych możliwości nie muszę ingerować w kod, który już został napisany i przetestowany".

— Bartek, 32, programista

„Podoba mi się, że dane i metody, które na tych danych operują, są zgrupowane w jednej klasie".

— Jonatan, 22, miłośnik piwa

„Podoba mi się możliwość wykorzystywania kodu w innych aplikacjach. Tworząc nową klasę, mogę ją napisać w sposób na tyle elastyczny, by można ją było wykorzystać w przyszłości w innych programach".

— Krzysiek, 39, menedżer projektu

„Nie mogę uwierzyć, że Krzysiek, który od pięciu lat nie napisał nawet jednej linijki kodu, powiedział coś takiego".

— Darek, 44, pracuje dla Krzyśka

„Oprócz możliwości zdobycia Superfotela?"

— Ania, 34, programistka

MOC UMYSŁU

Nadszedł czas, żeby trochę rozruszać neurony.

Przeczytałeś właśnie opowieść o programiście proceduralnym konkurującym z programistą obiektowym. Przy okazji mogłeś się przyjrzeć krótkiej prezentacji kluczowych pojęć związanych z programowaniem obiektowym, takich jak klasy, metody oraz atrybuty. Dalsza część rozdziału zostanie poświęcona dokładniejszej prezentacji klas i obiektów (do zagadnień dziedziczenia i przesłaniania metod powrócimy w kolejnych rozdziałach).

Bazując na informacjach uzyskanych do tej pory (a może także na wiedzy zdobytej podczas korzystania z innego języka zorientowanego obiektowo), poświęć chwilkę na przemyślenie poniższych pytań i próbę podania na nie odpowiedzi:

Jakie są podstawowe zagadnienia, które należy przemyśleć, projektując klasy w języku Java? Jakie pytania należy sobie przy tym zadać? Gdybyś miał stworzyć listę rzeczy, które należy zrobić podczas projektowania nowych klas, to co znalazłoby się na tej liście?

Metapoznaniowa podpowiedź

Jeśli utknąłeś i nie możesz znaleźć rozwiązania ćwiczenia, spróbuj porozmawiać o nim na głos ze sobą. Mówienie (i słuchanie) aktywizuje różne części mózgu. Choć metoda ta daje najlepsze rezultaty, kiedy można porozmawiać z inną osobą, to jednak można także porozmawiać z ulubionym zwierzakiem. To właśnie w ten sposób nasz pies dowiedział się, czym jest polimorfizm.

Projektując klasę, myśl o obiektach, które będą tworzone na podstawie jej typu. Pomyśl o:

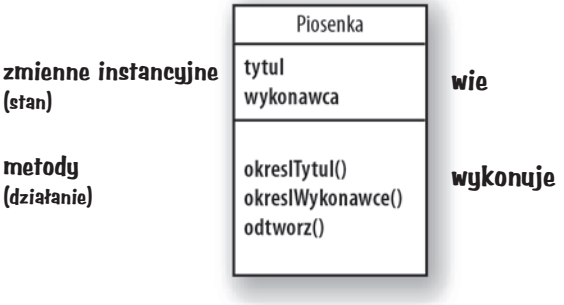
- informacjach, jakie obiekt **zna**,
- czynnościach, jakie obiekt **wykonuje**.

Informacje, jakie obiekt *ma* o sobie, są nazywane

- zmiennymi instancyjnymi

Czynności, jakie obiekt jest w stanie *wykonywać*, są nazywane

- metodami



To, co obiekt ***wie*** na swój temat, nazywamy **zmiennymi instancyjnymi**. Reprezentują one stan obiektu (dane) i mogą przyjmować unikalne wartości w każdym z obiektów danego typu.

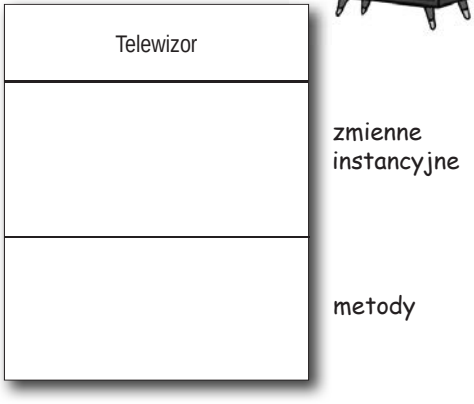
Pamiętaj, że mówiąc o instancji, mamy na myśli obiekt.

Czynności, jakie obiekt może ***wykonywać***, nazywamy **metodami**. Projektując klasę, należy określić, jakie informacje na swój temat obiekt musi znać, jak również zaprojektować metody, które będą operować na tych danych. Bardzo często się zdarza, że obiekty mają metody służące do ustawiania oraz odczytywania wartości zmiennych instancyjnych. Na przykład obiekt Alarm ma zmienną instancyjną określającą czas alarmu oraz dwie metody służące do określania i pobierania tego czasu.

Obiekty mają zatem instancje, zmienne instancyjne oraz metody, jednak zarówno zmienne instancyjne, jak i metody stanowią części klasy.

Zaostrz ołówek

Poniżej wpisz, **co obiekt Telewizor** powinien wiedzieć i robić.



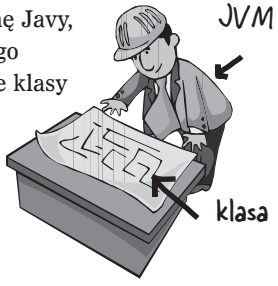
→ Do samodzielnego rozwiązania.

Jaka jest różnica pomiędzy klasą a obiektem?

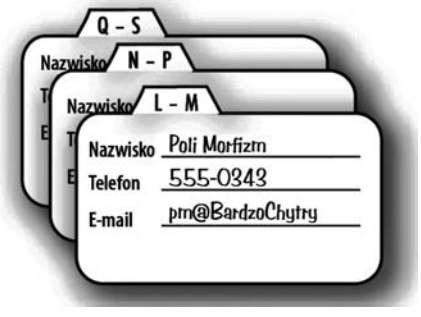
Klasa nie jest obiektem

(jednak służy do ich tworzenia)

Klasa jest jak gdyby *matrycą* służącą do tworzenia obiektów. Informuje ona wirtualną maszynę Javy, *jak* należy utworzyć obiekt tego konkretnego typu. Każdy obiekt utworzony na podstawie klasy może mieć unikalne wartości zmiennych instancyjnych. Na przykład można użyć klasy `Przycisk` do stworzenia kilkunastu różnych przycisków, z których każdy będzie mieć inne kolor, wielkość, kształt, etykietę i tak dalej. Każdy z tych różnych przycisków byłby odrębnym *obiektem*.



Wyobraź to sobie w następujący sposób...



Obiekt można by porównać z jednym wpisem na liście kontaktów.

Jedną z możliwych analogii klas i obiektów jest lista kontaktów w telefonie. Każdy z kontaktów ma takie same pola do wypełnienia (odpowiadające zmiennym instancyjnym obiektu). Kiedy tworzysz nowy kontakt, tworzysz instancję (obiekt), a informacje zapisane w tym kontakcie będą reprezentować jego stan.

Metody klasy to czynności, jakie można wykonywać na konkretnym kontakcie; klasa `Kontakt` mogłaby mieć następujące metody: `pobierzNazwisko()`, `zmienNazwisko()`, `okreslNazwisko()`.

Każdy kontakt może zatem wykonywać te same operacje (pobrać nazwisko, zmienić je i tak dalej), jednak każdy z nich zawiera unikalne informacje, charakterystyczne wyłącznie dla niego.

Tworzenie pierwszego obiektu

Czego więc będziesz potrzebować do stworzenia i wykorzystania swojego pierwszego obiektu? Będą Ci potrzebne *dwie* klasy. Pierwsza będzie klasą obiektu, którego chcesz użyć (Pies, Budzik, Telewizor i tak dalej), natomiast druga posłuży do *przetestowania* nowej klasy. To właśnie w tej klasie *testującej* zostanie umieszczona metoda `main()`, a w niej będzie tworzony obiekt Twojej nowej klasy. Klasa testująca ma tylko jedno zadanie — *przetestować* metody i zmienne instancyjne obiektu Twojej nowej klasy.

Zaczynając od tego miejsca, w wielu przykładach przedstawionych w dalszej części książki będziesz mógł znaleźć dwie klasy. Pierwsza z nich będzie tą *właściwą* — czyli klasą, której obiektów chcemy używać; z kolei druga będzie klasą testującą, a jej nazwa będzie odpowiadać nazwie klasy właściwej z dodanym na końcu słowem **`Tester`**. Na przykład jeśli stworzymy klasę **`Bungee`**, to będzie nam potrzebna także klasa **`BungeeTester`**. Wyłącznie klasa <*jakaśTamKlasa*>**`Tester`** będzie posiadać metodę `main()`, a celem jej istnienia będzie stworzenie obiektów nowej klasy (nie klasy testującej) i wykorzystanie operatora kropki (.) w celu uzyskania dostępu do metod i zmiennych tych obiektów. Wszystkie te zasady staną się całkowicie jasne, gdy przeanalizujesz poniższe przykłady.

Operator kropki (.)

Operator kropki (.) zapewnia dostęp do stanu oraz do zachowania obiektu (zmiennych instancyjnych i metod).

```
// tworzymy nowy obiekt
Pies p = new Pies();

// każemy psu szczekać,
// dodając do zmiennej p
// operator kropki w celu
// wywołania metody
// szczekaj()
p.szczekaj();

// określamy wielkość psa,
// także tym razem
// używając operatora
// kropki (.)
p.wielkosc = 40;
```

1 Napisz kod klasy.

```
class Pies {
  int wielkosc;
  String rasa;
  String imie;

  void szczekaj() {
    System.out.println("Hau! Hau, hau!");
  }
}
```

zmienne instancyjne

metoda

PIES
wielkosc
rasa
imie
szczekaj()

2 Napisz klasę testującą (Tester).

```
class PiesTester {
  public static void main(String[] args) {
    // Tutaj umieścimy kod
    // testujący klasę Pies
  }
}
```

klasa posiada tylko metodę main() (w następnym kroku umieścimy w tej metodzie jakiś kod)

3 W klasie testującej stwórz obiekt i użyj jego zmiennych instancyjnych i metod.

```
class PiesTester {
  public static void main(String[] args) {
    Pies p = new Pies();
    p.wielkosc = 40;
    p.szczekaj();
  }
}
```

utwórz obiekt Pies

operator kropki

użyj operatora kropki (.) w celu określenia wielkości psa

i wywołania jego metody szczekaj()

Jeśli masz już jakieś rozeznanie w programowaniu obiektowym, to będziesz wiedział, że nie używamy hermetyzacji. Tym zagadnieniem zajmiemy się w rozdziale 4., „Jak działają obiekty?".

Tworzenie i testowanie obiektów Film


```
class Film {
  String tytul;
  String gatunek;
  int ocena;

  void odtworz() {
    System.out.println("Odtwarzamy film.");
  }
}
public class FilmTester {
  public static void main(String[] args) {
    Film pierwszy = new Film();
    pierwszy.tytul = "Przeminęło z hossą";
    pierwszy.gatunek = "Tragedia";
    pierwszy.ocena = -2;
    Film drugi = new Film();
    drugi.tytul = "Matrix dla zuchwałych";
    drugi.gatunek = "Komedia";
    drugi.ocena = 5;
    drugi.odtworz();
    Film trzeci = new Film();
    trzeci.tytul = "Byte Club";
    trzeci.gatunek = "Tragedia, ale o wydźwięku optymistycznym";
    trzeci.ocena = 127;
  }
}
```

Zaostrz ołówek

FILM
tytul gatunek ocena
odtworz()

Klasa `FilmTester` tworzy trzy obiekty (instancje) klasy `Film`, a następnie, przy wykorzystaniu operatora kropki (`.`), przypisuje konkretne wartości ich zmiennym instancyjnym. Klasa ta wywołuje także metodę jednego z tych obiektów. Na rysunku z prawej strony w pustych miejscach wpisz wartości, jakie będą miały odpowiednie zmienne obiektów pod koniec działania metody `main()`.

obiekt 1
tytuł
rodzaj
ocena

obiekt 2
tytuł
rodzaj
ocena

obiekt 3
tytuł
rodzaj
ocena

→ Do samodzielnego rozwiązania.

Szybko! Opuszczamy metodę main()!

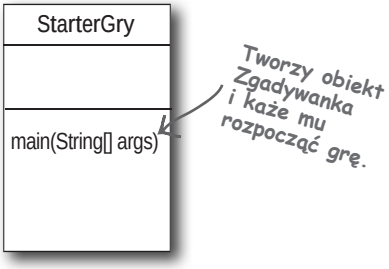
Dopóki działasz w obrębie metody `main()`, dopóty tak naprawdę nie dotarłeś do Obiektowa. Wykonywanie operacji w tej metodzie jest dobre dla prostego programu testowego, jednak prawdziwe aplikacje pisane obiektowo wymagają, aby obiekty komunikowały się z innymi obiektami, a nie były tworzone i testowane w jakiejś statycznej metodzie.

Dwa zastosowania metody main():

- do **testowania** klas wykorzystywanych w aplikacji,
- do **uruchamiania** bądź wykonywania **aplikacji**.

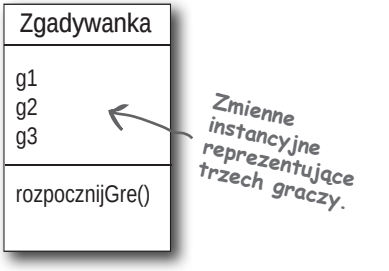
Prawdziwa aplikacja napisana w Javie to w zasadzie nic innego jak grupa obiektów, które komunikują się pomiędzy sobą. W tym przypadku *komunikowanie się* oznacza wywoływanie metod obiektów. Na poprzedniej stronie (jak również w rozdziale 4. książki, „Jak działają obiekty?") metoda `main()` została umieszczona w niezależnej klasie Tester i służyła do utworzenia i sprawdzenia metod oraz zmiennych innych klas. W rozdziale 6., „Korzystanie z biblioteki Javy", przyjrzymy się wykorzystaniu klasy, w której metoda `main()` służy do uruchomienia *prawdziwej* aplikacji napisanej w Javie (czyli między innymi do utworzenia obiektów i zapewnienia im możliwości interakcji z innymi obiektami).

Na razie przedstawimy jednak prosty przykład tego, jak może działać prawdziwa aplikacja Javy. Ponieważ wciąż znajdujemy się na samym początku nauki Javy, nasz warsztat jest bardzo ograniczony, dlatego też uznasz zapewne, że przedstawiony program jest nieco głupkowaty i nieefektywny. Możesz się zastanowić, co zrobić, aby go poprawić; swoją drogą, dokładnie to zrobimy w następnych rozdziałach. Nie przejmuj się, jeśli będziesz mieć trudności ze zrozumieniem fragmentów kodu — jego podstawowym celem jest przedstawienie komunikacji pomiędzy obiektami.



Zgadywanka

Podsumowanie:
Nasza gra — zgadywanka — wykorzystuje obiekt gry oraz trzy obiekty graczy. Gra generuje liczbę losową z zakresu od 0 do 9, a trzej gracze starają się ją odgadnąć. (Nikt nie mówił, że to ma być *pasjonująca* gra).

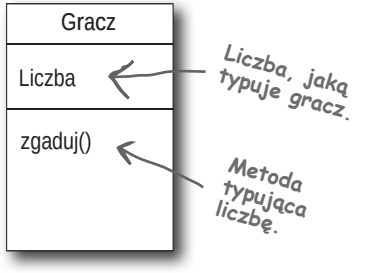


Klasy:

```
Zgadywanka.class Gracz.class StarterGry.class
```

Logika działania:

1. Działanie aplikacji rozpoczyna się w klasie `StarterGry`; klasa ta posiada metodę `main()`.
2. W metodzie `main()` jest tworzony obiekt `Zgadywanka`, a następnie zostaje wywołana metoda `rozpocznijGre()`.
3. Cała gra odbywa się wewnątrz metody `rozpocznijGre()` obiektu `Zgadywanka`. Metoda ta tworzy trzech graczy, po czym „wymyśla" losową liczbę (którą gracze mają odgadnąć). Następnie metoda prosi graczy o odgadnięcie liczby, sprawdza podane przez nich wartości i wyświetla informacje o zwycięzcy (zwycięzcach) albo prosi o ponowną próbę odgadnięcia.



```
public class Zgadywanka {
  Gracz g1;
  Gracz g2;
  Gracz g3;

  public void rozpocznijGre() {
    g1 = new Gracz();
    g2 = new Gracz();
    g3 = new Gracz();

    int typg1 = 0;
    int typg2 = 0;
    int typg3 = 0;

    boolean g1odgadl = false;
    boolean g2odgadl = false;
    boolean g3odgadl = false;

    int liczbaOdgadywana = (int) (Math.random() * 10);
    System.out.println("Myślę o liczbie z zakresu od 0 do 9...");

    while (true) {
      System.out.println("Należy wytypować liczbę: " + liczbaOdgadywana);

      g1.zgaduj();
      g2.zgaduj();
      g3.zgaduj();

      typg1 = g1.liczba;
      System.out.println("Gracz pierwszy wytypował liczbę: " + typg1);

      typg2 = g2.liczba;
      System.out.println("Gracz drugi wytypował liczbę: " + typg2);

      typg3 = g3.liczba;
      System.out.println("Gracz trzeci wytypował liczbę: " + typg3);

      if (typg1 == liczbaOdgadywana) {
        g1odgadl = true;
      }
      if (typg2 == liczbaOdgadywana) {
        g2odgadl = true;
      }
      if (typg3 == liczbaOdgadywana) {
        g3odgadl = true;
      }

      if (g1odgadl || g2odgadl || g3odgadl) {
        System.out.println("Mamy zwycięzcę!");
        System.out.println("Czy gracz pierwszy wytypował poprawnie?" + g1odgadl);
        System.out.println("Czy gracz drugi wytypował poprawnie?" + g2odgadl);
        System.out.println("Czy gracz trzeci wytypował poprawnie?" + g3odgadl);
        System.out.println("Koniec gry.");
        break; // Gra skończona, zatem wychodzimy z pętli while
      } else {
        // musimy kontynuować, gdyż nikomu nie udało się wytypować poprawnie!
        System.out.println("Gracze będą musieli spróbować jeszcze raz.");
      } // koniec if - else
    } // koniec while
  } // koniec metody rozpocznijGre
} // koniec klasy
```

Klasa Zgadywanka ma trzy zmienne instancyjne służące do przechowywania trzech obiektów Gracz.

Utworzenie trzech obiektów Gracz i zapisanie ich w trzech zmiennych instancyjnych.

Deklaracja trzech zmiennych, w których będą przechowywane trzy liczby wytypowane przez poszczególnych graczy.

Deklaracja trzech zmiennych, które będą przechowywać wartości true (prawda) lub false (fałsz), w zależności od odpowiedzi konkretnego gracza.

Wyznaczenie liczby, jaką będą musieli odgadnąć gracze.

Wywołanie metody zgaduj() każdego z graczy.

Pobranie liczb wytypowanych przez każdego z graczy (wyników wywołania metody zgaduj()) poprzez odczytanie ich ze zmiennych instancyjnych obiektów graczy.

Sprawdzenie liczb wytypowanych przez graczy w celu określenia, czy odpowiadają one wyznaczonej liczbie. Jeśli gracz wytypował poprawnie, to odpowiedniej zmiennej przypisywana jest wartość true (pamiętaj, że domyślnie zmienna ta ma wartość false).

Jeśli gracz pierwszy LUB gracz drugi, LUB gracz trzeci odgadł… (operator || to logiczne LUB).

W przeciwnym przypadku pętla jest dalej realizowana, a gracze są proszeni o wytypowanie kolejnych liczb.

Uruchamianie zgadywanki

Java sama wynosi śmieci

Za każdym razem, gdy w Javie jest tworzony obiekt, trafia on do obszaru pamięci nazywanego **stertą**. Wszystkie obiekty, niezależnie od tego, kiedy, jak i gdzie zostaną utworzone, zawsze są przechowywane na stercie. Jednak nie jest to sterta starej, zapomnianej pamięci; w Javie sterta jest także nazywana **stertą automatycznie odśmiecaną**. Kiedy tworzysz obiekt, Java rezerwuje na stercie obszar o wielkości odpowiadającej potrzebom konkretnego obiektu. Obiekt posiadający, dajmy na to, 15 zmiennych instancyjnych będzie prawdopodobnie potrzebował więcej miejsca niż obiekt mający jedynie 2 zmienne instancyjne. Co się jednak dzieje, kiedy będzie trzeba odzyskać miejsce przydzielone na stercie? W jaki sposób można usunąć z niej obiekt, kiedy skończymy go już używać? To Java zarządza pamięcią za nas! Kiedy wirtualna maszyna Javy (JVM) „zauważy", że obiekt nie będzie już mógł być wykorzystywany w programie, zostaje on uznany za *nadający się do odśmiecenia*, a gdy w systemie zacznie brakować pamięci, zostanie uruchomiony odśmiecacz, który usunie z niej wszystkie nieosiągalne obiekty. W ten sposób pamięć zostanie zwolniona i będzie można ją ponownie wykorzystać. Więcej informacji na ten temat znajdziesz w kolejnym rozdziale.

Wyniki (za każdym razem będą inne)

```
Wiersz polecenia
H:>java StarterGry
Myślę o liczbie z zakresu od 0 do 9...
Należy wytypować liczbę: 4
Typuj liczb: 3
Typuj liczb: 7
Typuj liczb: 3
Gracz pierwszy wytypował liczbę: 3
Gracz drugi wytypował liczbę: 7
Gracz trzeci wytypował liczbę: 3
Gracze będą musieli spróbować jeszcze raz.
Należy wytypować liczbę: 4
Typuj liczb: 9
Typuj liczb: 3
Typuj liczb: 1
Gracz pierwszy wytypował liczbę: 9
Gracz drugi wytypował liczbę: 3
Gracz trzeci wytypował liczbę: 1
Gracze będą musieli spróbować jeszcze raz.
Należy wytypować liczbę: 4
Typuj liczb: 6
Typuj liczb: 1
Typuj liczb: 1
Gracz pierwszy wytypował liczbę: 6
Gracz drugi wytypował liczbę: 1
Gracz trzeci wytypował liczbę: 1
Gracze będą musieli spróbować jeszcze raz.
Należy wytypować liczbę: 4
Typuj liczb: 7
Typuj liczb: 7
Typuj liczb: 9
Gracz pierwszy wytypował liczbę: 7
Gracz drugi wytypował liczbę: 7
Gracz trzeci wytypował liczbę: 9
Gracze będą musieli spróbować jeszcze raz.
Należy wytypować liczbę: 4
Typuj liczb: 9
Typuj liczb: 7
Typuj liczb: 2
Gracz pierwszy wytypował liczbę: 9
Gracz drugi wytypował liczbę: 7
Gracz trzeci wytypował liczbę: 2
Gracze będą musieli spróbować jeszcze raz.
Należy wytypować liczbę: 4
Typuj liczb: 4
Typuj liczb: 7
Typuj liczb: 2
Gracz pierwszy wytypował liczbę: 4
Gracz drugi wytypował liczbę: 7
Gracz trzeci wytypował liczbę: 2
Mamy zwycięzcę!
Czy gracz pierwszy wytypował poprawnie? true
Czy gracz drugi wytypował poprawnie? false
Czy gracz trzeci wytypował poprawnie? false
Koniec gry.
```

Nie istnieją głupie pytania

P: Co zrobić, jeśli będę potrzebować globalnych zmiennych i metod? Jak to zrobić, jeśli wszystko musi być umieszczane wewnątrz klas?

O: W programach obiektowych pisanych w Javie nie istnieje pojęcie zmiennych lub metod globalnych. W szczególnych zastosowaniach istnieją jednak sytuacje, gdy chcemy, aby metoda (lub stała) była dostępna dla dowolnego fragmentu kodu działającego w dowolnej części programu. Przypomnij sobie metodę `random()` zastosowaną w programie krasomówczym — stanowi ona doskonały przykład metody, którą można wywołać w dowolnym miejscu programu. Albo na przykład stała *pi*. W rozdziale 10. dowiesz się, że oznaczenie metod jako publicznych (przy użyciu słowa kluczowego `public`) i statycznych (przy użyciu słowa kluczowego `static`) sprawia, że zachowują się one jak metody „globalne" — będzie miał do nich dostęp dowolny kod działający w dowolnej klasie wchodzącej w skład programu. Z kolei jeśli zmienna zostanie oznaczona jako publiczna, statyczna i finalna (odpowiednio przy użyciu słów kluczowych: `public`, `static` oraz `final`), to w efekcie stanie się ona globalnie dostępną *stałą*.

P: Ale co to za obiektowość, skoro wciąż można tworzyć zarówno funkcje, jak i dane globalne?

O: Przede wszystkim to, co jest tworzone w Javie, musi być umieszczone w jakiejś klasie. Stała *pi* oraz metoda `random()`, choć statyczne i publiczne, są zatem jednak zdefiniowane w klasie `Math`. Poza tym należy pamiętać, że takie dane i metody „globalne", stanowią w Javie raczej wyjątek, a nie regułę. Są one szczególnym przypadkiem, w którym nie trzeba tworzyć wielu kopii obiektu, aby skorzystać z jego danych lub metod.

P: Czym jest program pisany w Javie? Co się w zasadzie *rozpowszechnia*?

O: Program napisany w Javie to grupa klas (a przynajmniej *jedna* klasa). W każdej aplikacji dokładnie *jedna* klasa musi mieć metodę `main()`, która służy do uruchamiania programu. Ty, jako programista, musisz zatem napisać jedną lub większą liczbę klas. I właśnie one są rozpowszechniane jako program. Jeśli użytkownik końcowy nie posiada JVM, to do klas tworzących aplikację trzeba będzie dołączyć także środowisko wykonawcze Javy, dzięki któremu użytkownicy będą mogli uruchomić program. Dostępnych jest wiele programów instalacyjnych pozwalających na łączenie własnych klas z JVM i umieszczanie ich w katalogu lub pliku, który będzie można udostępnić w dowolnie wybrany sposób (na przykład przez internet). Następnie użytkownik końcowy może zainstalować odpowiednią wersję JVM (zakładając, że na jego komputerze wirtualna maszyna Javy nie jest jeszcze zainstalowana).

P: A co w sytuacji, gdy moją aplikację tworzy sto klas? Albo tysiąc? Czy dostarczanie tylu plików nie jest poważnym utrudnieniem? Czy nie można z nich zrobić jednego dużego, *wykonywalnego pliku aplikacji*?

O: Owszem, dostarczanie użytkownikowi końcowemu tak dużej ilości plików byłoby kłopotliwe. Na szczęście nie jest to konieczne. Można umieścić wszystkie pliki tworzące aplikację w jednym „archiwum Javy" — pliku *.jar* — bazującym na formacie archiwów pkzip. W pliku jar można umieścić odpowiednio sformatowany plik tekstowy stanowiący tak zwany *manifest* i określający, która klasa umieszczona w danym archiwum zawiera metodę `main()`, którą należy wywołać.

CELNE SPOSTRZEŻENIA

- Programowanie obiektowe pozwala na rozszerzanie programów bez konieczności modyfikowania przetestowanego wcześniej działającego kodu.
- W Javie cały tworzony kod jest umieszczany wewnątrz **klas**.
- Klasa opisuje, jak należy tworzyć obiekty danego typu. **Można ją zatem porównać do wzorca.**
- Obiekt potrafi o siebie zadbać; nie musisz ani wiedzieć, ani zaprzątać sobie głowy tym, *jak* obiekt coś robi.
- Obiekt **posiada** informacje i **wykonuje** czynności.
- Informacje, jakie obiekt ma na swój temat, są przechowywane w tak zwanych **zmiennych instancyjnych**. Reprezentują one *stan* danego obiektu.
- Czynności, jakie obiekt wykonuje, są nazywane **metodami**. Określają one *działanie* (lub *zachowanie*) obiektu.
- Tworząc klasę, można także stworzyć niezależną klasę testową służącą do tworzenia i sprawdzania obiektów nowej klasy.
- Klasa może **dziedziczyć** zmienne instancyjne i metody po bardziej abstrakcyjnych **klasach bazowych**.
- W czasie wykonywania program Javy jest w zasadzie grupą wzajemnie komunikujących się obiektów.

Ćwiczenie

BĄDŹ kompilatorem

Każdy z plików przedstawionych na tej stronie stanowi niezależny kompletny plik źródłowy. Twoim zadaniem jest stać się kompilatorem i określić, czy przedstawione programy skompilują się, czy nie. Jeśli nie można ich skompilować, to jak je poprawić? Jeśli można je skompilować, to jakie wyniki wygenerują?

A

```
class StreamingUtworu {

  String tytul;
  String artysta;
  int czasTrwania;

  void odtworz() {
    System.out.println("Odtwarzam piosenkę");
  }

  void wyswietlSzczegoly() {
    System.out.println("To jest utwór "
      + tytul + " grany przez " + artysta);
  }
}

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

    utwor.artysta = "The Beatles";
    utwor.tytul = "Come Together";
    utwor.odtworz();
    utwor.wyswietlSzczegoly();
  }
}
```

B

```
class Odcinek {

  int numerSezonu;
  int numerOdcinka;

  void pominIntro() {
    System.out.println("Pomijam intro...");
  }

  void przeskoczDoNastepnego() {
    System.out.println("Wczytuję następny odcinek...");
  }
}

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

    Odcinek odcinek = new Odcinek();
    odcinek.numerSezonu = 4;
    odcinek.odtworz();
    odcinek.pominIntro();
  }
}
```

→ Odpowiedzi znajdziesz na stronie 46.

Ćwiczenie

Magnesiki z kodem

Odpowiedź znajdziesz na stronie 46.

Działający program Javy został podzielony na fragmenty i zapisany na małych magnesach, które przyczepiono do lodówki. Czy jesteś w stanie złożyć go z powrotem w jedną całość, tak aby wygenerował przedstawione poniżej wyniki? Niektóre nawiasy klamrowe spadły na podłogę i były zbyt małe, aby można je było podnieść; dlatego w razie potrzeby możesz je dodawać!

```
p.zagrajNaBebnie();
```

```
Perkusja p = new Perkusja();
```

```
boolean talerze = true;
boolean beben = true;
```

```
void zagrajNaBebnie() {
  System.out.println("bam, bam, baaaa-am-am");
}
```

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

```
if (p.beben == true) {
  p.zagrajNaBebnie();
}
```

```
p.beben = false;
```

```
class PerkusjaTester {
```

```
p.zagrajNaTalerzach();
```

```
class Perkusja {
```

```
void zagrajNaTalerzach() {
  System.out.println("brzdęk, brzrzrzdęęk");
}
```

Wiersz polecenia

```
H:>java PerkusjaTester
bam, bam, baaaa-am-am
brzdęk, brzrzrzdęęk
```

Zagadkowy basen

Twoim ***zadaniem*** jest wybranie fragmentów kodu z basenu i umieszczenie ich w miejscach kodu oznaczonych podkreśleniami. Każdy fragment kodu **może** być użyty więcej niż raz, a co więcej, nie wszystkie fragmenty zostaną wykorzystane. ***Zadanie*** polega na stworzeniu klasy, którą będzie można skompilować i która wygeneruje wyniki przedstawione poniżej. Nie daj się zwieść pozorom — ta zagadka jest trudniejsza, niż można by przypuszczać.

```
public class EchoTester {
  public static void main(String[] args) {
    Echo e1 = new Echo();

    _____________________________________
    int x = 0;
    while ( _____ ) {
      e1.witaj();

      ____________________
      if ( _______ ) {
        e2.ilosc = e2.ilosc + 1;
      }
      if ( ______ ) {
        e2.ilosc = e2.ilosc + e1.ilosc;
      }
      x = x + 1;
    }
    System.out.println(e2.ilosc);
  }
}
```

Wyniki

```
Wiersz polecenia

H:>java EchoTester
sieeeeemasz...
sieeeeemasz...
sieeeeemasz...
sieeeeemasz...
10
```

Dodatkowe pytanie!

Jak rozwiązałbyś zagadkę, gdyby w ostatnim wierszu wyników pojawiła się liczba **24**, a nie **10**?

```
class _____ {
  int ______ = 0;
  void _____ {
    System.out.println("sieeeeemasz...");
  }
}
```

Notatka: każdy fragment kodu z basenu może zostać użyty tylko raz!

```
x
y
e2
ilosc

x < 4
x < 5
x > 0
x > 1

Echo
Tester
echo( )
ilosc( )
witaj( )

e2 = e1;
Echo e2;
Echo e2 = e1;
Echo e2 = new Echo( );

x == 3
x == 4

e1 = e1 + 1;
e1 = ilosc + 1;
e1.ilosc = ilosc + 1;
e1.ilosc = e1.ilosc + 1;
```

→ Odpowiedź znajdziesz na stronie 47.

Kim jestem?

Grupa zamaskowanych komponentów Javy gra w grę towarzyską o nazwie „Zgadnij, kim jestem". Komponenty dają Ci podpowiedzi, a Ty na ich podstawie starasz się odgadnąć, kim one są. Załóż, że komponenty zawsze mówią prawdę. Jeśli mówią coś, co może być prawdą w odniesieniu do kilku z nich, wybierz wszystkie komponenty, dla których podane stwierdzenie jest prawdziwe. W pustych miejscach obok podanych podpowiedzi podaj nazwy komponentów biorących udział w zabawie. Odpowiedź na pierwszą podpowiedź podaliśmy sami.

W dzisiejszej zabawie udział biorą:

Klasa Metoda Obiekt Zmienna instancyjna

Jestem kompilowana na podstawie pliku .java. — **klasa**

Wartości moich zmiennych instancyjnych mogą się różnić od wartości zmiennych instancyjnych mojego brata bliźniaka. _____

Działam jak wzorzec.

Lubię działać. _____

Mogę mieć wiele metod. _____

Reprezentuję „stan". _____

Mam swoje „działanie". _____

Przebywam w obiektach. _____

Istnieję na stercie. _____

Służę do tworzenia kopii obiektów. _____

Mój stan może się zmieniać. _____

Deklaruję metody. _____

Mogę się zmieniać w trakcie działania programu. _____

→ Odpowiedź znajdziesz na stronie 47.

Ćwiczenie Rozwiązanie

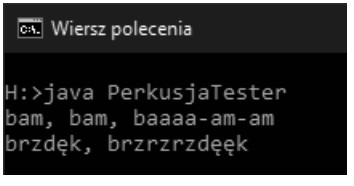
Magnesiki z kodem (ze strony 43)

```
class Perkusja {
  boolean talerze = true;
  boolean beben = true;

  void zagrajNaTalerzach() {
    System.out.println("brzdęk, brzrzrzdęęk");
  }
  void zagrajNaBebnie() {
    System.out.println("bam, bam, baaaa-am-am");
  }
}

class PerkusjaTester {
  public static void main(String[] args) {
    Perkusja p = new Perkusja();
    p.zagrajNaBebnie();
    p.beben = false;
    p.zagrajNaTalerzach();

    if (p.beben == true) {
      p.zagrajNaBebnie();
    }
  }
}
```



BĄDŹ kompilatorem (ze strony 42)

A

```
class StreamingUtworu {
  String tytul;
  String artysta;
  int czasTrwania;

  void odtworz() {
    System.out.println("Odtwarzam piosenkę");
  }

  void wyswietlSzczegoly() {
    System.out.println("To jest utwór "
      + tytul + " grany przez " + artysta);
  }
}
```

Mamy już szablon, teraz musimy stworzyć obiekt!

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

    StreamingUtworu utwor = new StreamingUtworu();
    utwor.artysta = "The Beatles";
    utwor.tytul = "Come Together";
    utwor.odtworz();
    utwor.wyswietlSzczegoly();
  }
}
```

```
class Odcinek {
  int numerSezonu;
  int numerOdcinka;

  void odtworz() {
    System.out.println(„Odtwarzam odcinek „ + numerOdcinka);
  }

  void pominIntro() {
    System.out.println("Pomijam intro...");
  }

  void przeskoczDoNastepnego() {
    System.out.println("Wczytuję następny odcinek...");
  }
}
```

Bez metody odtworz() w klasie Odcinek nie udałoby się skompilować wiersza: odcinek.odtworz().

B

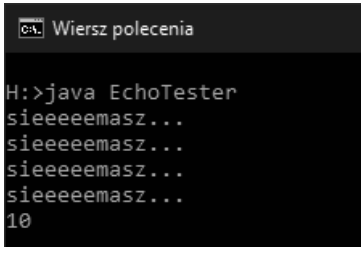
```
class OdcinekTester {
  public static void main(String[] args) {

    Odcinek odcinek = new Odcinek();
    odcinek.numerSezonu = 4;
    odcinek.odtworz();
    odcinek.pominIntro();
  }
}
```

Rozwiązania zagadek

Zagadkowy basen (ze strony 44)

```
public class EchoTester {
  public static void main(String[] args) {
    Echo e1 = new Echo();
    Echo e2 = new Echo(); // poprawna odpowiedź
    // -- lub --
    Echo e2 = e1; // odpowiedź na pytanie dodatkowe
    int x = 0;
    while (x < 4) {
      e1.witaj();
      e1.ilosc = e1.ilosc + 1;
      if (x == 3) {
        e2.ilosc = e2.ilosc + 1;
      }
      if (x > 0) {
        e2.ilosc = e2.ilosc + e1.ilosc;
      }
      x = x + 1;
    }
    System.out.println(e2.ilosc);
  }
}
class Echo {
  int ilosc = 0;
---------------------------------------
  void witaj() {
    System.out.println("sieeeeemasz... ");
  }
}
```



Kim jestem? (ze strony 45)

Jestem kompilowana na podstawie pliku .java.	klasa
Wartości moich zmiennych instancyjnych mogą się różnić od wartości zmiennych instancyjnych mojego brata bliźniaka.	obiekt
Działam jak wzorzec.	klasa
Lubię działać.	obiekt, metoda
Mogę mieć wiele metod.	klasa, obiekt
Reprezentuję „stan”.	zmienna instancyjna
Mam swoje „działanie”.	obiekt, klasa
Przebywam w obiektach.	metoda, zmienna instancyjna
Istnieję na stercie.	obiekt
Służę do tworzenia kopii obiektów.	klasa
Mój stan może się zmieniać.	obiekt, zmienna instancyjna
Deklaruję metody.	klasa
Mogę się zmieniać w trakcie działania programu.	obiekt, zmienna instancyjna

Notatka: mówi się, że zarówno klasy, jak i obiekty mają stan i działanie. Są one co prawda definiowane w klasie, jednak mówimy, że także obiekty je „posiadają”. Na razie nie obchodzi nas, gdzie się one znajdują z *technicznego* punktu widzenia.

Skorowidz

A

adnotacja, 684
  @FunctionalInterface, 392
adres IP
  klienta, 583
  serwera, 583, 593
aktualizacja wartości, 686
animacja, 486
  wykorzystująca klasę wewnętrzną, 488
anonimowa klasa
  wewnętrzna, 333
API
  dokumentacja, 156–158, 315
  klasy ArrayList, 310
  strumieni, Stream API, 365, 369, 371
aplikacja
  klienta, 590
  MuzMachina, 444, 449, 523
  serwera, 593, 594
argumenty, 74, 49
  formatowania, 298
  klasy bazowej, 186
  polimorficzne, 186, 354
  typu T, 337
  zmienna lista, 683
atomowy proces, 659
atrapa, mock, 306
atrybuty, 30
automatyczna konwersja, 288
  argumenty metod, 288
  operacje na liczbach, 289
  przypisania, 289
  wartości wynikowe, 288
  wyrażenia logiczne, 288
automatyczne opakowywanie, 287

B

bezpieczeństwo typów, 316
biblioteka, 123
  JavaSound, 419
  kolekcji Javy, Java Collections Framework, 305, 342
  Swing, 456, 503
    komponenty, 504
blok
  catch polimorficzny, 433
  finally, 429
  try-catch, 423, 427
    sterowanie przepływem, 428
blokada, 637–640, 659
bloki catch
  odrębne dla wyjątków, 433
  uporządkowanie, 434
blokowanie optymistyczne, optimistic locking, 648
błąd
  kompilacji, 117, 275, 276, 280, 355
  serializacji, 540
BorderLayout, 507, 516
  dodawanie przycisków, 508–511
BoxLayout, 507, 516
  komponenty jeden nad drugim, 515
bufor, 557

C

CSV, Comma Separated Values, 405
cykl życia obiektu, 233, 252, 256–262, 268, 484

D

dane MIDI, 442
definicja
  interfejsu, 222
  klasy, 12
  metody, 12
deklarowanie
  metody, 76, 142
  obiektu, 182
  tablicy, 19
  wyjątku, 425, 427, 437
  zmiennej, 39, 50, 62, 142, 238
    instancyjnej, 84
    iteracyjnej, 116
    referencyjnej, 55
    tablicowej, 59
    typu ogólnego, 317
deserializacja, 543, 548, 550, 572
dokumentacja
  API, 156–158, 315
  klas, 159
  klasy ArrayList, 318
  metody sort(), 334
dostęp
  modyfikatory, 681
  poziomy dostępu, 681
drzewo dziedziczenia, 168, 176
działanie
  animacji, 486
  obiektów, 71
  polimorfizmu, 204
dziedziczenie, 31, 41, 163–194
  metody publiczne klasy bazowej, 181
  unikanie powielania kodu, 180
  wielokrotne, 220, 221, 227

E

elementy
  kolekcji TreeSet, 350
  konstrukcyjne, 376
    dostosowywanie, 377

F

filtrowanie
  łańcucha, 397
  strumieni, 396–399
FlowLayout, 507, 516
  ciągłość ułożenia komponentów, 512
  dodanie dwóch przycisków, 514
  dodanie panelu, 512
  dodanie przycisku do panelu, 513
formatowanie liczb, 292
  dodawanie spacji, 294
  łańcuch formatujący, 295, 296
  modyfikator typu, 297
  specyfikator formatu, 296
  użycie
    instrukcji, 293
    spacji, 292
    wielu argumentów, 298
    znaków podkreślenia, 292
  znak procentu, %, 294, 295
funkcja, 225
  printf(), 292

G

generowanie
  dźwięków, 442
  liczby losowej, 111
gniazda sieciowe, sockets, 588, 606
gradient, 469
graficzny interfejs użytkownika,
  GUI, 455, 461, 463, 472
  komponenty, 464
  komponenty własne, 466
  przycisk w ramce, 457
  układy, 472, Patrz także menedżer układu
  tworzenie
    utworzenie ramki, 456
    utworzenie komponentu, 456
    dodanie komponentu do ramki, 456
    wyświetlenie ramki, 456
  wyświetlanie elementów, 465
  zdarzenia, 458
  źródła zdarzeń, 460

H

hermetyzacja, 80, 82, 87
hierarchia
  dziedziczenia, 220
  klas, 172

I

identyfikacja interfejsów funkcyjnych, 392
identyfikator serialVersionUID, 549
implementacja
  interfejsu, 222, 224
    Comparable, 325
    Runnable, 605
  metod, 142, 143
  metod abstrakcyjnych, 202
  wyrażeń lambda, 393
import statyczny, 299
inicjalizacja zmiennych
  instancyjnych, 84
  statycznych, 279
inicjalizator statyczny, 280
instancja, 34
instrukcja, Patrz słowo kluczowe
instrukcje, 7, 12, 14
  formatowania, 293
  if i switch z typami wyliczeniowymi, 689
  MIDI, 445
  try-operujące-na-zasobach, 567–569
interfejs, 222
  ActionListener, 461, 462
  Autoclosable, 569
  Comparable, 325–328, 386
  Comparator, 332
  ExecutorService, 607
  Iterable, 366
  List, 341
  Map, 341, 686
  MouseListener, 460
  Predicate, 398
  Runnable, 604, 605, 613
  Serializable, 539, 550
  Set, 341
  WindowListener, 460
interfejsy, 222
  funkcyjne, 337, 385, 390, 393
    adnotacja @FunctionalInterface, 392
    ActionListener, 485
  implementacja, 222
  metody
    abstrakcyjne, 392, 485
    domyślne, 392
    statyczne, 392
  odbiorcy zdarzeń, 460
  polimorfizm, 224
  SAM, 337
  tworzenie, 223
internet rzeczy, loT, 17

J

Java, 4
  API, 123, 154
  FX, 458
  NIO.2, 553
  REPL, 676
JavaSound API, 419
JShell, 676
JVM, Java Virtual Machine, 2–4

K

kanał, 595
  ServerSocketChannel, 595
  SocketChannel, 591, 593, 595
katalog, 678
klasa, 7–9, 31, 35, 41, 72
  ArrayList, 130, 310
    metody, 131
    pełna nazwa, 153
    porównanie z tablicą, 134, 135
  ArrayList<E>, 318
  Boolean, 290
  BufferedWriter, 569
  Collections, 310, 314, 327
  Collectors, 383, 405
  CopyOnWriteArrayList, 656, 657
  CountDownLatch, 617, 622
  Double, 290, 291
  Enum, 689
  Exception, 424, 434
  Executors, 607, 613, 622
  File, 556
  Graphics, 467
  Integer, 290
  JComponent, 504
  JFrame, 510
  JPanel, 466

Math, 111
metody, 272, 284
List, 310
MidiEvent, 445
Object, 206–208, 212
Random, 111
RuntimeException, 426
Socket, 588
SocketChannel, 583
Stream, 371, 373
String, 291
Thread, 602, 606, 613
Throwable, 424
TreeSet, 349
klasy, 7–9, 31, 35, 41, 72
abstrakcyjne, 198, 203, 208, 227
bazowe, 31, 41, 109, 164, 167
danych, 691
dokumentacja, 159
finalne, 281, 282
implementowanie interfejsów, 224
kod
przygotowawczy, 108, 109
testowy, 102, 103, 109
właściwy, 109
konkretne, 198, 200, 203
niezmienne, 659
odbiorców zdarzeń, 477
opakowujące, 286
pełna nazwa, 154
pomocnicze, 112
potomne, 31, 32, 164, 169, 178
rozszerzanie, 206
statyczne metody, 274
testujące, 107
tworzenie, 99
tworzenie hierarchii, 172
uogólnione, 317–320
wewnętrzne, 478, 482
tworzenie animacji, 486, 488
użycie składowej klasy zewnętrznej, 478
zmienne statyczne, 279
klient, 581, 595
pogawędek, 596, 600
kod
animacji, 488
aplikacji
CodziennePoradyKlient, 591
CodziennePoradySerwer, 594
KartaKwizowaEdytor, 554
klienta pogawędek, 596, 624
KwizGra, 560
Miniodtwarzacz3, 496
MuzMachina, 666
PomocnikGry, 150, 151
Roberta i Moniki, 634–636
serwera MuzMachina, 673
serwera pogawędek, 598
Startup, 148
StartupGraMax, 144, 146
mieszający, 345, 347
przygotowawczy, 99, 102, 109
klasy, 108, 142
metody, 128
testowy, 99, 109
dla klasy, 102, 103
właściwy, 99, 109
klasy, 106, 143
metody, 104
wynikowy, 2
źródłowy, 2, 3
kolekcja, 116, 305–357
ArrayList, 316, 319
HashSet, 343, 345
List, lista, 307, 341
Map, mapa, 341
Set, zbiór, 341
TreeSet, 348, 350
kolekcje, 116, 305–357
operacje, 370
kolektory, 374, 405
komentarze jednowierszowe, 12
komparator, 327
komparatory własne, 328
kompilator, 2, 10, 11
javac, 3
komponent, 464, 470
JCheckBox, 520
JList, 521
JTextArea, 518, 519
JTextField, 517
komponenty
biblioteki Swing, 504
interakcyjne, 504
tła, 504
zagnieżdżanie, 504
komunikat MIDI, 446
zmiana komunikatu, 447
konkatenacja obiektów String, 19
konstruktor
bezargumentowy, 243–247
domyślny, 239
inicjalizacja stanu obiektu, 241, 242
klasy bazowej, 249, 253
przeciążanie, 243, 245, 247
przesłanianie, 692
z argumentami, 242
kontrakt, 181, 188, 215, 216, 223
kontrola wersji, 548
konwersja
automatyczna, 288
liczby na łańcuch znaków, 291
łańcucha znaków, 290

L

lambdy, 685, 686
liczby
losowe, 19
zmiennoprzecinkowe, 51
lista, 204
ArrayList, 307, 308
do przechowywania obiektów, 205
List, 341

Ł

łańcuchy
formatujące, 295, 296
użycie metody split, 562

M

mapowanie typów, 401
mapy, 351, 685, 686
klucz i wartość, 351
Map, 341
maszyny wirtualne, 2
mechanizm zarządzający
wątkami, 611
menedżer układu, 505, 516
BorderLayout, 507–511, 515
BoxLayout, 507, 515, 516
FlowLayout, 507, 512–516
zasady rozmieszczania komponentów, 506

metoda, 7, 31, 34, 41, 72
abs(), 284
accept(), 593
actionPerformed(), 461
add(), 154
addActionListener(), 461
collect(), 374, 406
compare(), 329
compareTo(), 325–328
compute(), 686
computeIfAbsent(), 685
computeIfPresent(), 686
distinct(), 403, 405
draw3DRect(), 468
equals(), 86, 207, 337, 344–347, 692
fill3DRect(), 468
filter(), 399
finalna, 281
forEach(), 366–369, 383–385, 686
format(), 294
getClass(), 207
hashCode(), 207, 344–347, 692
indexOf(), 154
isEmpty(), 154
joining(), 405
limit(), 372, 374, 377
main(), 8, 12, 38, 110, 603
testowanie klas, 38
uruchamianie aplikacji, 38
map(), 400, 401, 402
max(), 285
merge(), 686
min(), 285
nextInt(), 19
open(), 583
paintComponent(), 466, 470, 489
parseBoolean(), 290
parseInt(), 290
print, 15
println, 15
random(), 111, 284
readObject(), 550
remove(), 154
repaint(), 487
replaceAll(), 686
rotate(), 468
round(), 285
run(), 604, 605
scale(), 468
setLayout(), 516
setMessage(), 446, 492
setRenderingHints(), 468
shear(), 468
shutdown(), 621, 622
shutdownNow(), 622
size(), 154
sleep(), 622
sort(), 310, 314, 321–328, 334
split(), 562, 563
sqrt(), 285
stream(), 372, 380, 383
toList(), 383, 405
toMap(), 383, 405
toSet(), 383, 405
toString(), 207, 291, 692
toUnmodifiableList(), 383, 405
toUnmodifiableMap(), 383, 405
toUnmodifiableSet(), 383, 405
transform(), 468
valueOf(), 291
writeObject(), 550
metody, 7, 31, 34, 41, 72
„get”, 691
abstrakcyjne, 201, 202, 227, 385, 392
implementowanie, 202
deklarowanie, 142
do tworzenia zdarzeń, 492
domyślne, 392
implementowanie, 142, 143
interfejsu, 223
klasy
ArrayList, 130, 131
bazowej, 226
Collectors, 383
Graphics2D, 470, 468
Math, 272, 284
Object, 207
potomnej, 226
Stream, 371
modyfikujące, 87
niejawne rozszerzanie wartości, 78
obsługi zdarzeń, 464
ogólne, 358
parametry, 76, 78
pobierające wyrażenie lambda, 390
prywatne, 179
przechwytujące wyjątki, 425, 431
przeciążone, 189
przekazywanie argumentów przez wartość, 76–78
przesłanianie, 32, 165
publiczne, 179
rekordów, 692
statyczne, 273-276, 282, 392, 641
używanie składowych niestatycznych, 275
wywoływanie metod niestatycznych, 276
stosowanie referencji, 404
strumieni, 372, 373
synchronizowane, 659
typów ogólnych, 320
typu void, 75
ustawiające, 79, 82, 87
wytwórcze, 352, 353, 607
wywoływanie, 274
z argumentami typów ogólnych, 317
ze zmienną listą argumentów, 683
zgłaszające wyjątek, 425, 427
zmienne lokalne, 49, 85
zwracające, 79, 82
zwracanie wartości, 75
modyfikacja drzewa klas, 216–220
modyfikator
abstract, 198
final, 280
private, 81, 247, 681
public, 81, 280, 681
static, 280
typu, 297

N

narzędzia do obsługi operacji wejścia-wyjścia, 553
nawiasy
kątowe, 154
klamrowe, 14, 223
kwadratowe, 296
nazwy
pakietów, 677
typów podstawowych, 286
zmiennych, 690
niezmienność
łańcuchów, 680
w klasach opakowujących, 680
NIO.2, 553, 564
numery portów, 584, 593–595

O

obiekt, 27, 35, 41, 141, 479
  ArrayList, 133, 139–141
    dynamiczna zmiana wielkości, 154
    używanie parametru typu, 154
  Event, 463
  ExecutorService, 607, 622
  File, 556, 563
  Graphics2D, 468, 469
  JFrame, 456, 470
  Math, 272
  Message, 446
  MidiEvent, 443, 445, 492
  Optional, 409
  Reader, 586, 591, 595
  Runnable, 604
  Sequence, 443
  Sequencer, 420, 443
  Thread, 601, 608, 613
  Track, 443
  Writer, 587
obiekty
  deklarowanie, 55, 182
    zmiennej, 238
    zmiennej referencyjnej, 55
  deserializacja, 543, 544
  działanie, 71, 531
  klasy
    abstrakcyjnej, 199
    wewnętrznej, 479, 480
    zewnętrznej, 479
  metody, 34, 49, 71
  na stercie, 536
  niezmienne, immutable objects, 650, 651
    stosowanie, 651
    zmienianie, 652
  niszczenie, 233, 258–261
  odwołania, 54, 56
  opakowujące, 408, 680
  połączenie
    obiektu i referencji, 182
    obiektu z odwołaniem, 238
  przechowywanie, 204, 205
  przypisywanie, 55, 182
  reprezentujące wartość, 286
  równe, 347
  serializacja, 536, 538
  stan, 531, 537
  tablic, 59
  tworzenie, 36, 55, 238
  w tablicy, 83
  wewnętrzne, 482, 483
  wyjątek jako obiekt, 424
  wyrażenia lambda, 385
  zapisywanie, 531, 548, 550, 564–568, 573
  zdarzenia, 460
  zmienne instancyjne, 34, 49, 71, 248
obsługa
  plików tekstowych, 552
  wyjątków, 417, 422, 431, 433, 439, 441
    blok try-catch, 423, 427
    kolejność bloków catch, 434
    osobne bloki catch, 433
    polimorficzny blok catch, 433
    rezygnacja poprzez zadeklarowanie, 437–439
    uporządkowanie bloków catch, 435
  zdarzeń, 459
odbieranie zdarzeń ControllerEvent, 494
odbiorca zdarzeń, 460–463, 470
  panel, 495
  rejestracja, 494, 475
  z wielu źródeł, 474–476
odczyt danych z serwera, 586
odczytywanie plików, 558
odśmiecacz, 40, 233
odwołania, 62
  „o”, 211
  do obiektów, 54–58, 258
  porównywanie, 86
  typu Object, 209, 211, 213
opakowywanie, 680
  Optional, 408–410
  wartości typów podstawowych, 286, 287
operacje
  atomowe, 638
  CAS, 647, 648
  końcowe, terminal operations, 373, 379, 383, 406
  leniwe, 378
  na kolekcjach, 370
  na plikach, 531–571
  na strumieniach, 375
  pośrednie, intermediate operations, 372
  wejścia-wyjścia na plikach, 533–577
    bufory, 557
operator, 105
  alternatywy, 39, 149
  dekrementacji, 109, 115
  diamentowy, <>, 308, 690
  inkrementacji, 109
  instanceof, 214
  koniunkcji, 149
  konkatenacji, 19, 291
  kropki, 36, 54, 80
  new, 239, 248, 249
  postinkrementacji, 115
  preinkrementacji, 115
  przypisania, 14
  równości, 14, 86, 87
operatory
  pierwszeństwo, 149
  przetwarzania pełnego, 149
  przetwarzania skróconego, 149

P

pakiet, 152, 677
  java.awt, 473
  java.awt.event, 460, 473
  java.lang, 152, 154
  java.net.Socket, 588
  java.nio, 589
  java.nio.channels, 589
  java.nio.file, 556, 564, 565
  java.util, 152, 310
  java.util.concurrent, 607, 659
  javax.sound.midi, 420
  javax.swing, 152, 473, 456
pakiety
  dodawanie własnych klas, 678
  kompilacja i uruchamianie programu, 679
parametr, 74
  typu, 319, 320
  typu T, 358
  typu ogólnego, 358
pętla
  for, 105, 106, 114, 366–369

pętla
  for each, Patrz pętla for rozszerzona
  for rozszerzona, 116, 109
  while, 13, 14, 39, 115
pisanie programu, 109
pliki
  .aif, 28
  .class, 9, 678
  .csv, 405
  .gif, 470
  .java, 9
  .jpeg, 467
  .jpg, 470
  .mp3, 29
  .txt, 551, 552, 558
  MIDI, 419
  odczytywanie, 558
  zapisywanie, 556
pobieranie danych wejściowych, 111
polimorficzne odwołania typu Object, 209
polimorfizm, 163–231, 356, 479
  interfejsów, 224
połączenia sieciowe, 582
  nawiązywanie połączenia, 582, 583, 587, 590
  odczytywanie danych z serwera, 586
  odczytywanie wiadomości, 582, 590
  stosowanie gniazd, 588
  stosowanie kanałów, 589
  wysyłanie wiadomości, 582
poprawianie błędu programu, 126–128, 136
porównywanie
  ArrayList z tablicą, 134, 135
  odwołań, 86
  zmiennych typów podstawowych, 86
port TCP, 584, 595
porty
  numery, 584, 593–595
potok strumienia, stream pipeline, 375, 376, 383, 403
  tworzenie, 380
  złożony, 377
poziom
  chroniony, 682
  domyślny, 682
  prywatny, 681
  publiczny, 681
problem „utraconej modyfikacji”, 642
procedura obracająca figury, 30
programowanie
  obiektowe, 33
  w oparciu o testy, TDD, 101
  współbieżne, 659
projektowanie dziedziczenia, 167–173, 177
przechowywanie obiektów, 204, 205
przechwytywanie
  wielu wyjątków, 431
  zdarzeń, 459, 462
przeciążanie
  konstruktorów, 245, 247
  metod, 189
przekazywanie
  kolekcji, 354, 355, 356
  przez wartość, 77
  zachowań, 384
przekształcanie elementów, 400
przesłanianie
  konstruktorów, 692
  metod, 32, 165, 170, 188
    hashCode() i equals(), 346
    odziedziczonych, 179
przetwarzanie łańcuchów znaków, 562
przypisywanie
  zmiennej, 52
  obiektu, 182

R

ramka
  regiony, 472
  umieszczanie komponentów, 472
referencje
  metod, 404
  wprzód, forward references, 676
reguły przesłaniania, 188
rejestracja odbiorcy zdarzeń, 475, 494
rekordy, 691, 692
relacja
  JEST, 175–179, 324
  MA, 175
REPL, 676
rozgałęzienia warunkowe, 15
rozszerzanie klasy, 179, 206
równoległość, 687
równość
  obiektów, 344
  referencji, 344
rysowanie, 466, 467
  w takt muzyki, 495
rzutowanie, 111, 117, 227
  odwołania do obiektu, 214

S

SAM, single abstract method, 337
schemat blokowy, 97
sekwenser, 420, 442
serializacja, 531–571
  kontrola wersji, 548
  obiektów, 536–542, 550
  zastosowanie, 546
serwer, 581, 595
  pogawędek, 598
  Telnet, 584
  WWW, 584
Set, zbiór, 341
składowe
  prywatne, 178
  publiczne, 178
  statyczne, 271–278, 282
słowo kluczowe, 53
  abstract, 198, 227
  break, 105, 109
  class, 227
  enum, 689
  extends, 206, 251, 324
  final, 280–283, 659
  finally, 429, 566, 567
  if, 15, 689
  implements, 227
  import, 153, 155
  instanceof, 214
  interface, 222, 227
  new, 239
  package, 678
  private, 247
  public, 280
  return, 386, 387
  static, 273, 280, 283
  super, 178, 226
  synchronized, 638, 640, 647, 659
  throw, 427
  throws, 422
  transient, 541, 550
  try, 567–569
  var, 690
  void, 75, 78

sortowanie, 310, 334
  listy, 306, 307, 322, 331, 327
  przy użyciu wyrażeń lambda, 338
  strumienia, 376
  usuwanie powtórzeń, 340
  własnych obiektów, 312
  z wykorzystaniem komparatorów, 332
  zbioru, 348
specyfikator formatu, 296
stan
  obiektu, 537
  wątku, 608
statyczne zmienne finalne, 280
sterta, 40, 236
  automatyczne odśmiecanie, 40
  obiekty, 234
  zmienne instancyjne, 237, 245
stos
  metody, 234-235
  zmienne lokalne, 234, 237
  zmienne przechowujące odwołania do obiektów, 236
stos wywołań, 602–606
stosowanie
  biblioteki Swing, 503
  buforów, 557
  dziedziczenia, 169, 171
  komponentu JTextArea, 519
  mapy, 351
  niezmiennych obiektów, 651
  polimorfizmu, 183
  rekordów, 691
  serializacji, 546
  strumieni, 380
  wielu odbiorców zdarzeń, 474, 476
  wyjątków, 440
struktury danych, 305
  współbieżne, 656
strumienie, 365–415
  elementy konstrukcyjne, 375–377
  filtrowanie, 396–399
  jednokrotne użycie, 380
  kolektory, 405
  lista wyników, 379
  łańcuchowe, 535, 550
  metoda sorted(), 377
  niezmienialna kolekcja źródłowa, 380, 381
  operacje, 370, 375
    końcowe, 373, 379, 383, 406
    leniwe, 378
    pośrednie, 372, 375, 383
  pobieranie wyników, 374
  połączeniowe, 535, 550
  potok, 376, 383
  potoki złożone, 377
  równoległe, 687
  składanie potoków, 403
  sortowanie, 376
  usuwanie powtórzeń, 402
  użycie metody
    distinct(), 403
    joining(), 405
    map(), 401
  zasady stosowania, 380
strumień
  BufferedReader, 563, 586, 595
  BufferedWriter, 563
  FileInputStream, 543
  FileOutputStream, 550
  FileReader, 563
  FileWriter, 563
  ObjectInputStream, 543
  ObjectOutputStream, 550
  PrintWriter, 587
  System.out, 587
synchronizacja metod, 638–640, 642, 644
syntezator, 419

T

tablice, 59, 62
  ArrayList<Object>, 209, 213, 227, 287
  łańcuchów znaków, 19
  obiektów, 60, 83
  tworzenie, 19
TDD, Test-Driven Development, 101
technologia MIDI, 419
test logiczny, 13, 15
tworzenie
  animacji, 486
  graficznego interfejsu użytkownika, GUI, 455–501
  hierarchii klas, 172
  interfejsu, 223
  klasy, 9, 99, 109, 138, 139
  kolekcji, 352
  komunikatu MIDI, 446
  listy, 353
  mapy, 353
  obiektu, 35, 55, 182, 238, 250
    klasy wewnętrznej, 480
    reprezentującego wartość, 286
  potoku strumienia, 380
  tablicy, 19, 59
  tablicy obiektów, 60, 83
  wątku, 601
  własnego komponentu, 466
  zadania dla wątku, 605
  zbioru, 353
  zdarzeń, 492, 493
typ
  „E”, 318
  Function, 401
  Optional, 406, 408, 410
  T, 337, 358
  void, 78
  wyliczeniowy, 688
typy
  ogólne, 305–357, 371, 373, 401
    AbstractList<E>, 318
    deklarowanie zmiennych, 317
  podstawowe, 49, 51, 286

U

układy GUI, 472
unikanie powielania kodu, 169, 180
uporządkowanie naturalne, 311
uruchamianie
  pojedynczego zadania, 607
  wątku, 606, 607
  wielu wątków, 618, 620
usuwanie powtórzeń, 340, 402
użycie
  dziedziczenia, 179
  interfejsu, 225
  klasy, 225

W

wartości
  liczbowe, 51
  logiczne, 51
  odwołań, 62
  zmiennych typów podstawowych, 62

wartość
  false, 14
  null, 58, 62, 258, 261
  true, 14
wątki, 601, 613
  chwilowo zablokowane, 609
  główne, 602, 603
  mechanizm zarządzający, 610–613
  odliczanie, 617
  pętla stanów, 609
  pula wątków, 619, 621
  robocze, 603
  stany, 608, 613
  synchronizacja działania, 617, 647
  uruchamianie wielu wątków, 618, 620
  usypianie, 614–616
  współzawodnictwo, 645–653
  wzajemna blokada, 646
  zamykanie, 621
wielowątkowość, 601, 622, 656–659, 687
wiersz poleceń
  przekazywanie argumentów, 448
wirtualna maszyna Javy, JVM, 2, 3, 10, 11
wnioskowanie typów, type inference, 308
  zmiennych lokalnych, 690
wskaźniki do obiektów, 54
współbieżne struktury danych, 656
współbieżność, 631, 642
współużytkowanie danych, 654
wyjątek, 417, 422
  ClassCastException, 11
  ConcurrentModificationException, 655
  MidiUnavailableException, 422, 440
  NullPointerException, 149
  NumberFormatException, 290
wyjątki
  deklarowanie, 425, 427, 437
  klasa bazowa, 432
  klasy RuntimeException, 426, 427
  niesprawdzane, 426, 427
  polimorficzne, 432
  przechwytywanie, 425
  zasady stosowania, 440
  zgłaszanie, 425
wyliczenia, 688, 689
wyrażenia logiczne, 149
wyrażenie lambda, lambda expression, 336, 365–415, 484
  anatomia, 387
  dowolna liczba parametrów, 389
  implementująca interfejs funkcyjny, 385
    Comparable, 386
  jednowierszowa, 388
  niczego nie zwracająca, 389
  postać, 386–388
  używana przy sortowaniu, 338
  wielowierszowa, 388
wyświetlanie plików JPEG, 467
wywołanie
  konstruktora, 239, 251, 254
  metody, 38, 39, 72, 76, 274
  metod klasy Object, 227
  super(), 251–253
  this(), 254
wzorzec, 225

Z

zapisywanie
  łańcucha znaków, 551
  obiektów, 531, 548, 550, 564–568, 573
  plików, 552, 556
  serializowanego obiektu, 534
  stanu obiektu, 533, 537, 550
zarządzanie wykonywaniem wątków, 610–612
zbiór, Set, 341
zdarzenia
  interfejsu odbiorcy, 460
  interfejsu użytkownika, 458
  przechowywanie danych, 463
  przechwytywanie, 459, 462, 463
  tworzenie, 492, 493
  wysyłanie, 463
  z dwóch źródeł, 475
  źródło zdarzeń, 460
zdarzenie
  ActionEvent, 461, 462, 470
  ControllerEvent, 491, 494
  MIDI, 442, 491
  MidiEvent, 445, 446
  MouseEvent, 460
  NOTE ON, 491
  WindowEvent, 460
zmienianie kontraktu, 216
zmienne
  atomowe, 647, 648
  całkowite, 14
  deklaracja, 12, 49, 50, 62
  finalne, 281
  iteracyjne, 116
  instancyjne, 34, 41, 72, 154, 234–237, 262
    deklarowanie, 84
    inicjalizacja, 84
    jako prywatne, 81, 82, 87
    wartości domyślne, 87
  lista argumentów, 683
  lokalne, 49, 74, 234, 236, 262
    błąd serializacji, 550
    czas istnienia, 257
    finalne, 282
    inicjalizacja, 85, 279
    zasięg, 257
  nazwa, 53, 62
  porównywanie, 86
  przechowujące odwołania do obiektów, 236
  przypisywanie wartości, 52
  referencyjne, 54, 62
    deklarowanie, 55
    typu bazowego, 185
    typu Object, 227
  statyczne, 277–282, 300, 301
  typ, 53
  typów podstawowych, 62
znak
  procentu, %, 294, 295
  strzałki, ->, 384
  wieloznaczny, ?, 357, 358

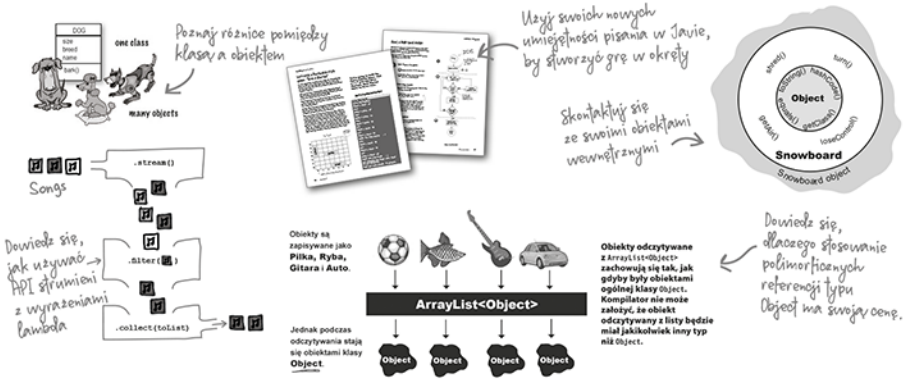
Ź

źródło zdarzeń, 460–463, 470

Co jest takiego wyjątkowego w tej książce?

Od momentu pojawienia się pierwszej wersji Java pociągała programistów ze względu na przyjazną składnię, cechy obiektowe, zarządzanie pamięcią, a przede wszystkim obietnicę przenośności. Mijają lata, a Java wciąż cieszy się ogromną popularnością, jest sukcesywnie rozwijana i używana do coraz to nowszych zastosowań. Wszystko wskazuje na to, że jeszcze długo programiści biegle posługujący się tym językiem będą mieli pełne ręce roboty.

Ta książka, podobnie jak inne pozycje z serii **Rusz głową!**, została przygotowana zgodnie z jedyną w swoim rodzaju metodyką nauczania, wykorzystującą zasady funkcjonowania ludzkiego mózgu. Dzięki zagadkom do rozwiązania, zabawnym ćwiczeniom i przystępnie podanej wiedzy bez trudu przyswoisz nawet dość złożone koncepcje, takie jak wyrażenia lambda, typy ogólne czy programowanie sieciowe i funkcyjne. Znajdziesz tu zabawne i niekonwencjonalne ilustracje, świetne analogie, pogawędki prowadzone przy kominku przez programistę i kompilator. To wszystko sprawia, że ta pozycja jest absolutnie wyjątkowym i niezwykle skutecznym podręcznikiem!



To najbardziej zajmująca książka do nauki programowania, jaką kiedykolwiek widziałam!

– Angie Jones
Java Champion

Technologie bazujące na Javie są wszędzie. Jeśli jesteś programistą i jeszcze nie znaszJavy, to zdecydowanie czas ją poznać, najlepiej z książką *Java. Rusz głową!*

– Scott McNealy
były wiceprezes,
prezes Sun Microsystems

Kathy Sierra od wielu lat zajmuje się teorią nauczania. Pracowała na Uniwersytecie Kalifornijskim i w Sun Microsystems. W 2015 roku otrzymała Pioneer Award – nagrodę przyznawaną przez Electronic Frontier Foundation.

Bert Bates był programistą systemów eksperckich i systemów operacyjnych czasu rzeczywistego. Od 2003 roku pisze książki, jest autorem licznych pozycji z serii *Rusz głową!*

Trisha Gee jest związana z Javą od 1997 roku, pisała w tym języku aplikacje dla różnych branż. Przez siedem lat była członkiem zespołu Java Advocacy w firmie JetBrains.

Helion
helion.pl
HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl
KOD KORZYŚCI
Sięgnij po więcej!
ISBN 978-83-283-9984-6
9 788328 399846
Cena: 149,00 zł