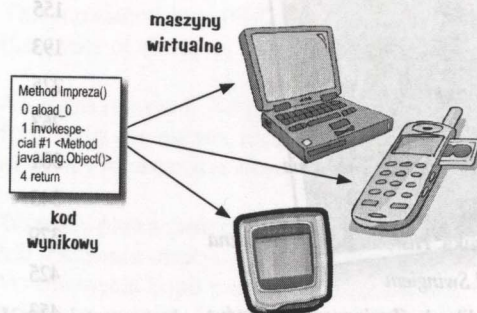


1 Przełamując zalew początkowych trudności

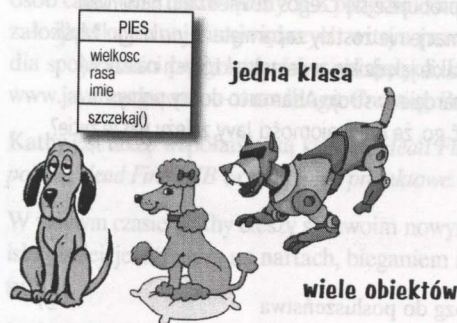
Java zabiera nas w nowe miejsca. Od momentu pojawienia się pierwszej, skromnej wersji o numerze 1.02 Java pociągała programistów ze względu na przyjazną składnię, cechy obiektowe, zarządzanie pamięcią, a przede wszystkim obietnicę przenośności. Od samego początku „wskoczmy na głęboką wodę” — napiszemy prosty program, skompilujemy go i wykonamy. Porozmawiamy o składni, pętlach, rozgałęzieniach oraz innych czynnikach, które sprawiają, że Java jest taka fajna. Zatem wskakuj!



Jak działa Java?	34
Struktura kodu w Javie	39
Anatomia klasy	40
Tworzenie klasy z metodą main	41
Pętle i pętle i...	43
Przykłady pętli while	44
Rozgałęzienia warunkowe	45
Tworzenie poważnej aplikacji biznesowej	46
Program krasomówczy	49

2 Wycieczka do Obiektowa

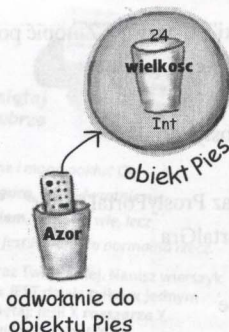
Mówiono mi, że będę tam same obiekty. W rozdziale 1. cały kod programu został umieszczony w metodzie `main()`. Nie jest to w pełni zgodne z zasadami programowania obiektowego. A zatem trzeba porzucić świat programowania proceduralnego i rozpocząć tworzenie własnych obiektów. Zobaczmy, co sprawia, że programowanie obiektowe w Javie jest takie fajne. Wyjaśnimy także, na czym polega różnica pomiędzy klasą a obiektem. W końcu pokażemy, w jaki sposób obiekty mogą ułatwić nam życie.



Wojna o hotel (albo Jak Obiekty Mogą Zmienić Twoje Życie)	60
Na plaży na laptopie Jurka	61
O tym beztrudno zapomniano napisać w specyfikacji	62
A co z metodą <code>obroc()</code> dla „ameby”?	64
Ta niepewność mnie zabije! Kto wygra Superhotel?	65
Tworzenie pierwszego obiektu	68
Tworzenie i testowanie obiektów Film	69
Szybko! Opuszczamy metodę <code>main()</code>	70

3 Poznaj swoje zmienne

Istnieją dwa rodzaje zmiennych: **zmienne typów podstawowych oraz odwołania**. W programach będą istnieć także inne typy danych niż jedynie liczby całkowite, łańcuchy znaków i tablice. Co zrobić, jeśli chcielibyśmy stworzyć obiekt **WłaścicielZwierzaka** zawierający składową **Pies**? Albo **Pojazd** ze składową **Silnik**? W tym rozdziale ujawnimy tajemnice typów danych w Javie oraz pokażemy, co można **zadeklarować** jako zmienną, **zapisać** w zmiennej oraz co z taką zmienną można potem zrobić. W końcu przekonamy się, jak wygląda życie na automatycznie porządkowanej stercie.

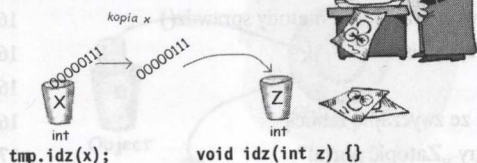


Deklarowanie zmiennej	82
„Proszę podwójną. Albo nie — całkowitą!”	83
Naprawdę nie chcesz niczego rozsypywać	84
Tabela słów zarezerwowanych	85
Odwołanie do obiektu to jedynie inna wartość zmiennej	87
Życie na odśmiecanej stercie	89
Tablice także są obiektami	91
Tworzymy tablicę obiektów Pies	92
Przykładowy obiekt Pies	94

4 Jak działają obiekty?

Stan wpływa na działanie, a działanie wpływa na stan. Wiemy, że obiekty mają swój **stan** i określone **działanie** reprezentowane odpowiednio przez **składowe** i **metody**. Teraz sprawdzimy, w jaki sposób stan i działanie obiektów są ze sobą **powiązane**. Otóż w swych działaniach obiekty wykorzystują swój unikalny stan. Innymi słowy, **metody wykorzystują wartości składowych obiektów**. Na przykład: „Jeśli pies waży mniej niż 20 kilogramów, szczekamy radośnie, w przeciwnym przypadku...” **Spróbujmy zmienić stan obiektu!**

Java przekazuje argumenty przez **wartość**.
To oznacza, że przekazywana jest **kopia**.



Wielkość ma wpływ na sposób szczekania	105
Do metod można przekazywać informacje	106
Metoda może coś zwrócić	107
Java przekazuje argumenty przez wartość	109
Ciekawe rozwiązania wykorzystujące parametry i wartości wynikowe	111
Hermetyzacja	112
Ukryj dane	113
Jak zachowują się obiekty w tablicy?	115
Deklarowanie i inicjalizacja składowych	116
Różnica pomiędzy składowymi a zmiennymi lokalnymi	117
Porównywanie zmiennych (typów podstawowych oraz odwołań)	118

5

Supermocne metody

Dodajmy naszym metodom nieco siły. Zajmowaliśmy się już zmiennymi, eksperymentowaliśmy z kilkoma obiektami i napisaliśmy kod paru programów. Jednak potrzeba nam więcej narzędzi. Na przykład **operatorów**. I **pętli**. Może przydałoby się **wygenerować kilka liczb losowych** i **zamienić łańcuch znaków na liczbę całkowitą**. O tak, to byłoby super. I dlaczego nie nauczyć się tego wszystkiego, **tworząc coś rzeczywistego**, by przekonać się, jak wygląda pisanie (i testowanie) programu w Javie od samego początku do końca. **Może jakąś grę**, taką jak „Zatopić portal” (podobną do gry w okręty).

Stworzmy grę „Zatopić portal!”

A							
B							
C	Gb2.com						
D				www.onet.pl			
E							
F							
G					www.wp.pl		
	0	1	2	3	4	5	6

Napiszmy grę przypominającą „statki”, o nazwie „Zatopić portal”	128
Łagodne wprowadzenie do prostszej wersji gry	130
Pisanie implementacji metod	133
Pisanie kodu testowego dla klasy ProstyPortal	134
Kod testowy dla klasy ProstyPortal	135
Ostateczny kod klas ProstyPortal oraz ProstyPortalTester	138
Kod przygotowawczy klasy ProstyPortalGra	140
Trochę więcej o pętlach for	146
Różnica pomiędzy pętlami for i while	147
Rozszerzone pętle	148
Rzutowanie wartości typów podstawowych	149

6

Korzystanie z biblioteki Javy

Java jest wyposażona w setki gotowych klas. Jeśli tylko dowiesz się, jak znaleźć w bibliotece Javy (nazywanej także **Java API**) to, czego szukasz, nie będziesz musiał ponownie wymyślać koła. *W końcu masz ciekawsze rzeczy do roboty.* Jeśli masz zamiar napisać program, to równie dobrze możesz napisać *tylko* te jego fragmenty, które są unikalne. Standardowa biblioteka Javy to gigantyczny zbiór klas, które tylko czekają, aby użyć ich jako klocków przy tworzeniu programów.

„Dobrze jest 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	156
Oryginalny kod przygotowawczy fragmentu metody sprawdz()	160
Niektóre możliwości klasy ArrayList	163
ArrayList	164
Porównanie klasy ArrayList ze zwykłą tablicą	167
Napiszmy właściwą wersję gry „Zatopić portal”	170
Co (i kiedy) robią poszczególne obiekty w grze?	172
Kod przygotowawczy właściwej klasy PortalGraMax	174
Wyrażenia logiczne o bardzo dużych możliwościach	181
Stosowanie biblioteki (Java API)	184
Jak poznać API?	188

7

Wygodniejsze życie w Obiekcie

Planuj swoje programy, myśląc o przyszłości. A gdyby tak można tworzyć kod, który inni programiści mogliby rozbudowywać, i to w prosty sposób? A gdyby tak można było tworzyć elastyczny kod z myślą o tych najnowszych zmianach wprowadzanych w specyfikacji? Kiedy dowiesz się o Planie Polimorfizmu, poznasz 5 kroków służących projektowaniu lepszych klas, 3 sztuczki z polimorfizmem oraz 8 sposobów tworzenia elastycznego kodu, a jeśli zaczniesz już teraz, dodatkowo otrzymasz 4 odpowiedzi odnośnie do wykorzystywania dziedziczenia.



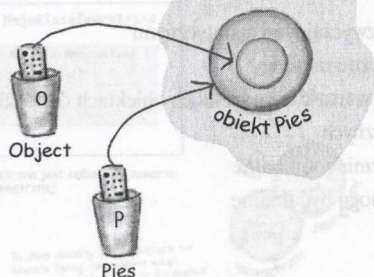
Zrozumienie dziedziczenia	196
Przykład dziedziczenia	197
Jakie metody należy przesłonić?	200
Jaka metoda jest wywoływana?	203
Projektowanie drzewa dziedziczenia	204
Ale poczekaj! To jeszcze nie wszystko!	206
Jak możesz określić, czy dobrze zaprojektowałeś hierarchię dziedziczenia?	207
Czy korzystanie z dziedziczenia przy projektowaniu klas jest „używaniem”, czy „nadużywaniem”?	209
Co w rzeczywistości daje nam dziedziczenie?	210
Jak dotrzymać kontraktu — reguły przesłaniania	218
Przeciążanie metody	219

8

Poważny polimorfizm

Dziedziczenie to tylko początek. Aby w pełni wykorzystać polimorfizm, niezbędne nam będą interfejsy. Musimy pójść dalej, poza proste dziedziczenie, i uzyskać elastyczność, jaką może zapewnić wyłącznie projektowanie i kodowanie z wykorzystaniem interfejsów. Czym jest interfejs? W 100% abstrakcyjną klasą. A czym jest klasa abstrakcyjna? To klasa, która nie pozwala na tworzenie obiektów. A co nam po takiej klasie? Przeczytaj, to się dowiesz...

```
Object o = lista.get(indeks);
Pies p = (Pies) o;
p.szczekaj();
```



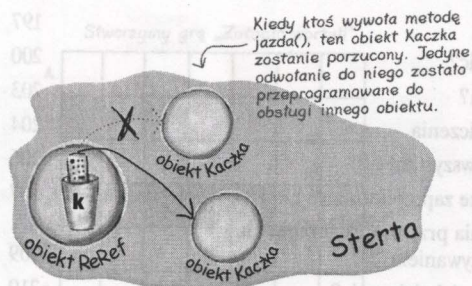
Abstrakcyjne kontra konkretne	230
Metody abstrakcyjne	231
Polimorfizm w działaniu	234
Czym jest „superultramegaklasa” Object?	237
Stosowanie polimorficznych odwołań typu Object ma swoją cenę...	239
Kiedy Pies nie zachowuje się jak Pies	240
Object nie szczeka	241
Połącz się ze swoim wewnętrznym Object-em	242
A co, jeśli musimy zmienić kontrakt?	246
Na pomoc spieszą interfejsy!	252



9

Życie i śmierć obiektu

Obiekty się rodzą i umierają. To Ty jesteś za to odpowiedzialny. To Ty decydujesz, kiedy i jak *skonstruować* obiekt. Ty także decydujesz, kiedy go *porzucić*. **Odśmiecacz pamięci** odzyska pamięć zajmowaną przez niepotrzebne obiekty. Przyjrzymy się, w jaki sposób obiekty są tworzone, gdzie „żyją” i w jaki sposób efektywnie je przechowywać lub porzucać. Oznacza to, że będziemy dyskutować o stercie, stosie, zasięgach, konstruktorach, konstruktorach nadrzędnych, odwołaniach pustych oraz przydatności odśmiecacza pamięci.



W składowej „k” jest zapisywany nowy obiekt Kaczka, przez co oryginalny (pierwszy) obiekt zostaje porzucony. Teraz ten pierwszy obiekt do niczego się nam już nie przyda.

Stos i sterta. Gdzie są przechowywane informacje?	264
Metody są zapisywane na stosie	265
A co ze zmiennymi lokalnymi, które są obiektami?	266
Jeśli zmienne lokalne są przechowywane na stosie, to gdzie są przechowywane składowe?	267
Cud utworzenia obiektu	268
Tworzenie obiektu Kaczka	270
Inicjalizacja stanu nowego obiektu Kaczka	271
Nanoprzegląd. Cztery rzeczy o konstruktorach, które należy zapamiętać	277
Znaczenie konstruktorów klasy bazowej w życiu obiektu	279
Konstruktory klas bazowych pobierające argumenty	283
Wywoływanie jednego przeciążonego konstruktora z poziomu innego	284

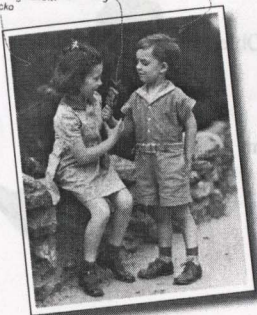
10

Liczby mają znaczenie

Zabaw się w matematyka. Java API udostępnia metody do wyznaczania wartości bezwzględnej, zaokrąglania liczb, określania wartości minimalnej i maksymalnej i tak dalej. A co z formatowaniem? Moglibyśmy chcieć wyświetlać liczby ze znakiem dolara na początku i dwoma miejscami dziesiętnymi. Albo wyświetlić daną w formie daty. Albo daty w zapisie stosowanym w Anglii. A co z przekształcaniem łańcucha znaków na liczbę? Lub zamianą liczby na łańcuch znaków? Zaczniemy jednak od wyjaśnienia, co oznacza, że zmienne lub właściwości są *statyczne*.

Zmienne statyczne są współdzielone przez wszystkie kopie klasy

składowa statyczna — lody
pierwszy obiekt — dziecko
drugi obiekt — dziecko



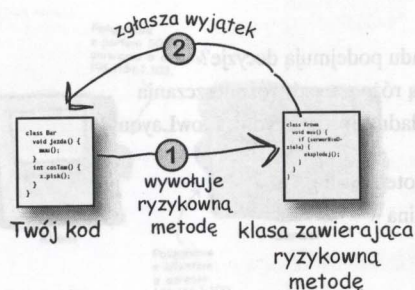
Składowe — po jednej w każdym obiekcie.
Składowe statyczne — jedna dla całej klasy.

Metody klasy Math — najlepsze z możliwych odpowiedników metod globalnych	302
Różnice pomiędzy metodami zwyczajnymi a statycznymi	303
Co oznacza, że klasa ma statyczne metody	304
Składowa statyczna — ta sama wartość we wszystkich obiektach danej klasy	307
Inicjalizacja składowych statycznych	309
Rolę stałych pełnią statyczne zmienne finalne	310
Nie tylko zmienne statyczne mogą być finalne	311
Formatowanie liczb	322
Specyfikator formatu	326
Operacje na datach (Java API)	330
Stosowanie obiektów Calendar	333
Wybrane możliwości API klasy Calendar	334

11

Ryzykowne działania

Czasami zdarzają się nieprzewidziane sytuacje. Pliku nie ma tam, gdzie powinien być. Serwer został wyłączony. Niezależnie od tego, jak dobrym jesteś programistą, nie jesteś w stanie kontrolować *wszystkiego*. Kiedy stworzysz metodę, której działanie jest opatrzone ryzykiem niepowodzenia, musisz także stworzyć kod, który obsłuży potencjalnie niebezpieczną sytuację. Ale skąd wiadomo, że metoda może nieść ze sobą potencjalne niebezpieczeństwo? Gdzie umieszczać kod *obsługujący wyjątkowe* sytuacje? W tym rozdziale stworzymy odtwarzacz MIDI wykorzystujący ryzykowny interfejs programistyczny JavaSound, zatem lepiej dowiedzmy się, jak zabezpieczyć się przed potencjalnymi niebezpieczeństwami.



Stwórzmy program MuzMachina	344
JavaSound API	345
Wyjątek jest obiektem... klasy Exception	350
Finally — blok kodu, który musi zostać wykonany niezależnie od wszystkiego	355
Przechwytywanie wielu wyjątków	357
Reguły związane ze stosowaniem wyjątków	366
Generowanie dźwięków	368
Twój pierwszy odtwarzacz muzyki	370
Tworzenie obiektów MidiEvent (danych piosenki)	371
Komunikat MIDI — serce zdarzenia MidiEvent	372
Zmiana komunikatu	373

12

Historia bardzo graficzna

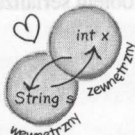
Musisz się z tym pogodzić — tworzenie interfejsów graficznych jest konieczne.

Nawet jeśli wierzysz, że już do końca życia będziesz pisać programy działające na serwerze, to jednak będziesz także musiał pisać programy narzędziowe i zapewne będziesz chciał, aby miały one jakiś interfejs graficzny. Zagadnieniom interfejsu graficznego poświęcimy dwa rozdziały, w których przedstawimy także inne cechy języka, takie jak obsługa zdarzeń oraz klasy wewnętrzne. Naciśniemy przycisk na ekranie, wskażemy coś myszą, wyświetlimy obraz zapisany w formacie JPEG, a nawet stworzymy małą animację.

```
class MojaKlasaZewnetrzna {
    class MojaKlasaWewnetrzna {
        void doDziela() {
        }
    }
}
```

Klasa wewnętrzna jest całkowicie zawarta w klasie zewnętrznej

Ta dwa obiekty przebywające na ofercie łączy szczególna więź. Obiekt wewnętrzny może korzystać ze składowych obiektu zewnętrznego (i na odwrót).



Wszystko zaczyna się od okna	380
Twój pierwszy interfejs graficzny — przycisk w ramce	381
Przechwytywanie zdarzeń generowanych przez działania użytkownika	383
Odbiorcy, źródła i zdarzenia	387
Stwórz własny komponent umożliwiający rysowanie	390
Odwwołanie do porządnej klasy Graphics ukrywa obiekt Graphics2D	392
Układy GUI — wyświetlanie w ramce więcej niż jednego komponentu	396
Jak stworzyć obiekt klasy wewnętrznej?	404
Wykorzystanie klas wewnętrznych do tworzenia animacji	408
Odbieranie zdarzeń niezwiązanych z interfejsem użytkownika	413

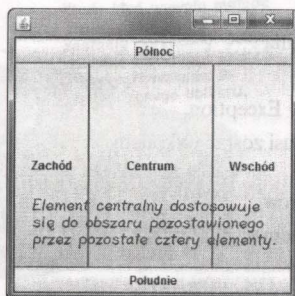
13

Popracuj nad Swingiem

Swing jest łatwy. Chyba że naprawdę zwracasz uwagę na to, co jest gdzie wyświetlane. Kod wykorzystujący Swing *wydać się* prosty, ale kiedy go skompilujesz; uruchomisz i popatrzyś na wyniki, to często będziesz mógł sobie pomyśleć: „Hej, przecież to nie miało być wyświetlone w tym miejscu”. To, co zapewnia prostotę kodu, sprawia jednocześnie, że trudne jest kontrolowanie położenia elementów — tym „czymś” jest **menedżer układu**. Jednak przy odrobinie pracy można nagiąć menedżer układu do naszej woli. W tym rozdziale popracujemy nad naszym Swingiem i dowiemy się czegoś więcej o różnych elementach graficznych.

Komponenty na wschodzie i zachodzie mają preferowaną szerokość.

Także komponenty na północy i południu mają preferowaną wysokość.



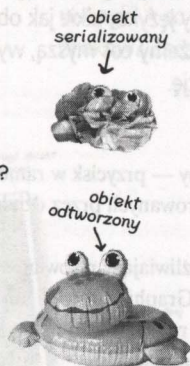
Komponenty biblioteki Swing	426
Komponenty można zagnieżdżać	426
Menedżery układu	427
W jaki sposób menedżery układu podejmują decyzje?	428
Różne menedżery układu mają różne zasady rozmieszczania	428
Wielka trójka menedżerów układu: BorderLayout, FlowLayout oraz BoxLayout	429
Zabawy z komponentami biblioteki Swing	439
Tworzenie aplikacji MuzMachina	445

14

Zapisywanie obiektów

Obiekty można pakować i odtwarzać. Obiekty mają swój stan i działanie. Działanie obiektu określa jego klasa, natomiast *stan* zależy od konkretnego *obektu*. Jeśli program musi zapisać stan obiektu, *możesz to zrobić w sposób trudny* — analizując każdy obiekt i pracowicie zapisując wartości wszystkich właściwości. **Możesz także zapisać obiekt w sposób łatwy i obiektowy** — „zamrażając” obiekt (przeprowadzając jego serializację), a następnie odtwarzając (poprzez jego deserializację).

jakiś pytania?

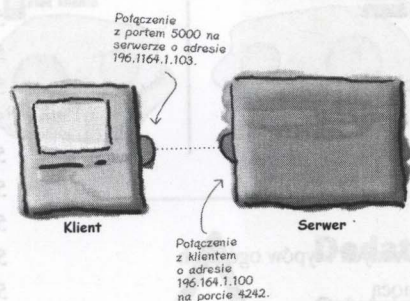


Odczytywanie taktów	454
Zapisywanie stanu	455
Zapisywanie serializowanego obiektu do pliku	456
Deserializacja — odtwarzanie obiektów	465
Zapisywanie łańcucha znaków w pliku tekstowym	471
Klasa java.io.File	476
Odczyt zawartości pliku tekstowego	478
Przetwarzanie łańcuchów znaków przy użyciu metody split()	482
Identyfikator wersji — wielki problem serializacji	484
Stosowanie serialVersionUID	485
Zapisywanie kompozycji	487

15

Nawiąż połączenie

Nawiąż połączenie ze światem zewnętrznym. To takie łatwe. Wszelkie szczegóły związane z komunikacją sieciową na niskim poziomie są obsługiwane przez klasy należące do biblioteki java.net. Jedną z najlepszych cech Javy jest to, iż wysyłanie i odbieranie danych przez sieć to w zasadzie normalne operacje wejścia-wyjścia, z tą różnicą, że są w nich wykorzystywane nieco inne strumienie. W tym rozdziale stworzymy gniazda używane przez programy klienckie. Stworzymy także gniazda używane przez programy pełniące funkcje serwerów. Napiszemy zarówno program klienta, jak i serwera. Zanim skończysz czytać ten rozdział, będziesz już dysponować w pełni funkcjonalnym, wielowątkowym programem do prowadzenia internetowych pogawędek. Zaraz... czy właśnie padło słowo *wielowątkowy*?

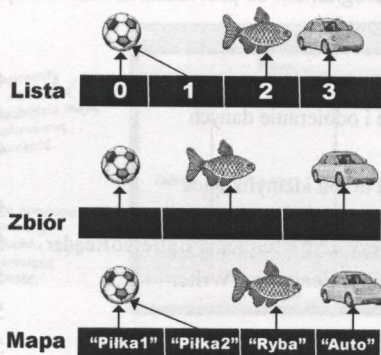


Muzyczne pogawędki w czasie rzeczywistym	496
Nawiązywanie połączenia, wysyłanie i odbieranie danych	498
Nawiązanie połączenia sieciowego	499
Port TCP to tylko numer. 16-bitowa liczba identyfikująca konkretny program na serwerze	500
Aby odczytywać dane z gniazda, należy użyć strumienia BufferedReader	502
Aby zapisać dane w gnieździe, użyj strumienia PrintWriter	503
Program CodziennePoradyKlient	504
Kod programu CodziennePoradyKlient	505
Tworzenie prostego serwera	507
Kod aplikacji CodziennePoradySerwer	508
Tworzenie klienta pogawędek	510
Możliwość stosowania wielu wątków w Javie	514
zapewnia jedna klasa — Thread	514
Jakie są konsekwencje posiadania więcej niż jednego stosu wywołań?	515
Aby stworzyć zadanie dla wątku, zaimplementuj interfejs Runnable	518
Mechanizm zarządzający wątkami	521
Uspianie wątku	525
Uspianie wątków w celu zapewnienia bardziej przewidywalnego działania programu	526
Tworzenie i uruchamianie dwóch wątków	527
Cóż... Tak, możemy. JEST pewien problem związany ze stosowaniem wątków	528
Problem Moniki i Roberta w formie kodu	530
Przykład Moniki i Roberta	531
Musimy wykonać metodę pobierzGotowke() jako operację atomową	534
Stosowanie blokady obiektu	535
Przerażający problem „utraconej modyfikacji”	536
Wykonajmy ten przykładowy kod...	537
Zadeklaruj metodę inkrementuj() jako metodę atomową. Synchronizuj ją!	538
Mroczna strona synchronizacji	540
Nowa i poprawiona wersja programu ProstyKlientPogawedek	542
Naprawdę prosty serwer pogawędek	544

16

Struktury danych

Sortowanie w Javie to pestka. Dysponujesz wszystkimi narzędziami do gromadzenia danych i manipulowania nimi, i to bez konieczności pisania własnych algorytmów. Biblioteka kolekcji Javy (*Java Collections Framework*) posiada struktury danych, które powinny sprostać praktycznie wszystkiemu, co będziesz chciał zrobić. Potrzebujesz listy, do której będziesz mógł łatwo dodawać elementy? Chcesz znaleźć coś na podstawie nazwy? A może chcesz stworzyć listę, która automatycznie będzie usuwać powtarzające się elementy? Albo posortować listę pracowników na podstawie liczby „przyjacielskich klepięć w plecy”?

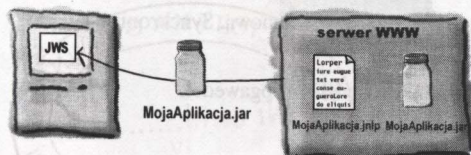
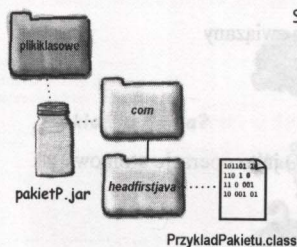


ArrayList nie jest jedyną dostępną kolekcją	557
Deklaracja metody sort()	563
Poznajemy typy ogólne	565
Używanie KLAS uogólnionych	566
Stosowanie METOD uogólnionych	568
Ponownie odwiedzimy metodę sort()	571
Stosowanie własnych komparatorów	576
Potrzebujemy zbioru, a nie listy	581
Biblioteka kolekcji (fragment)	582
Stosowanie argumentów polimorficznych i typów ogólnych	593
Znaki wieloznaczne śpieszą z pomocą	598

17

Rozpowszechnij swój kod

Już czas wypłynąć na „szerokie wody”. Napisałeś kod swojej aplikacji. Przetestowałeś go. Udoskonaliłeś. Powiedziałeś wszystkim znajomym, że świetnie by było, gdybyś już w życiu nie musiał napisać choćby jednej linijki kodu. Ale w końcu stworzyłeś dzieło sztuki. Przecież Twoja aplikacja działa! W ostatnich dwóch rozdziałach książki zajmiemy się zagadnieniami związanymi z organizowaniem, pakowaniem i wdrażaniem kodu. Przyjrzymy się możliwościom wdrażania lokalnego, mieszanego i zdalnego, w tym wykonywalnym plikom JAR, technologii Java Web Start, RMI oraz serwetom. Nie stresuj się! Niektóre z najbardziej niesamowitych rozwiązań, jakimi może się poszczycić Java, są prostsze, niż można by przypuszczać.

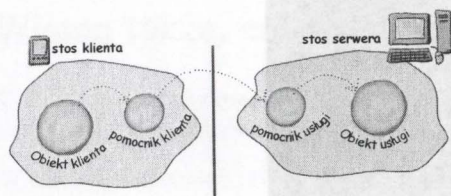


Wdrażanie aplikacji	606
Oddzielanie kodu źródłowego od plików klasowych	608
Umieszczanie programów w archiwach JAR	609
Uruchamianie (wykonywanie) archiwum JAR	610
Umieść klasy w pakietach	611
Zapobieganie konfliktom nazw pakietów	612
Kompilacja i uruchamianie programu, w którym wykorzystywane są pakiety	614
Flaga -d jest nawet lepsza, niż twierdziliśmy	615
Tworzenie wykonywalnego archiwum JAR zawierającego pakiety	616
Java Web Start	621
Plik .jnlp	623

18

Przetwarzanie rozproszone

Zdalne wykonywanie aplikacji nie zawsze jest złe. Oczywiście to prawda, że życie jest łatwiejsze, gdy wszystkie elementy aplikacji znajdują się w jednym miejscu i są zarządzane przez jedną wirtualną maszynę Javy. Jednak takie rozwiązanie nie zawsze jest możliwe. Co w sytuacji, gdy aplikacja realizuje bardzo złożone obliczenia? Co, jeśli potrzebuje informacji pochodzących z zabezpieczonej bazy danych? W tym rozdziale przedstawiona zostanie zadziwiająco prosta technologia wywoływania zdalnych metod — *RMI* (ang. *Remote Method Invocation*). Pobieźnie przyjrzymy się także innym rozwiązaniom — serwetom, komponentom *EJB* (ang. *Enterprise Java Bean*) oraz technologii Jini.

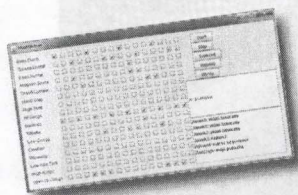


RMI udostępnia obiekty pomocnicze klienta i serwera	636
Bardzo prosty serwet	649
Tak dla zabawy przeróbmy nasz program krasomówcy na serwet	651
Enterprise JavaBeans — RMI na środkach dopingujących	653
I na samym końcu przedstawiamy... małego dzina Jini	654
Odkrywanie adaptacyjne w akcji	655

A

Dodatek A

Ostatnie doprowadzanie kodu. Kompletny kod aplikacji MuzMach i na w ostatecznej wersji klient-serwer z możliwością prowadzenia muzycznych pogawędek. Twoja szansa, by zostać gwiazdą rocka.



Ostateczna wersja programu MuzMachina	672
Ostateczna wersja serwera aplikacji MuzMachina	679

B

Dodatek B

Dziesięć najważniejszych zagadnień, które niema znalazły się w tej książce... Jeszcze nie możemy Cię zostawić i wypuścić w świat. Mamy da Ciebie kilka dodatkowych informacji, jednak to już jest koniec tej książki. I tym razem mówimy to zupełnie serio.

Lista dziesięciu zagadnień	682
----------------------------	-----

S

Skorowidz

	699
--	-----