

Przewodnik dla poszukujących pracy!



WYDANIE III



Programista szuka pracy

Kulisy rekrutacji w branży IT

John Mongan, Noah Kindler, Eric Giguere

Helion

Tytuł oryginału: Programming Interviews Exposed: Secrets to Landing Your Next Job, Third Edition

Tłumaczenie: Łukasz Piwko

ISBN: 978-83-246-9861-5

Copyright © 2013 by John Wiley & Sons, Inc., Indianapolis, Indiana. All Rights Reserved.

This translation published under license with the original publisher John Wiley & Sons, Inc. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise without either the prior written permission of the Publisher.

The Wrox Brand trade dress is a trademark of John Wiley & Sons, Inc. in the United States and/or other countries. Used by permission.

Wiley, the Wiley logo, Wrox, the Wrox logo, Wrox Programmer to Programmer, and related trade dress are trademarks or registered trademarks of John Wiley & Sons, Inc. and/or its affiliates, in the United States and other countries, and may not be used without written permission. All other trademarks are the property of their respective owners. John Wiley & Sons, Inc., is not associated with any product or vendor mentioned in this book.

Translation copyright © 2015 by Helion S.A.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz Wydawnictwo HELION dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Wydawnictwo HELION nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Wydawnictwo HELION

ul. Kościuszki 1c, 44-100 GLIWICE

tel. 32 231 22 19, 32 230 98 63

e-mail: helion@helion.pl

WWW: <http://helion.pl> (księgarnia internetowa, katalog książek)

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

<http://helion.pl/user/opinie/rokwpr>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

Spis treści

O AUTORACH	13
O KOREKTORACH MERYTORYCZNYCH	15
PODZIĘKOWANIA	17
PRZEDMOWA	19
WPROWADZENIE	23
ROZDZIAŁ 1. ZANIM ZACZNIESZ SZUKAĆ	27
Poznaj samego siebie	27
Poznaj rynek	29
Podstawowe informacje o rynku	29
Outsourcing	30
Rozwijanie atrakcyjnych umiejętności	31
Doprowadzaj projekty do końca	32
Zadbaj o swój profil internetowy	32
Podsumowanie	34
ROZDZIAŁ 2. PROCES UBIEGANIA SIĘ O PRACĘ	35
Znajdowanie firm i kontaktowanie się z nimi	35
Szukanie firm	35
Polecanie	36
Współpraca z łowcami głów	36
Bezpośredni kontakt z firmą	37
Targi pracy	38

Przebieg kwalifikacji	38
Rozmowy przesiewowe	38
Rozmowy kwalifikacyjne w siedzibie firmy	39
Strój	39
Rola rekrutera	40
Oferty i negocjowanie	41
Jak sobie radzić z presją wywieraną przez rekrutera?	41
Negocjowanie wynagrodzenia	41
Przyjmowanie i odrzucanie oferty	42
Podsumowanie	43
ROZDZIAŁ 3. RÓŻNE PODEJŚCIA DO PROBLEMÓW PROGRAMISTYCZNYCH	45
Proces	45
Scenariusz	46
Problemy	46
Jakich języków programowania używać?	47
Kluczem jest interaktywność	47
Rozwiązywanie problemów	48
Podstawowe kroki	48
Co robić, gdy się utknie?	50
Analizowanie rozwiązania	50
Zastosowanie notacji dużego O	51
Działanie notacji dużego O	52
Najlepszy, średni i najgorszy przypadek	53
Optymalizacja a notacja dużego O	53
Jak analizować algorytmy przy użyciu notacji dużego O?	54
Który algorytm jest lepszy?	54
Analizowanie zużycia pamięci	55
Podsumowanie	55
ROZDZIAŁ 4. LISTY POWIĄZANE	57
Dlaczego listy powiązane?	57
Rodzaje list powiązanych	58
Listy powiązane jednostronnie	58
Listy powiązane dwustronnie	59
Listy cykliczne	60
Podstawowe operacje na listach powiązanych	60
Zapamiętywanie elementu początkowego	60
Przeglądanie listy	62
Wstawianie i usuwanie elementów	62
Zadania związane z listami powiązanymi	64
Implementacja stosu	64
Obsługiwanie wskaźnika do ogona listy powiązanej	69
Błędy w funkcji removeHead	73
Zwracanie elementu listy o określonym numerze od końca	75
Spłaszczanie listy	77
Przywracanie spłaszczonej listy do pierwotnego stanu	80
NULL lub cykl	82
Podsumowanie	84

ROZDZIAŁ 5. DRZEWA I GRAFY **87**

Drzewa	87
Drzewa binarne	89
Binarne drzewa poszukiwań	90
Stery	92
Typowe sposoby przeszukiwania	92
Metody przeglądania drzew	93
Grafy	94
Zadania dotyczące drzew i grafów	95
Wysokość drzewa	95
Przeглядanie wzdłużne	96
Przeглядanie wzdłużne bez rekurencji	97
Najniższy wspólny przodek	99
Przekształcanie drzewa binarnego w stertę	100
Niezrównoważone binarne drzewo poszukiwań	103
Sześć stopni od Kevina Bacona	105
Podsumowanie	109

ROZDZIAŁ 6. TABLICE I ŁAŃCUCHY **111**

Tablice	111
Języki C i C++	112
Java	113
C#	113
JavaScript	114
łańcuchy	114
C	115
C++	115
Java	116
C#	117
JavaScript	117
Problemy dotyczące tablic i łańcuchów	117
Znajdowanie pierwszego niepowtarzającego się znaku	117
Usuwanie określonych znaków	120
Odwracanie kolejności słów w łańcuchu	123
Konwersje między liczbami całkowitymi i łańcuchami	127
Podsumowanie	131

ROZDZIAŁ 7. REKURENCJA **133**

Istota rekurencji	133
Problemy rekurencyjne	136
Wyszukiwanie binarne	137
Permutacje łańcucha	139
Kombinacje łańcucha	141
Słowa telefoniczne	144
Podsumowanie	148

ROZDZIAŁ 8. SORTOWANIE	149
Algorytmy sortujące	149
Sortowanie przez wybieranie	150
Sortowanie przez wstawianie	151
Szybkie sortowanie	152
Sortowanie przez scalanie	154
Problemy dotyczące sortowania	155
Najlepszy algorytm sortowania	155
Stabilne sortowanie przez wybieranie	158
Sortowanie przy użyciu wielu kluczy	160
Spraw, by sortowanie było stabilne	161
Optymalizacja szybkiego wyszukiwania	163
Sortowanie naleśników	166
Podsumowanie	168
ROZDZIAŁ 9. WSPÓŁBIEŻNOŚĆ	169
Podstawowe pojęcia wielowątkowości	169
Wątki	169
Wątki systemowe i wątki użytkownika	170
Monitory i semafony	170
Zakleszczenia	171
Przykład użycia wątków	172
Problemy dotyczące współbieżności	174
Aktywne oczekiwanie	174
Producent i konsument	176
Uczujący filozofowie	178
Podsumowanie	182
ROZDZIAŁ 10. PROGRAMOWANIE OBIEKTOWE	183
Podstawy	183
Klasy i obiekty	183
Dziedziczenie i polimorfizm	184
Konstrukcja i destrukcja obiektów	185
Problemy dotyczące programowania obiektowego	186
Interfejsy i klasy abstrakcyjne	186
Metody wirtualne	188
Wielodziedziczenie	189
Podsumowanie	190
ROZDZIAŁ 11. WZORCE PROJEKTOWE	191
Czym są wzorce projektowe?	191
Po co są wzorce projektowe?	191
Wzorce projektowe na rozmowach kwalifikacyjnych	192
Najczęściej używane wzorce projektowe	192
Wzorce kreatywne	193
Wzorce czynnościowe	195
Wzorce strukturalne	196

Problemy związane z wzorcami projektowymi	196
Implementacja Singletonu	196
Dekorator kontra dziedziczenie	199
Wydajne aktualizacje Obserwatora	200
Podsumowanie	200
ROZDZIAŁ 12. BAZY DANYCH	201
Podstawowe wiadomości o bazach danych	201
Relacyjne bazy danych	201
SQL	202
Transakcje w bazach danych	206
Problemy dotyczące baz danych	207
Proste zadanie z użyciem języka SQL	207
Baza danych firmowych i pracowniczych	207
Znajdowanie największej wartości bez użycia funkcji agregacyjnych	209
Logika trójwartościowa	211
Podsumowanie	212
ROZDZIAŁ 13. OPERACJE GRAFICZNE I NA BITACH	213
Grafika	213
Szperanie przy bitach	214
Binarne uzupełnienie dwójkowe	214
Operatory bitowe	215
Optymalizacja wydajności przy użyciu operatorów przesunięcia	216
Problemy graficzne	216
Jedna ósma okręgu	217
Nakładanie się prostokątów	219
Problemy dotyczące szperania przy bitach	222
Big-endian czy little-endian	222
Liczba jedynek	224
Podsumowanie	227
ROZDZIAŁ 14. ŁAMIGŁÓWKI DOTYCZĄCE MIERZENIA, LICZENIA I PORZĄDKOWANIA	229
Rozwiązywanie zagadek	229
Docieraj do sedna problemu	230
Nie daj się przytłoczyć	231
Strzeż się prostych problemów	232
Problemy szacunkowe	232
Łamigłówki	233
Liczenie otwartych schowków	233
Trzy przełączniki	235
Przechodzenie przez most	236
Ciężka kulka	238
Liczba stacji benzynowych w USA	243
Podsumowanie	244

ROZDZIAŁ 15. ŁAMIGŁÓWKI GRAFICZNE I PRZESTRZENNE 245

Najpierw to narysuj	245
Zadania graficzne i przestrzenne	246
Łódź i dok	246
Liczenie kostek	248
Lis i kaczką	251
Pałace się lonty	252
Uciekanie przed pociągiem	254
Podsumowanie	256

ROZDZIAŁ 16. PYTANIA MERYTORYCZNE 257

Przygotowanie	257
Problemy	258
C++ a Java	259
Klasy zaprzyjaźnione	259
Przekazywanie argumentów	260
Makra i funkcje inline	261
Dziedziczenie	263
Automatyczne usuwanie nieużywanych obiektów	263
Aplikacje 32-bitowe a aplikacje 64-bitowe	265
Wydajność sieci	265
Bezpieczeństwo aplikacji sieciowych	266
Kryptografia	268
Tablice mieszające a binarne drzewa poszukiwań	269
Podsumowanie	269

ROZDZIAŁ 17. PYTANIA NA TEMATY NIETECHNICZNE 271

Po co zadawane są nietechniczne pytania?	271
Pytania	272
Co chciałby Pan robić?	272
Jaki jest Pana ulubiony język programowania?	273
Jaki styl pracy Pan preferuje?	274
Co może Pan powiedzieć o swoim doświadczeniu zawodowym?	274
Jakie są Pana cele zawodowe?	274
Czemu chce Pan zmienić pracę?	274
Jakich zarobków Pan oczekuje?	275
Ile wcześniej Pan zarabiał?	277
Dlaczego uważa Pan, że powinniśmy Pana zatrudnić?	277
Dlaczego chce Pan pracować w tej firmie?	278
Czy ma Pan jakieś pytania?	278
Podsumowanie	278

DODATEK A. CV 279

CV inżyniera	279
Przykład słabego CV	279
Sprzedaj siebie	283
Pisz zwięźle	283
Zamieść tylko ważne informacje	284
Pisz przejrzysto i bądź konkretny	285

Uwzględnij tylko najważniejsze informacje	286
Zastosuj odwrotną chronologię	286
Wykonaj korektę	287
Poprawiony przykład	287
Menedżery i starsi programiści	289
Dostosuj CV do stanowiska, o które się ubiegasz	295
Przykładowe CV	295
PODSUMOWANIE	299
SKOROWIDZ	301

10

Programowanie obiektowe

Większość profesjonalnych programistów używa obiektowych języków programowania, takich jak Java, C# czy C++. Nawet język JavaScript, mimo że nie jest obiektowy, ma pewne cechy obiektowych języków programowania pod postacią obiektów prototypowych i sprytnego wykorzystania definicji funkcji. Dlatego każdy kandydat na rozmowie kwalifikacyjnej powinien dobrze rozumieć zasady programowania obiektowego.

PODSTAWY

Korzenie obiektowych technik programowania sięgają kilkadziesiąt lat wstecz, do takich języków jak Simula i Smalltalk. Programowanie obiektowe stało się tematem wielu opracowań naukowych od czasu, gdy aktywni programiści zaczęli powszechnie z niego korzystać.

Klasy i obiekty

Istnieje wiele sposobów i nie ma pełnej zgody co do tego, jak opisać i zdefiniować obiektowość jako technikę programowania, ale wszyscy zgodnie mówią o klasach i obiektach. **Klasa** to abstrakcyjna definicja czegoś, co ma **atrybuty** (czasami nazywane **własnościami** lub **stanami**) i wzorce zachowań (**możliwości** lub **metody**). **Obiekt** to konkretny egzemplarz klasy, który ma własny stan niepowiązany ze stanami innych egzemplarzy tej klasy. Poniżej znajduje się definicja klasy punktu będącego parą liczb całkowitych reprezentujących wartości x i y w kartezjańskim układzie współrzędnych.

```
public class Point {
    private int x;
    private int y;
    public Point( int x, int y ){
        this.x = x;
        this.y = y;
    }
    public Point( Point other ){
        x = other.getX();
        y = other.getY();
    }
}
```

```

public int getX(){ return x; }
public int getY(){ return y; }
public Point relativeTo( int dx, int dy ){
    return new Point( x + dx, y + dy );
}
public String toString(){
    StringBuffer b = new StringBuffer();
    b.append( '(' );
    b.append( x );
    b.append( ',' );
    b.append( y );
    b.append( ')' );
    return b.toString();
}
}

```

Aby reprezentować dowolny punkt, należy utworzyć egzemplarz klasy `Point` przy użyciu odpowiednich wartości.

```

Point p1 = new Point( 5, 10 );
Point p2 = p1.relativeTo( -5, 5 );
System.out.println( p2.toString() ); // Drukuje (0,15).

```

Ten prosty przykład jest ilustracją jednej z podstawowych zasad programowania obiektowego — **hermetyzacji**, czyli ukrywania szczegółów implementacji.

Dziedziczenie i polimorfizm

Dwa inne, bardzo ważne pojęcia obiektowości to dziedziczenie i polimorfizm, które są ściśle spokrewnione. **Dziedziczenie** polega na tym, że można zdefiniować klasę jako zmodyfikowaną lub bardziej wyspecjalizowaną wersję innej klasy. Gdy klasa B dziedziczy po klasie A (w niektórych językach programowania, np. Javie, oznacza się to słowem kluczowym `extends`), klasa A jest **rodzicem** klasy B lub jej **klasą bazową** albo **nadklasą**, natomiast klasa B jest **podklasą** lub **klasą podrzędną** klasy A. Wszystkie zachowania zdefiniowane w klasie A znajdują się też w klasie B, chociaż mogą być nieco zmienione. Egzemplarza klasy B można nawet użyć wszędzie tam, gdzie można zastosować egzemplarz klasy A.

Polimorfizm to możliwość dostarczania wielu implementacji zachowania i wybierania jednej z nich na podstawie kontekstu. Przykładowo klasa może zawierać definicje dwóch wersji jednej metody, z których każda przyjmuje inne parametry wywołania. Ta sama metoda może też być zdefiniowana zarówno w nadklasie, jak i podklasie, przy czym ta druga *przesłania* pierwszą w przypadku wywołań na obiektach podklasy. Wybór metody do wywołania może odbywać się podczas kompilacji kodu lub działania programu.

Klasyfikacyjnym przykładem przywoływanym przy objaśnianiu dziedziczenia i polimorfizmu jest implementacja biblioteki kształtów geometrycznych używanej w aplikacji do rysowania wektorowego. Na szczycie hierarchii znajduje się klasa o nazwie `Shape` zawierająca definicje cech wspólnych wszystkich kształtów.

```

public abstract class Shape {
    protected Point center;
    protected Shape( Point center ){
        this.center = center;
    }
    public Point getCenter(){
        return center; // Ponieważ punkt jest niezmienny.
    }
}

```

```

    public abstract Rectangle getBounds();
    public abstract void draw( Graphics g );
}

```

Następnie można zdefiniować specjalne podklasy `Rectangle` (prostokąt) i `Ellipse` (elipsa).

```

public class Rectangle extends Shape {
    private int h;
    private int w;

    public Rectangle( Point center, int w, int h ){
        super( center );
        this.w = w;
        this.h = h;
    }
    public Rectangle getBounds(){
        return this;
    }
    public int getHeight(){ return h; }
    public int getWidth(){ return w; }
    public void draw( Graphics g ){
        .... // Kod rysujący prostokąt.
    }
}

public class Ellipse extends Shape {
    private int a;
    private int b;
    public Ellipse( Point center, int a, int b ){
        super( center );
        this.a = a;
        this.b = b;
    }
    public Rectangle getBounds(){
        return new Rectangle( center, a * 2, b * 2 );
    }
    public int getSemiMajorAxis(){ return a; }
    public int getSemiMinorAxis(){ return b; }
    public void draw( Graphics g ){
        .... // Kod rysujący elipsę.
    }
}

```

Klasy `Rectangle` i `Ellipse` można jeszcze dalej wyspecjalizować przez utworzenie ich podklas `Square` (kwadrat) i `Circle` (koło).

Mimo że w bibliotece może być zdefiniowanych wiele kształtów, część aplikacji rysująca te figury nie jest skomplikowana.

```

void paintShapes( Graphics g, List<Shape> shapes ){
    for( Shape s : shapes ){
        s.draw( g );
    }
}

```

Aby dodać nowy kształt do biblioteki, wystarczy utworzyć podklasę jednej z istniejących klas i zaimplementować w niej różnice.

KONSTRUKCJA I DESTRUKCJA OBIEKTÓW

Obiekty są egzemplarzami klas. Tworzenie obiektu nazywa się **konstrukcją obiektu**. W procesie tym wywołuje się metodę klasy zwaną **konstruktorem**. Konstruktor inicjuje stan obiektu,

co zazwyczaj wymaga wywołania (jawnie lub niejawnie) konstruktorów klas nadrzędnych, aby te zainicjowały swoją część stanu obiektu.

Niszczenie obiektów jest trochę bardziej skomplikowane od ich tworzenia. W języku C++ wywołuje się w tym celu funkcję zwaną **destruktoem**, której zadaniem jest skasowanie stanu obiektu. Destruktory są wywoływane automatycznie, gdy obiekt wychodzi poza zakres dostępności lub w wyniku użycia operatora `delete` służącego do usuwania dynamicznie utworzonych obiektów — ważne jest zapamiętywanie obiektów, aby uniknąć wycieków pamięci. Natomiast w językach, takich jak C# i Java, nieużywane obiekty znajduje i usuwa mechanizm zwany **śmieciarką** (ang. *garbage collector*). W takim przypadku czas i miejsce (często dzieje się to w osobnym wątku zdefiniowanym przez system) destrukcji są poza kontrolą aplikacji. Opcjonalnie system może wywołać metodę *finalizującą* przed destrukcją obiektu, aby dać mu szansę na samooczyszczenie, zanim go ostatecznie zniszczy. (W językach C# i Java obiekt może się „wskrzesić” w finalizatorze).

PROBLEMY DOTYCZĄCE PROGRAMOWANIA OBIEKTOWEGO

Zadania związane z programowaniem obiektowym najczęściej skupiają się na samych pojęciach obiektowości, zwłaszcza na kwestiach dotyczących języków programowania używanych w firmie.

Interfejsy i klasy abstrakcyjne

PROBLEM Czym różni się interfejs od klasy abstrakcyjnej w programowaniu obiektowym?

Odpowiedź na to pytanie zależy od języka programowania, ale można przedstawić ogólne definicje.

- **Interfejs** zawiera deklaracje zestawu powiązanych metod, poza jakąkolwiek klasą.
- **Klasa abstrakcyjna** to niekompletna definicja klasy, zawierająca deklaracje wszystkich metod, ale niezawierająca wszystkich definicji.

Zatem interfejs definiuje **interfejs programistyczny** (ang. *application programming interface* — API) niezależny od jakiegokolwiek hierarchii klas. Interfejsy są najważniejsze w językach programowania obsługujących tylko pojedyncze dziedziczenie, czyli takich, w których klasa może bezpośrednio dziedziczyć tylko po jednej innej klasie. Mówi się, że klasa zawierająca wszystkie metody opisane w interfejsie *implementuje* ten interfejs.

Klasa abstrakcyjna, w odróżnieniu od interfejsu, jest właściwą klasą, tzn. może zawierać dane składowe i definicje metod oraz może być podklasą. Jednak, w odróżnieniu od klasy konkretnej (nieabstrakcyjnej), klasa abstrakcyjna nie zawiera definicji niektórych metod, które są pozostawione do zaimplementowania w jej podklasach. W związku z tym nie można utworzyć egzemplarza klasy abstrakcyjnej — można tylko budować egzemplarze jej konkretnych podklas.

Interfejs jest równoznaczny z klasą abstrakcyjną bez danych składowych i definicji metod. W języku C++ tak właśnie definiuje się interfejsy: deklaruje się klasę bez danych składowych i zawierającą tylko funkcje czysto wirtualne. Oto przykład.

```
class StatusCallback {
    public:
        virtual void updateStatus( int oState, int nState ) = 0;
}
```

Klasa implementuje taki interfejs przez dziedziczenie po nim i dostarczenie definicji zadeklarowanej w nim metody.

```
class MyClass : SomeOtherClass, StatusCallback {
    public:
        void updateStatus( int oState, int nState ){
            if( nState > oState ){
                .... // Robi coś.
            }
        }
        .... // Reszta klasy.
}
```

W Javie interfejsy definiuje się przy użyciu słowa kluczowego `interface`.

```
public interface StatusCallback {
    void updateStatus( int oState, int nState );
}
```

Taki interfejs można następnie zaimplementować w klasie.

```
public class MyClass implements StatusCallback {
    public void updateStatus( int oState, int nState ){
        if( nState > oState ){
            .... // Robi coś.
        }
    }
    .... // Reszta klasy.
}
```

W językach programowania obsługujących zarówno interfejsy, jak i klasy abstrakcyjne często tworzy się *domyślne implementacje* interfejsów w klasach abstrakcyjnych. Przykładowo poniższy interfejs:

```
public interface XMLReader {
    public XMLObject fromString( String str );
    public XMLObject fromReader( Reader in );
}
```

może mieć następującą domyślną implementację:

```
public abstract class XMLReaderImpl implements XMLReader {
    public XMLObject fromString( String str ){
        return fromReader( new StringReader( str ) );
    }
    public abstract XMLObject fromReader( Reader in );
}
```

Programista, który chce zaimplementować interfejs `XMLReader`, mógłby utworzyć podklasę klasy `XMLReaderImpl` (prawdopodobnie jako klasę zagnieżdżoną) i zaimplementować tylko jedną metodę zamiast dwóch.

Ogólnie rzecz biorąc, klasy abstrakcyjne są przydatne, gdy wyprowadzane z nich podklasy są bardziej wyspecjalizowanymi typami klasy bazowej (pozostają w relacji *jest*), zwłaszcza gdy w klasie abstrakcyjnej istnieje jakaś wspólna funkcjonalność (np. dane składowe albo definicje metod), z której mogą korzystać klasy pochodne. Interfejsy są przydatne wtedy, kiedy trzeba zapewnić, aby niepowiązane klasy umożliwiały wywoływanie koncepcyjnie powiązanych funkcji w taki sam sposób, ale implementacja tej funkcjonalności może znacznie się różnić między tymi klasami.

Metody wirtualne

PROBLEM Czym są metody wirtualne? Do czego służą?

W programowaniu obiektowym klasy potomne mogą przesłaniać (przedefiniowywać) metody zdefiniowane w klasach nadrzędnych. Jeśli metoda jest wirtualna, definicja metody do wywołania jest określana w czasie działania programu na podstawie typu (klasy) obiektu, na którym została wywołana. W Javie wirtualne są wszystkie metody niestaticzne, ale w językach C# i C++ deklaruje się je przy użyciu słowa kluczowego `virtual` — domyślnie tworzone są metody niewirtualne. Jeżeli metoda nie jest wirtualna, definicja do wywołania jest wybierana w czasie kompilacji na podstawie typu referencji (albo wskaźnika).

Łatwiej to zrozumieć na konkretnych przykładach. Spójrz na poniższe trzy klasy w języku C++.

```
class A {
public:
    void print() { cout << "A"; }
}
class B : A {
public:
    void print() { cout << "B"; }
}
class C : B {
public:
    void print() { cout << "C"; }
}
```

Ponieważ funkcja `print` nie jest wirtualna, to, która wersja zostanie wywołana, zależy od typu użytego w *czasie kompilacji*.

```
A *a = new A();
B *b = new B();
C *c = new C();
a->print(); // "A"
b->print(); // "B"
c->print(); // "C"
((B *)c)->print(); // "B"
((A *)c)->print(); // "A"
((A *)b)->print(); // "A"
```

Gdyby funkcja `print` była wirtualna:

```
class A {
public:
    virtual void print() { cout << "A"; }
}
class B : A {
public:
    virtual void print() { cout << "B"; }
}
class C : B {
public:
    virtual void print() { cout << "C"; }
}
```

to wybór definicji do wywołania zostałby podjęty na podstawie *typu użytego w czasie wykonywania*:

```
A *a = new A();
B *b = new B();
C *c = new C();
```



```

a->print(); // "A"
b->print(); // "B"
c->print(); // "C"
((B *)c)->print(); // "C"
((A *)c)->print(); // "C"
((A *)b)->print(); // "B"

```

Metody wirtualne są mechanizmem wykorzystywanym w polimorfizmie: umożliwiają w jednym wywołaniu metody wywoływanie różnych definicji metod w zależności od tego, do jakiej klasy należy obiekt. W wersji w języku C++ klasy Shape przedstawionej na początku tego rozdziału metoda `draw` musiałaby być zdefiniowana jako wirtualna, aby metoda `paintShapes` — która uzyskuje dostęp do obiektów jako referencji typu `Shape` — mogła działać. Specjalnym rodzajem metody wirtualnej w języku C++ jest tzw. funkcja czysto wirtualna, czyli funkcja zadeklarowana, ale celowo niezdefiniowana. (Wprawdzie w języku C++ istnieje możliwość zadeklarowania w klasie funkcji czysto wirtualnej i jednocześnie jej zdefiniowania, ale definicję tę można wywołać tylko w podklasie. Jeśli potrzebujesz czegoś skomplikowanego, język C++ jest w sam raz dla Ciebie). Każda klasa zawierająca funkcję czysto wirtualną lub dziedzicząca taką funkcję i jej niedefiniującą jest klasą abstrakcyjną. (W Javie i C# odpowiednikami funkcji czysto wirtualnych są metody abstrakcyjne).

Funkcje wirtualne nie są darmowe. Wywołanie takiej funkcji prawie zawsze trwa dłużej niż normalnej, ponieważ trzeba znaleźć adres odpowiedniej funkcji w tablicy. Ponadto tablica ta również zajmuje niewielką ilość pamięci. Mimo to, w większości aplikacji narzut funkcji wirtualnych jest tak mały, że nieistotny.

Wielodziedziczenie

PROBLEM Dlaczego w językach C# i Java zabronione jest bezpośrednie dziedziczenie po wielu klasach?

W języku C++ klasa może dziedziczyć (pośrednio i bezpośrednio) po więcej niż jednej klasie i nazywa się to **wielodziedziczeniem** (ang. *multiple inheritance*). Natomiast w językach C# i Java klasa może bezpośrednio dziedziczyć tylko po jednej klasie i nazywa się to **dziedziczeniem pojedynczym** (ang. *single inheritance*).

Wielodziedziczenie to technika umożliwiająca tworzenie klas zawierających cechy dwóch różnych hierarchii klas, co często jest przydatne, gdy używa się różnych systemów klas w jednej aplikacji. Jeśli w dwóch takich systemach jest np. zdefiniowana inna klasa bazowa dla wyjątków, to za pomocą wielodziedziczenia można utworzyć klasy wyjątków nadające się do użytku w obu tych systemach.

Problem z wielodziedziczeniem polega na tym, że może prowadzić do niejednoznaczności. Klasycznym przykładem jest sytuacja, w której klasa dziedziczy po dwóch innych klasach, z których każda dziedziczy po tej samej klasie.

```

class A {
    protected:
        bool flag;
};
class B : public A {};
class C : public A {};
class D : public B, public C {

```

```

public:
    void setFlag( bool nflag ){
        flag = nflag; // Niejednoznaczne.
    }
};

```

W tym przykładzie składowa `flag` jest zdefiniowana w klasie A. Klasa D dziedziczy po klasach B i C, które obie dziedziczą po klasie A, więc zasadniczo w klasie D znajdują się dwie kopie składowej `flag`, ponieważ w hierarchii klasy D klasa A występuje dwa razy. Którą z tych składowych należy ustawiać? Kompilator zgłosi błąd polegający na tym, że odniesienie do składowej `flag` w klasie D jest niejednoznaczne. Jednym z rozwiązań jest własnoręczne rozstrzygnięcie wątpliwości:

```
B::flag = nflag;
```

Inną możliwością jest zadeklarowanie B i C jako *wirtualnych klas bazowych*, dzięki czemu w hierarchii będzie mogła występować tylko jedna kopia A, co również wyeliminuje niejednoznaczność.

Istnieją jeszcze inne problemy z wielodziedziczeniem, np. dotyczące kolejności inicjacji klas bazowych podczas tworzenia obiektu klasy pochodnej czy niezamierzonego ukrywania składowych przed klasą pochodną. Aby uniknąć tego wszystkiego, w niektórych językach programowania zezwolono tylko na pojedyncze dziedziczenie. Podczas gdy to znacznie upraszcza kwestie dziedziczenia, jednocześnie je ogranicza, ponieważ tylko klasy o wspólnym przodku mogą mieć wspólne zachowania. Trochę łagodzą to interfejsy, które umożliwiają klasom z różnych hierarchii udostępnianie wspólnych interfejsów nawet wtedy, gdy nie są zaimplementowane przy użyciu wspólnego kodu.

PODSUMOWANIE

Obiektowe języki programowania są powszechnie używane, więc aby dostać pracę, zazwyczaj należy dobrze znać podstawy obiektowości. Kandydat musi wiedzieć, czym różni się klasa od obiektu, oraz znać pojęcia polimorfizmu i dziedziczenia.

Należy też znać różnice między językami programowania w odniesieniu do różnych aspektów obiektowości.

Skorowidz

A

- agencje pośrednictwa pracy, 29, 37
- aktualizacje wzorca Obserwator, 200
- aktywne oczekiwanie, busy waiting, 174
- algorytm Dijkstry, 106
- algorytmy
 - online, 157
 - rekurencyjne, 133, 136
 - sortujące, 149
- analiza
 - algorytmu, 54
 - czasu wykonywania, 55
 - działania funkcji, 53
 - rozwiązania, 50
 - zużycia pamięci, 55
- API, application programming interface, 186
- aplikacje
 - 32-bitowe, 265
 - 64-bitowe, 265
 - sieciowe, 266
- ASCII, 124
- automatyczne usuwanie, 263

B

- baza danych, 201, 207
- bezpieczeństwo aplikacji sieciowych, 266
- BFS, breadth-first search, 92
- binarne
 - drzewa poszukiwań, 90, 269
 - uzupełnienie dwójkowe, 214

- bity, 214
 - big-endian, 222
 - little-endian, 222
- blog, 33
- blokada pętlowa, spinlock, 176
- błędy w removeHead, 73
- BMP, Basic Multilingual Plane, 119
- BST, binary search tree, 90
- Budowniczy, Builder, 193

C

- C++ a Java, 259
- certyfikat, 31
- ciężka kulka, 238
- CV, 279
 - dostosowane do stanowiska, 295
 - inżynier, 279
 - korekta, 287
 - menedżer, 289
 - najważniejsze informacje, 286
 - odwrotna chronologia, 286
 - starszy programista, 289
 - wady, 279, 283
 - ważne informacje, 284
 - zwięzłość, 283, 285
- cykl, 82
- czarna lista, blacklisting, 267
- czas oczekiwania, latency, 265

D

debugger, 73
 Dekorator, 196, 199
 Dekorator kontra dziedziczenie, 199
 destruktor, 186
 DFS, depth-first search, 93
 diagnozowanie błędów, 28
 domieszka, salt, 268
 drzewa, 87

- AVL, 104
- binarne, binary tree, 89
- BST, 90
- czerwono-czarne, 104
- najniższy wspólny przodek, 99
- poszukiwań, 103
- przeглядanie, 93
- przeглядanie poprzeczne, 96
- przeглядanie wzdłużne, 96, 97
- przekształcanie w stertę, 100
- przeszukiwanie w głąb, 93
- przeszukiwanie wszerz, 92
- wysokość, 95

 dziecko, 89
 dziedziczenie, 184, 199, 263
 dziedziczenie pojedyncze, single inheritance, 189
 dzielenie problemu, 231

E

edukacja, 31
 element osiowy, pivot, 152
 eliminacja wywołania ogonowego, 134

F

Fabryka Abstrakcyjna, Abstract Factory, 195
 funkcja

- createStack, 67
- deleteStack, 66, 67
- getCharKey, 148
- insertAfter, 71
- intToStr, 129, 130
- pop, 65
- PrintTelephoneWords, 147
- push, 66
- removeHead, 73
- strlcpy(), 115
- strToInt, 129

 funkcje

- agregacyjne, 205
- inline, 261

opakowujące, 135
 rekurencyjne, 133

G

gra, 105
 grafika, 213
 grafy, 94

- nieskierowane, 94
- skierowane, 94

H

HCI, human computer interaction, 27
 hierarchia klas, 263
 HPC, high-performance computing, 176

I

implementacja

- stosu, 64
- wzorca Singleton, 196

 informacje o rynku, 29
 inicjacja

- leniwa, lazy initialization, 197
- opóźniona, deferred initialization, 197

 instrukcja SELECT, 203
 instrukcje preparowane, prepared statement, 267
 integralność referencyjna, 202
 interakcja komputera z człowiekiem, 27
 interfejs programistyczny, API, 186
 interfejsy, 186
 Iterator, 195

J

jedna ósma okręgu, 217
 język, 47

- C, 112
- C#, 113
- C++, 112, 259
- Java, 113, 259
- JavaScript, 114
- SQL, 202

K

klasa, 183

- Account, 172
- ActorGraphNode, 108
- Node, 96

klasy
 abstrakcyjne, 186
 zaprzyżnione, 259

klauzula WHERE, 209

klucz, 202
 obcy, 202
 wspólny, 204

kolejność
 big-endian, 224
 little-endian, 224

kombinacje łańcucha, 141

komparator, 161

konstrukcja GROUP BY, 209

konstruktor, 185

kontakt z firmą, 37

konwersja skanowania, 217

kończenie projektów, 32

kopiec, 92

korzeń, 88

krawędź, 94

kryptografia, 268
 publiczna, public cryptography, 268
 symetryczna, symmetric key cryptography, 268

kryptograficzna funkcja skrótów, 267

księgarnie, 30

kursy programistyczne, 30

L

leniwe ładowanie, 197

liczba
 jedynek, 224
 stacji benzynowych, 243

liczenie
 kostek, 248
 odniesień, 264
 otwartych schowków, 233
 populacji, population count, 227

LIFO, last-in-first-out, 64

LinkedIn, 33

lis i kaczka, 251

lista sąsiedztwa, adjacency list, 94

listy cykliczne, 60

listy potomne, 79

listy powiązane, 57
 cykliczne, 60
 dwustronne, 59
 element początkowy, 60
 implementacja stosu, 64
 jednostronne, 58
 przywracanie, 80
 spłaszczanie, 77

usuwanie elementów, 62
 wskaźnik do ogona, 69
 wstawianie elementów, 62
 zwracanie elementu, 75

listy powiązane dwustronnie, 59

listy powiązane jednostronnie, 58

liście, 89

logika trójwartościowa, 211

LSB, least significant bit, 223

Ł

łamigłówki, 229
 ciężka kulka, 238
 graficzne, 245
 liczba stacji, 243
 liczenie kostek, 248
 liczenie otwartych schowków, 233

lis i kaczka, 251

łódź i dok, 246

palące się lonty, 252

przechodzenie przez most, 236

przestrzenne, 245

trzy przełączniki, 235

uciekanie przed pociągami, 254

łańcuch skrótu, digest string, 267

łańcuchy, 114
 kombinacje, 141
 odwracanie kolejności słów, 123
 permutacje, 139
 usuwanie znaków, 120
 w języku C, 115
 w języku C#, 117
 w języku C++, 115
 w języku Java, 116
 w języku JavaScript, 117
 zamienianie na liczbę, 127
 zamienianie na łańcuch, 129

łowcy głów, headhunters, 36

łódź i dok, 246

M

macierz sąsiedztwa, adjacency matrix, 94

makra, 261

mechanizmy synchronizacji wątków, 170

menedżerzy, 289

metoda
 printValue, 96
 rotateRight, 104

Metoda Wytwórcza, factory method, 195

metody wirtualne, 188

model kooperacyjny, 170
 monitor, 170
 MSB, most significant bit, 223

N

najbardziej znaczący bit, 223
 najmniej znaczący bit, 223
 najniższy wspólny przodek, 99
 nakładanie się prostokątów, 219
 narzędzie marketingowe, 31
 negatywne aspekty oferty, 40
 negocjowanie wynagrodzenia, 41
 nietechniczne tematy, 271
 niezrównoważone binarne drzewo poszukiwań, 103
 notacja dużego O, 51–54

O

obiekty, 183
 obliczenie o wielkiej wydajności, 176
 Obserwator, 195, 200
 obsługa wskaźnika listy, 69
 odcisk cyfrowy, fingerprint, 267
 odrzucanie oferty, 42
 odwracanie kolejności słów, 123
 offshoring, 30
 ogon, tail, 58, 69
 operacja

- AND, 225
- push, 65

 operacje

- graficzne, 213
- na bitach, 213
- na listach powiązanych, 60

 operator >>>, 226
 operatory

- bitowe, 215
- logiczne, 216
- przesunięcia, 216

 optymalizacja

- algorytmu, 53
- wydajności, 216

 outsourcing, 30
 oznacz i usuń, mark and sweep, 264
 oznaczanie trójkolorowe, 264

P

palące się lonty, 252
 permutacje łańcucha, 139

pętla

- do-while, 131
- while, 131

 pisanie kodu, 28, 48
 planowanie wywłaszczające, preemptive scheduling, 170
 płaszczyzna BMP, 119
 początek, head, 58
 podejście metodyczne, 48
 podzapytania, 208
 polecanie, 36
 polimorfizm, 184
 portal LinkedIn, 33
 potomek, 89
 praca

- nad otwartymi projektami, 29
- nad projektem, 31
- w dużych firmach, 28
- w małej firmie, 29

 problem dotyczący

- baz danych, 207
- drzew binarnych, 89
- grafiki, 216
- grafów, 87
- łańcuchów, 117
- programowania obiektowego, 186
- rekurencji, 136
- sortowania, 155
- szperania przy bitach, 222
- tablic, 117
- współbieżności, 174, 178

 problemy

- należnikowe, pancake problems, 167
- szacunkowe, 232
- związane z wzorcami projektowymi, 196

 procedura

- pop, 65
- shakySort(), 161

 produkcja oprogramowania, 30
 profil

- internetowy, 32
- w portalu LinkedIn, 33

 programowanie

- aplikacji, 27
- obiektywne, 183
- systemowe, 27

 projektowanie interfejsów użytkownika, 27
 projekty

- długoterminowe, 29
- krótkoterminowe, 29

 prototypy funkcji, 260
 przechodzenie przez most, 236

przeglądanie
 list, 62
 poprzeczne, inorder, 93
 wsteczne, postorder, 93
 wzdłużne, preorder, 93, 96
 wzdłużne bez rekurencji, 97
 przekazywanie argumentów, 260
 przekształcanie drzewa, 100
 przełączanie kontekstu, 170
 przepustowość, bandwidth, 266
 przeszukiwanie
 w głąb, 93
 wszerz, 92
 przodek, 89
 przyjmowanie oferty, 42
 przykłady pytań merytorycznych, 258
 przypadek
 bazowy, 133
 rekurencyjny, 133
 przywracanie spłaszczonej listy, 80
 Publikacja-Subskrypcja, Publish-Subscribe, 195
 pytania dotyczące programowania, 45
 pytania merytoryczne, 257
 pytania nietechniczne, 271
 cele zawodowe, 274
 doświadczenie zawodowe, 274
 język programowania, 273
 styl pracy, 274
 zarobki, 275, 277
 zdolność adaptacji, 272
 zmiana pracy, 274

Q

QA, quality assurance, 28

R

referencja, 59, 87, 106
 rekruterzy, 37, 40, 41
 rekurencja, recursion, 133
 rekurencja ogonowa, tail recursion, 134
 relacyjne bazy danych, 201
 rodzaje list powiązanych, 58
 rodzic, 89
 rotacja drzewa, 104
 rotacja prawostronna, 104
 rozmowa kwalifikacyjna
 interaktywność, 47
 pisanie kodu, 48
 problemy, 46
 rozwiązywanie zadań, 48, 50

scenariusz, 46
 wzorce projektowe, 192
 rozmowy kwalifikacyjne, 23, 39
 rozmowy przesiewowe, screening interview, 38
 rozszerzenie znakowe, sign extension, 216
 rozwiązania
 iteracyjne, 135
 rekurencyjne, 135
 rozwiązywanie
 zadań, 48, 50
 zagadek, 229
 rynek pracy, 29
 rysowanie schematu, 245

S

semafory, 170
 z licznikiem, 171
 zdarzeniowe, 171
 sieci społecznościowe, 29
 Singleton, 193
 słabe referencje, 264
 słowa telefoniczne, 144
 słowo kluczowe friend, 260
 sortowanie, 149
 hybrydowe, 154
 naleśników, 166
 optymalizacja, 163
 przez odwracanie prefiksu, 167
 przez scalanie, merge sort, 154
 przez wstawianie, insertion sort, 151
 przez wybieranie, selection sort, 150
 przy użyciu wielu kluczy, 160
 stabilne, 158, 161
 szybkie, quicksort, 152
 tablic, 100, 102
 wybór algorytmu, 155
 wydajność algorytmu, 157
 spłaszczanie listy, 77
 SQL, Structured Query Language, 202
 stała wskaźnikowa, 112
 starsi programiści, 289
 staże, 32
 sterta, heap, 92
 maksymalna, 92
 przeszukiwanie wszerz, 92
 stos, 64
 strona osobista, 33
 strój, 39
 symbol \neg , 222
 szukanie firm, 35

Ś

śledzący ośmiemiecacz pamięci, 264
 śmieciarka, garbage collector, 186

T

tablice, 88, 111
 dynamiczne, 64, 112
 mieszające, 119, 269
 postrzępione, jagged arrays, 113
 statyczne, 112
 targi pracy, 38
 testowanie, 28
 tęczowe tabele, rainbow table, 268
 transakcje, 205
 atomowość, 206
 izolacja, 206
 spójność, 206
 trwałość, 206
 w bazach danych, 206
 trójargumentowa operacja
 AND, 211
 NOT, 211
 OR, 211
 trzy przełączniki, 235
 tworzenie
 obiektu, 185
 sterty, 101

U

uciekanie przed pociągami, 254
 uczujący filozofowie, 178
 Unicode, 119
 unikanie problemów, 232
 usuwanie
 elementów listy, 62
 nieużywanych obiektów, 263
 określonych znaków, 120
 UX, user experience, 27
 użycie wątków, 172

W

wartość NULL, 70, 82, 209
 wątki
 aktywne oczekiwanie, 174
 główne, 170
 konsumenta, 176
 macierzyste, native thread, 170

 monitory, 170
 poziomu jądra, kernel-level thread, 170
 producenta, 176
 semafory, 170
 systemowe, 170
 użytkownika, 170
 zakleszczenia, 171
 zielone, green thread, 170

wersja
 iteracyjna, 138
 rekurencyjna, 137
 węzeł, 87
 wielodziedziczenie, multiple inheritance, 189
 wierzchołek, 94
 więzy integralności, 202
 właściwości transakcji, 206
 wpisy na forach, 33
 wrażenia użytkownika, 27
 wskaźnik, 58, 112
 wskaźnik do ogona, 80
 współbieżność, 169
 wybór języka, 47
 wydajność sieci, 265
 wykształcenie, 31
 wysokość drzewa, 95
 wyszukiwanie binarne, 137
 wyświetlacz rastrowy, 213
 wzorce
 czynnościowe, 192, 195
 kreatywne, 192, 193
 projektowe, 191
 strukturalne, 192, 196
 wzorzec
 Budowniczy, 193
 Dekorator, 196, 199
 Fabryka Abstrakcyjna, 195
 Iterator, 195
 Metoda Wytwórcza, 195
 Obserwator, 195, 200
 Publikacja-Subskrypcja, 195
 Singleton, 193

Z

zakleszczenie wątków, deadlock, 171
 zapewnianie jakości, 28
 zapętlenie, livelock, 180
 zarządzanie, 28, 31
 zasada Hornera, 128
 zdolność adaptacji, 272
 zlecenia zewnętrzne, 30

złączenie

wewnętrzne, 204

zewewnętrzne, 204

zewewnętrzne lewe, 205

zewewnętrzne pełne, 205

zewewnętrzne prawe, 205

złożoność

czasowa, 91

obliczeniowa, 54, 91

znajdowanie

największej wartości, 209

znaku, 117

znak, 114

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION

- 
1. ZAREJESTRUJ SIĘ
 2. PREZENTUJ KSIĄŻKI
 3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW
w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

<http://program-partnerski.helion.pl>

GRUPA WYDAWNICZA



Helion SA

Lektura obowiązkowa przed rozmową rekrutacyjną!

Rozmowa rekrutacyjna to ten rodzaj spotkania, które wywołuje stres nawet u najbardziej odpornych. Podechwytlwe pytania, zadania do rozwiązania, skomplikowane testy dotyczące spraw technicznych oraz nierzadko kilkugodzinny dialog wymagają dużej odporności i wiedzy. Zastanawiasz się, jak przygotować się do tej rozmowy? Na to pytanie nie ma jednej odpowiedzi, ale wszystkie znajdziesz w tej książce!

Ta lektura doskonale przygotowuje Cię do kolejnej rozmowy rekrutacyjnej. Na samym początku dowiesz się, jak szukać ofert pracy oraz jak rozpocząć proces ubiegania się o wybrane stanowisko. W kolejnych rozdziałach zaznajomisz się z różnymi problemami programistycznymi oraz sposobami ich rozwiązania. Zobacz, czym mogą próbować Cię zaskoczyć rekruterzy w obszarach związanych z listami, drzewami, grafami i rekurencją. Poznaj najczęściej zadawane pytania dotyczące wzorców projektowych, programowania obiektowego oraz baz danych. Książka ta jest obowiązkową lekturą dla wszystkich osób chcących pomyślnie przejść proces rekrutacji na wymarzone stanowisko!

Dzięki tej książce:

- nauczysz się skutecznie szukać odpowiednich dla Ciebie ofert
- stworzysz profesjonalny życiorys
- poznasz pytania najczęściej pojawiające się podczas rozmów rekrutacyjnych
- zrobisz miły krok do przodu w kierunku wymarzonej pracy

Helion

27246 numer katalogowy

księgarnia internetowa

<http://helion.pl>

zamówienia telefoniczne



0 801 339900



0 601 339900

informatyka w najlepszym wydaniu

Sprawdź najnowsze promocje:
• <http://helion.pl/promocje>
Książki najchętniej czytane:
• <http://helion.pl/bestsellery>
Zamów informacje o nowościach:
• <http://helion.pl/nowosci>

Helion SA
ul. Kościuski 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
<http://helion.pl>

sięgnij po WIĘCEJ



KOD KORZYSCI

ISBN 978-83-246-9861-5



cena: 49,00 zł

Wrox
An Imprint of
 WILEY