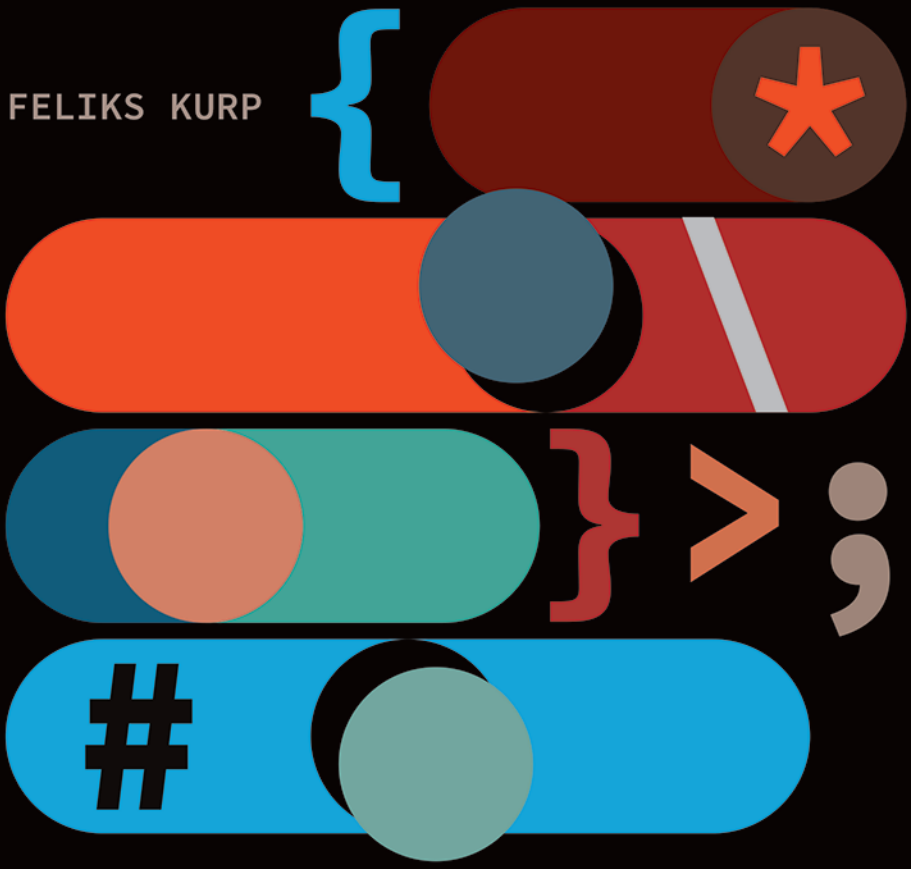
FELIKS KURP



JĘZYKI I PARADYGMATY PROGRAMOWANIA

TEORIA I PRAKTYKA

Helion

Redaktor prowadzący: Małgorzata Kulik

Projekt okładki: Studio Gravite/Olsztyn
Obarek, Pokoński, Pazdrijowski, Zaprucki

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)

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

ISBN: 978-83-289-0813-0

Printed in Poland.

Spis treści

Wstęp

Żyjemy w okresie rozwoju cywilizacji cyfrowej, w której umiejętność programowania komputerowego jest jedną z najbardziej pożądanych umiejętności, a zawód programisty — jednym z najbardziej poszukiwanych. Proces rozwoju oprogramowania ciągle przyspiesza. Oprogramowanie towarzyszy nam praktycznie wszędzie, chociaż często sobie tego nie uświadamiamy.

Jeśli sięgnąć do historii, to jeszcze stosunkowo niedawno, bo w latach 60. ubiegłego wieku, możliwości pierwszych komputerów, mimo ich znacznych rozmiarów, zaledwie przewyższały możliwości zwykłych kalkulatorów.

Obecnie trudno znaleźć dziedzinę techniki, ekonomii, finansów, życia społecznego, edukacji, twórczości artystycznej i rozrywki, gdzie nie spotkalibyśmy oprogramowania komputerowego. Jeszcze kilka lat temu zafascynowani byliśmy *internetem rzeczy* (ang. *Internet of Things*). Dziś już nas specjalnie nie dziwi, że różne urządzenia techniczne porozumiewają się między sobą, wykonując różne czynności bez naszego udziału. Wszystkie te sytuacje wymagają wcześniejszego, nieraz bardzo złożonego, oprogramowania.

Prawdziwa rewolucja jednak dopiero przed nami. Wkraczamy w epokę, w której rządzić będzie sztuczna inteligencja (AI). Jeszcze nie wiemy, czy to dobrze, czy źle. Dziedzina ta wymaga bardzo zawansowanego, specjalistycznego oprogramowania i dużych zasobów sprzętowych.

W chwili obecnej mamy do dyspozycji co najmniej 20 liczących się, ale bardzo różnych języków programowania. Odpowiedź na często zadawane pytanie o to, *jakiego języka warto się uczyć*, jest bardzo trudna, nawet jeżeli wiemy, jakiego rodzaju systemy chcemy budować. I to z trzech powodów.

Po pierwsze dlatego, że nie jesteśmy w stanie w krótkim czasie opanować praktycznie choćby podstaw wielu języków jednocześnie w celu porównania ich własności.

Po drugie dlatego, że współczesne języki programowania bardzo szybko się rozwijają i trudno jest za tym rozwojem nadążać.

Po trzecie, że powstają jak grzyby po deszczu coraz to nowsze środowiska uruchomieniowe i narzędzia wspomagające tworzenie oprogramowania, komplikując jednocześnie sytuację adeptom nauki programowania. Autor tej książki ma nadzieję, że po zapoznaniu się z jej treścią czytelnik będzie miał ułatwione zadanie w odpowiedzi na wyżej postawione pytanie „Jakiego języka programowania warto się uczyć?".

Celem tej książki nie jest, jak to zwykle bywa, twarda nauka programowania w wybranym języku, ale przede wszystkim zapoznanie się z podstawowymi własnościami i możliwymi zastosowaniami kilku odległych od siebie, ale aktualnie ważnych języków programowania. Podążać będziemy w kierunku pogłębionego zrozumienia filozofii programowania. Z wieloletniego doświadczenia autora jako dydaktyka wynika bowiem, że im większe zrozumienie towarzyszy naszemu działaniu, tym bardziej jesteśmy w tym działaniu twórczy i efektywni. Dotyczy to zwłaszcza tak skomplikowanej czynności jak programowanie.

Nie znaczy to jednak, że w trakcie studiowania treści tej książki nie nabędziemy praktycznych umiejętności programistycznych. Wręcz przeciwnie. Tyle tylko, że ograniczą się one do podstawowych konstrukcji omawianych języków programowania. Zawarte w książce starannie dobrane kody zilustrują i pokażą możliwe zastosowania wybranych języków programowania. Pomocą w nauce będą też zadania programistyczne do samodzielnego rozwiązania.

ROZDZIAŁ 1.

Paradygmaty programowania

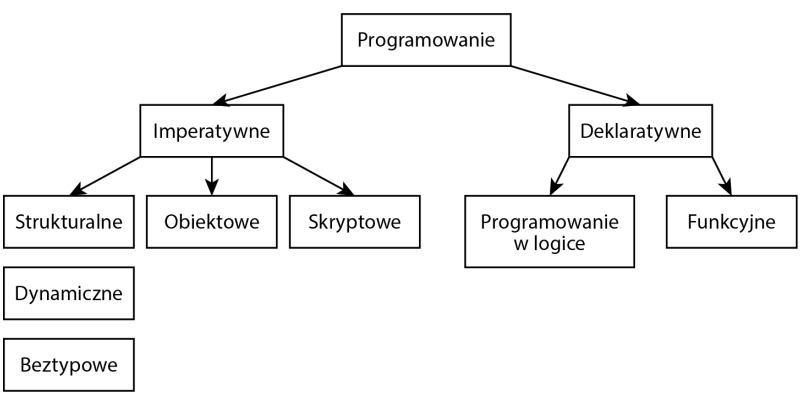
Paradygmat (z greckiego: wzorzec, przykład) to, jak podaje *Słownik języka polskiego PWN*, „przyjęty sposób widzenia rzeczywistości w danej dziedzinie, doktrynie itp.”.

W informatyce ***paradygmat programowania*** rozumieć możemy jako *pewną filozofię podejścia do tworzenia programu*.

W praktyce wyraża się on przede wszystkim poprzez zbiór mechanizmów, jakich używa programista, pisząc program, oraz przez to, w jaki sposób program ten jest wykonywany przez komputer.

Jest oczywiste, że oczekiwania programisty są inne, gdy tworzy system, w którym *dominuje przetwarzanie danych*, inne — gdy tworzy system z *dominującym przepływem sterowania* (systemy decyzyjne, gry komputerowe), a jeszcze inne — gdy tworzy *aplikację internetową*, i trochę inne — gdy tworzy *aplikację mobilną*.

Rysunek 1.1 pokazuje główne aktualnie wykorzystywane paradygmaty programowania oraz współzależności między niektórymi z nich.



RYSUNEK 1.1. Ogólna struktura współzależności głównych paradygmatów programowania

Pojęcie paradygmatu programowania pozwala w znacznym stopniu uporządkować chaos związany z istnieniem wielu języków programowania, które w różnym zakresie, a nawet w różny sposób realizują postawione im zadania programistyczne.

To nie języki programowania definiują paradygmaty programowania, ale odwrotnie — języki programowania mogą wspierać różne paradygmaty programowania.

Pojęcie paradygmatu programowania traktować będziemy, po pierwsze jako zbiór zaleceń dla programisty tworzącego program z wykorzystaniem wybranego paradygmatu programowania i po drugie jako pewien zbiór zakazów i ograniczeń związanych z danym paradygmatem programowania.

W dalszej części książki zamiast słowa *paradygmat* używać będziemy zamiennie pojęcia *programowanie*.

Na początek rozważymy, czym różni się podejście *imperatywne* od *deklaratywnego*? Otóż programy zapisane w podejściu **imperatywnym** oparte są na algorytmach, których zasadniczym celem są *obliczenia pewnych wartości w oparciu o dostarczone dane*. Odbywa się to poprzez wykonywanie rozkazów skierowanych do komputera. Łacińskie słowo *impero* oznacza „rozkazywać".

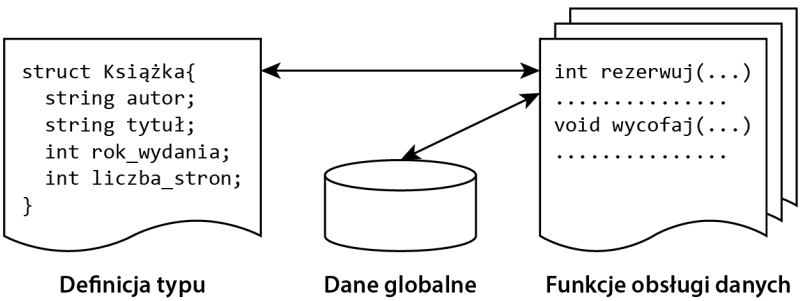
Natomiast programując w języku **deklaratywnym**, programista zamiast opisywać sposób rozwiązania problemu, to jest zapisywać algorytm przetwarzania danych (co stanowi istotę języków imperatywnych), *opisuje w języku logiki lub za pomocą funkcji pewną rzeczywistość* (podając fakty i związki między nimi).

1.1. Programowanie strukturalne

Przykładem programowania imperatywnego jest programowanie strukturalne.

Programowanie strukturalne to paradygmat programowania zalecający dzielenie kodu na procedury (*funkcje*), to jest fragmenty wykonujące ściśle określone operacje. Funkcje te mogą wchodzić w skład większych *modułów programowych*. Charakterystyczną cechą tego

paradygmatu tworzenia oprogramowania jest rozproszenie w programie takich elementów programu jak tworzone przez użytkownika złożone **typy danych** (struktury, tablice, listy), następnie **funkcje** ich obsługi oraz same **dane** (rysunek 1.2).



RYSUNEK 1.2. Ogólny schemat aplikacji zbudowanej zgodnie z paradygmatem programowania strukturalnego

Pokazana na rysunku 1.2 struktura o nazwie `Książka` jest przykładem złożonego typu danych, który grupuje dane różnych typów, pozwalając opisać obiekty w pożądany sposób.

W paradygmacie programowania strukturalnego zaleca się, aby funkcje obsługi danych globalnych obsługiwały te dane poprzez swoje parametry umożliwiające dostęp do tych danych.

Jeśli chodzi o zakazy i ograniczenia, to takie instrukcje (poznamy je częściowo później) jak `goto`, `break`, `continue`, `switch`, jakkolwiek dostępne, są tutaj na ogół niemile widziane, jako niestrukturalne, zaburzające logikę kodu i czyniące ją mniej zrozumiałą.

Języki programowania strukturalnego realizujące ściśle powyższe zalecenia to klasyczne języki programowania: **Basic**, **Fortran**, **Pascal**, **C**. Wraz z pojawieniem się *podejścia obiektowego* programowanie strukturalne zaczęło przechodzić do historii. Dopiero język **Python**, bardzo obecnie popularny, realizuje w znacznym zakresie ideę programowania strukturalnego. Jego popularność wynika z potrzeby tworzenia małych rozproszonych programów obsługujących strony internetowe, aplikacje mobilne, systemy sterowania i gry komputerowe.

1.2. Programowanie obiektowe

Programowanie obiektowe pojawiło się stosunkowo dawno, bo już w latach 60. ubiegłego wieku. Musiało upłynąć jednak sporo czasu, zanim wraz z rozwojem możliwości sprzętowych to podejście mogło być w pełni rozwinięte.

Szybko bowiem stało się oczywiste, że podejście strukturalne, głównie ze względu na rozproszenie kodu, jest nie do przyjęcia w sytuacji tworzenia dużych systemów informatycznych. Podejście obiektowe spowodowało znaczne (można by powiedzieć — bardzo znaczne) uporządkowanie kodu. Pozwoliło też modelować obiektowo w sposób systematyczny i czytelny systemy o dowolnych rozmiarach.

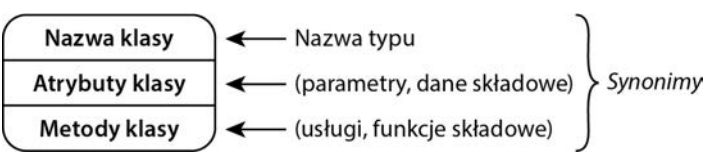
Programowanie obiektowe (OOP — ang. *object-oriented programming*) jest paradygmatem programowania, gdzie zasadniczą część programu stanowią definicje klas.

Klasy zawierają:

- podobnie jak w strukturach, **deklaracje danych składowych klasy**,
- **funkcje składowe klasy**, których zadaniem jest obsługa danych składowych, ponadto komunikacja z obiektami innych klas; funkcje te nazywamy też **metodami klas**.

Klasy nie występują pojedynczo. Zwykle definiuje się zbiór różnych, wzajemnie ze sobą powiązanych i zależnych klas, które tworzą tak zwany **model obiektowy**. Za pomocą takiego modelu możemy opisać każdą, nawet najbardziej złożoną rzeczywistość. Wynika stąd ogromne znaczenie i popularność paradygmatu obiektowego.

Ogólnie struktura definicji klasy ma postać uwidocznioną na rysunku 1.3.



RYSUNEK 1.3. Struktura definicji klasy

Aby jednak otrzymać działający program, trzeba poza modelem obiektowym zbudować część wykonawczą programu obiektowego.

W części wykonawczej programu tworzone są konkretne obiekty, zazwyczaj różnych klas, a realizacja programu oparta jest na *komunikacji międzyobiektowej*.

Obiekty są *egzemplifikacjami klas*. O ile klasa jest pewną abstrakcyjną definicją, opisującą pewien wycinek rzeczywistości, to obiekt jest konkretem tej rzeczywistości — jego dane przechowują konkretne wartości, a metody są odpowiedzialne za obsługę danych składowych klasy, jak również komunikowanie się z innymi obiektami tej samej klasy lub obiektami innych klas.

Każdy obiekt reprezentuje jakąś klasę. Może istnieć jednocześnie wiele różnych obiektów tej samej klasy. Mają one jednak wspólne metody obsługi danych składowych, ponieważ należą do tej samej klasy.

Paradygmat programowania obiektowego jest tak doskonały, że można śmiało zaryzykować tezę, iż nie ma lepszej metody opisu rzeczywistości niż podejście obiektowe.

Nie wszystkie współczesne języki programowania realizują w pełnym zakresie programowanie obiektowe. Prawie wszystkie jednak mają ambicję realizacji obiektowości. Języki, które *w pełni* realizują paradygmat obiektowości, to przede wszystkim **C++**, **C#** i **Java**.

Większość pozostałych języków programowania imperatywnego, zaczynając od tych stojących na drugim biegunie (jak na przykład język Python), ma ambicję, aby poszerzać swoje możliwości, zarówno o programowanie obiektowe, jak i funkcyjne.

Jak wynika z rysunku 1.1, programowanie obiektowe zaliczamy do programowania imperatywnego, ponieważ realizuje ono (chociaż w inny sposób) cele programowania imperatywnego. Do programowania imperatywnego zaliczamy również **programowanie skryptowe**. W tym paradygmacie uwaga jest skupiona przede wszystkim na komunikacji z użytkownikiem. Typowym zastosowaniem są gry komputerowe, gdzie skrypty służą do sterowania przebiegiem gry.

1.3. Programowanie deklaratywne

Na drugim biegunie względem programowania imperatywnego mamy **programowanie deklaratywne**, które opisuje rzeczywistość w zupełnie inny sposób. Najważniejsze paradygmaty programowania deklaratywnego to **programowanie funkcyjne** i **programowanie w logice**. Są to aktualnie bardzo ważne paradygmaty programowania (zwłaszcza programowanie funkcyjne). Dlatego też programowaniu funkcyjnemu i programowaniu w logice poświęcone będą odrębne obszerne rozdziały książki.

1.4. Pozostałe paradygmaty programowania

Poza omówionymi wyżej paradygmatami programowania występuje szereg innych paradygmatów, wspieranych przez różne języki programowania. Omówmy krótko najważniejsze z nich.

1.4.1. Paradygmat programowania dynamicznego

Paradygmat **programowania dynamicznego** realizują języki, które w czasie wykonywania programu są w stanie: zmieniać typy zmiennych, zmieniać rozmiary danych złożonych, wstrzykiwać nowy kod do funkcji i przeprowadzać inne jeszcze operacje, które zwykle w językach wysokiego poziomu takich jak C++, Java są realizowane na etapie *kompilacji*. Siłą języka programowania Python jest właśnie możliwość programowania dynamicznego w znacznym zakresie. Inne języki również starają się wspierać paradygmat programowania dynamicznego. Przykładem jest popularny język obiektowy C#.

1.4.2. Paradygmat programowania uogólnionego (generycznego)

Paradygmat ten pozwala na pisanie szablonów kodu programu bez wcześniejszego określenia typów danych, na których kod ten będzie działał. Programowanie uogólnione ma bliski związek z metaprogramowaniem, w przeciwieństwie jednak do niego nie wymaga od programisty generowania kodu w sposób jawny.

Jest wspierane przez takie popularne języki jak C++, Java, C#, Haskell, TypeScript.

1.4.3. Języki kompilowane, interpretowane i beztypowe

Z punktu widzenia omówionych w podrozdziałach 1.1 i 1.2 dwóch kategorii prawie wszystkie języki programowania możemy podzielić na dwie duże grupy języków. Do pierwszej zaliczymy języki, które wymagają jawnej kompilacji całego zapisanego kodu przed jego uruchomieniem.

Kompilacja to proces, który polega na sprawdzaniu poprawności składniowej i semantycznej (znaczeniowej) napisanego kodu. Kompilacja może dotyczyć zarówno pojedynczej instrukcji, funkcji, definicji klasy, a nawet całego modelu obiektowego zbudowanego z wielu klas. W przypadku omawianych w podrozdziale 1.2 języków obiektowych wprowadzenie jakichkolwiek zmian w kodzie wymaga jawnej kompilacji całego kodu. Dlatego języki te nazywamy **językami kompilowanymi**. Do grupy tej należą takie popularne języki jak C++, C# i Java.

Do drugiej grupy zaliczamy języki, które nie wymagają wcześniejszej kompilacji przed uruchomieniem kodu. Kod jest bowiem po jego uruchomieniu na bieżąco (polecenie za poleceniem) analizowany i w przypadku braku błędów natychmiast wykonywany. Języki te nazywamy **językami interpretowanymi**. Typowym przedstawicielem jest popularny język Python.

Języki beztypowe to języki, które w fazie pisania programu nie wymagają określania typu deklarowanych zmiennych. Typ danej jest określany automatycznie w momencie przypisania wartości tym zmiennym.

Pamiętajmy, że języki mogą wspierać jednocześnie (na ogół w różnym zakresie) różne paradygmaty programowania. Na przykład o języku Python mówimy, że jest językiem wieloparadygmatowym.

Omawiając różne paradygmaty programowania i języki programowania, jak również związki między pierwszymi i drugimi, robimy to jednak, jak łatwo zauważyć, w sposób bardzo ogólny. Tymczasem programowanie jest czynnością konkretną, czysto praktyczną. Stąd wynika pozorna przewaga umiejętności nad wiedzą. Wszyscy jednak zgodzimy się z faktem, że szczegółowe umiejętności bez szerszego spojrzenia na tak skomplikowaną materię i wielość różnorodnych języków programowania czynią nas mało twórczymi. Mimo naszych umiejętności możemy poczuć się jak mały trybik w ogromnej machinie systemów

informatycznych. Zadaniem tej książki jest między innymi, aby czytelnicy wyrobili sobie pogląd na to, czym jest w istocie programowanie i jakie są jego możliwości i ograniczenia.

Miernikiem ważności języka programowania w danym okresie jest jego *popularność*. Popularność języka jest głównie dyktowana przez rynek, to jest zapotrzebowanie na programistów potrafiących programować w danym języku, z czym związane są ich uposażenia. W chwili obecnej najbardziej popularnymi językami są Python, Java, JavaScript, C++ i C#.

W kolejnych rozdziałach poprzez omawianie licznych przykładów i rozwiązywanie problemów o różnym charakterze poznawać będziemy praktycznie paradygmaty programowania realizowane w wybranych językach programowania. Najwięcej uwagi zostanie poświęcone językom **Python**, **Java SE** oraz **JavaScript** (w zakresie *podstaw programowania funkcyjnego*).


PROGRAM
PARTNERSKI
GRUPY 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
Helion

Czym w rzeczywistości jest programowanie? I jak zacząć programować?

Oprogramowanie jest dziś praktycznie wszędzie, a programiści od dawna należą do najbardziej poszukiwanych specjalistów. Na podstawie napisanego przez nich kodu funkcjonują już nie tylko komputery i smartfony. Oprogramowanie steruje sprzętami domowymi, telewizorem czy lodówką. W ramach tak zwanego internetu rzeczy wiele urządzeń technicznych komunikuje się między sobą bez udziału człowieka. Gwałtownie rozwija się sztuczna inteligencja, wymagająca specjalistycznego oprogramowania. Nie dziwi więc, że jego rozwój ciągle przyspiesza. W obliczu tych faktów odpowiedź na pytanie, **jakiego języka programowania warto się nauczyć**, jest trudna. Nawet dla osoby, która wie, w jaki sposób zamierza w przyszłości korzystać ze swoich informatycznych umiejętności.

Autor książki proponuje nieco inne podejście do nauki programowania. Zachęca do zapoznania się z podstawowymi własnościami i możliwymi zastosowaniami kilku odległych od siebie, ale niezwykle ważnych aktualnie języków programowania, takich jak **Python**, **Java SE**, **JavaScript** i **Prolog**. W trakcie ich poznawania czytelnicy będą mieli okazję zgłębić filozofię programowania, a równocześnie zdobywać praktyczne umiejętności programistyczne na podstawowym poziomie. Starannie dobrany kod pokazuje możliwe zastosowania wybranych języków programowania. Pomoc w nauce stanowią też zadania do samodzielnego rozwiązania.

Helion

helion.pl

HELION S.A.
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

KOD KORZYŚCI
Sięgnij po więcej! ▶

ISBN 978-83-289-0813-0

9 788328 908130

Cena: 44,90 zł