

The background of the cover is a solid teal color. In the center, there is a faint, semi-transparent illustration of a horse's lower legs and hooves, rendered in a darker shade of teal. The hooves are positioned as if the horse is standing on a surface, with the front hooves slightly ahead of the back ones.

Programowanie w języku Kotlin

THE BIG NERD RANCH GUIDE

Josh Skeen, David Greenhalgh

Tytuł oryginału: Kotlin Programming: The Big Nerd Ranch Guide

Tłumaczenie: Piotr Rajca

ISBN: 978-83-283-5536-1

Authorized translation from the English language edition, entitled KOTLIN PROGRAMMING: THE BIG NERD RANCH GUIDE, 1st Edition by SKEEN, JOSH; GREENHALGH, DAVID; published by Pearson Education, Inc, publishing as The Big Nerd Ranch Guides.
Copyright © 2018 Big Nerd Ranch, LLC

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from Pearson Education Inc.

Polish language edition published by HELION S.A. Copyright © 2019.

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 Helion SA 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 Helion SA nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Materiały graficzne na okładce zostały wykorzystane za zgodą Shutterstock Images LLC.

Helion SA

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)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:

<ftp://ftp.helion.pl/przyklady/prokot.zip>

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

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

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

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

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

Spis treści

Podziękowania	13
Prezentacja Kotliny	15
Dlaczego Kotlin?	15
Dla kogo jest przeznaczona ta książka?	16
Jak korzystać z tej książki?	16
Dla ciekawskich	17
Wyzwania	17
Konwencje typograficzne	17
Patrząc w przyszłość	17
1 Pierwsza aplikacja w Kotlinie	19
Instalowanie IntelliJ IDEA	19
Pierwszy projekt programu w Kotlinie	20
Tworzenie pierwszego pliku źródłowego w Kotlinie	25
Wykonywanie pliku źródłowego	27
Kotlin REPL	29
Dla ciekawskich: Dlaczego warto używać właśnie IntelliJ?	30
Dla ciekawskich: Pisanie kodu przeznaczonego na JVM	31
Wyzwanie: Arytmetyka REPL	32
2 Zmienne, stałe i typy	33
Typy	33
Deklarowanie zmiennych	34
Wbudowane typy języka Kotlin	36
Zmienne tylko do odczytu	37
Wnioskowanie typów	40
Stałe czasu kompilacji	41
Oglądanie kodów bajtowych	42
Dla ciekawskich: Podstawowe typy danych Javy w Kotlinie	45
Wyzwanie: hasSteed	46
Wyzwanie: Szynek Hipolit'a	46
Wyzwanie: Magiczne lustro	46

3	Instrukcje warunkowe	47
	Instrukcje if/else	47
	Dodawanie kolejnych warunków	50
	Zagnieżdżone instrukcje if/else	52
	Bardziej eleganckie wyrażenia warunkowe	53
	Zakresy	59
	Wyrażenia when	60
	Szablony łańcuchowe	62
	Wyzwanie: Eksperymenty z zakresami	64
	Wyzwanie: Rozbudowa aury	64
	Wyzwanie: Konfigurowalny format statusu	65
4	Funkcje	67
	Wyodrębnianie kodu do funkcji	67
	Anatomia funkcji	69
	Nagłówek funkcji	70
	Ciało funkcji	72
	Zasięg funkcji	73
	Wywoływanie funkcji	74
	Refaktoryzacja funkcji	75
	Pisanie własnych funkcji	76
	Argumenty domyślne	78
	Funkcje jednowyrażeniowe	79
	Funkcje typu Unit	80
	Nazwane argumenty funkcji	81
	Dla ciekawskich: Typ Nothing	82
	Dla ciekawskich: Funkcje plikowe w Javie	83
	Dla ciekawskich: Przeciążanie funkcji	84
	Dla ciekawskich: Nazwy funkcji w odwrotnych apostrofach	85
	Wyzwanie: Funkcje jednowyrażeniowe	87
	Wyzwanie: Poziom upojenia magicznego	87
	Wyzwanie: Status upojenia magicznego	87
5	Funkcje anonimowe i typ funkcyjny	89
	Funkcje anonimowe	89
	Typ funkcyjny	91
	Niejawne instrukcje return	92
	Argumenty funkcyjne	92
	Słowo kluczowe it	93
	Akceptowanie wielu argumentów	94
	Wsparcie dla wnioskowania typów	95
	Definiowanie funkcji akceptujących inne funkcje	96
	Składnia skrócona	97
	Wpisywanie funkcji	98
	Referencje funkcji	99
	Typ funkcyjny jako typ wyniku	100
	Dla ciekawskich: Funkcje lambda w Kotlinie są domknięciami	102
	Dla ciekawskich: Funkcje lambda a anonimowe klasy wewnętrzne	102

6	Bezpieczeństwo wartości pustych i wyjątki	105
	Akceptowanie wartości pustych	105
	Jawny typ null w Kotlinie	107
	W czasie kompilacji czy w czasie wykonywania?	108
	Bezpieczeństwo wartości pustych	109
	Opcja pierwsza: bezpieczny operator wywołania	110
	Opcja druga: operator podwójnego wykrzyknika	111
	Opcja trzecia: użycie if do sprawdzania, czy wartość jest równa null	112
	Wyjątki	115
	Zgłaszanie wyjątków	116
	Niestandardowe wyjątki	117
	Obsługa wyjątków	118
	Warunki wstępne	120
	Null: do czego się przydaje?	122
	Dla ciekawskich: Wyjątki sprawdzane i niesprawdzone	123
	Dla ciekawskich: Jak wymuszana jest możliwość stosowania wartości null?	123
7	Łańcuchy	125
	Pobieranie fragmentów łańcuchów	125
	Funkcja substring	125
	Funkcja split	127
	Operacje na łańcuchach	129
	Łańcuchy są niezmiennie	131
	Porównywanie łańcuchów	131
	Dla ciekawskich: Unicode	133
	Dla ciekawskich: Przeglądanie znaków w łańcuchu	133
	Wyzwanie: Usprawnianie smoczjej mowy	134
8	Liczby	135
	Typy liczbowe	135
	Liczby całkowite	136
	Liczby dziesiętne	138
	Konwersja łańcuchów na typy liczbowe	138
	Konwersja typu Int na Double	139
	Formatowanie wartości typu Double	141
	Konwertowanie wartości typu Double na Int	142
	Dla ciekawskich: Operacje bitowe	143
	Wyzwanie: Pozostałe kwaterki	144
	Wyzwanie: Obsługa ujemnego stanu sakiewki	144
	Wyzwanie: Smoczykojn	145
9	Funkcje standardowe	147
	Funkcja apply	147
	Funkcja let	148
	Funkcja run	149

Funkcja with	150
Funkcja also	151
Funkcja takeIf	151
Funkcja takeUnless	152
Stosowanie funkcji standardowych	152
10 Listy i zbiory	155
Listy	155
Dostęp do elementów listy	157
Zmianianie zawartości listy	159
Iteracja	163
Wczytywanie pliku do listy	167
Destrukturyzacja	169
Zbiory	169
Tworzenie zbiorów	169
Dodawanie elementów do zbioru	171
Pętle while	173
Wyrażenie break	175
Konwersje kolekcji	175
Dla ciekawskich: Typy tablicowe	176
Dla ciekawskich: Tylko do odczytu a niezmienny	177
Wyzwanie: Formatowanie menu gospody	178
Wyzwanie: Zaawansowane formatowanie menu	179
11 Mapy	181
Tworzenie map	181
Dostęp do wartości mapy	183
Dodawanie elementów do map	183
Modyfikowanie wartości mapy	186
Wyzwanie: Wykidajło	190
12 Definiowanie klas	191
Definiowanie klasy	191
Tworzenie instancji	191
Funkcje klasowe	192
Widoczność i hermetyzacja	194
Właściwości klas	195
Akcesory get i set i ich właściwości	197
Widoczność właściwości	200
Właściwości obliczane	200
Refaktoryzacja kodu projektu NyetHack	201
Stosowanie pakietów	209
Dla ciekawskich: Bliższe spojrzenie na właściwości var i val	210
Dla ciekawskich: Zapobieganie występowaniu wyścigu	213
Dla ciekawskich: prywatny w pakiecie	214

13 Inicjalizacja	217
Konstruktory	218
Konstruktory podstawowe	218
Definiowanie właściwości w konstruktorze podstawowym	219
Konstruktory dodatkowe	220
Argumenty domyślne	221
Argumenty nazwane	222
Blok inicjalizatora	223
Inicjalizacja właściwości	224
Kolejność inicjalizacji	227
Odraczanie inicjalizacji	228
Inicjalizacja opóźniona	228
Inicjalizacja leniwa	229
Dla ciekawskich: Kruczki inicjalizacji	231
Wyzwanie: Zagadka Excalibura	233
14 Dziedziczenie	235
Definiowanie klasy Room	235
Tworzenie klas pochodnych	236
Sprawdzanie typów	242
Hierarchia typów w języku Kotlin	244
Rzutowanie typów	245
Inteligentne rzutowanie	246
Dla ciekawskich: Klasa Any	247
15 Obiekty	249
Słowo kluczowe object	249
Deklaracje obiektów	250
Wyrażenie obiektowe	255
Obiekty uzupełniające	255
Klasy zagnieżdżone	256
Klasy danych	259
Funkcja toString	260
Funkcja equals	261
Funkcja copy	261
Deklaracje destrukuryzujące	261
Klasy wyliczeniowe	263
Przeciążanie operatorów	264
Eksplorowanie świata NyetHack	266
Dla ciekawskich: Definiowanie porównań strukturalnych	269
Dla ciekawskich: Algebraiczne typy danych	271
Wyzwanie: Polecenie „Wyjdz”	273
Wyzwanie: Implementacja mapy świata	274
Wyzwanie: Dzwonimy	274

16	Interfejsy i klasy abstrakcyjne	275
	Definiowanie interfejsów	275
	Implementacja interfejsu	276
	Domyślne implementacje	279
	Klasy abstrakcyjne	279
	Walka w świecie NyetHack	282
17	Typy sparametryzowane	287
	Definiowanie typów sparametryzowanych	287
	Funkcje sparametryzowane	289
	Wiele parametrów typów sparametryzowanych	290
	Ograniczenia typów sparametryzowanych	291
	vararg i get	293
	in i out	295
	Dla ciekawskich: Słowo kluczowe reified	299
18	Rozszerzenia	303
	Definiowanie funkcji rozszerzenia	303
	Definiowanie rozszerzenia dla klasy bazowej	304
	Sparametryzowane funkcje rozszerzeń	305
	Właściwości rozszerzające	307
	Rozszerzenia dla typów akceptujących wartości puste	308
	Funkcje rozszerzeń — za kulisami	309
	Wyodrębnianie kodu do rozszerzeń	310
	Definiowanie plików rozszerzeń	311
	Zmiana nazwy rozszerzenia	313
	Rozszerzenia w standardowej bibliotece Kotlina	314
	Dla ciekawskich: Literały funkcyjne z odbiorcami	315
	Wyzwanie: Funkcja rozszerzenia toDragonSpeak	316
	Wyzwanie: Funkcja rozszerzenia frame	316
19	Podstawy programowania funkcyjnego	319
	Kategorie funkcji	319
	Przekształcenia	319
	Filtry	321
	Złączenia	323
	Dlaczego programowanie funkcyjne?	324
	Sekwencje	325
	Dla ciekawskich: Profilowanie	326
	Dla ciekawskich: Arrow.kt	327
	Wyzwanie: Odwracanie wartości w mapie	328
	Wyzwanie: Zastosowanie programowania funkcyjnego w pliku Tavern.kt	328
	Wyzwanie: Ruchome okno	329
20	Współdziałanie z Javą	331
	Współdziałanie z klasami Javy	331
	Współdziałanie a stosowanie wartości pustych	332
	Odwzorowywanie typów	335

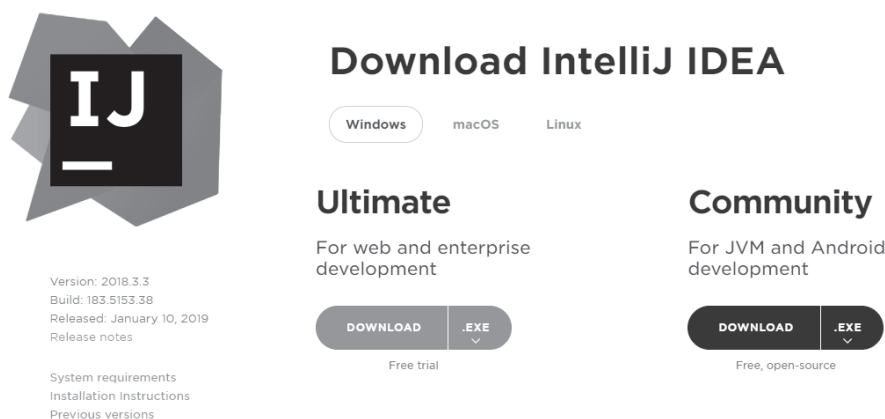
Aksesory get, set i współdziałanie	337
Więcej niż tylko klasy	339
Wyjątki a współdziałanie	347
Typy funkcyjne w Javie	350
21 Tworzenie w Kotlinie pierwszej aplikacji na Androida	353
Android Studio	353
Konfiguracja Gradle	356
Organizacja projektu	359
Definiowanie interfejsu użytkownika	359
Uruchamianie aplikacji w emulatorze	362
Generowanie postaci	363
Klasa aktywności	365
Podłączanie widoków	366
Syntetyczne właściwości rozszerzeń Kotlin dla Androida	368
Tworzenie obiektu nasłuchującego kliknięć	370
Zapisany stan instancji	371
Odczyt zapisanego stanu instancji	374
Refaktoryzacja do postaci rozszerzeń	374
Dla ciekawskich: Biblioteki Kotlin KTX i Anko	377
22 Wprowadzenie do Kotlin Coroutines	379
Parsowanie danych postaci	379
Pobieranie rzeczywistych danych	381
Główny wątek aplikacji na Androida	384
Włączanie koprocedur	385
Tworzenie koprocedur przy użyciu funkcji async	385
Funkcje launch a async i await	387
Funkcje zawieszające	387
Wyzwanie: Rzeczywiste dane	388
Wyzwanie: Minimalna siła	388
23 Poślowie	389
Co dalej?	389
Bezwstydna reklama	389
Dziękujemy!	390
A Więcej wyzwań	393
Korzystanie z Exercism	393
Skorowidz	401

Pierwsza aplikacja w Kotlinie

W tym rozdziale Czytelnik napisze swoją pierwszą aplikację w języku Kotlin, używając przy tym IntelliJ IDEA. Przejście tej programistycznej inicjacji pozwoli poznać to środowisko programistyczne, utworzyć nowy projekt programu w języku Kotlin, napisać pierwszy fragment kodu w tym języku oraz przyrzeć się generowanym przez niego wynikom. Projekt utworzony w tym rozdziale posłuży jako poligon doświadczalny do sprawdzania nowych pojęć prezentowanych w dalszej części książki.

Instalowanie IntelliJ IDEA

IntelliJ IDEA jest zintegrowanym środowiskiem programistycznym (tak zwanym *IDE* od ang.: *Integrated Development Environment*) przeznaczonym do pisania programów w języku Kotlin stworzonym przez firmę JetBrains (która jest także twórcą samego języka Kotlin). W pierwszej kolejności należy zacząć od pobrania IntelliJ IDEA Community Edition z witryny JetBrains: <https://www.jetbrains.com/idea/download> (patrz rysunek 1.1).



Download IntelliJ IDEA

Windows macOS Linux

Ultimate
For web and enterprise development

Community
For JVM and Android development

Version: 2018.3.3
Build: 183.5153.38
Released: January 10, 2019
Release notes

System requirements
Installation Instructions
Previous versions

DOWNLOAD .EXE
Free trial

DOWNLOAD .EXE
Free, open-source

Rysunek 1.1. Pobieranie programu instalacyjnego IntelliJ IDEA Community Edition

Po pobraniu programu należy postępować zgodnie z instrukcjami dotyczącymi używanej platformy systemowej podanymi na stronie poświęconej instalacji i konfiguracji IDE — <https://www.jetbrains.com/help/idea/install-and-set-up-product.html>.

Środowisko IntelliJ IDEA, które dalej będziemy nazywali IntelliJ, ułatwia pisanie odpowiednio sformatowanego kodu w języku Kotlin. Upraszcza także cały proces programistyczny, gdyż udostępnia wbudowane narzędzia do uruchamiania pisanych aplikacji, debugowania ich, sprawdzania oraz refaktoryzacji kodu. Więcej informacji na temat tego, dlaczego do pisania kodu w Kotlinie polecamy właśnie to zintegrowane środowisko programistyczne, można znaleźć pod koniec tego rozdziału w podrozdziale pt. „Dla ciekawskich: Dlaczego warto używać właśnie IntelliJ?”.

Pierwszy projekt programu w Kotlinie

Gratulujemy! A zatem Czytelnik dysponuje już językiem Kotlin oraz potężnym zintegrowanym środowiskiem programistycznym do pisania programów w tym języku. Teraz pozostaje już tylko jedno: nauczyć się płynnie posługiwać tym językiem. Pierwszym zadaniem na tej drodze będzie utworzenie nowego projektu programu pisanego w Kotlinie.

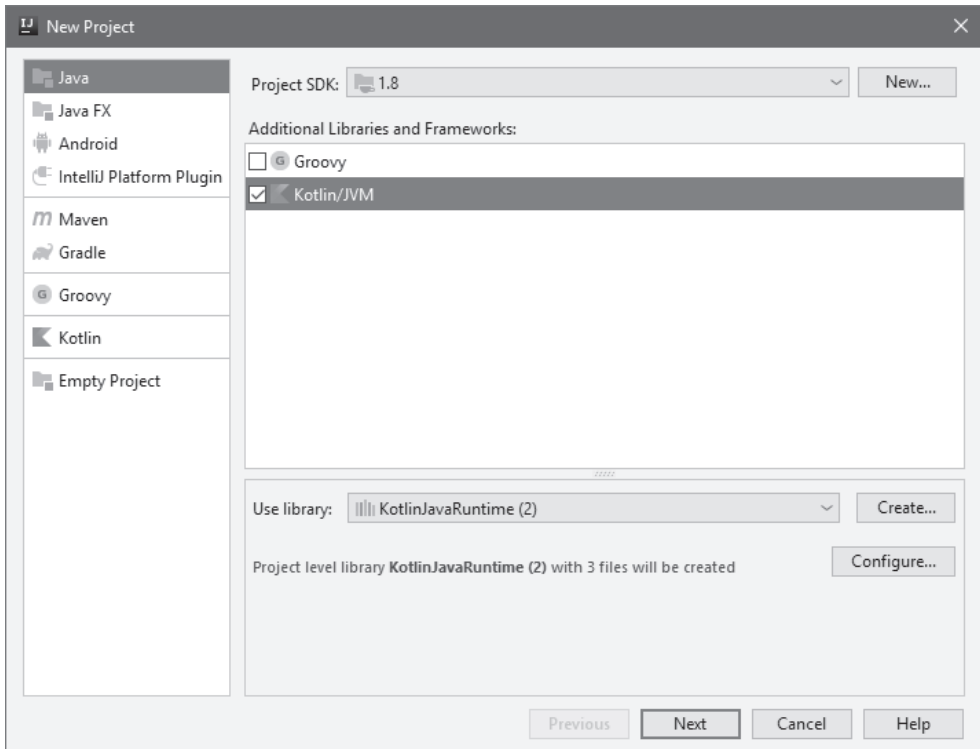
Przede wszystkim należy uruchomić IntelliJ; w efekcie na ekranie zostanie wyświetlone okno dialogowe przedstawione na rysunku 1.2.



Rysunek 1.2. Powitalne okno dialogowe IDE IntelliJ

(Jeśli IDE zostanie uruchomione po raz kolejny, a nie pierwszy raz, to najprawdopodobniej zostanie w nim od razu wyświetlony projekt, nad którym pracowaliśmy wcześniej. Aby ponownie wyświetlić to powitalne okno dialogowe, wystarczy wybrać z menu opcję *File/Close Project*).

Kliknięcie przycisku *Create New Project* spowoduje wyświetlenie okna dialogowego *New Project* przedstawionego na rysunku 1.3.

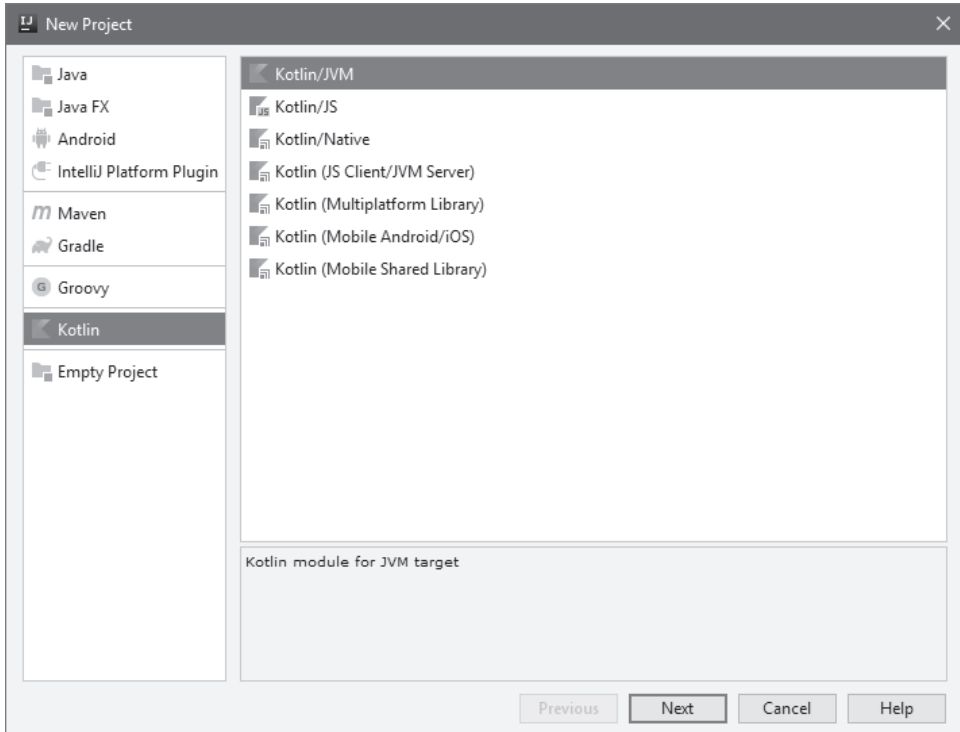


Rysunek 1.3. Okno dialogowe New Project

W tym oknie w jego lewej kolumnie należy zaznaczyć opcję *Kotlin*, a w obszarze po prawej — opcję *Kotlin/JVM* (jak pokazano na rysunku 1.4).

IntelliJ można używać do pisania kodu także w innych językach programowania niż Kotlin, takich jak: Java, Python, Scala oraz Groovy. Wybór opcji *Kotlin/JVM* informuje IDE, że chcemy pisać w języku Kotlin. Konkretnie rzecz biorąc, informuje ona, że chcemy pisać kod *przeznaczony* do uruchamiania na wirtualnej maszynie Javy. Jedną z zalet języka Kotlin jest to, że udostępnia on zestaw narzędzi pozwalających na uruchamianie pisanego w nim kodu w różnych systemach operacyjnych i na różnych platformach.

(Od tego miejsca określeń „wirtualna maszyna Javy” oraz „JVM” będziemy używali zamiennie, gdyż oba stanowią standardowe określenia stosowane w środowisku programistów języka Java. Więcej informacji dotyczących pisania kodu przeznaczonego do uruchamiania na JVM można znaleźć pod koniec tego rozdziału, w podrozdziale pt. „Dla ciekawskich: Pisanie kodu przeznaczonego na JVM”).

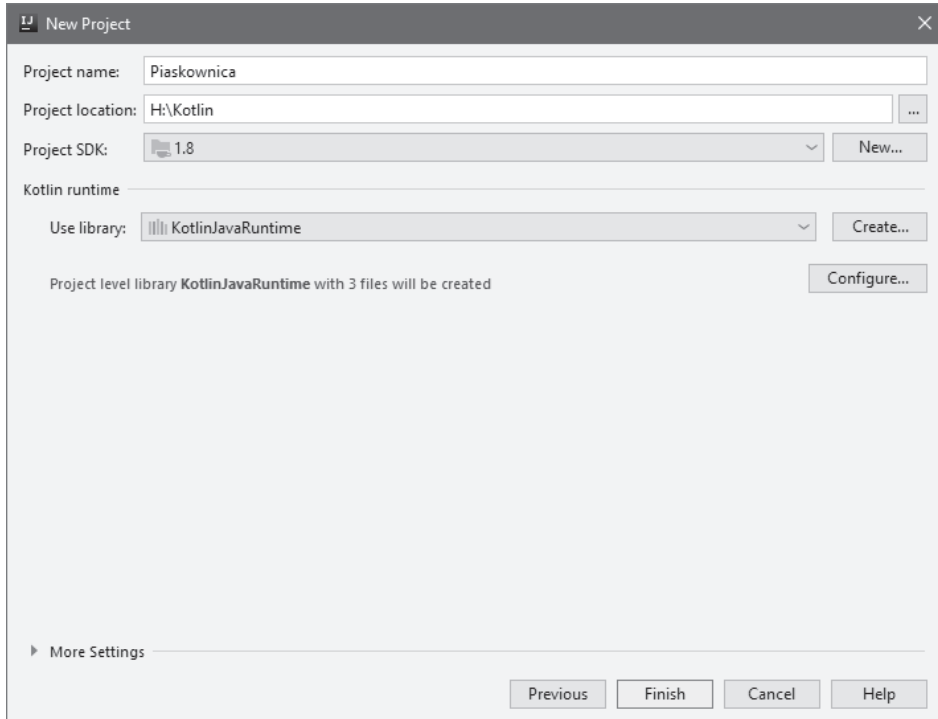


Rysunek 1.4. Tworzenie projektu Kotlin/JVM

Następnie w oknie dialogowym *New Project*, należy kliknąć przycisk *Next*. W efekcie IntelliJ wyświetli kolejne okno dialogowe (patrz rysunek 1.5), w którym można podać ustawienia tworzonego projektu. W polu *Project Name* wpisujemy nazwę projektu: *Piaskownica*. Kolejne pole *Project location* zostanie automatycznie wypełnione przez IDE. Tę domyślnie wybraną lokalizację projektu można pozostawić bez zmian bądź też można wskazać nowe miejsce — w tym celu należy kliknąć przycisk *...* umieszczony z prawej strony pola. Z listy w polu *Project SDK* należy wybrać opcję *1.8, aby zaznaczyć*, że w projekcie ma być używany Java Development Kit (JDK) w wersji 8.

Dlaczego do pisania programów Kotlinie potrzebny jest JDK? Otóż JDK daje IntelliJ dostęp do wirtualnej maszyny Javy oraz narzędzi tego języka, które są konieczne do konwersji kodu w Kotlinie do postaci kodów bajtowych (więcej na ich temat napiszemy w dalszej części rozdziału). Z technicznego punktu widzenia można używać każdej wersji JDK, o ile będzie to wersja 6 lub nowsza. Z naszych doświadczeń wynika jednak, że najlepiej do tego celu używać JDK 8; tak przynajmniej było w czasie pisania niniejszej książki.

Jeśli na liście w polu *Project SKD* nie jest widoczna opcja *1.8*, oznacza to, że ta wersja Javy nie jest zainstalowana na komputerze. W takim przypadku, przed przystąpieniem do dalszej lektury należy zainstalować tę wersję JDK. JDK 8 w wersji dla odpowiedniego systemu operacyjnego można pobrać ze strony: <https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>. Po zainstalowaniu JDK należy ponownie uruchomić IntelliJ i jeszcze raz wykonać wszystkie opisane wcześniej czynności, aby utworzyć nowy projekt.



Rysunek 1.5. Określanie nazwy projektu

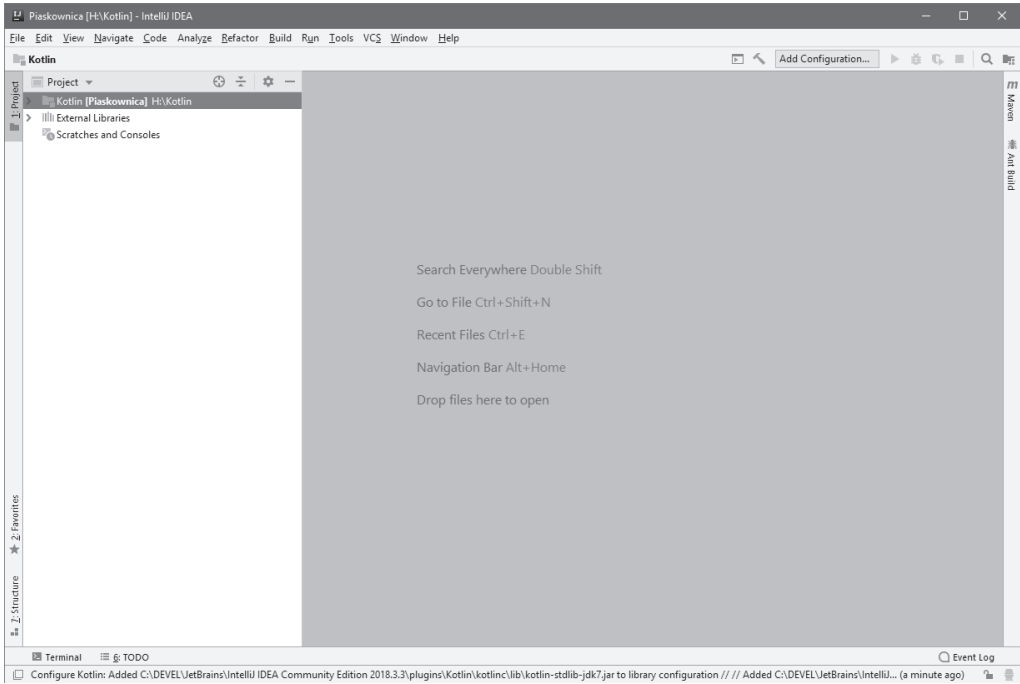
Kiedy opcje będą wyglądały tak jak na rysunku 1.5, można kliknąć przycisk *Finish*.

W efekcie IntelliJ wygeneruje nowy projekt o nazwie Piaskownica i wyświetli go w głównym oknie IDE składającym się z dwóch paneli (patrz rysunek 1.6). Generując nowy projekt, IntelliJ utworzy na dysku jego katalog, a w nim zestaw podkatalogów i plików, przy czym umieści je w katalogu wskazanym w polu *Project location*.

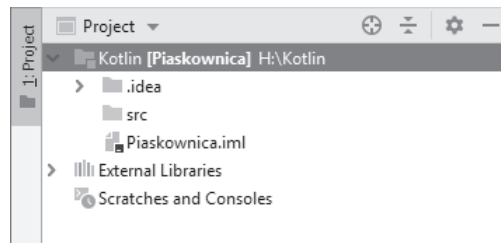
Lewy panel prezentuje *okno narzędzia Project*. Panel z prawej strony początkowo będzie pusty. To właśnie w nim, korzystając z *edytora*, będziemy przeglądać i edytować pliki źródłowe pisane w języku Kotlin. Zwróćmy teraz uwagę na okno narzędzia *Project* widoczne z lewej strony i kliknijmy strzałkę umieszczoną z lewej strony nazwy projektu *Piaskownica*. Spowoduje to wyświetlenie listy katalogów i plików wchodzących w skład projektu, jak pokazaliśmy na rysunku 1.7.

Projekt zawiera wszystkie pliki źródłowe tworzonych programów, jak również informacje o jego konfiguracji i zależnościach. Projekt można także podzielić na jeden lub większą liczbę *modułów*, które można sobie wyobrazić jako mniejsze podprojekty. Domyślnie każdy nowy projekt zawiera tylko jeden moduł i to nam w zupełności wystarczy na potrzeby naszego prostego pierwszego projektu.

Plik *Piaskownica.iml* zawiera informacje konfiguracyjne dotyczące naszego jedyne modułu. Katalog *.idea* zawiera pliki z ustawieniami dotyczącymi całego projektu, jak również z informacjami o naszych iteracjach z samym IDE (na przykład: z informacjami o plikach otworzonych w edytorze). Na razie można pozostawić pliki wygenerowane podczas tworzenia projektu.



Rysunek 1.6. Domyślne okno IntelliJ składające się z dwóch paneli



Rysunek 1.7. Widok projektu

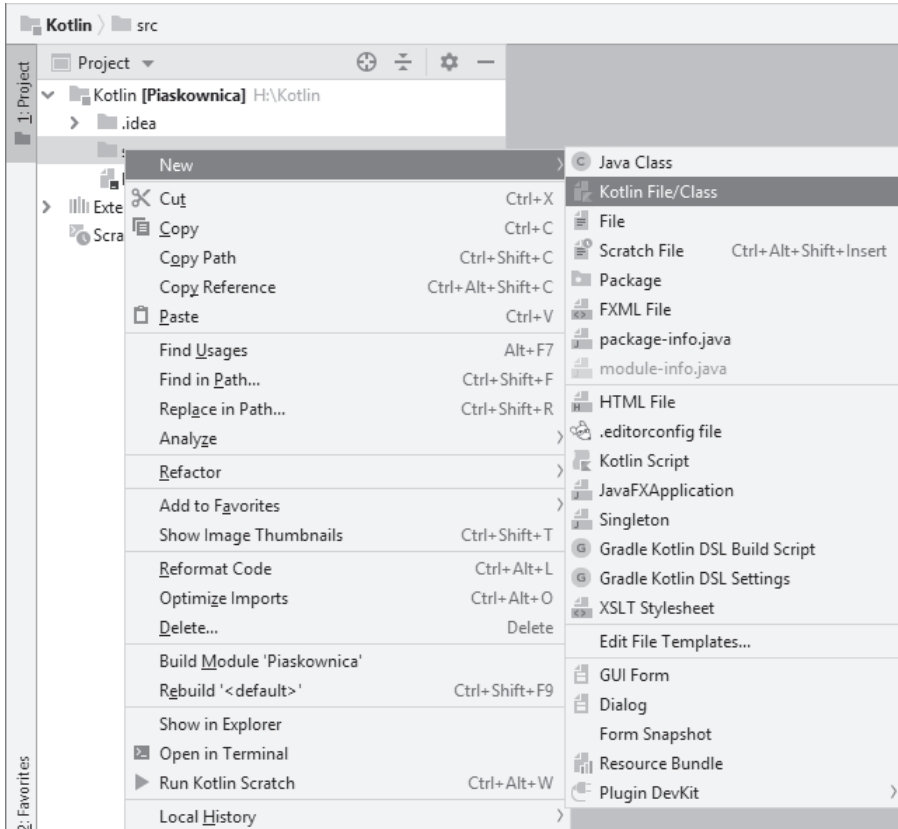
Opcja *External Libraries* zawiera informacje o bibliotekach, od których zależy działanie projektu. Jeśli ją rozwiemy, zauważymy że IntelliJ automatycznie dodało do niej dwie zależności: *1.8* (reprezentującą Java 1.8) oraz *KotlinJavaRuntime*.

(Więcej informacji na temat struktury projektów IntelliJ można znaleźć w dokumentacji na witrynie firmy JetBrains pod linkiem: https://www.jetbrains.org/intellij/sdk/docs/basics/project_structure.html).

Pliki źródłowe tworzone w ramach projektu będą umieszczane w katalogu *src*. A skoro o plikach źródłowych mowa, to możemy utworzyć taki plik i wyświetlić go w edytorze.

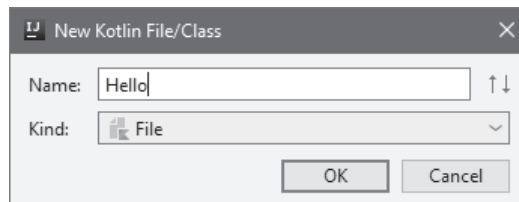
Tworzenie pierwszego pliku źródłowego w Kotlinie

Aby utworzyć plik źródłowy, należy kliknąć prawym przyciskiem myszy katalog `src` wyświetlony w oknie narzędzia *Project*, a następnie z wyświetlonego menu kontekstowego (patrz rysunek 1.8), wybrać opcję *New/Kotlin File/Class*.



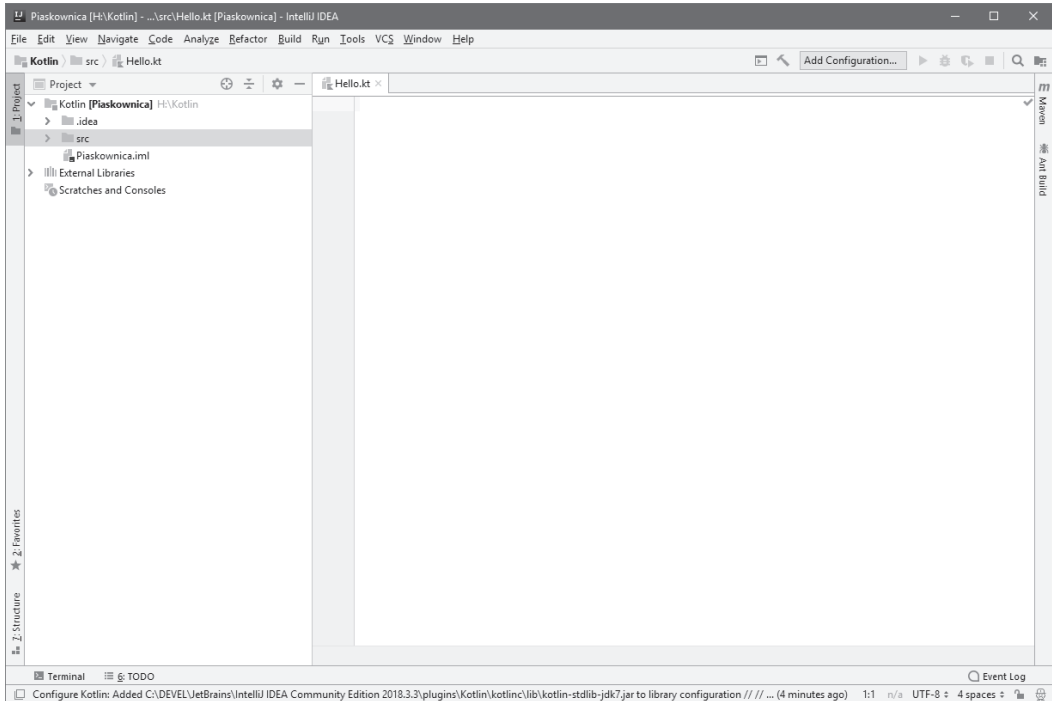
Rysunek 1.8. Tworzenie nowego pliku źródłowego w języku Kotlin

W wyświetlonym oknie dialogowym *New Kotlin File/Class* (patrz rysunek 1.9) w polu *Name* wpiszmy teraz nazwę pliku: `Hello`; opcję *Kotlin* wybraną na liście w polu *Kind* można zostawić bez zmian.



Rysunek 1.9. Określanie nazwy tworzonego pliku

Aby utworzyć plik, wystarczy teraz kliknąć przycisk OK. W efekcie IntelliJ utworzy plik `src\Hello.kt` i wyświetli jego zawartość w edytorze w prawym panelu okna IDE (patrz rysunek 1.10). Rozszerzenie `.kt` informuje, że jest to plik źródłowy z kodem napisanym w języku Kotlin; analogicznie: tak jak pliki źródłowe Javy mają rozszerzenie `.java`, a Pythona — `.py`.



Rysunek 1.10. Pusty plik `Hello.kt` wyświetlony w edytorze

Teraz można już przystąpić do napisania pierwszego fragmentu kodu w Kotlinie. Proszę zatem rozciągnąć palce i pisać w edytorze kod przedstawiony na listingu 1.1. (Warto przy tym zapamiętać, że w dalszej części książki kod który należy wpisać, będzie oznaczany pogrubioną czcionką).

Listing 1.1. „Witaj, świecie!” w Kotlinie (`Hello.kt`)

```
fun main(args: Array<String>) {  
    println("Witaj, świecie!")  
}
```

Ten kod będzie zapewne wyglądał obco. Nie ma się jednak, czym przejmować — pod koniec lektury tej książki czytanie i pisanie kodu w Kotlinie stanie się naszą drugą naturą. Jak na razie w zupełności wystarczy, byśmy zrozumieli ten kod na wysokim poziomie abstrakcji.

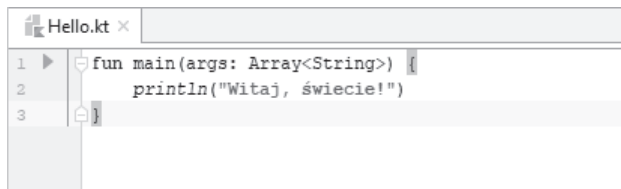
Kod z listingu 1.1 definiuje nową *funkcję*. Funkcja to grupa instrukcji, które kiedyś można wykonać. Wszelkie szczegółowe informacje dotyczące definiowania funkcji oraz sposobu ich działania można znaleźć w rozdziale 4.

Ta konkretna funkcja — `main` — ma w języku Kotlin specjalne znaczenie. Oznacza ona miejsce, od którego rozpocznie się wykonywanie programu. Jest ona określana jako *punkt wejścia aplikacji* (ang. *application entry point*), a w naszym projekcie (jak również w każdym innym) musi być zdefiniowany jeden taki punkt, by program można było uruchomić.

Powyższa funkcja `main` zawiera tylko jedną *instrukcję*: `println("Witaj, świecie!")`. `println()` to także funkcja, jednak tym razem wchodzi ona w skład *standardowej biblioteki Kotliny*. Kiedy program zostanie uruchomiony, a instrukcja `println("Witaj, świecie!")` wykonana, IntelliJ wyświetli wartość podaną w nawiasach na ekranie (w tym przypadku zostanie wyświetlony łańcuch `Witaj, świecie!` bez cudzysłowów).

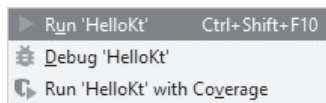
Wykonywanie pliku źródłowego

Krótko po zakończeniu wpisywania kodu z listingu 1.1 IntelliJ wyświetli z lewej strony jego pierwszego wiersza zieloną strzałkę ► nazywaną także „przyciskiem uruchamiania” (ang. *run button*; patrz rysunek 1.11). (Jeśli strzałka nie zostanie wyświetlona lub jeśli pod nazwą pliku na karcie albo pod którymkolwiek wierszem kodu pojawi się czerwona linia, będzie to oznaczało, że w kodzie jest błąd. W takim przypadku należy sprawdzić, czy kod został wpisany dokładnie tak jak na listingu 1.1. Z drugiej strony, jeśli zostanie wyświetlona czerwono-niebieska litera K stanowiąca symbol języka Kotlin, to będzie ona oznaczać to samo co przycisk uruchamiania).



Rysunek 1.11. Przycisk uruchamiania

W końcu nadszedł czas, by powołać program do życia i by powitał on cały świat. Wystarczy w tym celu kliknąć przycisk uruchamiania i z wyświetlonego menu wybrać opcję *Run 'HelloKt'* (patrz rysunek 1.12).

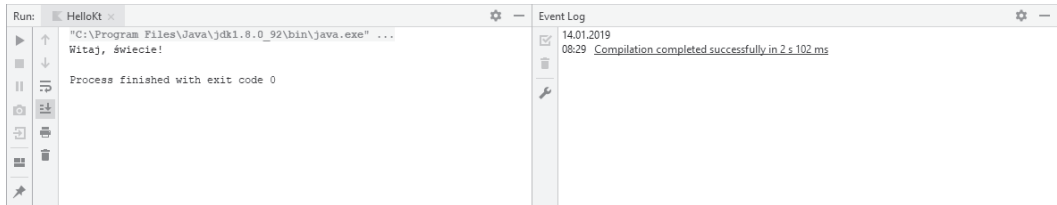


Rysunek 1.12. Wykonywanie pliku Hello.kt

Po uruchomieniu programu IntelliJ wykona wiersz po wierszu kod umieszczony pomiędzy nawiasami klamrowymi (`{}`), a następnie zakończy jego działanie. Jednocześnie u dołu głównego okna IntelliJ zostaną wyświetlone dwa nowe okna narzędzi (pokazane na rysunku 1.13).

Z lewej strony widoczne będzie *okno narzędzi Run* nazywane także *oknem konsoli* (i tak też od tej pory będziemy je nazywać). Wyświetlane są w nim informacje na temat tego, co się stało po uruchomieniu programu przez IntelliJ, jak również wszelkie wyniki generowane przez program. W naszym przypadku w oknie konsoli zostanie wyświetlony komunikat `Witaj, świecie!`. Poniżej niego widoczny będzie kolejny komunikat: `Process finished with exit code 0`¹ oznaczającym prawidłowe wykonanie programu.


¹ Proces zakończony z kodem wyjścia o wartości 0 — *przyp. tłum.*



Rysunek 1.13. Okna narzędzi Run oraz Event log

Ten wiersz pojawia się na samym końcu informacji prezentowanych w oknie konsoli, kiedy podczas wykonywania programu nie pojawiają się żadne błędy; w dalszej części książki, prezentując wyniki generowane w oknie konsoli, będziemy pomijać ten komunikat końcowy.

(Użytkownicy systemu macOS mogą także zobaczyć w oknie konsoli czerwony komunikat błędu informujący o jakichś problemach z JavaLauncherHelper. Nie trzeba jednak zwracać na niego uwagi. Jest to niefortunny efekt uboczny sposobu w jaki Java Runtime Environment — środowisko uruchomieniowe Javy — jest instalowane w systemie macOS. Wyeliminowanie tego problemu wymagałoby sporo pracy, a w zasadzie w niczym o nie szkodzi — komunikat ten można zignorować).

Z prawej strony widoczne będzie natomiast *okno narzędzia Event Log* (nazywane także *dziennikiem zdarzeń*), a w nim informacje o tym, co IntelliJ zrobiło, by przygotować program do wykonania. O dzienniku zdarzeń nie będziemy już wspominać w dalszej części książki, gdyż znacznie więcej interesujących informacji można uzyskać w oknie konsoli. (Z tego względu nie trzeba się także przejmować, jeśli w przyszłości okno dziennika zdarzeń nie będzie już otwierane). Okno to można zamknąć, klikając przycisk  umieszczony w jego prawym górnym rogu.

Kompilacja i wykonywanie kodu w Kotlinie

W tym krótkim czasie pomiędzy naciśnięciem przycisku uruchamiania a wyświetleniem komunikatu `Witaj, świecie!` w oknie konsoli dzieje się bardzo dużo.

W pierwszej kolejności IntelliJ *kompiluje* kod źródłowy napisany w Kotlinie przy użyciu kompilatora `kotlin-jvm`. Oznacza to, że IntelliJ tłumaczy kod źródłowy na *kody bajtowe Javy* — język, którym porozumiewa się JVM. Jeśli kompilator napotka jakiegokolwiek błąd podczas tłumaczenia kodu źródłowego na kody bajtowe, wyświetli stosowny komunikat błędu (lub większą liczbę takich komunikatów), który może nam pomóc w rozwiązaniu problemu. W przeciwnym razie, jeśli proces kompilacji nie napotka żadnych problemów, IntelliJ przejdzie do fazy wykonywania programu.

W fazie wykonywania kody bajtowe wygenerowane wcześniej przez kompilator `kotlin-jvm` zostają wykonane przez wirtualną maszynę Javy. W oknie konsoli wyświetlane są jedynie wyniki generowane przez program, takie jak teksty podane w wywołaniach funkcji `println()`.

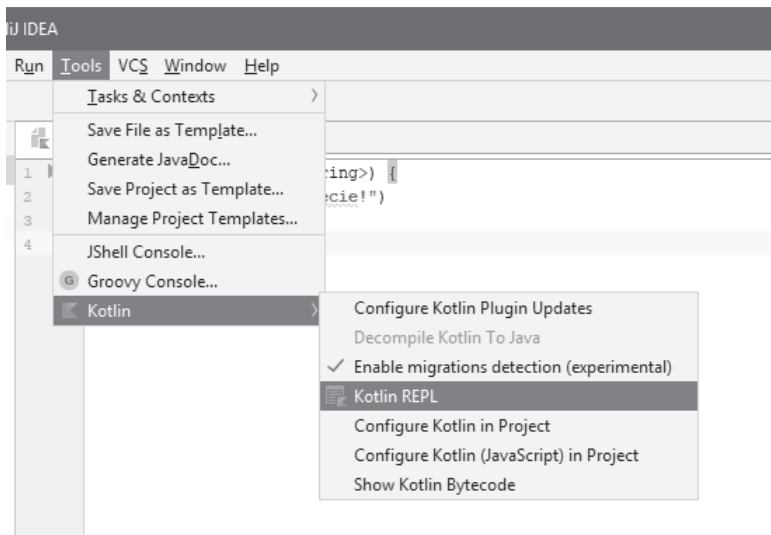
Kiedy nie będzie już żadnych kolejnych kodów bajtowych do wykonania, JVM zakończy działanie. Kod zakończenia zostanie wyświetlony w oknie konsoli, informując w ten sposób o tym, czy program został wykonany prawidłowo, czy też zwrócił jakiś kod błędu.

Do lektury tej książki nie będzie potrzebna dokładna znajomość procesu kompilacji kodu źródłowego pisanego w Kotlinie, niemniej jednak zagadnienia związane z kodami bajtowymi zostały opisane w rozdziale 2.

Kotlin REPL

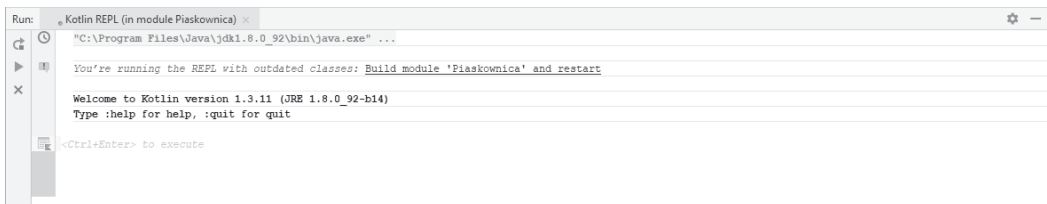
Czasami może się zdarzyć, że będziemy chcieli przetestować niewielkie fragmenty kodu napisanego w Kotlinie i sprawdzić, co się dzieje podczas ich wykonywania — podobnie jak moglibyśmy zapisywać kolejne etapy prostych obliczeń na kawałku papieru. Taka możliwość będzie szczególnie przydatna podczas nauki pisania w języku Kotlin. Na szczęście IntelliJ udostępnia narzędzie do szybkiego sprawdzania kodu bez konieczności zapisywania go w odrębnym pliku. Narzędzie to nosi nazwę *Kotlin REPL*. Już za moment wyjaśnimy znaczenie tej nazwy, na razie otworzmy to narzędzie, by przekonać się, co za jego pomocą można zrobić.

Aby otworzyć narzędzie *Kotlin REPL*, należy wybrać z menu głównego IDE opcję *Tools/Kotlin/Kotlin REPL* (patrz rysunek 1.14).



Rysunek 1.14. Wyświetlenie okna narzędzia Kotlin REPL

Okno narzędzia *Kotlin REPL* zostanie wyświetlone u dołu głównego okna IDE (patrz rysunek 1.15).



Rysunek 1.15. Okno narzędzia Kotlin REPL

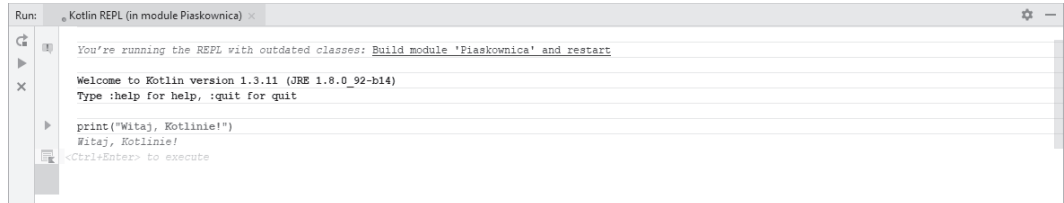
W oknie *Kotlin REPL* można wpisywać kod tak samo jak w edytorze. Różnica polega jednak na tym, że tu wpisany kod może zostać błyskawicznie przetworzony bez konieczności kompilowania całego projektu.

Spróbujmy wpisać w tym oknie poniższy fragment kodu (patrz listing 1.2).

Listing 1.2. „Witaj, Kotlinie!” (REPL)

```
println("Witaj, Kotlinie!")
```

Po wpisaniu tego kodu należy nacisnąć kombinację klawiszy *Ctrl+Enter*, a w efekcie kod zostanie przetworzony i wykonany. Po krótkiej chwili poniżej kodu zostaną także wyświetlone wyniki; w naszym przypadku będzie to tekst *Witaj, Kotlinie!* (patrz rysunek 1.16).



```
Run: Kotlin REPL (in module Piaskownica) x
You're running the REPL with outdated classes: Build module 'Piaskownica' and restart
Welcome to Kotlin version 1.3.11 (JRE 1.8.0_92-b14)
Type :help for help, :quit for quit
println("Witaj, Kotlinie!")
Witaj, Kotlinie!
<Ctrl+Enter> to execute
```

Rysunek 1.16. Przetwarzanie kodu źródłowego

REPL to skrót od angielskich słów: „read, evaluate, print, loop”². W oknie *Kotlin REPL* wpisujemy kod, a następnie przesyłamy go, klikając zieloną strzałkę wyświetloną z prawej strony okna lub naciskając kombinację klawiszy *Ctrl+Enter*. Następnie narzędzie REPL *odczytuje* kod, *przetwarza* go (czyli wykonuje), *wyświetla* wyniki lub efekty uboczne. Po zakończeniu wykonywania kodu narzędzie zwraca sterowanie, a cały proces można *powtórzyć* od nowa.

I w ten sposób rozpoczęliśmy podróż po języku Kotlin! W tym rozdziale udało się nam zrobić całkiem dużo — zbudowaliśmy fundamenty do dalszego poznawania Kotlinia. W następnym rozdziale zajmiemy się szczegółami języka, a konkretnie: zmiennymi, stałymi oraz typami reprezentującymi dane.

Dla ciekawskich: Dlaczego warto używać właśnie IntelliJ?

Kod źródłowy w języku Kotlin można pisać w dowolnym edytorze tekstów. Niemniej jednak zachęcamy do korzystania z IntelliJ, zwłaszcza na etapie uczenia się programowania w Kotlinie. Podobnie jak dobry edytor tekstów wyposażony w narzędzia do sprawdzania ortografii i gramatyki potrafi ułatwić pisanie poprawnej prozy, tak IntelliJ ułatwia pisanie dobrego i właściwie sformatowanego kodu w Kotlinie. Oto kilka rzeczy, w których pomaga IntelliJ:

- pisanie kodu poprawnego syntaktycznie i semantycznie, dzięki takim mechanizmom jak: kolorowanie składni, podpowiedzi kontekstowe oraz automatyczne uzupełnianie kodu;
- uruchamianie i debugowanie kodu dzięki takim mechanizmom jak: punkty wstrzymania oraz krokowe wykonywanie kodu bezpośrednio podczas działania aplikacji;
- modyfikowanie struktury kodu dzięki opcjom refaktoryzacji (takim jak zmienianie nazw lub wyodrębnianie stałych) oraz formatowania kodu (pozwalającym uporządkować wcięcia i odstępy).

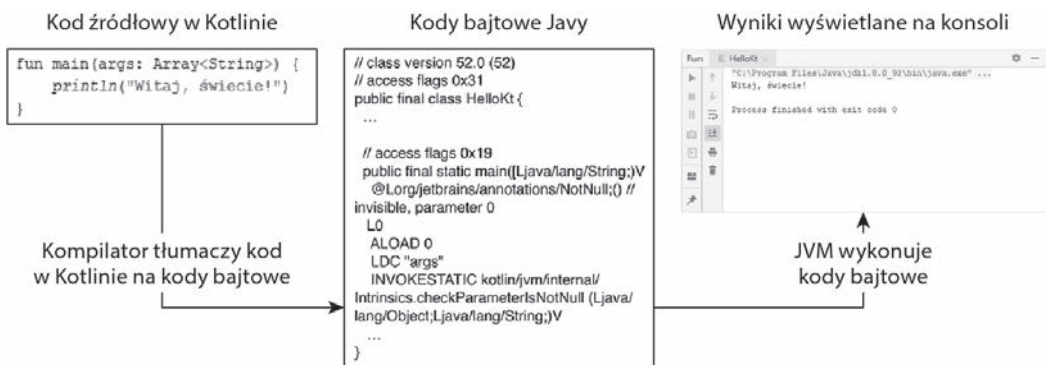
Co więcej, ponieważ Kotlin został opracowany przez firmę JetBrains, integracja tego języka z IntelliJ została uważnie zaprojektowana — co bardzo często można zauważyć i docenić podczas edycji kodu.

² Czytaj, przetwarzaj, wyświetlaj, powtarzaj w pętli — *przyp. tłum.*

Dodatkową zaletą IntelliJ jest fakt, że stanowi podstawę środowiska programistycznego Android Studio, a zatem skróty i narzędzia, których nauczymy się teraz, będzie można później wykorzystać podczas pisania w Android Studio.

Dla ciekawskich: Pisanie kodu przeznaczonego na JVM

JVM to oprogramowanie, które wie, jak wykonywać pewien zestaw instrukcji nazywanych kodami bajtowymi. „Pisanie kodu przeznaczonego na JVM” oznacza kompilowanie oraz tłumaczenie kodu źródłowego napisanego w języku Kotlin na kody bajtowe Javy w celu wykonywania ich na wirtualnej maszynie Javy (patrz rysunek 1.17).



Rysunek 1.17. Proces kompilacji i wykonywania

Każda platforma systemowa taka jak Windows czy macOS ma własny zestaw instrukcji. JVM działa jako swoisty pomost pomiędzy kodami bajtowymi a różnymi środowiskami sprzętowymi i systemowymi, na jakich ona działa, odczytując kody bajtowe i wykonując odpowiednie, odpowiadające im instrukcje używanej platformy. Właśnie z tego względu istnieje wiele wersji JVM przeznaczonych do użycia na różnych platformach systemowych i sprzętowych. To właśnie dzięki temu programiści używający Kotliny mogą pisać kod niezależny od platformy, który można napisać raz, a następnie skompilować do postaci kodów bajtowych i wykonywać na różnych urządzeniach niezależnie od używanego na nich systemu operacyjnego.

Ponieważ kod napisany w Kotlinie można przekształcić na kody bajtowe, które z kolei mogą być wykonywane przez JVM, dlatego uważa się go za język JVM. Prawdopodobnie najbardziej znanym językiem JVM jest oczywiście Java, gdyż to ona powstała jako pierwsza. Niemniej jednak istnieje więcej języków JVM, takich jak Kotlin lub Scala, które zostały opracowane, by uzupełnić pewne niedostatki Javy występujące w niej z programistycznego punktu widzenia.

Możliwości Kotliny nie ograniczają się jednak do wykonywania napisanych w nim programów tylko na JVM. W czasie pisania niniejszej książki można je było kompilować do kodu JavaScript, a nawet do rodzimych kodów konkretnej platformy systemowej, takiej jak Windows czy macOS, eliminując w ten sposób konieczność stosowania warstwy maszyny wirtualnej.

Wyzwanie: Arytmetyka REPL

Na końcu wielu rozdziałów tej książki umieszczonych zostało jedno wyzwanie lub kilka wyzwań. Są one przeznaczone do samodzielnego wykonania w celu pogłębienia wiedzy i nabrania dodatkowego doświadczenia.

Celem tego wyzwania jest poznanie operatorów arytmetycznych stosowanych w Kotlinie: `+`, `-`, `*` oraz `/`. Na przykład w oknie *Kotlin REPL* można wpisać wyrażenie: $(9+12)/2$. Czy wyświetlone wyniki będą zgodne z oczekiwaniami?

Aby zdobyć więcej informacji na temat tego zagadnienia, można przejrzeć informacje na temat funkcji matematycznych dostępnych w standardowej bibliotece języka Kotlin (są one dostępne na stronie: <https://kotlinlang.org/api/latest/jvm/stdlib/kotlin.math/index.html>) i wypróbować jedną z nich, na przykład `min(94, -99)`, która zwraca mniejszą z dwóch liczb podanych w nawiasach.

Skorowidz

A

adnotacja
 @JvmField, 344, 345, 346
 @JvmName, 340
 @JvmOverloads, 340, 342, 343
 @JvmStatic, 347
 @NotNull, 124, 335
 @Nullable, 334
 @Throws, 349, 350
akcesor
 definiowanie, 197
 get, 197, 198, 199, 200, 211, 219, 279, 308, 337, 338, 367
 przesłanie, 197
 set, 197, 198, 199, 200, 211, 219, 308, 337, 338, 367
 tworzenie, 212
Android
 dziennik, 383
 emulator, 362
Android KTX, 377
Android SDK, 353
Android Studio, 353, 357, 383
aplikacja punkt wejścia, *Patrz:*
 punkt wejścia aplikacji
application entry point, *Patrz:*
 punkt wejścia aplikacji
argument, 60, 74, *Patrz też:*
 parametr
 domyślny, 78, 221
 nazwany, 81, 82, 223
ATD, 271, 272

B

backing field, *Patrz:* pole wspierające
biblioteka
 koprocedur, 385
 Kotlina, 27, 314, 315
blok inicjalizatora, 223, 226, 227
błąd, 27
 czasu kompilacji, 108
 informacja, 35
 JavaLauncherHelper, 28

C

Church Alonzo, 96
class property, *Patrz:* właściwość
closure, *Patrz:* domknięcie
combine, *Patrz:* złączenie
companion object, *Patrz:* obiekt uzupełniający
computed property, *Patrz:* właściwość obliczana
control flow, *Patrz:* przepływ sterowania
coroutine dispatcher, *Patrz:* koprocedura dyspozytor
czas kompilacji, 41
 stała, *Patrz:* stała czasu kompilacji

D

dane
 destrukuryzacja,
 Patrz: destrukuryzacja
 typ, *Patrz:* typ
data class, *Patrz:* klasa danych
default argument, *Patrz:* argument domyślny
deserializacja, 374
destrukuryzacja, 128, 129, 189, 261
 lista, 169
domain-specific language,
 Patrz: język dziedziny
domknięcie, 101, 102
dziedziczenie, 235, 242, 275, 303, 304
dziennik
 Logcat, 383
 zdarzeń, 28

E

eager collection, *Patrz:* kolekcja gorliwa
encapsulation, *Patrz:* hermetyzacja
enumerated class, *Patrz:* klasa wycieniowa
exception, *Patrz:* wyjątek
Exercism, 393, 396
extension, *Patrz:* rozszerzenie
extension function, *Patrz:* funkcja rozszerzająca

F

factory function, *Patrz:* funkcja
 wytwórcza

filter, *Patrz:* filtr

filtr, 319, 321

- predykat, 321

framework

- Android, 228
- Exposed, 316
- Ktor, 379

functional programming, *Patrz:*
 programowanie funkcyjne

funkcja, 67

- add, 160, 163, 170, 173
- addAll, 163, 173
- akcesor, *Patrz:* akcesor
- also, 151, 153, 214
- and, 144
- anonimowa, 89, 90, 92, 102, 111, *Patrz też:* lambda
- parametr, 92, 93, 94, 96, 97
- składnia, 90
- tworzenie, 90, 91
- apply, 147, 148, 149, 152, 315, 316
- arrayOf, 177
- assert, 121
- async, 385, 387
- await, 387
- booleanArrayOf, 177
- byteArrayOf, 177
- capitalize, 109, 110, 111
- checkNotNull, 120, 121
- ciało, 69, 72
- clear, 163, 173
- compareTo, 266
- contains, 158, 159, 170, 266
- containsAll, 159, 170
- copy, 261, 262
- count, 89, 90, 93
- displayBalance, 186
- distinct, 176
- doubleArrayOf, 177
- downTo, 60, 64
- elementAt, 170

- equals, 132, 260, 261, 262, 264, 266
- przesłanianie, 270, 271
- error, 121
- exitProcess, 283
- filter, 322, 327
- filtra, *Patrz:* filtr
- findViewById, 366
- flatMap, 320, 322, 327
- floatArrayOf, 177
- fold, 323
- forEach, 133, 164, 165, 171, 188
- forEachIndexed, 165, 168, 171
- format, 141
- generateSequence, 325, 326
- get, 266, 294, 295
- getOrDefault, 184
- getOrNull, 158
- getOrPut, 186
- getValue, 184
- hashCode, 260, 262, 266
- przesłanianie, 270, 271
- inArrayOf, 176
- indexOf, 125, 126
- intArrayOf, 177
- Integer.toBinaryString, 144
- inv, 144
- isNotBlank, 111
- iteratora, 325
- jednowyrażeniowa, 79, 87
- klasowa, 192, 194
- prywatna, 194, 195, 207, 214
- publiczna, 194, 195, 214
- lambda, *Patrz:* lambda
- launch, 386, 387
- let, 110, 111, 148, 149, 152
- listOf, 156, 159, 161
- longArrayOf, 177
- main, 27, 33
- map, 320, 322, 327
- mapOf, 181
- measureNanoTime, 326
- measureTimeInMillis, 326
- mutableListOf, 160, 161
- mutableMapOf, 181

- nagłówek, 69, 70
- nazwa, 71
- konwencja, 71
- w odwrotnych apostrofach, 86
- nazwana, 89
- onCreate, 366
- onEach, 325
- parametr, *Patrz:* parametr
- performPurchase, 186
- plikowa
- w Javie, 83, 84
- plus, 115, 264, 266
- plusAssign, 266
- println, 27, 28, 34, 48, 65, 80
- proficiencyCheck, 119
- prywatna, 71, 77
- przeciążanie, 84, 85
- przekształcająca, 319
- przekształcenia, *Patrz:* przekształcenie
- przesłanianie, 241
- publiczna, 71
- put, 185
- putAll, 185
- rangeTo, 266
- readText, 149, 167
- refaktoryzacja, 75
- referencja, 99, 100
- remove, 160, 173, 186
- removeAll, 173
- removeIf, 163
- replace, 129, 130, 131
- require, 122
- requireNotNull, 121
- roundToInt, 143
- rozszerzenia, 303
- prywatna, 311
- sparametryzowana, 305, 306, 307
- w JVM, 309
- zmiana nazwy, 313
- run, 149, 150, 153, 289, 367
- setExecutable, 149
- setOf, 169
- setReadable, 149

setWriteable, 149
 shl, 144
 shortArrayOf, 177
 sparametryzowana, 289, 290
 split, 127, 128, 155, 257
 standardowa, 147, 148, 149,
 150, 151, 152
 substring, 125, 126, 127
 takeIf, 151, 153
 takeUnless, 152, 153
 to, 182
 toBigDecimal, 139
 TODO, 82, 83, 277
 toDouble, 139
 toDoubleOrNull, 139
 toFloat, 139
 toInt, 139, 142
 toIntArray, 176
 toIntOrNull, 139
 toList, 64, 161, 175
 toLong, 139
 toMap, 323
 toMutableList, 161, 175
 toMutableSet, 175
 toSet, 175
 toString, 260, 262
 trim, 315
 tworzenie, 67, 68, 75, 76, 77
 typu
 Unit, 80, 81
 until, 60, 64
 valueOf, 268
 warunku wstępnego, 120, 121
 widoczność, 70
 with, 150, 153
 wynik, 72, 79
 typ, 72, 80, 81
 wytwórcza, 101
 wywołanie
 zawieszone, 387
 wywoływanie, 74
 bezpieczne, 110
 wyższego rzędu, 101, 319
 xor, 144
 zasięg, 73
 względny, 148
 zip, 323, 324

G

generic type, *Patrz:* typ
 uogólniony
 getter, *Patrz:* akcesor get
 gra
 NetHack, 47
 NyetHack, 47, 52, 67
 gracz, 48, 49, 51
 Gradle, 393
 biblioteka, 359
 konfiguracja, 356, 357, 358,
 359

H

hermetyzacja, 194, 197, 201, 207,
 251, 337

I

IDE, 19
 IntelliJ IDEA, *Patrz:* IntelliJ
 inheritance, *Patrz:* dziedziczenie
 initialization, *Patrz:* instancja
 inicjalizacja
 initializer block, *Patrz:* blok
 inicjalizatora
 instancja, 192, 196, 217
 inicjalizacja, 217, 221, 223
 stan zapisany, 371, 374
 utworzenie, 217
 instantiation, *Patrz:* instancja
 utworzenie
 instrukcja
 if/else, 48, 49, 50, 56, 61, 112
 kolejność warunków, 51
 nawiasy klamrowe, 48, 49,
 57, 58
 zagnieżdżanie, 52
 return, 72, 73, 79, 80, 92
 try/catch, 118, 119, 123
 Integrated Development
 Environment, *Patrz:* IDE
 IntelliJ, 19, 20, 75, 201
 interfejs, 276
 Kotlin, 21

nowy projekt, 23
 panel, 23
 zalety, 30
 IntelliJ IDEA, *Patrz:* IntelliJ
 interfejs, 281
 definiowanie, 275, 276
 implementacja, 276
 domyślna, 279
 interface, 275
 OnClickListener, 371
 użytkownika, 359, 360
 definiowanie, 360
 rejestrowanie elementu, 366
 iteracja, 162, 164
 iterator function, *Patrz:* funkcja
 iteratora

J

Java, 15, 331
 maszyna wirtualna,
 Patrz: JVM
 Java Development Kit, *Patrz:* JDK
 JDK, 22
 język
 DSL, 357, 377
 dziedziny, 316
 Haskell, 327
 Java, *Patrz:* Java
 kompilowany, 108
 Kotlin, *Patrz:* Kotlin
 JVM, 21, 31, 42, 309

K

klasa, 331
 abstrakcyjna, 279, 280, 281
 aktywności, 365
 Any, 244, 247, 260
 AppCompatActivity, 365, 367
 bazowa, 236, 281
 danych, 259, 261, 262
 definiowanie, 191, 192
 instancja, *Patrz:* instancja
 pochodna, 236
 niejawna, 244

klasa
 prywatna, 257
 sparametryzowana, *Patrz:*
 typ sparametryzowany
 TextView, 368
 URL, 382
 właściwość, *Patrz:* właściwość
 wyliczeniowa, 263, 264, 271
 zagnieżdżona, 256, 257
 zapieczętowana, 272

klauzula else, 49, 51

kod
 refaktoryzacja, 201, 203, 205,
 208
 źródłowy, 28

kolekcja, 155, 169, 181, 319
 gorliwa, 325
 konwersja, 176
 leniwa, 325
 łączenie, 323
 tworzenie, 156

komentarz, 54, 108

kompilator, 108, 110

komunikat
 error: val cannot be
 reassigned, 199
 Process finished with exit
 code 0, 27

konstruktor, 192
 argument domyślny, 221, 222
 argument nazwany, 222, 223
 dodatkowy, 220, 221
 podstawowy, 192, 218, 220
 definiowanie właściwości, 219

konsument, 297

kontrawariancja, 299

koprocedura, 379, 387
 dyspozytor, 386
 tworzenie, 385, 386
 włączanie, 385

Kotlin, 15
 współdziałanie z Javą, 331,
 332, 339, 340, 341
 funkcja lambda, 350
 klasa, 331
 odwzorowywanie typów,
 335, 336

przeciążanie funkcji, 342
 typ funkcyjny, 350
 wartość pusta, 332, 334, 335
 zmienna klasowa, 337

Kotlin Coroutines, *Patrz:*
 koprocedura
 Kotlin REPL, 29, 30,
 Patrz też: REPL
 kowariancja, 299

L

lambda, 96, 97, 98, 101, 102, 103,
 147, 148, 149, 151, 289, 350
 jako argument, 97, 98, 99

lazy collection, *Patrz:* kolekcja
 leniwa

lazy initialization, *Patrz:*
 właściwość inicjalizacja leniwa

liczba, 135
 całkowita, 136
 dziesiętna, 138
 reprezentacja bitowa, 143
 utrata precyzji, 142

lista, 155, 266, 287, 298, 325, 326
 destrukuryzacja, *Patrz:*
 destrukuryzacja lista
 element, 157
 dodawanie, 160
 indeks, 157
 modyfikowanie, 159, 162
 usuwanie, 160

iteracja, *Patrz:* iteracja

konwersja na zbiór, 175

niezmienna, 160

wczytywanie danych, 167

zawartość, 158, 159

zmienna, 160

litera K czerwono-niebieska, 27

literał
 funkcyjny, 371
 z odbiorcami, 315
 łańcuchowy, 63, *Patrz też:*
 łańcuch znaków

Ł

łańcuch
 konwersja na typ liczbowy,
 138
 szablon, *Patrz:* szablon
 łańcuchowy
 znaków, 49, 62, 125
 długość, 89
 fragment, 125
 konkatenacja, 49

M

manifest, 381

mapa, 155, 181
 element
 dodawanie, 182, 183, 185
 modyfikowanie, 186, 188
 typ, 182
 usuwanie, 186

klucz, 181, 182, 183, 271
 powtórzony, 182
 typ, 182

par klucz-wartość, 182

tworzenie, 181

maszyna wirtualna Javy,
 Patrz: JVM

metoda
 Intrinsic.checkParameterIsNot
 ↳ Null, 124

modyfikator
 internal, 195
 override, 278
 private, 195, 238, 311
 protected, 195, 238, 239
 public, 195
 widoczności, 70, 71, 194

mutable list, *Patrz:* lista zmienna

mutator, 162, 163

N

notacja infix, 309

not-nullable, *Patrz:* typ
 nieakceptujący wartości null

null, 105, 110, 112, 122, 158, 332
 null coalescing operator, *Patrz:*
 operator łączenia wartości null
 nullable, *Patrz:* typ akceptujący
 wartość null

O

obiekt, 249
 deklaracja, 249, 250, 251, 256
 inicjowanie, 251
 instancja, 250
 nasłuchujący kliknięć, 370
 uzupełniający, 249, 255, 256
 odbiorca, 147, 148, 151
 okno
 Android Virtual Device
 Manager, 362
 dialogowe
 Android SDK, 353
 Extract Function, 68, 71
 Search Everywhere, 304
 wyszukiwania, 42
 konsoli, 27, 28
 narzędzi Run, *Patrz:* okno
 konsoli
 narzędzia Event Log, 28
 operator
 as, 245, 313
 dostępu za pomocą indeksu,
 184, *Patrz też:* znak []
 hierarchia, 55
 inkrementacji, *Patrz:* znak ++
 is, 86, 242, 243
 logiczny, 53, 54
 łączenia wartości null, 113,
 114, 122, 148,
Patrz też: znak ?:
 podwójnego wykrzyknika, 111,
 112, 113, 115, 116,
Patrz też: znak !:
 porównania, 49, 50, 59
 przeciążanie, 264, 266, 294
 przypisania
 z dodawaniem,
Patrz: znak +=

z odejmowaniem,
Patrz: znak -=
 równości
 referencyjnej,
Patrz: znak ===
 strukturalnej,
Patrz: znak ==
 throw, 116
 ustawiania, 161,
Patrz też: znak []=
 wywołania bezpieczny, 110,
 113, 122, *Patrz też:* znak ?.
 overriding, *Patrz:* przesłanianie

P

pakiet
 kotlin.collection, 209
 kotlin.system.exitProcess, 283
 tworzenie, 209
 parametr, 71, 73, 74, 92, 96
 interfejs, 276
 typ, 72
 parametrized type,
Patrz: typ parametru typu
 pętla
 for, 162, 164
 forEach, 328
 while, 173, 174, 175, 251
 plik
 activity_main.xml, 360
 build.gradle, 357
 źródłowy
 tworzenie, 25, 26
 klasa, 281
 pole, 197
 wspierające, 198, 308
 polimorfizm, 241, 243
 połykanie wyjątków, 123
 precondition function, *Patrz:*
 funkcja warunku wstępnego
 primary constructor, *Patrz:*
 konstruktor podstawowy
 producent, 297, 298
 programowanie
 funkcjonalne, 321

funkcyjne, 319, 324, 327, 328
 imperatywne, 324, 327
 obiektowe, 191, 208, 324, 327
 projekt
 Android KTX,
Patrz: Android KTX
 Anko, *Patrz:* Anko
 Exercism, *Patrz:* Exercism
 przekształcenie, 319
 przepływ sterowania, 47
 przesłanianie, 237
 przycisk uruchamiania, 27
 punkt wejścia aplikacji, 27

R

receiver type, *Patrz:* typ odbiorcy
 REPL
 arytmetyka, 32
 zmiennoprzecinkowa, 140
 zakres, 64
 rozszerzenie, 303
 dla klasy bazowej, 304
 typ akceptujący wartość null,
 308
 run button, *Patrz:* przycisk
 uruchamiania

S

saved instance state,
Patrz: instancja stan zapisany
 sealed class, *Patrz:* klasa
 zapieczętowana
 sekwencja, 325, 326
 serializacja, 373
 typ, 170
 setter, *Patrz:* akcesor set
 single-expression function,
Patrz: funkcja
 jednowyrażeniowa
 singleton, 249, 255, 256
 składnia z kropką, 89, 196
 słowo kluczowe
 class, 191, 249, 255, 256
 const, 42

słowo kluczowe

data, 259
 else, 49
 field, 198
 final, 241, 242
 fun, 71, 99
 in, 59, 62, 164, 295, 297, 298, 299
 infix, 309
 inline, 98, 99, 307
 is, 86
 it, 93, 94, 185
 lateinit, 229
 object, 249, 255, 273
 open, 236, 242, 278
 out, 293, 295, 297, 298, 299
 override, 237, 238, 239, 242, 278
 private, 193
 reified, 299, 300
 return, 72, 92
 val, 37, 39, 40, 159, 160, 177, 195, 201, 210, 218, 299
 var, 34, 37, 40, 159, 160, 195, 201, 210
 vararg, 293
 void, 81
 stała, 33
 czasu kompilacji, 41
 zapis, 42
 sterowanie przepływu,
 Patrz: przepływ sterowania
 string template, *Patrz:* szablon łańcuchowy
 subclass, *Patrz:* klasa pochodna
 superclass, *Patrz:* klasa bazowa
 suspended function call, *Patrz:* funkcja wywołanie zawieszona
 synthetic property, *Patrz:* właściwość syntetyczna
 szablon łańcuchowy, 62

Ś

środowisko programistyczne zintegrowane, *Patrz:* IDE

T

test isInitialized, 229
 thread, *Patrz:* wątek
 transform, *Patrz:* przekształcenie
 transformer function, *Patrz:* funkcja przekształcająca
 typ
 akceptujący wartość null, 105, 106, 108, 109, 110
 algebraiczny, *Patrz:* ATD
 Any, 244, 245, 247, 304
 Array, 177
 Arrays, 176
 BigDecimal, 141
 BigInteger, 141
 Boolean, 36, 41
 BooleanArray, 177
 Byte, 41, 135, 136
 ByteArray, 177
 Char, 36, 41
 Deferred, 387
 Double, 36, 41, 135, 138, 140
 formatowanie, 141
 konwersja na Int, 142, 143
 DoubleArray, 177
 Float, 41, 135, 138
 FloatArray, 177
 Function, 91
 funkcyjny, 91, 92, 95, 102
 jako typ wyniku, 100, 101
 Java, 350
 predicate, 93
 Int, 36, 41, 135, 136, 137
 konwersja na Double, 139, 140
 IntArray, 176, 177
 kontrola, 33, 35
 liczbowy, 135
 List, 36, 155, 156, 157, 177, 189, 325
 Long, 41, 135, 136
 LongArray, 177
 Map, 36, 155, 181, 189, 260, 271, 325
 MutableMap, 185

nieakceptujący wartości null, 107, 109, 124
 Nothing, 82, 83
 odbiorcy, 303
 Pair, 182
 parametru typu, 156
 platformy, 334
 podstawowy, 45
 referencyjny, 45
 rzutowanie, 245, 246
 SAM, 371
 Sequence, 325
 Set, 36, 155, 169, 189, 325
 Short, 41, 135, 136
 ShortArray, 177
 sparametryzowany, 81, 156, 287, 288, 298, 299
 definiowanie, 287
 ograniczenia, 291
 parametr, 288, 290
 sprawdzanie, 242, 243
 String, 36, 41, 122, 125, 129, 315, 335
 tablicowy, 176, 177
 Unit, 80, 81, 82
 uogólniony, 81, 156
 usuwanie, 300
 wnioskowanie, 40, 41, 42, 95, 172, 288
 wyliczeniowy, 263,
 Patrz też: klasa wyliczeniowa

U

Unit function, *Patrz:* funkcja typu Unit
 urządzenie zmiana konfiguracji, 371

W

wartość null, *Patrz:* null
 wątek, 384, 387
 zablokowanie, 384

widok, 366
 TextView, 368
 właściwość, 197
 definiowanie w konstruktorze podstawowym, *Patrz:* konstruktor podstawowy definiowanie właściwości
 inicjalizacja, 224, 225, 226, 227
 leniwa, 229, 230
 odraczanie, 228
 modyfikowalna, 195, 210, 211
 obliczana, 200
 rozszerzająca, 307, 308
 syntetyczna, 369
 ustalona, 195, 210
 wartość początkowa, 225, 226, 227
 widoczność, 200, 214
 wyjątek, 116
 android.os.NetworkOnMain
 ↳ ThreadException, 384
 ArrayIndexOutOfBoundsException
 ↳ Exception, 157
 ArrayIndexOutOfBoundsException
 ↳ Exception, 157, 268
 ClassCastException, 246
 IllegalArgumentException, 224, 268
 IllegalStateException, 116, 117, 268
 IOException, 349
 KotlinNullPointerException, 116
 niesprawdzany, 123
 NullPointerException, 106, 107
 obsługa, 118
 sprawdzany, 123
 UninitializedPropertyAccess
 ↳ Exception, 229
 użycia wartości pustej, 111
 z komunikatem, 118, 120, 121
 zgłaszanie, 116
 wyliczenie, *Patrz:* klasa wyliczeniowa

wyrażenie
 break, 175
 lambda, 96, 103
 łańcuchowe, 63
 obiektowe, 249, 255
 warunkowe, 48, 53, 56
 zakres, *Patrz:* zakres
 when, 60, 61, 62, 243
 wyścig, 214

Z

zakres, 59, 62, 64
 wartości malejących, 60
 zbiór, 155, 169
 element
 dodawanie, 171, 172, 173
 usuwanie, 172, 173
 konwersja na listę, 175
 tworzenie, 169, 172
 złączenie, 319, 323
 zmienna, 33
 deklarowanie, 34
 operator logiczny, 55
 klasowa, 337
 lokalna, 73
 plikowa, 73
 przekazywana jako argument, 60, 74
 ustalona, 37, 38, 54
 znak
 \, 127
 !., 54, 55, 111, 112, 113, 115, 116
 !&&, 55
 !||, 55
 !<, 55
 !<=, 55
 !=, 50
 !==, 50, 55
 !>, 55
 !>=, 55
 ", 127
 \$, 63
 \\$, 127
 &&, 54

..., 266
 //, 54
 \?, 126
 ?., 110
 ?:, 113, 114
 [], 266
 [], 157, 184, 265, 294
 []=, 161, 163
 \\, 127
 {}, 48, 63, 72
 ||, 54
 +, 266
 ++, 174
 +=, 163, 173, 185, 266
 <, 49
 <=, 49
 -=, 163, 173, 186
 ==, 49, 132, 264, 266
 ===, 50, 132
 >, 49, 266
 >, 60
 >=, 49
 apostrof, 127
 \b, 127
 Backspace, 127
 cudzysłów, 127
 dolar, 127
 dwukrotnego ukośnika, 127
 in, 266
 litera, *Patrz:* litera
 \n, 127
 nowego wiersza, 127
 powrotu karetki, 127
 \r, 127
 sekwencja specjalna, 127
 strzałka
 zielona, *Patrz:* przycisk uruchamiania
 \t, 127
 tabulacji, 127
 \u, 127
 Unicode, 127, 133

Ż

żądanie sieciowe, 381

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 

Kotlin: najlepsze rozwiązania nie tylko dla Androida!

Gdy w 2017 roku na konferencji Google ogłoszono, że Kotlin jest jednym z oficjalnie wspieranych języków programowania aplikacji dla Androida, język ten błyskawicznie stał się popularny. Najważniejsze firmy technologiczne doceniają jego zalety, takie jak zwarta składnia i integracja z kodem Javy. Co ważne, twórcy Kotlina czerpali z doświadczeń projektantów Javy i zapewnili, że te dwa języki są ze sobą ściśle powiązane. W ten sposób świeżość rozwiązań i nowoczesne mechanizmy zostały znakomicie połączone z najlepszymi cechami Javy, a Kotlin stał się wszechstronnym wieloplatformowym językiem programowania.

Ta książka jest przeznaczona dla programistów, którzy chcą poznać język o możliwościach wykraczających poza Javę i pisać solidne aplikacje dla różnych platform. Pozwala w praktyczny sposób zapoznać się z unikalnymi możliwościami Kotlina i — na podstawie licznych przykładowych projektów — stopniowo zgłębiać złożone zagadnienia. Znakomitym uzupełnieniem prezentowanych treści są informacje o wewnętrznych mechanizmach działania języka wraz z ćwiczeniami do samodzielnego wykonania.

W tej książce między innymi:

- wprowadzenie do Kotlina
- funkcje w Kotlinie, w tym funkcje anonimowe
- praca na obiektach: dziedziczenie, klasy, klasy abstrakcyjne
- programowanie funkcyjne w Kotlinie
- najciekawsze biblioteki i współdzielenie z Javą

Big Nerd Ranch mieści się w Atlancie w USA. Firma została założona w 2001 roku przez Aarona Hillegassa, znakomitego programistę i trenera programowania. Jej misją jest krzewienie wiedzy o rozwijających się technologiach mobilnych. Stygnie ze świetnych szkoleń i znakomitych trenerów.

Josh Skeen szkoli dla Big Nerd Ranch — uczy Javy, Kotlina i pisania aplikacji dla Androida. Programowanie fascynuje go jako forma sztuki interaktywnej. W wolnych chwilach biega, trenuje brazylijskie jiu-jitsu i bawi się synteizatorem modularnym.

David Greenhalgh prowadzi kursy w Big Nerd Ranch. Uwielbia odkrywać nowe możliwości programowania dla Androida. Dla relaksu gotuje, czyta i narzeka na Buffalo Bills.

 helion.pl	<i>Sprawdź nasze szkolenia!</i> SZKOLENIA  AKADEMIA IT & BUSINESS WWW.SZKOLENIA.HELION.PL	KOD KORZYŚCI Sięgnij po więcej! ▶  ISBN 978-83-283-5536-1  9 788328 355361
INFORMATYKA W NAJLEPSZYM WYDANIU		Cena: 79,00 zł

