

Spis treści

Wprowadzenie

Dla kogo jest przeznaczona ta książka? (xxii)

Wiemy, co sobie myślisz (xxiii)

Wiemy, co sobie myśli Twój mózg (xxiii)

Metapoznanie - myślenie o myśleniu (xxv)

Oto co zrobiliśmy (xxvi)

Przeczytaj to (xxviii)

Zespół recenzentów technicznych (xxx)

Podziękowania (xxxi)

ROZDZIAŁ 1. Zaczynamy. Szybki skok

Witamy w Kotliczynie (2)

Kotlina można używać niemal wszędzie (3)

Co zrobimy w tym rozdziale (4)

Instalowanie IntelliJ IDEA (Community Edition) (7)

Stwórzmy prostą aplikację (8)

Właśnie utworzyłeś swój pierwszy projekt w Kotlinie (11)

Dodaj do projektu nowy plik Kotliny (12)

Anatomia funkcji main (13)

Dodaj funkcję main do pliku App.kt (14)

Jazda próbna (15)

Co możemy nakazać w funkcji main? (16)

W kółko i w kółko, i w kółko... (17)

Przykład pętli (18)

Rozgałęzienia warunkowe (19)

Używanie if do zwracania wartości (20)

Aktualizujemy funkcję main (21)

Stosowanie interaktywnej powłoki Kotlina (23)

W REPL można wpisywać fragmenty mające wiele wierszy kodu (24)

Wymieszane komunikaty (27)

Twój przybornik do Kotlina (30)

ROZDZIAŁ 2. Typy proste i zmienne. Być zmienną

Twój kod potrzebuje zmiennych (32)

Co się dzieje, kiedy zadeklarujesz zmienną? (33)

Zmienna zawiera referencję do obiektu (34)

Typy proste Kotlina (35)

Jak jawnie zadeklarować typ zmiennej? (37)

Używaj wartości dostosowanej do typu zmiennej (38)

Przypisywanie wartości innej zmiennej (39)

Trzeba skonwertować wartość (40)

Co się dzieje podczas konwertowania wartości? (41)

Uważaj na przepełnienie (42)

Zapisywanie wielu wartości w tablicy (45)

Tworzymy aplikację HasłoMator (46)

Dodaj kod do pliku HasłoMator.kt (47)

Kompilator domyśla się typu tablicy na podstawie jej wartości (49)

var oznacza, że zmienna może wskazywać na inną tablicę (50)

val oznacza, że zmienna zawsze będzie wskazywać tę samą tablicę... (51)

Zamotane referencje (54)

Twój przybornik do Kotlina (58)

ROZDZIAŁ 3. Funkcje. Wychodzimy poza main

Napiszmy grę: Kamień, nożyce, papier (60)

Ogólny projekt gry (61)

Wybieranie opcji przez grę (63)

Jak się tworzy funkcje? (64)

Do funkcji można przekazywać więcej informacji (65)

Pobieranie informacji z funkcji (66)

Funkcje, których ciałem jest jedno wyrażenie (67)

Dodajemy funkcję `getGameChoice` do pliku `Game.kt` (68)

Funkcja `getUserChoice` (75)

Jak działają pętle `for`? (76)

Zapytaj gracza, jaką opcję wybiera (78)

Pomieszane komunikaty (79)

Musimy sprawdzić dane wpisane przez gracza (81)

Dodajemy `getUserChoice` do pliku `Game.kt` (83)

Dodajemy funkcję `printResult` do pliku `Game.kt` (87)

Twój przybornik do Kotlin (89)

ROZDZIAŁ 4. Klasy i obiekty. Trochę klasy

Typy obiektowe są definiowane przy użyciu klas (92)

Jak projektować własne klasy? (93)

Zdefiniujmy klasę `Dog` (94)

Jak utworzyć obiekt `Dog`? (95)

Jak można odwoływać się do właściwości i funkcji? (96)

Tworzymy aplikację Piosenki (97)

Cud tworzenia obiektu (98)

Jak są tworzone obiekty? (99)

Za kulisami: Wywoływanie konstruktora (100)

Dokładniejsze poznawanie właściwości (105)

Elastyczna inicjalizacja właściwości (106)

Jak używać bloków inicjalizatora? (107)

Właściwości MUSZĄ zostać zainicjowane (108)

Jak można weryfikować wartości właściwości? (111)

Pisanie własnego akcesora get (112)

Pisanie własnego akcesora set (113)

Kompletny kod projektu Psy (115)

Twój przybornik do Kotlin (120)

ROZDZIAŁ 5. Klasy pochodne i bazowe. Stosowanie dziedziczenia

Dziedziczenie umożliwia unikanie powielania kodu (122)

Co mamy zamiar zrobić? (123)

Projektujemy strukturę dziedziczenia klas zwierząt (124)

Używaj dziedziczenia, by unikać powielania kodu w klasach pochodnych (125)

Co powinny przesłaniać klasy pochodne? (126)

Możemy pogrupować niektóre zwierzęta (127)

Dodajemy klasy Canine i Feline (128)

Sprawdzaj hierarchie klas, używając testu JEST (129)

Test JEST działa w dowolnym miejscu drzewa dziedziczenia (130)

Stworzymy parę kotliczyńskich zwierząt (133)

Deklarujemy klasę bazową oraz jej właściwości i funkcje jako otworzone (134)

Jak klasa pochodna dziedziczy po bazowej? (135)

Jak (i kiedy) przesłaniać właściwości? (136)

Przesłanianie właściwości pozwala na więcej niż tylko określanie wartości domyślnych (137)

Jak przesłaniać funkcje? (138)

Przesłonięte funkcje i właściwości pozostają otworzone... (139)

Dodajemy klasę Hippo do projektu Zwierzęta (140)

Dodajemy klasy Canine i Wolf (143)

Która funkcja zostanie wywołana? (144)

Kiedy wywołujemy funkcję na rzecz zmiennej, wywoływana jest funkcja obiektu (146)

Typów bazowych można używać w parametrach funkcji i jako typu zwracanego wyniku (147)

Zaktualizowany kod pliku Animals.kt (148)

Twój przybornik do Kotlin (153)

ROZDZIAŁ 6. Klasy abstrakcyjne i interfejsy. Poważny polimorfizm

Hierarchia klasy Animal raz jeszcze (156)

Obiektów niektórych klas po prostu nie powinno się tworzyć (157)

Abstrakcyjna czy konkretna? (158)

Klasy abstrakcyjne mogą mieć abstrakcyjne właściwości i funkcje (159)

Klasa Animal ma dwie funkcje abstrakcyjne (160)

Jak zaimplementować klasę abstrakcyjną? (162)

TRZEBA zaimplementować wszystkie abstrakcyjne właściwości i funkcje (163)

Zaktualizujemy kod projektu Zwierzęta (164)

Niezależne klasy mogą mieć wspólne zachowania (169)

Interfejsy pozwalają definiować wspólne zachowania POZA hierarchią klasy bazowej (170)

Zdefiniujmy interfejs Roamable (171)

Jak definiować właściwości w interfejsach? (172)

Zadeklaruj, że klasa implementuje interfejs... (173)

Jak implementować wiele interfejsów? (174)

Jak określić, czy stworzyć klasę, klasę pochodną, klasę abstrakcyjną czy interfejs? (175)

Aktualizujemy projekt Zwierzęta (176)

Interfejsy pozwalają na stosowanie polimorfizmu (181)

Gdzie używać operatora is? (182)

Używaj when do porównywania zmiennej z grupą opcji (183)

Operator is zazwyczaj wykonuje inteligentne rzutowanie (184)

Używaj operatora as w celu jawnego rzutowania (185)

Aktualizujemy projekt Zwierzęta (186)

Twój przybornik do Kotlin (189)

ROZDZIAŁ 7. Klasy danych. Używanie danych

Operator == wywołuje funkcję o nazwie equals (192)

Funkcja equals pochodzi z klasy bazowej Any (193)

Wspólne zachowania zdefiniowane w klasie Any (194)

Możemy chcieć, by funkcja equals sprawdzała, czy obiekty są równoważne (195)

Klasa danych pozwala na tworzenie obiektów danych (196)

Klasy danych przesłaniają odziedziczone zachowania (197)

Kopiowanie obiektów danych przy użyciu funkcji copy (198)

Klasy danych definiują funkcje componentN... (199)

Tworzymy projekt Przepisy (201)

Pomieszane komunikaty (203)

Wygenerowane funkcje używają jedynie właściwości zdefiniowanych w konstruktorze (205)

Inicjalizowanie wielu właściwości może powodować powstawanie kłopotliwego kodu (206)

Jak używać domyślnych wartości konstruktora? (207)

Także funkcje mogą używać wartości domyślnych (210)

Przeciążanie funkcji (211)

Zaktualizujemy projekt Przepisy (212)

Ciąg dalszy kodu... (213)

Twój przybornik do Kotlin (217)

ROZDZIAŁ 8. Wartość null i wyjątki. Cały i zdrow

Jak usuwać ze zmiennych referencje do obiektu? (220)

Referencje usuwamy, używając null (221)

Typów akceptujących null możesz używać wszędzie tam, gdzie typów, które null nie akceptują (222)

Jak utworzyć tablice typu akceptującego null? (223)

Jak odwoływać się do funkcji i właściwości typów akceptujących null? (224)

Dbaj o bezpieczeństwo, używając bezpiecznych wywołań (225)

Bezpieczne wywołania można łączyć w łańcuch (226)

Ciąg dalszy historii... (227)

Bezpiecznych wywołań można używać do przypisywania wartości... (228)

Używaj let, by wykonywać kod, jeśli wartości będą różne od null (231)

Stosowanie let z elementami tablic (232)

Zamiast używać wyrażenia if... (233)

Operator !! celowo zgłasza wyjątek NullPointerException (234)

Tworzymy projekt Wartości Null (235)

Ciąg dalszy kodu... (236)

Wyjątki są zgłaszane w wyjątkowych sytuacjach (239)

Przechwytuj wyjątki, używając try i catch (240)

Użyj finally, by określić czynności, które mają być wykonane zawsze (241)

Wyjątek to obiekt typu Exception (242)

Wyjątki można zgłaszać jawnie (244)

Try i catch to wyrażenia (245)

Twój przybornik do Kotlin (250)

ROZDZIAŁ 9. Kolekcje. Zorganizuj się

Tablice mogą być przydatne... (252)

...są jednak rzeczy, których tablice nie potrafią (253)

Jeśli masz wątpliwości, sięgnij do Biblioteki (254)

List, Set oraz Map (255)

Fantastyczne listy... (256)

Tworzenie listy MutableList... (257)

Możemy usunąć wartość... (258)

Można zmieniać kolejność i operować na większych grupach elementów... (259)

Utwórz projekt Kolekcje (260)

Listy dopuszczają duplikaty (263)

Jak tworzyć zbiory? (264)

Jak zbiory wyszukują duplikaty? (265)

Kody mieszające i równość (266)

Reguły przesłaniania funkcji hashCode i equals (267)

Jak używać klasy MutableSet? (268)

Aktualizujemy projekt Kolekcje (270)

Czas na mapy (276)

Sposoby korzystania z map (277)

Tworzenie map MutableMap (278)

Z mapy MutableMap można usuwać elementy (279)

Mapy Map i MutableMap można kopiować (280)

Kompletny kod projektu Kolekcje (281)

Wymieszane komunikaty (285)

Twój przybornik do Kotlin (287)

ROZDZIAŁ 10. Typy sparametryzowane. Odróżniaj wariację od kontrawariacji

Kolekcje używają typów sparametryzowanych (290)

Jak jest zdefiniowany typ MutableList? (291)

Stosowanie parametrów typów w MutableList (292)

Co można robić ze sparametryzowanymi klasami i interfejsami? (293)

Oto co mamy zamiar zrobić (294)

Tworzymy hierarchię klasy Pet (295)

Definiujemy klasę Contest (296)

Dodajemy właściwość scores (297)

Tworzymy funkcję getWinners (298)

Tworzymy kilka obiektów Contest (299)

Tworzymy projekt TypySparametryzowane (301)

Hierarchia typu Retailer (305)

Definiujemy interfejs Retailer (306)

Możemy tworzyć obiekty CatRetailer, DogRetailer i FishRetailer... (307)

Użyj out, by typ sparametryzowany był kowariantny (308)

Aktualizujemy projekt TypySparametryzowane (309)

Potrzebujemy klasy Vet (313)

Tworzenie obiektów Vet (314)

Użyj in, by typ sparametryzowany był kontrawariantny (315)

Typ sparametryzowany może być lokalnie kontrawariantny (316)

Aktualizujemy projekt TypySparametryzowane (317)

Twój przybornik do Kotlina (324)

ROZDZIAŁ 11. Lambdy i funkcje wyższego rzędu. Kod używany jak dane

Prezentacja lambd (326)

Jak wygląda kod lambdy? (327)

Lambdy można zapisywać w zmiennych (328)

Wyrażenia lambda mają typ (331)

Kompilator może wywnioskować typ parametrów lambdy (332)

Używaj lambdy dostosowanej do typu zmiennej (333)

Tworzenie projektu Lambdy (334)

Możesz przekazać lambdę do funkcji (339)

Wywołanie lambdy w ciele funkcji (340)

Co się dzieje podczas wywołania funkcji? (341)

Lambdę można przenieść poza nawiasy ()... (343)

Aktualizujemy projekt Lambdy (344)

Funkcje mogą zwracać lambdy (347)

Piszemy funkcję, która pobiera i zwraca lambdy (348)

Jak można używać funkcji combine? (349)

Użyj typealias, by nadać inną nazwę istniejącemu typowi danych (353)

Aktualizujemy kod projektu (354)

Twój przybornik do Kotlina (361)

ROZDZIAŁ 12. Wbudowane funkcje wyższego rzędu. Wzmocnij swój kod

Kotlin udostępnia sporo funkcji wyższego rzędu (364)

Funkcje min i max operują na prostych typach danych (365)

Bliższe spojrzenie na parametry funkcji minBy i maxBy (366)

Funkcje sumBy i sumByDouble (367)

Tworzymy projekt Spożywczak (368)

Poznaj funkcję filter (371)

Używaj funkcji map, by przekształcać kolekcje (372)

Co się dzieje podczas wykonywania kodu? (373)

Ciąg dalszy historii... (374)

Funkcja forEach działa jak pętla (375)

Funkcja forEach nie zwraca wyniku (376)

Aktualizujemy projekt Spożywczak (377)

Użyj groupBy, by podzielić kolekcję na grupy (381)

Funkcji groupBy można używać w łańcuchach wywołań (382)

Jak używać funkcji fold (383)

Za kulisami: funkcja fold (384)

Więcej przykładów użycia funkcji fold (386)

Aktualizujemy projekt Spożywczak (387)

Pomieszane komunikaty (391)

Twój przybornik do Kotlina (394)

Wyjeżdżamy... (395)

Dodatek A. Koprocedury. Współbieżne wykonywanie kodu

Dodatek B. Testowanie. Pociągnij swój kod do odpowiedzialności

Dodatek C. Pozostałości. Dziesięć najważniejszych rzeczy (których nie opisaliśmy)

1. Pakiety i importowanie (416)

2. Modyfikatory widoczności (418)
3. Klasy wyliczeniowe (420)
4. Klasy zapieczętowane (422)
5. Klasy zagnieżdżone i wewnętrzne (424)
6. Deklaracje obiektów i wyrażenia (426)
7. Rozszerzenia (429)
8. Return, break i continue (430)
9. Więcej zabawy z funkcjami (432)
10. Współdziałanie (434)