

# Оглавление

---

1	■ Создание безопасных программ .....	31
2	■ Функциональное программирование на Kotlin: обзор.....	46
3	■ Программирование с функциями .....	81
4	■ Рекурсия, сорекурсия и мемоизация .....	120
5	■ Обработка данных с использованием списков .....	161
6	■ Необязательные данные .....	197
7	■ Обработка ошибок и исключений.....	222
8	■ Дополнительные операции со списками.....	248
9	■ Ленивые вычисления .....	282
10	■ Обработка данных с использованием деревьев.....	323
11	■ Решение задач с использованием усовершенствованных деревьев ...	363
12	■ Функциональный ввод/вывод .....	392
13	■ Общее изменяемое состояние и акторы.....	420
14	■ Решение типичных проблем функциональным способом.....	448

# Содержание

---

Оглавление .....	5
Предисловие .....	15
Благодарности .....	19
О книге .....	20
Об авторе .....	28
Об иллюстрации на обложке .....	29
<b>1 Создание безопасных программ .....</b>	<b>31</b>
1.1 Программные ловушки .....	33
1.1.1 Безопасная обработка эффектов .....	35
1.1.2 Увеличение безопасности программ за счет ссылочной прозрачности .....	36
1.2 Выгоды безопасного программирования .....	37
1.2.1 Использование подстановочной модели в рассуждениях о программе .....	39
1.2.2 Применение принципов соблюдения безопасности на простом примере .....	40
1.2.3 Максимальное использование абстракций .....	44
Итоги .....	45
<b>2 Функциональное программирование на Kotlin:   обзор .....</b>	<b>46</b>
2.1 Поля и переменные в Kotlin .....	47
2.1.1 Тип можно опустить для простоты .....	47
2.1.2 Изменяемые поля .....	47
2.1.3 Отложенная инициализация .....	48
2.2 Классы и интерфейсы в Kotlin .....	49
2.2.1 Еще большее сокращение кода .....	51
2.2.2 Реализация интерфейса или расширение класса .....	51

2.2.3	Создание экземпляра класса .....	52
2.2.4	Перегрузка конструкторов.....	52
2.2.5	Создание методов <i>equals</i> и <i>hashCode</i> .....	53
2.2.6	Деструктуризация объектов данных.....	55
2.2.7	Реализация статических членов в <i>Kotlin</i> .....	55
2.2.8	Синглтоны .....	56
2.2.9	Предотвращение создания экземпляров служебных классов.....	56
2.3	В <i>Kotlin</i> нет элементарных типов .....	57
2.4	Два типа коллекций в <i>Kotlin</i> .....	57
2.5	Пакеты в <i>Kotlin</i> .....	59
2.6	Видимость в <i>Kotlin</i> .....	60
2.7	Функции в <i>Kotlin</i> .....	61
2.7.1	Объявление функций.....	61
2.7.2	Локальные функции .....	62
2.7.3	Переопределение функций.....	63
2.7.4	Функции-расширения .....	63
2.7.5	Лямбда-выражения .....	64
2.8	Пустое значение <i>null</i> в <i>Kotlin</i> .....	66
2.8.1	Приемы работы с типами, поддерживающими <i>null</i> .....	66
2.8.2	Оператор Элвис и значение по умолчанию.....	67
2.9	Поток выполнения программы и управляющие структуры.....	67
2.9.1	Условная конструкция .....	68
2.9.2	Использование конструкций с несколькими условиями.....	69
2.9.3	Циклы .....	70
2.9.4	Какие проблемы может вызывать поддержка вариантности?.....	76
2.9.5	Когда использовать объявления ковариантности и контрвариантности .....	77
2.9.6	Объявление вариантности в точке определения и точке использования .....	78
	Итоги .....	79

<b>3</b>	<b>Программирование с функциями .....</b>	<b>81</b>
3.1	Что такое функция? .....	82
3.1.1	Отношения между двумя областями функций .....	83
3.1.2	Обратные функции в <i>Kotlin</i> .....	84
3.1.3	Частичные функции .....	85
3.1.4	Композиция функций.....	86
3.1.5	Функции нескольких аргументов .....	86
3.1.6	Каррирование функций .....	87
3.1.7	Частично-примененные функции .....	87
3.1.8	Функции не имеют эффектов.....	88
3.2	Функции в <i>Kotlin</i> .....	89

3.2.1	Функции как данные .....	89
3.2.2	Данные как функции.....	89
3.2.3	Конструкторы объектов как функции .....	89
3.2.4	Использование функций <code>fun</code> в <code>Kotlin</code> .....	90
3.2.5	Объектная и функциональная нотация .....	93
3.2.6	Использование функций как значений .....	94
3.2.7	Ссылки на функции .....	96
3.2.8	Композиция функций <code>fun</code> .....	97
3.2.9	Повторное использование функций .....	98
3.3	<b>Дополнительные особенности функций</b> .....	99
3.3.1	Функции с несколькими аргументами? .....	99
3.3.2	Применение каррированных функций.....	100
3.3.3	Реализация функций высшего порядка.....	100
3.3.4	Полиморфные функции высшего порядка .....	102
3.3.5	Анонимные функции.....	104
3.3.6	Локальные функции .....	106
3.3.7	Замыкания.....	106
3.3.8	Частичное применение функций и автоматическое каррирование .....	107
3.3.9	Перестановка аргументов частично примененных функций .....	112
	<b>Итоги</b> .....	119

<b>4</b>	<b>Рекурсия, сорекурсия и мемоизация</b> .....	120
4.1	<b>Рекурсия и сорекурсия</b> .....	121
4.1.1	Реализация сорекурсии.....	121
4.1.2	Реализация рекурсии .....	123
4.1.3	Различия между рекурсивными и сорекурсивными функциями .....	124
4.1.4	Выбор между рекурсией и сорекурсией.....	125
4.2	<b>Удаление хвостового вызова</b> .....	127
4.2.1	Использование механизма удаления хвостового вызова .....	128
4.2.2	Преобразование циклов в хвостовую рекурсию .....	128
4.2.3	Рекурсивные функции-значения .....	132
4.3	<b>Рекурсивные функции и списки</b> .....	135
4.3.1	Дважды рекурсивные функции .....	137
4.3.2	Абстракция рекурсии в списках .....	140
4.3.3	Обращение списка .....	143
4.3.4	Сорекурсивные списки .....	145
4.3.5	Опасность строгости.....	149
4.4	<b>Мемоизация</b> .....	149
4.4.1	Мемоизация в программировании на основе циклов .....	149
4.4.2	Мемоизация рекурсивных функций .....	150
4.4.3	Неявная мемоизация .....	152
4.4.4	Автоматическая мемоизация .....	154
4.4.5	Мемоизация функций нескольких аргументов .....	157

4.5	Являются ли мемоизованные функции чистыми? .....	159
	Итоги .....	159
<b>5</b>	<b>Обработка данных с использованием списков</b> .....	<b>161</b>
5.1	Классификация коллекций данных .....	162
5.2	Разные типы списков .....	162
5.3	Производительность списков .....	164
5.3.1	Время в обмен на объем памяти и сложность .....	165
5.3.2	Отказ от изменения на месте .....	165
5.4	Какие виды списков доступны в Kotlin? .....	167
5.4.1	Использование постоянных структур данных .....	168
5.4.2	Реализация неизменяемого, постоянного, односвязного списка .....	168
5.5	Совместное использование данных в операциях со списками .....	172
5.6	Дополнительные операции со списками .....	174
5.6.1	Преимущества объектной нотации .....	175
5.6.2	Объединение списков .....	178
5.6.3	Удаление элементов с конца списка .....	180
5.6.4	Использование рекурсии для свертки списков с помощью функций высшего порядка .....	181
5.6.5	Вариантность .....	182
5.6.6	Безопасная рекурсивная версия <code>foldRight</code> .....	191
5.6.7	Отображение и фильтрация списков .....	193
	Итоги .....	196
<b>6</b>	<b>Необязательные данные</b> .....	<b>197</b>
6.1	Проблемы с пустым указателем .....	198
6.2	Как пустые ссылки обрабатываются в Kotlin .....	201
6.3	Альтернативы пустым ссылкам .....	202
6.4	Тип <code>Option</code> .....	205
6.4.1	Извлечение значения из <code>Option</code> .....	207
6.4.2	Применение функций к необязательным значениям .....	209
6.4.3	Композиция функций с типом <code>Option</code> .....	210
6.4.4	Примеры использования <code>Option</code> .....	212
6.4.5	Другие способы комбинирования типа <code>Options</code> .....	215
6.4.6	Комбинирование <code>List</code> и <code>Option</code> .....	218
6.4.7	Когда и как использовать тип <code>Option</code> .....	220
	Итоги .....	221
<b>7</b>	<b>Обработка ошибок и исключений</b> .....	<b>222</b>
7.1	Проблемы с отсутствующими данными .....	223
7.2	Тип <code>Either</code> .....	224
7.3	Тип <code>Result</code> .....	227

7.4	Приемы использования типа <code>Result</code> .....	230
7.5	Дополнительные способы использования <code>Result</code> .....	236
7.5.1	<i>Применение предикатов</i> .....	236
7.6	Преобразование ошибок .....	238
7.7	Дополнительные фабричные функции .....	239
7.8	Применение эффектов .....	240
7.9	Дополнительные способы комбинирования с типом <code>Result</code> .....	243
	Итоги .....	247
<b>8</b>	<b><i>Дополнительные операции со списками</i></b> .....	<b>248</b>
8.1	Проблемы функции <code>length</code> .....	249
8.2	Проблема производительности .....	249
8.3	Преимущества мемоизации .....	250
8.3.1	<i>Недостатки мемоизации</i> .....	250
8.3.2	<i>Оценка увеличения производительности</i> .....	252
8.4	Комбинирование <code>List</code> и <code>Result</code> .....	253
8.4.1	<i>Списки, возвращающие <code>Result</code></i> .....	253
8.4.2	<i>Преобразование <code>List&lt;Result&gt;</code> в <code>Result&lt;List&gt;</code></i> .....	255
8.5	Абстракции операций со списками .....	257
8.5.1	<i>Упаковка и распаковка списков</i> .....	258
8.5.2	<i>Доступ к элементам по индексам</i> .....	261
8.5.3	<i>Разбиение списков</i> .....	266
8.5.4	<i>Поиск подсписков</i> .....	270
8.5.5	<i>Разные функции для работы со списками</i> .....	271
8.6	Автоматическое распараллеливание операций со списками .....	276
8.6.1	<i>Не все вычисления могут выполняться параллельно</i> .....	276
8.6.2	<i>Деление списка на подсписки</i> .....	276
8.6.3	<i>Параллельная обработка подсписков</i> .....	278
	Итоги .....	280
<b>9</b>	<b><i>Ленивые вычисления</i></b> .....	<b>282</b>
9.1	Строгий и ленивый подходы .....	283
9.2	Строгие вычисления в <code>Kotlin</code> .....	284
9.3	Ленивые вычисления в <code>Kotlin</code> .....	286
9.4	Реализация ленивых вычислений .....	288
9.4.1	<i>Комбинирование ленивых значений</i> .....	290
9.4.2	<i>Преобразование обычных функций в ленивые</i> .....	294
9.4.3	<i>Отображение ленивых значений</i> .....	296
9.4.4	<i>Комбинирование типов <code>Lazy</code> и <code>List</code></i> .....	298
9.4.5	<i>Обработка исключений</i> .....	299
9.5	Другие способы комбинирования ленивых вычислений .....	302

9.5.1	Ленивое применение эффектов .....	302
9.5.2	Вычисления, невозможные без ленивых значений.....	304
9.5.3	Создание ленивого списка .....	305
9.6	Работа с потоками.....	308
9.6.1	Свертка потоков .....	314
9.6.2	Трассировка вычислений и применение функций.....	317
9.6.3	Использование потоков для решения конкретных задач.....	319
	Итоги.....	322

<b>10</b>	<b>Обработка данных с использованием деревьев .....</b>	<b>323</b>
10.1	Бинарное дерево.....	324
10.2	Сбалансированные и несбалансированные деревья .....	325
10.3	Размер, высота и глубина дерева.....	325
10.4	Пустые деревья и рекурсивное определение.....	326
10.5	Лиственные деревья .....	327
10.6	Упорядоченные бинарные деревья, или бинарные деревья поиска.....	327
10.7	Порядок вставки и структура дерева .....	329
10.8	Рекурсивный и нерекурсивный обход дерева .....	330
10.8.1	Рекурсивный обход дерева .....	330
10.8.2	Нерекурсивный обход дерева .....	332
10.9	Реализация бинарного дерева поиска.....	333
10.9.1	Деревья и вариантность .....	334
10.9.2	Об абстрактных функциях в классе Tree .....	336
10.9.3	Перегрузка операторов.....	336
10.9.4	Рекурсия в деревьях .....	336
10.9.5	Удаление элементов из дерева.....	340
10.9.6	Слияние произвольных деревьев .....	341
10.10	О свертке деревьев.....	347
10.10.1	Свертка с двумя функциями .....	348
10.10.2	Свертка с одной функцией .....	350
10.10.3	Выбор реализации свертки .....	350
10.11	О преобразовании элементов деревьев.....	353
10.12	О балансировке деревьев .....	354
10.12.1	Вращение деревьев .....	354
10.12.2	Алгоритм Дея/Стоута/Уоррен .....	357
10.12.3	Самобалансирующиеся деревья.....	361
	Итоги.....	362

<b>11</b>	<b>Решение задач с использованием усовершенствованных деревьев .....</b>	<b>363</b>
11.1	Улучшение производительности и безопасности деревьев добавлением самобалансировки.....	364
11.1.1	Структура красно-черных деревьев.....	365

11.1.2	Добавление элемента в красно-черное дерево .....	367
11.1.3	Удаление элементов из красно-черного дерева .....	373
11.2	Практические примеры использования красно-черных деревьев: ассоциативные массивы .....	373
11.2.1	Реализация Map .....	373
11.2.2	Расширение ассоциативных массивов .....	376
11.2.3	Использование ключей, не поддерживающих сравнение .....	377
11.3	Реализация функциональной приоритетной очереди .....	380
11.3.1	Протоколы доступа к приоритетной очереди .....	380
11.3.2	Варианты использования приоритетных очередей .....	380
11.3.3	Требования к реализации .....	381
11.3.4	Левосторонняя куча .....	381
11.3.5	Реализация левосторонней кучи .....	382
11.3.6	Реализация интерфейса, характерного для очередей .....	385
11.4	Элементы и сортированные списки .....	386
11.5	Приоритетная очередь для несопоставимых элементов .....	388
	Итоги .....	391

<b>12</b>	<b>Функциональный ввод/вывод .....</b>	<b>392</b>
12.1.	Что означает «эффект внутри контекста»? .....	393
12.1.1	Обработка эффектов .....	394
12.1.2	Реализация эффектов .....	394
12.2	Чтение данных .....	397
12.2.1	Чтение данных с клавиатуры .....	398
12.2.2	Чтение из файла .....	402
12.3	Тестирование программ с вводом .....	404
12.4	Полностью функциональный ввод/вывод .....	405
12.4.1	Как сделать ввод/вывод полностью функциональным .....	405
12.4.2	Реализация чисто функционального ввода/вывода .....	406
12.4.3	Комбинирование ввода/вывода .....	407
12.4.4	Обработка ввода с IO .....	409
12.4.5	Расширение типа IO .....	411
12.4.6	Добавление в IO защиты от переполнения стека .....	414
	Итоги .....	419

<b>13</b>	<b>Общее изменяемое состояние и акторы .....</b>	<b>420</b>
13.1	Модель акторов .....	422
13.1.1	Асинхронный обмен сообщениями .....	422
13.1.2	Параллельное выполнение .....	422
13.1.3	Управление изменением состояния актора .....	423
13.2	Реализация инфраструктуры акторов .....	424
13.2.1	Обзор ограничений .....	425
13.2.2	Интерфейсы инфраструктуры акторов .....	425
13.3	Реализация AbstractActor .....	427



13.4	Включение акторов в работу .....	428
13.4.1	Реализация примера игры в пинг-понг.....	429
13.4.2	Параллельное выполнение вычислений .....	431
13.4.3	Переупорядочение результатов .....	437
13.4.4	Оптимизация производительности .....	440
Итоги	.....	447

<b>14</b>	<b>Решение типичных проблем функциональным способом.....</b>	<b>448</b>
14.1	Утверждения и проверка данных .....	449
14.2	Повторный вызов функций и эффектов .....	453
14.3	Чтение свойств из файла .....	456
14.3.1	Загрузка файла со свойствами .....	457
14.3.2	Чтение свойств как строк .....	458
14.3.3	Вывод более информативных сообщений об ошибках .....	459
14.3.4	Чтение свойств как списков .....	462
14.3.5	Чтение значений перечислений .....	463
14.3.6	Чтение свойств произвольных типов .....	464
14.4	Преобразование императивной программы: чтение файлов XML .....	467
14.4.1	Шаг 1: императивное решение .....	468
14.4.2	Шаг 2: превращаем императивную программу в функциональную .....	470
14.4.3	Шаг 3: делаем программу еще более функциональной .....	473
14.4.4	Шаг 4: исправление проблемы с аргументами одного типа .....	477
14.4.5	Шаг 5: передача функции обработки элемента в параметре .....	478
14.4.6	Шаг 6: обработка ошибок в именах элементов .....	479
14.4.7	Шаг 7: дополнительные улучшения в прежде императивном коде .....	481
Итоги	.....	483

<b>Приложение А. Смешивание кода на Kotlin и Java.....</b>	<b>484</b>
Создание и управление смешанными проектами.....	485
Создание простого проекта в GRADLE .....	485
Импортирование Gradle-проекта в IntelliJ .....	487
Добавление зависимостей в проект .....	488
Создание проектов с несколькими модулями .....	488
Добавление зависимостей в проект с несколькими модулями.....	489
Вызов Java-методов из Kotlin.....	490
Использование примитивов Java .....	490
Использование числовых типов-объектов Java .....	491
Быстрый отказ со значениями null .....	492
Использование строковых типов Kotlin и Java .....	492

Преобразование других типов.....	493
Вызов Java-методов с переменным числом параметров .....	494
Управление поддержкой null в Java .....	494
Доступ к свойствам в JAVA с зарезервированными именами .....	497
Вызов контролируемых исключений .....	497
СAM-интерфейсы .....	498
Вызов Kotlin-функций из Java .....	498
Преобразование свойств Kotlin.....	498
Использование общедоступных полей Kotlin.....	499
Статические поля.....	499
Вызов функций Kotlin из методов Java.....	500
Преобразование типов Kotlin в типы Java .....	503
Типы функций .....	504
Характерные проблемы смешанных проектов на Kotlin/Java.....	504
Итоги .....	505
<b>Приложение В. Тестирование на основе свойств .....</b>	<b>507</b>
Зачем нужно тестирование на основе свойств? .....	508
Интерфейс .....	509
Тест .....	509
Что такое тестирование на основе свойств? .....	510
Абстракция и тестирование на основе свойств.....	511
Зависимости для модульного тестирования на основе свойств .....	513
Разработка тестов на основе свойств .....	514
Создание своих генераторов .....	517
Использование своих генераторов .....	518
Упрощение кода дальнейшим абстрагированием.....	522
Итоги .....	524
Предметный указатель .....	526

# Предисловие

---

Kotlin появился еще в 2011 году, но по-прежнему остается одним из самых новых языков в экосистеме Java. С тех пор появилась версия Kotlin, работающая на виртуальной машине JavaScript, а также версия, выполняющая компиляцию в машинный код. Это делает Kotlin гораздо более универсальным языком, чем Java, хотя между этими версиями есть большие различия, потому что версия для виртуальной машины Java опирается на стандартную библиотеку Java, недоступную в двух других версиях Kotlin. Сотрудники компании JetBrains, где создан язык Kotlin, прикладывают все силы, чтобы вывести все версии на один уровень, но, как вы понимаете, это требует времени.

Версия для JVM (Java Virtual Machine – виртуальной машины Java), безусловно, является наиболее широко используемой, и она получила дополнительный импульс к развитию, когда в Google объявили Kotlin одним из официальных языков разработки приложений для Android. Основная причина такого решения Google состоит в том, что последняя доступная в Android версия Java – это Java 6, тогда как Kotlin предлагает большинство особенностей Java 11 и многое другое. Kotlin также был объявлен официальным языком сценариев сборки в Gradle вместо Groovy, что позволяет использовать один и тот же язык и для сборки программ, и для самих собираемых программ.

В первую очередь язык Kotlin ориентирован на программистов на Java. Вполне возможно, что в будущем они будут изучать Kotlin как основной язык. Но пока основная масса программистов приходит в Kotlin из Java.

У каждого языка свой путь, определяемый некоторыми фундаментальными понятиями. Язык Java создавался на основе нескольких мощных идей. Он должен работать везде, т. е. в любой среде, где доступна JVM. Главный девиз: «Пиши один раз, запускай где угодно». Хотя некоторые могут утверждать обратное, но в целом Java соответствует этому девизу. Более того, теперь можно запускать почти везде не только программы на Java, но и программы на других языках, скомпилированные для JVM. Kotlin – один из таких языков.

Еще одна основополагающая идея Java – никакие нововведения никогда не станут причиной неработоспособности существующего кода. Хотя эта идея соблюдалась не всегда, но все же разработчики языка старались следовать ей. Конечно, это не всегда хорошо. Основным следствием является невозможность внесения в Java многих улучшений, имеющих в других языках, потому что они могут привести к нарушению совместимости. Любая программа, скомпилированная с предыдущей версией Java, должна оставаться работоспособной в более новых версиях без перекомпиляции. Является ли это полезным или нет, это спорный вопрос, но в результате стремление максимально сохранить обратную совместимость постоянно играет против развития Java.

Также предполагалось, что Java сделает программы более безопасными за счет использования контролируемых исключений, что вынуждает программистов принимать эти исключения во внимание. Для многих это оказалось тяжким бременем, ведущим к практике постоянного преобразования контролируемых исключений в неконтролируемые.

Хотя Java является объектно-ориентированным языком, он должен быть настолько же быстрым, как и большинство языков, используемых для вычислительных задач. В свое время разработчики языка решили, что Java только выиграет, если, помимо объектов, представляющих числа и логические значения, в языке будут поддерживаться элементарные числовые типы, позволяющие выполнять вычисления намного быстрее. Из-за этого отсутствует возможность помещать значения примитивных типов в коллекции, такие как списки, множества и ассоциативные массивы. А когда были добавлены потоки данных (streams), разработчики языка решили создать специальные версии для примитивов, но не для всех, а только для наиболее часто используемых. Если вы пользуетесь некоторыми из неподдерживаемых примитивов, значит, вам не повезло.

То же самое произошло с функциями. В Java 8 появилась поддержка обобщенных функций, но обобщение возможно только для объектов. Поэтому для обработки целых, длинных целых, вещественных чисел двойной точности и логических значений были разработаны специализированные функции. (И снова, к сожалению, поддержка добавлена не для всех элементарных типов – для работы с однобайтовыми и короткими целыми, а также для вещественных чисел одинарной точности нет специализированных функций.) Что еще хуже, потребовались дополнительные функции для преобразования из одного элементарного типа в другой или из элементарных типов в объекты и обратно.

Язык Java был разработан более 20 лет назад. С тех пор многое изменилось, но большинство этих изменений нельзя было перенести в Java, потому что это нарушило бы совместимость. Некоторые изменения все же вносились, и совместимость сохранилась, но за счет удобства использования.

Для устранения этих ограничений было создано много новых языков, таких как Groovy, Scala и Clojure. Они в определенной степени совместимы с Java и позволяют применять существующие библиотеки Java,

а программисты на Java могут использовать библиотеки, написанные на этих языках.

Kotlin пошел другим путем. Он гораздо сильнее интегрирован с Java, что позволяет без проблем смешивать исходный код на Kotlin и Java в одном проекте! В отличие от других языков для JVM Kotlin не выглядит сильно отличающимся от Java (хотя разница, конечно же, имеется). Он больше похож на язык, которым должен стать язык Java. Некоторые даже говорят, что Kotlin – это Java, созданный правильно, потому что решает большинство проблем, характерных для Java. (Однако Kotlin пока не предлагает решений, связанных с ограничениями, свойственными JVM.)

Но, что еще более важно, Kotlin разрабатывался так, чтобы быть намного более восприимчивым ко многим новым приемам, появляющимся в функциональном программировании. В Kotlin есть изменяемые и неизменяемые ссылки, но предпочтение отдается неизменяемому. Kotlin также поддерживает большую часть абстракций функционального программирования, которые позволяют избегать управляющих структур (хотя он имеет традиционные управляющие структуры, помогающие упростить переход от традиционных языков).

Еще одна замечательная особенность Kotlin – он уменьшает потребность в шаблонном коде, позволяя сократить его до минимума. На Kotlin можно написать класс с необязательными свойствами (обладающий также функциями `equals`, `hashCode`, `toString` и `copy`) в одной строке кода, тогда как для объявления эквивалентного класса на Java потребуются около тридцати строк (включая методы свойств и перегруженные конструкторы).

Да, есть другие языки программирования, разработанные для преодоления ограничений Java в среде JVM, но Kotlin отличается тем, что прекрасно интегрируется с Java-программами на уровне исходного кода. Вы можете смешивать исходные файлы на Java и Kotlin в проектах и использовать единую цепочку сборки. Это меняет правила игры, особенно в отношении командного программирования, потому что использование Kotlin в среде Java – не более сложная задача, чем использование любой сторонней библиотеки. Это обеспечивает максимально плавный переход с Java на новый язык и позволяет писать программы, которые:

- безопаснее;
- проще в разработке, тестировании и сопровождении;
- более масштабируемые.

Я полагаю, что многие читатели – программисты на Java – рано или поздно осознают необходимость поиска новых решений своих повседневных проблем. Возможно, у вас уже возник вопрос, почему вы должны использовать Kotlin. Нет ли других языков в экосистеме Java, которые позволят легко применять безопасные методы программирования?

Конечно, есть, и одним из самых известных является Scala. Scala – очень хорошая альтернатива Java, но у Kotlin есть нечто большее. Scala может

взаимодействовать с Java на уровне библиотек, т. е. программы на Java могут использовать библиотеки Scala (объекты и функции), а программы на Scala могут использовать библиотеки Java (объекты и методы). Но программы на Scala и Java должны создаваться как отдельные проекты или хотя бы как отдельные модули, тогда как классы Kotlin и Java можно смешивать внутри одного модуля.

Прочитайте эту книгу, чтобы узнать больше о Kotlin.

# Благодарности

---

Я хотел бы поблагодарить всех, кто участвовал в создании этой книги. Прежде всего, большое спасибо моему редактору Марине Майклз (Marina Michaels). Мне было очень приятно работать с Вами. Также спасибо моему научному редактору Александару Драгосавлевичу (Aleksandar Dragosavljević).

Большое спасибо также Джоэлю Котарски (Joel Kotarski), Джошуа Уайту (Joshua White) и Риккардо Терреллу (Riccardo Terrell) – техническим редакторам, Алессандро Кампейсу (Alessandro Campeis) и Бренту Уотсону (Brent Watson) – корректорам, которые помогли мне сделать эту книгу намного лучше. Спасибо всем рецензентам, читателям, участвующим в программе MEAP, и всем, кто прислал свои отзывы и комментарии, спасибо вам! Эта книга не получилась бы такой, какая она есть, без вашей помощи. Также я хотел бы поблагодарить людей, которые нашли время, чтобы просмотреть и прокомментировать книгу: Алексея Слайковско-го (Aleksei Slaikovskii), Алессандро Кампейса (Alessandro Campeis), Энди Кириша (Andy Kirsch), Бенджамина Голдберга (Benjamin Goldberg), Бриджера Хауэлла (Bridger Howell), Конора Редмонда (Conor Redmond), Дилана Макнейми (Dylan McNamee), Эммануэль Медину Лопес (Emmanuel Medina López), Фабио Фальси Родригес (Fabio Falci Rodrigues), Федерико Кирхейса (Federico Kircheis), Герго Михай Надя (Gergő Mihály Nagy), Грегора Раймана (Gregor Raýman), Джейсона Ли (Jason Lee), Жан-Франсуа Морина (Jean-François Morin), Кента Р. Спилнера (Kent R. Spillner), Линн Норттроп (Leanne Northrop), Марка Элстона (Mark Elston), Мэтью Халверсона (Matthew Halverson), Мэтью Проктора (Matthew Proctor), Нуно Алес-сандра (Nuno Alexandre), Рафаэля Вентальо (Raffaella Ventaglio), Рональда Харинга (Ronald Haring), Шило Морриса (Shiloh Morri), Винсента Терона (Vincent Theron) и Уильяма Э. Уилера (William E. Wheeler).

Хочу также поблагодарить сотрудников издательства Manning: Дейр-дре Хиам (Deirdre Hiam), Фрэнсиса Бурана (Frances Buran), Кери Хейлза (Keri Hales), Дэвида Новака (David Novak), Мелоди Долоаб (Melody Dolab) и Николь Берд (Nichole Beard).

# О книге

---

## Кому адресована эта книга

Цель этой книги – не просто помочь выучить язык Kotlin, но также научить писать гораздо более безопасные программы. Это не означает, что Kotlin следует использовать, только если вы решите писать более безопасные программы, и тем более не означает, что писать более безопасные программы можно только на Kotlin. Эта книга представляет примеры, написанные на Kotlin, потому что это один из самых дружелюбных языков для разработки безопасных программ в экосистеме JVM.

В книге рассказывается о методах, разработанных давно и в самых разных окружениях, хотя многие из них своими корнями уходят в функциональное программирование. Но эта книга не о фундаменталистском функциональном программировании, она о прагматичном безопасном программировании.

Все описанные здесь приемы годами использовались в экосистеме Java и доказали свою эффективность в разработке программ с гораздо меньшим количеством ошибок реализации, чем традиционные методы императивного программирования. Эти безопасные приемы можно реализовать на любом языке, и некоторые из них уже многие годы используются в Java, но часто для этого приходится преодолевать ограничения Java.

Эта книга не учит программированию с самого начала. Она писалась для профессиональных программистов, ищущих более простые и безопасные способы разработки безошибочных программ.

## О чем вы узнаете

В этой книге вы познакомитесь с конкретными приемами, которые могут отличаться от уже знакомых вам, если прежде вы программировали на Java. Большинство из них покажутся вам неизвестными или даже противоречащими приемам, которые программисты привыкли считать



оптимальными. Но многие (хотя и не все) оптимальные приемы были выработаны во времена, когда компьютеры имели 640 Кб памяти, 5 Мб дискового пространства и одноядерный процессор. Времена изменились. Сейчас простой смартфон – это настоящий компьютер с 3 Гб или более оперативной памяти, с твердотельным накопителем на 256 Гб и 8-ядерным процессором. Компьютеры тоже имеют много гигабайт оперативной памяти, терабайты дискового пространства и многоядерные процессоры.

В этой книге я расскажу о:

- дальнейшем увеличении абстракции;
- предпочтительном использовании неизменяемых объектов;
- ссылочной прозрачности;
- инкапсуляции общего изменяемого состояния;
- абстрагировании условных и управляющих структур;
- использовании правильных типов данных;
- использовании отложенных вычислений;
- и многом другом.

## **Дальнейшее увеличение абстракции**

Один из наиболее важных методов, с которыми вы познакомитесь, – это дальнейшее увеличение абстракции (традиционные программисты привыкли считать преждевременную абстракцию злом, как и преждевременную оптимизацию). Но дальнейшее увеличение абстракции помогает лучше понять решаемую задачу, что в свою очередь гарантирует правильность решения.

Вы можете спросить, что в действительности подразумевается под дальнейшим увеличением абстракции. Здесь все просто – это умение распознавать типичные шаблоны в различных вычислениях и их абстрагирование, чтобы избежать повторяющегося кода.

## **Неизменяемость**

Суть неизменяемости заключается в использовании только неизменяемых данных. Многим традиционным программистам трудно представить, как можно писать полезные программы, используя только неизменяемые данные. Разве программирование не основано в первую очередь на изменении данных? Если следовать такой логике, тогда бухгалтерский учет – это прежде всего изменение значений в бухгалтерской книге.

Переход от использования изменяемого к использованию неизменяемого учета произошел еще в XV веке, и с тех пор принцип неизменяемости был признан основным элементом безопасности для учета. Этот принцип также применим к программированию, как вы увидите в этой книге.

## **Ссылочная прозрачность**

Ссылочная прозрачность позволяет писать детерминированные программы, т. е. программы, результаты работы которых можно предсказать

и осмыслить. Такие программы всегда дают одинаковые результаты для одних и тех же входных данных. Это не означает, что они всегда дают одинаковые результаты, но различия в результатах зависят только от изменений на входе, а не от внешних условий.

Такие программы не только безопаснее (потому что обладают предсказуемым поведением), но их также проще писать, сопровождать, обновлять и тестировать. А программы, которые легче тестировать, обычно проверяются более полно и, следовательно, получаются более надежными.

## ***Инкапсуляции общего изменяемого состояния***

Неизменяемые данные автоматически оказываются защищены от случайного изменения при совместном использовании, что часто вызывает много проблем при конкурентной и параллельной обработке данных, таких как взаимоблокировка, простои потоков выполнения и устаревание данных. Но отсутствие общего изменяемого состояния превращается в проблему, когда оно действительно необходимо. Это в первую очередь относится к конкурентному и параллельному программированию.

Устранением изменяемого состояния исключается возможность случайного обмена ошибочными данными, благодаря чему программы становятся безопаснее. Но конкурентное и параллельное программирование подразумевает совместное изменение общего состояния. Без этого конкурирующие и параллельные потоки выполнения не смогут сотрудничать друг с другом. Этот конкретный вариант использования изменяемого общего состояния можно абстрагировать и инкапсулировать так, чтобы получить возможность повторного использования без риска, потому что будет иметься одна полностью протестированная универсальная реализация.

В этой книге вы узнаете, как абстрагировать и инкапсулировать общее изменяемое состояние, чтобы реализовать его только один раз и потом использовать везде, где оно необходимо.

## ***Абстрагирование условных и управляющих структур***

Вторым распространенным источником ошибок в программах после общего изменяемого состояния являются управляющие структуры. Традиционные программы состоят из таких управляющих структур, как циклы и проверки условий. С этими структурами так легко ошибиться, что разработчики языка постарались максимально абстрагировать детали. Одним из лучших примеров является цикл `for-each`, который ныне присутствует в большинстве языков (хотя в Java он все еще называется `for`).

Другая распространенная проблема – правильное использование `while` и `do while` (или `repeat until`) и, в частности, выбор места, где должно проверяться условие. Еще одна проблема – конкурентное изменение элементов коллекций во время их обхода в цикле, когда проблема общего изменяемого состояния возникает даже при использовании единственного потока выполнения! Абстрагирование управляющих структур позволяет полностью устранить подобные проблемы.

## Использование правильных типов данных

В традиционном программировании такие универсальные типы, как `int` и `String`, используются для представления величин без учета единиц измерения. Как следствие, очень легко допустить ошибку, сложив мили с галлонами или доллары с минутами. Использование типов значений может полностью устранить проблемы такого рода при очень низких затратах, даже если используемый язык не предлагает истинных типов значений.

## Отложенные вычисления

Большинство распространенных языков называют *строгими*, в том смысле, что аргументы, передаваемые методу или функции, вычисляются заранее, перед их обработкой. На первый взгляд, такое поведение вполне оправданно, хотя часто это не так. Отложенные, или «ленивые», вычисления – это прием, заключающийся в откладывании вычисления элементов до момента, когда они действительно становятся необходимы. Программирование почти целиком основано на отложенных вычислениях.

Например, условие в конструкции `if...else` вычисляется немедленно, перед его проверкой, но выполнение ветвей откладывается, в том смысле, что выполняется только ветвь, соответствующая условию. Эта отложенность скрыта, и программист не контролирует ее. Явное использование отложенных вычислений помогает писать гораздо более эффективные программы.

## Читатели

Эта книга адресована читателям, уже имеющим опыт программирования на Java. Необходимо также некоторое понимание параметризованных (обобщенных) типов. В этой книге широко используются параметризованные функции, или варианты, которые редко применяются в Java (хотя это мощный инструмент). Не пугайтесь, если вы еще не знакомы с этими приемами; я расскажу о них и объясню, зачем они нужны, когда подойдет время.

## Структура книги

Книга предполагает последовательное чтение, потому что каждая следующая глава основана на понятиях, рассматриваемых в предыдущих главах. Я использую слово «чтение», но эта книга предназначена не только для чтения. Очень немногие разделы содержат одну лишь теорию.

Чтобы извлечь максимум пользы из книги, выполняйте все упражнения, встречающиеся в процессе чтения. Каждая глава содержит ряд упражнений с необходимыми инструкциями и подсказками, которые помогут вам прийти к решению. Каждое упражнение сопровождается предлагаемым решением и тестом, который вы можете использовать для проверки своего решения.

Конец ознакомительного фрагмента.

Приобрести книгу можно

в интернет-магазине

«Электронный универс»

[e-Univers.ru](http://e-Univers.ru)