МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Магнитогорский государственный технический университет им. Г.И.
Носова»

УТВЕРЖДАЮ
Директор ИЭиАС
В.Р. Храмшин

04.02.2025 г.

РАБОЧАЯ ПРОГРАММА ДИСЦИПЛИНЫ (МОДУЛЯ)

СОЗДАНИЕ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ ДЛЯ IOT

Направление подготовки (специальность)
11.03.04 Электроника и наноэлектроника

Направленность (профиль/специализация) программы
Интернет вещей в промышленной электронике

Уровень высшего образования - бакалавриат

Форма обучения
очная

Институт/ факультет	Институт энергетики и автоматизированных систем
Кафедра	Электроники и микроэлектроники
Курс	2
Семестр	4

Магнитогорск
2025 год

Рабочая программа составлена на основе ФГОС ВО - бакалавриат по направлению подготовки 11.03.04 Электроника и наноэлектроника (приказ Минобрнауки России от 19.09.2017 г. № 927)

Рабочая программа рассмотрена и одобрена на заседании кафедры Электроники и микроэлектроники
15.01.2025, протокол № 5

Зав. кафедрой _____ Д.Ю. Усатый

Рабочая программа одобрена методической комиссией ИЭиАС
04.02.2025 г. протокол № 3

Председатель _____ В.Р. Храмшин

Рабочая программа составлена:
доцент кафедры ЭиМЭ, к.т.н. _____ Малахов О.С.

Рецензент:
директор сервисного центра ООО «Техноап-Инжиниринг» , к.т.н.
_____ Суспицын Е.С.

Лист актуализации рабочей программы

Рабочая программа пересмотрена, обсуждена и одобрена для реализации в 2027 - 2028 учебном году на заседании кафедры Электроники и микроэлектроники

Протокол от __ _____ 20__ г. № __
Зав. кафедрой _____ Д.Ю. Усатый

Рабочая программа пересмотрена, обсуждена и одобрена для реализации в 2028 - 2029 учебном году на заседании кафедры Электроники и микроэлектроники

Протокол от __ _____ 20__ г. № __
Зав. кафедрой _____ Д.Ю. Усатый

Рабочая программа пересмотрена, обсуждена и одобрена для реализации в 2029 - 2030 учебном году на заседании кафедры Электроники и микроэлектроники

Протокол от __ _____ 20__ г. № __
Зав. кафедрой _____ Д.Ю. Усатый

Рабочая программа пересмотрена, обсуждена и одобрена для реализации в 2030 - 2031 учебном году на заседании кафедры Электроники и микроэлектроники

Протокол от __ _____ 20__ г. № __
Зав. кафедрой _____ Д.Ю. Усатый

1 Цели освоения дисциплины (модуля)

Получение обучающимися знаний и практических навыков в разработке программного обеспечения для встраиваемых систем, функционирующих под управлением операционных систем, основанных на Android, на языке Kotlin.

2 Место дисциплины (модуля) в структуре образовательной программы

Дисциплина Создание мобильных приложений для IoT входит в часть учебного плана формируемую участниками образовательных отношений образовательной программы.

Для изучения дисциплины необходимы знания (умения, владения), сформированные в результате изучения дисциплин/ практик:

Основы Интернет вещей

Информатика и информационные технологии

Дискретная математика

Знания (умения, владения), полученные при изучении данной дисциплины будут необходимы для изучения дисциплин/практик:

Основы программирования (Java Script)

3 Компетенции обучающегося, формируемые в результате освоения дисциплины (модуля) и планируемые результаты обучения

В результате освоения дисциплины (модуля) «Создание мобильных приложений для IoT» обучающийся должен обладать следующими компетенциями:

Код индикатора	Индикатор достижения компетенции
ПК-1 Способен разрабатывать структурные и функциональные схемы электронных систем и комплексов, принципиальных схем устройств с использованием средств компьютерного проектирования, проведением проектных расчетов и технико-экономическим обоснованием принимаемых решений	
ПК-1.1	Разрабатывает эскизный проект, включающий: выбор структурной схемы электронного устройства или системы путем сопоставления различных вариантов и их оценки с точки зрения технических и экономических требований; рассчитывает все необходимые показатели структурной схемы электронного устройства или системы, в том числе показатели качества; выбирает и обосновывает схемы вспомогательных устройств
ПК-1.2	Производит технико-экономическое обоснование принятого решения с расчетами себестоимости устройства и стоимости его эксплуатации; сравнивает с аналогами по технико-экономическим характеристикам

4. Структура, объём и содержание дисциплины (модуля)

Общая трудоемкость дисциплины составляет 3 зачетных единиц 108 акад. часов, в том числе:

– контактная работа – 45,85 акад. часов:

– аудиторная – 45 акад. часов;

– внеаудиторная – 0,85 акад. часов;

– самостоятельная работа – 62,15 акад. часов;

– в форме практической подготовки – 0 акад. час;

Форма аттестации - зачет с оценкой

Раздел/ тема дисциплины	Семестр	Аудиторная контактная работа (в акад. часах)			Самостоятельная работа студента	Вид самостоятельной работы	Форма текущего контроля успеваемости и промежуточной аттестации	Код компетенции
		Лек.	лаб. зан.	практ. зан.				
1. Введение в язык программирования Kotlin								
1.1 Основные сведения о Kotlin	4	1				Чтение литературы	Устный опрос	ПК-1.1
1.2 Среда разработки IntelliJ IDEA		1		2	2	Ознакомление с IntelliJ IDEA	Устный опрос	ПК-1.1
Итого по разделу		2		2	2			
2. Основы языка программирования Kotlin								
2.1 Структура программы	4	2		2	2	Создание первого проекта в IntelliJ IDEA	Устный опрос	ПК-1.1
2.2 Переменные и типы данных		2		2	2	Чтение литературы	Устный опрос	ПК-1.1
2.3 Арифметические операции		2		2	2	Написание программы по заданию	Устный опрос	ПК-1.2
2.4 Условные выражения и логические операции		2		2	2	Написание программы по заданию	Устный опрос	ПК-1.2
2.5 Условные конструкции		2		2	2	Написание программы по заданию	Устный опрос	ПК-1.2
2.6 Циклы, массивы и диапазоны		1		2	2	Написание программы по заданию	Устный опрос	ПК-1.2
Итого по разделу		11		12	12			
3. Функции								
3.1 Функции и их параметры, перегрузка функций, лямбда-выражения	4	2		2	4	Чтение литературы, написание программы по заданию	Устный опрос	ПК-1.2

Итого по разделу		2		2	4			
4. Объектно-ориентированное программирование								
4.1 Классы, объекты, конструкторы, интерфейсы	4			4	6	Написание программы по заданию, чтение литературы по теме	Устный опрос	ПК-1.2
Итого по разделу				4	6			
5. Коллекции и последовательности								
5.1 Изменяемые и неизменяемые коллекции, операции с элементами коллекций	4				10	Изучение материала по теме, написание программы по заданию	Устный опрос	ПК-1.2
Итого по разделу					10			
6. Среда разработки Android Studio								
6.1 Знакомство со средой разработки. Создание первого проекта	4			2	10	Изучение функционала среды разработки	Устный опрос	ПК-1.2
6.2 Структура проекта, система сборки Gradle. Файл AndroidManifest.xml				4	10	Создание проекта, подключение модулей	Устный опрос	ПК-1.2
6.3 Эмуляторы устройств Android и Wear Os				4	8,15	Тестирование эмуляторов	Устный опрос	ПК-1.2
Итого по разделу				10	28,15			
Итого за семестр		15		30	62,15		зао	
Итого по дисциплине		15		30	62,15		зачет с оценкой	

5 Образовательные технологии

Для реализации предусмотренных видов учебной работы в качестве образовательных технологий в преподавании дисциплины используются традиционная и модульно-компетентностная технологии.

Лекции проходят в традиционной форме и в форме лекций-консультаций. На лекциях-консультациях изложение нового материала сопровождается постановкой вопросов и дискуссией в поисках ответов на эти вопросы.

При выполнении практических работ студенты учатся навыками написания программ, рассмотренных на лекционных занятиях. При защите практических работ перед студентами ставятся задачи, требующие логического мышления, принципа обобщения и сопоставления.

Самостоятельная работа стимулирует студентов в процессе подготовки домашних заданий и при подготовке к промежуточной аттестации.

6 Учебно-методическое обеспечение самостоятельной работы обучающихся

Представлено в приложении 1.

7 Оценочные средства для проведения промежуточной аттестации

Представлены в приложении 2.

8 Учебно-методическое и информационное обеспечение дисциплины

а) Основная литература:

1. Петросян, Л. Э. Разработка мобильных приложений на Kotlin : учебное пособие / Л. Э. Петросян, Н. А. Приходько. — Москва : РТУ МИРЭА, 2024. — 101 с. — ISBN 978-5-7339-2215-7. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/421091 (дата обращения: 14.04.2025). — Режим доступа: для авториз. пользователей.

б) Дополнительная литература:

1. Пирская, Л. В. Разработка мобильных приложений в среде Android Studio : учебное пособие / Л. В. Пирская ; Южный федеральный университет. - Ростов-на-Дону ; Таганрог : Издательство Южного федерального университета, 2019. - 123 с. - ISBN 978-5-9275-3346-6. - Текст : электронный. - URL: https://znanium.com/catalog/product/1894469 (дата обращения: 14.04.2025). – Режим

в) Методические указания:

Представлены в приложении 3

г) Программное обеспечение и Интернет-ресурсы:

Программное обеспечение

Наименование ПО	№ договора	Срок действия лицензии
7Zip	свободно распространяемое ПО	бессрочно
Браузер Yandex	свободно распространяемое ПО	бессрочно
Браузер Mozilla Firefox	свободно распространяемое ПО	бессрочно
FAR Manager	свободно распространяемое ПО	бессрочно
MS Windows 10 Pro	К-79-21 от 22.11.2021	бессрочно

Профессиональные базы данных и информационные справочные системы

Название курса	Ссылка
Электронная база периодических изданий East View Information Services, ООО «ИВИС»	URL: https://dlib.eastview.com/

9 Материально-техническое обеспечение дисциплины (модуля)

Материально-техническое обеспечение дисциплины включает:

Материально-техническое обеспечение дисциплины включает:

Учебные аудитории для проведения занятий лекционного типа мультимедийные средства хранения, передачи и представления информации

Учебные аудитории для практической и самостоятельной работы обучающихся с персональными компьютерам с ПО из п. 8(г), выходом в Интернет и с доступом в электронную информационно-образовательную среду университета

Приложение 1

Учебно-методическое обеспечение самостоятельной работы обучающихся

По дисциплине «Создание мобильных приложений для IoT» предусмотрена аудиторная и неаудиторная самостоятельная работа обучающихся.

Аудиторная самостоятельная работа студентов предполагает ответы на вопросы на рактических занятиях при защите работ.

Вопросы к защите работы №2:

1. Что такое переменная?
2. Какие виды переменных в языке *Kotlin* вы знаете? В чем их различия?
3. Какие типы переменных в языке *Kotlin* вы знаете?
4. Можно ли в ходе программы изменить значение ранее инициализированной переменной, объявленной ключевым словом *val*?
5. Можно ли инициализировать переменную типа *UByte* значением -255? Ответ обосновать.
6. Какими типами объявляются переменные, хранящие дробные значения?

Вопросы к защите работы №3:

1. Чем отличается префиксный инкремент от постфиксного? Приведите примеры.
2. Как при делении целых чисел получить результат, содержащий дробную часть?
3. Каков будет результат операции: **val a = 6 and 7**?

Вопросы к защите работы №4:

1. В чем разница выполнения операции *or* над логическими и числовыми операндами?
2. В чем разница между операциями = и ==?
3. Какая операция над логическими операндами обозначается восклицательным знаком?

Вопросы к защите работы №5:

1. Можно ли заменить конструкцию *if* конструкцией *when*?
2. В каких случаях следует использовать *when*, а в каких *if*?
3. Объяснить синтаксис конструкции *if*.
4. Объяснить синтаксис конструкции *when*.

Вопросы к защите работы №6:

1. В каком случае следует использовать цикл *for*, а в каком *while*?
2. Каковы функции служебных слов *break* и *continue*?
3. В чем разница между циклами с предусловием и постусловием?

Вопросы к защите работы №7:

1. Что такое массив?
2. Какие варианты объявления и инициализации массива вы знаете?
3. С какого числа начинается нумерация элементов массива?
4. Как в языке Kotlin можно объявить массив из дробных чисел?

Практические задания

1. Создать и запустить проект, выполнив указанные в теоретической части шаги.
2. В созданном на первом практическом занятии проекте объявить и инициализировать целочисленную неизменяемую переменную. Вывести ее значение в консоль, используя строковый шаблон.

3. Написать программу вывода в консоль результатов приведенных в теоретической части арифметических и поразрядных операций.
4. Написать программу вывода в консоль результатов условных выражений и логических операций, рассмотренных в теоретической части.
5. Повторить приведенные в теоретической части примеры в IntelliJ IDEA.
6. Изменить значения переменных и проверить результаты выполнения программы.
7. Написать программу вывода в консоль квадратов чисел от 1 до 20, используя циклы.
8. Написать программу вывода в консоль таблицы умножения, используя циклы.
9. Написать программу вывода в консоль английского алфавита, используя диапазоны и цикл.

Приложение 2

а) Планируемые результаты обучения и оценочные средства для проведения промежуточной аттестации:

Код индикатора	Индикатор достижения компетенции	Оценочные средства
ПК-1: Способен разрабатывать структурные и функциональные схемы электронных систем и комплексов, принципиальных схем устройств с использованием средств компьютерного проектирования, проведением проектных расчетов и технико-экономическим обоснованием принимаемых решений		
ПК-1.1	Разрабатывает эскизный проект, включающий: выбор структурной схемы электронного устройства или системы путем сопоставления различных вариантов и их оценки с точки зрения технических и экономических требований; рассчитывает все необходимые показатели структурной схемы электронного устройства или системы, в том числе показатели качества; выбирает и обосновывает схемы вспомогательных устройств	Примерные вопросы для получения зачета с оценкой: 7. Что такое переменная? 8. Какие виды переменных в языке *Kotlin* вы знаете? В чем их различия? 9. Какие типы переменных в языке *Kotlin* вы знаете? 10. Можно ли в ходе программы изменить значение ранее инициализированной переменной, объявленной ключевым словом *val*? 11. Можно ли инициализировать переменную типа *UByte* значением -255? Ответ обосновать. 12. Какими типами объявляются переменные, хранящие дробные значения? 13. Чем отличается префиксный инкремент от постфиксного? Приведите примеры. 14. Как при делении целых чисел получить результат, содержащий дробную часть? 15. Каков будет результат операции: **val a = 6 and 7**? 16. В чем разница выполнения операции *or* над логическими и числовыми операндами? 17. В чем разница между операциями = и ==? 18. Какая операция над логическими операндами обозначается восклицательным знаком? 19. Можно ли заменить конструкцию *if* конструкцией *when*? 20. В каких случаях следует использовать *when*, а в каких *if*? 21. Объяснить синтаксис конструкции *if*. 22. Объяснить синтаксис конструкции *when*. 23. В каком случае следует использовать цикл *for*, а в каком *while*? 24. Каковы функции служебных слов *break* и *continue*? 25. В чем разница между циклами с

		предусловием и постусловием? 26. Что такое массив? 27. Какие варианты объявления и инициализации массива вы знаете? 28. С какого числа начинается нумерация элементов массива? 29. Как в языке Kotlin можно объявить массив из дробных чисел?
ПК-1.2	Производит технико-экономическое обоснование принятого решения с расчетами себестоимости устройства и стоимости его эксплуатации; сравнивает с аналогами по технико-экономическим характеристикам	Практические задания для получения зачета: 10. Создать и запустить проект, выполнив указанные в теоретической части шаги. 11. В созданном на первом практическом занятии проекте объявить и инициализировать целочисленную неизменяемую переменную. Вывести ее значение в консоль, используя строковый шаблон. 12. Написать программу вывода в консоль результатов приведенных в теоретической части арифметических и поразрядных операций. 13. Написать программу вывода в консоль результатов условных выражений и логических операций, рассмотренных в теоретической части. 14. Повторить приведенные в теоретической части примеры в IntelliJ IDEA. 15. Изменить значения переменных и проверить результаты выполнения программы. 16. Написать программу вывода в консоль квадратов чисел от 1 до 20, используя циклы. 17. Написать программу вывода в консоль таблицы умножения, используя циклы. 18. Написать программу вывода в консоль английского алфавита, используя диапазоны и цикл.

б) Порядок проведения промежуточной аттестации, показатели и критерии оценивания:

Промежуточная аттестация по дисциплине «Создание мобильных приложений для IoT» включает теоретические вопросы, позволяющие оценить уровень усвоения обучающимися знаний, и практические задания, выявляющие степень сформированности умений и владений.

Показатели и критерии промежуточной аттестации:

– на оценку **«отлично»** (5 баллов) – обучающийся демонстрирует высокий уровень сформированности компетенций, всестороннее, систематическое и глубокое знание

учебного материала, свободно выполняет практические задания, свободно оперирует знаниями, умениями, применяет их в ситуациях повышенной сложности.

– на оценку **«хорошо»** (4 балла) – обучающийся демонстрирует средний уровень сформированности компетенций: основные знания, умения освоены, но допускаются незначительные ошибки, неточности, затруднения при аналитических операциях, переносе знаний и умений на новые, нестандартные ситуации.

– на оценку **«удовлетворительно»** (3 балла) – обучающийся демонстрирует пороговый уровень сформированности компетенций: в ходе контрольных мероприятий допускаются ошибки, проявляется отсутствие отдельных знаний, умений, навыков, обучающийся испытывает значительные затруднения при оперировании знаниями и умениями при их переносе на новые ситуации.

– на оценку **«неудовлетворительно»** (2 балла) – обучающийся демонстрирует знания не более 20% теоретического материала, допускает существенные ошибки, не может показать интеллектуальные навыки решения простых задач.

– на оценку **«неудовлетворительно»** (1 балл) – обучающийся не может показать знания на уровне воспроизведения и объяснения информации, не может показать интеллектуальные навыки решения простых задач.

Обучающийся получает отметку **«зачтено»** при условии выполнения и защиты всех предусмотренных практических заданий. Оценка промежуточной аттестации соответствует средней оценке по результатам практических работ.

Приложение 3

Методические указания для практических занятий

Практическое занятие №1

Среда разработки приложений IntelliJ IDEA

Цель занятия: знакомство со средой разработки приложений IntelliJ IDEA.

Теоретическая часть

Android – операционная система, которая широко используется в мобильных устройствах (смартфонах, смарт часах и планшетных компьютерах) и во встраиваемых системах. Это бесплатная операционная система, основанная на ядре *Linux* с интерфейсом программирования *Java*. Приложение для *Android* пишется на языке *Java* или на более современном языке *Kotlin*. Наиболее популярной средой разработки приложений для мобильных устройств и встраиваемых систем в настоящее время является *Android Studio*, основанная на *IntelliJ IDEA* от компании *JetBrains*. Она предоставляет интегрированные инструменты для разработки и отладки. Для отладки приложений используется эмулятор мобильного устройства – виртуальная машина, на которой будет запускаться созданное приложение.

В курсе «Создание мобильных приложений для IoT» для изучения основ языка *Kotlin* нами будет использоваться *IntelliJ IDEA*. Загрузить бесплатную версию *Community* можно по адресу https://www.jetbrains.com/idea/download/. Данная среда доступна как для *Windows*, так и для *MacOS* и *Linux*.

После установки запустим *IntelliJ IDEA*. Нам откроется стартовое окно программы, в котором выберем создание нового проекта *New Project*.

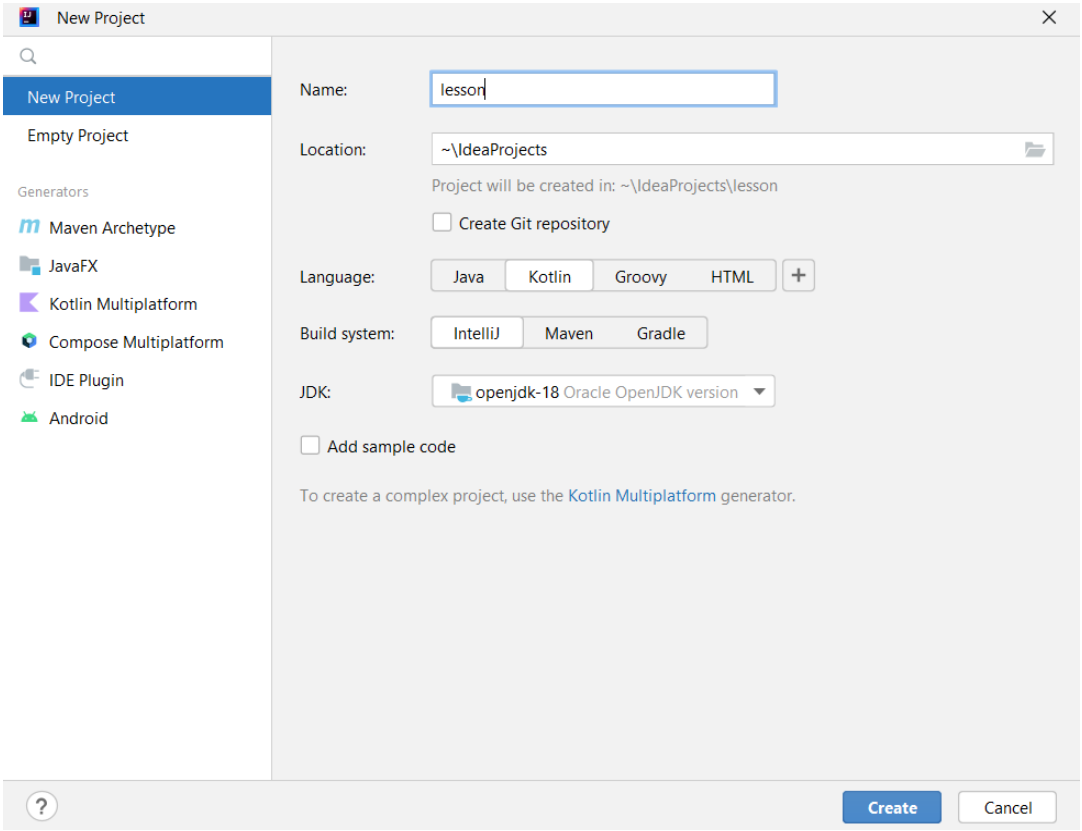

Рис. 1. Окно менеджера SDK

В поле *Name* укажем имя проекта, в поле *Location* можно указать путь к проекту, если не устраивает путь по умолчанию. Поскольку мы будем работать с языком *Kotlin*, в поле *Language* выберем пункт *Kotlin*

Кроме того, в поле *JDK* можно указать путь к *Java SDK*, который будет использоваться в проекте. Как правило, это поле по умолчанию уже содержит путь к *JDK*, который установлен на локальном компьютере. Если это поле пусто, то его надо установить.

После этого нажмем на кнопку *Create*, среда создаст и откроет проект.

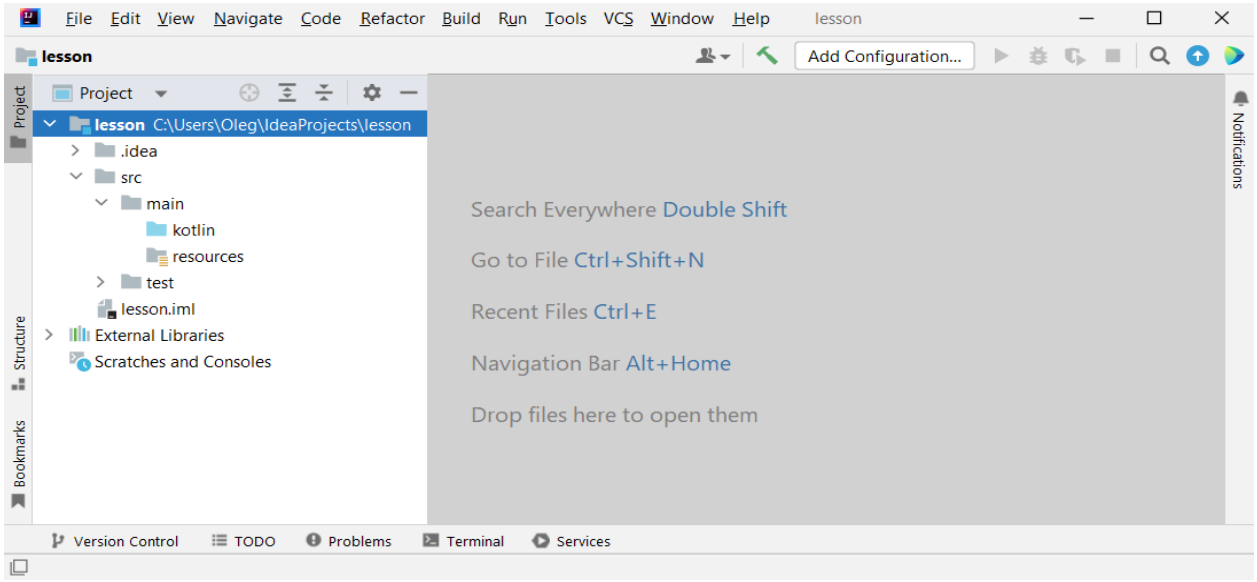

Рис. 2. Окно проекта

В левой части представлена структура проекта. Все файлы с исходным кодом помещаются в папку *src*. По умолчанию она содержит две папки: папка *main* (предназначена для кода программы) и папка *tests* (предназначена для тестов).

В папке *main* также по умолчанию создается папка *kotlin* для файлов с кодом на языке *Kotlin*. По умолчанию эта папка пуста. Добавим файл с исходным кодом. Для этого нажмем на папку *src/main/kotlin* правой кнопкой мыши и в контекстном меню выберем пункт *New -> Kotlin Class/File*:

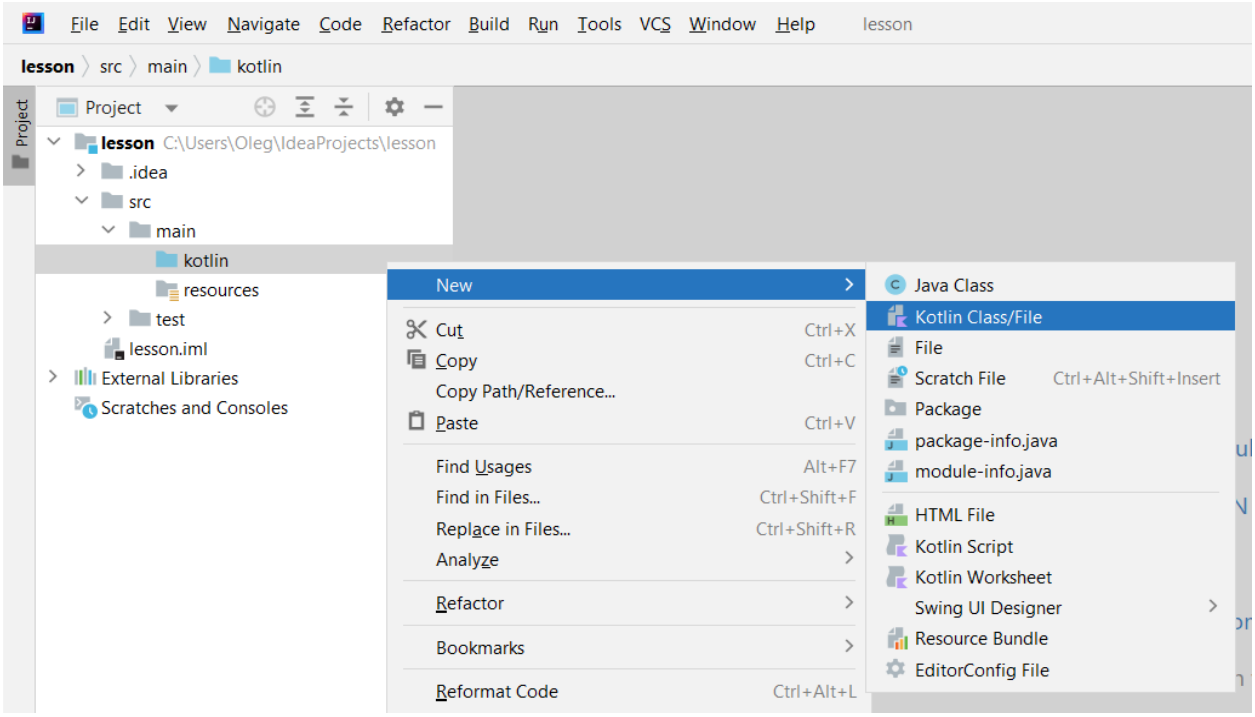

Рис. 3. Создание нового файла с исходным кодом

В появившемся окне следует указать имя нового файла, указать тип *File* и нажать *Enter*.

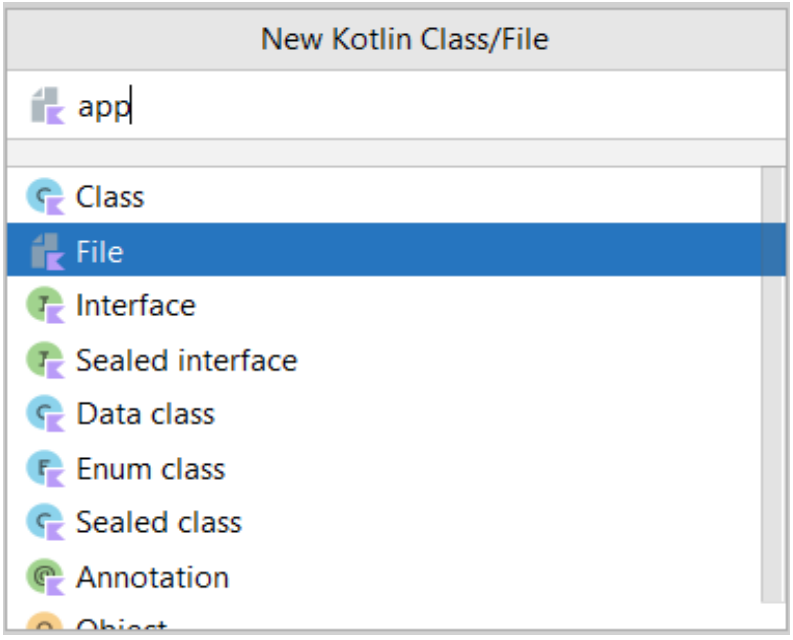

Рис. 4. Создание нового файла с исходным кодом

После нажатия на клавишу *Enter* в папку *src* будет добавлен новый файл с кодом *Kotlin* (в случае выше файл *app.kt*). А в центральной части откроется его содержимое. По умолчанию новый файл пуст. Добавим в него следующий код:

```
fun main() {
    println("Hello World")
}
```

Точкой входа в программу на *Kotlin* является функция *main*. Для определения функции применяется ключевое слово *fun*, после которого идет название функции – то есть *main*. Данная функция не принимает никаких параметров, поэтому после названия функции указываются пустые скобки.

В фигурных скобках определяются те действия, которые выполняет функция *main*. В данном случае внутри функции *main* выполняется другая функция – *println()*, которая выводит некоторое сообщение на консоль.

Для запуска программы нажмем на значок *Kotlin* рядом с первой строкой кода или на название файла и выберем в появившемся меню пункт *Run 'AppKt'*:

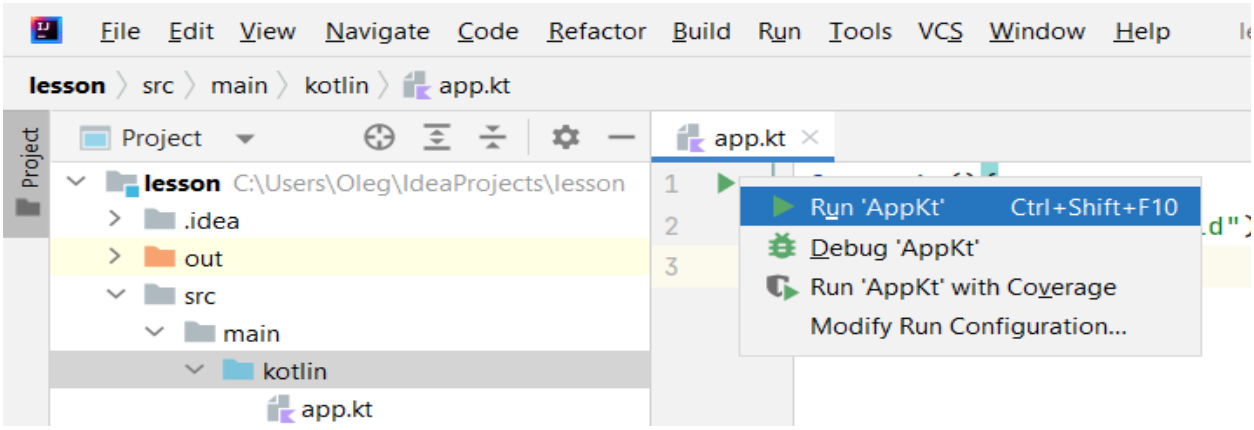

Рис. 5. Запуск программы

После этого будет выполнено построение проекта, и скомпилированная программа будет запущена в консоли в *IntelliJ IDEA*:

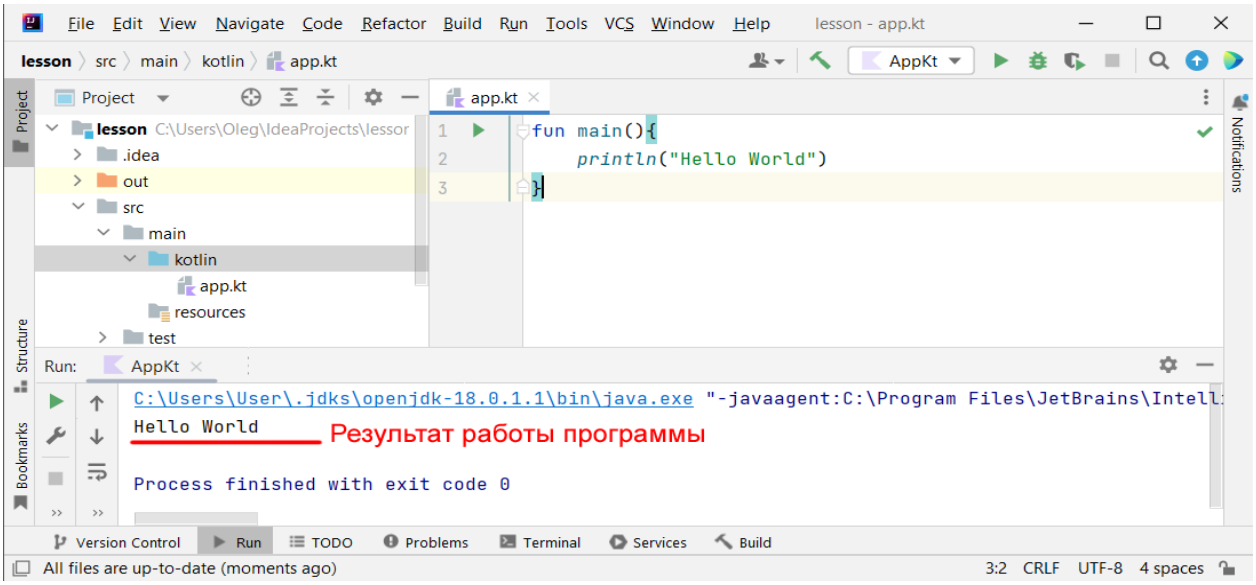

Рис. 6. Результат работы программы

Практическое задание

Создать и запустить проект, выполнив указанные в теоретической части шаги.

Практическое занятие №2

Переменные и типы данных

Цель занятия: изучение видов переменных и их базовых типов.

Теоретическая часть

Для хранения данных в программе в *Kotlin*, как и в других языках программирования, применяются переменные. Переменная представляет собой именованный участок памяти, который хранит некоторое значение.

Каждая переменная характеризуется определенным именем, типом данных и значением. Имя переменной представляет произвольный идентификатор, который может содержать алфавитно-цифровые символы или символ подчеркивания и должен начинаться либо с алфавитного символа, либо со знака подчеркивания. Для определения переменной можно использовать либо ключевое слово *val*, либо ключевое слово *var*.

Синтаксис определения переменной:

```
val|var имя_переменной: тип_переменной
```

Например, определим переменную *age*:

```
val age : Int
```

В данном случае объявлена переменная *age*, которая имеет тип *Int*, который говорит о том, что переменная будет содержать целочисленные значения.

После определения переменной ей можно присвоить значение:

```
fun main(){
    val age : Int
    age = 20
    println(age)
}
```

Для присвоения значения переменной используется знак равно. Затем мы можем производить с переменной различные операции. Например, в данном случае с помощью функции *println()* значение переменной выводится в консоль. И при запуске этой программы в консоль будет выведено число 20.

Присвоение значения переменной должно производиться только после ее объявления. Мы можем присвоить переменной начальное значение сразу при ее объявлении. Такой прием называется инициализацией:

```
fun main(){
    val age : Int = 20
    println(age)
}
```

Использовать переменную до ее инициализации нельзя.

С помощью ключевого слова *val* определяется неизменяемая переменная (*immutable variable*). Присвоить значение такой переменной можно только один раз, но изменить его

после первого присвоения мы уже не сможем. Например, в следующем случае мы получим ошибку:

```
fun main(){
     val age : Int = 20
     age = 25              //В этой строке ошибка
     println(age)
}
```

Значение переменной, которая определена с помощью ключевого слова *var* мы можем многократно менять (*mutable variable*):

```
fun main(){
     var age : Int = 20
     age = 25              //Ошибки нет
     println(age)
}
```

Поэтому если не планируется изменять значение переменной в программе, то лучше определять ее с ключевым словом *val*.

В *Kotlin* все компоненты программы, в том числе переменные, представляют объекты, которые имеют определенный тип данных. Тип данных определяет, какой размер памяти может занимать объект данного типа и какие операции с ним можно производить. В *Kotlin* есть несколько базовых типов данных: числа, символы, строки, логический тип и массивы.

Целочисленные типы

Byte: хранит целое число от -128 до 127 и занимает 1 байт

Short: хранит целое число от -32 768 до 32 767 и занимает 2 байта

Int: хранит целое число от -2 147 483 648 (-231) до 2 147 483 647 (231 - 1) и занимает 4 байта

Long: хранит целое число от –9 223 372 036 854 775 808 (-263) до 9 223 372 036 854 775 807 (263-1) и занимает 8 байт

В последней версии *Kotlin* также добавлена поддержка для целочисленных типов без знака:

UByte: хранит целое число от 0 до 255 и занимает 1 байт

UShort: хранит целое число от 0 до 65 535 и занимает 2 байта

UInt: хранит целое число от 0 до 232 - 1 и занимает 4 байта

ULong: хранит целое число от 0 до 264-1 и занимает 8 байт

```
fun main(){
     val a : Int = -10
     val b : Int = 15
```

```
}
```

Для передачи значений объектам, которые представляют беззнаковые целочисленные типы данных, после числа указывается суффикс *U*:

```
fun main(){
     val a : UByte = 10U
     val b : UInt = 15U
}
```

Кроме целочисленных типов в *Kotlin* есть два типа для чисел с плавающей точкой, которые позволяют хранить дробные числа:

Float: хранит число с плавающей точкой от $-3.4*10^{38}$ до $3.4*10^{38}$ и занимает 4 байта.

Double: хранит число с плавающей точкой от $\pm5.0*10^{-324}$ до $\pm1.7*10^{308}$ и занимает 8 байт.

В качестве разделителя целой и дробной части применяется точка.

Чтобы присвоить число объекту типа *Float* после числа указывается суффикс *f* или *F*.

Тип *Boolean* может хранить одно из двух значений: *true* (истина) или *false* (ложь).

Символьные данные представлены типом *Char*. Он представляет отдельный символ, который заключается в одинарные кавычки.

Строки представлены типом *String*. Строка представляет последовательность символов, заключенную в двойные кавычки, либо в тройные двойные кавычки.

Шаблоны строк (*string templates*) представляют удобный способ вставки в строку различных значений, в частности, значений переменных. Так, с помощью знака доллара *$* мы можем вводить в строку значения различных переменных:

```
fun main(){
     val text = "World"
     println("Hello $text")
}
```

В данном случае вместо *$text* будет подставляться значение переменной *text*. При этом переменные необязательно должны представлять строковый тип.

Kotlin позволяет выводить тип переменной на основании данных, которыми переменная инициализируется. Поэтому при инициализации переменной тип можно опустить:

```
val age = 10
```

В данном случае компилятор увидит, что переменной присваивается значение типа *Int*, поэтому переменная *age* будет представлять тип *Int*. После того, как тип переменной установлен, он не может быть изменен.

В *Kotlin* также есть тип *Any*, который позволяет присвоить переменной данного типа любое значение.

Практическое задание

1. В созданном на первом практическом занятии проекте объявить и инициализировать целочисленную неизменяемую переменную. Вывести ее значение в консоль, используя строковый шаблон.

Контрольные вопросы

30. Что такое переменная?
31. Какие виды переменных в языке *Kotlin* вы знаете? В чем их различия?
32. Какие типы переменных в языке *Kotlin* вы знаете?
33. Можно ли в ходе программы изменить значение ранее инициализированной переменной, объявленной ключевым словом *val*?
34. Можно ли инициализировать переменную типа *UByte* значением -255? Ответ обосновать.
35. Какими типами объявляются переменные, хранящие дробные значения?

Практическое занятие №3

Арифметические операции

Цель занятия: изучение основных арифметических операций над числами.

Теоретическая часть

Kotlin поддерживает базовые арифметические операции:

«+» – (сложение): возвращает сумму двух чисел;

«-» – (вычитание): возвращает разность двух чисел;

«*» – (умножение): возвращает произведение двух чисел;

«/» – (деление): возвращает частное двух чисел.

Если в операции деления оба операнда представляют целые числа, то результатом тоже будет целое число, а если в процессе деления образовалась дробная часть, то она отбрасывается.

Чтобы результатом было дробное число, один из операндов должен представлять число с плавающей точкой:

```
fun main(){
     val a = 11
     val b = 5.0
     val c = a / b
     println(c)      //Результат вывода 2.2
}
```

«%» – возвращает остаток от целочисленного деления двух чисел;

«++» – (инкремент): увеличивает значение на единицу.

Префиксный инкремент возвращает увеличенное значение:

```
val a = 4
val b = ++a
println(b)              //Вывод: 6
```

Постфиксный инкремент возвращает значение до увеличения на единицу:

```
val a = 4
val b = a++
println(b)              //Вывод: 4
```

«--» – (декремент): уменьшает значение на единицу. Префиксный и постфиксный декременты работают аналогично инкрементам.

«+=» – присваивание после сложения. Присваивает левому операнду сумму левого и правого операндов: A += B эквивалентно A = A + B;

«-=» – присваивание после вычитания. Присваивает левому операнду разность левого и правого операндов: A -= B эквивалентно A = A - B;

«*=» – присваивание после умножения. Присваивает левому операнду произведение левого и правого операндов: A *= B эквивалентно A = A * B;

«/=» – присваивание после деления. Присваивает левому операнду частное левого и правого операндов: A /= B эквивалентно A = A / B;

«%=» – присваивание после деления по модулю. Присваивает левому операнду остаток от целочисленного деления левого операнда на правый: A %= B эквивалентно A = A % B

Ряд операций выполняется над двоичными разрядами числа. Например, число 4 в двоичном виде - 100, а число 15 - 1111.

«*shl*» – сдвиг битов числа со знаком влево:

```
val a = 3 shl 2        //a = 11 << 2 = 1100, т.е. 12
```

«*shr*» – сдвиг битов числа со знаком вправо;

«*ushr*» – сдвиг битов беззнакового числа вправо;

«*and*» – побитовая операция AND (логическое умножение или конъюнкция). Эта операция сравнивает соответствующие разряды двух чисел и возвращает единицу, если эти разряды обоих чисел равны 1. Иначе возвращает 0;

«*or*» – побитовая операция OR (логическое сложение или дизъюнкция). Эта операция сравнивает два соответствующих разряда обоих чисел и возвращает 1, если хотя бы один разряд равен 1. Если оба разряда равны 0, то возвращается 0;

«*xor*» – побитовая операция XOR. Сравнивает два разряда и возвращает 1, если один из разрядов равен 1, а другой равен 0. Если оба разряда равны, то возвращается 0;

«*inv*» – логическое отрицание или инверсия - инвертирует биты числа.

Практическое задание

1. Написать программу вывода в консоль результатов приведенных в теоретической части арифметических и поразрядных операций.

Контрольные вопросы

4. Чем отличается префиксный инкремент от постфиксного? Приведите примеры.
5. Как при делении целых чисел получить результат, содержащий дробную часть?
6. Каков будет результат операции: `val a = 6 and 7`?

Практическое занятие №4

Условные выражения и логические операции

Цель занятия: изучение основных условных выражений и логических операций в *Kotlin*.

Теоретическая часть

Условные выражения представляют некоторое условие, которое возвращает значение типа *Boolean*: либо *true* (если условие истинно), либо *false* (если условие ложно).

«>» – (больше чем): возвращает *true*, если первый операнд больше второго. Иначе возвращает *false*;

«<» – (меньше чем): возвращает *true*, если первый операнд меньше второго. Иначе возвращает *false*;

«>=» – (больше чем или равно): возвращает *true*, если первый операнд больше или равен второму. Иначе возвращает *false*;

«<=» – (меньше чем или равно): возвращает *true*, если первый операнд меньше или равен второму. Иначе возвращает *false*;

«==» – (равно): возвращает *true*, если первый операнд равен второму. Иначе возвращает *false*;

«!=» – (равно): возвращает *true*, если операнды не равны. Иначе возвращает *false*;

Операндами в логических операциях являются два значения типа Boolean. Нередко логические операции объединяют несколько операций отношения:

«*and*» – возвращает *true*, если оба операнда равны *true*;

«*or*» – возвращает *true*, если хотя бы один из операндов равен *true*;

«x*or*» – возвращает *true*, если только один из операндов равен *true*. Если операнды равны, возвращается *false*;

«!» – возвращает *true*, если операнд равен *false*. И, наоборот, если операнд равен *true*, возвращается *false*.

В качестве альтернативы оператору ! можно использовать метод *not*():

```
val a = true
val b = a.not()
```

«*in*» – возвращает *true*, если операнд имеется в некоторой последовательности

```
val a = 5
val b = a in 1..6     //Результат true
```

Практическое задание

1. Написать программу вывода в консоль результатов условных выражений и логических операций, рассмотренных в теоретической части.

Контрольные вопросы

4. В чем разница выполнения операции *or* над логическими и числовыми операндами?
5. В чем разница между операциями = и ==?
6. Какая операция над логическими операндами обозначается восклицательным знаком?

Практическое занятие №5

Условные конструкции

Цель занятия: изучение условных конструкций и их синтаксиса в *Kotlin*.

Теоретическая часть

Условные конструкции позволяют направить выполнение программы по одному из путей в зависимости от условия.

Конструкция *if...else*

Конструкция *if* принимает условие, и если это условие истинно, то выполняется последующий блок инструкций.

```
val a = 5
if (a == 5) {
    println(a)
}
```

В данном случае в конструкции *if* проверяется истинность выражения *a* == 5. Если оно истинно, то выполняется последующий блок кода в фигурных скобках, и в консоль выводится значение *a*. Если же выражение ложно, тогда блок кода не выполняется.

Если необходимо задать альтернативный вариант, то можно добавить блок *else*:

```
val a = 5
if (a == 5) {
    println("true")
} else {
    println("false")
}
```

Таким образом, если условное выражение после оператора *if* истинно, то выполняется блок после *if*, если ложно – выполняется блок после *else*.

Если блок кода состоит из одного выражения, то фигурные скобки можно опустить.

Если необходимо проверить несколько альтернативных вариантов, то можно добавить выражения *else if*:

```
val a = 5
if (a == 5) {
    println("a = 5")
} else if (a == 6) {
    println("a = 6")
} else {
    println("not 5 and not 6")
}
```

В языке *Kotlin* конструкция *if* может возвращать значение. Например, найдем максимальное из двух чисел:

```
val a = 5
val b = 6
val c = if (a > b) a else b
println(c)            //Результат 6
```

Если при определении возвращаемого значения надо выполнить еще какие-нибудь действия, то можно заключить эти действия в блоки кода:

```
val a = 5
val b = 6
val c = if (a > b) {
    println(a)
    a
} else {
    println(b)
    b
}
```

В конце каждого блока указывается возвращаемое значение.

Конструкция *when*

Конструкция *when* проверяет значение некоторого объекта и в зависимости от его значения выполняет тот или иной код. Конструкция *when* аналогична конструкции *switch* в других языках. Синтаксис:

```
when (объект) {
    значение1 -> действия1
    значение2 -> действия2
    ...
}
```

Если значение объекта равно одному из значений в блоке кода *when*, то выполняются соответствующие действия, которые идут после оператора **->** после соответствующего значения.

Например:

```
fun main(){
    val isEnabled = true
    when (isEnabled) {
        false -> println("Button disabled")
        true -> println("Button enabled")
    }
}
```

Здесь в качестве объекта в конструкцию *when* передается переменная *isEnabled*. Далее ее значение по порядку сравнивается со значениями в *false* и *true*. В результате в консоль будет выведено «*Button enabled*».

В примере выше переменная *isEnabled* имела только два возможных варианта: *true* и *false*. Однако чаще бывают случаи, когда значения в блоке *when* не покрывают все

возможные значения объекта. Дополнительное выражение *else* позволяет задать действия, которые выполняются, если объект не соответствует ни одному из значений.

Например:

```
fun main(){
     val a = 5
     when (a) {
         10 -> println("a = 10")
         20 -> println("a = 20")
         else -> println("other value")
     }
}
```

Если нужно, чтобы при совпадении значений выполнялось несколько инструкций, то для каждого значения можно определить блок кода в фигурных скобках.

Можно определить одни и те же действия сразу для нескольких значений. В этом случае значения перечисляются через запятую.

Например:

```
fun main(){
     val a = 5
     when (a) {
         10,20 -> println("a = 10 or a = 20")
         else -> println("other value")
     }
}
```

Также можно сравнивать с целым диапазоном значений с помощью оператора *in*:

```
fun main(){
     val a = 5
     when (a) {
         in 10..19 -> println("a is in 10..19")
         in 20..29 -> println("a is in 20..29")
         !in 10..19 -> println("a is not in 10..19")
         else -> println("other value")
     }
}
```

Если оператор *in* позволяет узнать, есть ли значение в определенном диапазоне, то связка операторов !*in* позволяет проверить отсутствие значения в определенной последовательности.

Выражение в *when* также может сравниваться с динамически вычисляемыми значениями:

```
fun main(){
     val a = 10
     val b = 5
     val c = 2
```

```
    when (a) {
        b - c -> println("a = b - c")
        b + 5 -> println("a = b + 5")
        else -> println("other value")
    }
}
```

Кроме того, *when* также может принимать динамически вычисляемый объект.

Например:

```
when (a + b) {
    ...
}
```

Можно определять переменные, которые будут доступны внутри блока *when*:

```
fun main(){
    val a = 10
    val b = 5
    when (val c = a + b) {
        10 -> println("a + b = 10")
        15 -> println("a + b = 15")
        else -> println("c = $c")
    }
}
```

Как и *if* конструкция *when* может возвращать значение. Возвращаемое значение указывается после оператора **->**.

Практическое задание

1. Повторить приведенные в теоретической части примеры в IntelliJ IDEA.
2. Изменить значения переменных и проверить результаты выполнения программы.

Контрольные вопросы

5. Можно ли заменить конструкцию *if* конструкцией *when*?
6. В каких случаях следует использовать *when*, а в каких *if*?
7. Объяснить синтаксис конструкции *if*.
8. Объяснить синтаксис конструкции *when*.

Практическое занятие №6

Циклы

Цель занятия: изучение циклов и их синтаксиса в *Kotlin*.

Теоретическая часть

Циклы представляют вид управляющих конструкций, которые позволяют в зависимости от определенных условий выполнять некоторое действие множество раз.

Цикл *for*

Цикл *for* перебирает все элементы коллекции. Его синтаксис выглядит следующим образом:

```
for (переменная in последовательность) {
    действия
}
```

Например, выведем все квадраты чисел от 1 до 9, используя цикл *for*:

```
for (a in 1..9) {
    print("${a * a} \t")
}
```

В данном случае перебирается последовательность чисел от 1 до 9. При каждом проходе цикла (итерации цикла) из этой последовательности будет извлекаться элемент и помещаться в переменную *a*. И через переменную *a* можно манипулировать значением элемента. То есть в данном случае мы получим следующий консольный вывод:

```
1   4   9   16   25   36   49   64   81
```

Цикл с предусловием *while*

Цикл *while* повторяет определенные действия пока истинно некоторое условие:

```
var i = 10
while (i > 0) {
    println(i*i)
    i--
}
```

Здесь пока переменная *i* больше 0, будет выполняться цикл, в котором на консоль будет выводиться квадрат значения *i*. В данном случае вначале проверяется условие ($i > 0$) и если оно истинно (то есть возвращает *true*), то выполняется цикл.

Цикл с постусловием *do..while*

```
var i = -1
do {
    println(i*i)
    i--
```

```
}
while (i > 0)
```

В данном случае вначале выполняется блок кода после ключевого слова *do*, а потом оценивается условие после *while*. Если условие истинно, то повторяется выполнение блока после *do*. То есть несмотря на то, что в данном случае переменная *i* меньше 0 и она не соответствует условию, тем не менее блок *do* выполнится хотя бы один раз.

Иногда при использовании цикла возникает необходимость при некоторых условиях не дожидаться выполнения всех инструкций в цикле, перейти к новой итерации. Для этого можно использовать оператор *continue*:

```
for (a in 1..9) {
    if (a == 5) continue
    print("${a * a} \t")
}
```

В данном примере когда *a* будет равно 5, сработает оператор *continue*. И последующая инструкция, которая выводит в консоль квадрат числа, не будет выполняться. Цикл перейдет к обработке следующего элемента в массиве.

Бывает, что при некоторых условиях нам вовсе надо выйти из цикла, прекратить его выполнение. В этом случае применяется оператор *break*:

```
for (a in 1..9) {
    if (a == 5) break
    print("${a * a} \t")
}
```

В данном случае когда *a* окажется равен 5, то с помощью оператора *break* будет выполнен выход из цикла. Цикл полностью завершится.

Практическое задание

1. Написать программу вывода в консоль квадратов чисел от 1 до 20, используя циклы.
2. Написать программу вывода в консоль таблицы умножения, используя циклы.

Контрольные вопросы

4. В каком случае следует использовать цикл *for*, а в каком *while*?
5. Каковы функции служебных слов *break* и *continue*?
6. В чем разница между циклами с предусловием и постусловием?

Практическое занятие №7

Массивы и диапазоны

Цель занятия: изучение приемов работы с массивами и диапазонами в *Kotlin*.

Теоретическая часть

Для хранения набора значений в *Kotlin*, как и в других языках программирования, можно использовать массивы. При этом массив может хранить данные только одного того же типа. В *Kotlin* массивы представлены типом *Array*.

При определении массива после типа *Array* в угловых скобках необходимо указать, объекты какого типа могут храниться в массиве. Например, определим массив целых чисел:

```
var numbers : Array<Int>
```

С помощью встроенной функции *arrayOf*() можно передать набор значений, которые будут составлять массив:

```
var numbers : Array<Int> = arrayOf(1, 2, 3, 4, 5)
```

То есть в данном случае в массиве 5 чисел от 1 до 5.

С помощью индексов мы можем обратиться к определенному элементу в массиве. Индексация начинается с нуля, то есть первый элемент будет иметь индекс 0. Индекс указывается в квадратных скобках:

```
var numbers : Array<Int> = arrayOf(1, 2, 3, 4, 5)
var n = numbers[1]          //Получили значение 2-го элемента
numbers[2] = 8              //Изменили значение 3-го элемента
```

Также инициализировать массив значениями можно следующим способом:

```
var numbers = Array(3,{5})//массив [5, 5, 5]
```

Здесь применяется конструктор класса *Array*. В этот конструктор передаются два параметра. Первый параметр указывает, сколько элементов будет в массиве. В данном случае 3 элемента. Второй параметр представляет выражение, которое генерирует элементы массива. Оно заключается в фигурные скобки. В данном случае в фигурных скобках стоит число 5, то есть все элементы массива будут представлять число 5. Таким образом, массив будет состоять из трех пятерок.

Для перебора массивов можно применять цикл *for*:

```
var numbers = arrayOf(1, 2, 3, 4, 5)
for (number in numbers) {
    print("$number \t")
}
```

Вывод: 1 2 3 4 5

Подобным образом можно перебирать массивы и других типов.

Для упрощения создания массива в *Kotlin* определены дополнительные типы *BooleanArray*, *ByteArray*, *ShortArray*, *IntArray*, *LongArray*, *CharArray*, *FloatArray* и *DoubleArray*, которые позволяют создавать массивы для определенных типов. Например, тип *IntArray* позволяет определить массив объектов *Int*, а *DoubleArray* - массив объектов *Double*:

```
var numbers = intArrayOf(1, 2, 3, 4, 5)
var numbers = doubleArrayOf(2.7, 3.5, 3.1)
```

Выше рассматривались одномерные массивы, которые можно представить в виде ряда или строки значений. Но кроме того, мы можем использовать многомерные массивы.

```
var table = Array(3, { Array(3, {0}) })
```

Диапазон представляет набор значений или неокторый интервал. Для создания диапазона применяется оператор .. (две точки).

```
var numbers = 1..5
```

Диапазон необязательно должна представлять числовые данные. Например, это могут быть строки:

```
var letters = "a"."e"
```

Практическое задание

1. Написать программу вывода в консоль английского алфавита, используя диапазоны и цикл.

Контрольные вопросы

5. Что такое массив?
6. Какие варианты объявления и инициализации массива вы знаете?
7. С какого числа начинается нумерация элементов массива?
8. Как в языке Kotlin можно объявить массив из дробных чисел?