# 1장. 강력하고 실용적인 코틀린

이 장에서는 현대적인 애플리케이션을 개발할 때 코틀린이 훌륭하고 효율적인 언어인 이유와 여러분이 코틀린을 배워야 하는 이유를 설명한다. 코틀린 설계를 뒷받침하는 기본적인 사상과 여러 응용 영역에서 사용할 수 있는 코틀린 라이브러리와 프레임워크를 전반적으로 살펴본다. 특히 안드로이드 애플리케이션, 동시성, 테스트, 웹 개발 분야를 살펴본다. 마지막으로 두 가지 유명한 개발 환경인 인텔리J(IntelliJ) IDEA와 이클립스(Eclipse)에서 코틀린 프로젝트를 설정하는 방법을 단계별로 알아보고, 대화형 코틀린 셸에 대해 소개한다.

## 구조(무엇을 배우는가? 배우는 내용 등으로 일괄 통일)

* 코틀린이란 무엇인가?
* 코틀린 생태계를 이루는 주요 구성요소
* 인텔리J와 이클립스에서 코틀린 프로젝트 설정하기

## 목표

이 장을 완료하면 코틀린의 기본적인 원리와 코틀린 생태계에 대해 이해하고, 간단한 코틀린 프로그램이 어떻게 생겼는지, 자주 쓰는 IDE에서 코틀린 프로젝트를 어떻게 설정하는지 알게 된다.

## 코틀린이란 무엇인가?

코틀린은 안전성, 간결성, 상호운용성을 강조하는 다중 패러다임, 다중 플랫폼 프로그래밍 언어다. 2010년 하반기 처음 생겨서 2016년 2월 처음 공개 릴리스됐으며, 안드로이드 개발, 데스크탑 애플리케이션, 서버 측 솔루션 등 여러 개발 분야에서 유망한 도구로 꾸준히 인기를 얻고 있다. 코틀린 언어를 뒷받침하고 코틀린 언어 개발에 계속 투자해온 회사는 젯브레인즈(JetBrains)로, 젯브레인즈는 인텔리J IDEA 등 여러 훌륭한 소프트웨어 엔지니어링 도구로도 유명하다.

2019년 11월 코틀린 1.3이 나왔고, 2021년 5월 1.5.0이 나왔다. 코틀린 커뮤니티도 아주 커졌고, 잘 개발된 생태계와 광범위한 도구들이 만들어졌다. 원래 언어를 만든 목적, 즉 자바에 대해 더 나은 대안을 제공하려는 목적을 넘어 코틀린이 크게 자라면서, 이제 코틀린은 자바 가상 기계, 안드로이드, 자바스크립트, 네이티브 애플리케이션 등 여러 플랫폼을 지원한다. 2017년 구글(Google)은 안드로이드 플랫폼의 공식 지원 언어로 코틀린을 선택했다. 이로 인해 코틀린 언어 사용자가 엄청나게 늘어났다. 현재 구글, 아마존(Amazon), 넷플릭스(Netflix), 핀터레스트(Pinterest), 우버(Uber) 등 여러 회사가 프로덕션 개발에 코틀린을 사용하고 있으며, 코틀린 개발자를 채용하려는 회사가 계속 늘어나고 있다.

이 모든 일을 이룰 수 있었던 것은 언어를 주의 깊게 설계하고 언어의 주요 특성에 그 설계를 녹여 내기 위해 헌신적으로 노력했기 때문이다. 이러한 노력으로 인해 코틀린은 아주 훌륭한 개발자 도구가 됐다. 언어 철학은 2010년 코틀린이 마주했던 문제에서 주로 기인한다. 그 당시 젯브레인즈에는 인텔리J 플랫폼을 중심으로 하는 자바 코드 기반이 누적되어 있었는데, 여기에는 가장 잘 알려져 있는 인텔리J IDEA뿐만 아니라 웹스톰(WebStorm), Php스톰(PhpStorm), 루비마인(RubyMine) 등 마이너한 IDE도 포함되었다.

하지만 코드 기반의 성장과 유지보수는 자바 언어 자체로 인해 방해 받기 시작했다. 자바의 발전이 느렸고 2010년 당시 스칼라나 C# 등의 언어에서 사용할 수 있었던 유용한 기능이 자바에는 없었기 때문이다. 당시 사용 가능한 JVM 언어를 검토한 결과 젯브레인즈는 자신의 필요를 충족할 만한 언어가 없다는 결론을 내리고, 그에 따라 자원을 투입하여 지신만의 언어를 개발하기로 결정했다. 새로운 언어에는 개발 팀원의 상당수가 위치했던 러시아 상트페테르부르크 근처의 섬인 코틀린이라는 이름이 붙었다.

그렇다면 처음에 코틀린 언어를 만들어간 특징은 무엇일까? 사실 코틀린 언어 정의에서 이미 이야기했다. 코틀린 개발을 뒷받침하는 이유는 안전성, 간결성, 상호운용성을 강조하는 다중 패러다임 언어가 필요하다는 것이었다. 각각의 특징을 더 자세히 살펴보자.

## 안전성

프로그래밍 언어가 안전하려면 프로그래머의 오류를 방지할 수 있어야 한다. 실전에서 오류를 방지하려면 대가가 필요하기 때문에 안전성을 고려해 언어를 설계하는 것은 트레이드오프(tradeoff)의 문제다. 컴파일러에게 프로그램에 대한 정보를 더 자세히 알려주거나, 컴파일러가 더 오랫동안 프로그램이 올바른지 추론하게 한다(또는 두 가지 다 할 수도 있다). 코틀린의 설계 목표 중 하나는 황금비를 찾아내는 것이었다. 즉, 자바보다 더 안전성을 보장하는 언어를 설계하되, (자바보다는) 개발자의 생산성을 덜 해치는 언어를 만드는 것이었다. 그리고 코틀린의 해법은 어떤 측면에서든 절대적인 것이 아니었으나, 실질적으로 효율적인 선택임을 반복적으로 증명해왔다.

이 책을 진행하면서 코틀린의 안전성을 여러 가지 측면에서 설명할 것이다. 여기서는 몇 가지 중요한 특징을 짚고 넘어가겠다.

* 타입 추론(type inference)으로 인해 개발자가 명시적으로 타입을 지정하지 않아도 된다(자바10에도 지역 변수에 대한 타입 추론이 도입됐다).
* 널이 될 수 있는 타입(nullable type)을 통해 null의 사용을 제한할 수 있고, 악명 높은 NullPointerException 예외를 좀 더 쉽게 방지할 수 있다.
* 스마트캐스트(smart cast)를 통해 타입을 안전하게 변환할 수 있고, 실행 시점에 타입 캐스트 오류를 방지할 수 있다.

### 다중 패러다임

초기에 코틀린이 다중 패러다임이라는 말의 의미는 자바와 같은 전형적인 주류 프로그래밍 언어가 지원하는 일반적인 객체지향 패러다임 외에 함수형 프로그래밍을 지원한다는 뜻이었다. 함수형 프로그래밍은 함수를 다른 값과 마찬가지로 다룬다는 생각에서 비롯된 프로그래밍 방법이다. 그래서 함수를 다른 함수의 인수로 넘기거나, 지역적으로 함수를 선언하거나, 변수에 저장하거나, 함수가 함수를 반환하는 등의 일이 가능하다. 함수형 프로그래밍의 또 다른 측면으로 불변성(immutability)이 있다. 이는 여러분이 만든 객체는 내부 상태를 바꿀 수 없고, 함수는 부수 효과(side effect)를 만들어낼 수 없다는 말이다.

이런 접근 방법의 큰 이점은 유연성이다. 새로운 유형의 추상화를 만들어낼 수 있어서 훨씬 더 표현력이 좋고 간결한 코드를 작성할 수 있으며, 이에 따라 생산성이 좋아진다.

<참고>

함수형 프로그래밍 원칙이 여러 언어에 도입됐음에도 불구하고(예를 들어 자바의 익명 클래스는 람다(lambda)를 도입하기 위한 좋은 바탕이 될 수 있다), 이런 언어들이 모두 다 함수형 코드를 작성하기 편리한 문법적인 도구를 제공하지는 않는다.

</참고>

반면, 코틀린은 처음부터 필요한 도구를 제공했다. 특히 함수 타입을 부드럽게 조화시킨 타입 시스템(type system)과 함수 타입의 값을 코드 블록에서 만들어낼 수 있는 람다를 처음부터 제공했다. 표준 라이브러리나 외부 프레임워크도 함수형 프로그래밍 스타일을 장려하는 방대한 API를 제공한다. 자바도 자바 8부터 함수형 프로그래밍을 지원했기 때문에, 요즘은 이 중 상당수가 자바에도 도입됐다. 하지만 자바의 표현력은 코틀린의 표현력에 비해 상당히 떨어진다.

함수형 프로그래밍의 기초에 대해서는 5장에서 다룰 것이다. 그리고 함수형 프로그래밍을 적용한 예제는 이 책 여러 부분에서 볼 수 있다.

코틀린 언어가 성장해 나가면서, 두 가지 프로그래밍 패러다임을 더 지원하기 시작했다. API를 **도메인 특화 언어**(domain specific language*)*로 정의할 수 있는 기능 덕분에 코틀린을 선언적인 스타일로 코딩할 수 있다. 사실 여러 코틀린 프레임워크들은 범용 프로그래밍 언어에 제공하는 타입 안전성이나 표현력을 해하지 않으면서 DSL을 제공한다. 예를 들어 exposed 프레임워크는 데이터베이스 스키마(schema)를 정의하고 데이터베이스에 저장된 데이터를 조작할 수 있는 DSL을 포함하며, kotlinx.html은 타입 안전성을 보장하는 새로운 HTML 템플릿 언어를 제시한다. **11장 도메인 특화 언어**에서 이런 예제를 더 자세히 살펴보고, 직접 DSL을 만드는 방법도 알아본다.

또 한 가지 패러다임은 이름 그대로 동시성(concurrent) 프로그래밍이다. 코루틴을 도입하면서 코틀린도 동시성 프로그래밍을 지원하기 시작했다. 자바를 포함한 여러 언어가 동시성을 지원하지만, 코틀린은 새로운 프로그래밍 접근 방법을 가능하게 해주는 풍부한 프로그래밍 패턴을 제공한다. 동시성에 대한 기본적인 내용은 **13장 동시성**에서 다룬다.

이 모두를 한꺼번에 제공하기 때문에 다중 패러다임은 언어의 표현력을 매우 많이 늘려주며, 언어를 더 유연하고 다용도에 사용할 수 있는 도구로 만들어준다.

## 간결성과 표현력

개발자 생산성은, 코드가 다른 사람이 짠 것이든 본인 스스로가 짠 것을 한참 후에 보는 것이든 관계없이, 빠르게 읽고 이해할 수 있는 코드를 작성할 수 있는지와 주로 연관되어 있다. 어떤 코드 조각의 의미를 이해하려면 그 조각이 프로그램의 다른 부분과 어떤 관계인지를 빠르게 이해해야 한다. 그래서 기존 코드의 의미를 파악하는 게 새로 코드를 작성하는 것보다 일반적으로 시간이 더 오래 걸리고, 언어의 간결성, 즉 프로그래머의 의도를 잡음 정보 없이 명확하게 표현할 수 있는 능력이 개발 도구인 프로그래밍 언어에 있어 중요하다.

코틀린 설계자들은 필드 게터(getter)나 세터(setter), 익명 클래스, 명시적인 위임 등 자바의 악명 높은 불필요한 준비 코드를 제거하고 언어를 가능한 간결하게 만들기 위해 노력했다. 반면 코틀린 설계자들은 간결성을 너무 과용하지 못하게 주의를 기울였다. 예를 들어 스칼라와 달리 코틀린에서는 커스텀 연산자를 정의할 수 없고, 기존의 연산자만 재정의할 수 있다. 이는 새 연산자로 인해 연산의 의미가 오히려 더 혼란스러워지기 때문이다. 이 책에서는 이런 결정으로 인한 결과와 결과적으로 이 결정이 얼마나 유용한지에 대한 예를 많이 볼 수 있다.

코틀린 간결성의 또 다른 측면은 DSL과 밀접히 연관된다(**11장 도메인 특화 언어** 참조). DSL은 문법적인 잡음을 최소화하면서 특정 프로그래밍 범위에 대한 기술을 아주 간결하게 만들어준다.

## 상호운용성

코틀린은 독립적으로 존재하는 언어가 아니며, 가능한 기존의 (자바) 코드 기반과 부드럽게 협력해야만 했기 때문에, 자바 상호운용성은 코틀린 설계시 중요한 목적이었다. 그래서 코틀린 설계자들은 기존 자바 코드를 코틀린에서 사용할 수 있게 했을 뿐 아니라, 코틀린 코드를 거의 아무 노력도 없이 자바에서 사용할 수 있도록 보장하기 위해 노력했다. 코틀린 언어에는 자바와 코틀린의 상호 운용성을 조절하기 위한 특별한 기능도 몇 가지 포함한다.

코틀린이 JVM을 능가하고 다른 플랫폼으로 퍼져 나감에 따라, 상호운용성 보장은 자바스크립트, 즉 JS 플랫폼이나 네이티브 애플리케이션인 C/C++/오브젝티브(Objective) C/스위프트(Swift) 코드와의 상호작용까지 포함하는 범위로 확장되었다.

**12장. 자바 상호 운용성**에서 자바 상호 운용성(Java Interoperability)을 다루고, 코틀린과 자바를 한 프로젝트 안에서 함께 사용하는 방법에 대해 설명한다.

## 다중 플랫폼

코틀린 설계자들의 원래 의도에 다중 플랫폼(Multiplatform)은 없었다. 하지만 코틀린 언어가 진화하고 채택됨에 따라 개발자 커뮤니티에 의해 다중 플랫폼 지원 필요성이 제기됐다. 여전히 JVM과 안드로이드가 코틀린 개발의 주 대상이지만, 요즘 코틀린은 다음과 같은 플랫폼도 지원한다.

* 브라우저나 노드(Node.js) 애플리케이션, 자바스크립트 라이브러리까지 포함하는 자바스크립트 지원
* macOS나 리눅스(Linux), 윈도우(Window) 라이브러리 및 애플리케이션을 포함하는 네이티브 지원

코틀린 1.3버전부터 다중 플랫폼 개발을 지원하며, 주된 용례로는 안드로이드와 iOS 애플리케이션 사이에 코드 기반을 공유하거나, JVM/JS/네이티브 세계에서 사용할 다중 플랫폼 라이브러리를 만드는 것 등이 있다.

## 코틀린 생태계

코틀린이 진화해 오면서 소프트웨어 개발의 거의 모든 측면에 대한 풍부한 라이브러리와 프레임워크가 만들어졌다. 여기서는 사용 가능한 도구를 전반적으로 설명하고, 이 가능성의 바다를 헤쳐나가기 위한 안내를 제공하려 한다. 하지만 생태계(ecosystem)가 끊임없이 자라고 있기 때문에, 이 책에서 최신이라고 소개한 내용이 시간이 지남에 따라 낡은 프로젝트가 될 것이다. 따라서 주저하지 말고 직접 관련 내용을 찾아보기 바란다. 커뮤니티는 사용 가능한 라이브러리와 프레임워크 목록을 ‘놀라운 코틀린(Awesome Kotlin)’ 웹사이트(<https://kotlin.link>)에 계속 업데이트하고 있다.

코틀린이 제공하는 좋은 자바 상호운용성 덕분에 기존 자바 라이브러리가 주는 이익을 활용할 수 있다는 점 또한 주목하자. 경우에 따라서는 자바 라이브러리를 좀 더 코틀린답게 사용할 수 있는 코틀린 확장이 있을 수도 있다.

## 코루틴

일시중단 가능한 계산(suspendable computation)이라는 개념 덕분에 코틀린은 async/await, 퓨처(future), 프라미스(promise), 액터(actor) 등 동시성 관련 프로그래밍 패턴을 지원할 수 있다. 코루틴 프레임워크는 서버 쪽이거나 모바일, 데스크톱을 위한 것인지와 관계 없이 코틀린 애플리케이션에서 동시성 문제를 해결할 수 있는 강력하고 우아하며 쉽게 확장 가능한 해법을 제공한다. 코루틴에 포함된 주요 기능은 다음과 같다.

* 스레드에 대한 더 가벼운(lightweight) 대안
* 유연한 스레드 디스패치(dispatch) 메커니즘
* 일시중단 가능한 시퀀스와 이터레이터(iterator)
* 채널(channel)을 사용한 메모리 공유
* 액터를 통해 메시지 전송 방식으로 상태를 공유

**13장 동시성**에서 코루틴 API와 코틀린에서 동시성을 다루는 방법의 기초를 다룰 것이다.

## 테스트

테스트를 정의하거나 객체를 모킹(mocking)하고 싶을 때 JUnit, TestNG, 모키토(Mockito) 같은 익숙한 자바 테스트 프레임워크 중 일부를 거의 아무런 노력도 기울이지 않고 코틀린에 적용할 수 있고, 개발자는 코틀린 맞춤용 테스트를 위해 유용한 DSL을 제공하는 프레임워크의 강력함을 즐길 수도 있다. 특히 다음 내용을 언급하고 싶다.

* 모키토-코틀린(Mockito-Kotlin)은 유명한 모키토 프레임워크를 확장한 것으로 코틀린 객체 모킹을 단순화해준다.
* 스펙(Spek)은 행위 주도(behavior-driven) 테스트 프레임워크로, 자스민(Jasmine)이나 저킨(Gherkin) 스타일의 테스트 케이스 정의를 지원한다.
* 코테스트(Kotest)는 스칼라테스트(ScalaTest)에 의해 영감을 얻은 프레임워크로, 유연한 테스트 정의와 단언문(assertion)을 지원한다.

**14장 코틀린 테스팅**에서는 이 중 코테스트가 지원하는 기능을 더 자세히 살펴보고, 프로젝트에서 어떻게 사용할 것인지 생각해본다.

## 안드로이드 개발

안드로이드는 코틀린의 주된 응용 분야이자 가장 활발하게 성장하는 분야다. 특히 구글이 코틀린을 1급 안드로이드 언어로 선언하면서 안드로이드 개발에서 코틀린이 중요해졌다. 안드로이드 도구들은 이제 코틀린의 특징을 염두에 두고 설계되고, 개발된다. 안드로이드 스튜디오 플러그인이 제공하는 훌륭한 프로그래밍 경험과 더불어, 안드로이드 개발자들은 대거(Dagger), 버터나이프(ButterKnife), DB플로우(DBFlow) 등 여러 유명한 프레임워크와의 원활한 상호 운용성을 얻을 수 있다. 코틀린에 특화된 안드로이드 개발 도구 중에서는 안코(Anko)와 코틀린 안드로이드 확장(Kotlin Android Extension) 등이 있었고, 현재는 구글이 제공하는 젯팩(Jetpack) 에 주목할 만하다.

코틀린 안드로이드 확장은 데이터바인딩이 주요 특징인 컴파일러 플러그인이다. 이를 사용하면 XML로 정의한 뷰를 코드 안에서 정의한 것처럼 사용할 수 있기 때문에 악명 높은 findViewById() 호출을 피할 수 있다. 그 외에도 뷰 캐싱을 지원하고, 사용자가 정의한 클래스에 대한 Parcelable 구현을 자동으로 생성해준다. 이로 인해 순수 코틀린 프로젝트에서 버터나이프 같은 외부 프레임워크를 채용할 필요가 없어졌다.

안코는 코틀린 라이브러리로, 안드로이드 앱 개발을 단순화시켜 준다. 여러 가지 도움이 되는 기능 중에는 안드로이드 SQLite에 대한 데이터베이스 질의(query) DSL과 안드로이드 스튜디오에서 사용할 수 있는 UI 미리보기 플러그인이 제공되는 동적 레이아웃 합성 영역 특화 언어(안코 레이아웃즈(Anko Layouts)가 있다.

구글이 안드로이드 개발 지원 기능을 젯팩에 대거 추가하면서 코틀린 안드로이드 확장과 안코 라이브러리의 기능을 상당부분 흡수했다. 아직 알파버전이지만 젯팩을 사용하면 편리하게 안드로이드 UI 개발을 진행할 수 있다.

**15장 안드로이드 애플리케이션**에서는 젯팩이 제공하는 기능 중 뷰 및 데이터 바인딩과 레이아웃 DSL에 대해 다루면서 코틀린을 활용한 안드로이드 개발에 대해 소개한다.

## 웹 개발

웹/엔터프라이즈 애플리케이션 개발자들도 코틀린의 이점을 살릴 수 있다. 스프링 5.0이나 Vert.x 3.0과 같은 유명한 프레임워크들은 자신의 기능을 좀 더 코틀린다운 방식으로 활용할 수 있게 해주는 코틀린용 확장을 제공한다. 그 외에도 다양한 프레임워크를 사용하는 순수 코틀린 해법을 사용할 수도 있다.

* Ktor는 젯브레인즈에서 만든 비동기 서버와 클라이언트 애플리케이션 작성을 위한 프레임워크다.
* kotlinx.html은 HTML 문서를 만들기 위한 DSL이다.
* 코데인(Kodein)은 의존관계 주입 프레임워크다.

16장과 17장에서 Ktor를 활용한 웹 애플리케이션과 스프링 부트를 활용한 마이크로서비스 개발의 세부 사항을 다룰 것이다.

## 코틀린 시작하기

지금까지 코틀린 생태계에 대해 알아보았다. 코틀린 언어를 탐구하기 위해 이제 작업 환경을 어떻게 설정하느냐만 남았다.

### 인텔리J 프로젝트 설정하기

대부분의 프로그래밍 언어와 마찬가지로 코틀린 자체는 어떤 특정 IDE나 텍스트 편집기에 매여 있지는 않다. 하지만 어떤 개발자 도구를 선택하느냐는 개발자의 생산성에 아주 큰 영향을 미친다. 현재는 젯브레인즈의 인텔리J 플랫폼이 코틀린 개발 생명 주기를 가장 강력하고 종합적으로 지원해준다. 처음부터 코틀린 IDE는 언어 자체와 밀접한 연관을 가지고 개발되어 왔으며, 이로 인해 최신 코틀린 변경 사항을 가장 잘 반영한다. 이런 이유로 인텔리J IDEA를 사용하기를 권장하며, 이 책의 예제에서도 이를 사용할 것이다.

인텔리J IDEA 15부터 IDE 배포판에 코틀린 지원이 기본 포함되어 있으므로, 코틀린 개발을 시작하기 위해 다른 플러그인을 설치할 필요가 없다. 이 책에서는 2021년 4월 말에 배포된 인텔리J IDEA 2021.1.2를 기본으로 설명한다.

IDEA를 설치하지 않았다면 www.jetbrains.com/idea/download에서 최종 버전을 내려받아(그림 1-1) https://www.jetbrains.com/help/idea/installation-guide.html에 있는 절차를 따라 설치하라. IDEA는 두 가지 에디션이 있다. 커뮤니티(Community) 에디션은 무료 오픈 소스 버전이며, 얼티밋(Ultimate)은 상업적으로 판매되는 버전이다. 얼티밋이 웹과 엔터프라이즈 개발에 관계된 기능뿐 아니라 데이터베이스 관련 기능까지 제공한다. 더 자세한 차이는 다운로드 페이지에서 확인할 수 있다. 이 책에서는 얼티밋이 제공하는 기능이 필요 없으므로 IDEA 커뮤니티 버전으로도 충분하다.

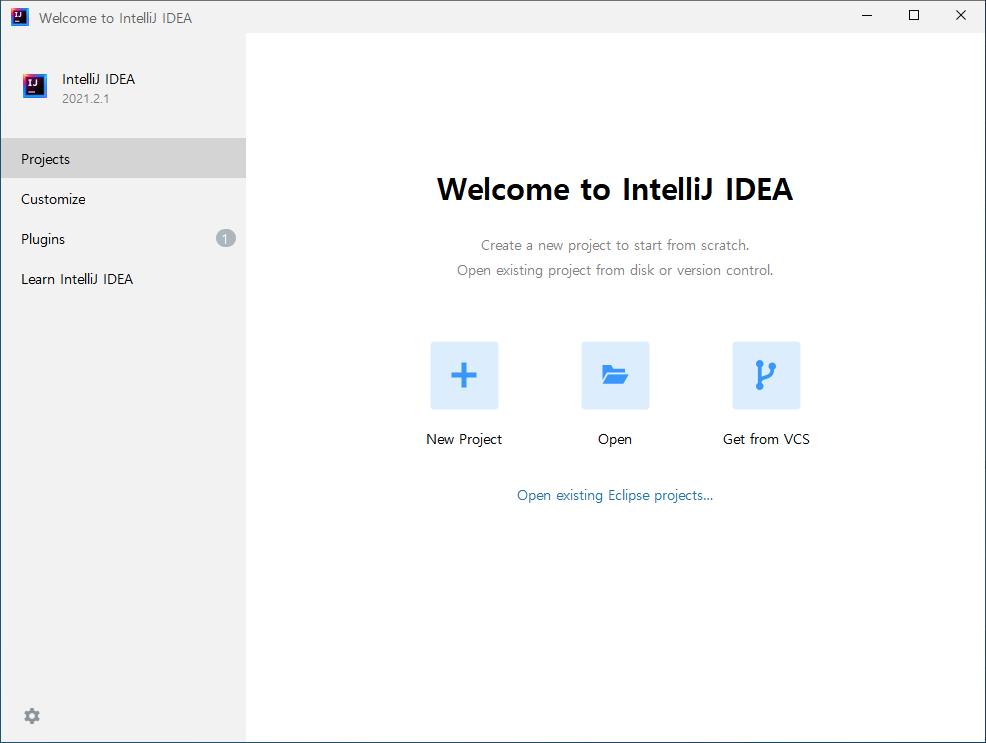
그림 1-1 인텔리J IDEA다운로드 페이지

텍스트이(가) 표시된 사진

자동 생성된 설명

인텔리J에서 프로젝트를 열어본 적이 없다면 시작 시 환영 창에서 "New Project" 옵션을 볼 수 있다(그림 1-2). 이 옵션을 선택하면 바로 프로젝트 마법사를 시작한다. 아니면 최근에 사용했던 프로젝트를 보여준다. 이 경우 File > New > Projet...를 애플리케이션 메뉴에서 선택하라.

그림 1-2 인텔리J 설치 후 최초 실행 시 나타나는 환영 창



프로젝트 유형은 분류 별로 창 왼쪽 사각 영역에 표시된다(그림 1-3). 프로젝트의 분류나 유형은 설치한 플러그인에 따라 다를 수 있는데, 현재는 기본적으로 설치되어 있는 코틀린 분류에 속하는 프로젝트에 주목하자. 왼쪽 영역에서 코틀린을 선택하면 그림 1-4와 같이 새로 만들 코틀린 프로젝트의 프로젝트 템플릿과 이름, 디렉터리 위치, 빌드 시스템, JDK 등을 선택하는 화면을 볼 수 있다.

그림 1-3 새 프로젝트 마법사

텍스트이(가) 표시된 사진

자동 생성된 설명

버전 1.5.2(2021년 8월 기준)에서 코틀린 플러그인은 JVM, 자바스크립트, 네이티브 애플리케이션을 대상으로 하는 프로젝트 생성을 지원하며, 안드로이드와 iOS도 지원한다. 플랫폼은 사용할 컴파일러 결과물의 유형(JVM의 경우 바이트코드, 자바스크립트의 경우 .js 파일, 코틀린/네이티브의 경우 플랫폼에 따라 달라지는 실행 파일)과 프로젝트에서 사용할 수 있는 의존관계의 종류를 결정한다. 예를 들어 자바스크립트를 대상으로 하는 프로젝트에서는 자바 클래스 라이브러리에 있는 클래스에 접근할 수 없다. 이 책에서는 주로 JVM 애플리케이션에 초점을 맞춘다. 따라서 이 예제에서는 JVM 아래 있는 Application 옵션을 선택하라.

그림 1-4 화면에서 프로젝트 이름과 위치도 결정해야 한다. 프로젝트 위치는 소스파일 등 프로젝트 관련 콘텐츠가 들어갈 최상위 디렉터리를 지정한다. 인텔리J가 입력한 프로젝트 이름에 따라 자동으로 위치를 제안한다. 하지만 원한다면 이를 변경할 수 있다.

그림 1-4 코틀린 유형을 선택하면 나타나는 화면

텍스트이(가) 표시된 사진

자동 생성된 설명

만들고 있는 프로젝트가 JVM 플랫폼을 대상으로 하므로 프로젝트를 컴파일할 때 사용할 JDK를 지정해야 한다. 이를 통해 어떤 표준 자바 라이브러리를 사용하고 언어를 혼합해 사용하는 프로젝트라면 자바 소스코드를 어떻게 컴파일할지 지정한다. **12장 자바 상호운용성**에서 이런 프로젝트에 대해 더 자세히 다루고, 기존 자바 프로젝트에 코틀린 지원을 추가하는 방법도 다룬다.

프로젝트 JDK로는 JDK8 이상을 권장한다. 이 예제에서는 www.oracle.com에서 다운로드할 수 있는 최신(2021년 5월 기준) 버전인 JDK 16을 선택했다. 보통 인텔리J가 여러분의 시스템에 설치된 JDK를 자동으로 감지하지만, 그렇지 않거나 프로젝트 SDK 목록에 적합한 JDK가 없다면, 프로젝트 JDK 콤보박스에서 다운로드(Download JDK)를 선택해 새로운 JDK를 추가할 수 있다. 또는, JDK를 따로 설치하고 콤보박스에서 추가(Add JDK)를 선택해 인텔리J에 추가할 수도 있다. JDK를 직접 설치해야 할 경우에는 https://www.oracle.com/java/technologies/javase-downloads.html 에 들어가서 설치 절차를 따라 원하는 JDK를 설치하라.

인텔리J가 프로젝트에 미리 설정해주는 코틀린 런타입 라이브러리에 대해 잠시 설명하겠다. 프로젝트는 디폴트로 IDE 플러그인 디렉터리에 있는 라이브러리를 참조한다. 이 라이브러리는 플러그인이 업데이트될 때 자동으로 업그레이드된다. 하지만 여러분의 프로젝트가 어떤 정해진 코틀린 런타임 버전에 의존하고 필요할 때만 수동으로 이 런타임을 업데이트하고 싶다면, 이 디폴트 동작을 변경할 수 있다. Create 버튼을 누르고 Copy를 선택해서 라이브러리를 보관할 디렉터리의 이름을 설정할 수 있다.

빌드 시스템으로는 인텔리J를 선택하라. 실무에서 프로젝트를 진행한다면 그레이들(Gradle) 코틀린이나 그레이들 그루비(Groovy)를 사용할 가능성이 높지만, 다른 외부 라이브러리를 사용하지 않는 예제를 진행할 때는 인텔리J를 빌드 시스템으로 사용해도 큰 문제가 없다. 나중에 안드로이드 개발이나 웹 개발 등에 대해 설명할 때 그레이들을 사용하는 예를 볼 수 있다.

그림 1-4 화면에서 필요한 부분을 모두 선택하거나 입력하고, Next 버튼을 누르면 모듈 설정 화면이 나온다(그림 1-5). 이 책의 대부분의 예제에서는 웹 서버 기능을 사용할 일이 없으므로 템플릿으로는 콘솔 애플리케이션을 선택하고, 테스트 프레임워크로는 None을 선택한다. 테스트 설정에 대해서는 코테스트(KoTest)에 대해 설명할 때 다시 다룬다. 타깃 JVM 버전(Target JVM Version)은 컴파일러가 생성할 바이트코드(자바 가상머신이 실행할 수 있는 명령어 코드)의 버전을 결정한다. 실무라면 배포할 환경에서 사용할 JVM 버전에 따라 적절한 버전을 선택하면 되지만, 여기서는 1.8을 선택하라. 그림 1-3에서 선택한 JDK 버전보다 더 낮거나 같은 버전을 선택해야 한다. 그렇지 않으면 컴파일 후 실행 시 문제가 생길 수 있다.

그림 1-5 모듈 설정 화면

텍스트이(가) 표시된 사진

자동 생성된 설명

Finish 버튼을 클릭하면 인텔리J가 빈 프로젝트를 만들어서 열어준다. 디폴트로 인텔리J는 패널이 두 가지 있는 뷰를 보여준다. 왼쪽에는 프로젝트(Project) 도구 창이 보이고, 나머지 영역은 편집기(editor)가 차지한다. 아직 아무 파일도 열지 않았기 때문에 편집기 창은 비어 있다.

그림 1-6 인텔리 J 기본 화면

텍스트이(가) 표시된 사진

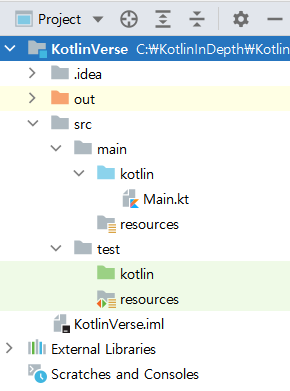
자동 생성된 설명

우선 프로젝트 창에 초점을 맞추고, 새로운 코틀린 파일을 만들자. 프로젝트 도구 창이 없다면 Project 버튼을 클릭하거나 [Alt]+[1](Meta+1) 단축키를 눌러서 열 수 있다(보통은 윈도우 보더의 왼쪽 부분을 프로젝트 창이 차지한다).

프로젝트 창은 프로젝트의 계층 구조를 나타낸다. 루트 노드를 확장해서 안의 내용을 살펴보자(그림 1-7). 흥미로운 원소가 세 가지 있다.

* src 디렉터리는 프로젝트에 포함된 소스 파일의 루트 역할을 한다[[1]](#footnote-1).
  + main 밑의 kotlin 디렉터리에는 프로젝트의 코틀린 소스코드가 들어간다.
  + test 밑의 kotlin 디렉터리에는 프로젝트를 테스트기 위한 테스트의 코틀린 소스코드가 들어간다.
  + main과 test 밑의 resources 디렉터리에는 프로그램이나 테스트에서 사용할 리소스 파일이 들어간다. 리소스 파일은 설정에 사용할 xml이나 프로퍼티(.properties) 파일, 각종 HTML 파일, 이미지 파일 등 프로그램이 실행 시점에 읽어와 사용해야 하는 다양한 파일을 뜻한다.
* out 디렉터리에는 컴파일러가 생성한 바이트코드가 들어간다. 프로젝트를 처음 만든 경우에는 이 디렉터리가 보이지 않지만, 코드를 한번 이상 빌드하면 이 디렉터리가 생긴다.
* 외부 라이브러리(External Libraries)는 프로젝트가 의존하는 모든 라이브러리를 포함한다.

그림 1-7 프로젝트 구조 도구 창



편의를 위해 인텔리J가 프로젝트에 사용할 기본 소스 코드 파일을 생성해준다. 파일은 src/main/kotlin/Main.kt다. 하지만 실습을 위해 새로 파일을 작성해보자. 우선 프로젝트 구조 트리에서 Main.kt 파일을 왼쪽 클릭해 선택하고, Del 키를 눌러 파일을 삭제하자. 삭제 확인 대화창이 나오면 OK를 선택하면 파일이 삭제된다.

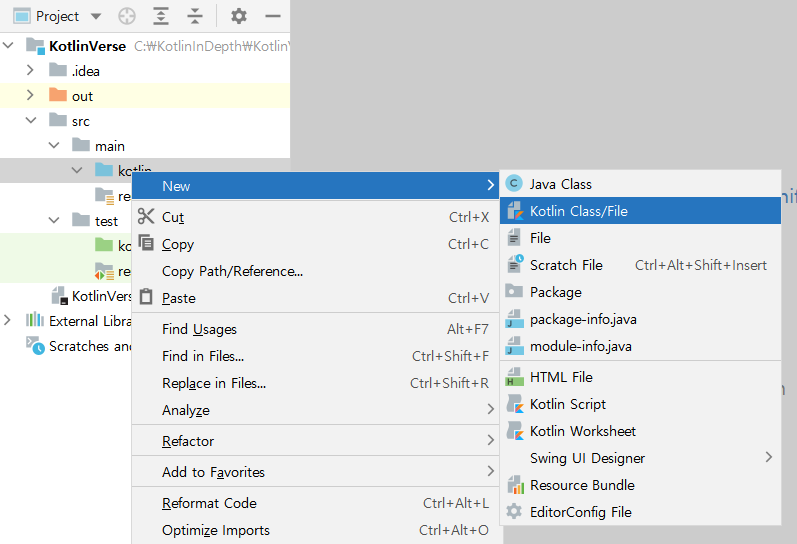
그림1-8 파일 삭제 확인 대화창

텍스트이(가) 표시된 사진

자동 생성된 설명

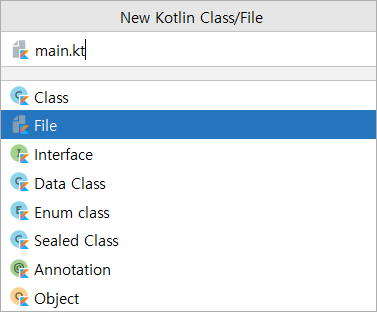
그 후, src/kotlin 디렉터리를 오른쪽 클릭해서 New > Kotlin Class/File 명령을 선택하라.

그림 1-9 New > Kotlin File/Class



대화창이 뜨면 파일 이름에 main.kt를 입력하고 Kind 필드를 File로 설정하고 [Enter]를 눌러라.

그림 1-10 main.kt



업로드한 프로젝트 창에 여러분이 입력한 이름의 파일이 새로 보이고, 편집창에도 같은 파일이 열린다. 코틀린 소스 파일의 확장자는 .kt여야 한다는 점에 유의하라.

그림 1-11 프로젝트 창과 편집창 확인

텍스트이(가) 표시된 사진

자동 생성된 설명

드디어 실제로 코드를 작성할 준비가 되었다. 편집창에 있던 인텔리 J가 미리 작성해준 내용을 모두 지우고, 다음 내용을 입력하라(그림 1-12).

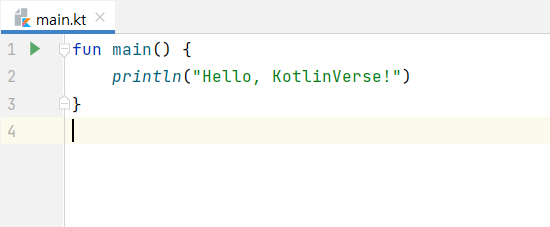
<코드>

fun main() {  
 println("Hello, KotlinVerse!")  
}

</코드>

이 코드는 코틀린 애플리케이션의 진입점(entry-point) 역할을 하는 main 함수를 보여준다. 함수 본문은 명령어 한 줄이다. 이 명령어는 인자로 받은 값을 표준 출력에 내보내주는 표준 라이브러리 함수 println()을 호출한다. 이 함수는 출력의 끝에 새줄 문자를 추가해준다.

그림 1-12 Hello, World 프로그램



자바 개발자는 분명 이 코드와 다음 자바 프로그램 사이의 유사성을 느낄 것이다.

<코드>

public class Main {  
 public static void main(String[] args) {  
 System.out.println("Hello, World!")  
 }  
}

</코드>

사실 코틀린 println() 함수를 JVM에 맞게 컴파일한 버전은 단지 System.out.println()에 대한 호출일 뿐이다. JVM 진입점은 반드시 정적 클래스 메서드(static class method)여야 하기 때문에, 이런 사실을 아는 자바 개발자라면 어떻게 코틀린 코드가 클래스를 정의하지 않고도 실행될 수 있는지 궁금할 것이다. 답은 여기서 명시적으로 클래스를 정의하지는 않았지만, 코틀린 컴파일러가 뒤에서 자동으로 클래스를 하나 만들고 그 안에 우리가 작성한 main() 함수를 호출하는 진입점을 넣어주기 때문이다. **12장 자바 상호운용성**에서 파사드(facade) 클래스라는 클래스를 살펴볼 것이다. 파사드 클래스는 코틀린/자바 상호 작용에서 가장 중요한 측면을 구성하기 때문이다.

그리고 파라미터로 반드시 명령줄 인수를 받기 위한 배열을 지정해야 하는 JVM 진입점과 달리 코틀린 main() 함수는 파라미터가 없어도 된다는 점에 유의하라. 명령줄 인수가 필요 없을 때 이런 점이 편리하다. 하지만 필요한 경우에는 인수를 받을 수도 있다.

<코드>

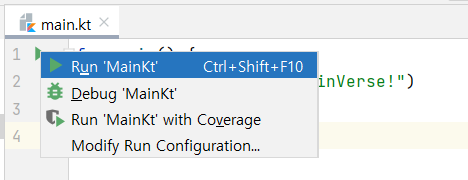
fun main() {  
 println("Hello, KotlinVerse!")  
}

</코드>

실제로 파라미터가 없는 main()은 코틀린 1.3부터 도입됐다. 1.3 버전 이전 코틀린에서는 진입점 main 함수에 String<Array> 타입의 파라미터가 반드시 있어야 했다.

main() 함수 정의 왼쪽에 있는 작은 녹색 삼각형을 눈치챈 독자도 있을 것이다. 이 마커는 main() 함수가 진입점이기 때문에 실행될 수 있다는 사실을 표시한다. 마커를 클릭하면 코드를 실행하거나 디버깅할 수 있는 메뉴가 뜬다. Run MainKt 옵션을 선택하고 어떤 일이 벌어지는지 보자(그림 1-11).

그림 1-13 프로그램 실행하기



MainKt는 몇 단락 앞에서 언급했던 컴파일러가 만들어낸 파사드 클래스의 이름이다. Run 명령을 선택하면 인텔리J는 코드를 컴파일하고 프로그램을 실행한다. 프로그램이 시작할 때 실행 도구 창이 열리고 자동으로 표준 I/O를 내장된 콘솔에 연결해준다. 모든 일이 잘 진행됐다면 프로그램이 Hello, KotlinVerse!를 콘솔에 출력하고 성공적으로 종료될 것이다.

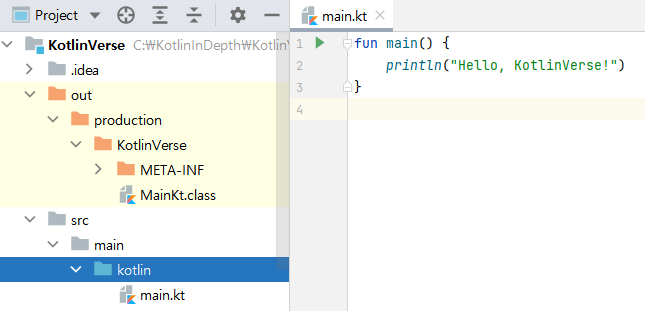
그림 1-14 Hello, KotlinVerse!

텍스트이(가) 표시된 사진

자동 생성된 설명

프로그램을 실행하고 나면 생기는(실제로는 컴파일시 생김) out 디렉터리 안을 살펴보면 소스 프로그램으로부터 코틀린 컴파일러가 생성한 .class 파일을 볼 수 있다.

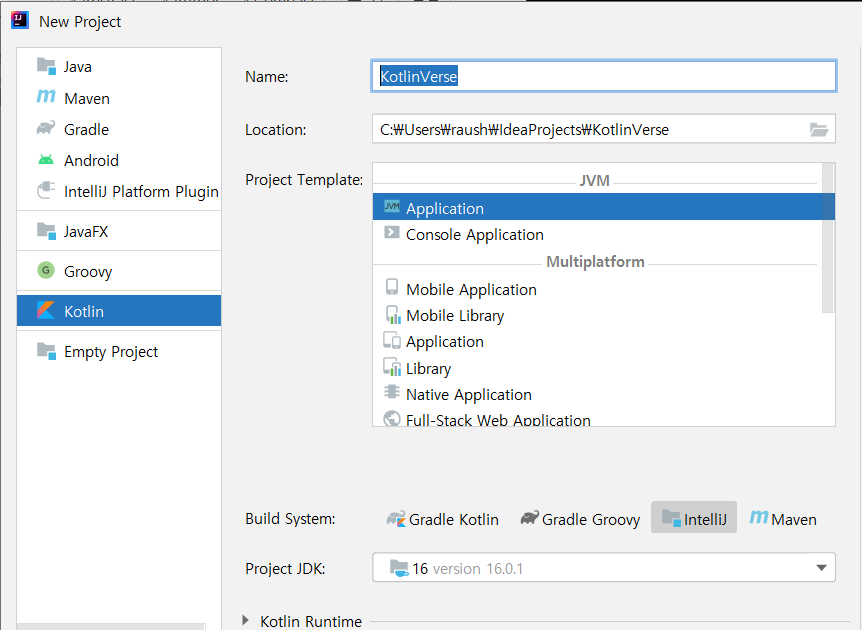
그림 1-15 out 디렉터리 안의 .class 파일



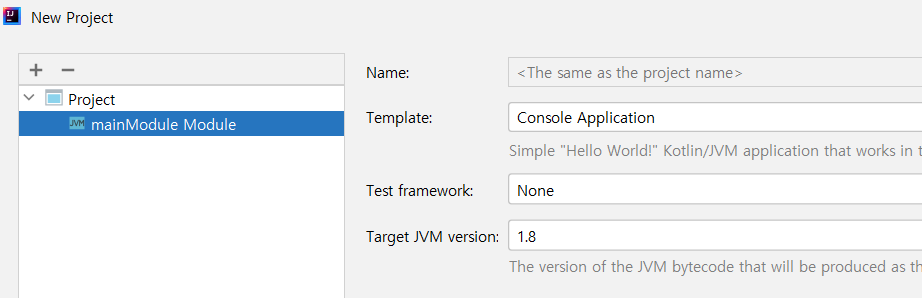
축하한다! 이제 인텔리J IDEA 환경에서 프로젝트를 설정하고 실행하는 방법을 이해했고, 코틀린의 기초인 코틀린세계(KotlinVerse)를 더 깊이 파고들 준비가 됐다. 이제 가보자!

**위의 내용 실습 시 문의사항입니다. 인텔리J 커뮤니티 2021.2 버전으로 새로 실습했습니다.**

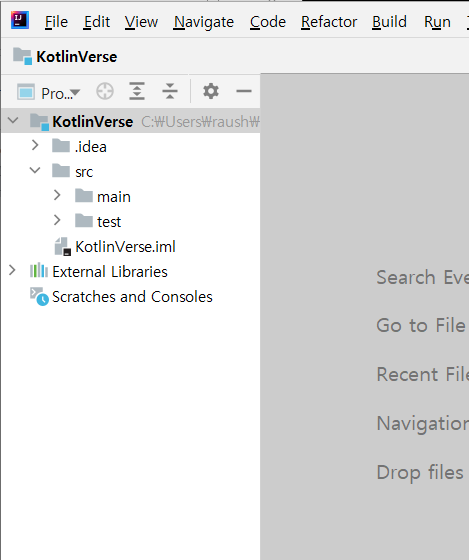
**KotlinVerse로 이름짓고, 인텔리J 선택하고, 버전은 16.**



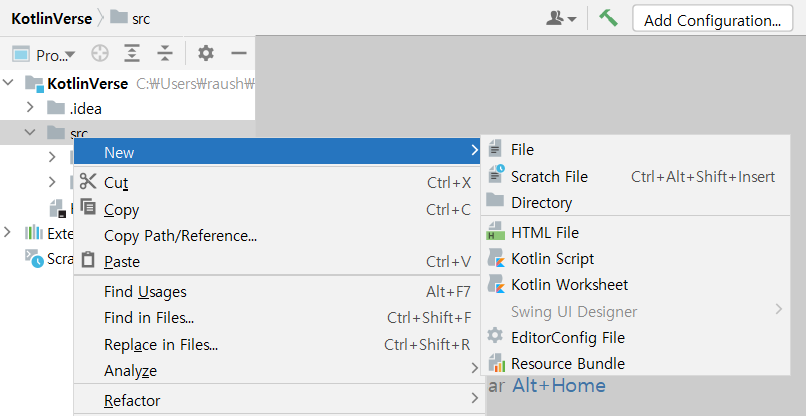
**1.8로 두고 Finish하면**



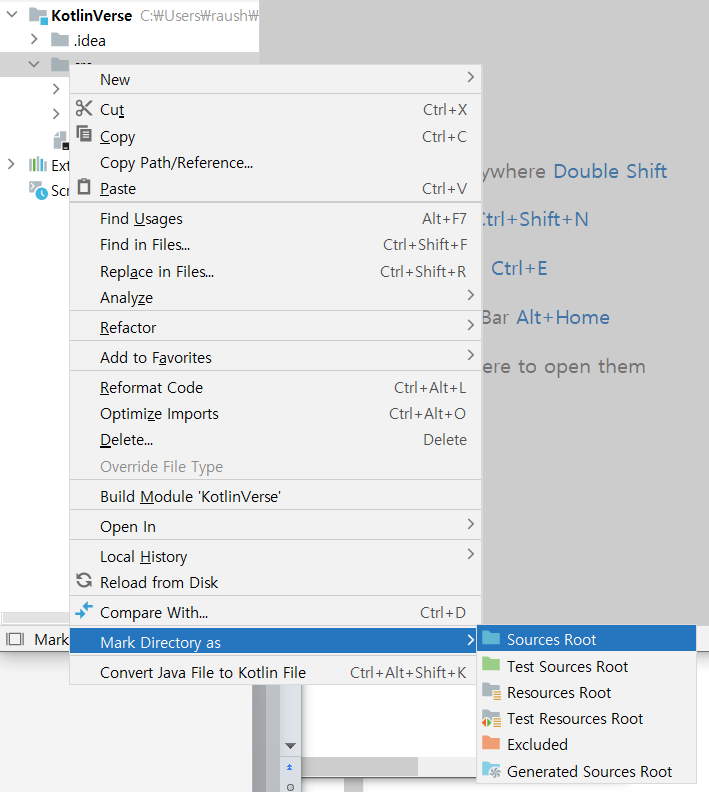
**아래와 같이 뜨는데 기존에 없던 main, test 폴더가 있습니다.**



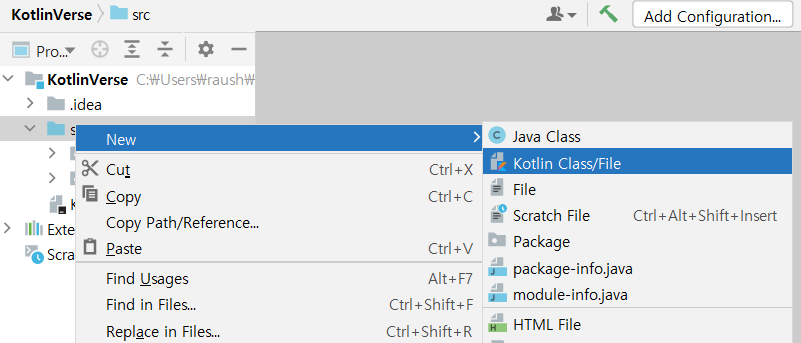
**그리고 src 디렉터리에서 New > Kotlin Class/File 명령이 없어서 선택할 수가 없습니다.**



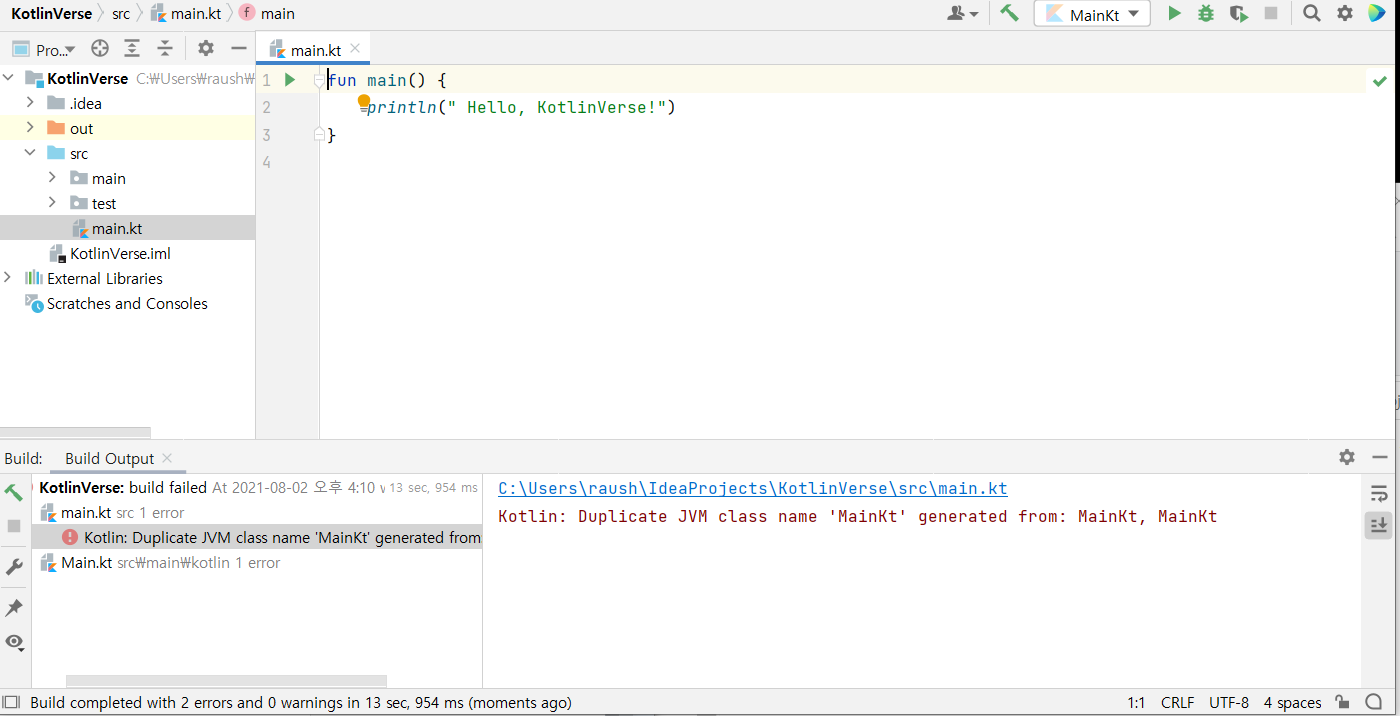
**src를 Sources Root로 만든 다음**



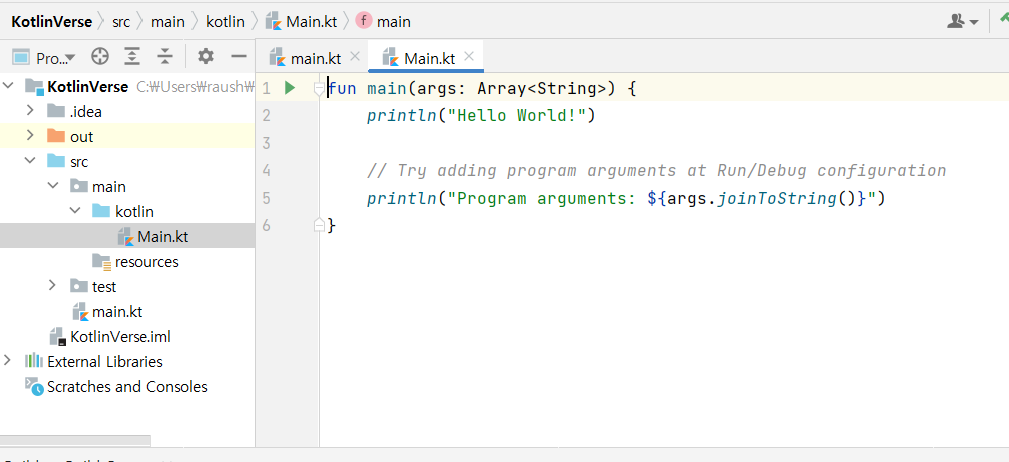
**New > Kotlin Class/File 명령을 선택하여 본문에 나온 main.kt 파일을 만들고 코드를 넣었습니다.**



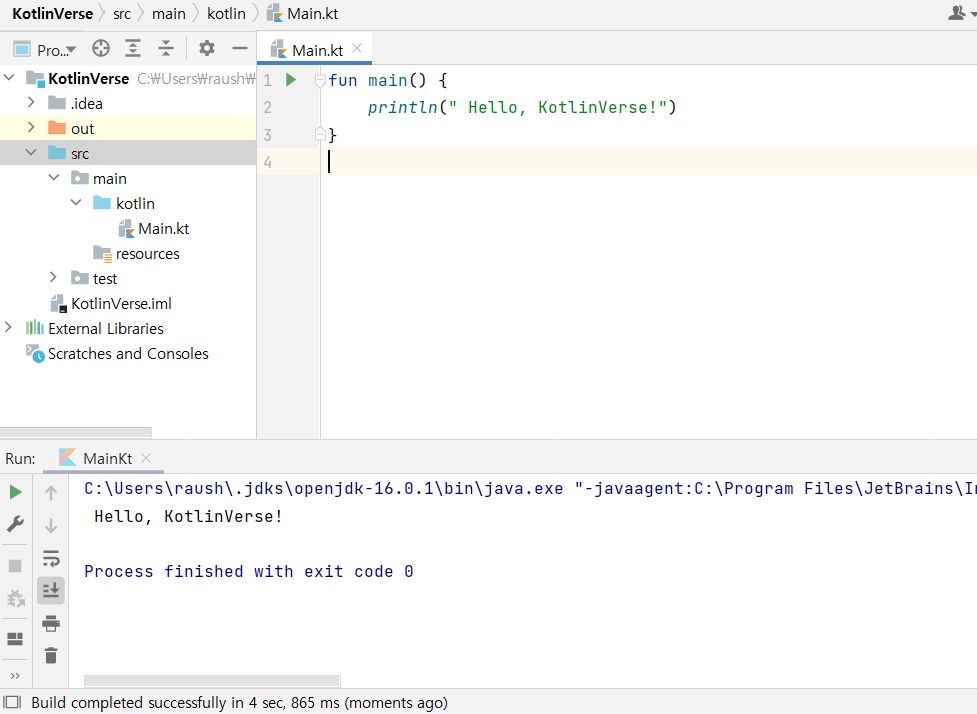
**그리고 Run을 하면 다음과 같은 에러가 뜨는데,**



**보니까 예전에는 없었던 main 폴더 안에 Main.kt 파일이 있습니다.**



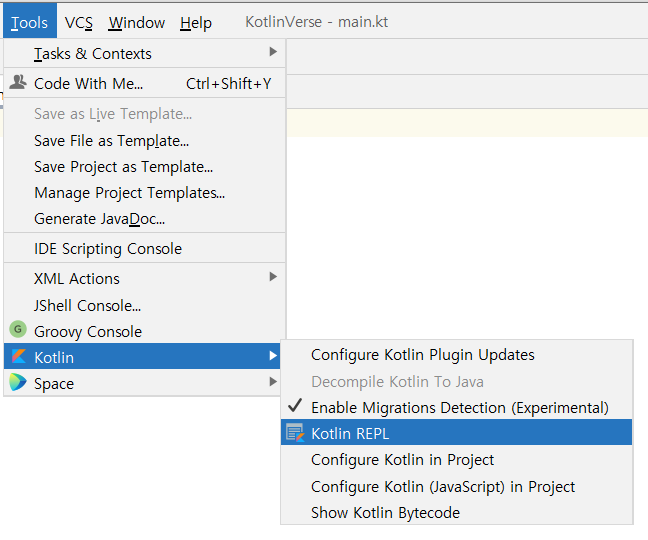
**여기에 코드를 넣고 돌리자 잘 나왔습니다. 현재 인텔리J에서 프로젝트를 만들었을 때 이렇게 Main.kt 파일까지 생기도록 바뀐 것일까요? 그렇다면 이에 맞춰서 설명을 바꿔야 할지 문의드립니다.**



## REPL 사용하기

인텔리J 코틀린 플러그인은 즉시 프로그램 실행을 평가(evaluate)할 수 있는 대화형 셸(interactive shell)을 제공한다. 이 셸을 사용해 코드를 빨리 테스트하거나, 라이브러리 함수를 실험해볼 수 있다. 그리고 코틀린 언어를 배우는 사람들에게도 대화형 셸이 유용하다. 이런 대화형 셸을 REPL이라고 부른다. REPL은 읽기/평가/출력 루프(Read/Evaluate/Print Loop)의 약자다. 이 말은 사용자가 입력한 코드를 읽어서, 평가하고, 결과를(결과가 있다면) 출력하는 과정을 계속 반복한다는 뜻이다. REPL을 열려면 Tool > Kotlin > Kotlin REPL을 선택하라.

<추가그림>



REPL 창에서는 편집기 창과 똑같이 원하는 대로 코틀린 코드를 타이핑할 수 있다. 가장 큰 차이는 [Enter] 키를 누르면 입력한 코드 조각이 실행된다는 점이다. 코드를 타이핑한 후 [Ctrl]+[Enter](Command-Return)를 눌러 입력한 내용을 IDE가 처리하게 할 수 있다. 다음을 입력하고 이 기능을 테스트해보자.

<코드>

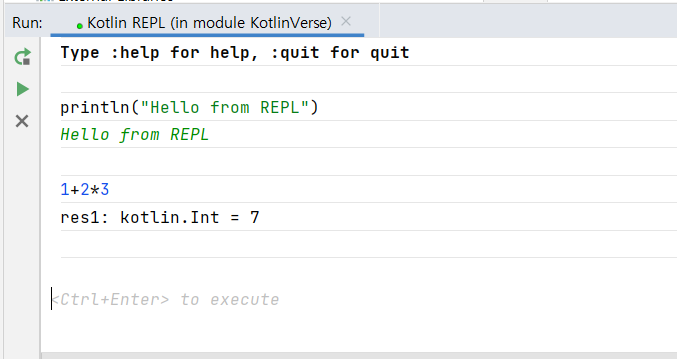
println(“Hello from REPL”)

</코드>

결과를 살펴보면 인텔리J는 Hello from REPL를 콘솔에 출력한다. 여기서는 REPL 창이 콘솔 역할을 같이 한다.

위에 입력한 문자 출력은 실제로는 println() 함수의 부수 효과이며, println 함수 자체는 아무 값도 반환하지 않는다. 하지만 의미 있는 결과를 만들어내는 식을 평가해보면 출력이 약간 달라진다. 1+2\*3을 시도해보자(그림 1-6).

그림 1-6: REPL 사용하기



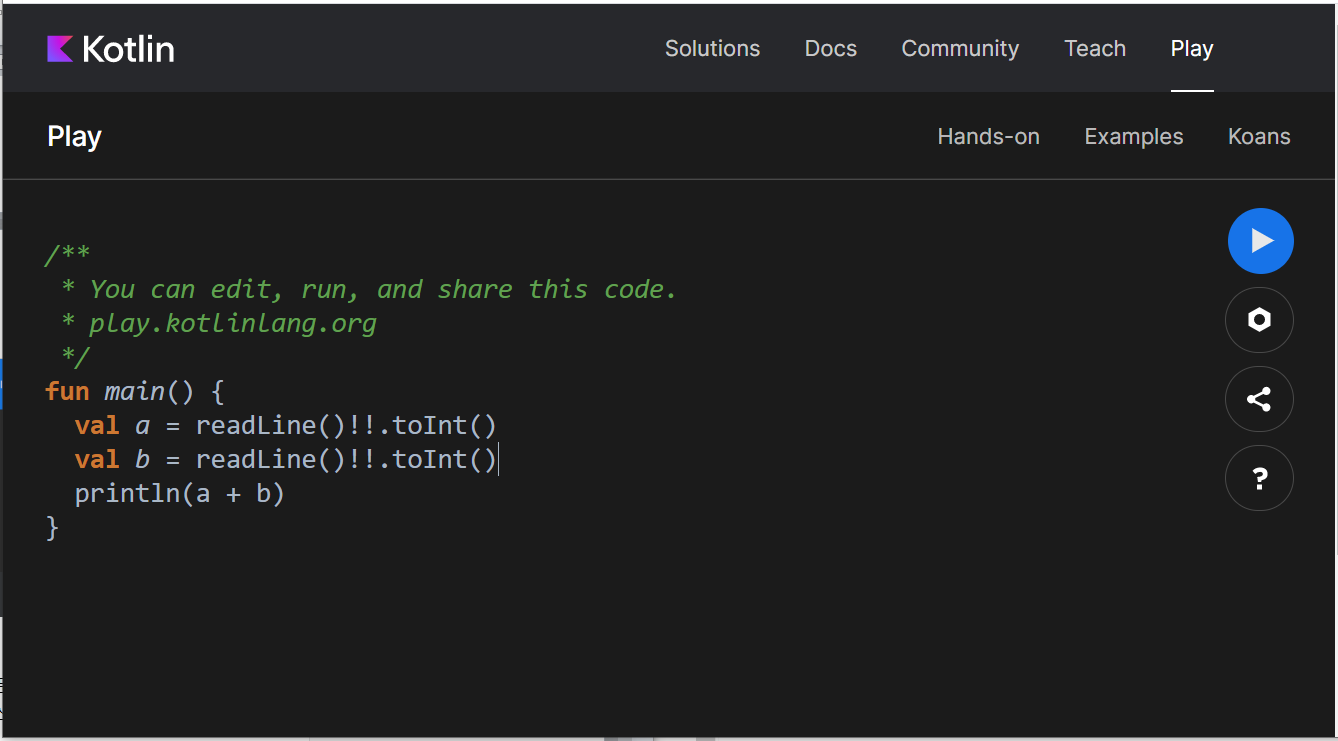
REPL은 식을 계산한 결과인 7을 표시한다. 조금 전의 println 예제의 결과와 달리 여기서는 결과 앞에 =이 표시됐음에 유의하라. 이 =는 표시된 7이 여러분이 입력한 코드의 실제 결과임을 나타낸다. 여러분이 이 도구에 익숙해지길 바란다. 앞으로 이 책에서 (그리고 여러분이 이 책을 다 읽은 후 실제 개발을 할 때도) 필요한 기능을 실험할 때 REPL을 계속 사용할 것이기 때문이다.

### 코틀린 놀이터

인텔리J에는 REPL 셸 외에도 REPL과 비슷하지만 REPL과 완전한 IDE 중간에 위치하는 더 강력한 온라인 도구가 있다. 이 도구는 코틀린 놀이터(playground)이다. 브라우저에서 https://play.kotlinlang.org에 방문하여 코틀린 놀이터를 사용해보라(그림 1-7).

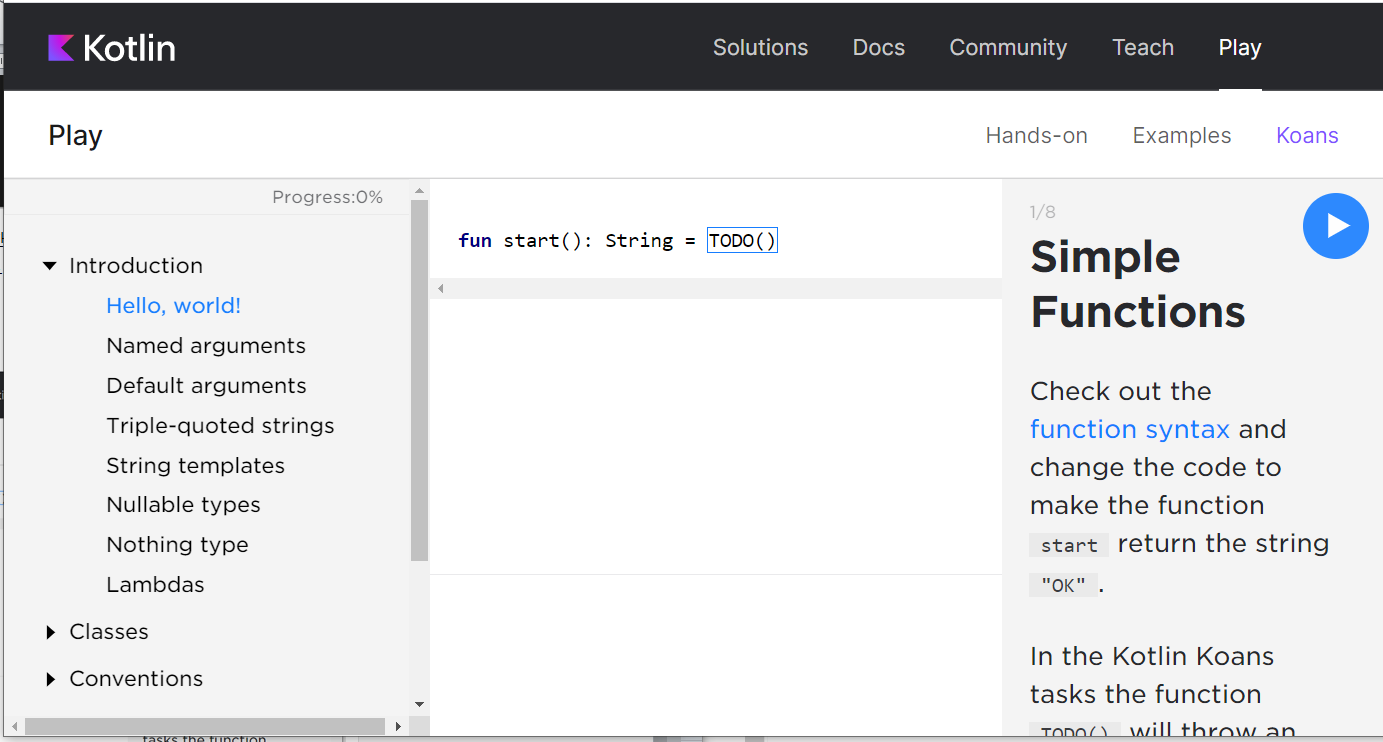
코틀린 놀이터는 기본적으로 실제 IDE를 사용하지 않고도 코틀린 언어를 탐험할 수 있는 온라인 환경이다. 코드 편집기, 문법과 오류 하이라이팅, 코드 완성과 콘솔 프로그램 실행기 등 여러 가지 똑똑한 기능을 제공한다.

그림 1-7: 코틀린 놀이터



놀이터는 개발자가 주요 코틀린 기능에 익숙해지기 위한 연습문제와 예제도 제공한다. 코틀린 선문답(koans)이라고 부르는 연습문제는 실패한 테스트 케이스 형태로 되어 있어 코드를 수정해야 통과할 수 있다(그림 1-8).

그림 1-8: 코틀린 선문답

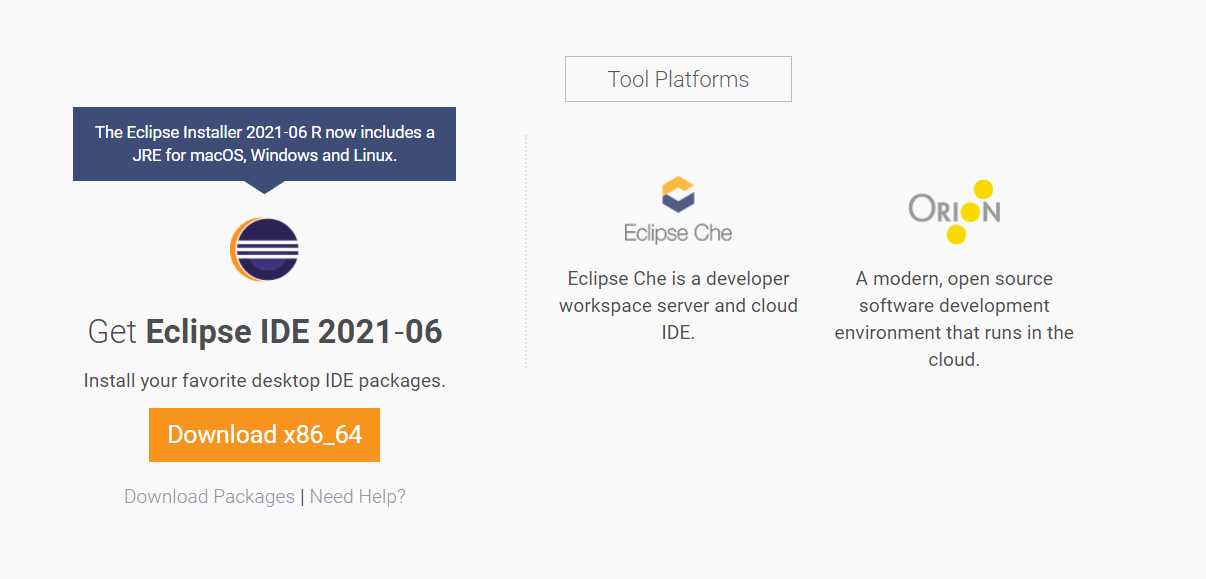


이러한 연습문제는 이 책을 보완해주는 귀중한 문제들이다. 여러분도 꼭 한번씩 풀어보기 바란다.

## 이클립스 프로젝트 설정하기

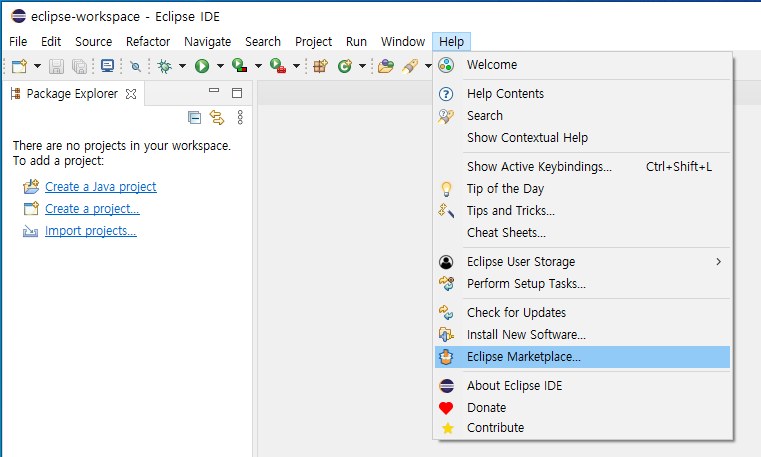
코틀린 프로그래밍 도구가 인텔리J만 있는 것은 아니다. 이클립스(Eclipse) IDE를 더 좋아하는 개발자는 이클립스 플러그인을 사용해 이클립스에서 코틀린 프로그램을 개발할 수 있다. 인텔리J만큼 광범위하게 코틀린 언어를 지원하지는 않지만, 이클립스도 개발자에게 도움이 되는 코드 지원 기능들을 제공한다. 예를 들어 코드 하이라이트, 코드 자동완성, 프로그램 실행과 디버깅, 기본적인 리팩토링 등이다. 이클립스가 없는 독자는 http://www.eclipse.org/downloads에서 무료로 이클립스를 내려받을 수 있다. 설치 프로그램을 실행한 후, "Eclipse IDE for Java Developers"(또는 "Enterprise Java Developers")를 선택하고 프로그램의 지시에 따라 설치하라. 이 책에서는 2021년에 나온 이클립스 2021-06버전을 사용한다.

<추가 그림>



인텔리J IDEA와 달리 이클립스는 기본적으로 코틀린을 지원하지 않는다. 따라서 코틀린 코드를 작성하기 전에 이클립스 마켓플레이스에서 플러그인을 설치해야 한다. Help > Eclipse Marketplace... 를 선택하고 Kotlin plugin을 검색하라(그림 1-9).

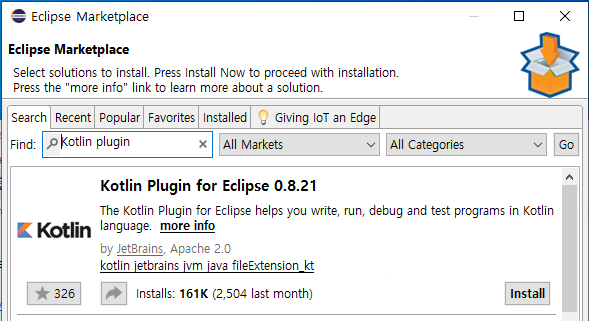
<추가 그림>



Install 버튼을 클릭하면 IDE가 플러그인을 다운로드해 설치한다. 라이선스 계약에 동의하고 이클립스를 재시작하면 설치가 완료된다.

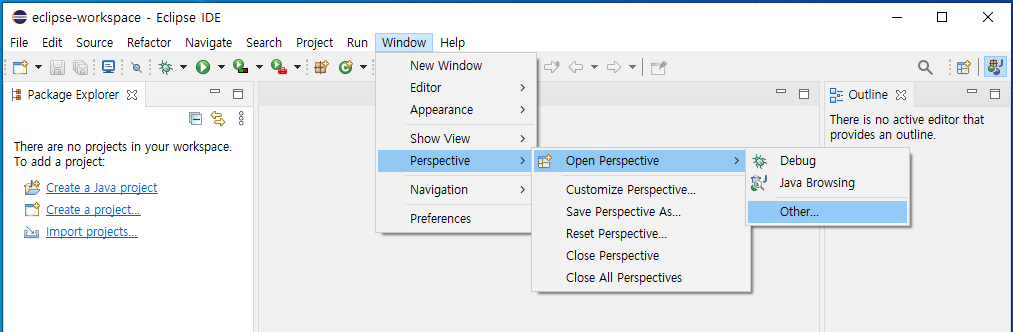
그림 1-9: 이클립스 마켓플레이스에서 코틀린 플러그인 설치하기

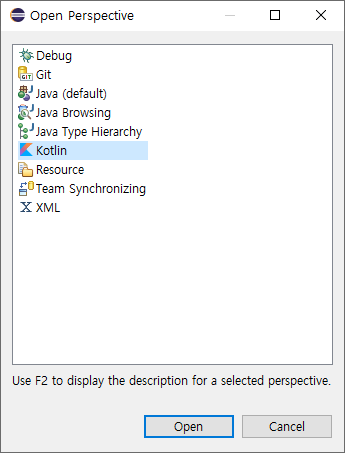
<추가 그림>



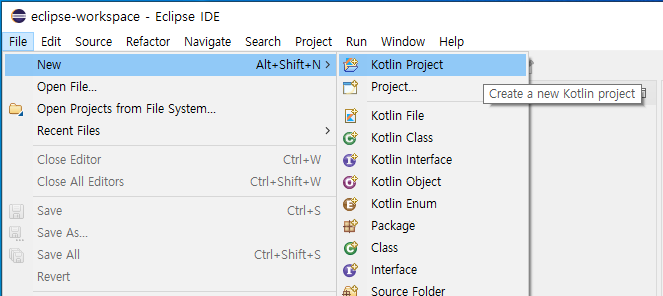
이제 프로젝트를 설정할 수 있다. 우선 Window > Perspective > Open Perspective > Other…를 선택하면 나오는 대화창에서 코틀린을 선택해 IDE를 코틀린 퍼스펙티브(perspective)로 바꾸자.

<추가 그림>

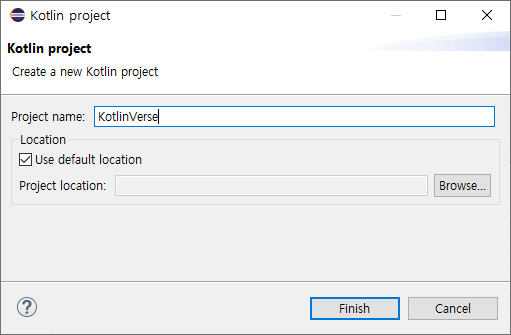




이 퍼스펙티브를 사용하면 코틀린 관련 액션을 애플리케이션 메뉴에서 직접 접근할 수 있다. 이제 프로젝트를 만들자. File > New > Kotlin Project…를 선택하고, 새 프로젝트의 이름을 KotlinVerse로 지정한 다음에 Finish를 클릭하자(그림 1-10).

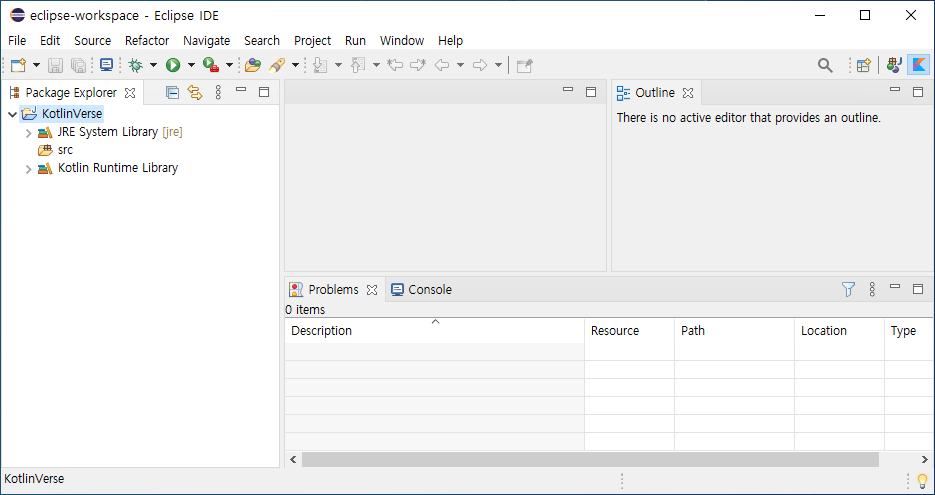


<추가그림>

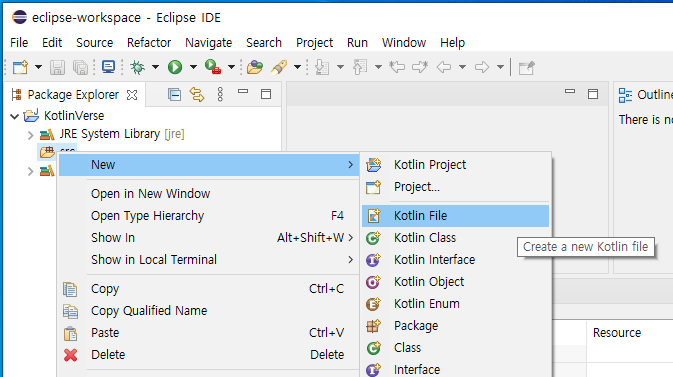


거의 다 됐다! 패키지 익스플로러(package explorer) 뷰에서 KotlinVerse 노드를 펼치면 새로 만든 프로젝트의 구성 요소를 볼 수 있다. 새 프로젝트에는 자바 런타임 환경(JRE) 라이브러리, 코틀린 표준 라이브러리, 소스코드가 들어갈 빈 src 디렉터리가 있다.

<추가그림>



이제 첫 번째 코틀린 파일을 생성하자. src 디렉터리에서 오른쪽 클릭을 하고 New > Kotlin File을 선택하라. 파일 이름을 입력한 후, Finish를 클릭하라(그림 1-11).



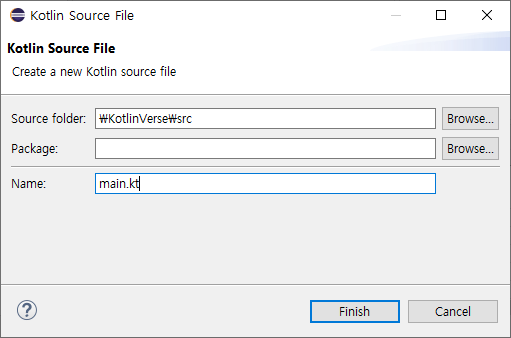
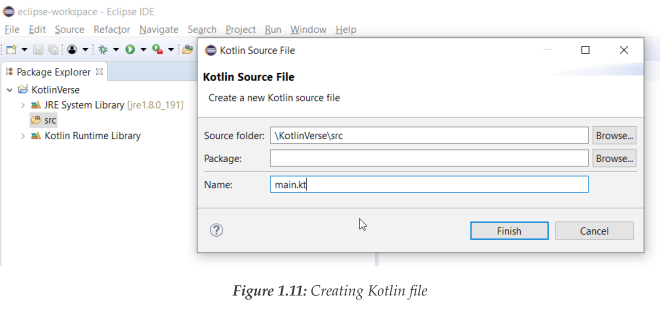
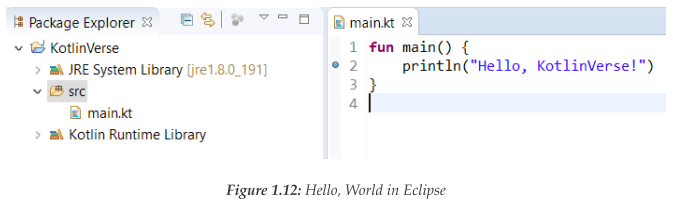


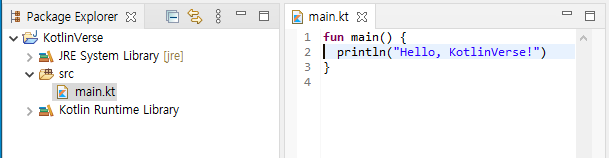
그림 1-11: 코틀린 파일 생성하기



이클립스가 자동으로 새 파일을 편집창에서 열어준다. 앞에서 예제로 사용한 "Hello, World" 프로그램을 입력하자(그림 1-12).

그림 1-12: 이클립스 "Hello, World" 프로그램





다 됐다! 프로그램을 실행하려면 Run > Run 명령을 사용한다. 이클립스가 여러분의 파일을 JVM 바이트코드로 컴파일하고 컴파일한 프로그램을 실행한 다음, 콘솔 뷰에 출력을 표시해줄 것이다.

## 결론

이번 장에서는 안전성, 간결성, 함수형 및 객체지향 프로그래밍 패러다임 지원 등 코틀린 언어의 여러 측면을 살펴봤다. 코틀린이 JVM, 안드로이드, 자바스크립트, 네이티브 애플리케이션 등 다양한 플랫폼을 지원한다는 점도 봤다. 자바나 다른 구체적인 플랫폼 코드 사이의 상호운용성을 위해 잘 설계된 언어이며 광범위한 도구, 라이브러리, 프레임워크를 제공하는 생태계가 있고, 빠르게 성장중인 커뮤니티가 있어서 코틀린은 배워볼 만한 훌륭한 언어다. 그리고 코틀린 프로그래밍을 시작할 때 사용할 수 있는 인텔리J IDE, 이클립스 IDE, 코틀린 놀이터 등의 도구에 대해 살펴봤다. 이제 앞으로 나갈 준비가 됐다. 다음 장에서는 변수, 식 등의 기본적인 구문 구조를 샅샅이 살펴보고, 기본 코틀린 타입을 알아본다.

1. 옮긴이 - 영문 원서에서는 src아래 코틀린 소스코드가 들어있는 간단한 디렉터리 레이아웃이 쓰였으나, 최신 인텔리J에서는 좀 더 복잡한 디렉터리 레이아웃을 사용하는 프로젝트 코드를 생성해준다. 코틀린 소스 코드에 src/main/kotlin/패키지, 자원 파일에 src/main/resources/패키지, 자바 소스 코드에 src/main/java/패키지를 사용하고, 테스트쪽 파일들을 src/test밑의 상응하는 패키지(src/test/kotlin/패키지, src/test/resources/패키지, src/test/java/패키지)에 두는 방식은 메이븐(maven)의 표준 디렉터리 레이아웃(https://bit.ly/3DPeoTW)에 따른 방식이다. 자바 세계에서는 이 방식이 사실상 표준으로 자리잡았고, 코틀린이나 스칼라 등의 다른 언어도 이를 채택했다. 다만, 언어 이름만 java에서 kotlin이나 scala로 쓴 디렉터리를 사용하며, 인텔리J를 통해 구성한 프로젝트도 이 디렉터리 레이아웃을 따른다. 프로젝트가 복잡해지거나, 자바 코드를 읽거나 자바와 코틀린을 함께 쓰거나, 그레이들, 메이븐 등의 빌드 도구를 사용할 경우 어차피 이런 디렉터리 구조를 사용하게 되니 빨리 익숙해지는게 좋다.

   한편 코틀린 컴파일러는 디렉터리 경로와 관계 없이 소스 코드 컴파일을 제대로 수행할 수 있다. 특별한 빌드 도구나 IDE를 쓰지 않고 텍스트 에디터를 활용해 명령줄에서 코틀린 소스를 컴파일하고 실행하고 싶을 때 가장 간단한 방법은, 현재 디렉터리에 코틀린 파일을 만들고(예: 현재 디렉터리가 /home/enshahar/main일 때, /home/enshahar/kotlin/main.kt), "kotlinc 파일이름.kt -include-runtime -d 파일이름.jar"으로 컴파일한 후(파일 확장자 .kt와 .jar를 반드시 포함시켜야 함), "java -jar 파일이름.jar"로 실행하는 방법이다. https://kotlinlang.org/docs/command-line.html를 참조하라. [↑](#footnote-ref-1)