جزوه درس برنامه نویسی موبایل ۲

برنامه نویسی با کاتلین

(Kotlin)

فهرست مطالب

Contents

معرفی کاتلین

در سال ۲۰۱۱، شرکت JetBrains توسعه زبان برنامه‌نویسی کاتلین را به عنوان جایگزینی برای نوشتن کد در زبان‌هایی مانند Java یا Scala جهت اجرا بر روی ماشین مجازی جاوا (Java Virtual Machine) اعلام کرد. شش سال بعد، Google اعلام کرد که کاتلین به یک مسیر توسعه با پشتیبانی رسمی برای سیستم‌عامل اندروید تبدیل خواهد شد.

دامنه کاربرد کاتلین به سرعت از زبانی با آینده‌ای روشن، به زبانی که برنامه‌های کاربردی را در برترین سیستم‌عامل موبایل جهان قدرت می‌بخشد، گسترش یافت. امروزه، شرکت‌های بزرگی مانند Google، Uber، Netflix، Capital One، Amazon و غیره، کاتلین را به خاطر مزایای فراوانش از جمله نحو (syntax) مختصر، ویژگی‌های مدرن و تعامل‌پذیری یکپارچه با کدهای قدیمی جاوا پذیرفته‌اند.

چرا کاتلین؟

برای درک جذابیت کاتلین، ابتدا باید نقش جاوا را در چشم‌انداز توسعه نرم‌افزار مدرن درک کنید. این دو زبان ارتباط تنگاتنگی با یکدیگر دارند، زیرا کد کاتلین در اغلب موارد برای ماشین مجازی جاوا نوشته می‌شود.

جاوا یک زبان قدرتمند و آزمون‌پس‌داده است و سال‌هاست که یکی از رایج‌ترین زبان‌های نوشته‌شده در پایگاه‌های کدِ محیط تولید (production codebases) به شمار می‌رود. با این حال، از زمان انتشار جاوا در سال ۱۹۹۵، دانش زیادی در مورد ویژگی‌های یک زبان برنامه‌نویسی خوب به دست آمده است. جاوا فاقد بسیاری از پیشرفت‌هایی است که توسعه‌دهندگان شاغل با زبان‌های مدرن‌تر از آن‌ها بهره‌مند هستند.

کاتلین از تجربیاتِ به‌دست‌آمده بهره می‌برد، زیرا برخی از تصمیمات طراحی اتخاذشده در جاوا (و زبان‌های دیگر مانند Scala) با گذشت زمان کارایی خود را از دست داده‌اند. این زبان فراتر از آنچه با زبان‌های قدیمی‌تر امکان‌پذیر بود تکامل یافته و مشکلات آزاردهنده آن‌ها را برطرف کرده است. در فصل‌های آینده در مورد اینکه چگونه کاتلین فراتر از جاوا عمل می‌کند و تجربه توسعه قابل‌اطمینان‌تری را ارائه می‌دهد، بیشتر خواهید آموخت.

علاوه بر این، کاتلین صرفاً یک زبان بهتر برای نوشتن کد جهت اجرا بر روی ماشین مجازی جاوا نیست. این یک زبان چندپلتفرمی (multiplatform) است که هدف آن کاربرد همه‌منظوره (general purpose) است: از کاتلین می‌توان برای نوشتن برنامه‌های بومی (native) برای macOS، iOS و Windows؛ برنامه‌های JavaScript؛ و البته برنامه‌های اندروید استفاده کرد. اخیراً، JetBrains در حال سرمایه‌گذاری بر روی این قابلیت‌های چندپلتفرمی (cross-platform) بوده است؛ قابلیت چندپلتفرمی کاتلین (Kotlin Multiplatform) روشی منحصربه‌فرد برای اشتراک‌گذاری کد بین برنامه‌های مختلف ارائه می‌دهد و منجر به افزایش استفاده از کاتلین فراتر از ماشین مجازی جاوا شده است.

این کتاب برای چه کسانی است؟

ما این کتاب را برای انواع توسعه‌دهندگان نوشته‌ایم: توسعه‌دهندگان باتجربه اندروید که خواهان ویژگی‌های مدرنی فراتر از امکانات جاوا هستند، توسعه‌دهندگان سمت سرور که علاقه‌مند به یادگیری ویژگی‌های کاتلین می‌باشند، توسعه‌دهندگانی که به دنبال اشتراک‌گذاری کد کاتلین بین برنامه‌های بومی یا تحت وب خود هستند، و توسعه‌دهندگان تازه‌کاری که می‌خواهند وارد دنیای یک زبان کامپایل‌شونده (compiled) با عملکرد بالا شوند.

پشتیبانی از اندروید ممکن است دلیل مطالعه این کتاب توسط شما باشد، اما این کتاب محدود به برنامه‌نویسی کاتلین برای اندروید نیست. در واقع، تمام کدهای کاتلین در این کتاب مستقل از چارچوب (framework) اندروید هستند. با این وجود، اگر علاقه‌مند به استفاده از کاتلین برای توسعه برنامه‌های اندروید هستید، این کتاب الگوهای رایجی را به نمایش می‌گذارد که نوشتن برنامه‌های اندروید با کاتلین را بسیار آسان می‌سازد.

اگرچه کاتلین تحت تأثیر تعدادی از زبان‌های دیگر قرار گرفته است، اما برای یادگیری کاتلین نیازی به دانستن جزئیات هیچ زبان دیگری ندارید. هر از گاهی، ما در مورد معادل کد جاوای قطعه کدهایی که به زبان کاتلین نوشته‌اید بحث خواهیم کرد. همچنین در صورت لزوم به شباهت‌هایی با زبان‌های دیگر اشاره خواهیم نمود. اگر در این زبان‌ها تجربه دارید، این امر به شما کمک می‌کند تا رابطه بین کاتلین و پلتفرم‌هایی که از آن‌ها پشتیبانی می‌کند را درک کنید. حتی اگر این مقایسه‌ها برای شما کمتر آشنا باشند، مشاهده اینکه چگونه یک زبان دیگر با مشکلات مشابه برخورد می‌کند، می‌تواند به شما در درک اصولی که توسعه کاتلین را شکل داده‌اند کمک کند.

نحوه استفاده از این کتاب

این کتاب یک راهنمای مرجع نیست. هدف ما راهنمایی شما در مهم‌ترین بخش‌های زبان برنامه‌نویسی کاتلین است. شما روی پروژه‌های نمونه کار خواهید کرد و با پیشرفت خود، دانشتان را افزایش خواهید داد. برای بهره‌گیری حداکثری از این کتاب، توصیه می‌کنیم در حین خواندن، مثال‌های کتاب را خودتان تایپ کنید. کار کردن روی پروژه‌ها به ایجاد حافظه عضلانی کمک کرده و اندوخته‌ای برای انتقال از یک فصل به فصل دیگر به شما می‌دهد.

همچنین، هر فصل بر مبنای مباحث ارائه‌شده در فصل قبل بنا شده است، بنابراین توصیه می‌کنیم مطالب را پرش نکنید. حتی اگر احساس می‌کنید با یک موضوع در زبان‌های دیگر آشنا هستید، پیشنهاد می‌کنیم آن را به طور کامل مطالعه کنید - کاتلین بسیاری از مشکلات را به روش‌های منحصربه‌فردی مدیریت می‌کند. شما با مباحث مقدماتی مانند متغیرها و روند کنترل (control flow) شروع خواهید کرد، مسیر خود را از طریق تکنیک‌های برنامه‌نویسی شیءگرا و تابعی ادامه می‌دهید، رویکرد اختصاصی (first-party) کاتلین برای اجرای کدهای ناهمگام (asynchronous) را امتحان کرده و به قابلیت‌های چندپلتفرمی کاتلین ورود خواهید کرد. در پایان کتاب، دانش خود را در مورد کاتلین از سطح یک فرد مبتدی به یک توسعه‌دهنده پیشرفته‌تر ارتقا خواهید داد.

با این وجود، عجله نکنید: موضوعات را بسط دهید، از مرجع کاتلین در kotlinlang.org/docs/reference برای پیگیری هر چیزی که کنجکاوی شما را برانگیخته است استفاده کنید و به آزمایش و تمرین بپردازید.

برای افراد کنجکاوتر

بیشتر فصل‌های این کتاب دارای یک یا دو بخش با عنوان «برای افراد کنجکاوتر» هستند. بسیاری از این بخش‌ها مکانیسم‌های زیربنایی زبان کاتلین را روشن می‌سازند. مثال‌های موجود در فصل‌ها به اطلاعات این بخش‌ها وابسته نیستند، اما اطلاعات اضافی و مکملی را فراهم می‌کنند که ممکن است برایتان جالب یا مفید باشد.

چالش‌ها

بسیاری از فصل‌ها با یک یا چند چالش به پایان می‌رسند. این‌ها مسائل اضافه‌ای برای حل کردن هستند که جهت تعمیق درک شما از کاتلین طراحی شده‌اند. ما شما را تشویق می‌کنیم که برای افزایش تسلط خود بر کاتلین، آن‌ها را امتحان کنید.

توصیه می‌کنیم قبل از کار روی یک چالش، از پروژه‌های خود یک کپی بگیرید؛ فصل‌های دیگر اغلب بر پایه راه‌حل‌های قبلی ساخته می‌شوند و شما نمی‌خواهید تغییراتتان در ادامه مسیر مانعی ایجاد کنند. همچنین می‌توانید راه‌حل‌های تمرین‌های این کتاب را از آدرس bignerdranch.com/kotlin-2e-solutions/ دانلود کنید.

https://bignerdranch.com/kotlin-2e-solutions/

قراردادهای تایپوگرافی

در حین ساخت پروژه‌های این کتاب، ما شما را با معرفی یک موضوع و سپس نشان دادن نحوه به‌کارگیری دانش جدیدتان راهنمایی خواهیم کرد. برای وضوح بیشتر، ما به قراردادهای تایپوگرافی زیر پایبند هستیم.

متغیرها، مقادیر و نوع‌ها (types) با فونت با عرض ثابت (fixed-width font) نشان داده می‌شوند. نام کلاس‌ها، توابع و رابط‌ها (interfaces) با فونت پررنگ نشان داده شده‌اند.

تمام قطعه کدها با فونت با عرض ثابت نشان داده شده‌اند. اگر قرار است کدی را در یک قطعه کد تایپ کنید، آن کد به صورت پررنگ مشخص شده است. اگر قرار است کدی را در یک قطعه کد حذف کنید، روی آن کد خط کشیده شده است. در مثال زیر، به شما دستور داده می‌شود خطی که متغیر y را تعریف می‌کند حذف کرده و متغیری به نام z اضافه کنید:

var × =“Python” var y = “Java” var z =“Kotlin”

کاتلین زبانی در حال بلوغ است و قراردادهای کدنویسی آن با گذشت زمان همچنان در حال تکامل هستند. احتمالاً شما سبک خاص خود را توسعه خواهید داد، اما ما تمایل داریم به راهنماهای سبک کاتلینِ متعلق به JetBrains و Google پایبند باشیم:

قراردادهای کدنویسی JetBrains: kotlinlang.org/docs/coding-conventions.html

راهنمای سبک Google: developer.android.com/kotlin/style-guide

استفاده از کتاب الکترونیکی

اگر این کتاب را روی یک کتابخوان الکترونیکی (eReader) می‌خوانید، باید اشـــاره کنیم که خواندن کدها ممکن اســـت گاهی دشوار باشد. خطوط طولانی‌ترِ کد ممکن است بسته به اندازه فونت انتخابی شما به خط دوم شکسته شوند (wrap شوند).

طولانی‌ترین خطوط کد در این کتاب ۸۶ کاراکتر با عرض ثابت (monospace) هستند، مانند مورد زیر.

```
println(playerCreateMessage(nameIsLong("Polarcubis, the Supreme Master of NyetHack")))
```

می‌توانید تنظیمات کتابخوان الکترونیکی خود را تغییر دهید تا بهترین حالت را برای مشاهده خطوط طولانی کد پیدا کنید.

اگر در حال مطالعه روی یک iPad با برنامه Books اپل هستید، توصیه می‌کنیم به برنامه Settings بروید، Books را انتخاب کنید و تنظیمات Full Justification و Auto-hyphenation را در وضعیت OFF (خاموش) قرار دهید.

وقتی به مرحله‌ای رسیدید که در واقع در حال تایپ کردن کد هستید، پیشنهاد می‌کنیم کتاب را روی رایانه شخصی (PC) یا Mac خود در نرم‌افزار Adobe Digital Editions باز کنید. (نرم‌افزار Adobe Digital Editions یک برنامه کتابخوان الکترونیکی رایگان اســـت که می‌توانید از www.adobe.com/products/digitaleditions دانلود کنید). پنجره برنامه را به اندازه‌ای بزرگ کنید که بتوانید کدها را بدون شکسته شدن خطوط مشاهده کنید. همچنین قادر خواهید بود اشکال را با جزئیات کامل ببینید.

نگاه به آینده

برای کار با مثال‌های این کتاب وقت بگذارید. زمانی که با نحو (syntax) کاتلین آشـــنا شـــدید، فکر می‌کنیم فرآیند توســـعه را واضح، عمل‌گرایانه و روان خواهید یافت. تا آن زمان، به تلاش خود ادامه دهید؛ یادگیری یک زبان جدید می‌تواند بسیار نتیجه‌بخش و لذت‌بخش باشد.

بخش اول: شروع کار

دو فصل اول این کتاب شما را با اصول اولیه استفاده از IntelliJ IDEA، محیط توسعه یکپارچه (IDE) رسمی برای توسعه برنامه‌های کاربردی کاتلین، آشنا می‌کند. شما کار خود را با ایجاد یک پروژه ساده آغاز خواهید کرد تا با ویژگی‌های اساسی این زبان احساس راحتی کنید. برای شروع، به بررسی انواع داده (types) در کاتلین می‌پردازید که داده‌هایی را که در یک برنامه با آن‌ها کار می‌کنید دسته‌بندی می‌کنند.

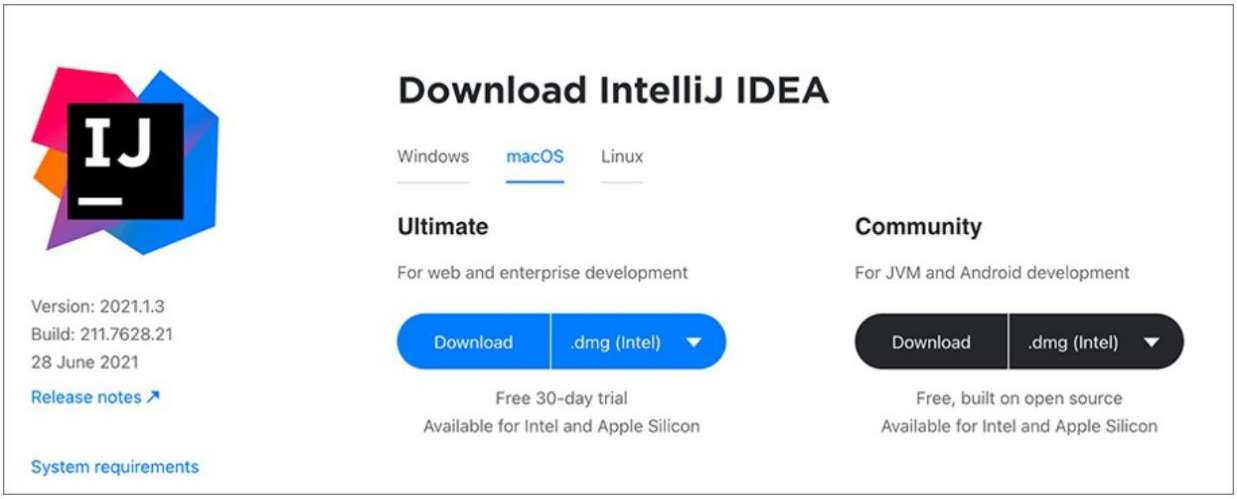
اولین برنامه کاتلین شما

در این فصل، اولین برنامه کاتلین خود را با استفاده از IntelliJ IDEA خواهید نوشت. در حین انجام این مرحله‌ی گذار برنامه‌نویسی، با محیط توسعه خود آشنا می‌شوید، یک پروژه جدید کاتلین ایجاد می‌کنید، کد کاتلین را نوشته و اجرا می‌کنید، و خروجی حاصل را بررسی می‌نمایید. پروژه‌ای که در این فصل ایجاد می‌کنید، به عنوان بستری آزمایشی برای ویژگی‌های بنیادی زبان که در سراسر برنامه‌های کاربردی کاتلین خود از آن‌ها استفاده خواهید کرد، عمل می‌کند.

نصب IntelliJ IDEA

نرم‌افزار IntelliJ IDEA یک محیط توسعه یکپارچه (IDE) برای کاتلین است که توسط شرکت JetBrains (که زبان کاتلین را نیز ایجاد کرده است) ساخته شده است. برای شروع، نسخه Community نرم‌افزار IntelliJ IDEA را از وب‌سایت JetBrains به آدرس jetbrains.com/idea/download دانلود کنید (شکل ۱.۱).

Figure 1.1 Downloading IntelliJ IDEA Community Edition



پس از دانلود، دستورالعمل‌های نصب مربوط به پلتفرم خود را که در صفحه نصب و راه‌اندازی JetBrains در آدرس jetbrains.com/help/idea/installationguide.html#standalone توضیح داده شده است، دنبال کنید.

IntelliJ IDEA که به اختصار IntelliJ نامیده می‌شود، به شما کمک می‌کند تا کدهای کاتلین را با ساختار مناسب بنویسید. این محیط همچنین فرآیند توسعه را با ابزارهای داخلی برای اجرا، اشکال‌زدایی (debugging)، بازرسی و بازآرایی (refactoring) کد شما ساده می‌کند. می‌توانید در بخش «برای افراد کنجکاوتر: چرا از IntelliJ استفاده کنیم؟» در اواخر این فصل، درباره دلایل توصیه ما برای استفاده از IntelliJ جهت نوشتن کد کاتلین بیشتر بخوانید.

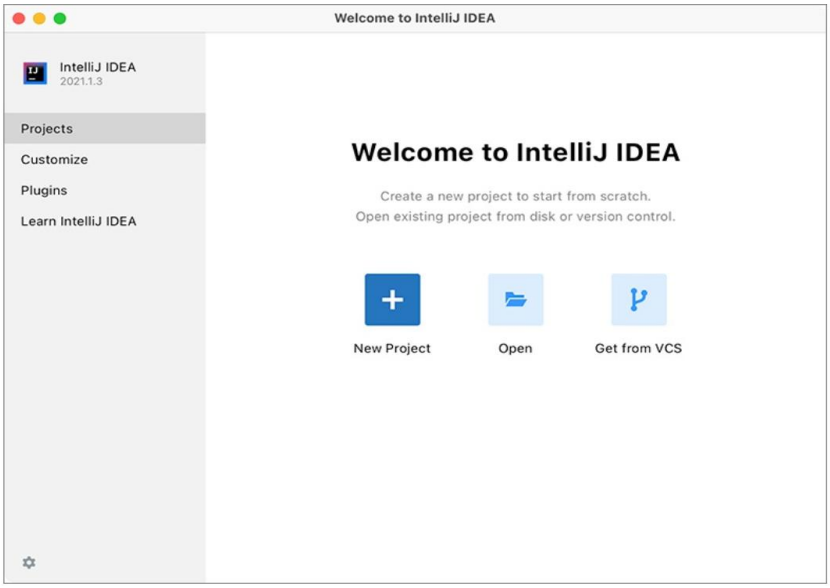
اولین پروژه کاتلین شما

تبریک می‌گوییم، اکنون زبان برنامه‌نویسی کاتلین و یک محیط توسعه قدرتمند برای نوشتن آن را در اختیار دارید. حالا تنها یک کار باقی مانده است: یاد بگیرید به روانی به زبان کاتلین صحبت کنید. اولین قدم - ایجاد یک پروژه کاتلین است.

در بیشتر بخش‌های این کتاب، پروژه‌هایی که روی آن‌ها کار می‌کنید بر روی یک دنیای بازی فانتزی تمرکز دارند که در آن یک قهرمان وارد مأموریت‌های قهرمانانه می‌شود، شیاطین شرور را شکست می‌دهد، شهرها را از خطرات حتمی نجات می‌دهد و به طور کلی کارهایی را انجام می‌دهد که قهرمانان انجام می‌دهند. اولین پروژه شما یک «تخته پاداش» (bounty board) می‌سازد، یک سیستم مأموریت که قهرمان داستان به نام Madrigal را به سمت وظایفی هدایت می‌کند که به توجه او نیاز دارند. شما تا فصل ۷ روی این پروژه کار خواهید کرد.

IntelliJ را باز کنید. پنجره خوش‌آمدگویی IntelliJ IDEA به شما نمایش داده خواهد شد (شکل ۱.۲).

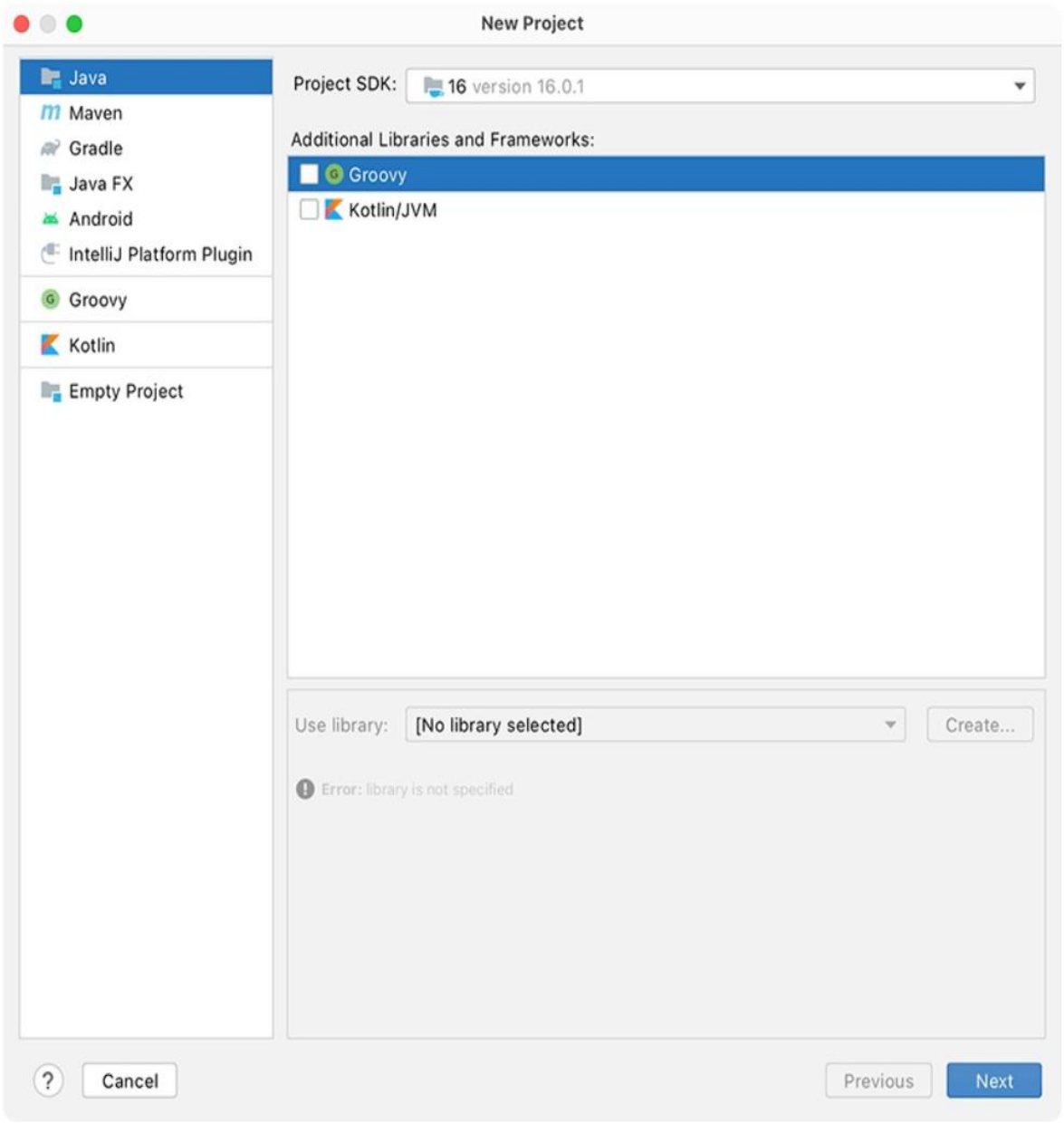
Figure 1.2 Welcome dialog



(اگر این اولین باری نیست که IntelliJ را پس از نصب باز می‌کنید، ممکن است مستقیماً به آخرین پروژه‌ای که باز کرده بودید هدایت شوید. برای بازگشت به صفحه خوش‌آمدگویی، پروژه را با استفاده از File سپس Close Project ببندید.)

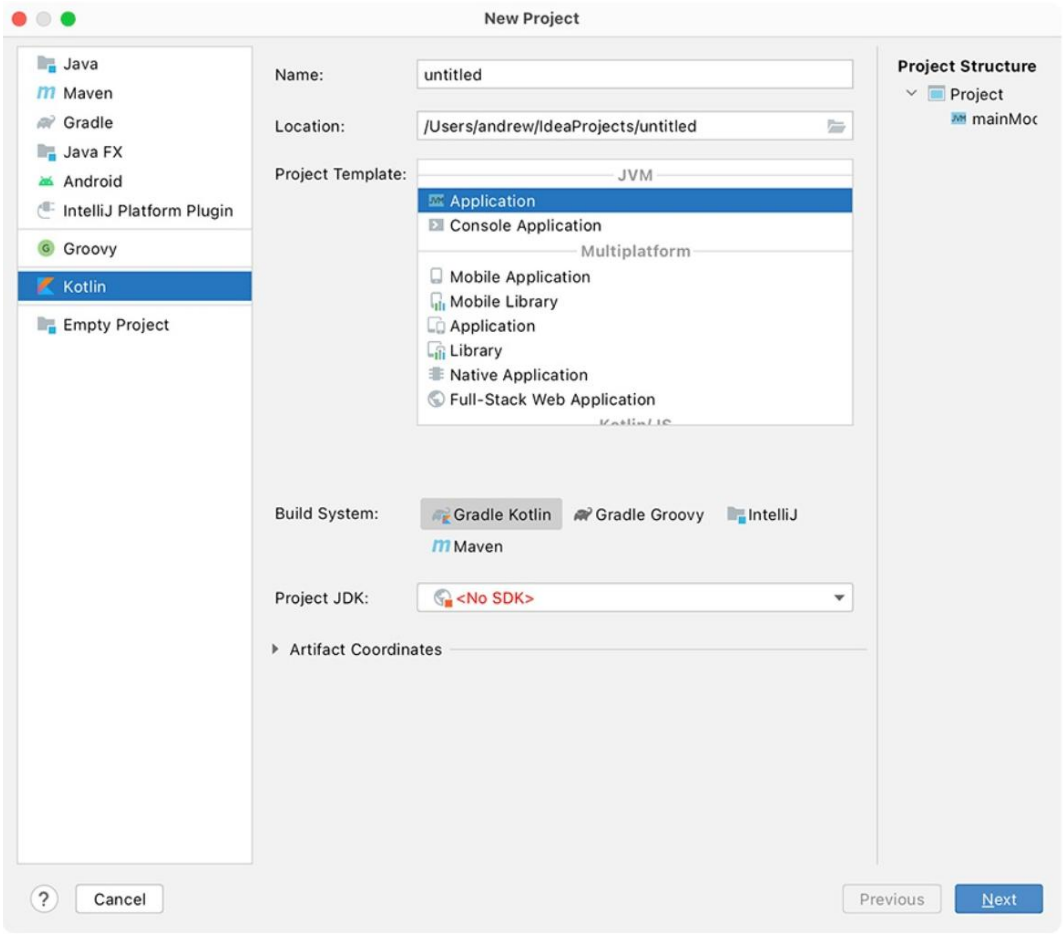
روی New Project کلیک کنید. IntelliJ پنجره New Project را نمایش می‌دهد، همانطور که در شکل ۱.۳ نشان داده شده است.

Figure 1.3 New Project dialog



در پنجره New Project، گزینه Kotlin را در سمت چپ انتخاب کنید، همانطور که در شکل ۱.۴ نشان داده شده است.

Figure 1.4 Creating a Kotlin project



شـما می‌توانید به طور پیش‌فرض از IntelliJ برای نوشـتن کد به زبان‌هایی غیر از کاتلین، از جمله Java و Groovy اسـتفاده کنید. با اسـتفاده از افزونه‌های (plugins) اضافی، IntelliJ می‌تواند برای نوشـتن کد به زبان‌هایی مانند Python، Scala، Dart و Rust نیز مورد اسـتفاده قرار گیرد. انتخاب Kotlin از بین گزینه‌های زبان در سـمت چپ پنجره New Project، به IntelliJ می‌گوید که شما قصد دارید از کاتلین استفاده کنید.

اکنون نگاهی به تنظیمات پروژه در بخش مرکزی پنجره بیندازید.

در بالای پنجره New Project، برای Name مقدار bounty-board را وارد کنید. فیلد Location به طور خودکار پر می‌شود. می‌توانید مکان را همانطور که هست رها کنید یا با کلیک روی نماد پوشه در سمت راست فیلد، مکان جدیدی را انتخاب نمایید.

منوی Project Template گزینه‌های مختلفی را در زیر سه عنوان اصلی دارد: JVM، Multiplatform و Kotlin/JS (و همچنین یک بخش Experimental). گزینه Application را در زیر عنوان JVM انتخاب کنید. این کار به IntelliJ می‌گوید که قصد دارید کد کاتلینی بنویسید که هدف آن (تارگت)، اجرا بر روی ماشین مجازی جاوا (Java Virtual Machine) است.

کد کاتلین می‌تواند برای هر یک از سه هدفی که توسط این زبان پشتیبانی می‌شود، کامپایل شود: ماشین مجازی جاوا، پلتفرم‌های بومی x86 و ARM (که در بخش Multiplatform در لیست Project Template قرار دارند) و جاوا اسکریپت (با نام اختصاری JS). از آنجایی که کاتلین می‌تواند در هر یک از این سه هدف کامپایل شود، گاهی اوقات اصطلاحات Kotlin/JVM، Kotlin/Native و Kotlin/JS را مشاهده خواهید کرد که "نوع" کاتلین را بر اساس پلتفرم هدف مشخص می‌کنند.

Kotlin/JVM همان چیزی است که بیشتر مردم هنگام گفتن "کاتلین" به آن فکر می‌کنند. هر زمان که کدی می‌نویسید که ماشین مجازی جاوا را هدف قرار می‌دهد، از Kotlin/JVM استفاده می‌کنید. به همین ترتیب، زمانی که می‌خواهید کدهای جاوا اسکریپت را از کد کاتلین خود خروجی بگیرید، از Kotlin/JS استفاده خواهید کرد. و Kotlin/Native یک اصطلاح کلی برای تمام برنامه‌های کاتلین است که به کد ماشین بومی کامپایل می‌شوند. همانطور که از عنوان Multiplatform پیداست، می‌توانید از Kotlin/Native برای ساخت نرم‌افزار برای پلتفرم‌های بسیاری استفاده کنید، مانند کتابخانه‌های iOS، برنامه‌های دسکتاپ بومی و دستگاه‌های تعبیه‌شده (embedded devices) در صورت تمایل و بلندپروازی.

(از این پس، ما به ماشین مجازی جاوا با عنوان "JVM" اشاره خواهیم کرد، زیرا در جامعه توسعه‌دهندگان جاوا معمولاً به این نام خوانده می‌شود. در بخش "برای افراد کنجکاوتر: هدف‌گیری JVM" در اواخر این فصل، می‌توانید در مورد هدف قرار دادن JVM بیشتر بیاموزید.)

پلتفرم Kotlin/JVM بالغ‌ترین پلتفرم از بین این سه مورد است. این پلتفرم اولین موردی بود که توسط کاتلین پشتیبانی شد و همچنان یکی از رایج‌ترین پلتفرم‌ها برای توسعه کاتلین به شمار می‌رود. برای بخش عمده‌ای از این کتاب، شما از Kotlin/JVM استفاده خواهید کرد. دلایل متعددی برای این تصمیم وجود دارد:

پلتفرم JVM دستیابی به استقلال از پلتفرم را بسیار آسان‌تر می‌کند: در هر جایی که JVM اجرا شود، کد شما نیز می‌تواند اجرا گردد. شما می‌توانید یک باینری (binary) توزیع کنید که روی هر کامپیوتری، صرف نظر از معماری آن، کار کند. جاوا یک زبان بسیار بالغ است و رابط‌های برنامه‌نویسی کاربردی (API) فراوانی دارد که در سراسر این کتاب از آن‌ها استفاده خواهید کرد. این APIهای سطح بالا انجام وظایف خاصی را در مقایسه با APIهای سطح پایین موجود در پلتفرم‌های بومی بسیار آسان‌تر می‌کنند. مثال‌های این کتاب در داخل IDE شما اجرا می‌شوند، اما قابلیت اجرا از هر ترمینالی (terminal) را نیز دارند. این موضوع جاوا اسکریپت را به هدفی با مطلوبیت کمتر تبدیل می‌کند، زیرا در درجه اول در یک مرورگر اجرا می‌شود تا یک ترمینال.

کاتلین در سراسر این سه پلتفرمِ هدف رفتار بسیار مشابهی دارد، بنابراین دانش مربوط به Kotlin/JVM مستقیماً برای Kotlin/JS و Kotlin/Native نیز کاربرد دارد. همه APIها در تمام پلتفرم‌های هدف در دسترس نیستند، بنابراین ما در زمان استفاده از APIای که فقط روی JVM قابل استفاده است، به آن اشاره خواهیم کرد. در فصل‌های ۲۴، ۲۵ و ۲۶ درباره نحوه استفاده از کاتلین روی سایر پلتفرم‌ها بیشتر خواهید آموخت.

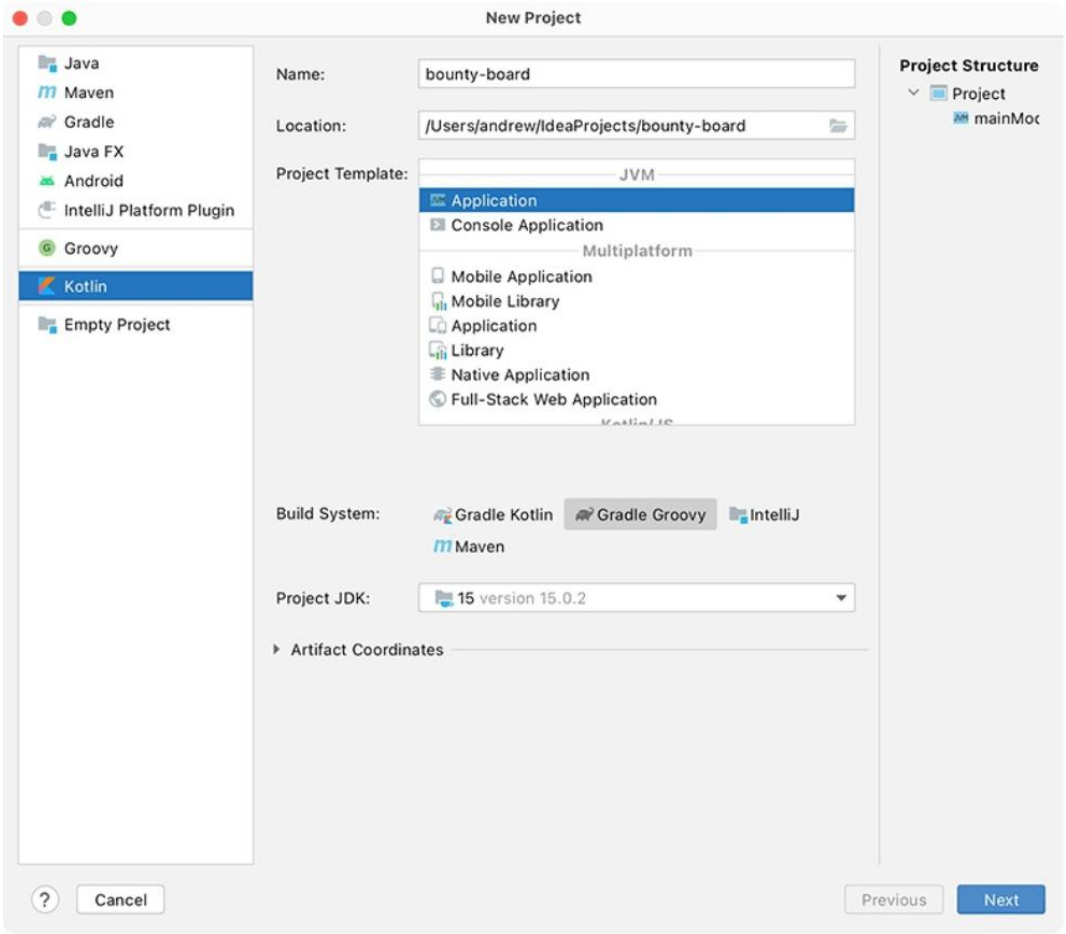
به راه‌اندازی پروژه جدیدتان بازگردیم. برای سیستم ساخت (Build System)، گزینه Gradle Groovy را انتخاب کنید. در نهایت، برای Project JDK، نسخه‌ای از جاوا را انتخاب نمایید تا پروژه شما به کیت توسعه جاوا (JDK) متصل شود. ما استفاده از نسخه‌ای بین جاوا ۸ (که اغلب با عنوان Java 1.8 لیست می‌شود) و جاوا ۱۵ را توصیه می‌کنیم.

اگر نسخه مناسبی از جاوا را در لیست کشویی مشاهده نمی‌کنید، IntelliJ نصبی را در کامپیوتر شما شناسایی نکرده است. اگر مطمئن هستید که یک JDK نصب کرده‌اید و می‌خواهید از آن نسخه استفاده کنید، می‌توانید از گزینه Add JDK… استفاده کرده و محل نصب را در دیسک پیدا کنید. در غیر این صورت، IntelliJ می‌تواند از طریق گزینه Download JDK… یکی را برای شما نصب کند. (توصیه می‌کنیم Version را روی ۱۵ و Vendor را روی AdoptOpenJDK (HotSpot) تنظیم کنید، اما هر ارائه‌دهنده‌ای (vendor) باید کار کند.) به محض اینکه IntelliJ نصب را به پایان رساند، آماده کار هستید.

چرا برای نوشتن یک برنامه کاتلین به JDK نیاز دارید؟ JDK به IntelliJ دسترسی به JVM و ابزارهای جاوا را می‌دهد که برای تبدیل کد کاتلین شما به بایت‌کد (bytecode) ضروری هستند (در این مورد کمی بعد بیشتر توضیح خواهیم داد). از نظر فنی، هر نسخه ۶ یا بالاتر کار خواهد کرد. اما تجربه ما، تا زمان نگارش این کتاب، نشان می‌دهد که JDK 8 یا نسخه‌های جدیدتر به روان‌ترین شکل ممکن کار می‌کنند.

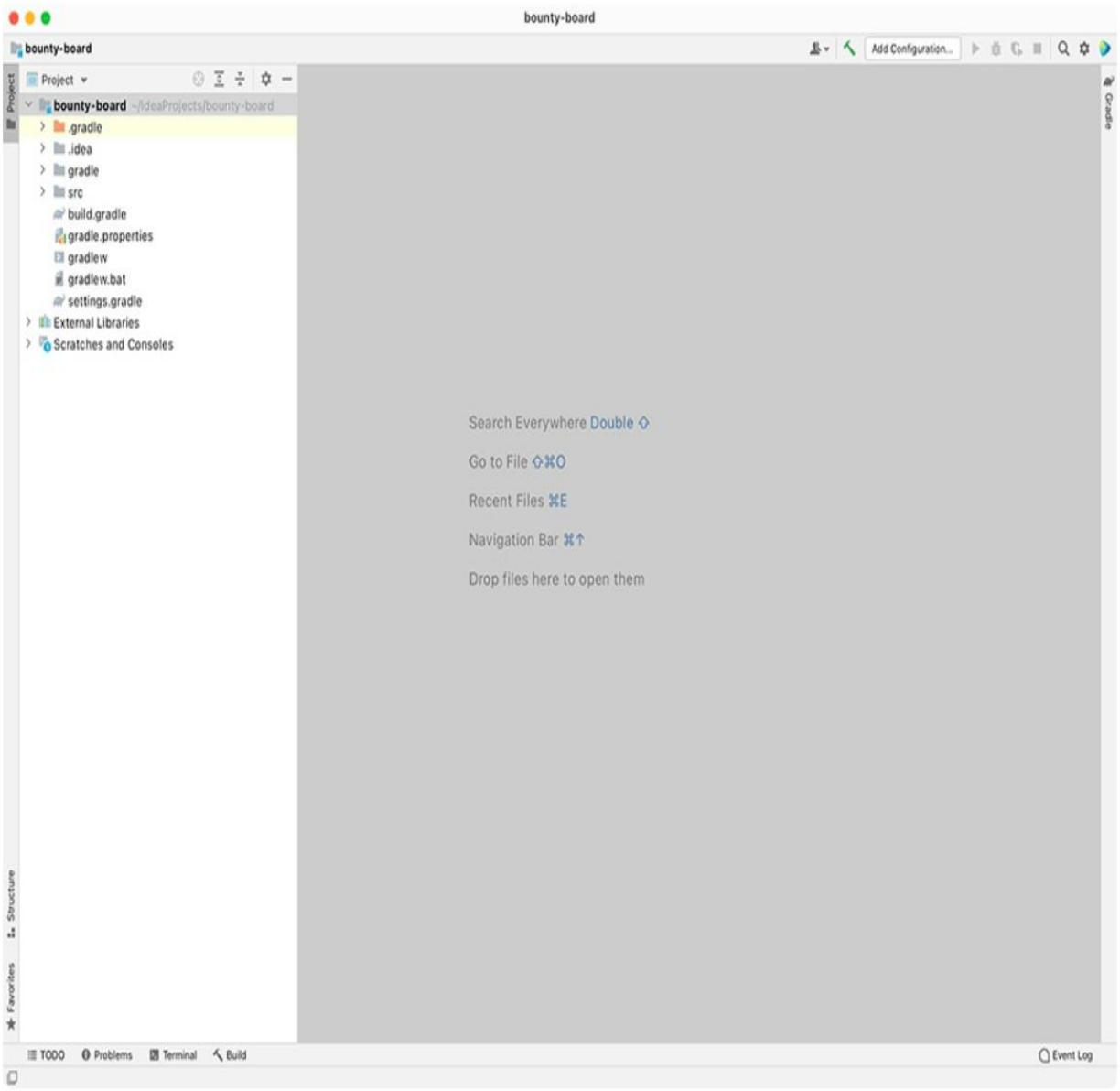
وقتی پنجره تنظیمات شما شبیه شکل ۱.۵ شد، روی Next و سپس روی Finish در پنجره بعدی کلیک کنید تا تنظیمات تأیید شوند.

Figure 1.5 Setting up a project



نرم‌افزار IntelliJ یک پروژه به نام bounty-board تولید کرده و پروژه جدید را در یک نمای پیش‌فرض با دو پنل (شکل 1.6) نمایش می‌دهد. روی دیسک، IntelliJ یک پوشه و مجموعه‌ای از زیرپوشه‌ها و فایل‌های پروژه را در مکانی که در فیلد Project location مشخص شده، ایجاد می‌کند.

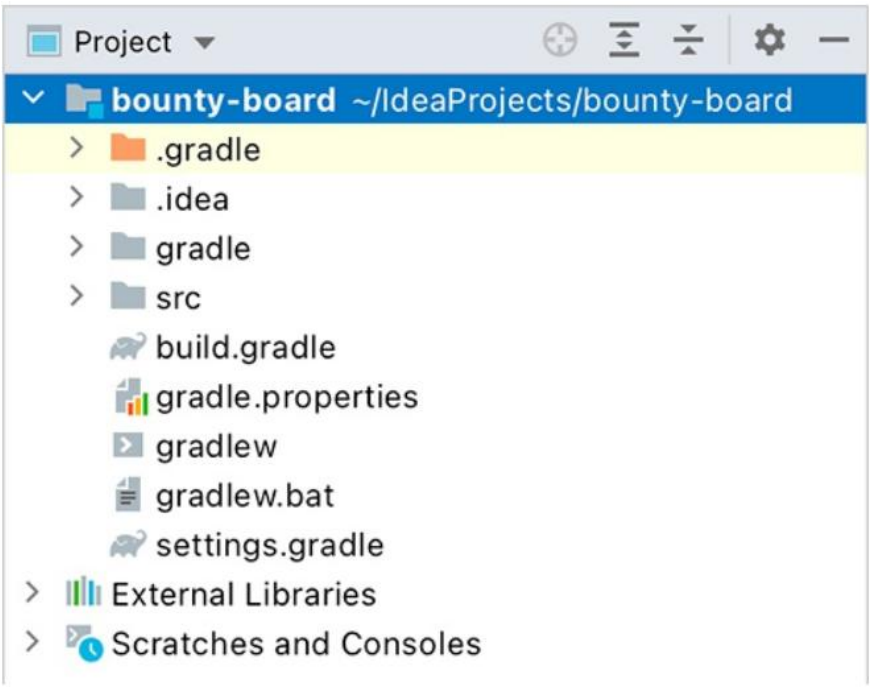
Figure 1.6 Default two-pane view



پنل سمت چپ پنجره ابزار پروژه (project tool window) را نشان می‌دهد. پنل سمت راست در حال حاضر خالی است. این قسمت جایی است که محتویات فایل‌های کاتلین خود را در ویرایشگر (editor) مشاهده و ویرایش خواهید کرد.

پنجره ابزار پروژه فایل‌های موجود در پروژه bounty-board را نمایش می‌دهد، همانطور که در شکل ۱.۷ مشاهده می‌کنید.

Figure 1.7 Project view



یک پروژه شـامل تمام کد منبع (source code) برنامه شـما، همراه با اطلاعات مربوط به وابسـتگی‌ها (dependencies) و پیکربندی‌هاسـت. یک پروژه می‌تواند به یک یا چند ماژول (module) که مانند پروژه‌های فرعی هسـتند، تقسـیم شـود. به طور پیش‌فرض، یک پروژه جدید دارای یک ماژول است که این تمام چیزی است که برای اولین پروژه ساده خود نیاز دارید.

پوشه‌های .gradle و .idea که در شکل ۱.۷ نشان داده شده‌اند، ممکن است هنگام باز کردن پروژه در یک مرورگر فایل (file explorer) پنهان باشند. آن‌ها پوشه‌های «پشت صحنه» هستند که نیازی به دسترسی به آن‌ها نخواهید داشت؛ .gradle حاوی حافظه پنهان (caches) مورد استفاده توسط سیستم ساخت است، و .idea شامل فایل‌های تنظیمات برای پروژه و IDE شما است.

پوشـه gradle و فایل‌هایی مانند gradlew و gradlew.bat مربوط به Gradle Wrapper هسـتند؛ یک کپی خودکفا از سـیسـتم سـاخت Gradle که می‌توانید بدون نیاز به نصـب Gradle روی کامپیوترتان، از آن اسـتفاده کنید. فایل‌های build.gradle، gradle.properties و settings.gradle، فایل‌های پیکربندی سـیسـتم سـاخت Gradle هسـتند. آن‌ها بخش‌های مختلفی از اطلاعات مانند نام پروژه، سـطح زبان (language level)، ماژول‌های پروژه و وابسـتگی‌های پروژه شـما را تعریف می‌کنند. این فایل‌های تولید شده به طور خودکار را همان‌طور که هستند رها کنید.

بخش External Libraries شامل اطلاعات مربوط به کتابخانه‌هایی است که پروژه به آن‌ها وابسته است. اگر این بخش را باز کنید، خواهید دید که IntelliJ به طور خودکار جاوا و چندین بسته از کتابخانه استاندارد کاتلین را به عنوان وابستگی برای پروژه شما اضافه کرده است.

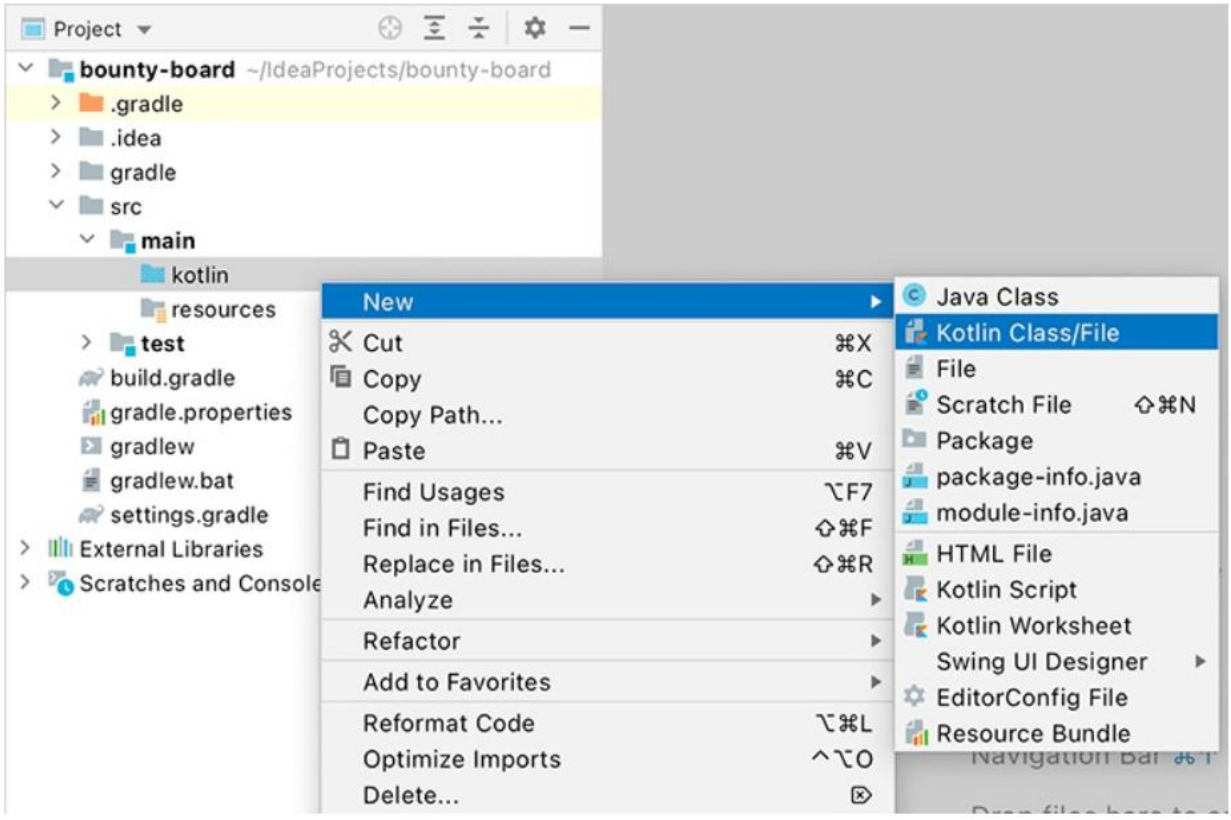
(شما می‌توانید در وب‌سایت مستندات JetBrains به آدرس jetbrains.org/intellij/sdk/docs/basics/project_structure.html درباره ساختار پروژه IntelliJ اطلاعات بیشتری کسب کنید. همچنین می‌توانید در وب‌سایت Gradle به آدرس docs.gradle.org/current/userguide/organizing_gradle_projects.html درباره ساختار یک پروژه Gradle بیشتر بیاموزید.)

پوشه src/main/kotlin مکانی است که شما تمام فایل‌های کاتلینی که برای پروژه‌تان ایجاد می‌کنید را در آن قرار خواهید داد. و با انجام این کار، زمان ایجاد و ویرایش اولین فایل کاتلین شما فرا رسیده است.

ایجاد اولین فایل کاتلین شما

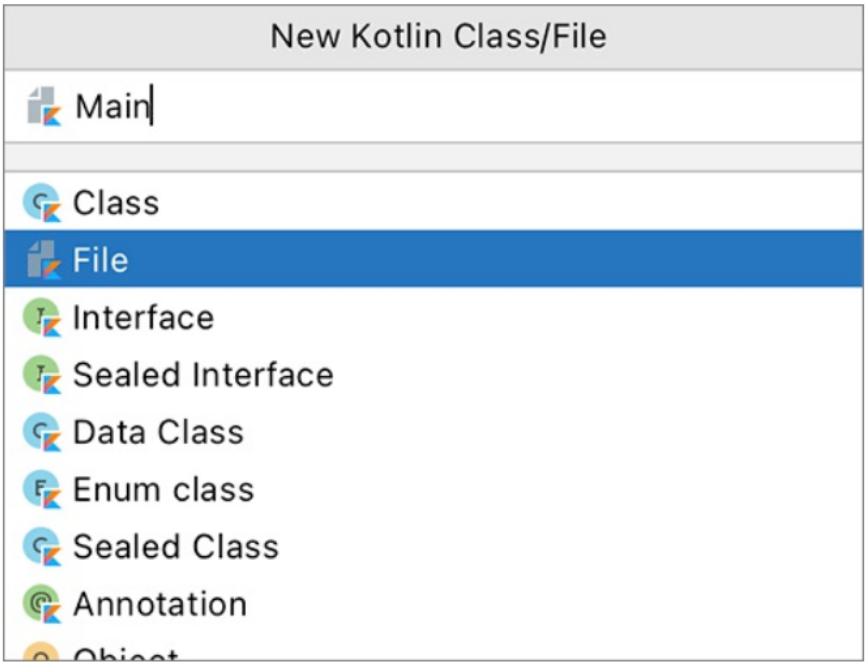
در پنجره ابزار پروژه پوشه src/main/kotlin را باز کرده و روی آن راست‌کلیک کنید. از منوی ظاهر شده، New و سپس Kotlin Class/File را انتخاب کنید (شکل ۱.۸).

Figure 1.8 Creating a new Kotlin file



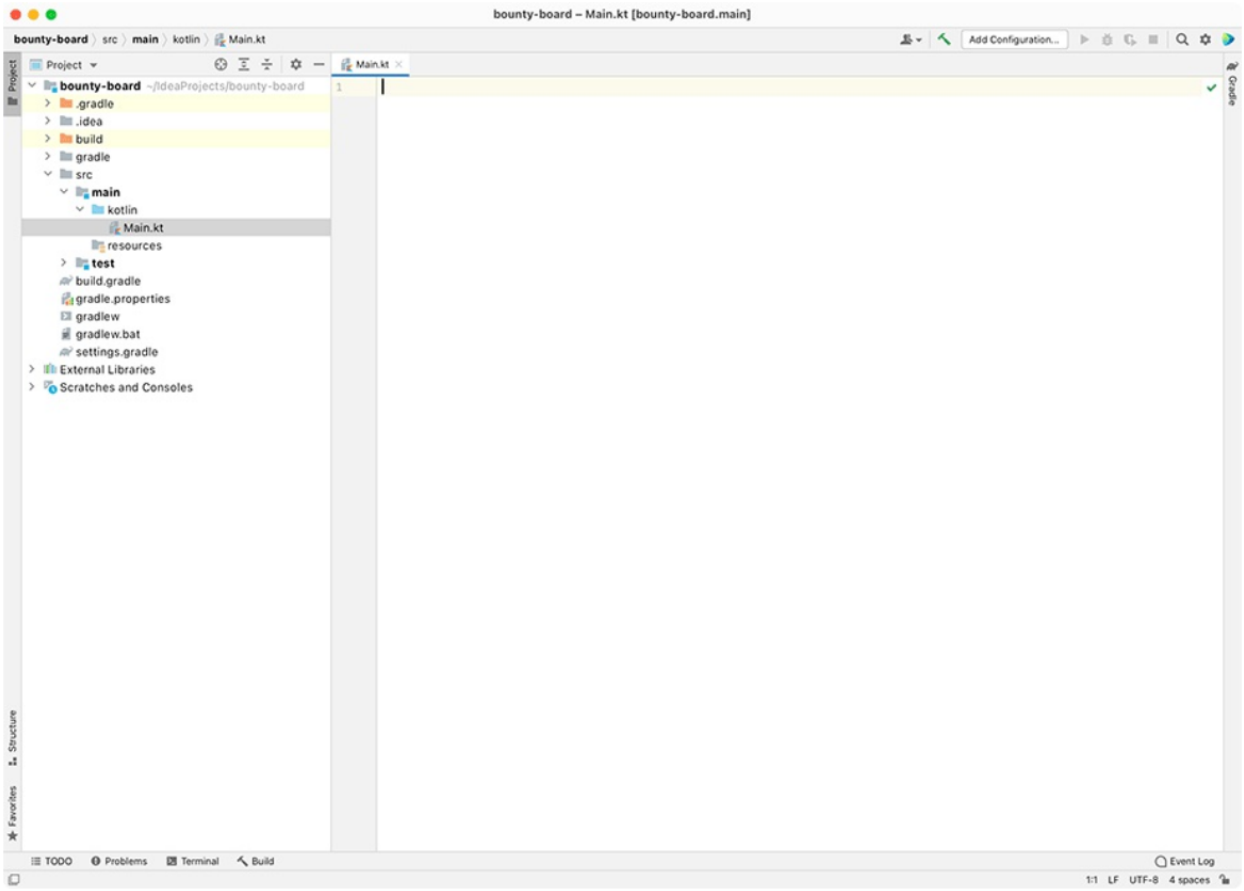
در پنجره New Kotlin Class/File، مقدار Main را در فیلد Name تایپ کنید و در لیســـت نوع فایل‌های زیرین، روی File دابل‌کلیک کنید (شکل ۱.۹).

Figure 1.9 Naming the file



نرم‌افزار IntelliJ فایل جدیدی را در پروژه شما ایجاد خواهد کرد، به آدرس src/main/kotlin/Main.kt، و محتویات فایل را در ویرایشگرِ سمت راست پنجره IntelliJ نمایش می‌دهد (شکل ۱.۱۰). پسوند .kt نشان می‌دهد که فایل حاوی کدهای کاتلین است، دقیقاً همان‌طور که از پسوند .java برای فایل‌های جاوا و .py برای فایل‌های پایتون استفاده می‌شود.

Figure 1.10 Empty Main.kt file displays in editor



سـرانجام، آماده نوشـتن کد کاتلین هسـتید. کمی به انگشـتان خود کشـش دهید و دسـت به کار شـوید. کد زیر را در ویرایشـگر Main.kt تایپ کنید. (به یاد داشته باشید که در سراسر این کتاب، کدی که باید وارد کنید با فونت پررنگ نشان داده شده است.)

Listing 1.1 “Hello, world!” in Kotlin (Main.kt)

```
fun main() {
    println("Hello, world!")
}
```

کدی که به تازگی نوشتید ممکن است ناآشنا به نظر برسد. نترسید - تا پایان این کتاب، خواندن و نوشتن کاتلین برای شما مانند طبیعت ثانویه خواهد شد. در حال حاضر، کافیست کد را در یک سطح کلی درک کنید.

کد در Listing 1.1 یک تابع (function) جدید را تعریف می‌کند. یک تابع، گروهی از دسـتورالعمل‌هاسـت که می‌توانند بعداً اجرا شوند. شما در فصل ۴ به طور مفصل در مورد نحوه تعریف و کار با توابع خواهید آموخت.

این تابع خاص – تابع main – در کاتلین معنای ویژه‌ای دارد. تابع main نقطه شروع برنامه شما را مشخص می‌کند. این نقطه، نقطه ورود برنامه (application entry point) نامیده می‌شـود و برای قابل اجرا بودن پروژه bounty-board (یا هر برنامه‌ای)، یک نقطه ورود از این دست باید تعریف شود. هر پروژه‌ای که در این کتاب می‌نویسید با یک تابع main شروع خواهد شد.

تابع main شما شامل یک دستورالعمل (که به عنوان statement نیز شناخته می‌شود) است: println(“Hello, world!”). دستور println() نیز یک تابع است؛ تابعی که در کتابخانه استاندارد کاتلین تعبیه شده و در تمام پلتفرم‌های هدفی که کاتلین پشتیبانی می‌کند در دسترس است. هنگامی که برنامه اجرا شده و println(“Hello, world!”) پردازش می‌شود، IntelliJ محتویات داخل پرانتز را (بدون علامت‌های نقل قول، بنابراین در این مورد Hello, world!) روی صفحه چاپ خواهد کرد.

اجرای فایل کاتلین شما

مدت کوتاهی پس از اینکه تایپ کد موجود در Listing 1.1 را به پایان رساندید، IntelliJ یک فلش سبز رنگ، معروف به “دکمه اجرا” (run button) را در سمت چپ خط اول نمایش می‌دهد (شکل ۱.۱۱). (اگر نماد ظاهر نشد، یا اگر خط قرمزی را در زیر نام فایل در تب یا زیر هر یک از کدهای وارد شده مشاهده کردید، این بدان معناست که خطایی در کد خود دارید. دوباره بررسی کنید که کد را دقیقاً همانطور که در Listing 1.1 نشان داده شده است تایپ کرده باشید.)

Figure 1.11 Run button

زمان آن رسیده است که برنامه شما جان بگیرد و به دنیا سلام کند. روی دکمه اجرا کلیک کنید. از منوی ظاهر شده Run ’MainKt‘ را انتخاب کنید (شکل ۱.۱۲). این کار به IntelliJ می‌گوید که می‌خواهید برنامه خود را در حال اجرا ببینید.

Figure 1.12 Running Main.kt

هنگامی که برنامه خود را اجرا می‌کنید، IntelliJ کدهای داخل آکولادها ({}) را خط به خط اجرا کرده و سپس اجرا را خاتمه می‌دهد. همچنین یک پنجره ابزار جدید در پایین پنجره IntelliJ نمایش می‌دهد (شکل ۱.۱۳).

Figure 1.13 Run tool window (console)

این پنجره، پنجره ابزار اجرا (run tool window) است که با عنوان کنسول (console) نیز شناخته می‌شود (که از این پس ما نیز آن را همین‌گونه خواهیم نامید). این بخش اطلاعاتی در مورد آنچه با اجرای برنامه شما توسط IntelliJ اتفاق افتاده و همچنین هرگونه خروجی که برنامه شما چاپ می‌کند را نمایش می‌دهد. باید عبارت Hello, world! را مشاهده کنید که در کنسول شما چاپ شده است. شما همچنین باید عبارت Process finished with exit code 0 را ببینید که نشان دهنده تکمیل موفقیت آمیز اجراست. این خط در صورت عدم وجود خطا در انتهای تمام خروجی‌های کنسول ظاهر می‌شود؛ ما از این به بعد آن را در نتایج کنسول نشان نخواهیم داد.

کامپایل و اجرای کد Kotlin/JVM

در زمان کوتاه بین زمانی که گزینه 'Run 'MainKt دکمه اجرا را انتخاب می‌کنید تا زمانی که Hello, world! در کنسول چاپ شود، اتفاقات زیادی می‌افتد.

ابتدا، IntelliJ کد کاتلین را با استفاده از کامپایلر kotlinc-jvm کامپایل می‌کند. کامپایلر برنامه‌ای است که کد منبع را به یک زبان سطح پایین‌تر ترجمه می‌کند تا یک برنامه قابل اجرا ایجاد کند. اگر در حال هدف‌گذاری پلتفرم متفاوتی بودید، IntelliJ حسب مورد از kotlinc-js یا kotlinc-native استفاده می‌کرد.

هر سه نسخه از کامپایلر kotlinc تقریباً به یک روش کار می‌کنند. هنگامی که kotlinc-jvm اجرا می‌شود، کد کاتلینی را که شما نوشته‌اید به بایت‌کد (bytecode) ترجمه می‌کند؛ زبانی که JVM با آن "صحبت" می‌کند. اگر kotlinc برای ترجمه کد کاتلین شما با مشکلی روبرو شود، پیام (یا پیام‌های) خطایی را نمایش می‌دهد و به شما راهنمایی می‌کند که چگونه مشکلات را برطرف کنید. در غیر این صورت، اگر فرآیند کامپایل به آرامی پیش برود، IntelliJ به مرحله اجرا (execution phase) می‌رود.

در مرحله اجرا، بایت‌کدی که توسط kotlinc-jvm تولید شده است روی ماشین مجازی جاوا (JVM) اجرا می‌شود. کنسول هرگونه خروجی از برنامه شما را نمایش می‌دهد، مانند چاپ متنی که در فراخوانی خود برای تابع println() مشخص کرده‌اید، همان‌طور که JVM دستورالعمل‌ها را اجرا می‌کند.

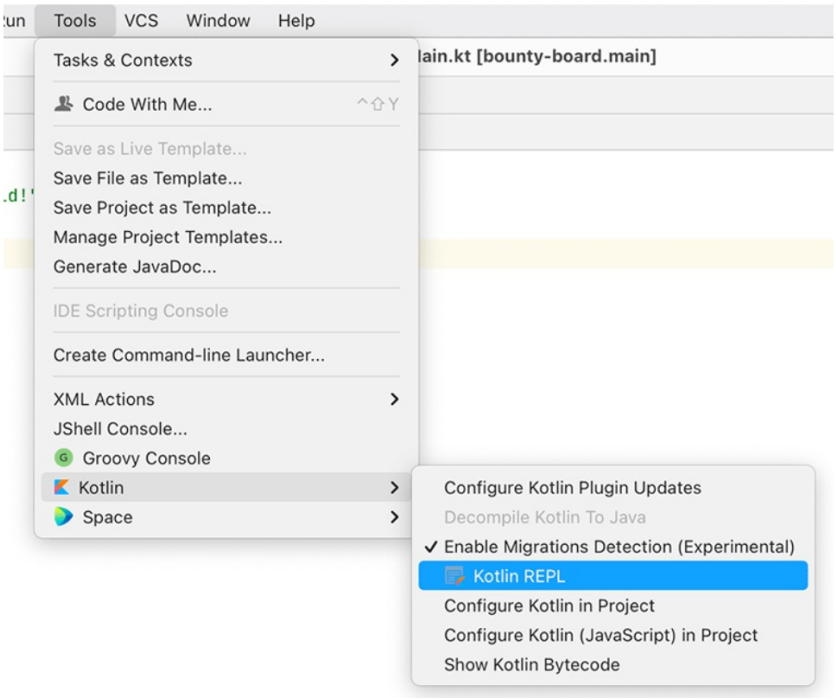
وقتی دیگر هیچ دستورالعمل بایت‌کدی برای اجرا وجود نداشته باشد، JVM متوقف می‌شود. IntelliJ وضعیت خاتمه را در کنسول نشان می‌دهد و به شما اطلاع می‌دهد که آیا اجرا با موفقیت به پایان رسیده یا با کد خطا خاتمه یافته است.

برای کار با این کتاب به درک جامعی از فرآیند کامپایل کاتلین نیاز نخواهید داشت. با این حال، ما در فصل ۲ درباره بایت‌کد با جزئیات بیشتری بحث خواهیم کرد.

قابلیت REPL در کاتلین

گاهی اوقات ممکن است بخواهید بخش کوچکی از کد کاتلین را تست کنید تا ببینید هنگام اجرای آن چه اتفاقی می‌افتد، مشابه اینکه چگونه از یک تکه کاغذ چک‌نویس برای یادداشت مراحل یک محاسبه کوچک استفاده می‌کنید. این امر به خصوص زمانی که در حال یادگیری زبان کاتلین هستید بسیار مفید است. خوشبختانه برای شما، IntelliJ ابزاری را برای تست سریع کد بدون نیاز به ایجاد فایل ارائه می‌دهد. این ابزار Kotlin REPL نامیده می‌شود. در ادامه در مورد این نام توضیح خواهیم داد - فعلا آن را باز کنید و ببینید چه کاری می‌تواند انجام دهد.

در IntelliJ، با انتخاب Tools سپس Kotlin و بعد Kotlin REPL پنجره ابزار REPL را باز کنید (شکل ۱.۱۴).



نرم‌افزار IntelliJ پنجره REPL را در پایین صفحه نمایش می‌دهد (شکل ۱.۱۵).

Figure 1.15 The Kotlin REPL tool window

شما می‌توانید کدهای خود را مستقیماً در REPL تایپ کنید، درست مانند ویرایشگر. تفاوت در این است که می‌توانید آن را به سرعت ارزیابی کنید، بدون اینکه نیاز به کامپایل کل پروژه داشته باشید.

(ممکن است هنگام شروع کار با REPL متن هشدار قرمزی را مشاهده کنید یا هشدارهایی درباره "اجرای REPL با کلاس‌های قدیمی (outdated classes)". به طور کلی می‌توانید این گونه هشدارها را نادیده بگیرید. مشکلات عمومی مربوط به پلتفرم یا JVM هنگام استفاده از REPL هیچ آسیبی وارد نمی‌کند و از آنجایی که شما در REPL به کد پروژه bounty-board دسترسی نخواهید داشت، نیازی به نگرانی در مورد هشدار کلاس‌های قدیمی ندارید.)

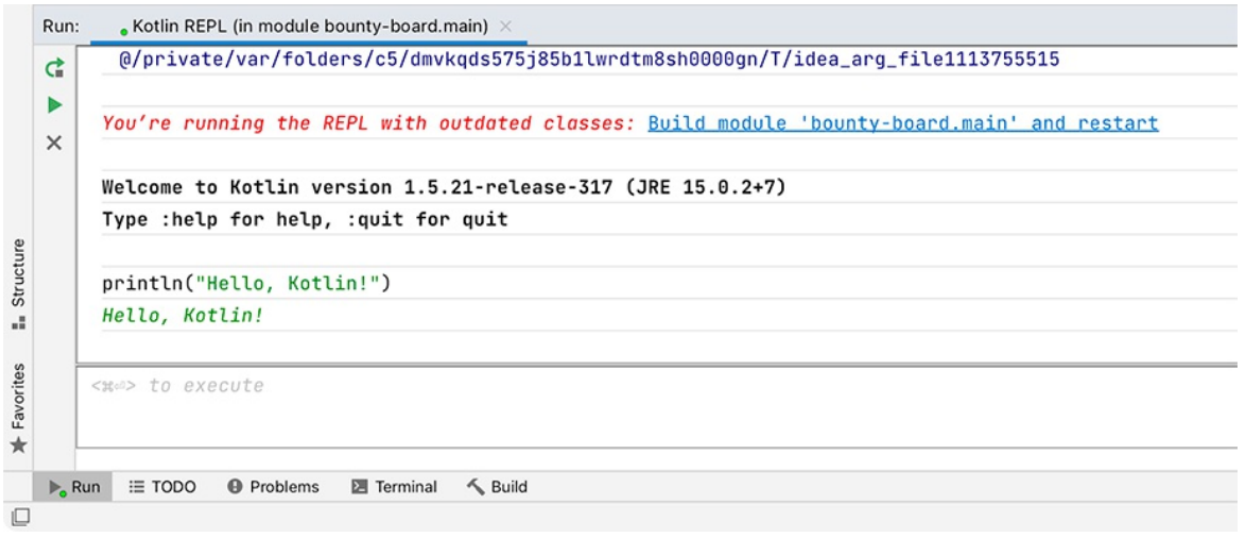
کد زیر را در REPL وارد کنید:

Listing 1.2 “Hello, Kotlin!” (REPL)

```
println("Hello, Kotlin!")
```

پس از وارد کردن متن، دکمه Command-Return (یا Ctrl-Enter) را فشـار دهید تا کد در REPL ارزیابی شـود. پس از یک لحظه، خروجی حاصل را در زیر آن مشاهده خواهید کرد که باید Hello, Kotlin! باشد (شکل ۱.۱۶).

Figure 1.16 Evaluating the code



کلمه REPL مخفف عبارت “read, evaluate, print, loop” (بخوان، ارزیابی کن، چاپ کن، حلقه تکرار) اسـت. شـما قطعه کدی را در خط فرمان تایپ می‌کنید و آن را با کلیک روی دکمه سـبز رنگ اجرا در سـمت چپ REPL یا با فشـردن کلیدهای Command-Return (Ctrl-Enter) ارسـال می‌نمایید. REPL سـپس کد را می‌خواند (read)، آن را ارزیابی و اجرا می‌کند (evaluate)، و مقدار حاصـل یا اثر جانبی آن را چاپ می‌کند (print). هنگامی که اجرای REPL به پایان می‌رسـد، کنترل را به شما بازمی‌گرداند و حلقه فرآیند دوباره از ابتدا آغاز می‌شود (loop).

سفر کاتلینی شما آغاز شـده اسـت! شـما در این فصل کار بزرگی انجام دادید و پایه‌های دانش در حال رشد خود از برنامه‌نویسـی کاتلین را بنا نهادید. در فصـل بعد، با یادگیری نحوه اسـتفاده از متغیرها، ثابت‌ها و انواع داده‌ها برای نمایش اطلاعات، شـروع به کندوکاو در جزئیات این زبان خواهید کرد.

برای افراد کنجکاوتر: چرا از IntelliJ استفاده کنیم؟

کد کاتلین را می‌توان با استفاده از هر ویرایشگر متن ساده‌ای (plain text editor) نوشت. با این حال، ما استفاده از IntelliJ را توصیه می‌کنیم، مخصوصاً زمانی که در حال یادگیری هستید. همان‌طور که نرم‌افزار ویرایش متنی که دارای بررسی املا و دستور

زبان (spell and grammar check) است، نوشتن یک مقاله نثری خوش‌ساخت را آسان‌تر می‌کند، IntelliJ نیز نوشتن کاتلین با ساختار صحیح را آسان‌تر می‌سازد. IntelliJ به شما در این موارد کمک می‌کند:

نوشتن کدهایی با صحت نحوی (syntactically) و معنایی (semantically) به کمک ویژگی‌هایی مانند برجسته‌سازی نحو (syntax highlighting)، پیشنهادهای حساس به متن (context-sensitive suggestions) و تکمیل خودکار کد (automatic code completion).

اجرا و اشکال‌زدایی (debug) کد با ویژگی‌هایی مانند نقاط توقف اشکال‌زدایی (debug breakpoints) و پیمایش کدهای به صورت بلادرنگ (real-time code stepping) هنگام اجرای برنامه‌تان.

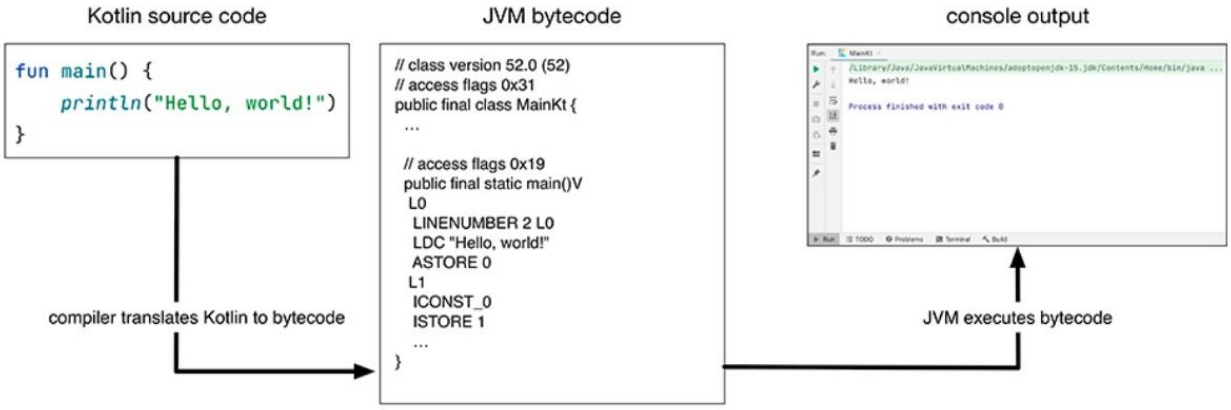
بازسازی و تغییر ساختار کدهای موجود با استفاده از میانبرهای بازآرایی کد (refactoring) (مانند تغییر نام و استخراج ثابت‌ها) و قالب‌بندی کد برای مرتب کردن دندانه‌گذاری (indentation) و فاصله‌ها.

همچنین از آنجایی که کاتلین توسط JetBrains ایجاد شده است، ادغام بین IntelliJ و کاتلین به دقت طراحی شده و اغلب منجر به تجربه ویرایش لذت‌بخشی می‌گردد. به عنوان یک مزیت اضافی، IntelliJ پایه و اساس نرم‌افزار Android Studio است، بنابراین در صورت تمایل شما، میانبرها و ابزارهایی که در اینجا یاد می‌گیرید قابل انتقال به محیط Android Studio خواهند بود.

برای افراد کنجکاوتر: هدف‌گیری JVM

ماشین مجازی جاوا (JVM) نرم‌افزاری است که می‌داند چگونه مجموعه‌ای از دستورالعمل‌ها را به نام بایت‌کد (bytecode) اجرا کند. "هدف‌گیری JVM" به معنای کامپایل کردن یا ترجمه کد منبع کاتلینِ شما به بایت‌کد جاوا، با هدف اجرای آن بایت‌کد روی ماشین مجازی جاوا است (شکل ۱.۱۷).

Figure 1.17 Compilation and execution flow



هر پلتفرمی مانند Windows یا macOS، مجموعه دستورالعمل‌های خاص خود را دارد. ماشین مجازی جاوا (JVM) به عنوان پلی بین بایت‌کد و محیط‌های سخت‌افزاری و نرم‌افزاری مختلفی که روی آن‌ها اجرا می‌شود، عمل می‌کند و با خواندن یک

قطعه بایت‌کد، دستورالعمل(های) مربوطه به آن پلتفرم خاص را که به آن بایت‌کد نگاشت می‌شود، فراخوانی می‌کند. بنابراین، نسخه‌های مختلفی از JVM برای پلتفرم‌های مختلف وجود دارد. این همان چیزی است که به توسعه‌دهندگان کاتلین اجازه می‌دهد تا کدهای مستقل از پلتفرمی بنویسند که تنها یک بار نوشته شده و سپس به بایت‌کد کامپایل شود و در دستگاه‌های مختلف صرف‌نظر از سیستم‌عامل‌هایشان اجرا گردد.

از آنجا که کاتلین می‌تولند به بایت‌کدی تبدیل شود که JVM قادر به اجرای آن است، کاتلین یک زبان مبتنی بر JVM محسوب می‌شود. جاوا احتمالاً شناخته‌شده‌ترین زبان مبتنی بر JVM است، زیرا اولین مورد بوده است. با این حال، دیگر زبان‌های JVM مانند Scala، Groovy و Kotlin به منظور رفع برخی از کاستی‌های جاوا از منظر توسعه‌دهندگان، پدیدار شده‌اند.

چالش: محاسبات ریاضی در REPL

بسیاری از فصل‌های این کتاب با یک یا چند چالش به پایان می‌رسند. این چالش‌ها برای این هستند که خودتان روی آن‌ها کار کنید تا درک خود را از کاتلین عمیق‌تر کرده و کمی تجربه بیشتر کسب کنید.

از REPL برای بررسی نحوه عملکرد عملگرهای ریاضی در کاتلین استفاده کنید: +, −,*,/ و %. به عنوان مثال، 2*(9+12)را در REPL تایپ کنید. آیا خروجی با چیزی که انتظار داشتید مطابقت دارد؟

اگر می‌خواهید عمیق‌تر شوید، توابع ریاضی موجود در کتابخانه استاندارد کاتلین را در آدرس kotlinlang.org/api/latest/jvm/stdlib/kotlin.math/ بررسی کنید و آن‌ها را در REPL امتحان کنید. به عنوان مثال، min(۹۴, -۹۹) را امتحان کنید که حداقل مقدار از دو عدد ارائه شده در پرانتز را به شما می‌گوید.

شما باید با وب‌سایت kotlinlang.org و به خصوص مستندات زبان موجود در آن احساس راحتی کنید. اگر قصد دارید از کاتلین در خارج از محیط JVM استفاده کنید، هنگام خواندن مراجع API به دایره‌های رنگی توجه کنید. این موارد نشان می‌دهند که یک API داده شده از چه پلتفرم‌هایی پشتیبانی می‌کند. بسیاری از APIها مانند absoluteValue (نشان داده شده در شکل ۱.۱۸) به عنوان APIهای مشترک (common) در نظر گرفته می‌شوند و روی تمامی پلتفرم‌های هدفی که کاتلین پشتیبانی می‌کند، کار می‌کنند.

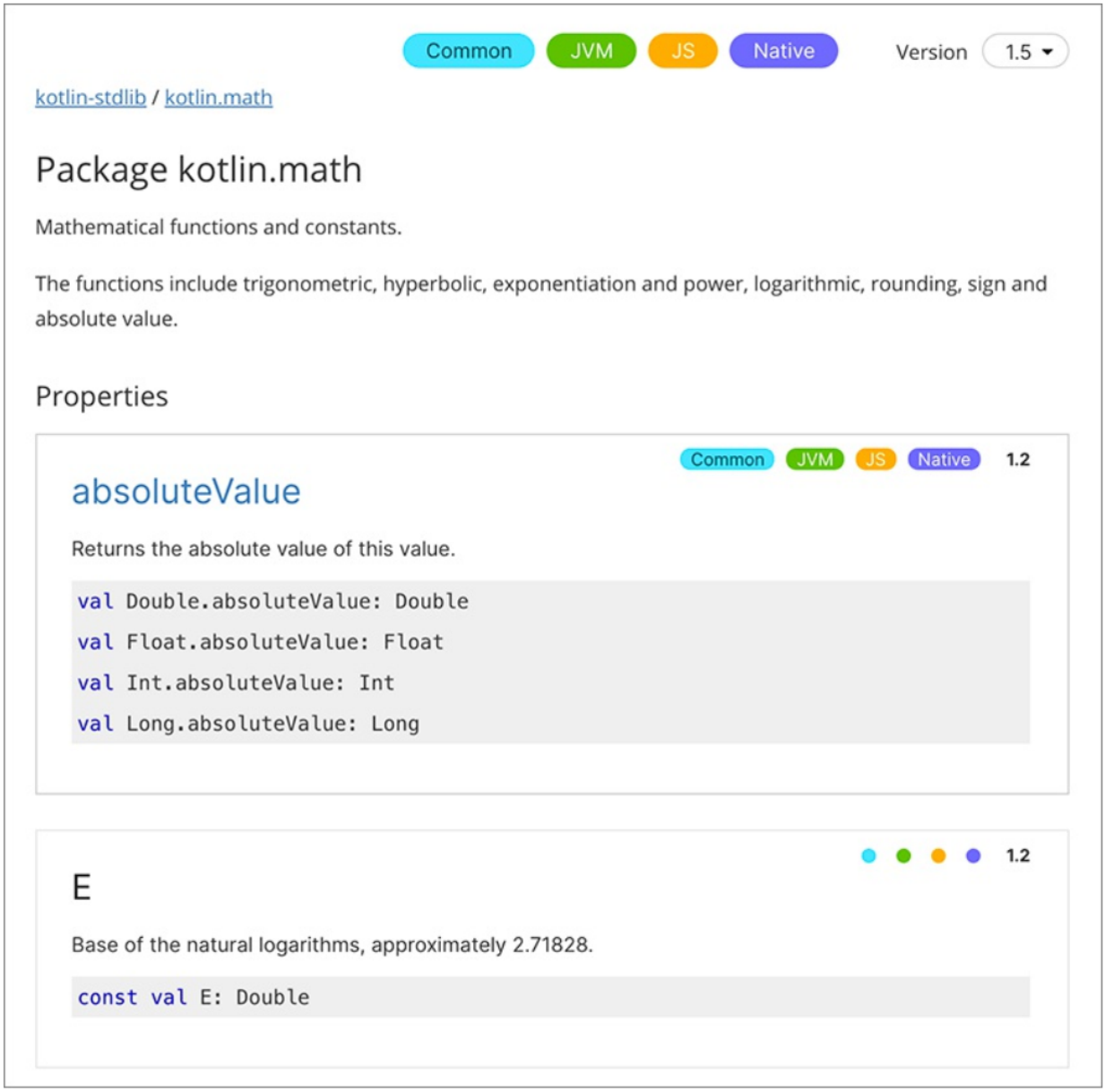


Figure 1.18 Platform support in the Kotlin API documentation

۲ متغیرها، ثابت‌ها و انواع داده

این فصل شما را با متغیرها، ثابت‌ها و انواع داده‌ی (data types) پایه‌ی کاتلین آشنا می‌کند - عناصر بنیادی هر برنامه‌ای. شما از متغیرها و ثابت‌ها برای ذخیره‌سازی مقادیر و انتقال داده‌ها در برنامه‌ی خود استفاده می‌کنید. انواع داده، نوع خاصی از داده را که توسط یک ثابت یا متغیر نگهداری می‌شود، توصیف می‌کنند.

تفاوت‌های مهمی میان هر یک از انواع داده و همچنین میان متغیرها و ثابت‌ها وجود دارد که نحوه‌ی استفاده از آن‌ها را شکل می‌دهد.

انواع داده

متغیرها و ثابت‌ها دارای یک نوع داده هستند که شما آن را مشخص می‌کنید. این نوع، داده‌ای را که توسط یک متغیر یا ثابت نگهداری می‌شود توصیف کرده و به کامپایلر می‌گوید که بررسی نوع (type checking) - ویژگی‌ای در کاتلین که از تخصیص نوع نامناسب داده به یک متغیر یا ثابت جلوگیری می‌کند - چگونه مدیریت خواهد شد.

برای مشاهده‌ی عملی این مفهوم، فایل Main.kt را در پروژه‌ی bounty-board که در فصل ۱ ایجاد کردید، ویرایش خواهید کرد. اگر از فصل گذشته محیط IntelliJ را بسته‌اید، دوباره آن را اجرا کنید. پروژه‌ی bounty-board احتمالاً به‌طور خودکار باز خواهد شد، زیرا IntelliJ جدیدترین پروژه‌ی شما را مجدداً باز می‌کند. در غیر این صورت، می‌توانید bounty-board را از لیست فایل‌های اخیر در مرکز دیالوگ خوش‌آمدگویی یا با انتخاب File سپس Open Recent و در نهایت bounty-board باز کنید.

اعلان یک متغیر

تصور کنید در حال نوشتن یک بازی ماجراجویی هستید که به بازیکنان مأموریت‌هایی برای انجام دادن محول می‌کند. با قوی‌تر شدن بازیکن و پیشرفت در بازی، سطح دشواری این مأموریت‌ها نیز باید افزایش یابد. احتمالاً به متغیری برای پیگیری پیشرفت بازیکن در بازی نیاز خواهید داشت.

در Main.kt، اولین متغیر خود را با نام playerLevel ایجاد کرده و مقداری به آن اختصاص دهید:

Listing 2.1 Declaring a playerLevel variable (Main.kt)

```
fun main()
{
println("Hello, world!")
var playerLevel: Int = 4
println(playerLevel)
}
```

در اینجا، شما نمونه‌ای از نوع Int را به متغیری به نام playerLevel اختصاص داده‌اید. بیایید هر بخش از کد را بررسی کنیم.

شما یک متغیر را با استفاده از کلمه‌ی کلیدی var تعریف کردید که نشان می‌دهد می‌خواهید یک متغیر جدید اعلان کنید، و به دنبال آن نام متغیر جدید آمده است.

سپس، تعریف نوع متغیر، یعنی : Int را مشخص کردید که نشان می‌دهد playerLevel یک مقدار صحیح (عدد کامل) را در خود نگه می‌دارد.

در نهایت، از عملگر تخصیص (=)استفاده کردید تا آنچه در سمت راست قرار دارد (نمونه‌ای از نوع Int، به طور خاص ۴) را به آنچه در سمت چپ قرار دارد (playerLevel) اختصاص دهید.

شکل ۲.۱ تعریف متغیر playerLevel را در قالب نمودار نشان می‌دهد.

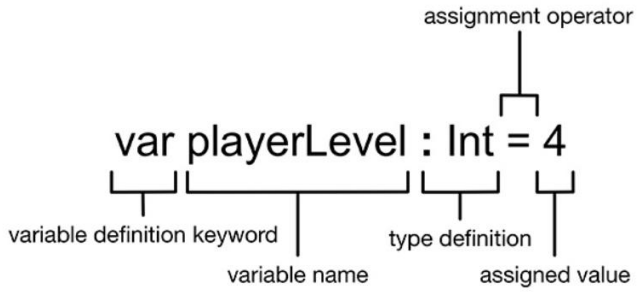


Figure 2.1 Anatomy of a variable definition

پس از تعریف متغیر، با استفاده از تابع println مقدار آن را در کنسول چاپ می‌کنید.

برنامه را با کلیک روی دکمه‌ی اجرا در کنار تابع main و انتخاب ‘MainKt Run’ اجرا کنید. همچنین می‌توانید با کلیک روی دکمه‌ی اجرا در نوار منوی IntelliJ برنامه را اجرا کنید. نتیجه‌ی چاپ شده در کنسول مقدار ۴ است، همان مقداری که به playerLevel اختصاص دادید.

اکنون، سعی کنید به جای آن مقدار "thirty-two" را به playerLevel اختصاص دهید. (خط‌خوردگی نشان‌دهنده‌ی کدی است که باید حذف کنید.)

Listing 2.2 Assigning “thirty-two” to playerLevel (Main.kt)

```
fun main()
{
println("Hello, world!")
~~var playerLevel: Int = 4~~
var playerLevel: Int = "thirty-two"
println(playerLevel)
}
```

دوباره main را با کلیک روی دکمه‌ی اجرا، اجرا کنید. این بار، کامپایلر کاتلین یک خطا نمایش می‌دهد:

e: Main.kt: (3, 28): Type mismatch: inferred type is String but Int was expected

هنگام تایپ این کد، ممکن است متوجه خط قرمز رنگی زیر "thirty-two" شده باشید. این نشانه‌ی IntelliJ است که برنامه دارای خطا می‌باشد. ماوس را روی "thirty-two" نگه دارید تا جزئیات مشکل شناسایی‌شده را بخوانید.

Figure 2.2 Type mismatch disclosure tooltip

کاتلین از یک سیستم نوع استاتیک (static type system) استفاده می‌کند - به این معنی که کامپایلر، کد منبعی را که شما تعریف می‌کنید با انواع داده برچسب‌گذاری می‌کند تا بتواند اطمینان حاصل کند کدی که می‌نویسید معتبر است. نوع‌گذاری استاتیک همچنین به این معناست که پس از تعریف یک متغیر، نمی‌توانید نوع آن را پس از اعلان تغییر دهید.

IntelliJ کد را هنگام تایپ بررسی می‌کند و متوجه خطاهای کامپایلر مانند خطای ناشی از تخصیص نادرست مقداری با نوع متفاوت به یک متغیر می‌شود. این ویژگی، بررسی استاتیک نوع (static type checking) نامیده می‌شود و پیش از آنکه حتی برنامه را کامپایل کنید، اشتباهات برنامه‌نویسی را به شما اطلاع می‌دهد.

برای رفع این خطا، با تغییر مجدد "thirty-two" به ۴، مقدار Intی را به playerLevel اختصاص دهید که با نوع اعلان‌شده‌ی آن مطابقت داشته باشد:

Listing 2.3 Fixing the type error (Main.kt)

```
fun main()
{
println("Hello, world!")
~~var playerLevel: Int = "thirty-two"~~
var playerLevel: Int = 4
println(playerLevel)
}
```

تا زمانی که به سیستم نوع‌گذاری احترام بگذارید، یک متغیر می‌تواند در طول برنامه‌ی شما مجدداً مقداردهی شود. به عنوان مثال، اگر رتبه‌ی بازیکن افزایش یابد، می‌توانید مقدار جدیدی به متغیر playerLevel اختصاص دهید. متغیر playerLevel را برابر با ۵ قرار دهید، همان‌طور که نشان داده شده است:

Listing 2.4 Increasing playerLevel by resetting the value (Main.kt)

```
fun main()
{
println("Hello, world!")
```

```
  var playerLevel: Int = 4
  println(playerLevel)
  println("The hero embarks on her journey to locate the enchanted sword.")
  playerLevel = 5
  println(playerLevel)
  }
```

تابع main به‌روزشده‌ی خود را اجرا کنید تا این تخصیص را در عمل ببینید. عدد ۵ را خواهید دید که در خط مخصوص به خود پس از رشته‌ی The hero embarks on her journey to locate the enchanted sword. چاپ می‌شود.

با اســـتفاده از عملگر مســـاوی (=)، می‌توانید متغیرها را برای گرفتن مقادیر جدید مقداردهی کنید. این روش کار می‌کند، اما بهتر اســت به جای تنظیم مقدار playerLevel روی ۵، یک واحد به آن اضــافه (increment) کنید. افزایش ســطح انتخاب بهتری است زیرا در صورتی که تصمیم بگیرید سطح شروع بازیکن را تغییر دهید، کد شما را انعطاف‌پذیرتر می‌کند.

مقداردهی مجدد را طوری به‌روز کنید که به جای آن، مقدار متغیر افزایش یابد. در حین انجام این تغییرات، پیام خوش‌آمدگویی برنامه‌ی خود را که در حال نامربوط شدن است، نیز به‌روزرسانی کنید.

Listing 2.5 Incrementing playerLevel (Main.kt)

```
  fun main()

  {

    ~~println("Hello, world!")~~
    println("The hero announces her presence to the world.")
    var playerLevel: Int = 4
    println(playerLevel)
    println("The hero embarks on her journey to locate the enchanted sword.")
    ~~playerLevel = 5~~
    playerLevel += 1
     println(playerLevel)
  }
```

پس از تخصیص مقدار ۴ به متغیر playerLevel، از عملگر جمع و تخصیص (+=)برای اضافه‌کردن ۱ به مقدار اصلی استفاده می‌کنید. برنامه را دوباره اجرا کنید. پیام خوش‌آمدگویی جدید را خواهید دید و سطح بازیکن مانند قبل از ۴ به ۵ افزایش می‌یابد.

کاتلین روش‌های دیگری را نیز برای اختصــاص مقادیر ارائه می‌دهد. از آنجا که شــما در حال افزایش playerLevel به میزان ۱ واحد هستید، می‌توانید از عملگر افزایش (++) به جای عملگر =+به شکل زیر استفاده کنید:

```
  playerLevel++
```

برای تفریق به جای جمع، می‌توانید از عملگر کاهش (--) برای کم کردن ۱ یا عملگر تفریق و تخصــیص (=−)برای کم کردن هر مقداری اســتفاده کنید. همچنین عملگرهایی برای ضرب و تخصیص (=*) و تقسیم و تخصیص (=/) وجود دارد. در فصل ۵ با جزئیات بیشتری نحوه‌ی عملکرد عملیات‌های ریاضی را خواهید دید.

انواع داده‌ی داخلی کاتلین (Kotlin’s Built-In Types)

شما متغیرهایی را دیده‌اید که از نوع Int هستند، و هنگام فراخوانی println از نوع String استفاده کرده‌اید. کاتلین همچنین انواع مختلفی برای مدیریت مقادیری مانند درست/نادرست (true/false)، لیست‌هایی از عناصر و جفت‌های کلید-مقدار برای تعریف نگاشت‌های عناصر دارد. جدول ۲.۱ بسیاری از انواع داده‌ی داخلی پرکاربرد موجود در کاتلین را نشان می‌دهد:

Table 2.1 Commonly used built-in types

Type	Description	Examples
`String`	Text	`"Madrigal"` `"happy meal"`
`Char`	Single character	`'X'` Unicode character U+0041
`Boolean`	True/false values	true false
`Int`	Whole numbers	`5` `"Madrigal".length`
`Double`	Decimal numbers	`3.14` `2.718`
`List`	Collections of elements	`3, 1, 2, 4, 3` `"root beer", "club soda", "coke"`
`Set`	Collections of unique elements	`"Larry", "Moe", "Curly"` `"Mercury", "Venus", "Earth", "Mars", "Jupiter", "Saturn", "Uranus", "Neptune"`
`Map`	Collections of key-value pairs	`"small" to 5.99, "medium" to 7.99, "large" to 10.99`

اگر همه‌ی این انواع را ندیده‌اید، نگران نباشید - در طول این کتاب با آن‌ها آشنا خواهید شد. به طور خاص، در مورد رشته‌ها در فصل ۶ و اعداد در فصل ۵ بیشتر خواهید آموخت، و در مورد لیست‌ها، مجموعه‌ها (sets) و نگاشت‌ها (maps) که روی هم انواع مجموعه‌ای (collection types) نامیده می‌شوند، در فصل‌های ۹ و ۱۰ یاد خواهید گرفت.

متغیرهای فقط-خواندنی (Read-Only Variables)

تاکنون، متغیرهایی را دیده‌اید که مقادیر آن‌ها قابل تخصیص مجدد است – که به عنوان متغیرهای تغییرپذیر (mutable) نیز شناخته می‌شوند. اما اغلب، شما می‌خواهید از متغیرهایی استفاده کنید که مقادیرشان نباید تغییر کند. برای مثال، در بازی ماجراجویی متنی، نام بازیکن پس از آنکه در ابتدا اختصاص داده شد، تغییر نخواهد کرد.

کاتلین سینتکس (نحو) متفاوتی برای اعلان متغیرهای فقط-خواندنی – متغیرهایی که پس از تخصیص، قابل تغییر نیستند – ارائه می‌دهد.

شما متغیری را که می‌تواند اصلاح شود با استفاده از کلمه‌ی کلیدی var اعلان می‌کنید. برای اعلان یک متغیر فقط-خواندنی، از کلمه‌ی کلیدی val استفاده می‌کنید.

در اصطلاح عامیانه، متغیرهایی که مقادیرشان می‌تواند تغییر کند با نام varها و متغیرهای فقط-خواندنی با نام valها شناخته می‌شوند. ما از این به بعد از این قرارداد پیروی خواهیم کرد، زیرا عبارات "متغیر" و "متغیر فقط-خواندنی" کمتر متمایز هستند. هم varها و هم valها جزو "متغیرها" در نظر گرفته می‌شوند، بنابراین ما به استفاده از این اصطلاح برای اشاره به گروه آن‌ها ادامه خواهیم داد.

یک تعریف val برای نام بازیکن اضافه کرده و آن را پس از سطح اولیه‌ی بازیکن چاپ کنید:

Listing 2.6 Adding a heroName val (Main.kt)

```
fun main()
{
println("The hero announces her presence to the world.")

val heroName: String = "Madrigal"
println(heroName)
var playerLevel: Int = 4
println(playerLevel)

println("The hero embarks on her journey to locate the enchanted sword.")
playerLevel + = 1
println(playerLevel)
}
```

برنامه را با کلیک روی دکمه‌ی اجرا، چه در کنار تابع main و چه در نوار منو، اجرا کنید. مقادیر playerLevel و heroName را خواهید دید که در کنسول چاپ شده‌اند:

```
The hero announces her presence to the world.
Madrigal
4
The hero embarks on her journey to locate the enchanted sword. 5
```

سپس، سعی کنید با استفاده از عملگر تخصیص = ، یک مقدار String متفاوت را به heroName اختصاص دهید و برنامه را دوباره اجرا کنید.

Listing 2.7 Trying to change heroName's value (Main.kt)

```
fun main()
{
println("The hero announces her presence to the world.")

val heroName: String = "Madrigal"
println(heroName)
var playerLevel: Int = 4
```

```
println(playerLevel)

heroName = "Estragon"

println("The hero embarks on her journey to locate the enchanted sword.")
playerLevel + = 1
println(playerLevel)
}
```

برنامه را اجرا کنید، خطای کامپایل زیر را مشاهده خواهید کرد:

```
e: Main.kt: (9, 5): Val cannot be reassigned
```

کامپایلر شـکایت می‌کند زیرا شـما سـعی کرده‌اید یک val را تغییر دهید. زمانی که یک val تخصـیص یافت، هرگز نمی‌تواند مجدداً مقداردهی شود.

تخصیص دوم را برای رفع خطای مقداردهی مجدد حذف کنید:

Listing 2.8 Fixing the val reassignment error (Main.kt)

```
fun main() {
    println("The hero announces her presence to the world.")

    val heroName: String = "Madrigal"
    println(heroName)
    var playerLevel: Int = 4
    println(playerLevel)

    ~~heroName = "Estragon"~~

    println("The hero embarks on her journey to locate the enchanted sword.")
    playerLevel + = 1
    println(playerLevel)
}
```

اسـتفاده از valها برای محافظت در برابر تغییر تصـادفی متغیرهایی که باید فقط-خواندنی باشـند، مفید اسـت. به همین دلیل، توصیه می‌کنیم هر زمان که به یک var نیاز ندارید، از یک val استفاده کنید.

IntelliJ می‌تواند با تحلیل اسـتاتیک کد شـما تشـخیص دهد که چه زمانی می‌توان یک var را به یک val تبدیل کرد. اگر یک var هرگز تغییر نکند، IntelliJ به شـما پیشـنهاد می‌دهد که آن را به یک val تبدیل کنید. ما پیشـنهاد می‌کنیم که پیشـنهاد IntelliJ را دنبال کنید – مگر اینکه بخواهید کدی بنویسـید که آن var را مجدداً مقداردهی می‌کند. برای اینکه ببینید پیشـنهاد IntelliJ به چه شکل است، heroName را به یک var تغییر دهید:

Listing 2.9 Changing heroName to be reassignable (Main.kt)

```
fun main() {
```

```
println("The hero announces her presence to the world.")

val heroName: String = "Madrigal"
var heroName: String = "Madrigal"
println(heroName)
var playerLevel: Int = 4
println(playerLevel)
println("The hero embarks on her journey to locate the enchanted sword.")
playerLevel + = 1
println(playerLevel)
}
```

از آنجایی که مقدار heroName هرگز مجدداً تخصیص نمی‌یابد، نیازی نیست (و نباید) یک var باشد. توجه داشته باشید که IntelliJ یک هایلایت خردلی‌رنگ به خط حاوی کلمه‌ی کلیدی var اضافه کرده است. اگر ماوس خود را روی کلمه‌ی کلیدی var ببرید، IntelliJ بهبود پیشنهادی را توضیح می‌دهد (شکل ۲.۳).

Figure 2.3 Variable never modified tooltip

همانطور که انتظار می‌رود، IntelliJ پیشـــنهاد تبدیل heroName به یک val را می‌دهد. برای اعمال این پیشـــنهاد، روی کلمه‌ی کلیدی var در کنار heroName کلیک کرده و کلیدهای Option-Return (Alt-Enter) را فشـار دهید. در پنجره‌ی بازشده، Change to val را انتخاب کنید (شکل ۲.۴).

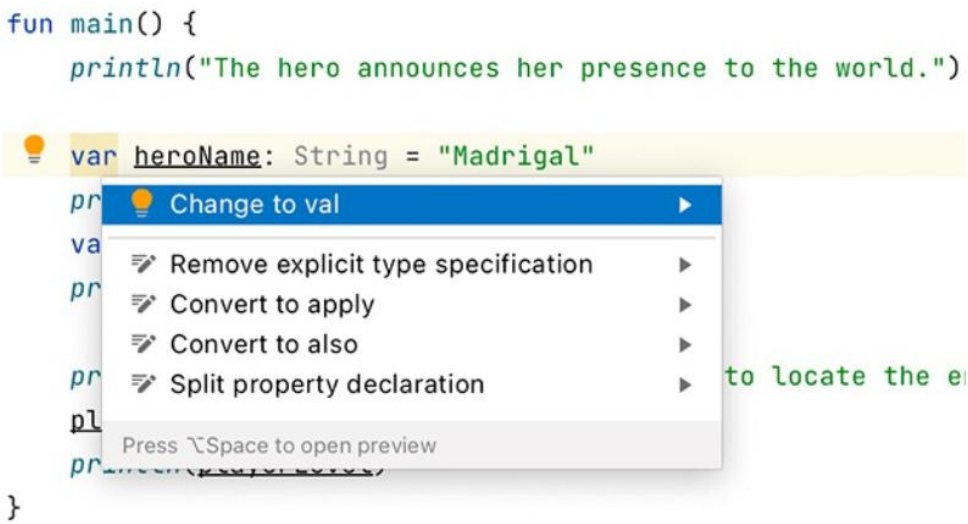


Figure 2.4 Making a variable immutable

IntelliJ به‌طور خودکار var را به یک val تبدیل می‌کند:

val heroName: String $=$ "Madrigal" println(heroName)

همانطور که پیش‌تر گفتیم، توصیه می‌کنیم هر زمان که می‌توانید از یک val استفاده کنید، تا کاتلین بتواند شما را از تخصیص‌های مجدد تصادفی آگاه سازد. همچنین پیشنهاد می‌کنیم که به پیشنهادات IntelliJ برای بهبود کد توجه کنید. ممکن است همیشه از آن‌ها استفاده نکنید، اما همیشه ارزش نگاه کردن را دارند.

استنتاج نوع (Type Inference)

توجه کنید که تعاریف نوعی که برای متغیرهای heroName و playerLevel مشخص کرده‌اید در IntelliJ به رنگ خاکستری درآمده‌اند. متن خاکستری‌شده نشان‌دهنده‌ی عنصری است که غیرضروری بوده یا استفاده نمی‌شود. ماوس خود را روی تعریف نوع String ببرید، IntelliJ توضیحی برای اینکه چرا این عنصر مورد نیاز نیست ارائه می‌دهد (شکل ۲.۵).

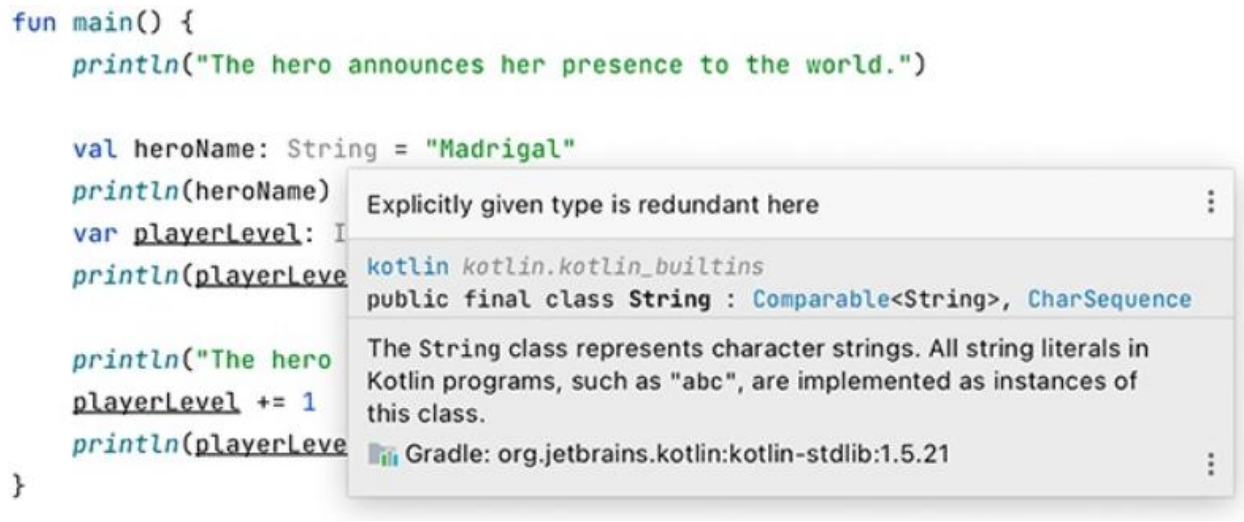


Figure 2.5 Redundant type information

IntelliJ نشان می‌دهد که اعلان نوع شما "اضافی (redundant)" است. این به چه معناست؟

کاتلین شامل ویژگی‌ای به نام استنتاج نوع (type inference) است که به شما اجازه می‌دهد تا تعریف نوع را برای متغیرهایی که در زمان اعلان مقداری به آن‌ها اختصاص می‌یابد، حذف کنید. از آنجایی که شما در هنگام اعلان، داده‌هایی از نوع String به heroName و از نوع Int به playerLevel اختصاص می‌دهید، کامپایلر کاتلین اطلاعات نوع مناسب را برای هر دو متغیر استنتاج (infer) می‌کند.

همانطور که IntelliJ می‌تواند به شما در تغییر یک var به یک val کمک کند، می‌تواند در حذف مشخصات نوعِ غیرضروری نیز یاری‌رسان باشد. روی تعریف نوع String (: String) در کنار heroName کلیک کرده و کلیدهای Option-Return

(Alt-Enter) را فشار دهید. سپس در پنجره‌ی بازشده روی Remove explicit type specification کلیک کنید (شکل ۲.۶).

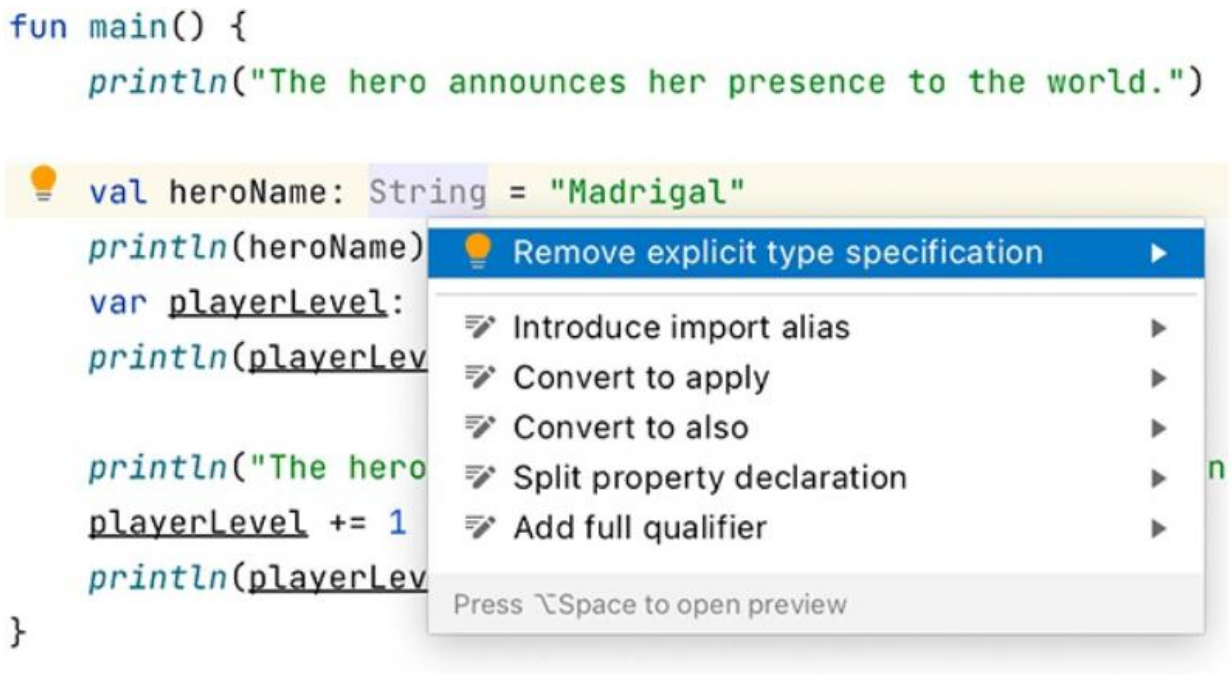


Figure 2.6 Removing explicit type specification

: String ناپدید خواهد شد. این فرآیند را برای متغیر playerLevel تکرار کنید تا : Int حذف شود.

چه از استنتاج نوع استفاده کنید و چه در هنگام اعلان متغیر، نوع آن را مشخص کنید، کامپایلر نوع آن را ردیابی خواهد کرد. در این کتاب، ما در مواردی که ابهامی وجود ندارد از استنتاج نوع استفاده می‌کنیم. استنتاج نوع کمک می‌کند کد تمیز، مختصر و در زمان تغییر برنامه‌ی شما، برای اصلاح آسان‌تر باشد.

به هر حال، IntelliJ در صورت درخواست، نوع هر متغیری را نمایش می‌دهد، از جمله متغیرهایی که از استنتاج نوع استفاده می‌کنند. اگر زمانی درباره‌ی نوع یک متغیر یا عبارت سؤالی داشتید، روی نام آن کلیک کنید یا بخشی از کد خود را برجسته (highlight) کرده و کلیدهای View Type Info (یا میانبر صفحه‌کلید [Ctrl-Shift-P] Control-Shift-P) را فشار دهید. IntelliJ نوع آن را نمایش می‌دهد (شکل ۲.۷).

Figure 2.7 Displaying the type info tooltip

ثابت‌های زمان کامپایل (Compile-Time Constants)

پیش‌تر به شما گفتیم که مقادیر varها می‌توانند تغییر کنند و valها نمی‌توانند. این ... یک دروغ مصلحتی بود. در واقع، موارد خاصی وجود دارد که در آن‌ها یک val می‌تواند مقادیر متفاوتی را بازگرداند، که در فصل ۱۳ در مورد آن‌ها بحث خواهیم کرد. اگر داده‌ای دارید که می‌خواهید کاملاً و قطعاً تغییرناپذیر (immutable) باشد - یعنی هرگز تغییر نکند - استفاده از یک ثابتِ زمان کامپایل (compile-time constant) را در نظر بگیرید.

یک ثابتِ زمان کامپایل باید در خارج از هر تابعی، از جمله main، تعریف شود، زیرا مقدار آن باید در زمان کامپایل (یعنی زمانی که برنامه کامپایل می‌شود) تخصیص یابد - به همین دلیل این نام را دارد. تابع main و سایر توابع شما در طول زمان اجرا (زمان اجرای برنامه) فراخوانی می‌شوند، و مقادیر متغیرهای درون آن‌ها در آن زمان اختصاص می‌یابند. یک ثابت زمان کامپایل پیش از انجام هر یک از این تخصیص‌ها وجود دارد.

ثابت‌های زمان کامپایل همچنین باید یکی از انواع پایه‌ی زیر باشند، زیرا استفاده از انواع پیچیده‌تر برای یک ثابت می‌تولند تضمینِ زمان کامپایل را به خطر بیندازد. در فصل ۱۴ درباره‌ی نحوه‌ی ساخت انواع بیشتر خواهید آموخت. در اینجا انواع پایه‌ی پشتیبانی‌شده برای یک ثابت زمان کامپایل آمده است:

- String
- Short
- Int
- Byte
- Double
- Char
- Float
- Boolean
- Long

نام قهرمان شما (Madrigal) هرگز تغییر نخواهد کرد. Madrigal شخصیت اصلی این بازی است و بازیکنان گزینه‌ای برای تغییر نام او نخواهند داشت. شما می‌توانید این سطح از تغییرناپذیری را در کد خود با استخراج این مقدار به یک ثابت بیان کنید. در Main.kt، متغیر heroName را به بالای اعلان تابع main منتقل کرده و مدیفایر const را اضافه کنید:

Listing 2.10 Declaring a compile-time constant (Main.kt)

```
const val heroName $\equiv$ "Madrigal"

fun main() {
    println("The hero announces her presence to the world.")

    ~~val heroName = "Madrigal"~~
    println(heroName)
    var playerLevel: Int = 4
```

```
    println(playerLevel)

    println("The hero embarks on her journey to locate the enchanted sword.")
    playerLevel + = 1
    println(playerLevel)
}
```

قراردادن پیشوند یک val با مدیفایر const به کامپایلر می‌گوید که می‌تواند مطمئن باشد این val هرگز تغییر نخواهد کرد. در این مورد، تضمین می‌شود که نام قهرمان در هر شرایطی مقدار رشته‌ای "Madrigal" را داشته باشد. این امر به کامپایلر انعطاف‌پذیری لازم برای انجام بهینه‌سازی در پشت صحنه را می‌دهد. کد خود را دوباره اجرا کنید تا مطمئن شوید که خروجی پس از اعمال این تغییر، یکسان است.

در کاتلین، قرارداد این است که از قالب camelCase برای نام متغیرها و قالب تمام‌بزرگ SNAKE_CASE برای نام ثابت‌ها استفاده شود. بنابراین، به جای نام‌گذاری ثابت جدید خود به نام heroName، معمول‌تر این است که آن را HERO_NAME بنامید.

از آنجا که پروژه‌ی شما کوچک است، می‌توانید به راحتی نام را تغییر داده و تنها ارجاع موجود در تابع main خود را به‌روزرسانی کنید. با این حال، در پروژه‌های بزرگ‌تر متوجه خواهید شد که این کار بار بسیار سنگینی است. خوشبختانه، IntelliJ شامل ابزارهای بازآرایی کد (refactoring) است تا تغییرات کدی در سطح پروژه را به نمایندگی از شما انجام دهد.

روی ثابت heroName کلیک راست کرده، سپس Refactor و پس از آن Rename… را انتخاب کنید (شکل ۲.۸).

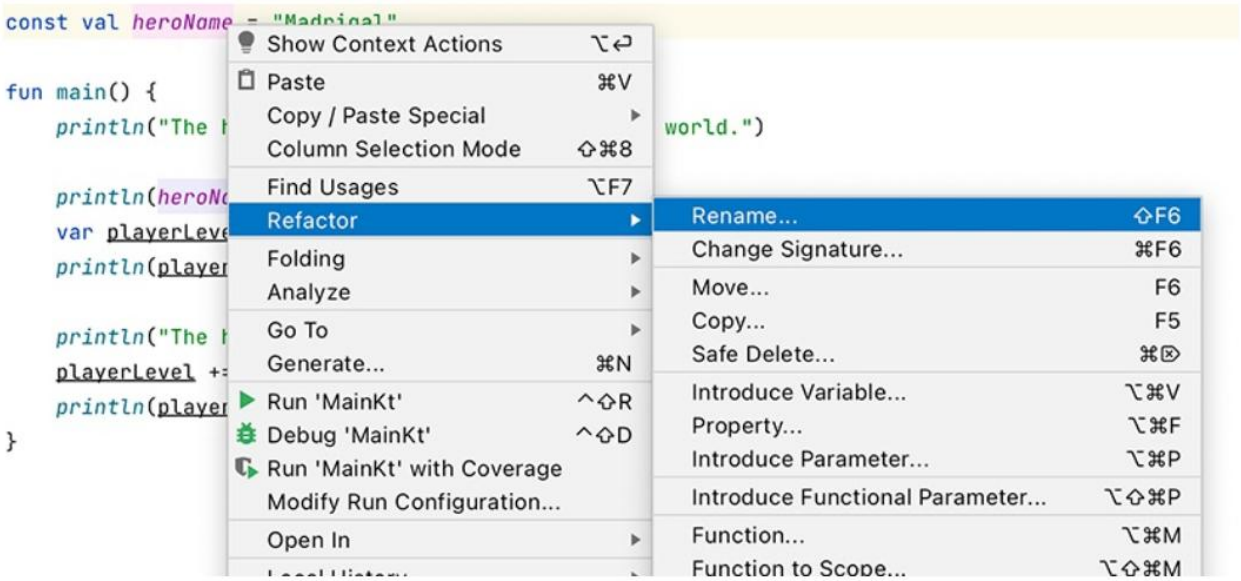


Figure 2.8 The Refactor Rename… menu item

نام ثابت برجسته خواهد شد. HERO_NAME را تایپ کنید تا جایگزین متن برجسته‌شده شود. همان‌طور که تایپ می‌کنید، IntelliJ به برجسته‌سازی ثابت ادامه خواهد داد، همان‌طور که در شکل ۲.۹ نشان داده شده است.

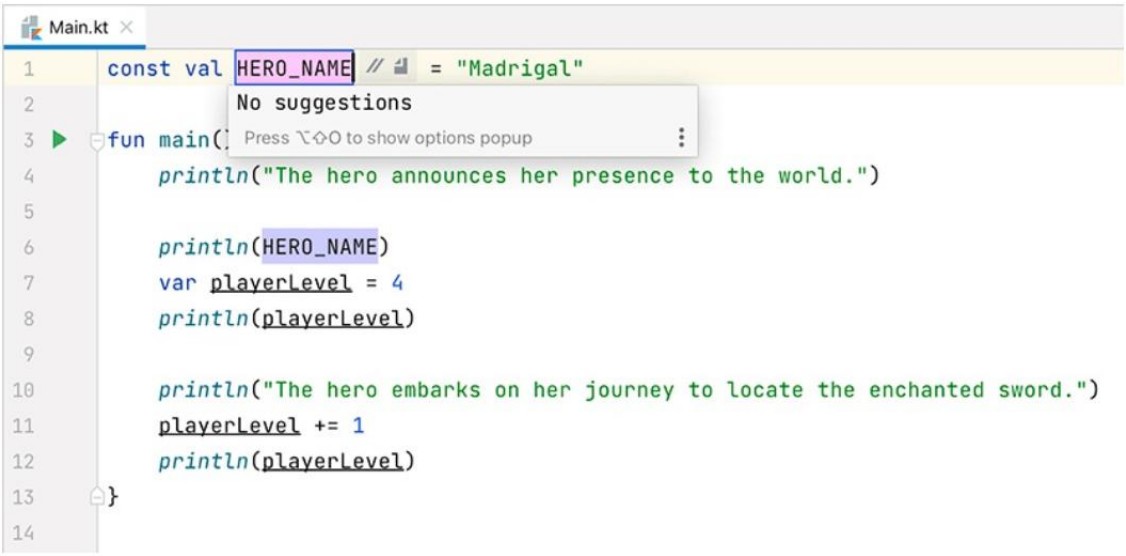


Figure 2.9 Renaming a constant

همان‌طور که تایپ می‌کنید، متوجه خواهید شد که خط println(heroName) با نام جدید به‌روزرسانی می‌شود. IntelliJ تمام کاربردهای این ثابت در پروژه‌ی شما را پیدا کرده و آن‌ها را با نام جدید به‌روزرسانی می‌کند. این تنها بر یک خط کد در bounty-board تأثیر می‌گذارد، اما در یک پروژه‌ی بزرگ IntelliJ می‌تواند به طور خودکار فایل‌های بسیاری را به‌روزرسانی کند.

هنگامی که وارد کردن نام جدید را به پایان رساندید، برای تأیید تغییر کلید Return را فشار دهید.

بررسی بایت‌کد کاتلین (Inspecting Kotlin Bytecode)

در فصل ۱ یاد گرفتید که از کاتلین می‌توان برای نوشتن برنامه‌هایی استفاده کرد که روی JVM، یعنی جایی که بایت‌کد جاوا اجرا می‌شود، قابل اجرا هستند. هنگام هدف‌گذاری JVM، اغلب مفید است که بایت‌کد جاوایی را که کامپایلر کاتلین برای اجرا روی JVM تولید می‌کند، بررسی کنید. شما در چندین جای این کتاب به عنوان راهی برای تحلیل نحوه‌ی عملکرد یک ویژگی خاصِ زبان روی JVM، به بررسی بایت‌کد خواهید پرداخت.

دانستن چگونگی بررسی معادل جاوای کد کاتلینی که می‌نویسید تکنیکی عالی برای درک چگونگی کارکرد کاتلین است، به خصوص اگر تجربه‌ی جاوا داشته باشید. اگر مشخصاً تجربه‌ی جاوا ندارید، کد جاوا احتمالاً ویژگی‌های آشنایی با زبانی که با آن کار کرده‌اید به اشتراک می‌گذارد - بنابراین برای کمک به درک خود، به آن به عنوان یک شبه‌کد (pseudocode) نگاه کنید. و اگر در برنامه‌نویسی کاملاً تازه‌کار هستید - تبریک می‌گوییم! با انتخاب کاتلین، شما زبانی را انتخاب کرده‌اید که، همان‌طور که خواهید دید، به شما اجازه می‌دهد همان منطقی که جاوا بیان می‌کند را معمولاً در کدی بسیار کمتر، بیان نمایید.

برای مثال، ممکن است تعجب کرده باشید که چگونه استفاده از استنتاج نوع هنگام تعریف متغیرها در کاتلین، بر بایت‌کد نهایی تولیدشده برای اجرای روی JVM تأثیر می‌گذارد. برای یافتن پاسخ، می‌توانید از پنجره‌ی ابزار بایت‌کد کاتلین استفاده کنید.

در Main.kt، کلید Shift را دوبار فشـــار دهید تا دیالوگ Search Everywhere باز شـــود. شـــروع به وارد کردن عبارت "show kotlin bytecode" در کادر جســـتجو کرده و هنگام ظاهر شـــدن لیســـت اکشـــن‌های در دســـترس، Show Kotlin Bytecode را انتخاب کنید (شکل ۲.۱۰).

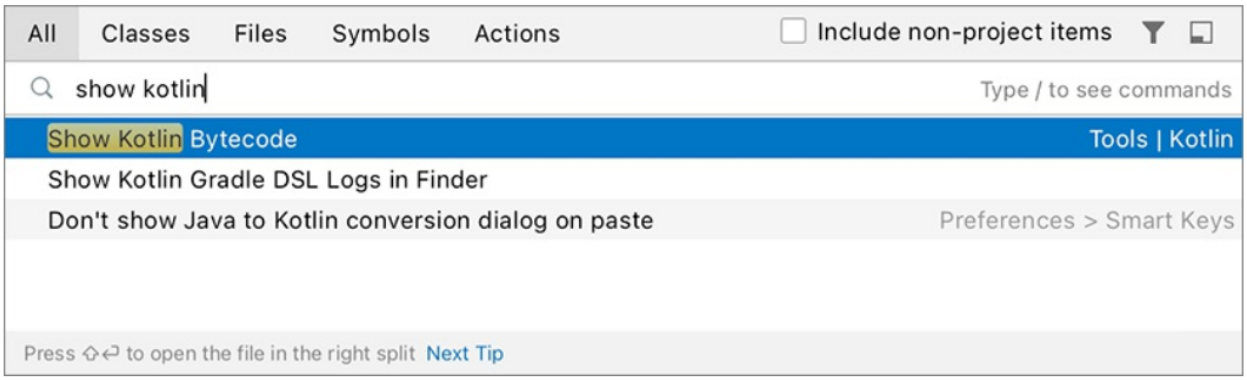


Figure 2.10 Showing Kotlin bytecode

پنجره ابزار بایت‌کد کاتلین باز خواهد شد (شکل ۲.۱۱). (همچنین می‌توانید این پنجره را از طریق Tools سپس Kotlin و بعد Show Kotlin Bytecode باز کنید.)

Figure 2.11 Kotlin bytecode tool window

اگر بایت‌کد زبان مادری شـما نیسـت، نترسـید. می‌توانید بایت‌کد را به جاوا ترجمه کنید (دیکامپایل کنید) تا آن را بر اسـاس اصـطلاحاتی که ممکن اسـت بیشـتر با آن‌ها آشـنا باشـید مشـاهده نمایید. در پنجره ابزار بایت‌کد، روی دکمه‌ی Decompile در گوشه بالا سمت چپ کلیک کنید.

تب جدیدی باز می‌شـود که Main.decompiled.java (شـکل ۲.۱۲) را نشـان می‌دهد، یک نسـخه‌ی جاوایی از بایت‌کدی که کامپایلر کاتلین برای JVM تولید کرده است.

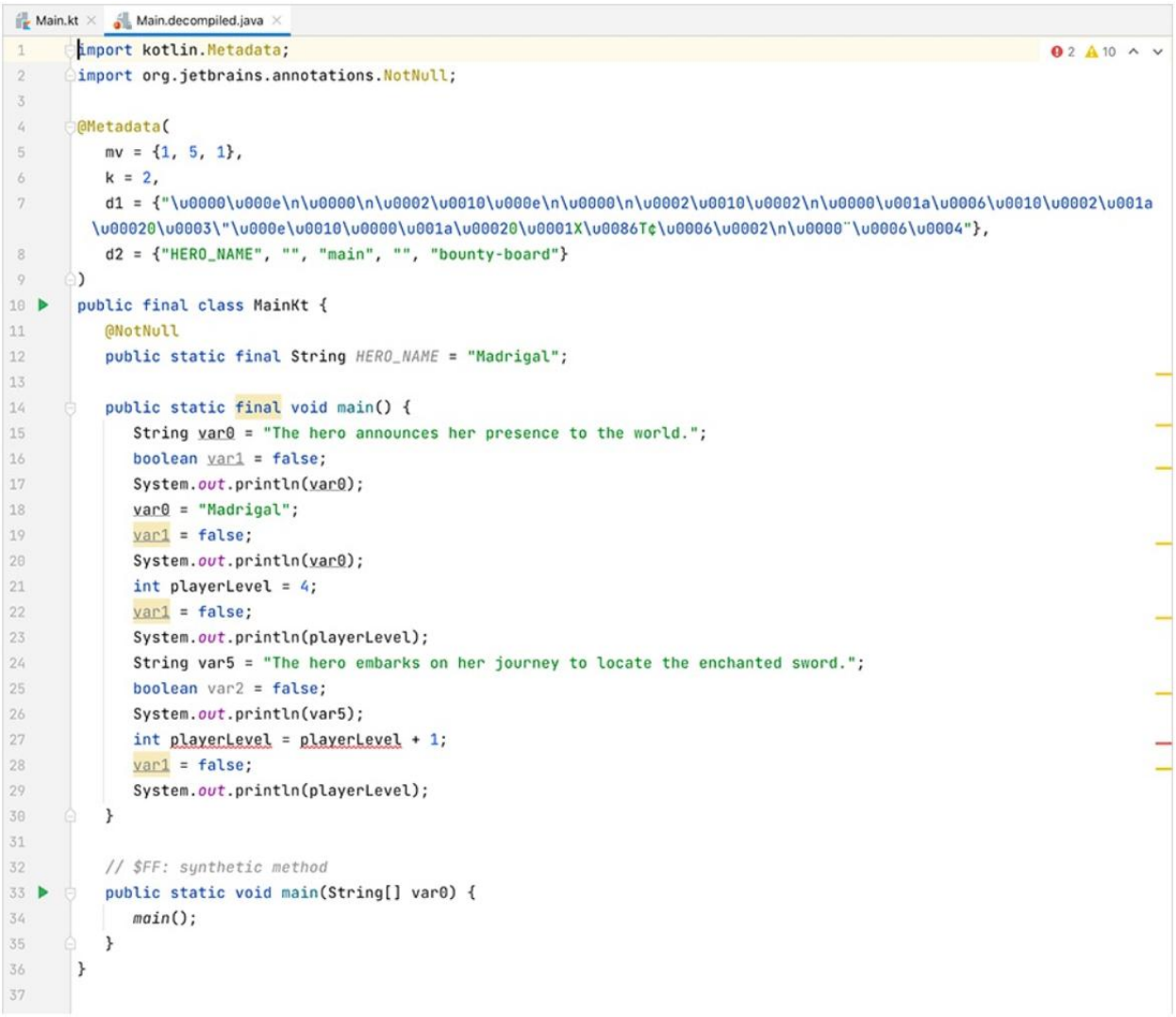


Figure 2.12 Decompiled bytecode

(گاهی اوقات، بازرس بایت‌کد زیر کدهای دیکامپایل‌شـده خطوط قرمزی نمایش می‌دهد. این امر به دلیل یک رفتار خاص در تعامل بین کاتلین و جاواسـت، نه یک مشـکل - شـما با اطمینان می‌توانید هشـدارها و خطاهایی را که در بایت‌کد جاوای تولیدشـده ظاهر می‌شوند نادیده بگیرید.)

اعلان متغیر برای playerLevel را پیدا کنید:

```
int playerLevel = 4 ;
```

با وجود اینکه شما اعلان‌های نوع را از تعاریف هر دو متغیر در کد منبع کاتلین حذف کرده‌اید، بایت‌کدی که تولید شده شامل تعاریف دقیق نوع است. این همان روشی است که متغیرها در جاوا اعلان می‌شوند، و بایت‌کد نگاهی پشت‌صحنه به پشتیبانی استنتاج نوع در کاتلین ارائه می‌دهد.

شما در فصل‌های بعدی عمیق‌تر به بایت‌کد جاوای دیکامپایل‌شده خواهید پرداخت. در حال حاضر، Main.decompiled.java (با استفاده از علامت Xدر تب آن) و پنجره ابزار بایت‌کد (با استفاده از آیکون در گوشه بالا سمت راست) را ببندید.

متأسفانه، این ابزار تنها زمانی کار می‌کند که کد کاتلین شما پلتفرم JVM را هدف قرار داده باشد. کد Kotlin/JS به جاوا اسکریپت ترجمه (transpile) می‌شود، و کد Kotlin/Native به کد ماشین بومی کامپایل می‌گردد. برای هیچ‌کدام از این دو هدف دیگر، بایت‌کدی برای بررسی وجود ندارد. ما شما را تشویق می‌کنیم که در طول این کتاب در زمانی که در حال هدف‌گیری JVM هستید از این ابزار بهره ببرید. این ابزار در حین یادگیری کاتلین می‌تواند بسیار ارزشمند باشد - گرچه ممکن است با راحت‌تر شدن در کار با این زبان، متوجه شوید که کمتر و کمتر از آن استفاده می‌کنید.

در این فصل یاد گرفتید که چگونه داده‌های پایه را در varها و valها ذخیره کنید و بر اساس اینکه آیا نیاز به تغییر مقادیر آن‌ها دارید، دیدید که چه زمانی باید از هر کدام استفاده کنید. شما نحوه‌ی اعلان مقادیر تغییرناپذیر با استفاده از ثابت‌های زمان کامپایل را مشاهده کردید. در نهایت، آموختید که کاتلین چگونه از قدرت استنتاج نوع بهره می‌گیرد تا در هر بار اعلان یک متغیر، در فشردن کلیدها صرفه‌جویی کند. با پیشروی در این کتاب، شما تمامی این ابزارهای پایه را بارها و بارها مورد استفاده قرار خواهید داد.

در فصل بعدی، خواهید آموخت که چگونه حالت‌های پیچیده‌تر را با استفاده از ساختارهای شرطی (conditionals) نمایش دهید.

برای افراد کنجکاوتر: انواع اولیه‌ی جاوا در کاتلین (Java Primitive Types in Kotlin)

در جاوا دو نوع داده وجود دارد: انواع ارجاعی (reference types) و انواع اولیه (primitive types). انواع ارجاعی یک تعریف معادل در کد منبع دارند. برخی از انواع ارجاعی با عنوان انواع "بسته‌بندی‌شده (boxed)" یا "شیء (object)" نیز شناخته می‌شوند. جاوا همچنین انواع اولیه (که اغلب فقط "اولیه‌ها" نامیده می‌شوند) را ارائه می‌دهد که هیچ فایل تعریف منبعی نداشته و در عوض با کلمات کلیدی خاصی نشان داده می‌شوند.

نام‌های انواع ارجاعی در جاوا با یک حرف بزرگ آغاز می‌شوند تا نشان دهند که توسط یک تعریف در منبع پشتیبانی می‌شوند. در اینجا playerLevel با استفاده از یک نوع ارجاعی جاوا تعریف شده است:

```
Integer playerLevel = 4 ;
```

نام انواع اولیه‌ی جاوا با یک حرف کوچک آغاز می‌شوند:

```
int playerLevel = 4 ;
```

تمام انواع اولیه در جاوا یک نوع ارجاعی متناظر دارند. (اما همه‌ی انواع ارجاعی دارای یک نوع اولیه‌ی متناظر نیستند.) چرا باید از یکی در مقابل دیگری استفاده کرد؟

یکی از دلایل انتخاب نوع ارجاعی این است که برخی ویژگی‌های زبان جاوا تنها در زمان استفاده از انواع ارجاعی در دسترس هستند. برای مثال، Genericها که در فصل ۱۸ درباره‌ی آن‌ها خواهید آموخت، با انواع اولیه کار نمی‌کنند. انواع ارجاعی همچنین می‌توانند راحت‌تر با ویژگی‌های شیءگرایی جاوا کار کنند. (شما درباره‌ی برنامه‌نویسی شیءگرا و ویژگی‌های شیءگرای کاتلین در فصل ۱۳ خواهید آموخت.)

از طرف دیگر، انواع اولیه عملکرد بهتری داشته و امتیازات دیگری نیز به همراه دارند.

برخلاف جاوا، کاتلین تنها یک نوع داده ارائه می‌دهد: انواع ارجاعی.

```
var playerLevel: Int = 4
```

کاتلین این تصمیم طراحی را به چند دلیل اتخاذ کرده است. اول اینکه، اگر انتخابی بین انواع مختلف داده‌ها وجود نداشته باشد، به راحتی زمانی که چندین گزینه برای انتخاب دارید در بن‌بستِ کدنویسی قرار نخواهید گرفت. برای مثال، چه می‌شود اگر شما نمونه‌ای از یک نوع اولیه را تعریف کنید، و بعداً متوجه شوید که به ویژگی generic نیاز دارید که نیازمند یک نوع ارجاعی است؟ داشتن تنها انواع ارجاعی در کاتلین به این معناست که شما هرگز با این مشکل روبرو نخواهید شد.

اگر با جاوا آشنایی دارید، ممکن است پیش خود فکر کنید: "اما انواع اولیه نسبت به انواع ارجاعی عملکرد بهتری دارند." این موضوع صحت دارد. اما نگاهی دوباره به متغیر playerLevel در بایت‌کد دیکامپایل‌شده‌ای که پیش‌تر دیدید بیندازید:

```
int playerLevel = 4
```

همان‌طور که حروف کوچک int نشان می‌دهد، یک نوع اولیه به جای نوع ارجاعی مورد استفاده قرار گرفته است. چرا این‌گونه است، در حالی که کاتلین تنها انواع ارجاعی دارد؟ کامپایلر کاتلین در صورت امکان، از انواع اولیه در بایت‌کد جاوا استفاده خواهد کرد، زیرا آن‌ها در واقع عملکرد بهتری را ارائه می‌دهند.

کاتلین سهولت انواع ارجاعی را به همراه عملکرد انواع اولیه در پشت صحنه به شما می‌بخشد. در کاتلین، یک نوع ارجاعی متناظر برای هشت نوع اولیه‌ای که ممکن است در جاوا با آن‌ها آشنا باشید، خواهید یافت.

چالش: hasSteed

در فصل ۱، چالش شما این بود که برخی از عملیات‌های ریاضی را در Kotlin REPL امتحان کنید. بیشتر چالش‌های بقیه‌ی کتاب مبتنی بر پروژه‌ای هستند که در حال کار روی آن هستید – در این مورد، پروژه‌ی bounty-board. پیش از شروع یک چالش، یک کپی از پروژه‌ی خود تهیه کرده و در آن نسخه کپی به چالش بپردازید. بسیاری از فصل‌ها بر پایه‌ی فصل‌های قبلی ساخته می‌شوند و کار روی چالش‌ها در نسخه‌ای کپی از پروژه تضمین می‌کند که قادر خواهید بود به پیشروی در کتاب ادامه دهید.

اینجا اولین چالش شما مبتنی بر bounty-board قرار دارد: در بازی ماجراجویی متنی، بازیکنان می‌توانند یک اژدها یا مینوتور (minotaur) را برای سواری به دست آورند. متغیری به نام hasSteed تعریف کنید تا مشخص نماید آیا بازیکن یکی را به دست آورده است یا خیر. به این متغیر یک وضعیت اولیه اختصاص دهید که نشان دهد بازیکن هنوز هیچ حیوانی برای سواری ندارد.

به یاد داشته باشید پیش از ایجاد این تغییرات یک کپی از bounty-board تهیه کنید. فصل بعدی به توسعه‌ی bounty-board ادامه خواهد داد و انتظار ندارد بازیکنان شما حیوان سواری داشته باشند.

چالش: شاخ تک‌شاخ (The Unicorn's Horn)

این صحنه از بازی ماجراجویی را تصور کنید:

قهرمان داستان، Madrigal به یک می‌خانه معروف به The Unicorn's Horn می‌رسد. صاحب می‌خانه می‌پرسد: "آیا برای اسب خود به اصطبل نیاز داری؟"

Madrigal پاسخ می‌دهد: "نه، من هیچ حیوان سواری ندارم. اما ۵۰ سکه‌ی طلا دارم و مایلم چیزی بنوشم."

صاحب می‌خانه می‌گوید: "عالی است. من نوشیدنی عسل (mead)، شراب (wine) و لا کروا (LaCroix) دارم. چه چیزی میل داری؟"

برای این چالش، متغیرهای مورد نیاز این صحنه در The Unicorn's Horn را زیر متغیر hasSteed خود اضافه کنید، و تا حد امکان از استنتاج نوع و مقادیر تخصیص‌یافته بهره ببرید. متغیرهایی را اضافه کنید تا مقادیرِ نام می‌خانه، نام صاحب می‌خانه‌ی فعلی که شیفت است، و مقدار طلایی که بازیکن تاکنون به دست آورده را نگه دارند.

توجه کنید که The Unicorn's Horn منویی از نوشیدنی‌ها دارد که قهرمان می‌تواند از آن‌ها انتخاب کند. برای نمایش این منو ممکن است از چه نوعی استفاده کنید؟ در صورت نیاز، به جدول ۲.۱ مراجعه کنید.

چالش: آینه‌ی جادویی (Magic Mirror)

با تجدید قوا، Madrigal برای یک مأموریت چالش‌برانگیز آماده است. شما چطور؟

قهرمان یک آینه‌ی جادویی پیدا می‌کند که بازتاب HERO_NAME بازیکن را به وی نشان می‌دهد. با استفاده از جادوی نوع String، رشته‌ی HERO_NAME که "Madrigal" است را به "lagirdaM", که بازتابی از مقدار آن است، تبدیل کنید.

برای حل این چالش، با مستنداتِ String در آدرس kotlinlang.org/api/latest/jvm/ stdlib/kotlin/-string/index.html مشورت کنید. خواهید دید که خوشبختلنه، اقداماتی که یک نوع خاص می‌تولند انجام دهد معمولاً به شیوه‌ای بسیار شهودی نام‌گذاری شده‌اند (یک راهنمایی).

https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-string/index.html