کاتلین به زبان ساده

راه‌های ارتباط با نویسنده

وب سایت: www.w3-farsi.com

لینک تلگرام: https://telegram.me/ebrahimi_younes

ID تلگرام: @ebrahimi_younes

پست الکترونیکی: younes.ebrahimi.1391@gmail.com

تقدیم به

همسر و پسر عزیزم

کاتلین چیست..... ۶
JDK و ntelliJ IDEA ۸
نصب JDK وIntelliJ IDEA ۱۰
ساخت یک برنامه ساده درKotlin ۲۴
کاراکترهای کنترلی..... ۳۷
توضیحات..... ۴۰
متغیر ۴۱
انواع داده..... ۴۳
استفاده از متغیرها ۴۴
تبدیل انواع داده ۵۰
عبارات و عملگرها..... ۵۳
عملگرهای ریاضی..... ۵۴
عملگرهای تخصیصی ۵۷
عملگرهای مقایسه‌ای ۵۹
عملگرهای بیتی..... ۶۰
گرفتن ورودی از کاربر..... ۶۷
ساختارهای تصمیم..... ۷۱
دستورif ۷۲
دستورif...else ۷۴
دستور if تو در تو ۷۶
دستور if چندگانه ۷۸
استفاده از عملگرهای منطقی..... ۷۹
دستورwhen ۸۳
تکرار..... ۸۶
حلقهWhile ۸۷
حلقهdo While ۸۹

حلقهfor ۹۰
خارج شدن از حلقه با استفاده از break وcontinue ۹۲
آرایه‌ها ۹۳
آرایه‌های چند بعدی ۹۸
آرایه دندانه دار ۱۰۴
تابع ۱۰۷
مقدار برگشتی از یک تابع ۱۰۹
پارامتر و آرگومان ۱۱۳
ارسال آرگومان به روش مقدار ۱۱۶
ارسال آرایه به عنوان آرگومان ۱۱۷
محدوده متغیر ۱۲۰
آرگومان‌های متغیر(VarArgs) ۱۲۰
سربارگذاری توابع ۱۲۲
بازگشت(Recursion) ۱۲۳
برنامه نویسی شیء گرا (OOP) ۱۲۶
کلاس ۱۲۷
سازنده ۱۳۰
سطح دسترسی ۱۳۷
کپسوله سازی (Encapsulation) ۱۳۹
وراثت ۱۴۰
خواص (Properties) ۱۴۳
سطح دسترسیProtect ۱۴۷
Companion Object ۱۴۹
Override ۱۵۱
عملگرis ۱۵۳
رابط(Interface) ۱۵۴
کلاسهای انتزاعی(Abstract Class) ۱۶۰

ایجاد آرایه‌ای از کلاس‌ها ۱۶۲
عبارات لامبدا ۱۶۳
Data Class ۱۶۴
توابع توسعه یافته ۱۶۹
مدیریت استثناءها و خطایابی ۱۷۰
استثناءهای اداره نشده ۱۷۱
دستور try و catch ۱۷۲
بلوک finally ۱۷۶
ایجاد استثناء ۱۷۸
تعریف یک استثناء توسط کاربر ۱۷۹
جنریک ها (Generics) ۱۸۱
توابع جنریک ۱۸۱
کلاس جنریک ۱۸۴

کاتلین چیست

کاتلین یک زبان برنامه نویسی جدید و قدرتمند است که توسط شرکت JetBrains ساخته شده است و یک زبان منبع باز محسوب می‌شود که ویژگی‌های برنامه نویسی شیء گرا و ماژولار را با یک دیگر ترکیب کرده است. نام کاتلین برگرفته از جزیره‌ای به همین نام در اطراف سن پتزربوگ گرفته شده است. کاتلین از سال ۲۰۱۱ معرفی شده و در حقیقت زبان جدیدی محسوب نمی‌شود اما پس از آنکه در کنفرانس Google I/O در سال ۲۰۱۷ به عنوان یک زبان رسمی برای توسعه اندروید معرفی شد، باعث شد کاتلین بر سر زبان‌ها بیفتد و افراد زیادی سراغ یادگیری آن بروند. در ابتدا برخی از مزایای کاتلین را بررسی می‌کنیم.

مزایا

- یادگیری کاتلین بسیار ساده است. کاتلین یک زبان برنامه نویسی functional بوده که یادگیری آن به سادگی امکان پذیر است. ساختار کاتلین بسیار شبیه به زبان جاواست که این موضوع به خصوص برای برنامه نویسان جاوا خبر خوبی است، زیرا می‌توانند به سرعت آن را فراگرفته و در پروژه‌های بعدی خود از آن استفاده کنند. استفاده از کاتلین می‌تواند کد شما را خواناتر و قابل فهم تر کند.
- کاتلین می‌تواند به JavaScript یا JVM bytecode کامپایل شود. دقیقاً مشابه جاوا، کاتلین نیز می‌تواند به فرمت bytecode کامپایل شود. روش کار به این صورت است که ابتدا کد ما به بایت کد که یک زبان قابل فهم برای ماشین مجازی جاوا (JVM) است کامپایل شده و سپس JVM برنامه را در ماشین مقصد اجرا می‌کند. با این کار برنامه می‌تواند مستقل از سیستم عامل اجرا شود.
- کاتلین می‌تواند از تمام کتابخانه‌ها و فرم ورک های جاوا استفاده کند: شما می‌توانید از تمام کتابخانه‌ها و فرم ورک های زبان جاوا در کاتلین استفاده کنید که این یکی از مهم‌ترین ویژگی‌های کاتلین است.

- جاوا به صورت خودکار قابل تبدیل به کاتلین است: ابزارهایی وجود دارد که می‌توانید به صورت خودکار کدهای جاوای خود را به کاتلین تبدیل کنید و به این ترتیب در وقت خود صرفه جویی کنید.
- بازبینی کدها دیگر کار مشکلی نخواهد بود: کاتلین تمرکز زیادی بر روی خوانایی و قابل فهم بودن کدها دارد که این موضوع فرآیند بازبینی کدها را ساده می‌کند، بنابراین همه اعضای تیم می‌توانند کدها را بازبینی کنند حتی اگر با این زبان آشنایی نداشته باشند.

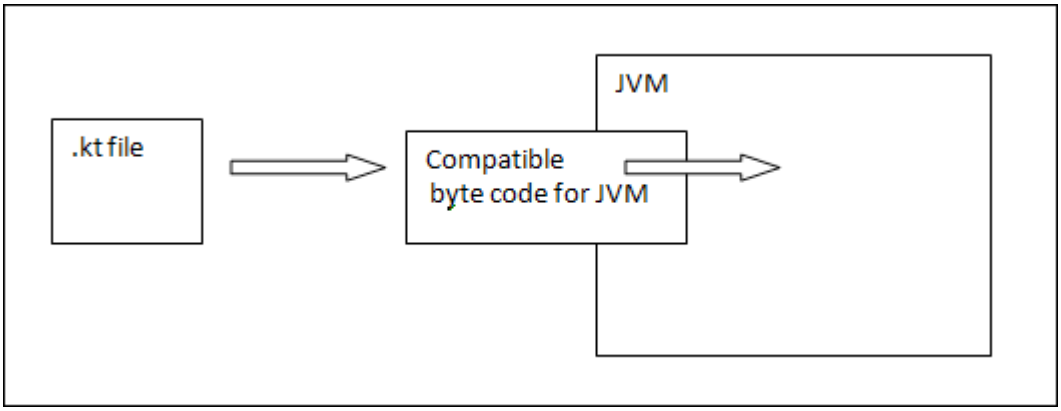
معایب

- عدم پشتیبانی از برخی ویژگی‌های زبان جاوا: کاتلین از برخی ویژگی‌های زبان جاوا نظیر انواع `Static Primitive Type`، ها یا انواع داده اولیه، عملگر شرطی سه تایی و ... پشتیبانی نمی‌کند که البته تعداد آن‌ها زیاد نیست.
- سرعت کامپایل: در برخی موارد کاتلین سریع‌تر از جاوا عمل می‌کند اما گاهی اوقات نیز به شکل قابل ملاحظه‌ای کندتر عمل می‌کند.

معماری کاتلین

کاتلین، مانند هر زبان دیگر، معماری خاص خودش را برای اختصاص حافظه و تولید خروجی با کیفیت برای کاربر نهایی دارد. کامپایلر کاتلین، سناریوهای متفاوتی را برای زبان‌های مقصد متفاوت از جمله جاوا یا جاوا اسکریپت دارد.

کامپایلر کاتلین یک بایت کد برای اجرا در `JVM` تولید می‌کند که دقیقاً همان بایت کدی است که برای یک فایل جاوا با پسوند `class`. تولید می‌شود. به عبارت دیگر هر دو بایت کدِ تولید شده، بر روی `JMV` اجرا می‌شوند و این بدین معنی است که یک قابلیت تعاملی بین جاوا و کاتلین وجود دارد.



در صورتی که مقصد کاتلین زبان جاوا اسکریپت باشد، کامپایلر فایل kt را به ES5.1 تبدیل می‌کند و یک کد سازگار با جاوا اسکریپت تولید می‌شود. به صورت پیش فرض کاتلین مانند جاوا با استفاده از JVM اجرا می‌شود و وابسته به آن است اما با قابلیت Kotlin/Native، کامپایلر کاتلین می‌تواند با استفاده از LLVM، برنامه‌های نوشته شده با کاتلین را بدون نیاز به ماشین مجازی نیز اجرا کند. در حال حاضر با توجه به مستندات JetBrains، ماشین‌های مقصد زیر پشتیبانی می‌شوند:

- Mac OS X 10.10 and later (x86-64)
- x86-64 Ubuntu Linux (14.04, 16.04 and later), other Linux flavours may work as well
- Apple iOS (arm64), cross-compiled on MacOS X host
- Raspberry Pi, cross-compiled on Linux host

JDK و IntelliJ IDEA

IntelliJ IDEA محیط توسعه یکپارچه‌ای است که، دارای ابزارهایی برای کمک به شما برای توسعه برنامه‌های Kotlin می‌باشد. توصیه می‌کنیم که از محیط IntelliJ IDEA برای ساخت برنامه استفاده کنید، چون این

محیط دارای ویژگی‌های زیادی برای کمک به شما جهت توسعه برنامه‌های Kotlin می‌باشد. توسط IntelliJ IDEA می‌توان در استانداردهای مختلف جاوا مانند J2SE، J2EE و J2ME برنامه نویسی کرد. همچنین از محیط زبان‌های PHP، HTML، C و نیز Groovy پشتیبانی می‌کند.

JDK که مخفف عبارت Java Development Toolkit می‌باشد ترکیبی از کامپایلر زبان جاوا، کلاس‌های کتابخانه‌ای(Java Class Libraries) و JVM و فایل‌های راهنمای آن‌ها می‌باشد. برای اینکه ما بتوانیم با استفاده از زبان برنامه نویسی Kotlin، برنامه بنویسیم به این مجموعه نیاز داریم. تعداد زیادی از پردازش‌ها که وقت شما را هدر می‌دهند به صورت خودکار توسط IntelliJ IDEA انجام می‌شوند. یکی از این ویژگی‌ها اینتلی سنس (Intellisense) است که شما را در تایپ سریع کدهایتان کمک می‌کند.

IntelliJ IDEA برنامه شما را خطایابی می‌کند و حتی خطاهای کوچک (مانند بزرگ یا کوچک نوشتن حروف) را برطرف می‌کند. با این برنامه‌های قدرتمند بازدهی شما افزایش می‌یابد و در وقت شما با وجود این ویژگیهای شگفت انگیز صرفه جویی می‌شود. IntelliJ IDEA رایگان است و می‌توان آن را دانلود و از آن استفاده کرد. این برنامه ویژگیهای کافی را برای شروع برنامه نویسی Kotlin در اختیار شما قرار می‌دهد. در آموزش‌ها از IntelliJ IDEA نسخه ۲۰۱۸ استفاده شده است و استفاده از این نسخه برای انجام تمرینات این سایت کافی می‌باشد. برای دانلود نرم افزارهای مورد نیاز بر روی لینک زیر کلیک کنید:

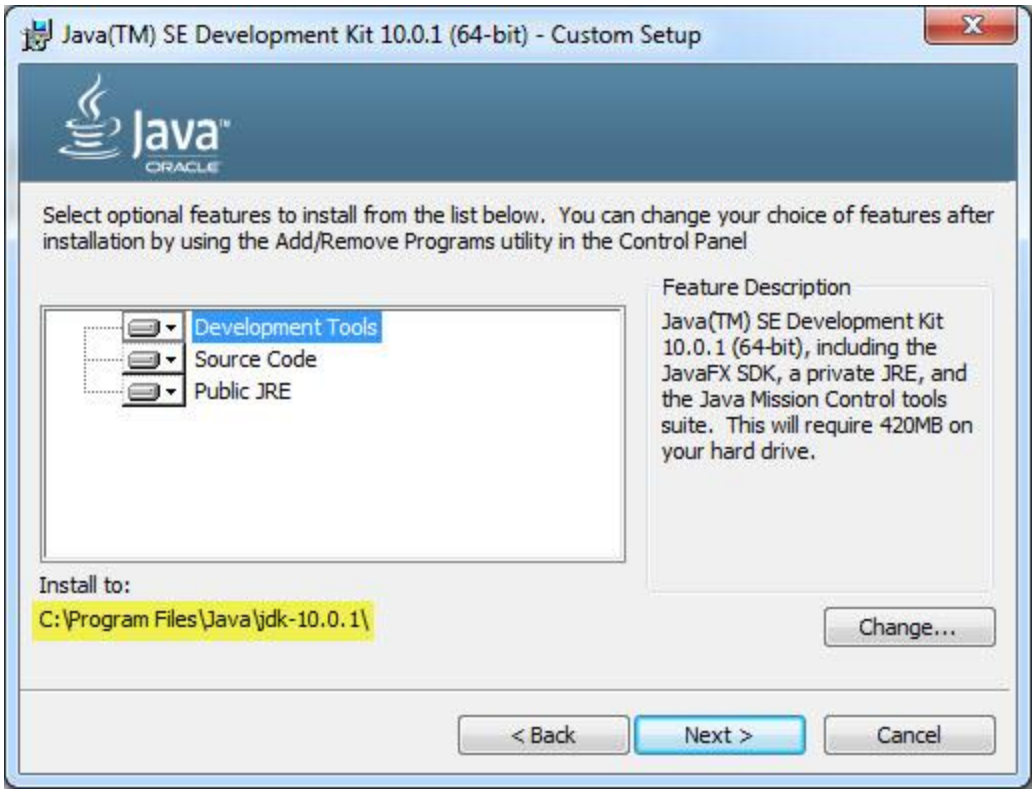
```
http://dl.w3-farsi.com/Software/Kotlin/Intellij-IDEA-and-JDK.rar
```

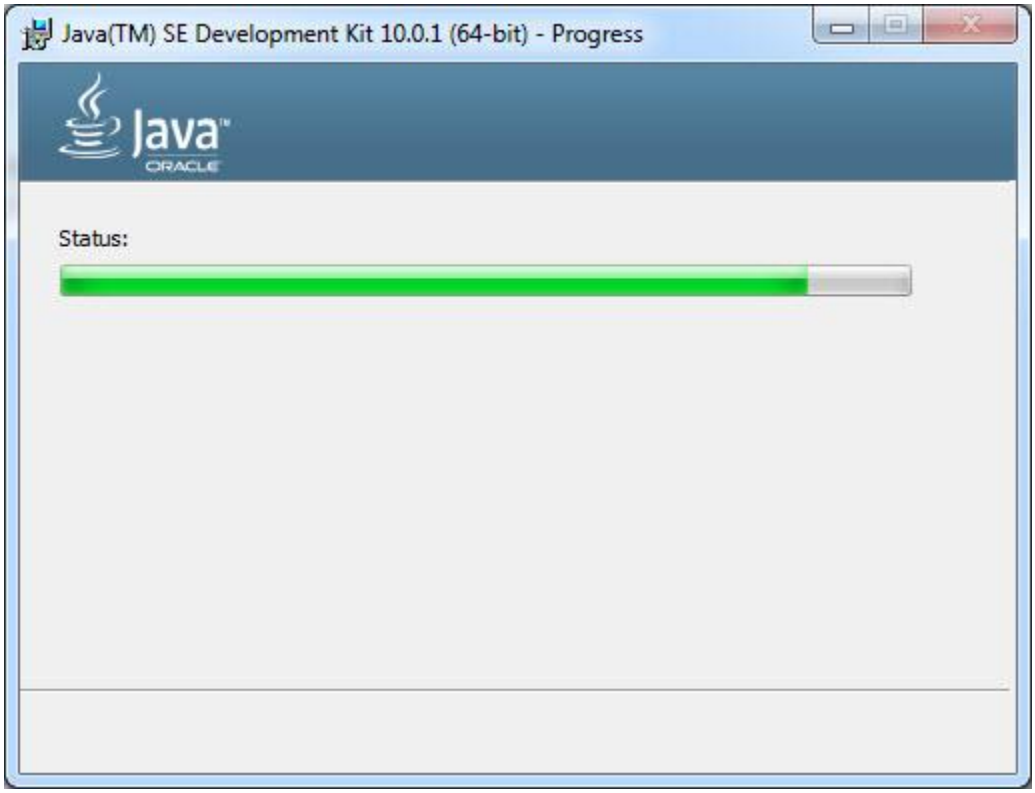
در درس آینده مراحل نصب و راه اندازی دو نرم افزار JDK و IntelliJ IDEA را توضیح می‌دهیم.

نصب JDK و IntelliJ IDEA

در درس قبل در مورد نرم افزارهای IntelliJ IDEA و JDK توضیحات مختصری ارائه دادیم. در این درس می‌خواهیم شما را با نحوه نصب این دو نرم افزار آشنا کنیم. نصب این نرم افزارها مانند اکثر نرم افزارهای دیگر بسیار آسان بود و بعد از زدن چند دکمه Next نصب می‌شوند. در لینکی که در درس قبل دانلود کردیه اید، دو نسخه ۳۲ و ۶۴ بیت از JDK وجود دارد که شما بسته به نوع ویندوز خود باید یکی از این دو را نصب کنید. در این درس چون که ویندوز ما ۶۴ بیت است، پس مراحل نصب نسخه ۶۴ بیتی JDK را به شما آموزش می‌دهیم. البته مراحل نصب هر دو نسخه دقیقاً شبیه هم است. در در زیر مراحل تصویری نصب این دو نرم افزار نشان داده شده است.

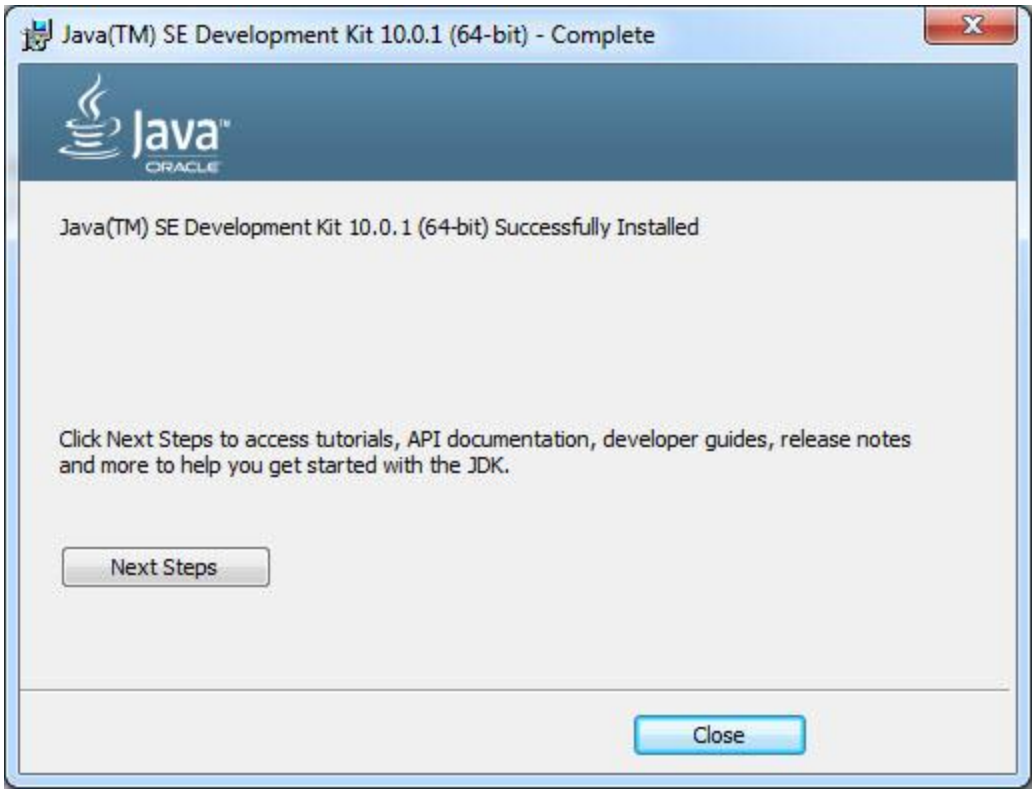
نصب JDK


Java(TM) SE Development Kit 10.0.1 (64-bit) - Custom Setup
Java
ORACLE
Select optional features to install from the list below. You can change your choice of features after installation by using the Add/Remove Programs utility in the Control Panel
Development Tools
Source Code
Public JRE
Feature Description
Java(TM) SE Development Kit 10.0.1 (64-bit), including the JavaFX SDK, a private JRE, and the Java Mission Control tools suite. This will require 420MB on your hard drive.
Install to:
C:\Program Files\Java\jdk-10.0.1\
Change...
< Back
Next >
Cancel


Java(TM) SE Development Kit 10.0.1 (64-bit) - Progress
Java
ORACLE
Status:

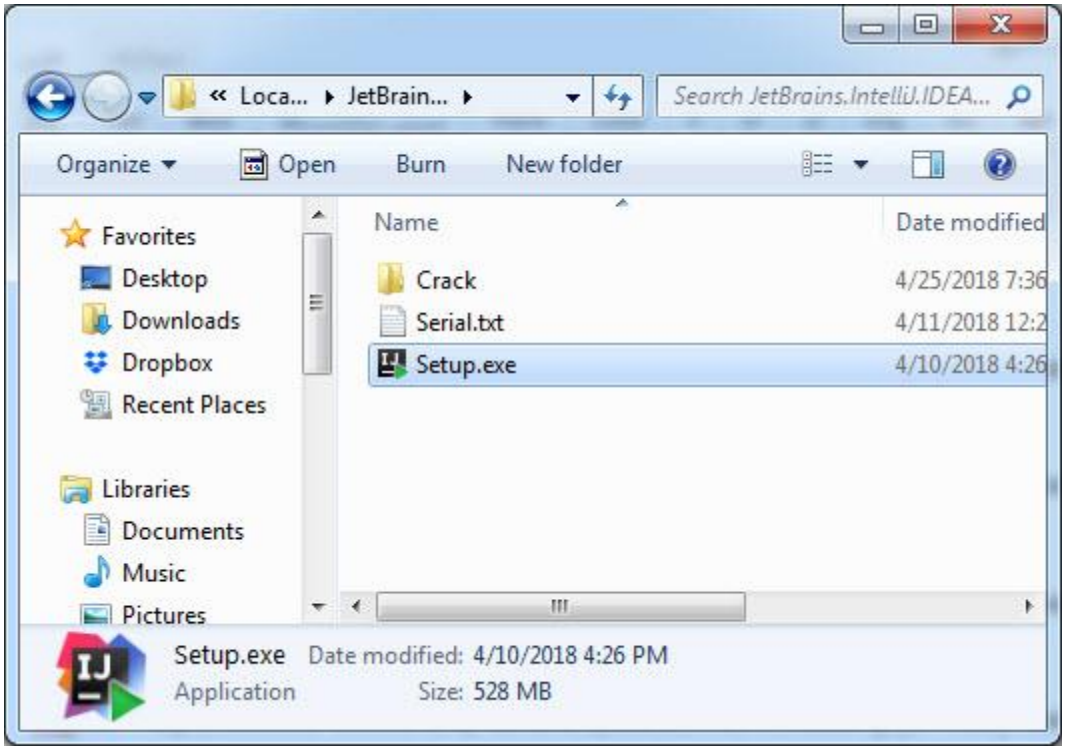

Java Setup - Custom Setup
Java
ORACLE
Custom Setup
Click "Change" to install Java to a different folder.
Install to:
C:\Program Files\Java\jre-10.0.1
Change...
Enable Java Content in the Browser
Next >


Java Setup - Progress
Java
ORACLE
Status: Installing Java
3 Billion
Devices Run Java
Java
#1 Development Platform
ORACLE



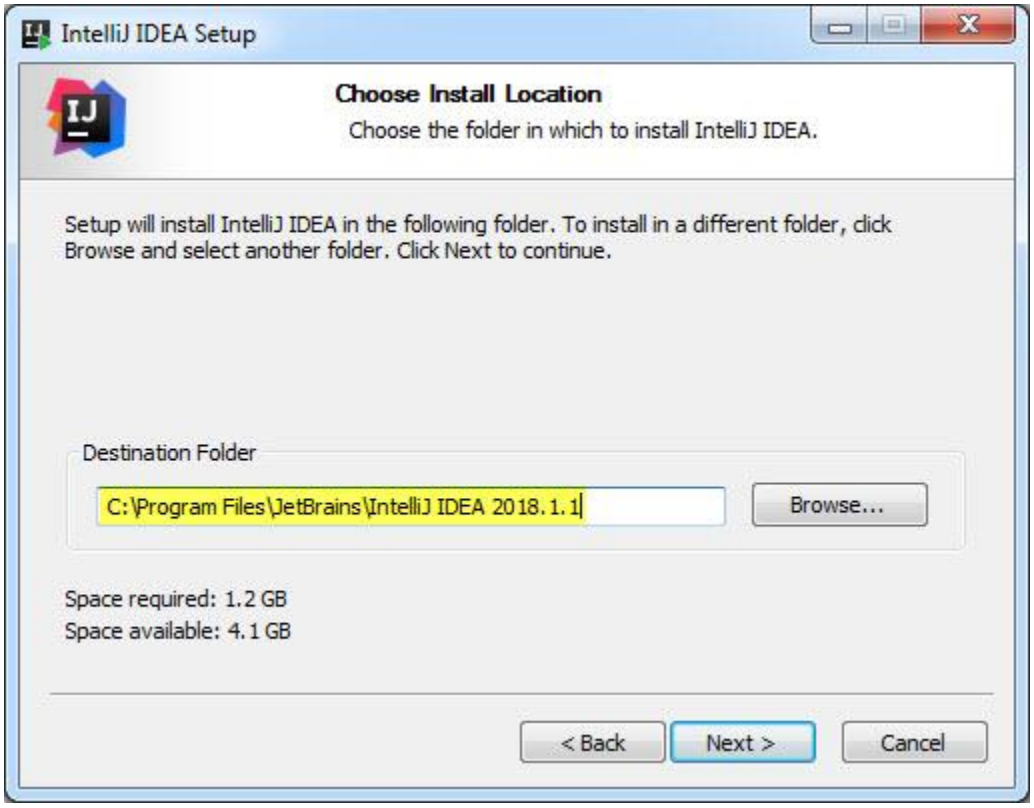
نصب IntelliJ IDEA

نصب IntelliJ IDEA کمی نیاز به دقت دارد. ابتدا به پوشه نرم افزار بروید:

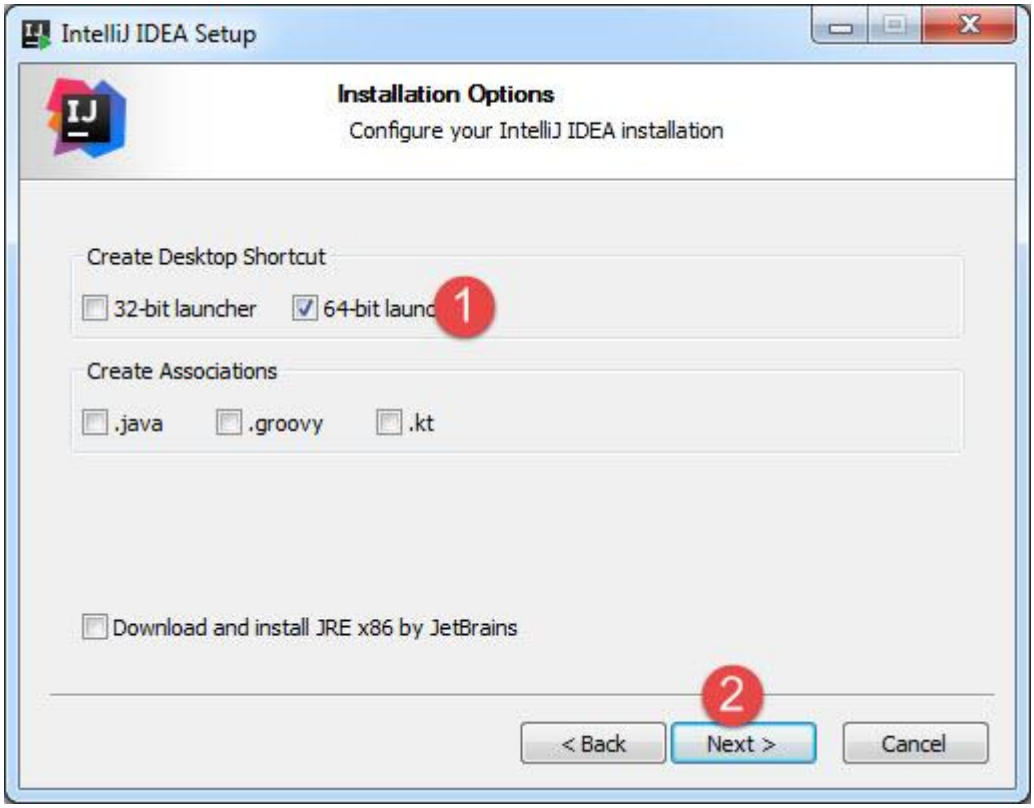


فایل `Setup.exe` را اجرا کرده تا وارد مراحل نصب نرم افزار شوید. در شکل زیر دکمه `Next` را کلیک کنید:

در صفحه زیر مسیری که قرار است برنامه نصب شود، نشان داده شده است. این مسیر ممکن است در کامپیوتر شما متفاوت باشد. در هر صورت این مسیر مهم است، و در مراحل بعدی با آن سر و کار داریم. آن را دستکاری نکنید و دکمه Next را بزنید:

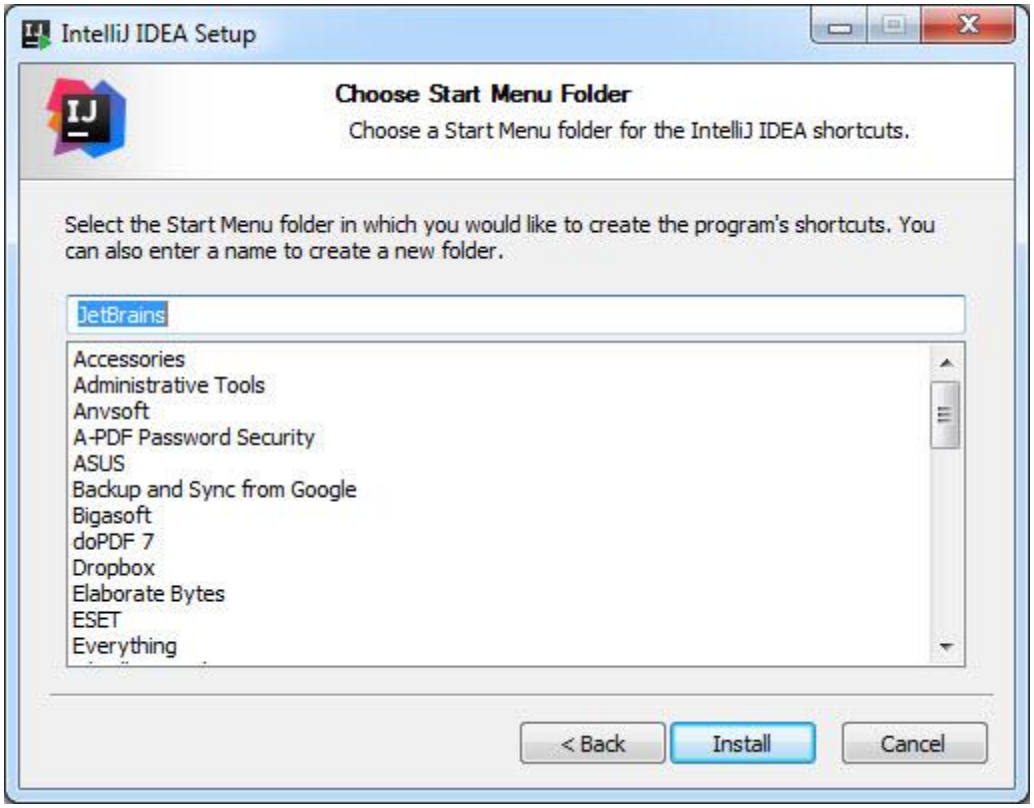


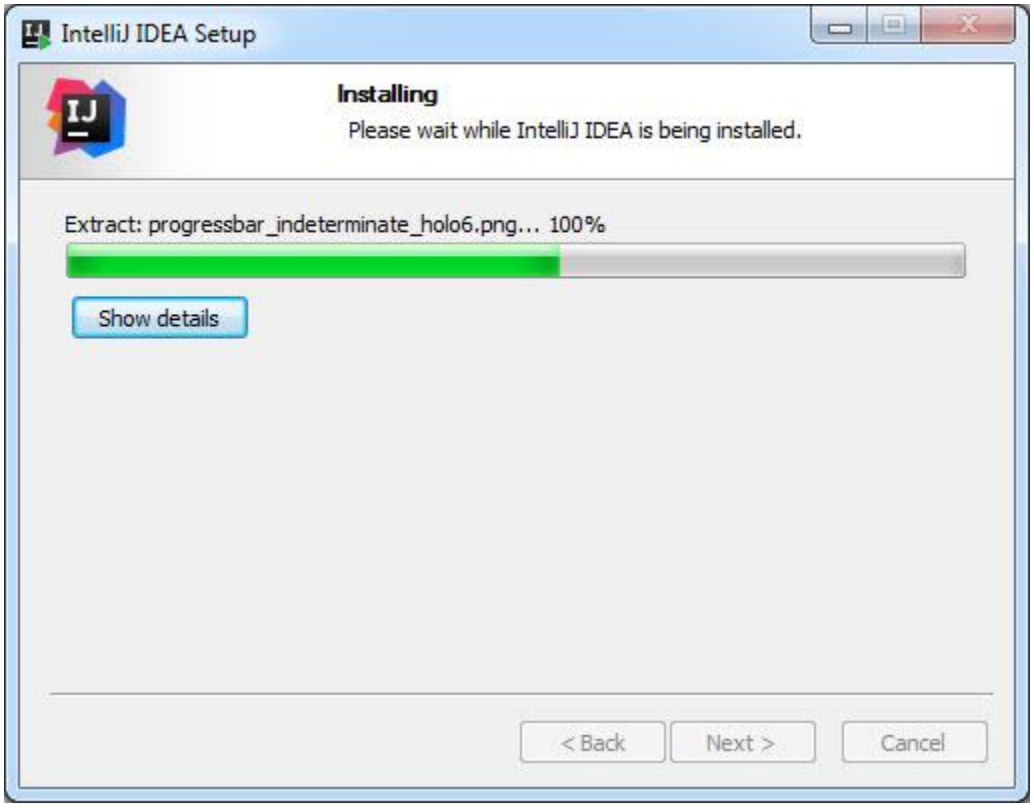
در صفحه زیر بسته به اینکه ویندوزتان ۳۲ یا ۶۴ بیتی است، گزینه مرتبط را تیک زده و سپس دکمه Next را بزنید:



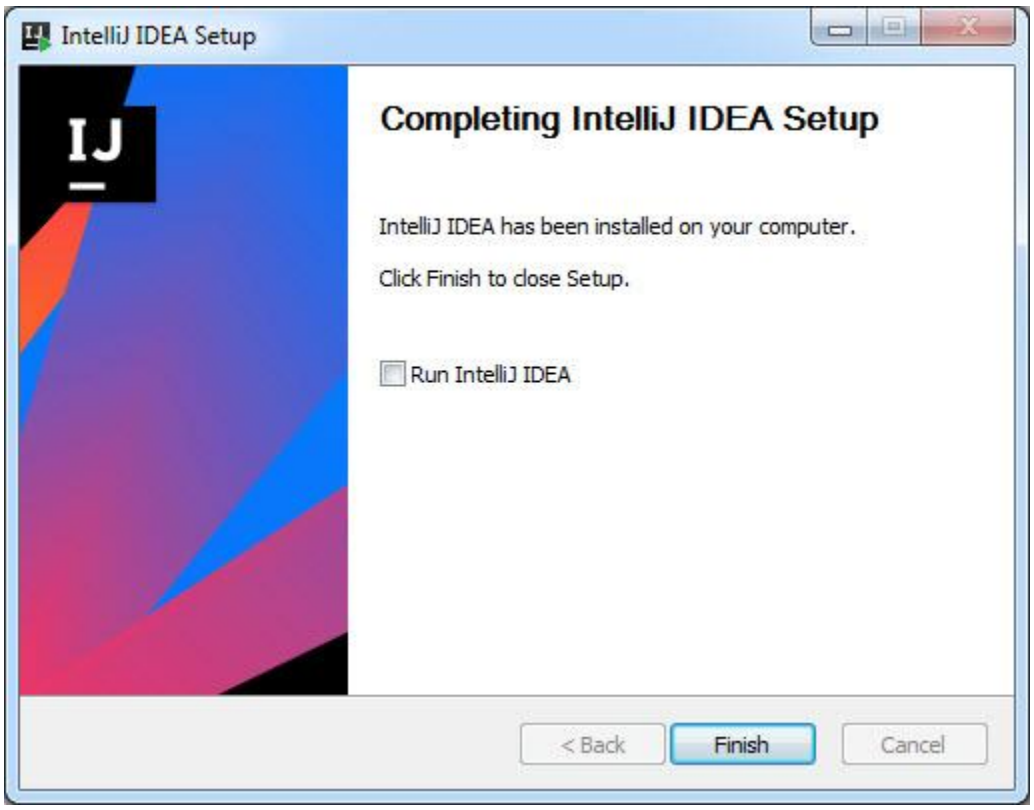
۳

در شکل زیر دکمه `Install` را زده تا وارد مرحله پایانی نصب شوید.

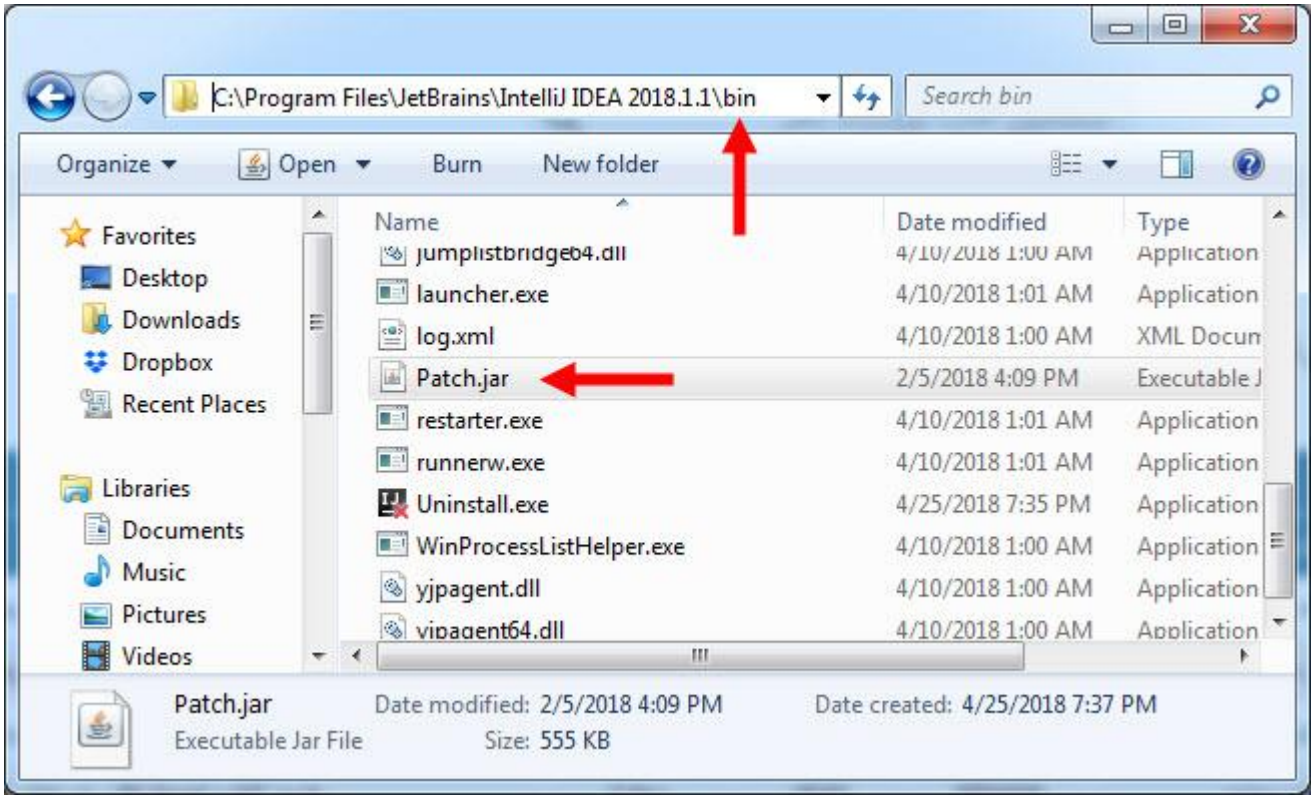

IntelliJ IDEA Setup
Choose Start Menu Folder
Choose a Start Menu folder for the IntelliJ IDEA shortcuts.
Select the Start Menu folder in which you would like to create the program's shortcuts. You can also enter a name to create a new folder.
JetBrains
Accessories
Administrative Tools
Anvsoft
A-PDF Password Security
ASUS
Backup and Sync from Google
Bigasoft
doPDF 7
Dropbox
Elaborate Bytes
ESET
Everything
< Back
Install
Cancel


IntelliJ IDEA Setup
Installing
Please wait while IntelliJ IDEA is being installed.
Extract: progressbar_indeterminate_holo6.png... 100%
Show details
< Back
Next >
Cancel

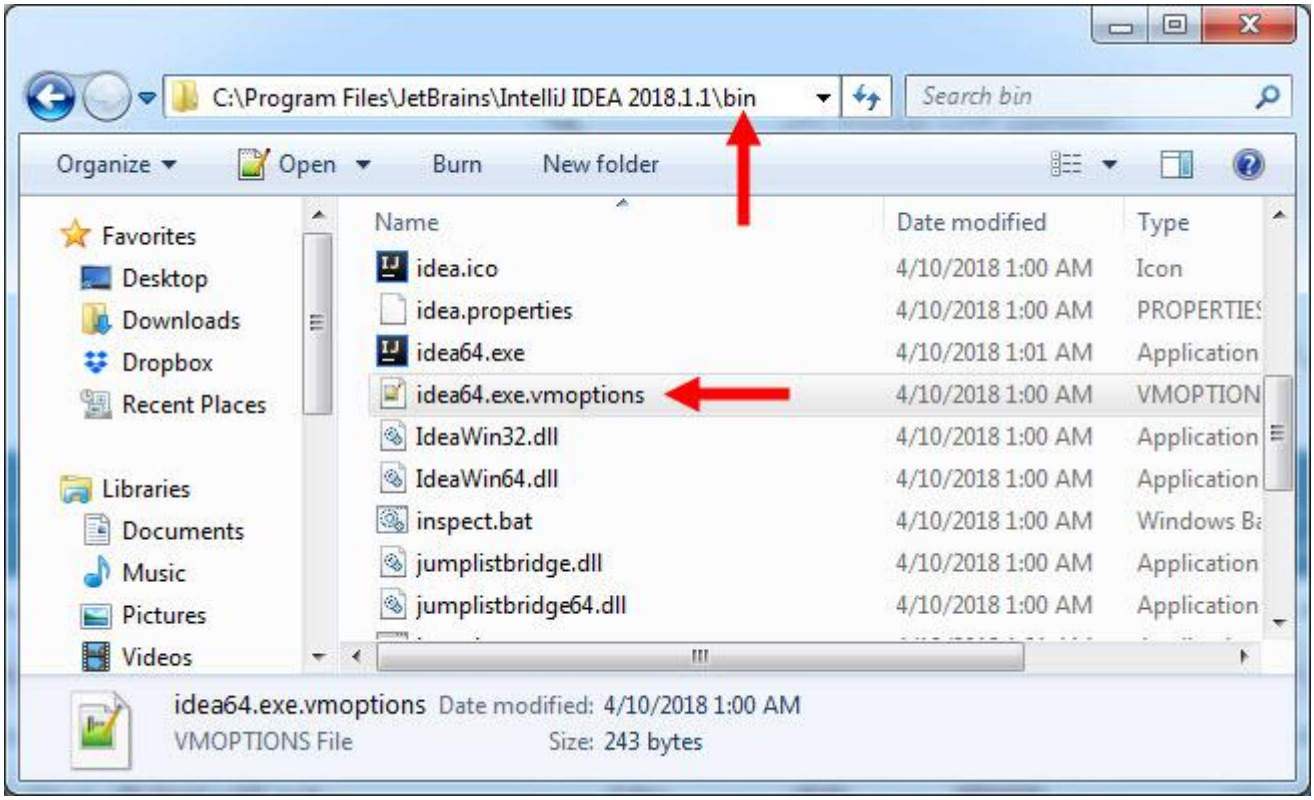
با زدن دکمه Close در شکل زیر، نصب برنامه به اتمام می‌رسد. و نوبت به کرک کردن آن می‌رسد:



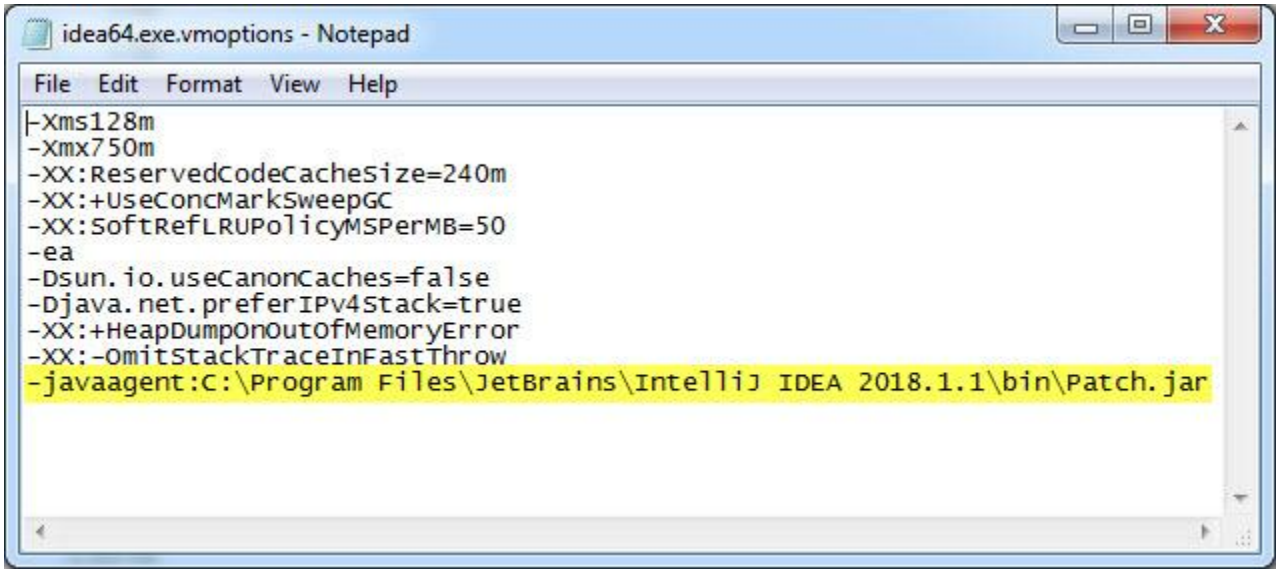
حال به پوشه برنامه که در شکل اول نشان داده شد، بروید و در داخل پوشه Crackآن فایل Patch.jar را Copy و در مسیری که نرم افزار IntelliJ IDEA نصب شده، در داخل پوشه bin، Paste کنید:



حال در داخل همین پوشه `bin` دو فایل با پسوند `vmoptions` هستند که یکی برای ویندوزهای ۶۴ و دیگری برای ویندوزهای ۳۲ می‌باشد. بسته به نوع ویندوزتان یکی از این دو را با `NotePad` باز کرده:

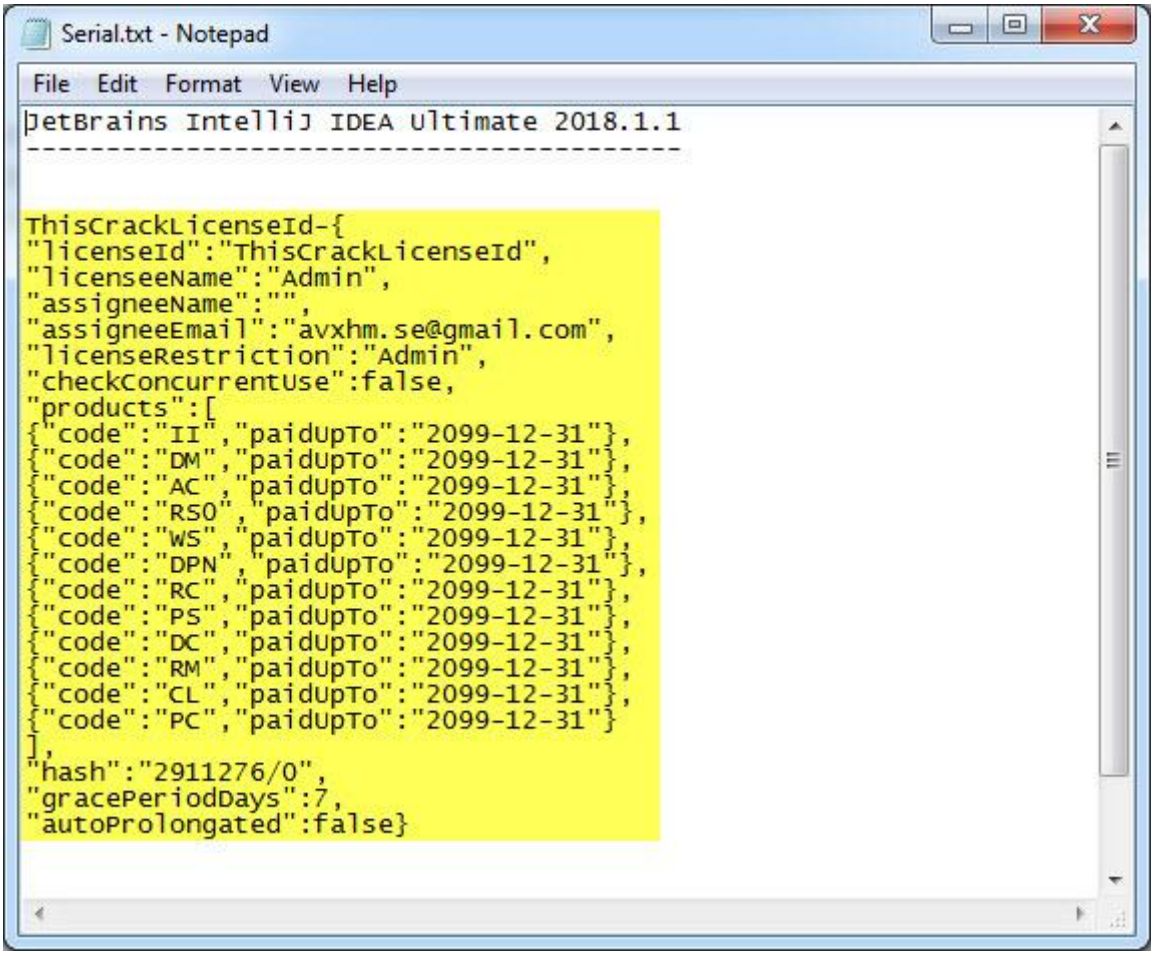


و مسیر فایل `Patch.jar` را در داخل آن به صورت زیر نوشته و ذخیره کنید:

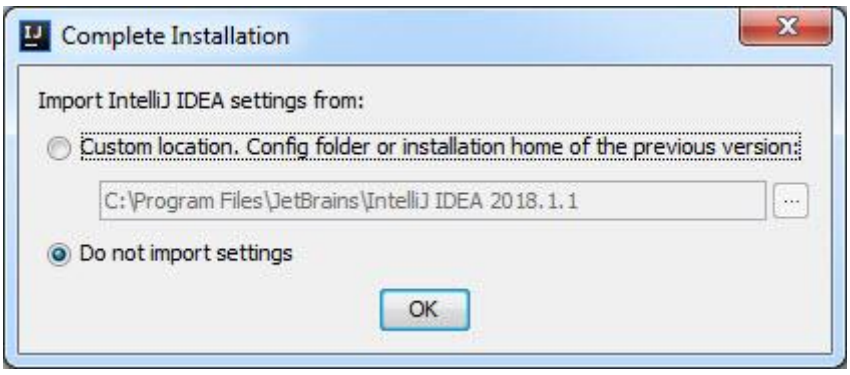


دوباره به پوشه برنامه که در شکل اول نشان داده شده است رفته و فایل `Serial.txt` را باز کرده و کدهایی که

در شکل زیر نشان داده شده است را کپی کرده:

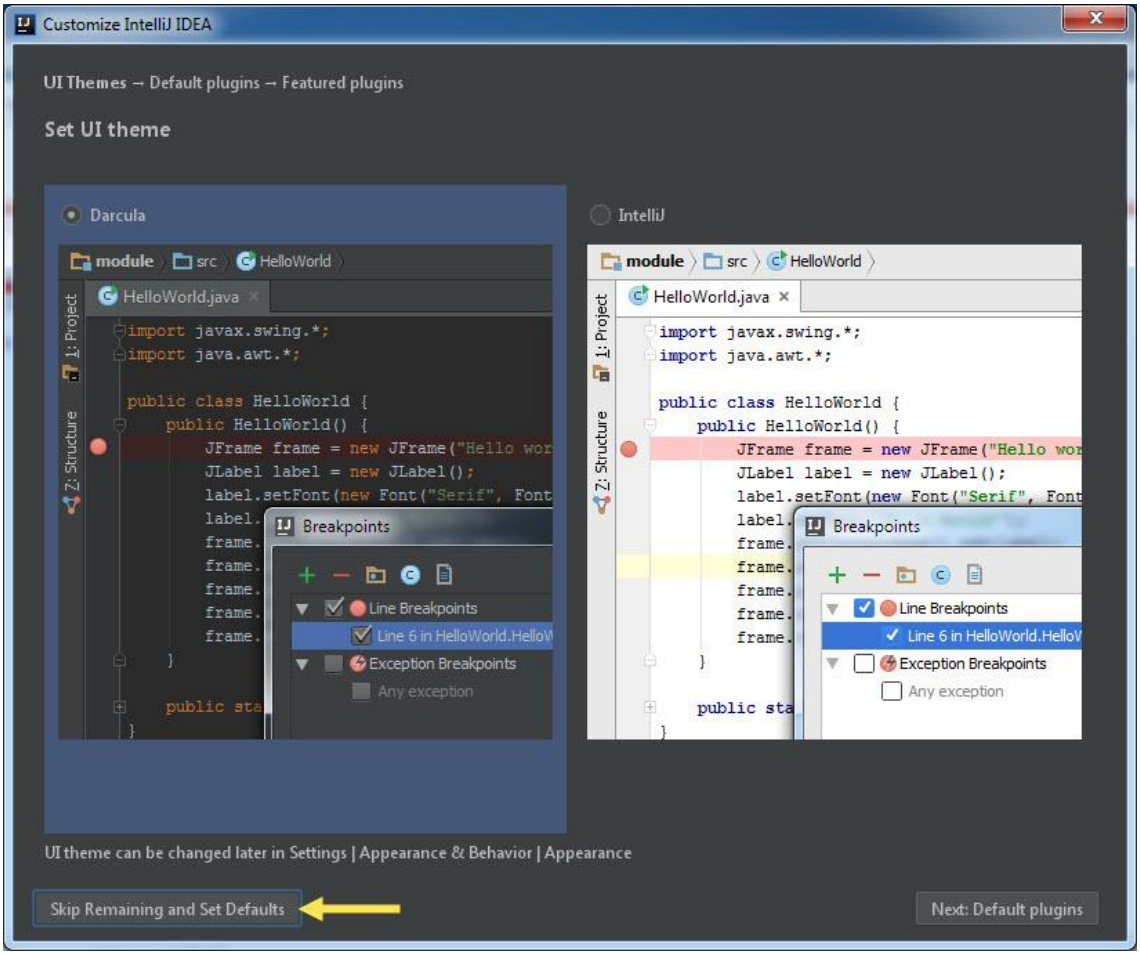


سپس برنامه را اجرا و دکمه Ok را در شکل زیر بزنید:

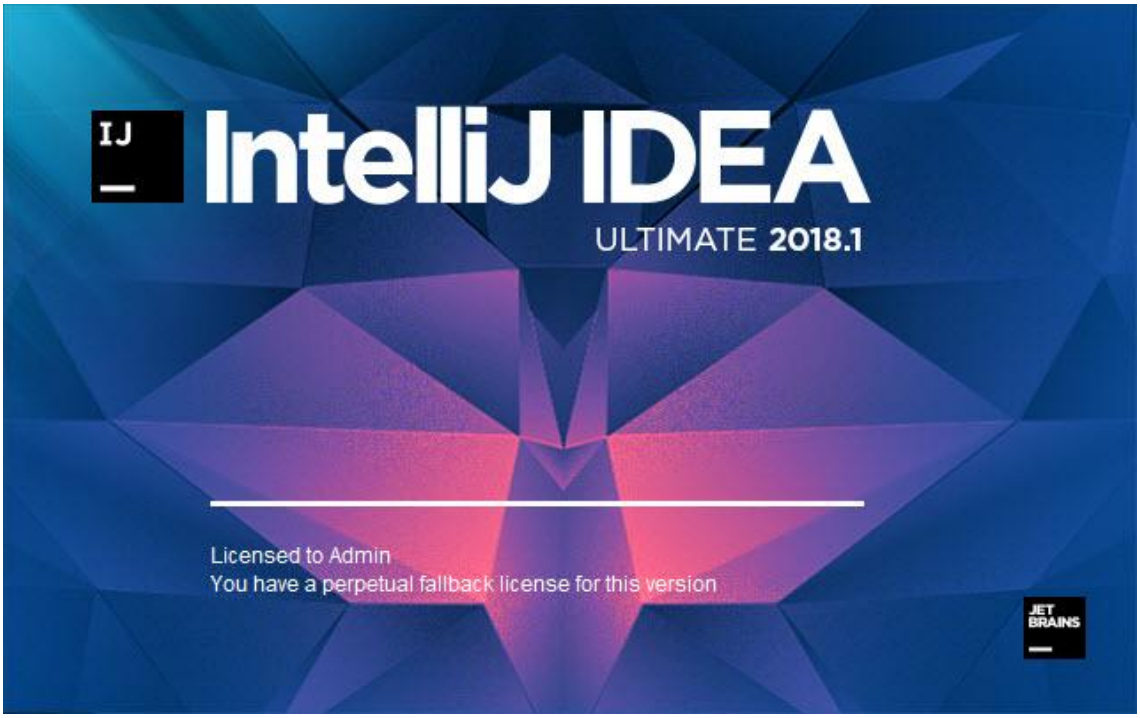


در شکل زیر سریالی را که کپی کرده‌اید، به صورت زیر Patse نمایید و دکمه OK را فشار دهید:

بعد از دکمه OK در شکل بالا شکل زیر طاهر می‌شود. در این شکل دکمه Skip Remaning and Set Defaults را بفشارید:



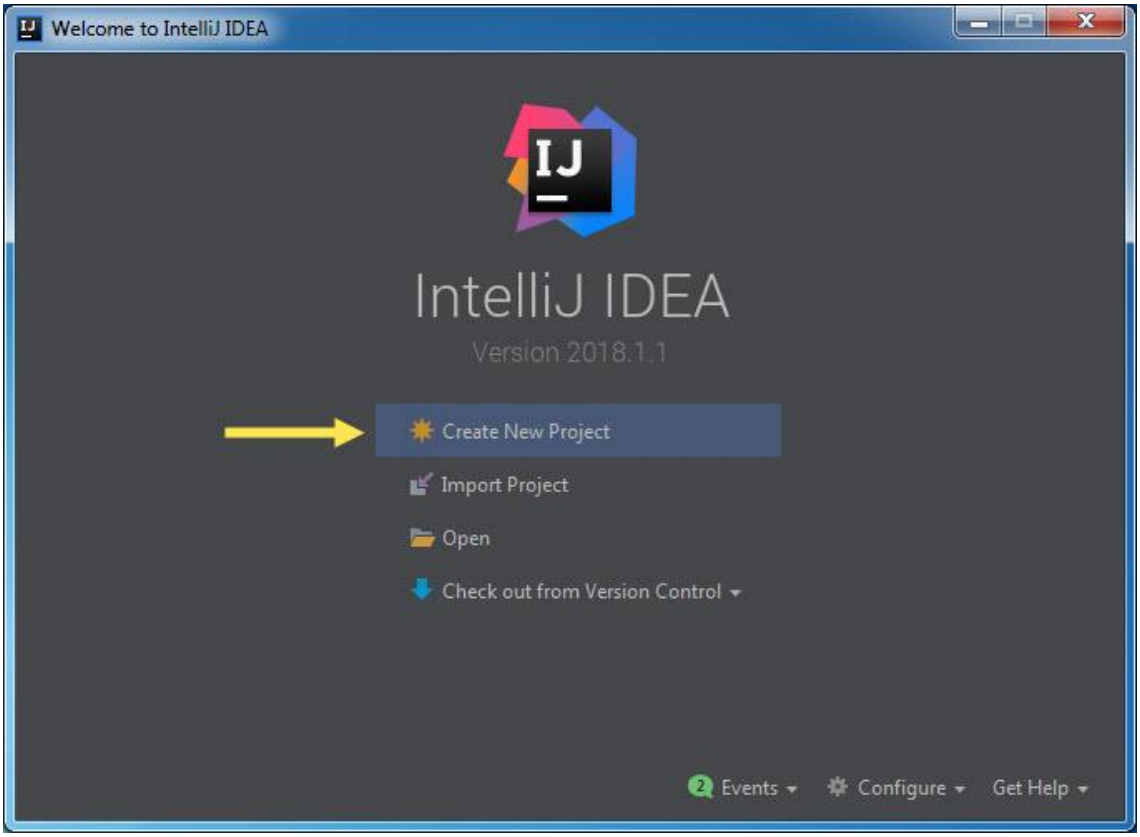
با ظاهر شدن شکل زیر نصب و کرک نرم افزار تمام می‌شود و شما می‌توانید با خیال راحت در آن کدنویسی کنید:



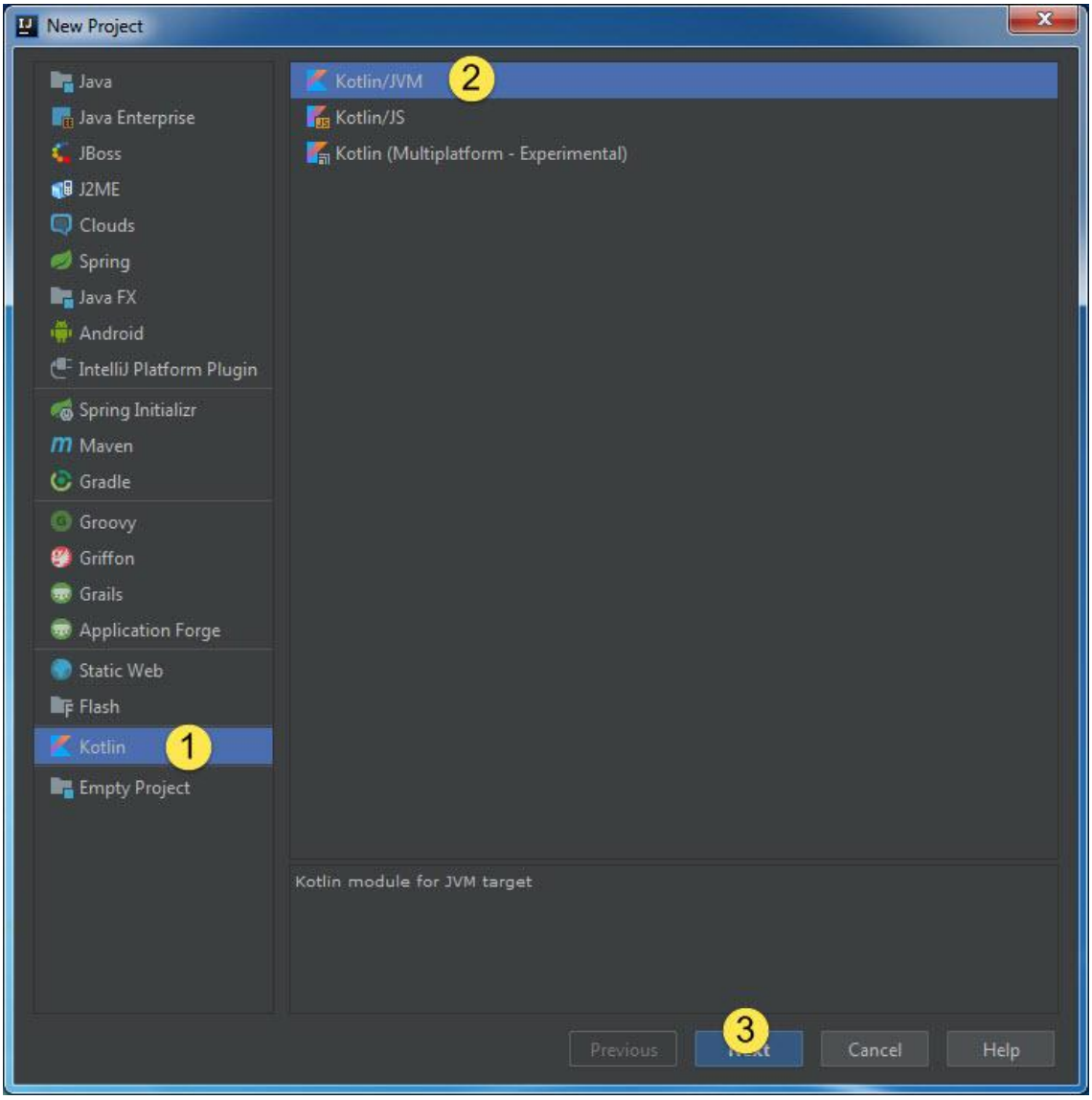
برای دریافت فایل‌ها و آپدیت‌های جدید این کتاب به سایت **www.w3-farsi.com** مراجعه فرمایید.

ساخت یک برنامه ساده در Kotlin

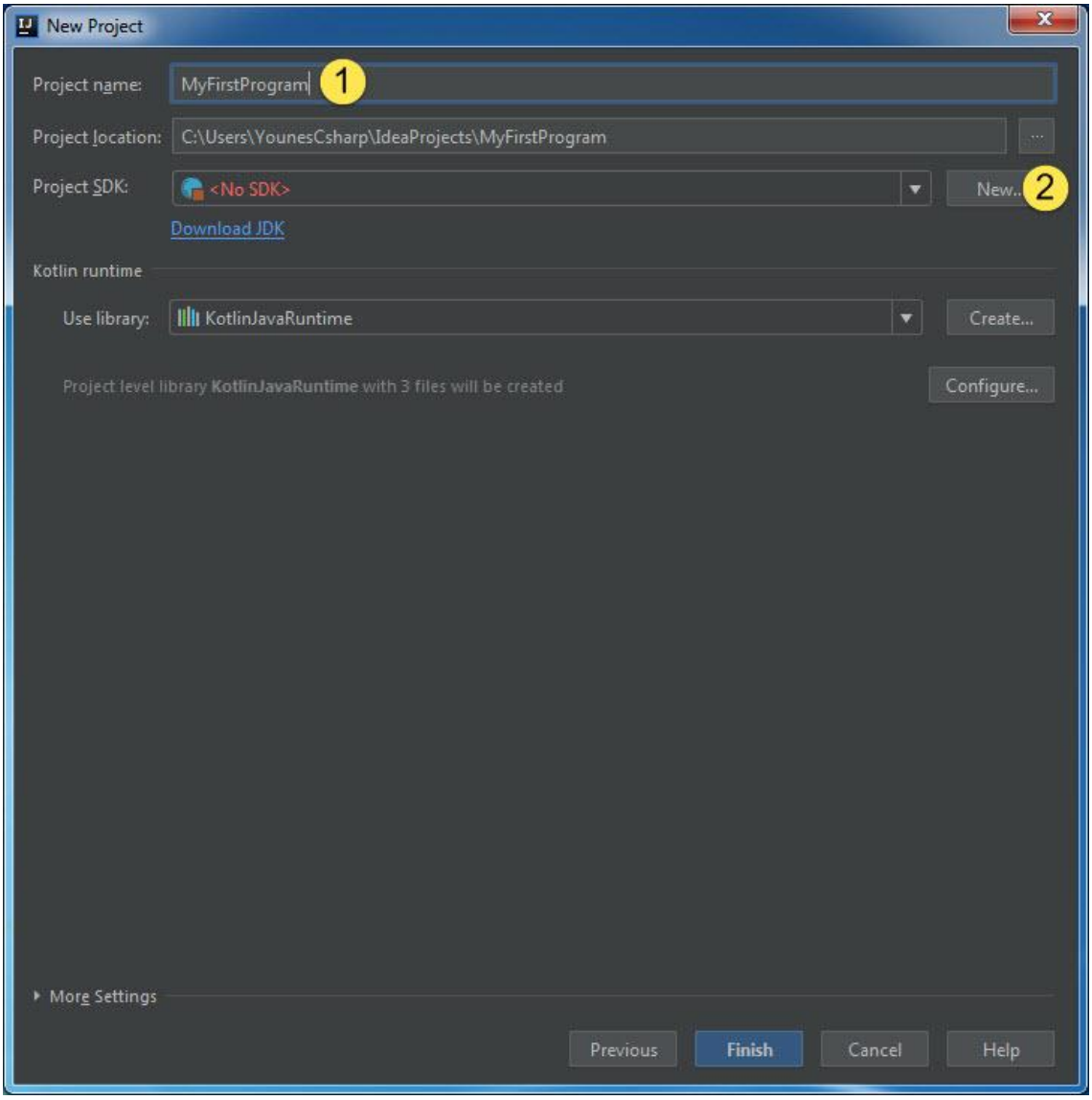
اجازه بدهید یک برنامه بسیار ساده به زبان Kotlin بنویسیم. این برنامه یک پیغام را نمایش می‌دهد. در این درس می‌خواهم ساختار و دستور زبان یک برنامه ساده Kotlin را توضیح دهم. ابتدا برنامه Intellij IDEA را اجرا کنید:



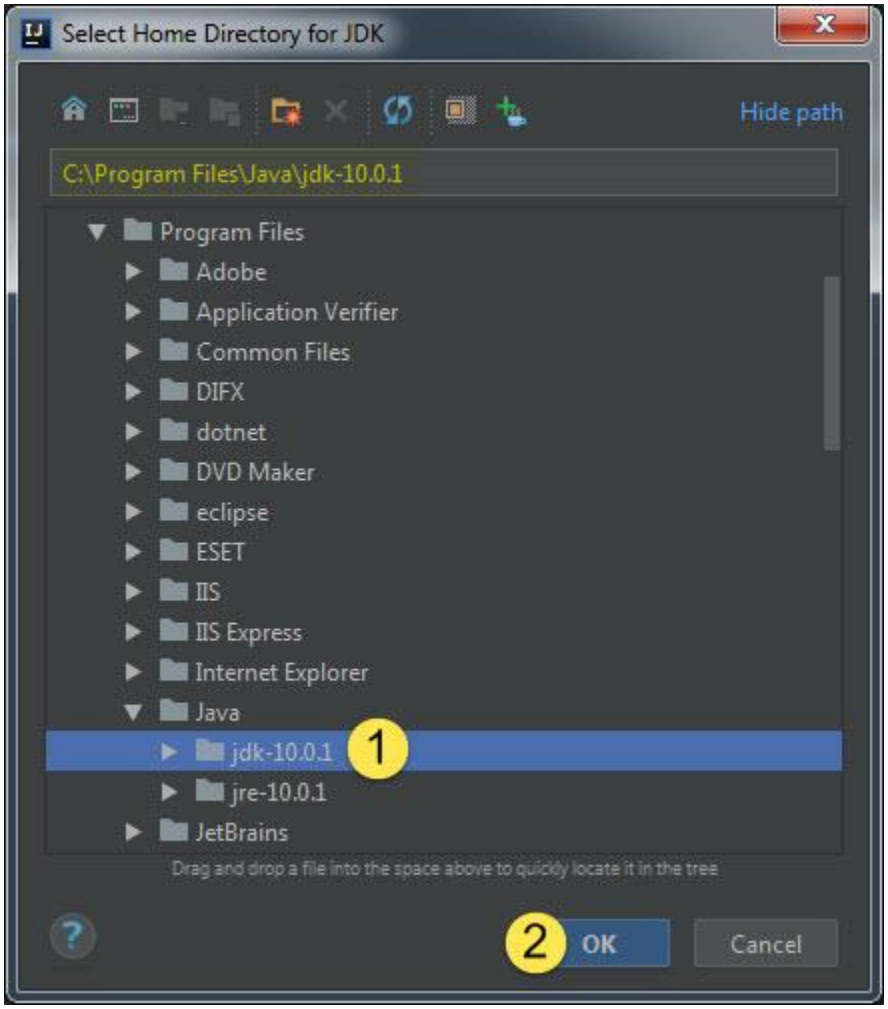
در شکل بالا، بر روی گزینه `Create New Project` کلیک کرده و در صفحه‌ای که به صورت زیر نمایش داده می‌شود هم، مراحل ۱ تا ۳ شکل زیر را طی نمایید:



در شکل زیر یک نام برای پروژه‌تان انتخاب کرده و سپس بر روی گزینه New کلیک نمایید:

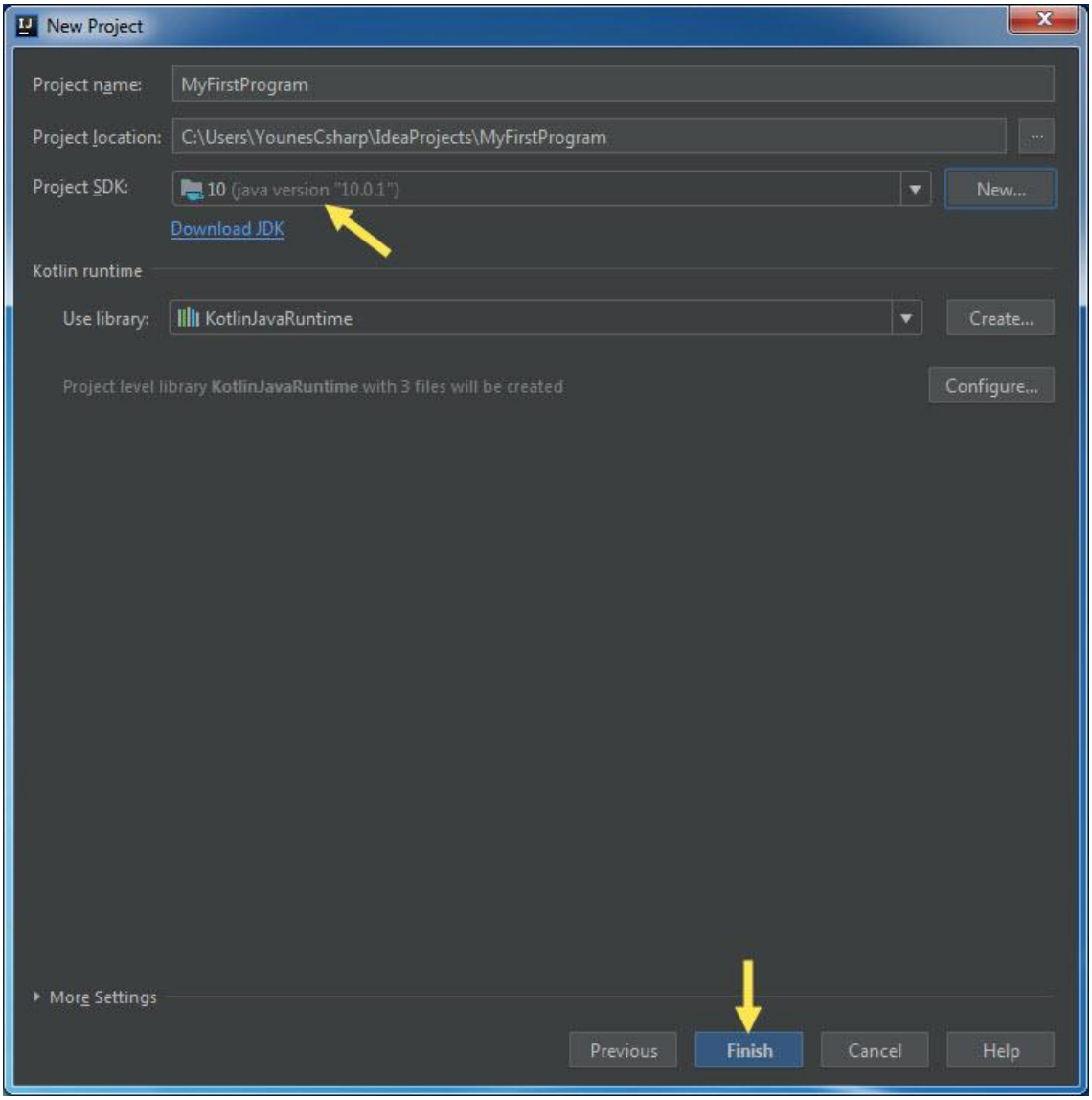


حال صفحه‌ای به صورت زیر نمایش داده می‌شود که از شما مسیر نصب فایل JDK را می‌خواهد. در درایو که آن را نصب کرده‌اید مسیر پوشه JDK را پیدا کرده و دکمه OK را کلیک کنید:



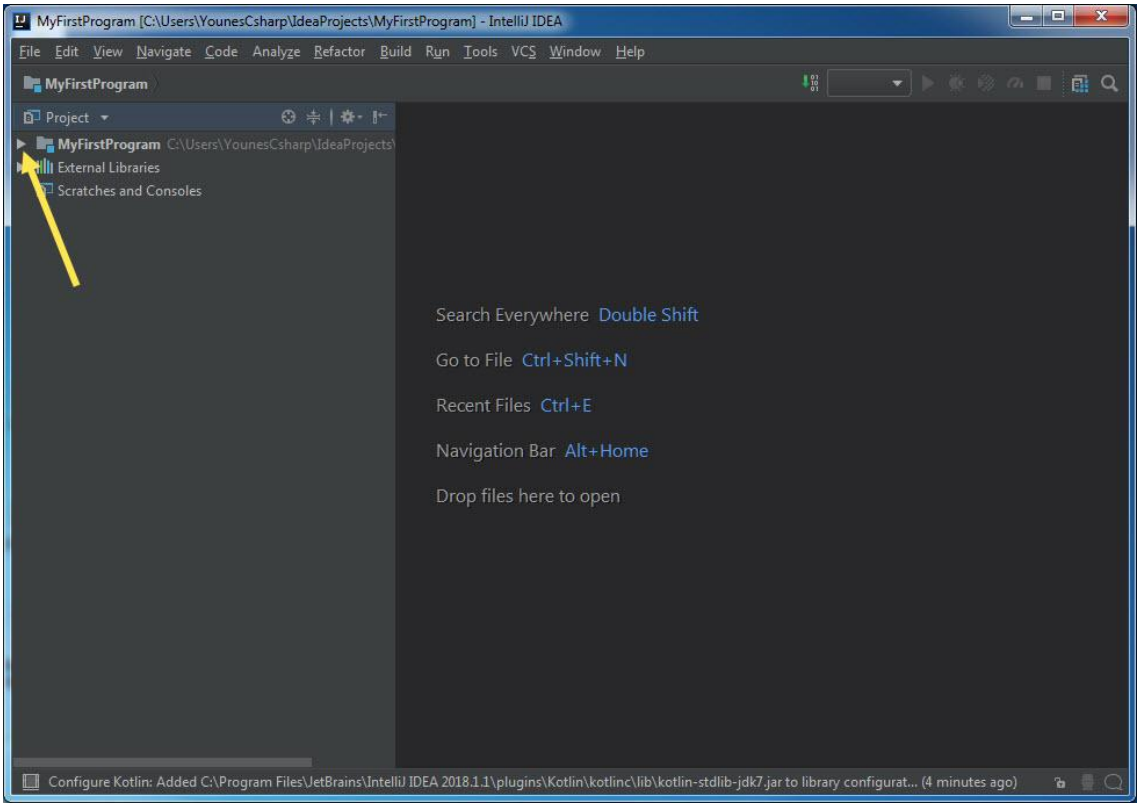
مشاهده می‌کنید که نام و نسخه JDK در کادر مربوطه نمایش داده می‌شود. سپس بر روی دکمه Finish

کلیک نمایید:

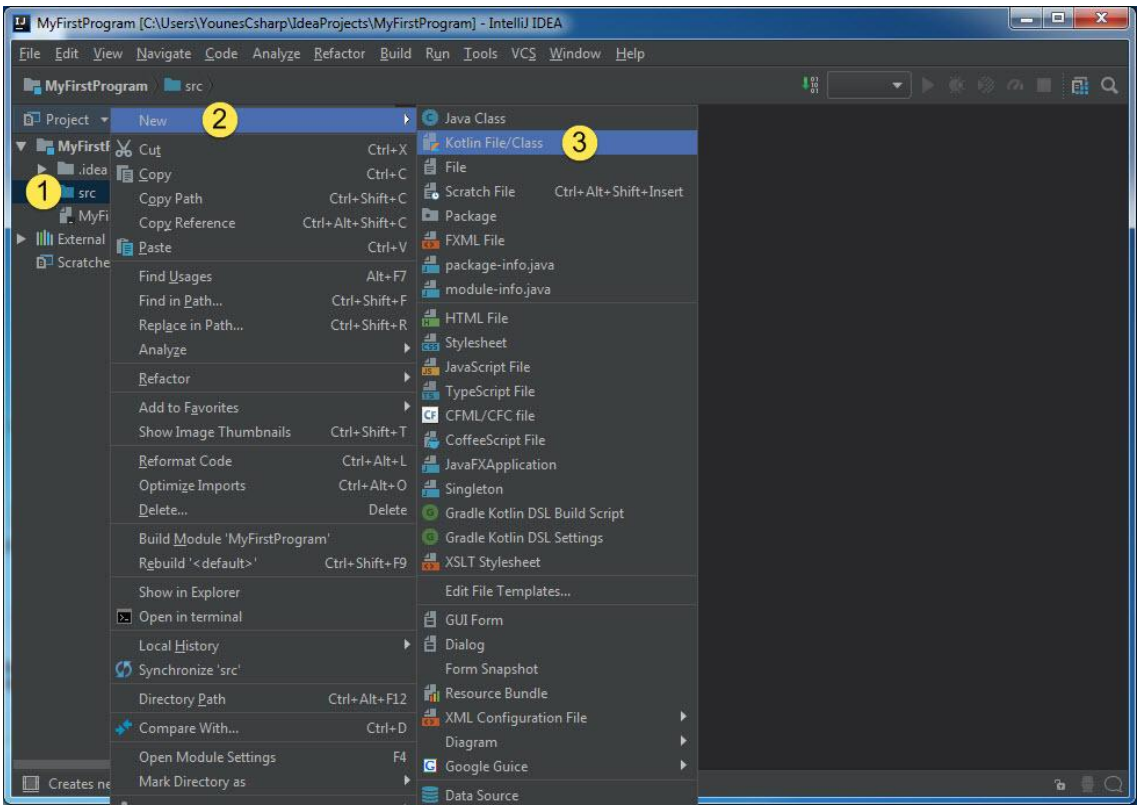


اگر صفحه‌ای به صورت زیر نمایش داده شد، تیک را بر داشته و دکمه `Close` را بزنید:

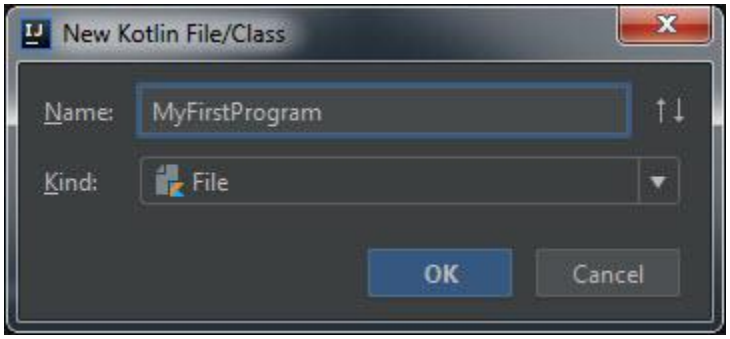
در شکل زیر بر روی فلش کوچک کنار نام پروژه کلیک کرده تا زیر مجموعه‌های آن نمایش داده شوند :



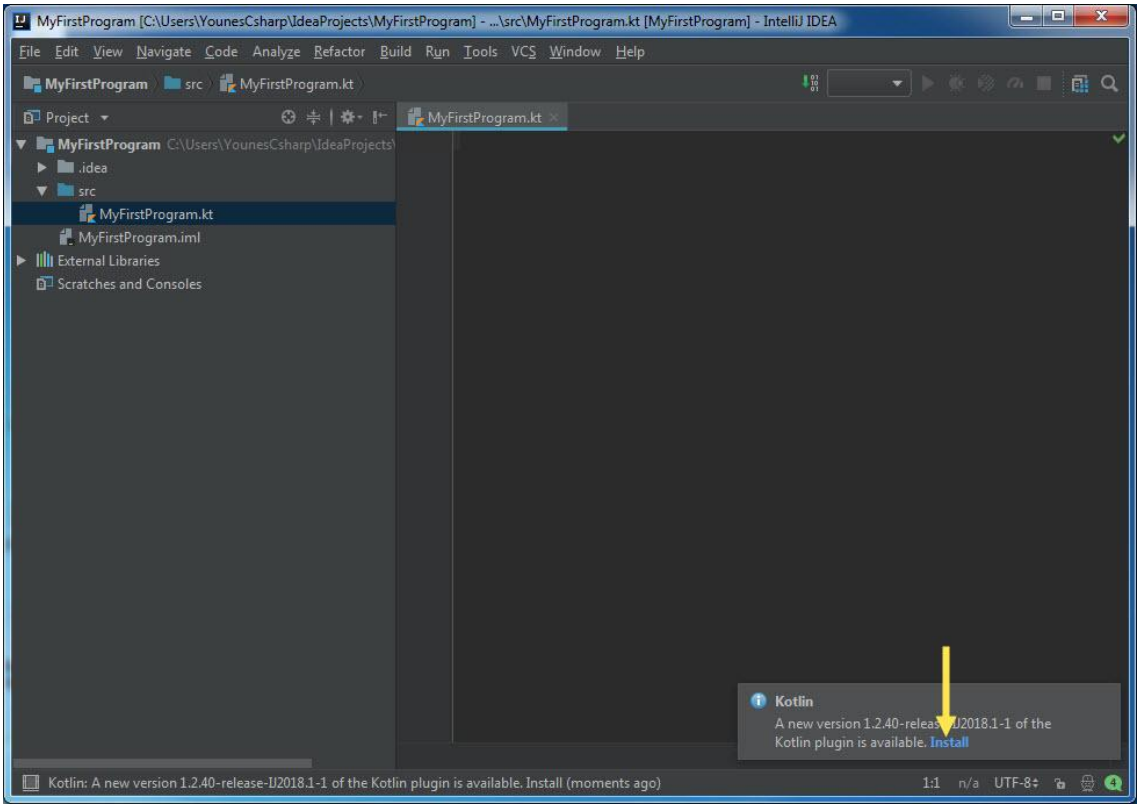
سپس بر روی پوشه src راست کلیک کرده و مراحل زیر را طی کنید:



در شکل زیر یک نام برای فایلی که قرار است ایجاد نمایید، انتخاب و سپس بر روی دکمه OK کلیک کنید:



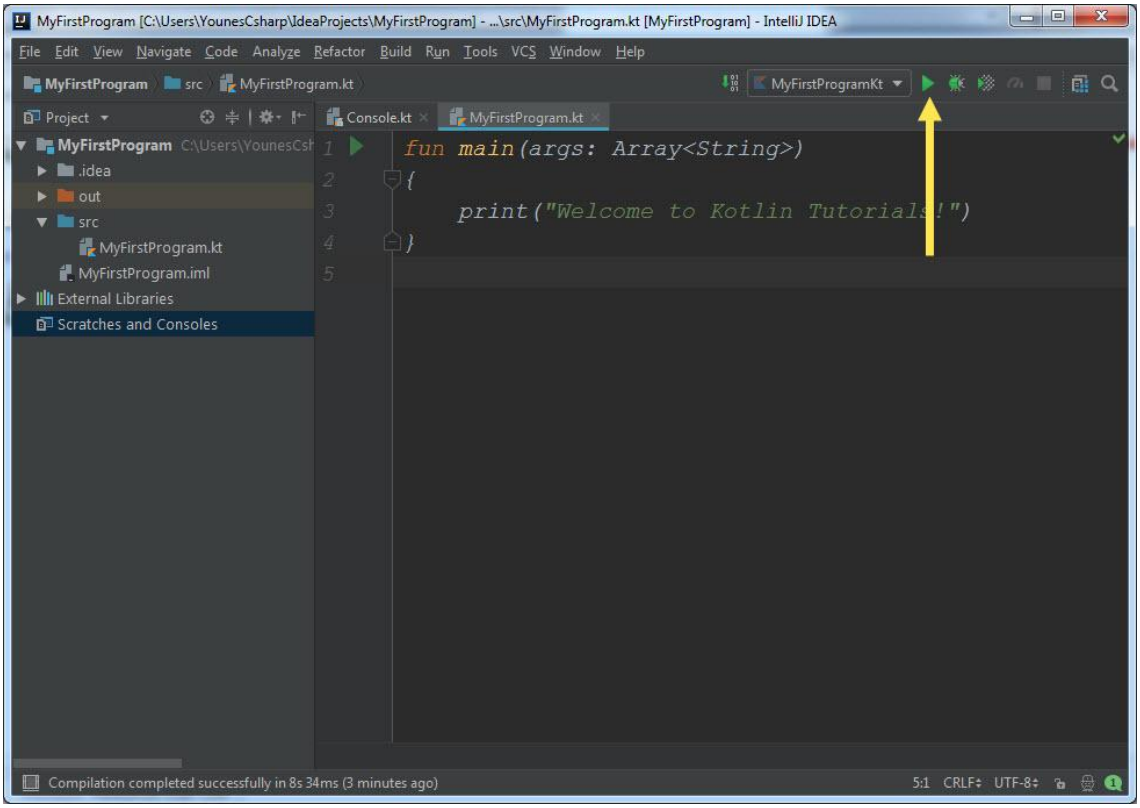
اگر پیغامی به صورت زیر مشاهده کردید بر روی دکمه Install کلیک کنید:



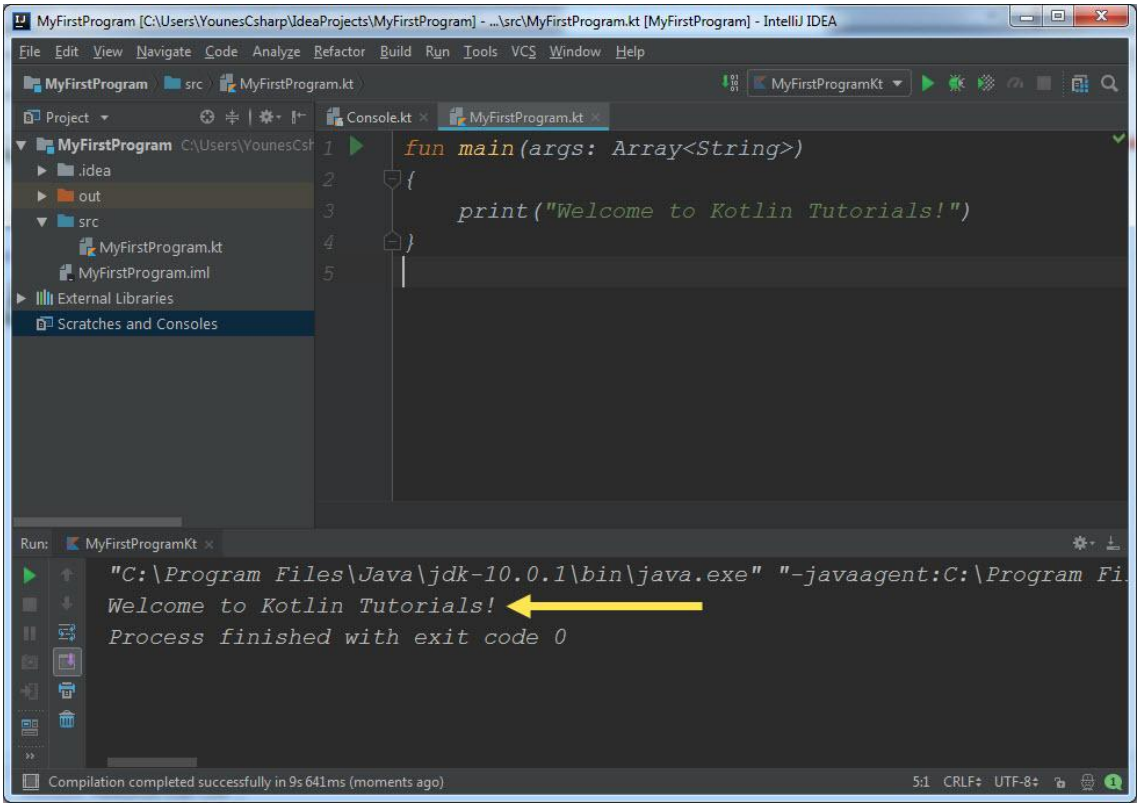
حال برنامه آماده کدنویسی است. کدهای زیر را داخل محیط کدنویسی بنویسید:

```
fun main(args: Array<String>)
{
   print("Welcome to Kotlin tutorials!")
}
```

محیط کدنویسی یا IDE جایی است که ما کدها را در آن تایپ می‌کنیم. کدها در محیط کدنویسی به صورت رنگی تایپ می‌شوند در نتیجه تشخیص بخشهای مختلف کد را راحت می‌کند. برای اجرای برنامه بر روی فلش کوچک و سبز رنگ کلیک کرده:



برنامه اجرا شده و پیغام `Welcome to Kotlin Tutorials!` نمایش داده می‌شود.



ساختار یک برنامه در کاتلین

مثال بالا ساده‌ترین برنامه‌ای است که شما می‌توانید در Kotlin بنویسید. هدف در مثال بالا نمایش یک پیغام در صفحه نمایش است:

```
fun main(args: Array<String>)
{
   print("Welcome to Kotlin tutorials!")
}
```

هر زبان برنامه نویسی دارای قواعدی برای کدنویسی است. اجازه بدهید هر خط کد را در مثال بالا توضیح بدهیم.

خط ۱ تابع main() یا تابع اصلی نامیده می‌شود. هر تابع شامل یک سری کد است که وقتی اجرا می‌شوند که تابع را صدا بزنیم. درباره تابع و نحوه صدا زدن آن در فصول بعدی توضیح خواهیم داد. تابع main() نقطه آغاز اجرای برنامه است. این بدان معناست که ابتدا تمام کدهای داخل تابع main() و سپس بقیه کدها اجرا می‌شود.

درباره تابع main() در فصول بعدی توضیح خواهیم داد. تابع main() و سایر توابع دارای آکولاد و کدهایی در داخل آن‌ها می‌باشند و وقتی کدها اجرا می‌شوند که توابع را صدا بزنیم.

در خط ۲ آکولاد ({) نوشته شده است. آکولاد برای تعریف یک بلوک کد به کار می‌رود. کاتلین یک زبان ساخت یافته است که شامل کدهای زیاد و ساختارهای فراوانی می‌باشد. هر آکولاد باز ({) در کاتلین باید دارای یک آکولاد بسته (}) نیز باشد. همه کدهای نوشته شده از خط ۲ تا خط ۴ یک بلوک کد است. مثالی از یک خط کد در کاتلین به صورت زیر است:

```
print("Welcome to Kotlin tutorials!")
```

این خط کد پیغام Welcome to Kotlin Tutorials! را در صفحه نمایش نشان می‌دهد. از تابع print() برای چاپ یک رشته استفاده می‌شود. یک رشته گروهی از کاراکترها است که به وسیله دابل کوتیشن (") محصور شده است. مانند:

```
"Welcome to Kotlin tutorials!"
```

یک کاراکتر می‌تواند یک حرف، عدد، علامت یا .... باشد. در کل مثال بالا نحوه استفاده از تابع print() نشان داده شده است. کاتلین فضای خالی و خطوط جدید را نادیده می‌گیرد. بنابراین شما می‌توانید همه برنامه را در یک خط بنویسید به شرط اینکه بین هر دو دستور علامت سمیکالن بگذارید:

```
Print("Welcome to "); print("Kotlin Tutorials!")
```

اما اینکار خواندن و اشکال زدایی برنامه را مشکل می‌کند. همیشه به یاد داشته باشید که کاتلین به بزرگی و کوچکی حروف حساس است. یعنی به طور مثال MAN و man در کاتلین با هم فرق دارند. رشته‌ها و توضیحات از این قاعده مستثنی هستند که در درسهای آینده توضیح خواهیم داد. مثلاً کدهای زیر با خطا مواجه می‌شوند و اجرا نمی‌شوند:

```
Print("Welcome to Kotlin Tutorials!")
PRINT("Welcome to Kotlin Tutorials!")
pRinT("Welcome to Kotlin Tutorials!")
```

تغییر در بزرگی و کوچکی حروف از اجرای کدها جلوگیری می‌کند. اما کد زیر کاملاً بدون خطا است:

```
print("Welcome to Kotlin tutorials!")
```

همیشه کدهای خود را در داخل آکولاد بنویسید.

```
{
      Statement
}
```

این کار باعث می‌شود که کدنویسی شما بهتر به چشم بیاید و تشخیص خطاها راحت‌تر باشد. یکی از ویژگیهای مهم Intellij IDEA نشان دادن کدها به صورت تو رفتگی است بدین معنی که کدها را به صورت تو رفتگی از هم تفکیک می‌کند و این در خوانایی برنامه بسیار مؤثر است.

مقایسه با جاوا

کسانی که با جاوا آشنایی دارند، می‌دانند که برای پیاده سازی کد همین درس در جاوا باید به صورت زیر عمل شود:

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

در بیان تفاوت بین کدهای جاوا و کاتلین به همین نکات زیر بسنده می‌کنیم:

- کدهای کاتلین نسب به جاوا خلاصه‌تر هستند.

- در برنامه‌های جاوا لازم است که یک کلاس ایجاد شود، در حالی که در کاتلین این کار توسط کامپایلر به صورت خودکار انجام می‌شود.
- برای استفاده از برخی از توابع مانند `print()` لازم نیست که کلاسی که این تابع در داخل آن قرار دارد، ذکر شود.

برای مشاهده نسخه کامل این کتاب و دیگر کتابهای یونس ابراهیمی به لینک زیر مراجعه فرمایید

```
https://goo.gl/Bsxr21
```

کاراکترهای کنترلی

کاراکترهای کنترلی کاراکترهای ترکیبی هستند که با یک بک اسلش (\) شروع می‌شوند و به دنبال آنها یک حرف یا عدد می‌آید و یک رشته را با فرمت خاص نمایش می‌دهند. برای مثال برای ایجاد یک خط جدید و قرار دادن رشته در آن می‌توان از کاراکتر کنترلی \n استفاده کرد:

```
print("Hello\nWorld!")
```

```
Hello
World
```

مشاهده کردید که کامپایلر بعد از مواجهه با کاراکتر کنترلی \n نشانگر ماوس را به خط بعد برده و بقیه رشته را در خط بعد نمایش می‌دهد. تابع `Println()` هم مانند کاراکتر کنترلی \n یک خط جدید ایجاد می‌کند، البته بدین صورت که در انتهای رشته یک کاراکتر کنترلی \n اضافه می‌کند:

```
println("Hello World!")
```

کد بالا و کد زیر هیچ فرقی با هم ندارند:

```
print("Hello World!\n")
```

تابع Print() کارکردی شبیه به Println() دارد با این تفاوت که نشان گر ماوس را در همان خط نگه می‌دارد و خط جدید ایجاد نمی‌کند. جدول زیر لیست کاراکترهای کنترلی و کارکرد آنها را نشان می‌دهد:

عملکرد	کاراکتر کنترلی	عملکرد	کاراکتر کنترلی
Form Feed	\f	چاپ کوتیشن	\’
خط جدید	\n	چاپ دابل کوتیشن	\”
سر سطر رفتن	\r	چاپ بک اسلش	\\
حرکت به صورت افقی	\t	حرکت به عقب	\b

ما برای استفاده از کاراکترهای کنترلی از بک اسلش (\) استفاده می‌کنیم. از آنجاییکه \ معنای خاصی به رشته‌ها می‌دهد برای چاپ بک اسلش (\) باید از (\\) استفاده کنیم:

```
println("We can print a \\ by using the \\\\ escape sequence.")
```

```
We can print a \ by using the \\ escape sequence.
```

یکی از موارد استفاده از \\، نشان دادن مسیر یک فایل در ویندوز است:

```
 println("C:\\Program Files\\Some Directory\\SomeFile.txt")
```

```
C:\Program Files\Some Directory\SomeFile.txt
```

از آنجاییکه از دابل کوتیشن (“) برای نشان دادن رشته‌ها استفاده می‌کنیم برای چاپ آن از \” استفاده می‌کنیم :

```
println("I said, \"Motivate yourself!\".")
```

```
I said, "Motivate yourself!".
```

همچنین برای چاپ کوتیشن (‘) از \’ استفاده می‌کنیم:

```
println("The programmer\'s heaven.")
```

```
The programmer's heaven.
```

برای ایجاد فاصله بین حروف یا کلمات از \t استفاده می‌شود:

```
println("Left\tRight")
```

```
Left    Right
```

هر تعداد کاراکتر که بعد از کاراکتر کنترلی \r بیایند به اول سطر منتقل و جایگزین کاراکترهای موجود می‌شوند

:

```
println("Mitten\rK")
```

```
K
```

مثلاً در مثال بالا کاراکتر K بعد از کاراکتر کنترلی \r آمده است. کاراکتر کنترلی حرف K را به ابتدای سطر برده و جایگزین Mitten می‌کند. برای مشاهده لیست مقادیر مبنای ۱۶ برای کاراکترهای یونیکد به لینک زیر مراجعه نمایید:

http://www.ascii.cl/htmlcodes.htm

اگر کامپایلر به یک کاراکتر کنترلی غیر مجاز برخورد کند، برنامه پیغام خطا می‌دهد. بیشترین خطا زمانی اتفاق می افتد که برنامه نویس برای چاپ اسلش (\) از \\ استفاده می‌کند.

توضیحات

وقتی که کدی تایپ می‌کنید شاید بخواهید که متنی جهت یادآوری وظیفه آن کد به آن اضافه کنید. در Kotlin (و بیشتر زبانهای برنامه نویسی) می‌توان این کار را با استفاده از توضیحات انجام داد. توضیحات متونی هستند که توسط کامپایلر نادیده گرفته می‌شوند و به عنوان بخشی از کد محسوب نمی‌شوند.

هدف اصلی از ایجاد توضیحات، بالا بردن خوانایی و تشخیص نقش کدهای نوشته شده توسط شما، برای دیگران است. فرض کنید که می‌خواهید در مورد یک کد خاص، توضیح بدهید، می‌توانید توضیحات را در بالای کد یا کنار آن بنویسید. از توضیحات برای مستند سازی برنامه هم استفاده می‌شود. در برنامه زیر نقش توضیحات نشان داده شده است:

```
1  fun main(args: Array<String>)
2  {
3      // This line will print the message hello world
4      println("Hello World!")
5  }
```

```
Hello World!
```

در کد بالا، خط ۳ یک توضیح درباره خط ۴ است که به کاربر اعلام می‌کند که وظیفه خط ۴ چیست؟ با اجرای کد بالا فقط جمله Hello World چاپ شده و خط ۳ در خروجی نمایش داده نمی‌شود چون کامپایلر توضیحات را نادیده می‌گیرد. توضیحات بر دو نوعند:

- توضیحات تک خطی

```
// single line comment
```

- توضیحات چند خطی

```
/* multi
line
comment */
```

توضیحات تک خطی همانگونه که از نامش پیداست، برای توضیحاتی در حد یک خط به کار می‌روند. این توضیحات با علامت // شروع می‌شوند و هر نوشته‌ای که در سمت راست آن قرار بگیرد جز توضیحات به حساب می‌آید. این نوع توضیحات معمولاً در بالا یا کنار کد قرار می‌گیرند. اگر توضیح درباره یک کد به بیش از یک خط نیاز باشد از توضیحات چند خطی استفاده می‌شود. توضیحات چند خطی با /* شروع و با */ پایان می‌یابند. هر نوشته‌ای که بین این دو علامت قرار بگیرد جز توضیحات محسوب می‌شود.

متغیر

متغیر مکانی از حافظه است که شما می‌توانید مقادیری را در آن ذخیره کنید. می‌توان آن را به عنوان یک ظرف تصور کرد که داده‌های خود را در آن قرار داده‌اید. محتویات این ظرف می‌تواند پاک شود یا تغییر کند. هر متغیر دارای یک نام نیز هست. که از طریق آن می‌توان متغیر را از دیگر متغیرها تشخیص داد و به مقدار آن دسترسی پیدا کرد. همچنین دارای یک مقدار می‌باشد که می‌تواند توسط کاربر انتخاب شده باشد یا نتیجه یک محاسبه باشد. مقدار متغیر می‌تواند تهی نیز باشد. متغیر دارای نوع نیز هست بدین معنی که نوع آن با نوع داده‌ای که در آن ذخیره می‌شود یکی است. متغیر دارای عمر نیز هست که از روی آن می‌توان تشخیص داد که متغیر باید چقدر در طول برنامه مورد استفاده قرار گیرد. در نهایت متغیر دارای محدوده استفاده نیز هست، که به شما می‌گوید، که متغیر در چه جای برنامه برای شما قابل دسترسی است. ما از متغیرها به عنوان یک انبار موقتی برای ذخیره داده استفاده می‌کنیم. هنگامی که یک برنامه ایجاد می‌کنیم احتیاج به یک مکان برای ذخیره داده، مقادیر یا داده‌هایی که توسط کاربر وارد می‌شوند داریم. این مکان، همان متغیر است. برای این از کلمه متغیر استفاده می‌شود چون ما می‌توانیم بسته به نوع شرایط هر جا که لازم باشد مقدار آن را تغییر دهیم. متغیرها موقتی هستند و فقط موقعی مورد استفاده قرار می‌گیرند که برنامه در حال اجراست و وقتی شما برنامه را می‌بندید،

محتویات متغیرها نیز پاک می‌شود. قبلاً ذکر شد که به وسیله نام متغیر می‌توان به آن دسترسی پیدا کرد. برای نامگذاری متغیرها باید قوانین زیر را رعایت کرد:

۱. نام متغیر باید با یکی از حروف الفبا و یا علامت زیر خط (_) شروع شود.
۲. حرف اول نمی‌تواند عدد باشد.
۳. نمی‌تواند شامل کاراکترهای غیرمجاز مانند #، ?، ^، $ و . باشد.
۴. نمی‌توان از کلمات رزرو شده در کاتلین برای نام متغیر استفاده کرد.
۵. نام متغیر نباید دارای فضای خالی (spaces) باشد.
۶. اگر اسم انتخاب شده برای متغیر چند کلمه‌ای باشد طبق قرارداد کلمه اول را با حروف کوچک نوشته و کلمات بعدی را با حرف بزرگ شروع کنید. مانند myFirstVariable.
۷. اسامی متغیرها نسبت به بزرگی و کوچکی حروف حساس هستند. در کاتلین دو حرف مانند a و A دو کاراکتر مختلف به حساب می‌آیند.

دو متغیر با نامهای myNumber و MyNumber دو متغیر مختلف محسوب می‌شوند چون یکی از آنها با حرف کوچک m و دیگری با حرف بزرگ M شروع می‌شود. شما نمی‌توانید دو متغیر را که دقیق شبیه هم هستند را در یک scope (محدوده) تعریف کنید.Scope به معنای یک بلوک کد است که متغیر در آن قابل دسترسی و استفاده است. در مورد Scope در فصلهای آینده بیشتر توضیح خواهیم داد. متغیر دارای نوع هست که نوع داده‌ای را که در خود ذخیره می‌کند را نشان می‌دهد. معمول‌ترین انواع داده Float، Double، Int، Short، Byte و Long می‌باشند. برای مثال شما برای قرار دادن یک عدد صحیح در متغیر باید از نوع Int استفاده کنید. لیست کلمات کلیدی کاتلین، که نباید از آنها در نامگذاری متغیرها استفاده کرد، در زیر آمده است:

```
as          break       class       continue          do          else
false       for         fun         if                in          interface
is          null        object      package           return      super
this        throw       true        try               typealias   typeof
val         var         when        while
```

انواع داده

انواع ساده، انواعی از داده‌ها هستند که شامل، اعداد، کاراکترها و مقادیر بولی می‌باشند. به انواع ساده، انواع اصلی نیز گفته می‌شود چون از آنها برای ساخت انواع پیچیده‌تری مانند کلاس‌ها و ... استفاده می‌شود. انواع ساده، دارای مجموعه مشخصی از مقادیر هستند و محدوده خاصی از اعداد را در خود ذخیره می‌کنند. در جدول زیر انواع ساده و محدود آنها آمده است:

نوع	دامنه
Byte	اعداد صحیح بین ۱۲۸- تا ۱۲۷
Short	اعداد صحیح بین ۳۲۷۶۸- تا ۳۲۷۶۷
Int	اعداد صحیح بین ۲۱۴۷۴۸۳۶۴۸- تا ۲۱۴۷۴۸۳۶۴۷
Long	اعداد صحیح بین ۹۲۲۳۳۷۲۰۳۶۸۵۴۷۷۵۸۰۸- تا ۹۲۲۳۳۷۲۰۳۶۸۵۴۷۷۸۰۷

جدول زیر انواعی که مقادیر با ممیز اعشار را می‌توانند در خود ذخیره کنند را نشان می‌دهد:

نوع	دامنه تقریبی	دقت
Float	±1.5E-45 to ±3.4E38	۷ رقم
Double	±5.0E-324 to ±1.7E308	۱۵-۱۶ رقم

برای به خاطر سپردن آنها باید از نماد علمی استفاده شود. نوع دیگری از انواع ساده برای ذخیره داده‌های غیر عددی به کار می‌روند و در جدول زیر نمایش داده شده‌اند:

نوع	مقادیر مجاز
Char	کاراکترهای یونیکد
Boolean	مقدار true یا false

نوع Char برای ذخیره کاراکترهای یونیکد استفاده می‌شود. کاراکترها باید داخل یک کوتیشن ساده قرار بگیرند مانند ('a').

نوع Boolean فقط می‌تواند مقادیر درست (true) یا نادرست (false) را در خود ذخیره کند و بیشتر در برنامه‌هایی که دارای ساختار تصمیم گیری هستند مورد استفاده قرار می‌گیرد.

استفاده از رشته‌ها

کاتلین دارای نوعی به نام رشته یا String است. از رشته برای ذخیره گروهی از کاراکترها مانند یک پیغام استفاده می‌شود. مقادیر ذخیره شده در یک رشته باید داخل دابل کوتیشن قرار گیرند تا توسط کامپایلر به عنوان یک رشته در نظر گرفته شوند، مانند ("massage"). دلیل اینکه در این قسمت درباره رشته‌ها مختصری توضیح دادیم این است که قرار است، در آینده به طور مفصل در مورد رشته‌ها توضیح دهیم.

استفاده از متغیرها

در مثال زیر نحوه تعریف و مقدار دهی متغیرها نمایش داده شده است:

```
fun main(args: Array<String>)
{
    //Declare variables
    val num1: Int
    val num2: Int
    val num3: Double
    val num4: Double
    val boolVal: Boolean
    val myChar: Char

    //Assign values to variables
    num1 = 1
    num2 = 2
    num3 = 3.54
    num4 = 4.12
    boolVal = true
    myChar = 'R'

    //Show the values of the variables
    println("num1    = $num1")
    println("num3    = $num3")
    println("num4    = $num4")
    println("boolVal = $boolVal")
    println("num2    = $num2")
    println("myChar  = $myChar")
}
```

```
num1    = 1
num3    = 3.54
num4    = 4.12
boolVal = true
num2    = 2
myChar  = R
```

تعریف متغیر

برای تعریف متغیر از دو کلمه `var` و یا `val` به صورت زیر استفاده می‌شود:

```
val identifier
//or
var identifier
```

`Identifier` نامی است که برای متغیر انتخاب می‌کنیم. تفاوت بین `var` و `val` را در ادامه توضیح می‌دهیم.

به تعریف دو متغیر توجه کنید:

```
val myString = "Hello World!"
val myNumber = 10
```

در کد بالا myString یک متغیر از نوع رشته و myNumber یک متغیر از نوع صحیح یا Int است. ما در کد بالا نوع متغیرها را صراحتاً اعلام نکرده‌ایم و این کار را بر عهده کامپایلر گذاشته‌ایم. روش دیگر تعریف متغیر این است که صراحتاً نوع متغیر را ذکر کنیم:

```
val myString: String = "Hello World!"
val myNumber: Int = 10
```

حال اجازه دهید که کد بالا را توضیح دهیم.

تعریف متغیر

در خطوط ۹-۴ متغیرهایی با نوع و نام متفاوت تعریف شده‌اند. ابتدا باید یکی از کلمات var یا val، سپس نام متغیر و علامت : و در نهایت نوعی که این متغیرها قرار است در خود ذخیره کنند را مشخص کنیم. همیشه به یاد داشته باشید که قبل از مقدار دهی و استفاده از متغیر باید آن را تعریف کرد.

```
val num1: Int
val num2: Int
val num3: Double
val num4: Double
val boolVal: Boolean
val myChar: Char
```

نحوه تعریف متغیر به صورت زیر است:

```
var/val identifier: data_type
```

date_type همان نوع داده است مانند Int، Double و ... . identifierنیز نام متغیر است که به ما امکان استفاده و دسترسی به مقدار متغیر را می‌دهد.

نامگذاری متغیرها

- نام متغیر باید با یک حرف یا زیرخط و به دنبال آن حرف یا عدد شروع شود.

- نمی‌توان از کاراکترهای خاص مانند #، %، & یا عدد برای شروع نام متغیر استفاده کرد مانند 2numbers.
- نام متغیر نباید دارای فاصله باشد. برای نام‌های چند حرفی می‌توان به جای فاصله از علامت زیرخط یا _ استفاده کرد.

نامهای مجاز:

```
num1  myNumber  studentCount  total       first_name    _minimum
num2  myChar    average       amountDue   last_name     _maximum
name  counter   sum           isLeapYear  color_of_car  _age
```

نامهای غیر مجاز:

```
123        #numbers#  #ofstudents  1abc2
123abc     $money     first name   ty.np
my number  this&that  last name    1:00
```

اگر به نامهای مجاز در مثال بالا توجه کنید متوجه قراردادهای به کار رفته در نامگذاری آن‌ها خواهید شد. یکی از روشهای نامگذاری، نامگذاری کوهان شتری است. در این روش که برای متغیرهای دو کلمه‌ای به کار می‌رود، اولین حرف اولین کلمه با حرف کوچک و اولین حرف دومین کلمه با حرف بزرگ نمایش داده می‌شود مانند :myNumber توجه کنید که اولین حرف کلمه Number با حرف بزرگ شروع شده است. مثال دیگر کلمه numberOfStudents است. اگر توجه کنید، بعد از اولین کلمه، حرف اول سایر کلمات با حروف بزرگ نمایش داده شده است.

محدوده متغیر

متغیرهای ابتدای درس در داخل تابع ()main تعریف شده‌اند. در نتیجه، این متغیرها فقط در داخل تابع ()main قابل دسترسی و استفاده هستند. محدوده یک متغیر مشخص می‌کند که متغیر در کجای کد قابل دسترسی است. هنگامیکه برنامه به پایان تابع ()main می‌رسد متغیرها از محدوده خارج و بدون استفاده می‌شوند تا زمانی که برنامه در حال اجراست. محدوده متغیرها انواعی دارد که در درسهای بعدی با آنها آشنا می‌شوید. تشخیص

محدوده متغیر بسیار مهم است چون به وسیله آن می‌فهمید که در کجای کد می‌توان از متغیر استفاده کرد. باید یاد آور شد که دو متغیر در یک محدوده نمی‌توانند دارای نام یکسان باشند. مثلاً کد زیر در برنامه ایجاد خطا می‌کند:

```
val num1: Int
val num1: Int
```

از آنجاییکه کاتلین به بزرگی و کوچک بودن حروف حساس است، می‌توان از این خاصیت برای تعریف چند متغیر هم نام ولی با حروف متفاوت (از لحاظ بزرگی و کوچکی) برای تعریف چند متغیر از یک نوع استفاده کرد مانند:

```
val num1: Int
val Num1: Int
val NUM1: Int
```

مقداردهی متغیرها

می‌توان فوراً بعد از تعریف متغیرها مقادیری را به آنها اختصاص داد. این عمل را مقداردهی می‌نامند. در زیر نحوه مقداردهی متغیرها نشان داده شده است:

```
var/val identifier: data_type = value
```

به عنوان مثال:

```
val myNumber: Int = 7;
```

تعریف متغیر با مقدار دهی متغیرها متفاوت است. تعریف متغیر یعنی انخاب نوع و نام برای متغیر ولی مقدار دهی یعنی اختصاص یک مقدار به متغیر.

اختصاص مقدار به متغیر

در زیر نحوه اختصاص مقادیر به متغیرها نشان داده شده است:

```
num1 = 1
num2 = 2
num3 = 3.54
num4 = 4.12
boolVal = true
myChar = 'R'
```

به این نکته توجه کنید که شما به متغیری که هنوز تعریف نشده، نمی‌توانید مقدار بدهید. شما فقط می‌توانید از متغیرهایی استفاده کنید که هم تعریف و هم مقدار دهی شده باشند. مثلاً متغیرهای بالا همه قابل استفاده هستند. در این مثال num1 و num2 هر دو تعریف شده‌اند و مقادیری از نوع صحیح به آنها اختصاص داده شده است. اگر نوع داده با نوع متغیر یکی نباشد برنامه پیغام خطا می‌دهد. به خطوط ۲۵-۲۰ می‌باشد. وجود علامت $ قبل از نام متغیر در اصل به منزله مقداری است که متغیر در خود ذخیره کرده است. مثلاً $num1 در خط ۲۰، یعنی مقدار این متغیر که همان عدد ۱ در ۱۲ است. حال همین علامت $ در خطوط ۲۵-۲۰ را حذف کرده و برنامه را اجرا و نتیجه را مشاهده نمایید. البته از این علامت زمانی استفاده می‌شود که شما بخواهید در داخل رشته یعنی علامت "" مقدار متغیر را چاپ کنید:

```
println("$num1")
```

ولی اگر بخواهید خارج از علامت "" مقدار یک متغیر را چاپ کنید، دیگر احتیاجی به علامت $ نیست:

```
println(num1)
```

تفاوت var و val

تفاوت این دو کلمه در این است که اگر یک متغیر را با کلمه val تعریف و مقدار دهی کنیم، دیگر نمی‌توانیم در طول برنامه مقدار متغیر را تغییر دهیم و اگر این کار را انجام دهیم برنامه با خطا مواجه می‌شود:

```
val number = 10
    number = 5
```

ولی اگر از کلمه var استفاده کنیم، می‌توانیم بعداً مقدار متغیر را تغییر دهیم.

```
var number = 10
    number = 5
```

تبدیل انواع داده

توصیه می‌کنیم قبل از شروع این درس ابتدا دو مطلب زیر را بخوانید:

http://www.w3-farsi.com/?p=5698

http://www.w3-farsi.com/?p=5710

در کاتلین، انواع عددی نمی‌توانند به طور خودکار یا ضمنی به هم تبدیل شوند. مثلاً در جاوا کد زیر قابل قبول است:

```
byte number1 = 5;

int number2 = number1;
```

در کد بالا نوع byte به نوع int تبدیل و این کار به صورت خودکار توسط کامپایلر انجام می‌شود. حال اگر همین کد را در کاتلین پیاده سازی کنیم با خطا مواجه می‌شویم:

```
val number1: Byte = 5

val number2: Int = number1
```

برای تبدیل یک نوع به نوع دیگر باید صریحاً نوع داده‌ای که قرار است تبدیل شود را مشخص کنیم و این کار با استفاده از توابعی در کاتلین امکان پذیر است که در جدول زیر به آنها اشاره شده است:

تابع	کاربرد

toByte()	مقدار یک متغیر را به نوع Byte تبدیل می‌کند.
toChar()	مقدار یک متغیر را به نوع Char تبدیل می‌کند.
toFloat()	مقدار یک متغیر را به نوع Float تبدیل می‌کند.
toInt()	مقدار یک متغیر را به نوع Int تبدیل می‌کند.
toLong()	مقدار یک متغیر را به نوع Long تبدیل می‌کند.
toShort()	مقدار یک متغیر را به نوع Short تبدیل می‌کند.
toString()	مقدار یک متغیر را به نوع String تبدیل می‌کند.

با وجود این توابع تبدیل انواع داده به هم امکان پذیر است. در نتیجه کد کاتلین بالا به صورت زیر قابل انجام است:

```
fun main(args: Array<String>)
{
    val number1: Byte = 5
    val number2: Int = number1.toInt()

    print(number2)
}
```

```
5
```

در خط ۲ کد بالا بعد از نام متغیر اول، علامت نقطه و نام تابع را ذکر کرده‌ایم. این بدین معناست که متغیر number1 را به نوعInt تبدیل کن و در داخل متغیر number2 قرار بده. در مثال نوع داده‌ای Byte می‌تواند مقادیر ۰ تا ۲۵۵ را در خود ذخیره کند و نوع داده‌ای int مقادیر ۲۱۴۷۴۸۳۶۴۸- تا ۲۱۴۷۴۸۳۶۴۷ را شامل می‌شود. پس می‌توانید بدون هیچ مشکلی یک متغیر از نوع Byte را به یک نوع Int تبدیل کنید. مقدار ۵ متغیر number1 در محدوده مقادیر Byte یعنی اعداد بین ۲۵۵-۰ قرار دارد اما متغیری از نوع بایت حافظه کمتری نسبت به متغیری از نوع صحیح اشغال می‌کند. نوع Byte شامل ۸ بیت یا ۸ رقم دودویی است در حالی که

نوع Int شامل ۳۲ بیت یا رقم باینری است. یک عدد باینری عددی متشکل از ۰ و ۱ است. برای مثال عدد ۵ در کامپیوتر به عدد باینری ۱۰۱ ترجمه می‌شود. بنابراین وقتی ما عدد ۵ را در یک متغیر از نوع بایت ذخیره می‌کنیم عددی به صورت زیر نمایش داده می‌شود:

```
00000101
```

و وقتی آن را در یک متغیر از نوع صحیح ذخیره می‌کنیم به صورت زیر نمایش داده می‌شود:

```
00000000000000000000000000000101
```

بنابراین قرار دادن یک مقدار Int، در یک متغیر Byte، درست مانند این است که ما سعی کنیم که یک توپ فوتبال را در یک سوراخ کوچک گلف جای دهیم. حال به مثال زیر توجه کنید:

```
fun main(args: Array<String>)
{
    val number1: Int = 300
    val number2: Byte = number1.toByte()

    print(number2)
}
```

```
44
```

در کد بالا یک number1 یک نوع صحیح با مقدار ۳۰۰ است که آن را با استفاده از تابع toByte() به نوع Byteتبدیل کرده و در متغیر number2 قرار داده‌ایم. پس چرا خروجی ۴۴ است؟ دلیل آن این است که نوع Byte اعداد بین ۲۵۵-۰ را در خود ذخیره می‌کند و عدد ۳۰۰ جز این محدود نیست. اما چرا عدد ۴۴؟ این کار به تعداد بیتها بستگی دارد. یک Byte دارای ۸ بیت است درحالی که Int دارای ۳۲ بیت است. حال اگر به مقدار باینری ۲ عدد توجه کنید متوجه می‌شوید که چرا خروجی عدد ۴۴ است.

```
300 = 00000000000000000000000100101100
255 =                         11111111
-------------------------------------
 44 =                         00101100
```

خروجی بالا نشان می‌دهد که بیشترین مقدار `byte` که عدد ۲۵۵ است می‌تواند فقط شامل ۸ بیت باشد (۱۱۱۱۱۱۱۱) بنابراین فقط ۸ بیت اول مقدار `Int` به متغیر `Byte` انتقال می‌یابد که شامل (۰۰۱۰۱۱۰۰) یا عدد ۴۴ در مبنای ۱۰ است.

عبارات و عملگرها

ابتدا با دو کلمه آشنا شوید:

- عملگر: نمادهایی هستند که اعمال خاص انجام می‌دهند.
- عملوند: مقادیری که عملگرها بر روی آنها عملی انجام می‌دهند.

مثلاً `X+Y`: یک عبارت است که در آن `X` و `Y` عملوند و علامت + عملگر به حساب می‌آیند.

زبانهای برنامه نویسی جدید دارای عملگرهایی هستند که از اجزاء معمول زبان به حساب می‌آیند. کاتلین دارای عملگرهای مختلفی از جمله عملگرهای ریاضی، تخصیصی، مقایسه‌ای، منطقی و بیتی می‌باشد. از عملگرهای ساده ریاضی می‌توان به عملگر جمع و تفریق اشاره کرد. دو نوع عملگر در کاتلین وجود دارد:

- یگانی - `(Unary)` به یک عملوند نیاز دارد
- دودویی - `(Binary)` به دو عملوند نیاز دارد

انواع مختلف عملگر که در این بخش مورد بحث قرار می‌گیرند عبارتند از:

- عملگرهای ریاضی
- عملگرهای تخصیصی
- عملگرهای مقایسه‌ای

- عملگرهای منطقی
- عملگرهای بیتی
- عملگر in
- عملگر دسترسی به اندیس
- عملگر invoke (درخواست)

عملگرهای ریاضی

کاتلین از عملگرهای ریاضی برای انجام محاسبات استفاده می‌کند. جدول زیر عملگرهای ریاضی کاتلین را نشان می‌دهد:

عملگر	مثال	نتیجه
+	var1 = var2 + var3	Var1 برابر است با حاصل جمع var2 و var3
-	var1 = var2 - var3	Var1 برابر است با حاصل تفریق var2 و var3
*	var1 = var2 * var3	Var1 برابر است با حاصلضرب var2 در var3
/	var1 = var2 / var3	Var1 برابر است با حاصل تقسیم var2 بر var3
%	var1 = var2 % var3	Var1 برابر است با باقیمانده تقسیم var2 و var3

حال می‌توانیم با ایجاد یک برنامه نحوه عملکرد عملگرهای ریاضی در کاتلین را یاد بگیریم:

```
1  fun main(args: Array<String>)
2  {
3      //Variable declarations
4      val num1: Int
5      val num2: Int
```

```
 6
 7       //Assign test values
 8       num1 = 5
 9       num2 = 3
10
11       println("The sum of $num1 and $num2 is "        + (num1 + num2))
12       println("The difference of $num1 and $num2 is " + (num1 - num2))
13       println("The product of $num1 and $num2 is "    + (num1 * num2))
14       println("The quotient of $num1 and $num2 is "   + (num1 / num2))
15       println("The remainder of $num1 and $num2 is "  + (num1 % num2))
16   }
```

```
The sum of 5 and 3 is 8
The difference of 5 and 3 is 2
The product of 5 and 3 is 15
The quotient of 5 and 3 is 1
The remainder of 5 divided by 3 is 2
```

برنامه بالا نتیجه هر عبارت را نشان می‌دهد. در این برنامه از تابع `println()` برای نشان دادن نتایج در سطرهای متفاوت استفاده شده است. خطوط ۱۵-۱۱ کد بالا را به صورت زیر هم می‌توان نوشت:

```
println("The sum of $num1 and $num2 is         ${num1 + num2}")
println("The difference of $num1 and $num2 is  ${num1 - num2}")
println("The product of $num1 and $num2 is     ${num1 * num2}")
println("The quotient of $num1 and $num2 is    ${num1 / num2}")
println("The remainder of $num1 and $num2 is   ${num1 % num2}")
```

دیگر عملگرهای کاتلین عملگرهای کاهش و افزایش هستند. این عملگرها مقدار ۱ را از متغیرها کم یا به آنها اضافه می‌کنند. از این متغیرها اغلب در حلقه‌ها استفاده می‌شود:

عملگر	مثال	نتیجه
++	var1 = ++var2	مقدار var1 برابر است با var2 بعلاوه ۱
--	var1 = --var2	مقدار var1 برابر است با var2 منهای ۱
++	var1 = var2++	مقدار var1 برابر است با var2 به متغیر var2 یک واحد اضافه می‌شود

--	var1 = var2--	مقدار var1 برابر است با var2 از متغیر var2 یک واحد کم می‌شود

به این نکته توجه داشته باشید که، محل قرار گیری عملگر در نتیجه محاسبات تأثیر دارد. اگر عملگر قبل از متغیر var2 بیاید، افزایش یا کاهش var1 اتفاق می افتد. چنانچه عملگرها بعد از متغیر var2 قرار بگیرند، ابتدا var1 برابر var2 می‌شود و سپس متغیر var2 افزایش یا کاهش می‌یابد. به مثال‌های زیر توجه کنید:

```
fun main(args: Array<String>)
{
    var x = 0
    var y = 1

    x = ++y

    println("x = $x")
    println("y = $y")
}
```

```
X = 2
Y = 2
```

```
fun main(args: Array<String>)
{
    var x = 0
    var y = 1

    x = --y

    println("x = $x")
    println("y = $y")
}
```

```
X = 0
Y = 0
```

همانطور که در دو مثال بالا مشاهده می‌کنید، درج عملگرهای -- و ++ قبل از عملوند y باعث می‌شود که ابتدا یک واحد از y کم و یا یک واحد به y اضافه شود و سپس نتیجه در عملوند x قرار بگیرد. حال به دو مثال زیر توجه کنید:

```
fun main(args: Array<String>)
{
    var x = 0
    var y = 1

    x = y--

    println("x = $x")
    println("y = $y")
}
```

```
X = ۱
Y = 0
```

```
fun main(args: Array<String>)
{
    var x = 0
    var y = 1

    x = y++

    println("x = $x")
    println("y = $y")
}
```

```
X = ۱
Y = ۲
```

همانطور که در دو مثال بالا مشاهده می‌کنید، درج عملگرهای -- و ++ بعد از عملوند y باعث می‌شود که ابتدا مقدار y در داخل متغیر x قرار بگیرد و سپس یک واحد از y کم و یا یک واحد به ان اضافه شود.

عملگرهای تخصیصی

نوع دیگر از عملگرهای کاتلین عملگرهای جایگزینی نام دارند. این عملگرها مقدار متغیر سمت راست خود را در متغیر سمت چپ قرار می‌دهند. جدول زیر انواع عملگرهای تخصیصی در کاتلین را نشان می‌دهد:

عملگر	مثال	نتیجه

=	var1 = var2	مقدار var1 برابر است با مقدار var2
+=	var1 += var2	مقدار var1 برابر است با حاصل جمع var1 و var2
-=	var1 -= var2	مقدار var1 برابر است با حاصل تفریق var1 و var2
*=	var1 *= var2	مقدار var1 برابر است با حاصل ضرب var1 در var2
/=	var1 /= var2	مقدار var1 برابر است با حاصل تقسیم var1 بر var2
%=	var1 %= var2	مقدار var1 برابر است با باقیمانده تقسیم var1 بر var2

از عملگر =+ برای اتصال دو رشته نیز می‌توان استفاده کرد. استفاده از این نوع عملگرها در واقع یک نوع خلاصه نویسی در کد است. مثلاً شکل اصلی کد var1 += var2 به صورت var1 = var1 + var2 می‌باشد. این حالت کدنویسی زمانی کارایی خود را نشان می‌دهد که نام متغیرها طولانی باشد. برنامه زیر چگونگی استفاده از عملگرهای تخصیصی و تأثیر آنها را بر متغیرها نشان می‌دهد.

```
fun main(args: Array<String>)
{
    var number: Int

    println("Assigning 10 to number...")
    number = 10
    println("Number = $number")

    println("Adding 10 to number...")
    number += 10
    println("Number = $number")

    println("Subtracting 10 to number...")
    number -= 10
    println("Number = $number")
}
```

```
Assigning 10 to number...
Number = 10
Adding 10 to number...
Number = 20
Subtracting 10 from number...
Number = 10
```

در برنامه از ۳ عملگر تخصیصی استفاده شده است. ابتدا یک متغیر و مقدار ۱۰ با استفاده از عملگر = به آن اختصاص داده شده است. سپس به آن با استفاده از عملگر =+ مقدار ۱۰ اضافه شده است. و در آخر به وسیله عملگر =- عدد ۱۰ از آن کم شده است.

عملگرهای مقایسه‌ای

از عملگرهای مقایسه‌ای برای مقایسه مقادیر استفاده می‌شود. نتیجه این مقادیر یک مقدار بولی (منطقی) است. این عملگرها اگر نتیجه مقایسه دو مقدار درست باشد، مقدار `true` و اگر نتیجه مقایسه اشتباه باشد، مقدار `false` را نشان می‌دهند. این عملگرها به طور معمول در دستورات شرطی به کار می‌روند به این ترتیب که باعث ادامه یا توقف دستور شرطی می‌شوند. جدول زیر عملگرهای مقایسه‌ای در کاتلین را نشان می‌دهد:

عملگر	مثال	نتیجه
==	var1 = var2 == var3	var1 در صورتی true است که مقدار var2 با مقدار var3 برابر باشد در غیر اینصورت false است.
!=	var1 = var2 != var3	var1 در صورتی true است که مقدار var2 با مقدار var3 برابر نباشد در غیر اینصورت false است.
<	var1 = var2 < var3	var1 در صورتی true است که مقدار var2 کوچک‌تر از var3 مقدار باشد در غیر اینصورت false است.
>	var1 = var2 > var3	var1 در صورتی true است که مقدار var2 بزرگ‌تر از مقدار var3 باشد در غیر اینصورت false است.
<=	var1 = var2 <= var3	var1 در صورتی true است که مقدار var2 کوچک‌تر یا مساوی مقدار var3 باشد در غیر اینصورت false است.

>=	var1 = var2 >= var3	var1 در صورتی true است که مقدار var2 بزرگ‌تر یا مساوی var3 مقدار باشد در غیر اینصورت false است.

برنامه زیر نحوه عملکرد این عملگرها را نشان می‌دهد:

```
fun main(args: Array<String>)
{
    val num1 = 10
    val num2 = 5

    println("$num1 == $num2 : " + (num1 == num2))
    println("$num1 != $num2 : " + (num1 != num2))
    println("$num1  < $num2 : " + (num1 <  num2))
    println("$num1  > $num2 : " + (num1 >  num2))
    println("$num1 <= $num2 : " + (num1 <= num2))
    println("$num1 >= $num2 : " + (num1 >= num2))
}
```

```
10 == 5 : false
10 != 5 : true
10  < 5 : false
10  > 5 : true
10 <= 5 : false
10 >= 5 : true
```

در مثال بالا ابتدا دو متغیر را که می‌خواهیم با هم مقایسه کنیم را ایجاد کرده و به آنها مقادیری اختصاص می‌دهیم. سپس با استفاده از یک عملگر مقایسه‌ای آنها را با هم مقایسه کرده و نتیجه را چاپ می‌کنیم. به این نکته توجه کنید که هنگام مقایسه دو متغیر از عملگر == به جای عملگر = باید استفاده شود.

عملگر = عملگر تخصیصی است و در عبارتی مانند x = y مقدار y را به x اختصاص می‌دهد. عملگر == عملگر مقایسه‌ای است که دو مقدار را با هم مقایسه می‌کند مانند x == y و اینطور خوانده می‌شود x برابر است با y.

عملگرهای بیتی

عملگرهای بیتی به شما اجازه می‌دهند که شکل باینری انواع داده‌ها را دستکاری کنید. برای درک بهتر این درس توصیه می‌شود که شما سیستم باینری و نحوه تبدیل اعداد دهدهی به باینری را از لینک زیر یاد بگیرید:

```
www.w3-farsi.com/?p=5698
```

در سیستم باینری (دودویی) که کامپیوتر از آن استفاده می‌کند وضعیت هر چیز یا خاموش است یا روشن. برای نشان دادن حالت روشن از عدد ۱ و برای نشان دادن حالت خاموش از عدد ۰ استفاده می‌شود. بنابراین اعداد باینری فقط می‌توانند صفر یا یک باشند. اعداد باینری را اعداد در مبنای ۲ و اعداد اعشاری را اعداد در مبنای ۱۰ می گویند. یک بیت نشان دهنده یک رقم باینری است و هر بایت نشان دهنده ۸ بیت است. به عنوان مثال برای یک داده از نوع Int به ۳۲ بیت یا ۸ بایت فضا برای ذخیره آن نیاز داریم، این بدین معناست که اعداد از ۳۲ رقم ۰ و ۱ برای ذخیره استفاده می‌کنند. برای مثال عدد ۱۰۰ وقتی به عنوان یک متغیر از نوع Int ذخیره می‌شود در کامپیوتر به صورت زیر خوانده می‌شود:

```
00000000000000000000000001100100
```

عدد ۱۰۰ در مبنای ده معادل عدد ۱۱۰۰۱۰۰ در مبنای ۲ است. در اینجا ۷ رقم سمت راست نشان دهنده عدد ۱۰۰ در مبنای ۲ است و مابقی صفرهای سمت راست برای پر کردن بیتهایی است که عدد از نوع Int نیاز دارد. به این نکته توجه کنید که اعداد باینری از سمت راست به چپ خوانده می‌شوند. عملگرهای بیتی کوتلین در جدول زیر نشان داده شده‌اند:

عملگر	مثال
and	x = y and z
or	x = y or z
xor	x = y xor z

inv	x = y.inv()

عملگر بیتی and

عملگر بیتی and کاری شبیه عملگر منطقی and انجام می‌دهد با این تفاوت که این عملگر بر روی بیتها کار می‌کند. اگر مقادیر دو طرف آن ۱ باشد مقدار ۱ را بر می‌گرداند و اگر یکی یا هر دو طرف آن صفر باشد مقدار صفر را بر می‌گرداند. جدول درستی عمگر بیتی and در زیر آمده است:

X	Y	X and Y
1	1	1
1	0	0
0	1	0
0	0	0

در زیر نحوه استفاده از عملگر بیتی and آمده است:

```
val result = 5 and 3

println(result)
```

```
1
```

همانطور که در مثال بالا مشاهده می‌کنید نتیجه عملکرد عملگر and بر روی دو مقدار ۵ و ۳ عدد یک می‌شود.

اجازه بدهید ببینیم که چطور این نتیجه را به دست می‌آید:

```
5:  00000000000000000000000000000101
3:  00000000000000000000000000000011
-----------------------------------
1:  00000000000000000000000000000001
```

ابتدا دو عدد ۵ و ۳ به معادل باینری‌شان تبدیل می‌شوند. از آنجاییکه هر عدد صحیح (Int) ۳۲ بیت است از صفر برای پر کردن بیتهای خالی استفاده می‌کنیم. با استفاده از جدول درستی عملگر بیتی and می‌توان فهمید که چرا نتیجه عدد یک می‌شود.

عملگر بیتی or

اگر مقادیر دو طرف عملگر بیتی or هر دو صفر باشند نتیجه صفر در غیر اینصورت ۱ خواهد شد. جدول درستی این عملگر در زیر آمده است:

X	Y	X or Y
1	1	1
1	0	1
0	1	1
0	0	0

نتیجه عملگر بیتی or در صورتی صفر است که عملوندهای دو طرف آن صفر باشند. اگر فقط یکی از دو عملوند یک باشد نتیجه یک خواهد شد. به مثال زیر توجه کنید:

```
val result = 7 or 9

println(result)
```

```
15
```

وقتی که از عملگر بیتی or برای دو مقدار در مثال بالا (۷ و ۹) استفاده می‌کنیم نتیجه ۱۵ می‌شود. حال بررسی می‌کنیم که چرا این نتیجه به دست آمده است؟

```
 7: 00000000000000000000000000000111
 9: 00000000000000000000000000001001
```

```
---------------------------------
15: 00000000000000000000000000001111
```

با استفاده از جدول درستی عملگر بیتی or می‌توان نتیجه استفاده از این عملگر را تشخیص داد. عدد ۱۱۱۱ باینری معادل عدد ۱۵ صحیح است.

عملگر بیتی xor

جدول درستی این عملگر در زیر آمده است:

X	Y	X xor Y
1	1	0
1	0	1
0	1	1
0	0	0

در صورتیکه عملوندهای دو طرف این عملگر هر دو صفر یا هر دو یک باشند نتیجه صفر در غیر اینصورت نتیجه یک می‌شود. در مثال زیر تأثیر عملگر بیتی xor را بر روی دو مقدار مشاهده می‌کنید:

```
val result = 5 xor 7

println(result)
```

```
2
```

در زیر معادل باینری اعداد بالا (۵ و ۷) نشان داده شده است.

```
5: 00000000000000000000000000000101
7: 00000000000000000000000000000111
---------------------------------
2: 00000000000000000000000000000010
```

با نگاه کردن به جدول درستی عملگر بیتی xor، می‌توان فهمید که چرا نتیجه عدد ۲ می‌شود.

عملگر بیتی inv

این عملگر یک عملگر یگانی است و فقط به یک عملوند نیاز دارد. در زیر جدول درستی این عملگر آمده است:

inv()	X
0	1
1	0

عملگر بیتی inv مقادیر بیتها را معکوس می‌کند. در زیر چگونگی استفاده از این عملگر آمده است:

```
val result = 7.inv()

println(result)
```

```
-8
```

به نمایش باینری مثال بالا که در زیر نشان داده شده است توجه نمایید.

```
7: 00000000000000000000000000000111
-----------------------------------
-8: 11111111111111111111111111111000
```

عملگر بیتی تغییر مکان

این نوع عملگرها به شما اجازه می‌دهند که بیتها را به سمت چپ یا راست جا به جا کنید. دو نوع عملگر بیتی تغییر مکان وجود دارد که هر کدام دو عملوند قبول می‌کنند. عملوند سمت چپ این عملگرها حالت باینری یک مقدار و عملوند سمت راست تعداد جابه جایی بیت‌ها را نشان می‌دهد.

عملگر	نام	مثال
shr	تغییر مکان به سمت چپ	x = y shl 2
shl	تغییر مکان به سمت راست	x = y shr 2

عملگر تغییر مکان به سمت چپ

این عملگر، بیت‌های عملوند سمت چپ را به تعداد n مکان مشخص شده، توسط عملوند سمت راست، به سمت چپ منتقل می‌کند. به عنوان مثال:

```
val result = 10 shl 2

println(result)
```

```
40
```

در مثال بالا، ما بیتهای مقدار ۱۰ را دو مکان به سمت چپ منتقل کرده‌ایم، حال بیایید تأثیر این انتقال را بررسی کنیم:

```
10: 00000000000000000000000000001010
------------------------------------
40: 00000000000000000000000000101000
```

مشاهده می‌کنید که همه بیت‌ها به اندازه دو واحد به سمت چپ منتقل شده‌اند. در این انتقال دو صفر از صفرهای سمت چپ کم می‌شود و در عوض دو صفر به سمت راست اضافه می‌شود.

عملگر تغییر مکان به سمت راست

این عملگر شبیه به عمگر تغییر مکان به سمت چپ است با این تفاوت که بیت‌ها را به سمت راست جا به جا می‌کند. به عنوان مثال:

```
val result = 100 shr 4

println(result)
```

```
6
```

با استفاده از عملگرتغییر مکان به سمت راست بیت‌های مقدار ۱۰۰ را به اندازه ۴ واحد به سمت چپ جا به جا می‌کنیم. اجازه بدهید تأثیر این جا به جایی را مورد بررسی قرار دهیم:

```
100: 00000000000000000000000001100100
-------------------------------------
  6: 00000000000000000000000000000110
```

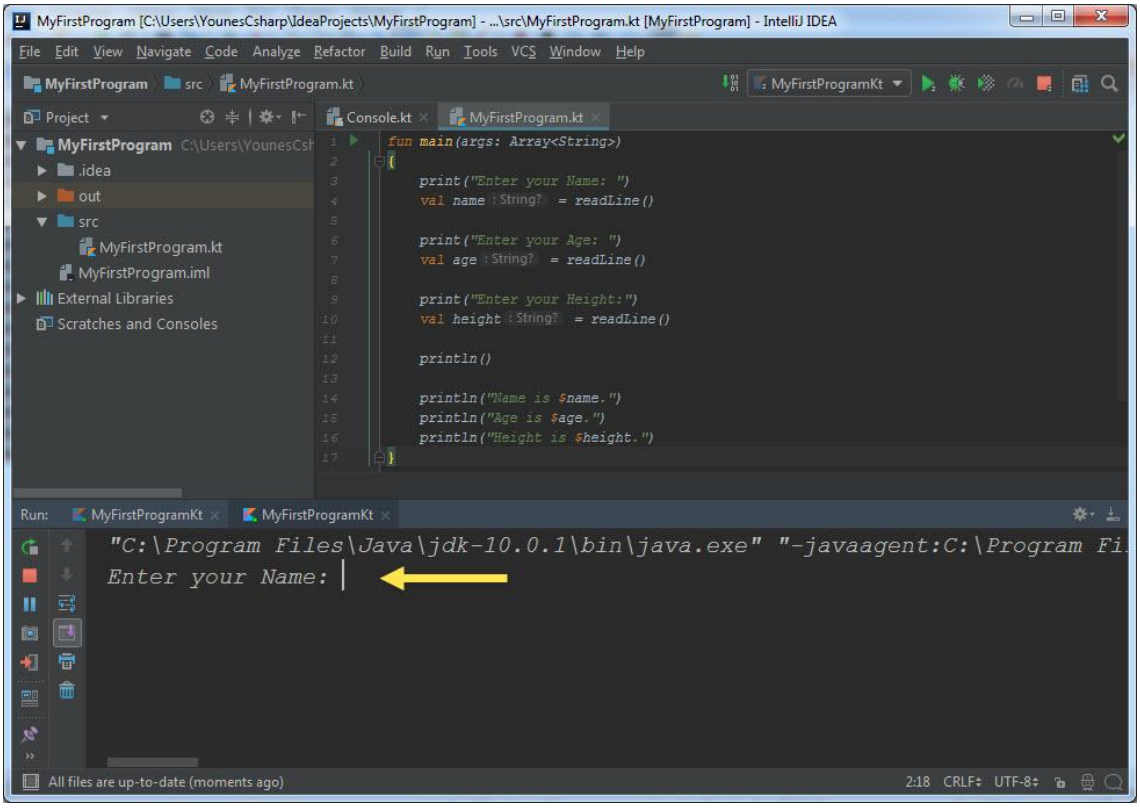
هر بیت به اندازه ۴ واحد به سمت راست منتقل می‌شود، بنابراین ۴ بیت اول سمت راست حذف شده و چهار صفر به سمت چپ اضافه می‌شود.

گرفتن ورودی از کاربر

کاتلین از تابع `readLine()` برای گرفتن ورودی از کاربر استفاده می‌کند. این تابع، تمام کاراکترهایی را که شما در محیط کنسول تایپ می‌کنید تا زمانی که دکمه `Enter` را می‌زنید، می‌خواند. به برنامه زیر توجه کنید:

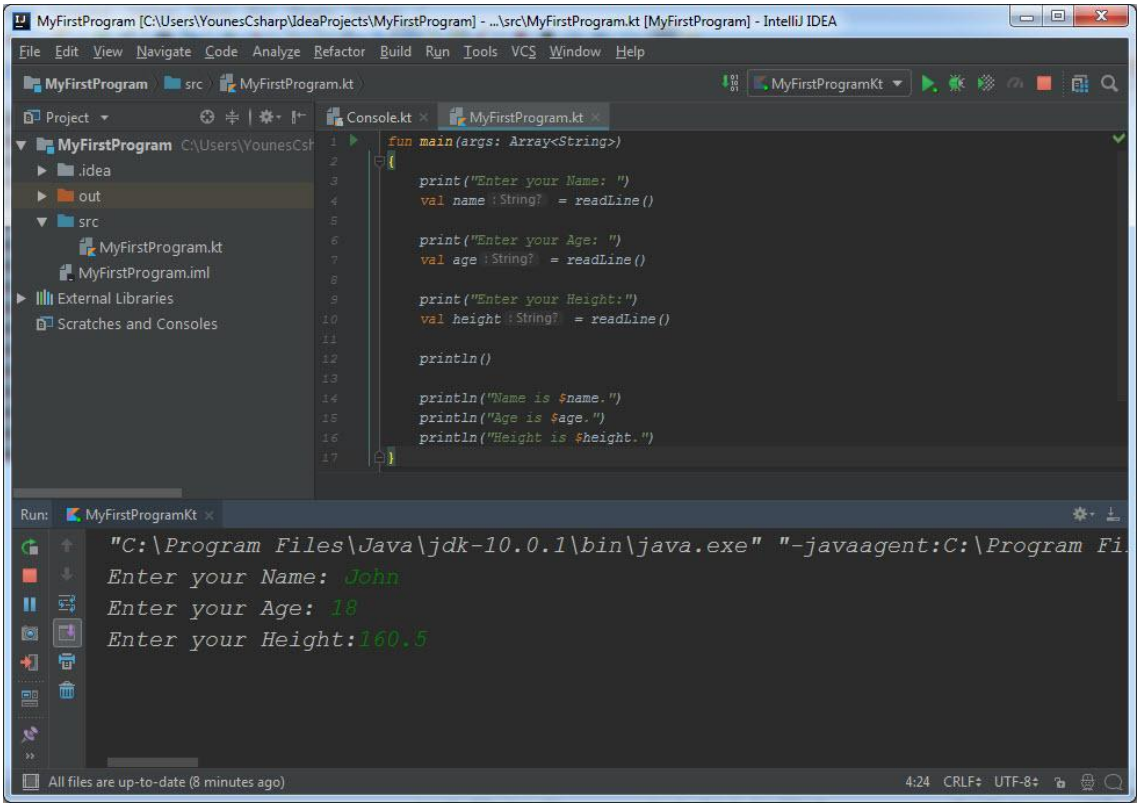
```
1  fun main(args: Array<String>)
2  {
3      print("Enter your Name: ")
4      val name = readLine()
5
6      print("Enter your Age: ")
7      val age = readLine()
8
9      print("Enter your Height: ")
10     val height = readLine()
11
12     println()
13
14     println("Name is $name.")
15     println("Age is $age.")
16     println("Height is $height.")
17 }
```

با اجرای برنامه و کلیک بر روی قسمت پایین برنامه `Intellj IDEA`، که به آن محیط کنسول هم گفته می‌شود، نشان گر ماوس نمایش داده می‌شود و برنامه از شما می‌خواهد که اطلاعات را وارد کنید:

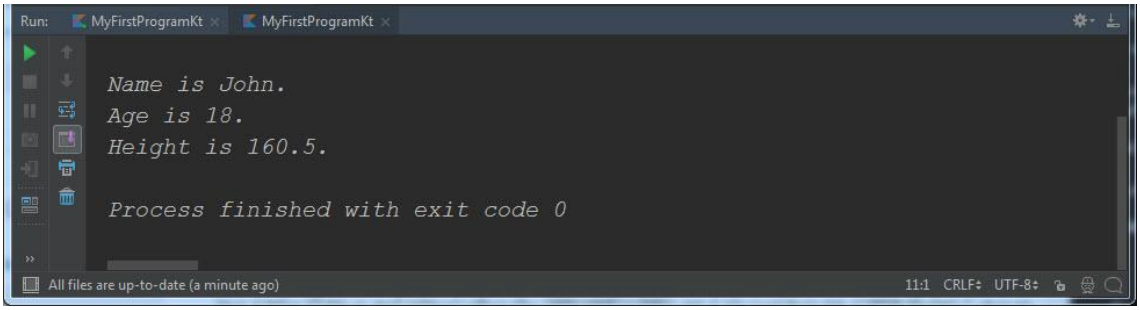


برنامه از شما نام، سن و قدتان را می‌خواهد و شما باید بعد از وارد کردن اطلاعات هر بخش، دکمه Enter را

بزنید:



با زدن دکمه Enter آخر، خروجی به صورت زیر نمایش داده می‌شود:



در همین حد کافیست که بدانید که تابع readLine() ورودی را از کاربر گرفته و آنها را به صورت رشته در متغیرهای age، name و height (خطوط ۴، ۷ و ۱۰) قرار می‌دهد. قبول دارید که سن و قد باید از نوع عددی باشند؟ اگر بخواهید فقط در محیط کنسول مقادیری را نمایش دهید که عددی یا غیر عددی بود مشکلی به وجود

نمی‌آورد. مانند مثال بالا، که ما فقط از کاربر اطلاعاتی را گرفته و چاپ کرده‌ایم. ولی اگر بخواهیم دو عدد را با هم جمع کنیم، باید چکار کنیم. به کد زیر توجه کنید:

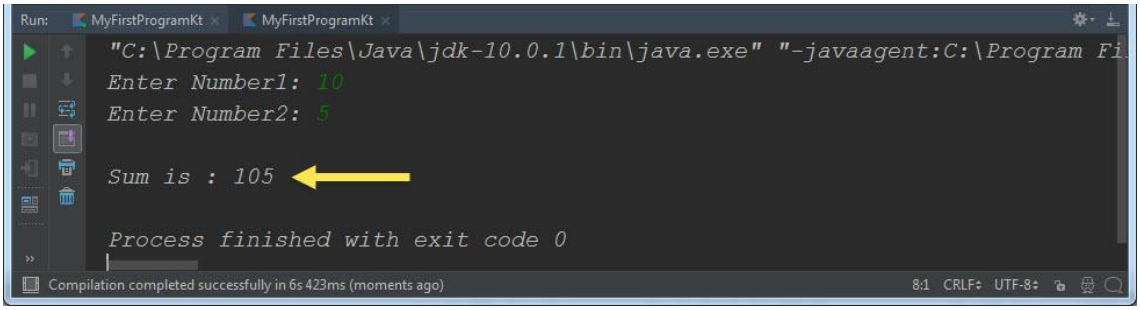
```
fun main(args: Array<String>)
{
    print("Enter Number1: ")
    val number1 = readLine()

    print("Enter Number2: ")
    val number2 = readLine()

    println()

    println("Sum is : " + (number1 + number2))
}
```

کد بالا را اجرا کرده و دو عدد را وارد نمایید:



همانطور که مشاهده می‌کنید بعد از وارد کردن اعداد ۱۰ و ۵ خروجی عدد ۱۰۵ می‌شود. البته در اصل عدد ۱۰۵ نیست، بلکه برنامه دو عدد را مانند دو رشته در نظر می‌گیرد و آنها را به هم می‌چسباند. اگر بخواهیم این دو عدد را واقعاً با هم جمع کنیم و عدد ۱۵ را به دست بیاوریم باید بعد از تابع `readLine()` از توابع تبدیل داده که در درس‌های قبل ذکر کردیم به صورت زیر استفاده نمایید:

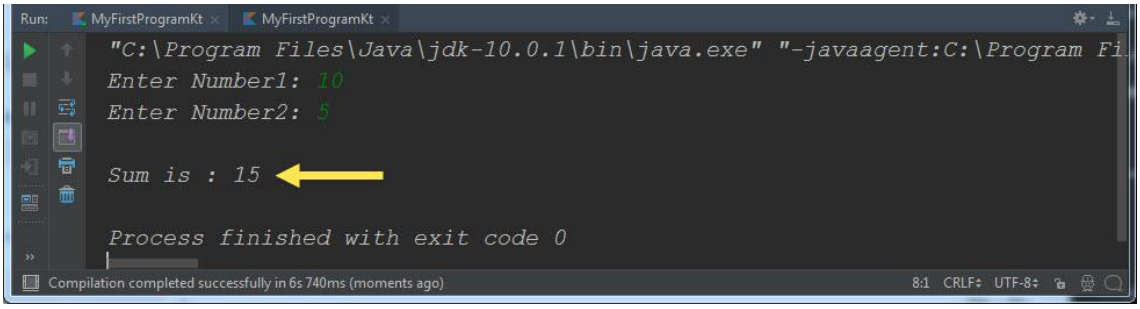
```
fun main(args: Array<String>)
{
    print("Enter Number1: ")
    val number1 = readLine()!!.toInt()

    print("Enter Number2: ")
    val number2 = readLine()!!.toInt()
```

```
    println()

    println("Sum is : " + (number1 + number2))
}
```

به خروجی کد بالا توجه کنید:



همانطور که گفتیم، تابع `readLine()` رشته در یافت می‌کند، برای تبدیل این رشته به عدد صحیح از تابع `toInt()` در استفاده کرده‌ایم. اینکه چرا قبل از تابع `toInt()` علامت !! را قرار داده‌ایم در درس‌های آینده توضیح می‌دهیم. در نتیجه تابع `toInt()` رشته گرفته شده توسط تابع `readLine()` را به نوع صحیح تبدیل کرده و در متغیرهای `number1` و `number2` می‌گذارد. با این وجود می‌توانیم دو متغیر را که عددی هستند با هم جمع کنیم.

ساختارهای تصمیم

تقریباً همه زبانهای برنامه نویسی به شما اجازه اجرای کد را در شرایط مطمئن می‌دهند. حال تصور کنید که یک برنامه دارای ساختار تصمیم گیری نباشد و همه کدها را اجرا کند. این حالت شاید فقط برای چاپ یک پیغام در صفحه مناسب باشد ولی فرض کنید که شما بخواهید اگر مقدار یک متغیر با یک عدد برابر باشد سپس یک پیغام چاپ شود آن وقت با مشکل مواجه خواهید شد. کاتلین راه‌های مختلفی برای رفع این نوع مشکلات ارائه می‌دهد.

در این بخش با مطالب زیر آشنا خواهید شد:

- دستورif
- دستور if…else
- دستور if چندگانه
- دستور if تو در تو
- دستور when

دستور if

می‌توان با استفاده از دستور if و یک شرط خاص که باعث ایجاد یک کد می‌شود یک منطق به برنامه خود اضافه کنید. دستور if ساده‌ترین دستور شرطی است که برنامه می‌گوید اگر شرطی برقرار است، کد معینی را انجام بده.

ساختار دستور if به صورت زیر است:

```
if (condition)
   code to execute
```

قبل از اجرای دستور if ابتدا شرط بررسی می‌شود. اگر شرط برقرار باشد یعنی درست باشد سپس کد اجرا می‌شود. شرط یک عبارت مقایسه‌ای است. می‌توان از عملگرهای مقایسه‌ای برای تست درست یا اشتباه بودن شرط استفاده کرد. اجازه بدهید که نگاهی به نحوه استفاده از دستور if در داخل برنامه بیندازیم. برنامه زیر پیغام Hello World را اگر مقدار number کمتر از ۱۰ و Goodbye World را اگر مقدار number از ۱۰ بزرگ‌تر باشد در صفحه نمایش می‌دهد.

```
1  fun main(args: Array<String>)
2  {
3      //Declare a variable and set it a value less than 10
4      var number = 5
5
6      //If the value of number is less than 10
7      if (number < 10)
8          println("Hello World.")
```

```
 9
10      //Change the value of a number to a value which
11      // is greater than 10
12      number = 15
13
14      //If the value of number is greater than 10
15      if (number > 10)
16          println("Goodbye World.")
17  }
```

```
Hello World.

Goodbye World.
```

در خط ۴ یک متغیر با نام number تعریف و مقدار ۵ به آن اختصاص داده شده است. وقتی به اولین دستور if در خط ۷ می‌رسیم برنامه تشخیص می‌دهد که مقدار number از ۱۰ کمتر است یعنی ۵ کوچک‌تر از ۱۰ است.

منطقی است که نتیجه مقایسه درست می‌باشد بنابراین دستور if دستور را اجرا می‌کند (خط ۸) و پیغام Hello World چاپ می‌شود. حال مقدار number را به ۱۵ تغییر می‌دهیم (خط ۱۲). وقتی به دومین دستور if در خط ۱۵ می‌رسیم برنامه مقدار number را با ۱۰ مقایسه می‌کند و چون مقدار number یعنی ۱۵ از ۱۰ بزرگ‌تر است برنامه پیغام Goodbye World را چاپ می‌کند (خط ۱۶). به این نکته توجه کنید که دستور if را می‌توان در یک خط نوشت:

```
if (number > 10) println("Goodbye World.")
```

شما می‌توانید چندین دستور را در داخل دستور if بنویسید. کافیست که از یک آکولاد برای نشان دادن ابتدا و انتهای دستورات استفاده کنید. همه دستورات داخل بین آکولاد جز بدنه دستور if هستند. نحوه تعریف چند دستور در داخل بدنه if به صورت زیر است:

```
if (condition)
{
    statement1
    statement2
    .
    .
    .
```

```
    statementN
}
```

این هم یک مثال ساده:

```
if (x > 10)
{
    println("x is greater than 10.")
    println("This is still part of the if statement.")
}
```

در مثال بالا اگر مقدار x از ۱۰ بزرگ‌تر باشد دو پیغام چاپ می‌شود. حال اگر به عنوان مثال آکولاد را حذف کنیم و مقدار x از ۱۰ بزرگ‌تر نباشد مانند کد زیر:

```
if (x > 10)
    println("x is greater than 10.");
    println("This is still part of the if statement. (Really?)");
```

کد بالا در صورتی بهتر خوانده می‌شود که بین دستورات فاصله بگذاریم.

```
if (x > 10)
    println("x is greater than 10.");

    println("This is still part of the if statement. (Really?)");
```

می‌بیند که دستور دوم (خط ۳) در مثال بالا جز دستور `if` نیست. اینجاست که چون ما فرض را بر این گذاشته‌ایم که مقدار x از ۱۰ کوچک‌تر است پس خط `This is still part of the if statement (Really?)` چاپ می‌شود. در نتیجه اهمیت وجود آکولاد مشخص می‌شود. به عنوان تمرین همیشه حتی اگر فقط یک دستور در بدنه `if` داشتید برای آن یک آکولاد بگذارید. فراموش نکنید که از قلم انداختن یک آکولاد باعث به وجود آمدن خطا شده و یافتن آن را سخت می‌کند.

دستور `if…else`

دستور if فقط برای اجرای یک حالت خاص به کار می‌رود. یعنی اگر حالتی برقرار بود، کارخاصی انجام شود. اما زمانی که شما بخواهید، اگر شرط خاصی برقرار شد، یک دستور و اگر برقرار نبود، دستور دیگر اجرا شود، باید از دستور if else استفاده کنید. ساختار دستور if else در زیر آمده است:

```
if (condition)
{
   code to execute if condition is true
}
else
{
   code to execute if condition is false
}
```

از کلمه کلیدی else نمی‌توان به تنهایی استفاده کرد بلکه حتماً باید با if به کار برده شود. اگر فقط یک کد اجرایی در داخل بدنه if و بدنه else دارید استفاده از آکولاد اختیاری است. کد داخل بلوک else فقط در صورتی اجرا می‌شود که شرط داخل دستور if نادرست باشد. در زیر نحوه استفاده از دستور if…else آمده است.

```
1  fun main(args: Array<String>)
2  {
3      var number = 5
4
5      //Test the condition
6      if (number< 10)
7      {
8          println("The number is less than 10.")
9      }
10     else
11     {
12         println("The number is either greater than or equal to 10.")
13     }
14
15     //Modify value of number
16     number = 15
17
18     //Repeat the test to yield a different result
19     if (number< 10)
20     {
21         println("The number is less than 10.")
22     }
23     else
24     {
```

```
25          println("The number is either greater than or equal to 10.")
26      }
27  }
```

```
The number is less than 10.
The number is either greater than or equal to 10.
```

در خط ۳ یک متغیر به نام number تعریف کرده‌ایم و در خط ۶ تست می‌کنیم که آیا مقدار متغیر number از ۱۰ کمتر است یا نه و چون کمتر است در نتیجه کد داخل بلوک if اجرا می‌شود (خط ۸) و اگر مقدار number را تغییر دهیم و به مقداری بزرگتر از ۱۰ تغییر دهیم (خط ۱۶)، شرط نادرست می‌شود (خط ۱۹) و کد داخل بلوک else اجرا می‌شود (خط ۲۵). مانند بلوک if نباید به آخر کلمه کلیدی else سیمیکولن اضافه شود.

دستور if تو در تو

می‌توان از دستور if تو در تو در کاتلین استفاده کرد. یک دستور ساده if در داخل دستور if دیگر.

```
if (condition)
{
    code to execute

    if (condition)
    {
         code to execute
    }
    else if (condition)
    {
        if (condition)
        {
           code to execute
        }
    }
}
else
{
    if (condition)
    {
       code to execute
    }
}
```

اجازه بدهید که نحوه استفاده از دستور if تو در تو را نشان دهیم:

```
1  fun main(args: Array<String>)
2  {
3      val age = 21
4
5      if (age > 12)
6      {
7          if (age< 20)
8          {
9              println("You are teenage")
10         }
11         else
12         {
13             println("You are already an adult.")
14         }
15     }
16     else
17     {
18         println("You are still too young.")
19     }
20 }
```

```
You are already an adult.
```

اجازه بدهید که برنامه را کالبد شکافی کنیم. ابتدا در خط ۳ یک متغیر به نام age تعریف می‌کنیم و مقدار آن را برابر ۲۱ قرار می‌دهیم. سپس به اولین دستور if می‌رسیم (خط ۵). در این قسمت اگر سن شما بیشتر از ۱۲ سال باشد برنامه وارد بدنه دستور if می‌شود در غیر اینصورت وارد بلوک else (خط ۱۸) مربوط به همین دستور if می‌شود.

حال فرض کنیم که ن شما بیشتر از ۱۲ سال است و شما وارد بدنه اولین if شده‌اید. در بدنه اولین if یک دستور if دیگر را مشاهده می‌کنید. اگر سن کمتر ۲۰ باشد دستور You are teenage چاپ می‌شود (خط ۹) در غیر اینصورت دستور You are already an adult (خط ۱۳) و چون مقدار متغیر تعریف شده در خط 7 بزرگ‌تر از ۲۰ است پس دستور مربوط به بخش else خط ۱۳ چاپ می‌شود. حال فرض کنید که مقدار متغیر age کمتر از ۱۲ بود، در این صورت دستور بخش else خط ۱۸ یعنی You are still too young چاپ می‌شد. پیشنهاد می‌شود که از if تو در تو در برنامه کمتر استفاده کنید چون خوانایی برنامه را پایین می‌آورد.

دستور if چندگانه

اگر بخواهید چند شرط را بررسی کنید چکار می‌کنید؟ می‌توانید از چندین دستور if استفاده کنید و بهتر است که

این دستورات if را به صورت زیر بنویسید:

```
if (condition)
{
   code to execute
}
else
{
   if (condition)
   {
      code to execute
   }
   else
   {
      if (condition)
      {
         code to execute
      }
      else
      {
         code to execute
      }
   }
}
```

خواندن کد بالا سخت است. بهتر است دستورات را به صورت تو رفتگی در داخل بلوک else بنویسید. می‌توانید

کد بالا را ساده‌تر کنید:

```
if (condition)
{
   code to execute
}
else if (condition)
{
   code to execute
}
else if (condition)
{
   code to execute
}
else
{
```

```
    code to execute
}
```

حال که نحوه استفده از دستور else if را یاد گرفتید باید بدانید که مانند else، else if نیز به دستور if وابسته است. دستور else if وقتی اجرا می‌شود که اولین دستور if اشتباه باشد حال اگر else if اشتباه باشد دستور else if بعدی اجرا می‌شود. و اگر آن نیز اجرا نشود در نهایت دستور else اجرا می‌شود. برنامه زیر نحوه استفاده از دستور if else را نشان می‌دهد:

```
fun main(args: Array<String>)
{
    val choice = 2

    if (choice == 1)
    {
        print("You might like my black t-shirt.")
    }
    else if (choice == 2)
    {
        print("You might be a clean and tidy person.")
    }
    else if (choice == 3)
    {
        print("You might be sad today.")
    }
    else
    {
        print("Sorry, your favorite color is not in the choices above.")
    }
}
```

```
You might be a clean and tidy person
```

خروجی برنامه بالا به متغیر choice وابسته است. بسته به اینکه شما چه چیزی انتخاب می‌کنید پیغامهای مختلفی چاپ می‌شود. اگر عددی که شما تایپ می‌کنید در داخل حالتهای انتخاب نباشد کد مربوط به بلوک else اجرا می‌شود.

استفاده از عملگرهای منطقی

عملگرهای منطقی به شما اجازه می‌دهند که چندین شرط را با هم ترکیب کنید. این عملگرها حداقل دو شرط را درگیر می‌کنند و در آخر یک مقدار بولی را بر می‌گردانند. در جدول زیر برخی از عملگرهای منطقی آمده است:

عملگر	تلفظ	مثال	تأثیر
&&	and	z = (x > 2) && (y < 10)	مقدار Z در صورتی true است که هر دو شرط دو طرف عملگر مقدارشان true باشد. اگر فقط مقدار یکی از شروط false باشد مقدار z، false خواهد شد.
||	or	z = (x > 2) || (y < 10)	مقدار Z در صورتی true است که یکی از دو شرط دو طرف عملگر مقدارشان true باشد. اگر هر دو شرط مقدارشان false باشد مقدار z، false خواهد شد.
!	not	z = !(x > 2)	مقدار Z در صورتی true است که مقدار شرط false باشد و در صورتی false است که مقدار شرط true باشد.

به عنوان مثال جمله z = (x > 2) && (y < 10)را به این صورت بخوانید: "در صورتی مقدار z برابر true است که مقدار x بزرگ‌تر از ۲ و مقدار y کوچک‌تر از ۱۰ باشد در غیر اینصورت false است". این جمله بدین معناست که برای اینکه مقدار کل دستور true باشد باید مقدار همه شروط true باشد. عملگر منطقی || تأثیر متفاوتی نسبت به عملگر منطقی && دارد. نتیجه عملگر منطقی || برابر true است اگر فقط مقدار یکی از شروط true باشد. و اگر مقدار هیچ یک از شروط true نباشد نتیجه false خواهد شد. می‌توان عملگرهای منطقی && و || را با هم ترکیب کرده و در یک عبارت به کار برد مانند:

```
if ( (x == 1) && ( (y > 3) || z < 10) )
{
    //do something here
}
```

در اینجا استفاده از پرانتز مهم است چون از آن در گروه بندی شرطها استفاده می‌کنیم. در اینجا ابتدا عبارت (y > 3) || (z < 10) مورد بررسی قرار می‌گیرد. (به علت تقدم عملگرها) سپس نتیجه آن بوسیله عملگر && با نتیجه (x == 1) مقایسه می‌شود. حال بیایید نحوه استفاده از عملگرهای منطقی در برنامه را مورد بررسی قرار دهیم:

```
1  fun main(args: Array<String>)
2  {
3      print("Enter your age: ")
4      val age = readLine() !!.toInt()
5
6      print("Enter your gender (male/female):")
7      val gender = readLine()
8
9      if (age > 12 && age < 20)
10     {
11         if (gender == "male")
12         {
13             println("You are a teenage boy.")
14         }
15         else
16         {
17             println("You are not a teenage girl.")
18         }
19     }
20     else
21     {
22         println("You are not a teenager.")
23     }
24 }
```

```
 Enter your age: 18
Enter your gender (male/female): female
You are a teenage girl.
Enter you age: 10
Enter your gender (male/female): male
You are not a teenager.
```

برنامه بالا نحوه استفاده از عملگر منطقی && را نشان می‌دهد (خط ۹). وقتی به دستور if می‌رسید (خط ۹) برنامه سن شما را چک می‌کند. اگر سن شما بزرگتر از ۱۲ و کوچکتر از ۲۰ باشد (سنتان بین ۱۲ و ۲۰ باشد) یعنی

مقدار هر دو true باشد سپس کدهای داخل بلوک if اجرا می‌شوند. اگر نتیجه یکی از شروط false باشد کدهای داخل بلوک else اجرا می‌شود.

عملگر && عملوند سمت چپ را مورد بررسی قرار می‌دهد. اگر مقدار آن false باشد دیگر عملوند سمت راست را بررسی نمی‌کند و مقدار false را بر می‌گرداند. بر عکس عملگر || عملوند سمت چپ را مورد بررسی قرار می‌دهد و اگر مقدار آن true باشد سپس عملوند سمت راست را نادیده می‌گیرد و مقدار true را بر می‌گرداند.

```
if ((x == 2) and (y == 3))
{
    //Some code here
}

if ((x == 2) or (y == 3))
{
    //Some code here
}
```

شما می‌توانید از عملگرهای and و or بیتی هم در شرط‌ها استفاده کنید. این عملگرها دو عملوند را بدون در نظر گرفتن مقدار عملوند سمت چپ مورد بررسی قرار می‌دهند. به عنوان مثال حتی اگر مقدار عملوند سمت چپ false باشد عملوند سمت چپ به وسیله عملگر بیتی & ارزیابی می‌شود.

اگر شرطها را در برنامه ترکیب کنید استفاده از عملگرهای منطقی && و || به جای عملگرهای بیتی and و or بهتر خواهد بود:

```
if ((age > 12) and (age < 20))
```

یکی دیگر از عملگرهای منطقی عملگر ! است که نتیجه یک عبارت را خنثی یا منفی می‌کند. به مثال زیر توجه کنید:

```
if (x != 2)
{
    println("x is not equal to 2.")
```

```
}
```

اگر نتیجه عبارت `x == 2` برابر `false` باشد عملگر ! آن را `true` می‌کند. می‌توان خط ۹ کد بالا را به صورت زیر هم نوشت:

```
if(age in 12 .. 20)
```

دستور when

در کاتلین ساختاری به نام when وجود دارد که به شما اجازه می‌دهد که با توجه به مقدار ثابت یک متغیر چندین انتخاب داشته باشید. دستور when معادل دستور `if` تو در تو است با این تفاوت که در دستور when متغیر فقط مقادیر ثابتی از اعداد، رشته‌ها و یا کاراکترها را قبول می‌کند. مقادیر ثابت مقادیری هستند که قابل تغیر نیستند.

در زیر نحوه استفاده از دستور when آمده است:

```
when (testVar)
{
  compareVal1 -> code to execute if testVar == compareVa11
  compareVal1 -> code to execute if testVar == compareVa12
  .
  .
  compareVal1 -> code to execute if testVar == compareVa13
  else        -> code to execute if none of the values above match the testVar
}
```

ابتدا یک مقدار در متغیر when که در مثال بالا `testVar` است قرار می‌دهید. این مقدار با هر یک از مقادیر قبل از علامت `->` داخل بلوک when مقایسه می‌شود. اگر مقدار متغیر با هر یک از مقادیر موجود برابر بود دستور مقابل علامت `->` اجرا خواهد شد. به این نکته توجه کنید که حتی اگر تعداد خط کدهای داخل دستور case از یکی بیشتر باشد نباید از آکولاد استفاده کنیم. دستور when یک بخش `else` دارد. این دستور در صورتی اجرا می‌شود که مقدار متغیر با هیچ یک از مقادیر موجود در بدنه برابر نباشد. دستور else اختیاری است و اگر از بدنه

when حذف شود هیچ اتفاقی نمی‌افتد. مکان این دستور مهم است و باید آن را در پایان دستورات بنویسید. به

مثالی در مورد دستور when توجه کنید:

```
fun main(args: Array<String>)
{
    println("What's your favorite pet?")
    println("[1] Dog")
    println("[2] Cat")
    println("[3] Rabbit")
    println("[4] Turtle")
    println("[5] Fish")
    println("[6] Not in the choices")
    print("Enter your choice: ")

    val choice: Int = readLine () !!. toInt ()

    when (choice)
    {
        1 -> println("Your favorite pet is Dog.")
        2 -> println("Your favorite pet is Cat.")
        3 -> println("Your favorite pet is Rabbit.")
        4 -> println("Your favorite pet is Turtle.")
        5 -> println("Your favorite pet is Fish.")
        6 -> println("Your favorite pet is not in the choices.")
        else -> println("You don't have a favorite pet.")
    }
}
```

```
What's your favorite pet?
[1] Dog
[2] Cat
[3] Rabbit
[4] Turtle
[5] Fish
[6] Not in the choices

Enter your choice: 2
Your favorite pet is Cat.
What's your favorite pet?
[1] Dog
[2] Cat
[3] Rabbit
[4] Turtle
[5] Fish
[6] Not in the choices

Enter your choice: 99
You don't have a favorite pet.
```

برنامه بالا به شما اجازه انتخاب حیوان مورد علاقه‌تان را می‌دهد. به اسم هر حیوان یک عدد نسبت داده شده است. شما عدد را وارد می‌کنید و این عدد در دستور when با مقادیر خطوط ۲۱-۱۶ مقایسه می‌شود و اگر با هر کدام از آن مقادیر که برابر بود پیغام مناسب نمایش داده خواهد شد. اگر هم با هیچ کدام از مقادیر برابر نبود دستور else اجرا می‌شود. یکی دیگر از ویژگیهای دستور when این است که شما می‌توانید از دو یا چند -> برای نشان داده یک مجموعه کد استفاده کنید. در مثال زیر اگر مقدار number، اعداد ۱، ۲ یا ۳ باشد یک کد اجرا می‌شود:

```
when (number)
{
    1, 2, 3 -> println("This code is shared by three values.")
}
```

به ازای صحت یک شرط می‌توان دو یا چند دستور را هم اجرا کرد:

```
when (number)
{
    1 -> println("Your favorite pet is Dog.")
    2 -> {
            println("statment1")
            println("statment2")
         }
    3 -> println("Your favorite pet is Cat.")
}
```

مثلاً در کد بالا اگر مقدار number برابر ۲ باشد، دو دستور اجرا می‌شود. از عملگر in هم می‌توان در بدنه when به صورت زیر استفاده کرد:

```
when (number)
{
    in 1..10 -> println("Number between 1 .. 10")
}
```

در کد بالا اگر مقدار number عددی بین محدوده ۱ تا ۱۰ باشد، دستور چاپ می‌شود. همانطور که قبلاً ذکر شد دستور when معادل دستور if تو در تو است. برنامه بالا را به صورت زیر نیز می‌توان نوشت:

```
if (choice == 1)
    println("Your favorite pet is Dog.")
else if (choice == 2)
    println("Your favorite pet is Cat.")
else if (choice == 3)
    println("Your favorite pet is Rabbit.")
else if (choice == 4)
    println("Your favorite pet is Turtle.")
else if (choice == 5)
    println("Your favorite pet is Fish.")
else if (choice == 6)
    println("Your favorite pet is not in the choices.")
else
    println("You don't have a favorite pet.")
```

کد بالا دقیقاً نتیجه‌ای مانند دستور when دارد.

تکرار

ساختارهای تکرار، به شما اجازه می‌دهند که یک یا چند دستور کد را، تا زمانی که یک شرط برقرار است، تکرار کنید. بدون ساختارهای تکرار، شما مجبورید همان تعداد کدها را بنویسید، که بسیار خسته کننده است. مثلاً شما مجبورید ۱۰ بار جمله "Hello World." را تایپ کنید مانند مثال زیر:

```
println("Hello World.");
println("Hello World.");
println("Hello World.");
println("Hello World.");
println("Hello World.");
println("Hello World.");
println("Hello World.");
println("Hello World.");
println("Hello World.");
println("Hello World.");
```

البته شما می‌توانید با کپی کردن این تعداد کد را راحت بنویسید ولی این کار در کل کیفیت کدنویسی را پایین می‌آورد. را بهتر برای نوشتن کدهای بالا استفاده از حلقه‌ها است. ساختارهای تکرار در کاتلین عبارتند از:

- while
- do while

- for

While حلقه

ابتدایی‌ترین ساختار تکرار در کاتلین حلقه While است. ابتدا یک شرط را مورد بررسی قرار می‌دهد و تا زمانیکه شرط برقرار باشد کدهای درون بلوک اجرا می‌شوند. ساختار این حلقه به صورت زیر است:

```
while(condition)
{
    code to loop
}
```

می‌بینید که ساختار While مانند ساختار if بسیار ساده است. ابتدا یک شرط را که نتیجه آن یک مقدار بولی است، می‌نویسیم. اگر نتیجه درست یا true باشد، سپس کدهای داخل بلوک While اجرا می‌شوند. اگر شرط غلط یا false باشد وقتی که برنامه به حلقه While برسد هیچکدام از کدها را اجرا نمی‌کند. برای متوقف شدن حلقه باید مقادیر داخل حلقه While اصلاح شوند.

به یک متغیر شمارنده در داخل بدنه حلقه نیاز داریم. این شمارنده برای آزمایش شرط مورد استفاده قرار می‌گیرد و ادامه یا توقف حلقه به نوعی به آن وابسته است. این شمارنده را در داخل بدنه باید کاهش یا افزایش دهیم.

در برنامه زیر نحوه استفاده از حلقه While آمده است:

```
1  fun main(args: Array<String>)
2  {
3      var counter = 1
4
5      while (counter <= 10)
6      {
7          println("Hello World!")
8          counter++
9      }
10 }
```

```
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
```

```
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
```

برنامه بالا ۱۰ بار پیغام Hello World! را چاپ می‌کند. اگر از حلقه در مثال بالا استفاده نمی‌کردیم، مجبور بودیم تمام ۱۰ خط را تایپ کنیم. اجازه دهید که نگاهی به کدهای برنامه فوق بیندازیم. ابتدا در خط ۳ یک متغیر تعریف و از آن به عنوان شمارنده حلقه استفاده شده است. سپس به آن مقدار ۱ را اختصاص می‌دهیم چون اگر مقدار نداشته باشد، نمی‌توان در شرط از آن استفاده کرد.

در خط ۵ حلقه While را وارد می‌کنیم. در این حلقه ابتدا مقدار اولیه شمارنده با ۱۰ مقایسه می‌شود که آیا از ۱۰ کمتر است یا با آن برابر است. نتیجه هر بار مقایسه ورود به بدنه حلقه و چاپ پیغام است. همانطور که مشاهده می‌کنید بعد از هر بار مقایسه مقدار شمارنده یک واحد اضافه می‌شود (خط ۸). حلقه تا زمانی تکرار می‌شود که مقدار شمارنده از ۱۰ کمتر باشد.

اگر مقدار شمارنده یک بماند و آن را افزایش ندهیم و یا مقدار شرط هرگز false نشود یک حلقه بینهایت به وجود می‌آید. به این نکته توجه کنید که در شرط بالا به جای علامت <از <= استفاده شده است. اگر از علامت < استفاده می‌کردیم کد ما ۹ بار تکرار می‌شد چون مقدار اولیه ۱ است و هنگامی که شرط به ۱۰ برسد false می‌شود، چون 10 < 10 نیست. اگر می‌خواهید یک حلقه بی نهایت ایجاد کنید که هیچگاه متوقف نشود باید یک شرط ایجاد کنید که همواره درست (true) باشد.

```
while(true)
{
   //code to loop
}
```

این تکنیک در برخی موارد کارایی دارد و آن زمانی است که شما بخواهید با استفاده از دستورات break و return که در آینده توضیح خواهیم داد از حلقه خارج شوید.

حلقه do While

حلقه do while یکی دیگر از ساختارهای تکرار است. این حلقه بسیار شبیه حلقه while است با این تفاوت که در این حلقه ابتدا کد اجرا می‌شود و سپس شرط مورد بررسی قرار می‌گیرد. ساختار حلقه dowhile به صورت زیر است:

```
do
{
   code to repeat;

} while (condition);
```

همانطور که مشاهده می‌کنید شرط در آخر ساختار قرار دارد. این بدین معنی است که کدهای داخل بدنه حداقل یکبار اجرا می‌شوند. برخلاف حلقه while که اگر شرط نادرست باشد دستورات داخل بدنه اجرا نمی‌شوند. یکی از موارد برتری استفاده از حلقه do while نسبت به حلقه while، زمانی است که، شما بخواهید اطلاعاتی از کاربر دریافت کنید. به مثال زیر توجه کنید:

استفاده از while

```
//while version

print("Enter a number greater than 10: ")
var number = readLine() !!.toInt()

while(number < 10)
{
    println("Enter a number greater than 10: ")
    number = readLine() !!.toInt()
}
```

استفاده از do while

```
//do while version

do
{
    println("Enter a number greater than 10: ")
```

```
    var number = readLine() !!.toInt()
}
while(number < 10)
```

مشاهده می‌کنید که از کدهای کمتری در بدنه do while نسبت به while استفاده شده است.

حلقه for

یکی دیگر از ساختارهای تکرار، حلقه for است. این حلقه عملی شبیه به حلقه while انجام می‌دهد و از آن برای گردش در میان یک مجموعه (collection) یا محدوده (range) استفاده می‌شود. ساختار حلقه for به صورت زیر است:

```
for (item in collection/range)
{
    //Some Code
}
```

در زیر یک مثال از حلقه for آمده است:

```
1  fun main(args: Array<String>)
2  {
3      for (i in 1..10)
4      {
5          println("Number $i")
6      }
7  }
```

```
Number 1
Number 2
Number 3
Number 4
Number 5
Number 6
Number 7
Number 8
Number 9
Number 10
```

برنامه بالا اعداد ۱ تا ۱۰ را با استفاده از حلقه for چاپ می‌کند. در خط ۳ و داخل پرانتز همانطور که مشاهده می‌کنید عبارت ۱..۱۰ آمده است که در اصل یک محدود است. برای ایجاد یک محدوده در کوتلین ابتدا و انتهای

مجموعه را مشخص کرده و سپس با استفاده از علامت.. از هم جدا می‌کنیم. عبارت ۱۰..۱ به معنای محدود اعداد از ۱ تا ۱۰ است. در کل معنای خط ۳ این است که به ازای اعداد ۱ تا ۱۰ ... فلان کار را انجام بده. چه کاری؟ کاری که در خط ۵ خواسته‌ایم و آن چاپ مقدار اعداد این محدود است. در اصل در خط ۳ با هر تکرار یکی از اعداد مجموعه در داخل متغیر موقتی `i` قرار می‌گیرد و در خط ۵ چاپ می‌شود و این کار تا چاپ آخرین مقدار یعنی ۱۰ ادامه می‌یابد. اگر بخواهید عدد انتهای محدود یعنی ۱۰ چاپ نشود باید از کلمه `until` استفاده نمایید:

```
for (i in 1 until 10)
```

و اگر بخواهید معکوس برنامه بالا را پیاده سازی کنید، یعنی اعداد از بزرگ به کوچک چاپ شوند، باید خط ۳ را به صورت زیر تغییر دهید:

```
for (i in 10 downTo 1)
```

عبارت `downTo` به معنای رو به پایین است و معنای کلی خط بالا: "به ازای اعداد از ۱۰ تا ۱" می‌باشد. حال فرض کنید که می‌خواهید چند عدد در میان این محدود را چاپ کنید. برای این منظور از کلمه `step` باید استفاده نمایید:

```
for (i in 1..10 step 2)
```

کد بالا یک در میان اعداد را چاپ می‌کند. برای مثلاً ۲ در میان یا سه در میان کافیست عدد جلوی این کلمه را به ۲ یا ۳ و یا هر عددی که مایل هستید، تغییر دهید. از حلقه `for` برای چاپ حروف یک رشته هم می‌توان استفاده کرد. چون رشته یک مجموعه از کاراکترهاست:

```
fun main(args: Array<String>)
{
    for (letter in "Hello World!")
    {
```

```
        println(letter)
    }
}
```

```
H
e
l
l
o

W
o
r
l
d
!
```

خارج شدن از حلقه با استفاده از break و continue

گاهی اوقات با وجود درست بودن شرط می‌خواهیم حلقه متوقف شود. سؤال اینجاست که چطور این کار را انجام دهید؟ با استفاده از کلمه کلیدی breakحلقه را متوقف کرده و با استفاده از کلمه کلیدی continue می‌توان بخشی از حلقه را رد کرد و به مرحله بعد رفت. برنامه زیر نحوه استفاده از continue و break را نشان می‌دهد :

```
1  fun main(args: Array<String>)
2  {
3      println("Demonstrating the use of break\n")
4
5      for (x in 1..9)
6      {
7          if (x == 5)
8              break
9
10         println("Number $x")
11     }
12
13     println("\nDemonstrating the use of continue\n")
14
15     for (x in 1..9)
16     {
17         if (x == 5)
18             continue
19
20         println("Number $x")
21     }
22 }
```

```
Demonstrating the use of break.

Number 1
Number 2
Number 3
Number 4

Demonstrating the use of continue.

Number 1
Number 2
Number 3
Number 4
Number 6
Number 7
Number 8
Number 9
```

در این برنامه از حلقه for برای نشان دادن کاربرد دو کلمه کلیدی فوق استفاده شده است اگر به جای for از حلقه‌های while و do…while استفاده می‌شد نتیجه یکسانی به دست می‌آمد. همانطور که در شرط برنامه (خط ۷) آمده است وقتی که مقدار x به عدد ۵ رسید سپس دستور break اجرا شود (خط ۸). حلقه بلافاصله متوقف می‌شود حتی اگر شرط x < 10 برقرار باشد. از طرف دیگر در خط ۱۷ حلقه for فقط برای یک تکرار خاص متوقف شده و سپس ادامه می‌یابد (وقتی مقدار x برابر ۵ شود حلقه از عدد ۵ رد شده و مقدار ۵ را چاپ نمی‌کند و بقیه مقادیر چاپ می‌شوند).

آرایه‌ها

آرایه، نوعی متغیر است که، لیستی از آدرسهای، مجموعه‌ای از داده‌های هم نوع را، در خود ذخیره می‌کند. تعریف چندین متغیر از یک نوع، برای هدفی یکسان، بسیار خسته کننده است. مثلاً اگر بخواهید صد متغیر از نوع اعداد صحیح تعریف کرده و از آنها استفاده کنید. مطمئناً تعریف این همه متغیر بسیار کسالت آور و خسته کننده است. اما با استفاده از آرایه می‌توان همه آنها را در یک خط تعریف کرد. در زیر راهی ساده برای تعریف یک آرایه نشان داده شده است:

```
val arrayName = datatypeArray(length)
```

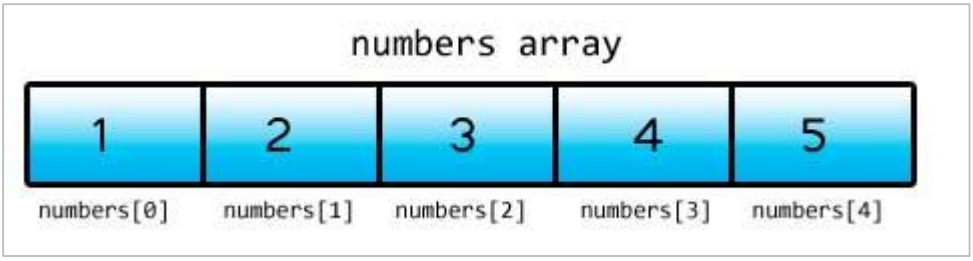
arrayName که نام آرایه را نشان می‌دهد. هنگام نامگذاری آرایه بهتر است که نام آرایه نشان دهنده نوع آرایه باشد. به عنوان مثال برای نامگذاری آرایه‌ای که اعداد را در خود ذخیره می‌کند از کلمه number استفاده کنید. بعد از علامت مساوی، کلاس‌های در کاتلین وجود دارند که نام آنها نشان دهنده نوع آرایه‌ای است که قرار است ایجاد کنیم. مثلاً برای ایجاد یک آرایه از نوع اعداد صحیح یا Int کلاسی با نام IntArray وجود دارد که در داخل سازنده آن طول (length) آرایه را می‌نویسیم. طول آرایه که به کامپایلر می‌گوید شما قصد دارید چه تعداد داده یا مقدار را در آرایه ذخیره کنید. حال برای تعریف یک آرایه که ۵ مقدار از نوع اعداد صحیح در خود ذخیره می‌کند باید به صورت زیر عمل کنیم:

```
val numbers = IntArray(5)
```

در این مثال ۵ آدرس از فضای حافظه کامپیوتر شما برای ذخیره ۵ مقدار رزرو می‌شود. حال چطور مقادیرمان را در هر یک از این آدرسها ذخیره کنیم؟ برای دسترسی و اصلاح مقادیر آرایه از اندیس یا مکان آنها استفاده می‌شود.

```
numbers[0] = 1
numbers[1] = 2
numbers[2] = 3
numbers[3] = 4
numbers[4] = 5
```

اندیس یک آرایه از صفر شروع شده و به یک واحد کمتر از طول آرایه ختم می‌شود. به عنوان مثال شما یک آرایه ۵ عضوی دارید، اندیس آرایه از ۰ تا ۴ می‌باشد چون طول آرایه ۵ است، پس ۱-۵ برابر است با ۴. این بدان معناست که اندیس ۰ نشان دهنده اولین عضو آرایه است و اندیس ۱ نشان دهنده دومین عضو و الی آخر. برای درک بهتر مثال بالا به شکل زیر توجه کنید:



به هر یک از اجزاء آرایه و اندیسهای داخل کروشه توجه کنید. کسانی که تازه شروع به برنامه نویسی کرده‌اند معمولاً در گذاشتن اندیس دچار اشتباه می‌شوند و مثلاً ممکن است در مثال بالا اندیسها را از ۱ شروع کنند. اگر بخواهید به یکی از اجزائ آرایه با استفاده از اندیسی دسترسی پیدا کنید که در محدوده اندیسهای آرایه شما نباشد با پیغام خطای `ArrayIndexOutOfBoundsException` مواجه می‌شوید و بدین معنی است که شما آدرسی را می‌خواهید که وجود ندارد. کلاس‌های دیگری برای تعریف انواع مختلف آرایه وجود دارند که در زیر به آنها اشاره شده است:

- `FloatArray`
- `StringArray`
- `DoubleArray`
- `CharArray`
- `ByteArray`
- `LongArray`
- `ShortArray`

کلاس‌های بالا، فقط حافظه را رزرو می‌کنند و شما باید به روشی که در ابتدای درس اشاره شد و با استفاده از اندیس، مقدار عناصر را مشخص کنید. یک راه بسیار ساده‌تر برای تعریف آرایه به صورت زیر است:

```
val numbers = intArrayOf(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

به سادگی و بدون احتیاج به اندیس می‌توان مقادیر را در داخل پرانتز قرار داد. کامپایلر به صورت اتوماتیک با شمارش مقادیر طول آرایه را تشخیص می‌دهد و به ترتیب از ۰، به هر کدام از عناصر داخل آرایه یک اندیس اختصاص می‌دهد. مثلاً اندیس عدد ۵ در کد بالا ۴ است. به اسم تابع `intArrayOf()` توجه کنید. توابع دیگری با همین ساختار وجود دارند، که با استفاده از آن‌ها می‌توان فوراً آرایه را مقدار دهی کرد. در زیر لیست این توابع آمده است:

- `floatArrayOf`
- `stringArrayOf`
- `doubleArrayOf`
- `charArrayOf`
- `byteArrayOf`
- `longArrayOf`
- `shortArrayOf`

دستیابی به مقادیر آرایه با استفاده از حلقه for

در زیر مثالی در مورد استفاده از آرایه‌ها آمده است. در این برنامه ۵ مقدار از کاربر گرفته شده و میانگین آنها حساب می‌شود:

```
fun main(args: Array<String>)
{
    val numbers = IntArray(5)
    var total = 0
    val average: Double

    for (i in numbers.indices)
    {
        print("Enter a number: ")
        numbers[i] = readLine()!!.toInt()
    }
    for (i in numbers.indices)
```

```
13      {
14          total += numbers[i]
15      }
16
17      average = total / numbers.size.toDouble()
18
19      println("Average = $average")
20  }
```

در خط ۳ یک آرایه تعریف شده است که می‌تواند ۵ عدد صحیح را در خود ذخیره کند. خطوط ۴ و ۵ متغیرهایی تعریف شده‌اند که از آنها برای محاسبه میانگین استفاده می‌شود. توجه کنید که مقدار اولیه `total` صفر است تا از بروز خطا هنگام اضافه شدن مقدار به آن جلوگیری شود. در خطوط ۷ تا ۱۱ حلقه `for` برای تکرار و گرفتن ورودی از کاربر تعریف شده است. از خاصیت `indices` آرایه برای تشخیص تعداد اجزای آرایه استفاده می‌شود. اگر چه می‌توانستیم به سادگی در حلقه `for` مقدار ۵ را برای شرط قرار دهیم ولی استفاده از خاصیت `indices`، کار راحت‌تری است و می‌توانیم طول آرایه را تغییر دهیم و شرط حلقه `for` با تغییر جدید هماهنگ می‌شود. در خط ۱۰ ورودی دریافت شده از کاربر با استفاده از تابع `readLine()` دریافت و سپس به نوع صحیح تبدیل و در آرایه ذخیره می‌شود. اندیس استفاده شده در `number` (خط ۱۰) مقدار `i` جاری در حلقه است. برای مثال در ابتدای حلقه مقدار `i`، عدد ۰ است. بنابراین وقتی در خط ۱۰ اولین داده از کاربر گرفته می‌شود، اندیس آن برابر ۰ می‌شود. در تکرار بعدی `i` یک واحد اضافه می‌شود و در نتیجه در خط ۱۰ و بعد از ورود دومین داده توسط کاربر اندیس آن برابر ۱ می‌شود. این حالت تا زمانی که شرط در حلقه `for` برقرار است ادامه می‌یابد. در خطوط ۱۵-۱۲ از حلقه `for` دیگر برای دسترسی به مقدار هر یک از داده‌های آرایه استفاده شده است. در این حلقه نیز مانند حلقه قبل از مقدار متغیر شمارنده به عنوان اندیس استفاده می‌کنیم.

هر یک از اجزای عددی آرایه به متغیر `total` اضافه می‌شوند. بعد از پایان حلقه می‌توانیم میانگین اعداد را حساب کنیم (خط ۱۷). مقدار `total` را بر تعداد اجزای آرایه (تعداد عددها) تقسیم می‌کنیم. برای دسترسی به تعداد اجزای آرایه می‌توان از خاصیت `size` آرایه استفاده کرد. توجه کنید که در اینجا ما مقدار خاصیت `size`

را به نوع Double تبدیل کرده‌ایم. بنابراین نتیجه عبارت یک مقدار از نوع Double خواهد شد و دارای بخش کسری می‌باشد. حال اگر عملوندهای تقسیم را به نوع Double تبدیل نکنیم نتیجه تقسیم یک عدد از نوع صحیح خواهد شد و دارای بخش کسری نیست. خط ۱۹ مقدار میانگین را در صفحه نمایش چاپ می‌کند. طول آرایه بعد از مقدار دهی نمی‌تواند تغییر کند. به عنوان مثال اگر یک آرایه را که شامل ۵ جز است مقداردهی کنید، دیگر نمی‌توانید آن را مثلاً به ۱۰ جز تغییر اندازه دهید. البته تعداد خاصی از کلاسها مانند آرایه‌ها عمل می‌کنند و توانایی تغییر تعداد اجزای تشکیل دهنده خود را دارند. آرایه‌ها در برخی شرایط بسیار پر کاربرد هستند و تسلط شما بر این مفهوم و اینکه چطور از آنها استفاده کنید بسیار مهم است.

آرایه‌های چند بعدی

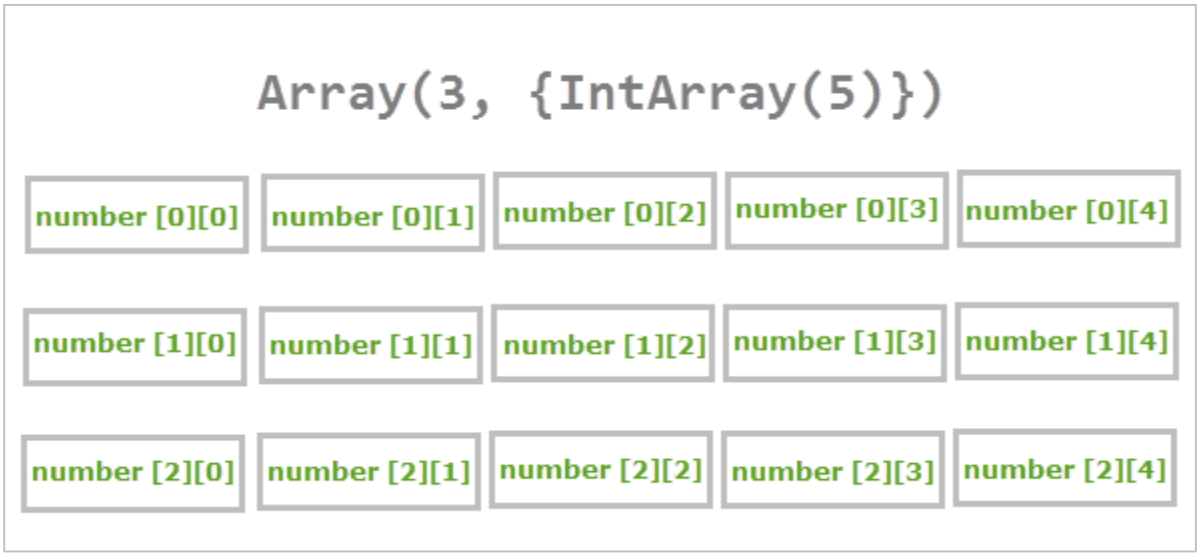
آرایه‌های چند بعدی، آرایه‌هایی هستند که برای دسترسی به هر یک از عناصر آنها باید از چندین اندیس استفاده کنیم. یک آرایه چند بعدی را می‌توان مانند یک جدول با تعدای ستون و ردیف تصور کنید. با افزایش اندیسها اندازه ابعاد آرایه نیز افزایش می‌یابد و آرایه‌های چند بعدی با بیش از دو اندیس به وجود می‌آیند. نحوه ایجاد یک آرایه با دو بعد به صورت زیر است:

```
val arrayName = Array(row, {typeArray(column)})
```

Array یک کلاس است که سازنده آن دو پارامتر می‌گیرد. یکی تعداد آرایه‌ها (row) و دیگری نوع آرایه که به وسیله کلاس‌هایی که در درس قبل ذکر شد (مانند IntArray) به همراه تعداد عناصر آن (column).کد بالا را به صورت زیر هم می‌توان نوشت:

```
val arrayName = Array(row) {typeArray(column)}
```

می‌توان یک آرایه با تعداد زیادی بعد ایجاد کرد به شرطی که هر بعد دارای طول مشخصی باشد. در یک آرایه دو بعدی برای دسترسی به هر یک از عناصر به دو مقدار نیاز داریم یکی مقدار X و دیگری مقدار Y که مقدار x نشان دهنده ردیف و مقدار Y نشان دهنده ستون آرایه است البته اگر ما آرایه دو بعدی را به صورت جدول در نظر بگیریم. در شکل زیر مکان هر عنصر در یک آرایه دو بعدی نشان داده شده است.



مقدار ۳ را به x اختصاص می‌دهیم چون ۳ سطر یا آرایه داریم و مقدار ۵ را به Y چون هر آرایه ۵ ستون دارد، اختصاص می‌دهیم. چطور یک آرایه چند بعدی را مقدار دهی کنیم؟ در این حالت که از کلاس Array استفاده کرده‌ایم باید مقدار دهی به عناصر را به صورت دستی انجام داد مانند:

```
val numbers = Array(3, {IntArray(5)})

numbers[0][0] = 1
numbers[0][1] = 2
numbers[0][2] = 3
numbers[0][3] = 4
numbers[0][4] = 5
numbers[1][0] = 6
numbers[1][1] = 7
numbers[1][2] = 8
numbers[1][3] = 9
numbers[1][4] = 10
numbers[2][0] = 11
numbers[2][1] = 12
```

```
numbers[2][2] = 13
numbers[2][3] = 14
numbers[2][4] = 15
```

همانطور که مشاهده می‌کنید برای دسترسی به هر یک از عناصر در یک آرایه دو بعدی به سادگی می‌توان از اندیسهای X و Y و یک جفت کروشه مانند مثال استفاده کرد. یک راه دیگر برای تعریف یک آرایه دو بعدی استفاده از تابع arrayOf() است. به عنوان مثال:

```
val numbers = arrayOf(
                        intArrayOf( 1,  2,  3,  4,  5),
                        intArrayOf( 6,  7,  8,  9, 10),
                        intArrayOf(11, 12, 13, 14, 15)

                      )
```

این تابع آرایه‌ای از آرایه‌ها را دریافت می‌کند. ما در کد بالا یک آرایه به نام number تعریف کرده‌ایم که سه آرایه از نوع اعداد صحیح در خود جای داده است و هر آرایه هم ۵ عنصر یا ستون دارد. یعنی آرایه number دارای ۳ سطر و ۵ ستون است.

گردش در میان عناصر آرایه‌های چند بعدی

گردش در میان عناصر آرایه‌های چند بعدی نیاز به کمی دقت دارد. یکی از راههای آسان استفاده از حلقه for تو در تو است:

```
1  fun main(args: Array<String>)
2  {
3      val numbers = arrayOf(
4                                intArrayOf(1, 2, 3, 4, 5),
5                                intArrayOf(6, 7, 8, 9, 10),
6                                intArrayOf(11, 12, 13, 14, 15)
7                              )
8
9      for (number in numbers)
10     {
11         for (num in number)
12         {
13             print(num.toString() + " ")
14         }
```

```
    }
}
```

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

مشاهده کردید که گردش در میان مقادیر عناصر یک آرایه چند بعدی چقدر راحت است. در برنامه بالا از یک حلقه برای گردش در میان آرایه‌ها و از حلقه دیگر برای گردش در میان عناصر آنها استفاده کرده‌ایم. ولی این کد ناقص است و تمام مقادیر را در یک خط چاپ می‌کند. اگر بخواهیم مقادیر مربوط به هر آرایه را در یک خط چاپ کنیم باید چکار کنیم؟ به برنامه زیر توجه کنید:

```
fun main(args: Array<String>)
{
    val numbers = arrayOf(
            intArrayOf(1, 2, 3, 4, 5),
            intArrayOf(6, 7, 8, 9, 10),
            intArrayOf(11, 12, 13, 14, 15)
    )

    for (row in numbers.indices)
    {
        for (col in 0 until numbers[row].size)
        {
            print(numbers[row][col].toString() + " ")
        }

        //Go to the next line
        println()
    }
}
```

```
1 2 3 4 5
6 7 8 9 10
15 14 13 12 11
```

همانطور که در مثال بالا نشان داده شده است با استفاده از یک حلقه ساده for نمی‌توان به مقادیر دسترسی یافت بلکه به یک حلقه for تو در تو نیاز داریم. به زبان ساده در کد بالا دو حلقه for داریم. یکی برای به دست آوردن تعداد آرایه‌ها و دیگری برای به دست آوردن ستون یا تعداد عناصر آرایه‌ها. در اولین حلقه for، با استفاده از خاصیت indices، اندیس آرایه‌های موجود در آرایه numbers که در اصل همان تعداد آرایه‌ها می‌باشد را به دست می‌آوریم. این خط را می‌توان به صورت زیر هم نوشت:

```
for(row in 0..2)
```

چرا ۰..۲؟ چون اندیس آرایه‌ها هم از صفر شروع می‌شود. یعنی اندیس اولین آرایه دارای اندیس ۰ و .... در حلقه for دوم با استفاده از خاصیت size، تعداد ستون‌های یک آرایه را به دست می‌آوریم. و چون تعداد عناص یا ستون‌های همه آرایه‌ها برابر ۵ است پس عبارت numbers[row].size عدد ۵ را بر می‌گرداند. در نتیجه خط ۱۱ را می‌توان به صورت زیر هم نوشت:

```
for (col in 0 until 5)
```

در مجموع می‌توان گفت که row اعداد ۰، ۱ و ۲ را در خود جای می‌دهد و column هم اعداد ۰، ۱، ۲، ۳ و ۴. در نتیجه وقتی که مقدار ردیف 0 (row) باشد، حلقه دوم از [۰][۰] تا [۴][۰] اجرا شود. سپس مقدار هر عنصر از آرایه را با استفاده از حلقه نشان می‌دهیم، اگر مقدار ردیف (row) برابر ۰ و مقدار ستون (col) برابر ۰ باشد مقدار عنصری که در ستون ۱ و ردیف ۱ (numbers[0][0]) قرار دارد نشان داده خواهد شد که در مثال بالا عدد ۱ است.

بعد از اینکه دومین حلقه تکرار به پایان رسید، فوراً دستورات بعد از آن اجرا خواهند شد، که در اینجا دستور println() که به برنامه اطلاع می‌دهد که به خط بعد برود (خط ۱۷). سپس حلقه با اضافه کردن یک واحد به مقدار row این فرایند را دوباره تکرار می‌کند. سپس دومین حلقه for اجرا شده و مقادیر دومین ردیف نمایش داده می‌شود. حال بیایید آنچه را از قبل یاد گرفته‌ایم در یک برنامه به کار بریم. این برنامه نمره چهار درس مربوط به سه دانش آموز را از ما می‌گیرد و معدل سه دانش آموز را حساب می‌کند.

```
1  fun main(args: Array<String>)
2  {
3      val studentGrades = Array(3) { DoubleArray(4) }
4      var total: Double
5
```

```
    for (student in studentGrades.indices)
    {
        total = 0.0

        println("Enter grades for Student ${student + 1}")

        for (grade in 0 until studentGrades[student].size)
        {
            print("Enter Grade #${grade + 1}: ")
            studentGrades[student][grade] = readLine() !!.toDouble()
            total += studentGrades[student][grade]
        }

        print("Average is ${total / studentGrades[student].size}")
        println()
    }
}
```

```
Enter grades for Student 1
Enter Grade #1: 92
Enter Grade #2: 87
Enter Grade #3: 89
Enter Grade #4: 95
Average is 90.75

Enter grades for Student 2
Enter Grade #1: 85
Enter Grade #2: 85
Enter Grade #3: 86
Enter Grade #4: 87
Average is 85.75

Enter grades for Student 3
Enter Grade #1: 90
Enter Grade #2: 90
Enter Grade #3: 90
Enter Grade #4: 90
Average is 90.00
```

در برنامه بالا یک آرایه چند بعدی از نوع `Double` تعریف شده است (خط ۳). این آرایه ۳ آرایه دارد و هر آرایه ۴ عنصر. در کل ۳ دانش آموز و هر دانش آموز ۴ درس. همچنین یک متغیر به نام `total` تعریف می‌کنیم که مقدار محاسبه شده معدل هر دانش آموز را در آن قرار دهیم (خط ۴). حال وارد حلقه `for` تو در تو می‌شویم (خط ۶). در اولین حلقه `for` تعداد دانش آموزان را با استفاده از خاصیت `indices` به دست می‌آوریم. در نتیجه متغیر `student` اعداد ۰، ۱ و ۲ را در خود ذخیره می‌کند. وارد بدنه حلقه `for` می‌شویم. در خط ۸ مقدار

متغیر total را برابر صفر قرار می‌دهیم. بعداً مشاهده می‌کنید که چرا این کار را انجام دادیم. سپس برنامه یک پیغام را نشان می‌دهد و از شما می‌خواهد که شماره دانش آموز را وارد کنید (student + 1). عدد ۱ را به student اضافه کرده‌ایم تا به جای نمایش student 0، با student 1 شروع شود، تا طبیعی‌تر به نظر برسد. سپس به دومین حلقه for در خط ۱۲ می‌رسیم. در این حلقه یک متغیر شمارنده به نام grade تعریف می‌کنیم و تعداد عناصر آرایه را با استفاده از studentGrades[student].size را در آن قرار می‌دهیم. این متغیر اعداد ۰، ۱، ۲ و ۳ را در خود جای می‌دهد. برنامه چهار نمره مربوط به دانش آموز را می‌گیرد. هر وقت که برنامه یک نمره را از کاربر دریافت می‌کند، نمره به متغیر total اضافه می‌شود.

وقتی همه نمره‌ها وارد شدند، متغیر total هم جمع همه نمرات را نشان می‌دهد. در خط ۱۹ معدل دانش آموز نشان داده می‌شود. معدل از تقسیم کردن total (جمع) بر تعداد نمرات به دست می‌آید.

آرایه دندانه دار

آرایه دندانه دار یا jagged array آرایه‌ای چند بعدی است که دارای سطرهای با طول متغیر می‌باشد. نمونه ساده‌ای از آرایه‌های چند بعدی، آرایه‌های مستطیلی است که تعداد ستون‌ها و سطرهای آنها برابر است. اما آرایه‌های دندانه دار دارای سطرهای (آرایه‌های) با طول متفاوت می‌باشند. بنابراین آرایه‌های دندانه دار را می‌توان آرایه‌ای از آرایه‌ها فرض کرد. دستور نوشتن این نوع آرایه‌ها به صورت زیر است:

```
val arrayName: arrayOfNulls<datatypeArray>(lenght)
```

datatypeArray نوع آرایه است و lenght هم طول آرایه یا تعداد آرایه‌های این آرایه دندانه دار را مشخص می‌کند. مقداردهی به این آرایه‌ها کمی گیج کننده است. به مثال زیر توجه کنید:

```
val myArrays = arrayOfNulls<IntArray>(3)
```

```
myArrays[0] = IntArray(3)
myArrays[1] = IntArray(5)
myArrays[2] = IntArray(2)
```

در کد بالا سه آرایه تعریف شده است که اندیس آنها از صفر شروع می‌شود. اعداد ۳ و ۵ و ۲ هم به تعداد عناصری که هر کدام از آنها در خود می‌توانند جای دهند، اشاره دارند. برای مقداردهی هر آرایه هم باید ابتدا اندیس آرایه و سپس اندیس عناصر آن را بنویسیم. مثلاً مقداردهی اولین عنصر اولین آرایه مثال بالا به صورت زیر عمل می‌کنیم :

```
myArrays[0]!![0] = 1;
```

و برای مثلاً دومین عنصر دومین آرایه هم به صورت زیر:

```
myArrays[1]!![1] = 4;
```

با توجه به توضیحاتی که داده شد می‌توان عناصر آرایه‌ای که در ابتدای درس ایجاد کردیم را به صورت زیر مقداردهی کرد:

```
myArrays[0]!![0] = 1
myArrays[0]!![1] = 2
myArrays[0]!![2] = 3

myArrays[1]!![0] = 5
myArrays[1]!![1] = 4
myArrays[1]!![2] = 3
myArrays[1]!![3] = 2
myArrays[1]!![4] = 1

myArrays[2]!![0] =11
myArrays[2]!![1] =22
```

یک روش بهتر برای مقدار دهی آرایه‌های دندانه دار به صورت زیر است که در آن می‌توان طول سطرها را هم مشخص نکرد:

```
val myArrays = arrayOf(
                          intArrayOf(1, 2, 3),
```

```
                        intArrayOf(5, 4, 3, 2, 1),
                        intArrayOf(11, 22)

                    )
```

برای دسترسی به مقدار عناصر یک آرایه دندانه دار باید اندیس سطر و ستون آن را در اختیار داشته باشیم:

```
array[row]!![column]
----------------------------
println(myArrays[1]!![2]);
```

نمی‌توان از یک حلقه for برای دسترسی به عناصر آرایه دندانه دار استفاده کرد:

```
for(int array in myArrays)
{
    println(array);
}
```

اگر از یک حلقه for استفاده کنیم با خطا مواجه می‌شویم چون عناصر این نوع آرایه‌ها، آرایه هستند، نه عدد یا رشته یا .... برای حل این مشکل باید نوع متغیر موقتی (array) را تغییر داده و از یک حلقه for دیگری برای دسترسی به مقادیر استفاده کرد.

```
for (array in myArrays)
{
    if (array != null)
    {
        for (number in array)
        {
            println(number)
        }
    }
}
```

می‌توان از کد زیر هم استفاده کرد:

```
for (row in 0 until myArrays.size)
{
    for (col in 0 until myArrays[row]!!.size)
    {
        println(myArrays[row]!![col])
    }
}
```

در اولین حلقه از `size` برای به دست آوردن تعداد سطرها (که همان آرایه‌های یک بعدی هستند) و در دومین حلقه از `size` برای به دست آوردن عناصر سطر جاری استفاده می‌شود.

تابع

توابع به شما اجازه می‌دهند که یک رفتار یا وظیفه را تعریف کنید و مجموعه‌ای از کدها هستند که در هر جای برنامه می‌توان از آنها استفاده کرد. توابع دارای آرگومانهایی هستند که وظیفه تابع را مشخص می‌کنند. تابع در داخل کلاس تعریف می‌شود. می‌توان یک تابع را در داخل تابع دیگر تعریف کرد. وقتی که شما در برنامه یک تابع را صدا می‌زنید برنامه به قسمت تعریف تابع رفته و کدهای آن را اجرا می‌کند. در کوتلین تابعی وجود دارد که نقطه آغاز هر برنامه است و بدون آن برنامه‌ها نمی‌دانند باید از کجا شروع شوند، این تابع `main()` نام دارد.

- پارامترها همان چیزهایی هستند که تابع منتظر دریافت آنها است.
- آرگومان‌ها مقادیری هستند که به پارامترها ارسال می‌شوند.

گاهی اوقات دو کلمه پارامتر و آرگومان به یک منظور به کار می‌روند. ساده‌ترین ساختار یک تابع به صورت زیر است:

```
fun functionName()
{
   code to execute;
}
```

به برنامه ساده زیر توجه کنید. در این برنامه از یک تابع برای چاپ یک پیغام در صفحه نمایش استفاده شده است:

```
fun printMessage()
{
    println("Hello World!")
}
```

```
fun main(args: Array<String>)
{
    printMessage()
}
```

```
Hello World!
```

در خطوط ۸-۵ یک تابع تعریف کرده‌ایم. مکان تعریف آن در برنامه مهم نیست. به عنوان مثال می‌توانید آن را زیر تابع `main()` تعریف کنید. می‌توان این تابع را در داخل تابع دیگر صدا زد (فراخوانی کرد). تابع دیگر ما در اینجا تابع `main()` است که می‌توانیم در داخل آن نام تابعی که برای چاپ یک پیغام تعریف کرده‌ایم (یعنی تابع `printMessage()`) را صدا بزنیم.

در تعریف تابع بالا ابتدا کلمه کلیدی `fun` آمده است که نشان دهنده آن است که ما می‌خواهیم یک تابع تعریف کنیم. نام تابع ما `printMessage()` است.

به این نکته توجه کنید که در نامگذاری تابع از روش کوهان شتری (اولین کلمه با حروف کوچک نوشته می‌شود و بقیه کلمات با حرف بزرگ شروع می‌شوند) استفاده کرده‌ایم. این روش نامگذاری قراردادی است و می‌توان از این روش استفاده نکرد، اما پیشنهاد می‌شود که از این روش برای تشخیص توابع استفاده کنید. بهتر است در نامگذاری توابع از کلماتی استفاده شود که کار آن تابع را مشخص می‌کند مثلاً نام‌هایی مانند `goToBed` یا `openDoor` . همچنین به عنوان مثال اگر مقدار برگشتی تابع یک مقدار بولی باشد می‌توانید اسم تابع خود را به صورت یک کلمه سوالی انتخاب کنید مانند `isLeapyear` یا `isTeenager` … . ولی از گذاشتن علامت سؤال در آخر اسم تابع خودداری کنید. دو پرانتزی که بعد از نام می‌آید، نشان دهنده آن است که، نام متعلق به یک تابع است. در این مثال در داخل پرانتزها هیچ چیزی نوشته نشده چون پارامتری ندارد. در درسهای آینده در مورد توابع بیشتر توضیح می‌دهیم.

بعد از پرانتزها دو آکولاد قرار می‌دهیم که بدنه تابع را تشکیل می‌دهد و کدهایی را که می‌خواهیم اجرا شوند را در داخل این آکولادها می‌نویسیم. در داخل تابع `main()` تابعی را که در خط ۱ ایجاد کرده‌ایم را صدا می‌زنیم. برای صدا زدن یک تابع کافیست نام آن را نوشته و بعد از نام پرانتزها را قرار دهیم.

اگر تابع دارای پارامتر باشد باید شما آرگومانها را به ترتیب در داخل پرانتزها قرار دهید. در این مورد نیز در درسهای آینده توضیح بیشتری می‌دهیم. با صدا زدن یک تابع کدهای داخل بدنه آن اجرا می‌شوند. برای اجرای تابع `printMessage()` برنامه از تابع `main()` به محل تعریف تابع `printMessage()` می‌رود. مثلاً وقتی ما تابع `printMessage()` را در خط ۸ صدا می‌زنیم برنامه از خط ۸ به خط ۱، یعنی جایی که تابع تعریف شده می‌رود و کدهای بدنه آن یعنی خط ۳ را اجرا می‌کند.

مقدار برگشتی از یک تابع

توابع، می‌توانند مقدار برگشتی از هر نوع داده‌ای داشته باشند. این مقادیر می‌توانند در محاسبات یا به دست آوردن یک داده مورد استفاده قرار بگیرند. در زندگی روزمره فرض کنید که کارمند شما یک تابع است و شما او را صدا می‌زنید و از او می‌خواهید که کار یک سند را به پایان برساند. سپس از او می‌خواهید که بعد از اتمام کارش سند را به شما تحویل دهد. سند همان مقدار برگشتی تابع است. نکته مهم در مورد یک تابع، مقدار برگشتی و نحوه استفاده شما از آن است. برگشت یک مقدار از یک تابع آسان است. کافیست در تعریف تابع به روش زیر عمل کنید:

```
fun functionName(): returnType
{
   return value;
}
```

`returnType` در اینجا نوع داده‌ای مقدار برگشتی را مشخص می‌کند (`Float`، `Int` و ...). در داخل بدنه تابع کلمه کلیدی `return` و بعد از آن یک مقدار یا عبارتی که نتیجه آن، یک مقدار است را می‌نویسیم. نوع این

مقدار برگشتی باید از انواع ساده بوده و در هنگام نامگذاری تابع و بعد از پرانتزها و علامت دو نقطه ذکر شود.

مثال زیر یک تابع که دارای مقدار برگشتی است را نشان می‌دهد:

```
1  fun calculateSum(): Int
2  {
3      val firstNumber = 10
4      val secondNumber = 5
5
6      return firstNumber + secondNumber
7  }
8
9  fun main(args: Array<String>)
10 {
11     val result = calculateSum()
12
13     println("Sum is $result.")
14 }
```

```
Sum is 15.
```

همانطور که در خط ۱ مثال فوق مشاهده می‌کنید هنگام تعریف تابع و بعد از علامت پرانتز بسته، از `Int:` استفاده کرده‌ایم که نشان دهنده آن است که تابع ما دارای مقدار برگشتی از نوع اعداد صحیح است. در خطوط ۳ و ۴ دو متغیر تعریف و مقدار دهی شده‌اند.

توجه کنید که این متغیرها، متغیرهای محلی هستند. و این بدان معنی است که این متغیرها در سایر توابع مانند تابع `main()` قابل دسترسی نیستند و فقط در تابعی که در آن تعریف شده‌اند، قابل استفاده هستند. در خط ۶ جمع دو متغیر توسط دستور `return` برگشت داده می‌شود. در داخل تابع `main()` یک متغیر به نام `result` در خط ۱۱ تعریف می‌کنیم و تابع `calculateSum()` را فراخوانی می‌کنیم.

تابع `calculateSum()` مقدار ۱۵ را بر می‌گرداند که در داخل متغیر `result` ذخیره می‌شود. در خط ۱۳ مقدار ذخیره شده در متغیر `result` چاپ می‌شود. تابعی که در این مثال ذکر شد تابع کاربردی و مفیدی نیست. با وجودیکه کدهای زیادی در تابع بالا نوشته شده ولی همیشه مقدار برگشتی ۱۵ است، در حالیکه می‌توانستیم به راحتی یک متغیر تعریف کرده و مقدار ۱۵ را به آن اختصاص دهیم. این تابع در صورتی کارامد است که پارامترهایی

به آن اضافه شود که در درسهای آینده توضیح خواهیم داد. هنگامی که می‌خواهیم در داخل یک تابع از دستور if یا when استفاده کنیم باید خود این دستورات را با کلمه return برگشت دهیم. برای درک بهتر این مطلب به مثال زیر توجه کنید:

```
1   fun getNumber(): Int
2   {
3       print("Enter a number greater than 10: ")
4       val number = readLine() !!.toInt()
5
6       return if (number > 10)
7       {
8           number
9       }
10      else
11      {
12          0
13      }
14  }
15
16  fun main(args: Array<String>)
17  {
18      val result = getNumber()
19
20      println("Result = $result.")
21  }
```

```
Enter a number greater than 10: 11
Result = 11
Enter a number greater than 10: 9
Result = 0
```

در خطوط ۱-۱۴ یک تابع با نام getNumber() تعریف شده است که از کاربر یک عدد بزرگتر از ۱۰ را می‌خواهد. اگر عدد وارد شده توسط کاربر درست نباشد تابع مقدار صفر را بر می‌گرداند. اگر قسمت else دستور if و یا دستور return را از آن حذف کنیم در هنگام اجرای برنامه با پیغام خطا مواجه می‌شویم.

چون اگر شرط دستور if نادرست باشد (کاربر مقداری کمتر از ۱۰ را وارد کند) برنامه به قسمت else می‌رود تا مقدار ۰ را بر گرداند و چون قسمت else حذف شده است برنامه با خطا مواجه می‌شود و همچنین اگر دستور

return حذف شود چون برنامه نیاز به مقدار برگشتی دارد، پیغام خطا می‌دهد. خطوط ۱۳-۶ را به صورت زیر هم می‌توان نوشت:

```
if (number > 10)
{
    return number
}
else
{
    return 0
}
```

و آخرین مطلبی که در این درس می‌خواهیم به شما آموزش دهیم این است که شما می‌توانید از یک تابع که مقدار برگشتی ندارد، با استفاده از کلمه کلیدی return خارج شوید. استفاده از return باعث خروج از بدنه تابع و اجرای کدهای بعد از آن می‌شود:

```
fun testReturnExit()
{
    println("Line 1 inside the method TestReturnExit()")
    println("Line 2 inside the method TestReturnExit()")

    return

    //The following lines will not execute
    println("Line 3 inside the method TestReturnExit()")
    println("Line 4 inside the method TestReturnExit()")
}

fun main(args: Array<String>)
{
    testReturnExit()
    println("Hello World!")
}
```

```
Line 1 inside the method TestReturnExit()
Line 2 inside the method TestReturnExit()
Hello World!
```

در برنامه بالا نحوه خروج از تابع با استفاده از کلمه کلیدی return و نادیده گرفتن همه کدهای بعد از این کلمه کلیدی نشان داده شده است. در پایان برنامه تابع تعریف شده (testReturnExit()) در داخل تابع main() فراخوانی و اجرا می‌شود.

پارامتر و آرگومان

پارامترها، داده‌های خامی هستند که، تابع آنها را پردازش می‌کند و سپس اطلاعاتی را که به دنبال آن هستید، در اختیار شما قرار می‌دهد. فرض کنید پارامترها مانند اطلاعاتی هستند که، شما به یک کارمند می‌دهید که بر طبق آنها کارش را به پایان برساند. یک تابع می‌تواند هر تعداد پارامتر داشته باشد. هر پارامتر می‌تواند از انواع مختلف داده باشد. در زیر یک تابع با N پارامتر نشان داده شده است:

```
fun functionName(param1: datatype , param2: datatype , ... paramN: datatype)
{
   code to execute;
}
```

پارامترها، بعد از نام تابع و بین پرانتزها قرار می‌گیرند. بر اساس کاری که تابع انجام می‌دهد، می‌توان تعداد پارامترهای زیادی به تابع اضافه کرد. بعد از فراخوانی یک تابع باید آرگومانهای آن را نیز تأمین کنید. آرگومان‌ها، مقادیری هستند که به پارامترها اختصاص داده می‌شوند. ترتیب ارسال آرگومانها به پارامترها مهم است. عدم رعایت ترتیب در ارسال آرگومانها باعث به وجود آمدن خطای منطقی و خطای زمان اجرا می‌شود. اجازه بدهید که یک مثال بزنیم:

```
1   fun calculateSum(number1: Int, number2: Int): Int
2   {
3       return number1 + number2
4   }
5
6   fun main(args: Array<String>)
7   {
8       print("Enter the first number: ")
9       val num1 = readLine() !!.toInt()
10      print("Enter the second number: ")
11      val num2 = readLine() !!.toInt()
12
13      println("Sum = ${calculateSum(num1, num2)}")
14  }
```

```
Enter the first number: 10
Enter the second number: 5
Sum = 15
```

در برنامه بالا یک تابع به نام calculateSum() (خطوط ۴-۱) تعریف شده است که وظیفه آن جمع مقدار دو عدد است. چون این تابع مقدار دو عدد صحیح را با هم جمع می‌کند پس نوع برگشتی ما نیز باید int باشد. تابع دارای دو پارامتر است که اعداد را به آنها ارسال می‌کنیم. به نوع داده‌ای پارامترها توجه کنید. هر دو پارامتر یعنی number1 و number2 مقادیری از نوع اعداد صحیح (int) دریافت می‌کنند. در بدنه تابع دستور return نتیجه جمع دو عدد را بر می‌گرداند. در داخل تابع main() برنامه از کاربر دو مقدار را درخواست می‌کند و آنها را داخل متغیرها قرار می‌دهد. حال تابع را که آرگومانهای آن را آماده کرده‌ایم فراخوانی می‌کنیم. مقدار num1 به پارامتر اول و مقدار num2 به پارامتر دوم ارسال می‌شود. حال اگر مکان دو مقدار را هنگام ارسال به تابع تغییر دهیم (یعنی مقدار num2 به پارامتر اول و مقدار num1 به پارامتر دوم ارسال شود) هیچ تغییری در نتیجه تابع ندارد چون جمع خاصیت جابه جایی دارد.

فقط به یاد داشته باشید که باید ترتیب ارسال آرگومانها هنگام فراخوانی تابع دقیقاً با ترتیب قرار گیری پارامترها تعریف شده در تابع مطابقت داشته باشد. بعد از ارسال مقادیر ۱۰ و ۵ به پارامترها، پارامترها آنها را دریافت می‌کنند. به این نکته نیز توجه کنید که نام پارامترها طبق قرارداد به شیوه کوهان شتری یا camelCasing (حرف اول دومین کلمه بزرگ نوشته می‌شود) نوشته می‌شود. در داخل بدنه تابع (خط ۳) دو مقدار با هم جمع می‌شوند و نتیجه به تابع فراخوان (تابعی که تابع calculateSum() را فراخوانی می‌کند) ارسال می‌شود.

در درس آینده از یک متغیر برای ذخیره نتیجه محاسبات استفاده می‌کنیم ولی در اینجا مشاهده می‌کنید که می‌توان به سادگی نتیجه جمع را نشان داد (خط ۳). در داخل تابع main() از ما دو عدد که قرار است با هم جمع شوند درخواست می‌شود.

در خط ۱۳ تابع calculateSum() را فراخوانی می‌کنیم و دو مقدار صحیح به آن ارسال می‌کنیم. دو عدد صحیح در داخل تابع با هم جمع شده و نتیجه آنها برگردانده می‌شود. مقدار برگشت داده شده از تابع به وسیله تابع

print() نمایش داده می‌شود (خط ۱۳). در برنامه زیر یک تابع تعریف شده است که دارای دو پارامتر از دو نوع داده‌ای مختلف است:

```
1  fun showMessageAndNumber(message: String, number: Int)
2  {
3      println(message)
4      println("Number = $number")
5  }
6
7  fun main(args: Array<String>)
8  {
9      showMessageAndNumber("Hello World!", 100)
10 }
```

```
Hello World!
Number = 100
```

در مثال بالا یک تابع تعریف شده است که اولین پارامتر آن مقداری از نوع رشته و دومین پارامتر آن مقداری از نوع Int دریافت می‌کند. تابع به سادگی دو مقداری که به آن ارسال شده است را نشان می‌دهد. در خط ۹ تابع را اول با یک رشته و سپس یک عدد خاص فراخوانی می‌کنیم. حال اگر تابع به صورت زیر فراخوانی می‌شد:

```
showMessageAndNumber(100, "Welcome to Gimme Kotlin!");
```

در برنامه خطا به وجود می‌آمد چون عدد ۱۰۰ به پارامتری از نوع رشته و رشته Hello World! به پارامتری از نوع اعداد صحیح ارسال می‌شد. این نشان می‌دهد که ترتیب ارسال آرگومانها به پارامترها هنگام فراخوانی تابع مهم است.

به مثال ۱ توجه کنید. در آن مثال دو عدد از نوع Int به پارامترها ارسال کردیم که ترتیب ارسال آنها چون هر دو پارامتر از یک نوع بودند مهم نبود. ولی اگر پارامترهای تابع دارای اهداف خاصی باشند ترتیب ارسال آرگومانها مهم است.

```
fun showPersonStats(age: Int, height:Int)
{
    println("Age = $age")
```

```
    println("Height = $height")
}

//Using the proper order of arguments
showPersonStats(20, 160)

//Acceptable, but produces odd results
showPersonStats(160, 20)
```

در مثال بالا، نشان داده شده است که حتی اگر تابع دو آرگومان با یک نوع داده‌ای قبول کند باز هم بهتر است ترتیب بر اساس تعریف پارامترها رعایت شود. به عنوان مثال، در اولین فراخوانی تابع بالا اشکالی به چشم نمی‌آید، چون سن شخص ۲۰ و قد او ۱۶۰ سانتی متر است. اگر آرگومانها را به ترتیب ارسال نکنیم سن شخص ۱۶۰ و قد او ۲۰ سانتی متر می‌شود که به واقعیت نزدیک نیست. دانستن مبانی مقادیر برگشتی و ارسال آرگومانها باعث می‌شود که شما توابع کارآمدتری تعریف کنید. تکه کد زیر نشان می‌دهد که، شما حتی می‌توانید مقدار برگشتی از یک تابع را به عنوان آرگومان به تابع دیگر، ارسال کنید.

```
fun myFunction() : Int
{
    return 5
}

fun anotherFunction(number:Int)
{
    println(number)
}

fun main(args: Array<String>)
{
    // Codes skipped for demonstration
    anotherFunction(myFunction())
}
```

چون مقدار برگشتی تابع `myFunction()` عدد ۵ است و به عنوان آرگومان به تابع `anotherFunction()` ارسال می‌شود، خروجی کد بالا هم عدد ۵ است.

ارسال آرگومان به روش مقدار

ارسال آرگومانها به روش مقدار، بدان معناست که شما یک کپی از مقدار متغیر را ارسال می‌کنید، نه اصل متغیر یا ارجاع به آن را. در این حالت وقتی که آرگومان ارسال شده را در داخل تابع اصلاح می‌کنیم، مقدار اصلی آرگومان در خارج از تابع تغییر نمی‌کند. اجازه دهید که ارسال با مقدار آرگومان را با یک مثال توضیح دهیم:

```
1   fun modifyNumberVal(number: Int)
2   {
3       var number = number
4       number += 10
5       println("Value of number inside method is $number")
6   }
7
8   fun main(args: Array<String>)
9   {
10      val num = 5
11      println("num = $num\n")
12
13      println("Passing num by value to method modifyNumberVal() ...")
14      modifyNumberVal(num)
15      println("Value of num after exiting the method is $num")
16  }
```

```
num = 5

Passing num by value to method modifyNumberVal() ...
Value of number inside method is 15.
Value of num after exiting the method is 5.
```

در برنامه بالا، تابعی تعریف شده است که کار آن اضافه کردن عدد ۱۰ به مقداری است که به آنها ارسال می‌شود (خطوط ۷-۱). این تابع دارای یک پارامتر است که نیاز به یک مقدار آرگومان (از نوع Int) دارد. وقتی که تابع را صدا می‌زنیم و آرگومانی به آن اختصاص می‌دهیم (خط ۱۴)، کپی آرگومان به پارامتر تابع ارسال می‌شود. بنابراین مقدار اصلی متغیر خارج از تابع، هیچ ارتباطی به پارامتر تابع ندارد. سپس مقدار ۱۰ را به متغیر پارامتر (number) اضافه کرده و نتیجه را چاپ می‌کنیم. برای اثبات اینکه متغیر num هیچ تغییری نکرده است مقدار آن را یکبار قبل از ارسال به تابع (خط ۱۱) و بار دیگر بعد از ارسال به تابع (خط ۱۵) چاپ کرده و مشاهده می‌کنیم که تغییری نکرده است.

ارسال آرایه به عنوان آرگومان

می‌توان آرایه‌ها را به عنوان آرگومان به تابع ارسال کرد. ابتدا شما باید پارامترهای تابع را طوری تعریف کنید که آرایه دریافت کنند. به مثال زیر توجه کنید.

```
fun TestArray(numbers: IntArray)
{
    for (number in numbers)
    {
        println(number)
    }
}

fun main(args: Array<String>)
{
    val array = intArrayOf(1, 2, 3, 4, 5)
    TestArray(array)
}
```

```
1
2
3
4
5
```

مشاهده کردید که به سادگی می‌توان نوع داده‌ای پارامتر را به صورت آرایه تعریف کرد. این کار باعث می‌شود که پارامتر، آرایه دریافت کند. وقتی تابع در خط ۱۲ فراخوانی می‌شود، آرایه را فقط با استفاده از نام آن و بدون استفاده از اندیس ارسال می‌کنیم. پس آرایه‌ها به روش ارجاع به توابع ارسال می‌شوند. در خطوط ۶-۳ از حلقه for برای دسترسی به اجزای اصلی آرایه که به عنوان آرگومان به تابع ارسال کرده‌ایم، استفاده می‌کنیم. در زیر نحوه ارسال یک آرایه به روش ارجاع نشان داده شده است:

```
fun IncrementElements(numbers: IntArray)
{
    for (i in numbers.indices)
    {
        numbers[i]++
    }
}

fun main(args: Array<String>)
{
    val array = intArrayOf(1, 2, 3, 4, 5)

    IncrementElements(array)
```

```
14
15      for (num in array)
16      {
17          println(num)
18      }
19  }
```

```
2
3
4
5
6
```

برنامه بالا، یک تابع را نشان می‌دهد که یک آرایه را دریافت می‌کند و به هر یک از عناصر آن یک واحد اضافه می‌کند. در داخل تابع ما مقادیر هر یک از اجزای آرایه را افزایش داده‌ایم. سپس از تابع خارج شده و نتیجه را نشان می‌دهیم. مشاهده می‌کنید که هر یک از مقادیر اصلی تابع هم اصلاح شده‌اند. راه دیگر برای ارسال آرایه به تابع، مقدار دهی مستقیم به تابع فراخوانی شده است. به عنوان مثال:

```
IncrementElements(intArrayOf(1, 2, 3, 4, 5))
```

در این روش، ما آرایه‌ای از مقادیر را به پارامتر ارسال می‌کنیم. از آنجاییکه در این روش آرایه‌ای تعریف نکرده‌ایم، نمی‌توانیم در تابع main نتیجه را چاپ کنیم. اگر از چندین پارامتر در تابع استفاده می‌کنید، همیشه برای هر یک از پارامترهایی که آرایه قبول می‌کنند، باید نوع آنها را به صورت آرایه تعریف کنید. به عنوان مثال:

```
fun myFunction(param1: IntArray, param2: Int)
{
    //code here
}
```

به پارامترهای تابع بالا توجه کنید، پارامتر اول (param1) آرگومانی از جنس آرایه قبول می‌کند ولی پارامتر دوم (param2) یک عدد صحیح. حال اگر پارامتر دوم (param2) هم آرایه قبول می‌کرد باید نوع آن را به صورت آرایه تعریف می‌کردیم:

```
fun myFunction(param1: IntArray, param2: IntArray)
{
```

```
    //code here
}
```

محدوده متغیر

متغیرها در کاتلین، دارای محدوده هستند. محدوده یا scope یک متغیر، به شما می‌گوید که در کجای برنامه می‌توان از متغیر استفاده کرد و یا متغیر قابل دسترسی است. به عنوان مثال، متغیری که در داخل یک تابع تعریف می‌شود، فقط در داخل بدنه تابع قابل دسترسی است. می‌توان دو متغیر با نام یکسان در دو تابع مختلف تعریف کرد. برنامه زیر این ادعا را اثبات می‌کند:

```
1  fun demonstrateScope()
2  {
3      val number = 5
4
5      println("number inside method demonstrateScope() = $number")
6  }
7
8  fun main(args: Array<String>)
9  {
10     val number = 10
11
12     demonstrateScope()
13
14     println("number inside the main method() = $number")
15 }
```

```
number inside method DemonstrateScope() = 5
number inside the Main method() = 10
```

مشاهده می‌کنید که، حتی اگر ما دو متغیر با نام یکسان تعریف کنیم که، دارای محدوده‌های متفاوتی باشند، می‌توان به هر کدام از آنها مقادیر مختلفی اختصاص داد. متغیر تعریف شده در داخل تابع main() در خط ۱۰ هیچ ارتباطی به متغیر داخل تابع demonstrateScope() در خط ۳ ندارد. وقتی به مبحث کلاسها رسیدیم در این باره بیشتر توضیح خواهیم داد.

آرگومان‌های متغیر (VarArgs)

آرگومان‌های متغیر (VarArgs)، در JDK 5 اضافه شده است و به شما اجازه می‌دهد، تعداد آرگومان‌های دلخواه (متغیر) و البته همنوع را به تابع ارسال کنید. برای ایجاد تابعی که به تعداد دلخواه پارامتر دریافت کند، از کلمه vararg به صورت زیر استفاده می‌شود:

```
fun functionName(vararg argument: dataType)
{
    // function body
}
```

به مثال زیر توجه کنید:

```
1   fun calculateSum(vararg numbers: Int): Int
2    {
3       var total = 0
4
5       for (number in numbers)
6       {
7           total += number
8       }
9
10      return total
11  }
12
13  fun main(args: Array<String>)
14  {
15      println("1 + 2 + 3 = " + calculateSum(1, 2, 3))
16      println("1 + 2 + 3 + 4 = " + calculateSum(1, 2, 3, 4))
17      println("1 + 2 + 3 + 4 + 5 = " + calculateSum(1, 2, 3, 4, 5))
18  }
```

```
1 + 2 + 3 = 6
1 + 2 + 3 + 4 = 10
15 = 5 + 4 + 3 + 2 + 1
```

همانطور که در مثال بالا مشاهده می‌کنید، یک تابع به نام calculateSum() در خط ۳ تعریف شده است. برای اینکه این تابع تعداد دلخواه پارامتر دریافت کند، از کلمه vararg استفاده شده است. در اصل کلمه numbers یک آرایه است، که وقتی ما آرگومان‌ها را به تابع ارسال می‌کنیم، در این آرایه ذخیره می‌شوند. حال تابع را سه بار با تعداد مختلف آرگومانها فراخوانی می‌کنیم و سپس با استفاده از حلقه for این آرگومانها را جمع و به تابع فراخوان برگشت می‌دهیم. وقتی از چندین پارامتر در یک تابع استفاده می‌کنید، فقط یکی از آنها باید

دارای کلمه vararg بوده و همچنین از لحاظ مکانی باید آخرین پارامتر باشد. اگر این پارامتر، در آخر پارامترهای دیگر قرار نگیرد و یا از چندین پارامتر دارای کلمه vararg استفاده کنید، با خطا مواجه می‌شوید. به مثالهای اشتباه و درست زیر توجه کنید:

```
fun someFunction(vararg x: Int, vararg y: Int) //ERROR

fun someFunction(vararg x: Int, y: Int, z: Int) //ERROR

fun someFunction(x: Int, y: Int, vararg z: Int) //Correct
```

سربارگذاری توابع

سربارگذاری توابع یا Function Overloading، به شما اجازه می‌دهد که، دو یا چند تابع با نام یکسان تعریف کنید که، دارای امضا و تعداد پارامترهای مختلف هستند. برنامه از روی آرگومانهایی که شما به تابع ارسال می‌کنید به صورت خودکار تشخیص می‌دهد که، کدام تابع را فراخوانی کرده‌اید، یا کدام تابع مد نظر شماست. امضای یک تابع نشان دهنده ترتیب و نوع پارامترهای آن است. به مثال زیر توجه کنید:

```
fun myFunction(x: Int, y: Double z: String)
```

که امضای تابع بالا

```
myFunction(Int, Double , String)
```

به این نکته توجه کنید که نوع برگشتی و نام پارامترها شامل امضای تابع نمی‌شوند. در مثال زیر نمونه‌ای از سربارگذاری توابع آمده است.

```
1  fun showMessage(number: Double)
2  {
3      println("Double version of the method was called.")
4  }
5
6  fun showMessage(number: Int)
```

```
7   {
8       println("Integer version of the method was called.")
9   }
10
11  fun main(args: Array<String>)
12  {
13      showMessage(9.99)
14      showMessage(9)
15  }
```

```
Double version of the method was called.
Integer version of the method was called.
```

در برنامه بالا دو تابع با نام مشابه تعریف شده‌اند. اگر سربارگذاری تابع توسط کاتلین پشتیبانی نمی‌شد برنامه زمان زیادی برای انتخاب یک تابع از بین توابعی که فراخوانی می‌شوند لازم داشت. رازی در نوع پارامترهای تابع نهفته است. کامپایلر بین دو یا چند تابع در صورتی فرق می‌گذارد که، پارامترهای متفاوتی داشته باشند. وقتی یک تابع را فراخوانی می‌کنیم، تابع نوع آرگومانها را تشخیص می‌دهد.

در فراخوانی اول (خط ۱۳) ما یک مقدار double را به تابع showMessage() ارسال کرده‌ایم در نتیجه تابع showMessage() (خطوط ۴-۱) که دارای پارامتری از نوع Double اجرا می‌شود. در بار دوم که تابع فراخوانی می‌شود (خط ۱۴) ما یک مقدار Intرا به تابع showMessage() ارسال می‌کنیم تابع showMessage() (خطوط ۹-۶) که دارای پارامتری از نوع Int است اجرا می‌شود. معنای اصلی سربارگذاری تابع همین است که توضیح داده شد.

هدف اصلی از سربارگذاری توابع این است که، بتوان چندین تابع که وظیفه یکسانی انجام می‌دهند، را تعریف کرد. تعداد زیادی از توابع در جاوا، سربارگذاری می‌شوند، مانند تابع println() از کلاس out. قبلاً مشاهده کردید که این تابع می‌تواند یک آرگومان از نوع رشته دریافت کند و آن را نمایش دهد، و در حالت دیگر می‌تواند دو یا چند آرگومان قبول کند.

بازگشت (Recursion)

بازگشت یا Recursion، فرایندی است که در آن تابع مدام خود را فراخوانی می‌کند، تا زمانی که به یک مقدار مورد نظر برسد. بازگشت یک مبحث پیچیده در برنامه نویسی است و تسط به آن کار راحتی نیست. به این نکته هم توجه کنید که بازگشت باید در یک نقطه متوقف شود وگرنه برای بی نهایت بار، تابع، خود را فراخوانی می‌کند. در این درس یک مثال ساده از بازگشت را برای شما توضیح می‌دهیم. فاکتوریل یک عدد صحیح مثبت (n!) شامل حاصل ضرب همه اعداد مثبت صحیح کوچکتر یا مساوی آن می‌باشد. به فاکتوریل عدد ۵ توجه کنید.

```
5! = 5 * 4 * 3 * 2 * 1 = 120
```

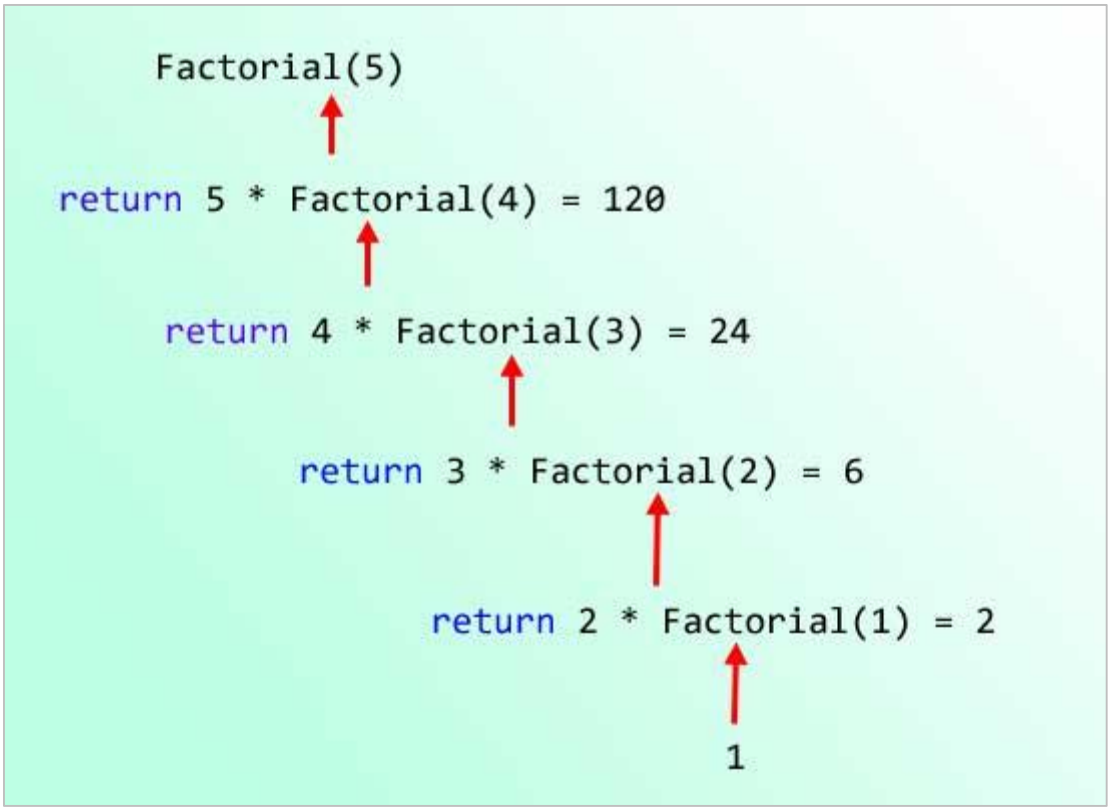
بنابراین برای ساخت یک تابع بازگشتی باید به فکر توقف آن هم باشیم. بر اساس توضیح بازگشت، فاکتوریل فقط برای اعداد مثبت صحیح است. کوچک‌ترین عدد صحیح مثبت ۱ است. در نتیجه از این مقدار برای متوقف کردن بازگشت استفاده می‌کنیم.

```
1  fun Factorial(number: Int): Long
2  {
3      return if (number == 1) 1 else number * Factorial(number - 1)
4  }
5
6  fun main(args: Array<String>)
7  {
8      println(Factorial(5))
9  }
```

```
120
```

تابع مقدار بزرگی را بر می‌گرداند چون محاسبه فاکتوریل می‌تواند خیلی بزرگ باشد. تابع یک آرگومان که یک عدد است و می‌تواند در محاسبه مورد استفاده قرار گیرد را می‌پذیرد. در داخل تابع یک دستور if می‌نویسیم و در خط ۳ می گوییم که اگر آرگومان ارسال شده برابر ۱ باشد سپس مقدار ۱ را برگردان در غیر اینصورت به خط بعد برو. این شرط باعث توقف تکرارها نیز می‌شود.

در خط ۳ مقدار جاری متغیر number در عددی یک واحد کمتر از خودش (number - 1) ضرب می‌شود. در این خط تابع Factorial خود را فراخوانی می‌کند و آرگومان آن در این خط همان number - 1 است. مثلاً اگر مقدار جاری number عدد ۱۰ باشد، یعنی اگر ما بخواهیم فاکتوریل عدد ۱۰ را به دست بیاوریم آرگومان تابع Factorialدر اولین ضرب ۹ خواهد بود. فرایند ضرب تا زمانی ادامه می‌یابد که آرگومان ارسال شده با عدد ۱ برابر نشود. شکل زیر فاکتوریل عدد ۵ را نشان می‌دهد.



کد بالا را به وسیله یک حلقه for نیز می‌توان نوشت.

```
factorial = 1;

for ( int counter = number; counter >= 1; counter-- )
   factorial *= counter;
```