Lesson 1: Introduction to Kotlin

Kotlin History

In July 2011, JetBrains unveiled Project Kotlin - a new language for the JVM - which has been under development for a year. One of the stated goals of Kotlin was to compile as quickly as Java. In February 2012, JetBrains open-sourced the project under the Apache 2 license. Kotlin is 100% interoperable with Java™ and Android™.

Kotlin Advantages

Kotlin is a great fit for Android applications development; it brings all the advantages of a modern programming language to the Android platform without any restrictions. Following are some interesting features of the Kotlin language:

1) **Performance**: A Kotlin program runs faster than Java code although it has similar byte code structure; Less code combined with greater readability. Kotlin takes less time writing your code and understanding the code of others. With Kotlin, you can perform more tasks and write less code.

2) **Easy to learn and intuitive**: In its most parts, Kotlin is similar to Java but the differences are in the basic concepts which can be learnt in no time. Kotlin is very easy to learn, especially for Java developers.
 Kotlin has mature language and environment. Since its creation in 2011, Kotlin has developed continuously, not only in terms of language but as a whole ecosystem with robust tooling. Now, it's seamlessly integrated in Android Studio and is actively used by many companies for developing Android applications.

3) **Free Integration with Android studio**: The Kotlin tooling is fully supported in Android Studio and compatible with the Android build system. It just requires simple configuration which we are going to learn about later on in the upcoming lessons. An interesting feature of Kotlin plug-in at Android Studio is the ability to convert Java code to Kotlin code. Also, Kotlin is supported in Android Jetpack and other libraries. KTX extensions add Kotlin language features, such as coroutines, extension functions, lambdas, and named parameters, to existing Android libraries.

4) **Interoperability**: Kotlin is 100% interoperable with Java and it allows importing all existing Android libraries in Kotlin application. It is even possible to create a project in both Java and Kotlin. You can use Kotlin along with the Java programming language in your applications without a need to migrate all your code to Kotlin.

5) **Null Safety**: In Java, we always need to check for null values before using them if we don't want to get Null Pointer Exception. Whereas Kotlin, like many other modern languages, is null safe because it uses the safe call operator (?). You will learn more about Null safety and see more examples in the following lessons.

6) **Support for Multiplatform Development:** You can use Kotlin for developing not only Android but also iOS, backend, and web applications. Enjoy the benefits of sharing the common code among the platforms.
With Kotlin, if you want to make an existing project suitable for multiple platforms, you don't have

to go back to the drawing board. You can use the code you've already written and simply modify it to be compatible with iOS. You can even migrate your code in stages. So no matter how large your project is, your existing code will not prevent you from integrating KMM (Kotlin Multiplatform Mobile).

7) **Code Safety**: Less code and better readability lead to less errors. The Kotlin compiler detects these remaining errors, making the code safe.

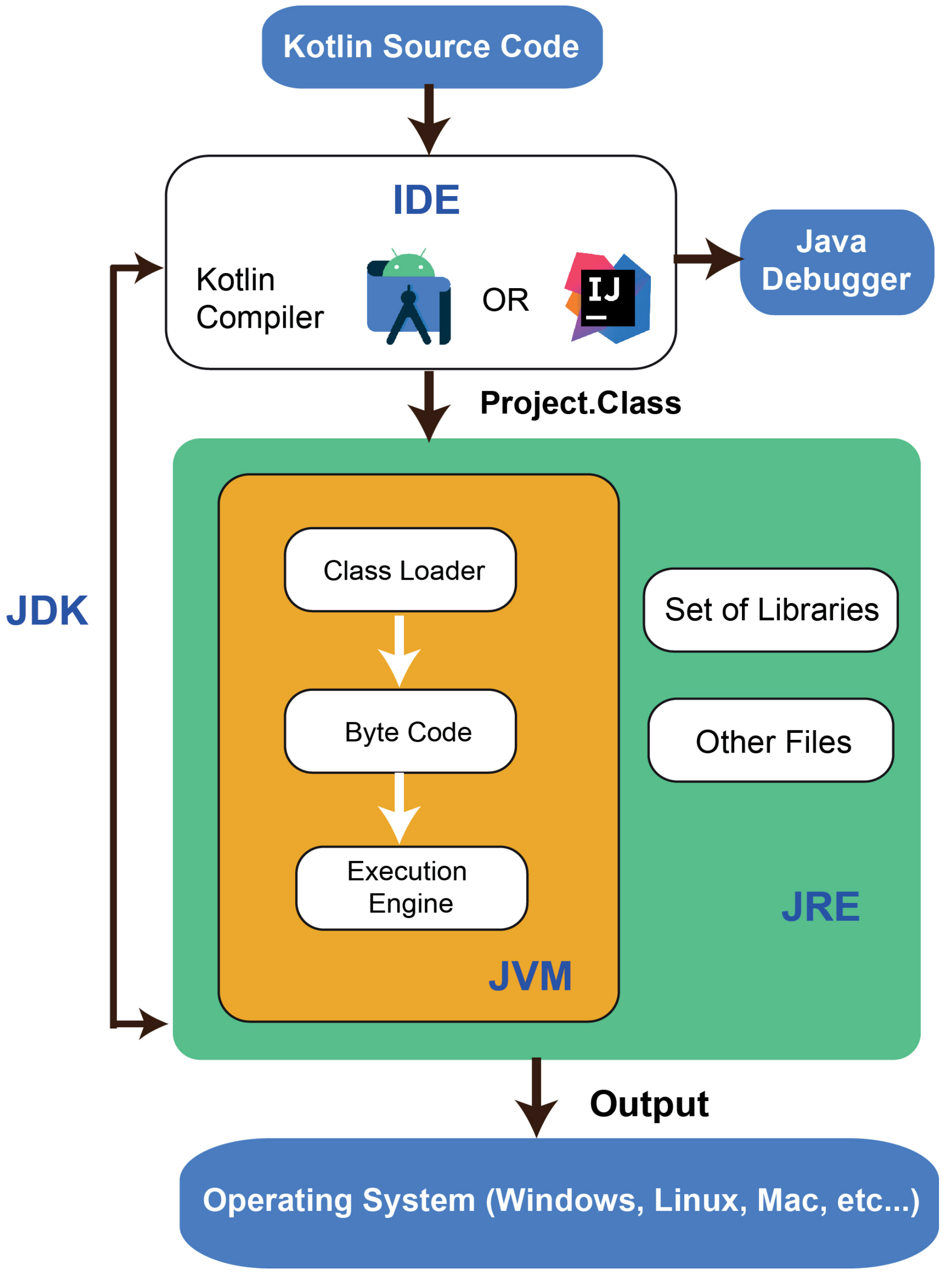
8) **Big Community:** Kotlin has great support and many contributions from the community, which is growing all over the world. According to Google, over 60% of the top 1000 apps on the Play Store use Kotlin. Also, many startups and Fortune 500 companies have already developed Android applications using Kotlin – see the list at the Google website for Kotlin developers.

Considering all these advantages of Kotlin, Google announced Kotlin as an official language for Android during Google I/O 2017. Android mobile development has been Kotlin-first since Google I/O in 2019.

How Kotlin Programs Work?

A Kotlin program or application works like C#, Java or any other programming language that needs a software called IDE (Integrated Development Environment) to create and read its content. There are a lot of IDEs that can be used to create a Kotlin application such as IntelliJ IDEA and Android Studio, but in this book we will use IntelliJ IDEA for this purpose, whereas we will use Android Studio to create the Android apps. This is to give our students an experience in using different types of IDEs.

The next workflow diagram displays how Kotlin programs work. The Kotlin files (class) source codes which were written in IntelliJ (IDE) will be considered as *Project.class* files that have the ability to run and get results in JRE (Java Runtime Environment) and where this class file will be moved. JRE in return has a Class Loader which is responsible for receiving the class file and executing it via the Execution Engine of the JRE. This means that we need a software to execute the Kotlin files which are called JDK (Java Development Kit).
Here, we are using a Java development kit to run Kotlin code. JDK software includes the part responsible for running the Kotlin code and then sending the result to the operating system.



Kotlin Software Prerequisites

From the **previous** workflow diagram, you can notice that, to write a Kotlin code, you must install IntelliJ or Android Studio as IDE and Java JDK to execute the Kotlin code. These two software work together to enable you to run and execute a Kotlin code. In other words, these two software can be considered the prerequisite software which must be installed on computers to be able to work with Kotlin programming language.

We will go through all the setup process you need to do in order to use Kotlin.

IntelliJ IDEA

IntelliJ IDEA is a Java integrated development environment for developing computer software. It is developed by JetBrains.

IntelliJ IDEA is a free software used to develop Java, Kotlin, Dart, and other programming languages.

The first three lessons of this course discuss the fundamentals of Kotlin programming language. In these three lessons, you will learn how to write small separate Kotlin programs to become familiar with the Kotlin syntaxes - which you will use later in writing the codes to create Android mobile applications for Android devices.

We recommend using IntelliJ IDEA as compiler software to create and run pure Kotlin programs.

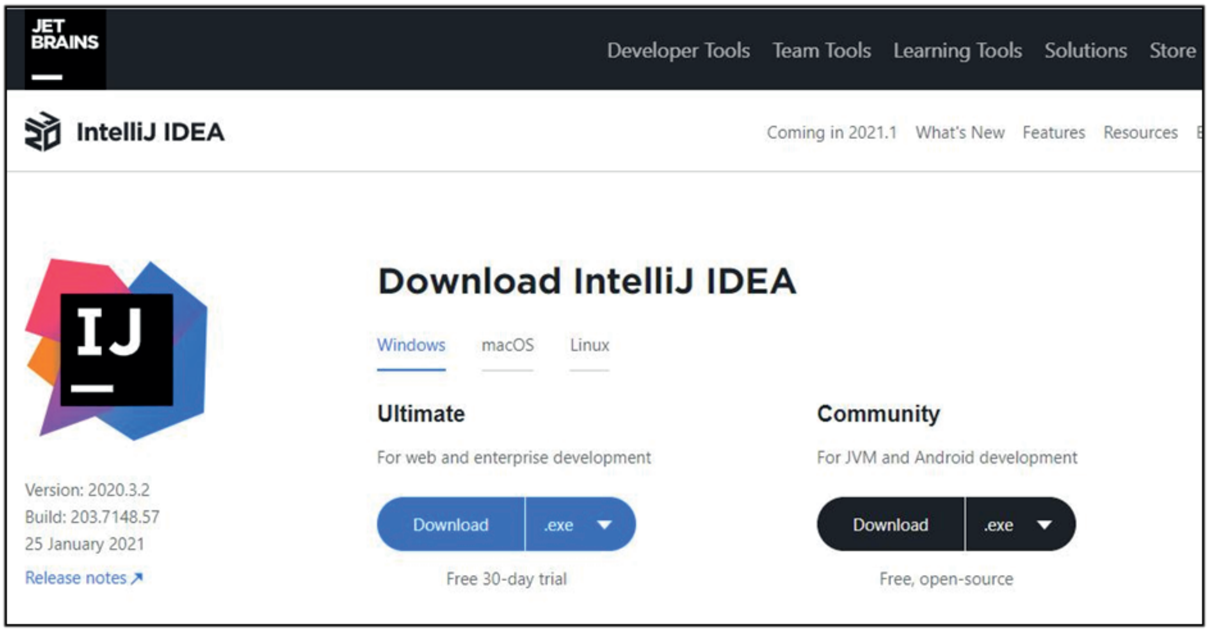
In this lesson, you will install IntelliJ IDEA step by step which will be used to create and run a Kotlin program.

Installing IntelliJ IDE

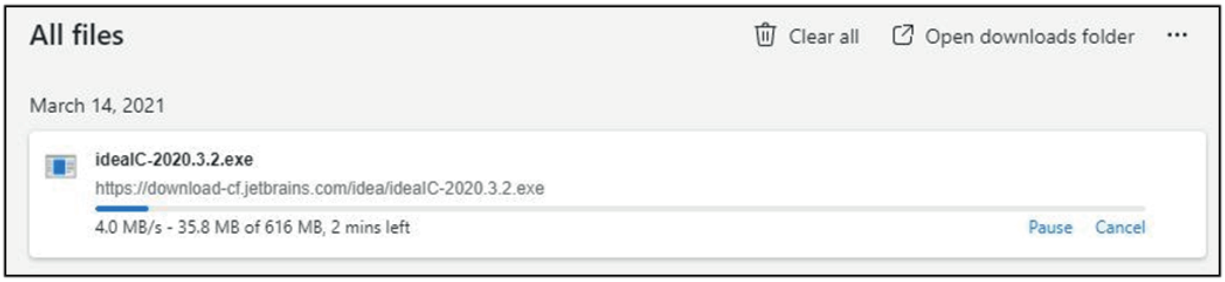
Follow the following steps to download **IntelliJ** IDEA:

1- Go to: **https://www.jetbrains.com/idea/download**

You will get the following download web page:



2- Under **Community**, click **DOWNLOAD** button. The download process will start to download IntelliJ IDEA as illustrated in the figure below. The size of this file is about 616 MB.



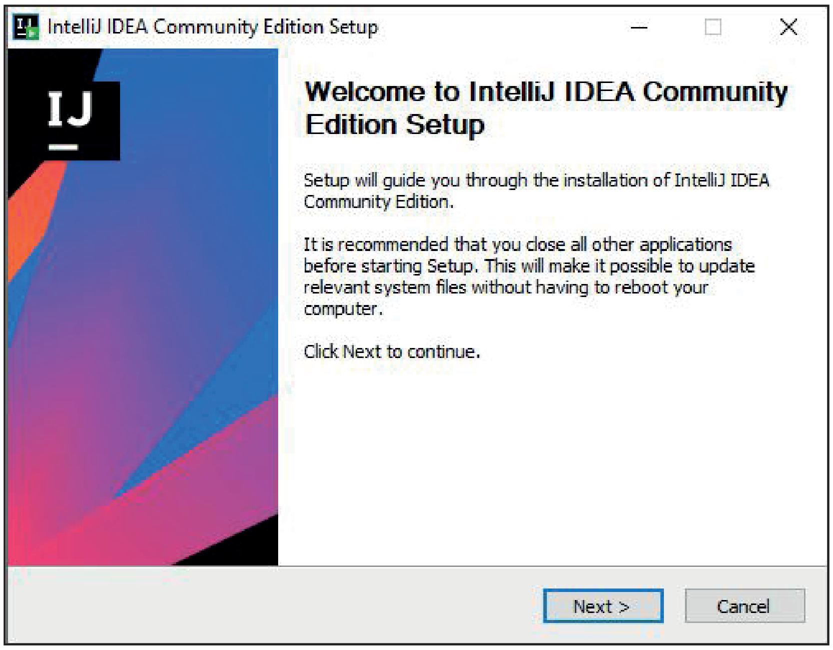
March 14, 2021

ideaIC-2020.3.2.exe

https://download-cf.jetbrains.com/idea/ideaIC-2020.3.2.exe

4.0 MB/s - 35.8 MB of 616 MB, 2 mins left

3- Click the download file to start the installation steps. First, in the figure below, click **Next**



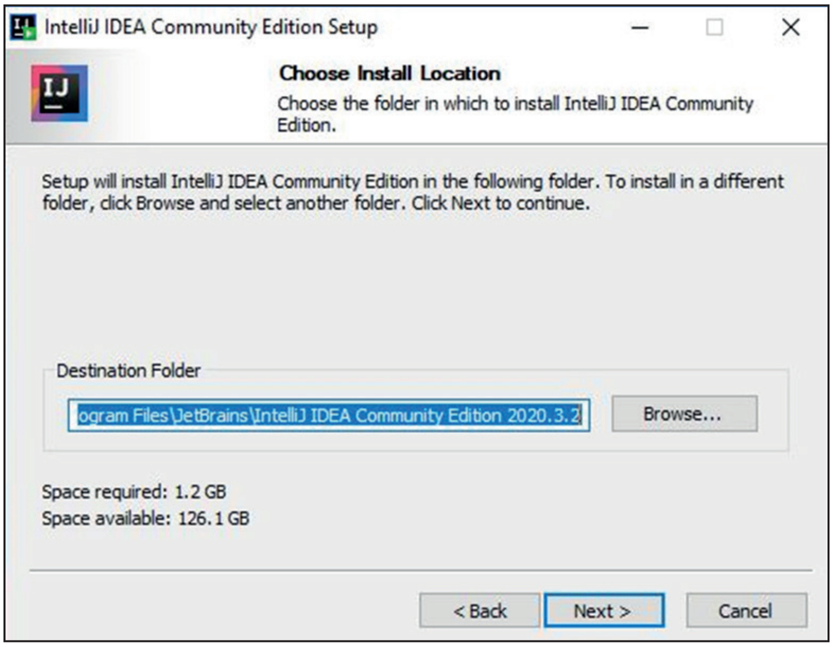
Welcome to IntelliJ IDEA Community Edition Setup

Setup will guide you through the installation of IntelliJ IDEA Community Edition.

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.

4- Keep the default destination folder, then click **Next.**



Choose Install Location

Choose the folder in which to install IntelliJ IDEA Community Edition.

Setup will install IntelliJ IDEA Community Edition in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.

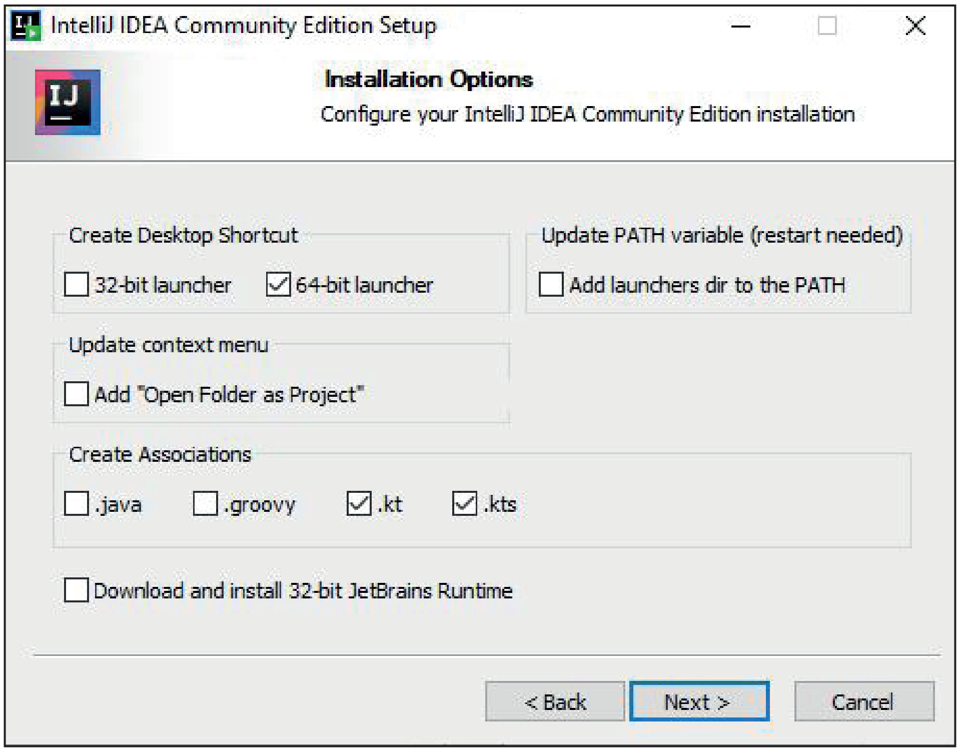
Destination Folder

ogram Files\JetBrains\IntelliJ IDEA Community Edition 2020.3.2

Space required: 1.2 GB

Space available: 126.1 GB

5- Our plan is to use IntelliJ to develop Kotlin; therefore, select **.kt** (for Kotlin files) and **.kts** (for Kotlin Script), as illustrated in the figure below. Click **Next.**



IntelliJ IDEA Community Edition Setup

Installation Options

Configure your IntelliJ IDEA Community Edition installation

Create Desktop Shortcut

☐ 32-bit launcher ☑ 64-bit launcher

Update PATH variable (restart needed)

☐ Add launchers dir to the PATH

Update context menu

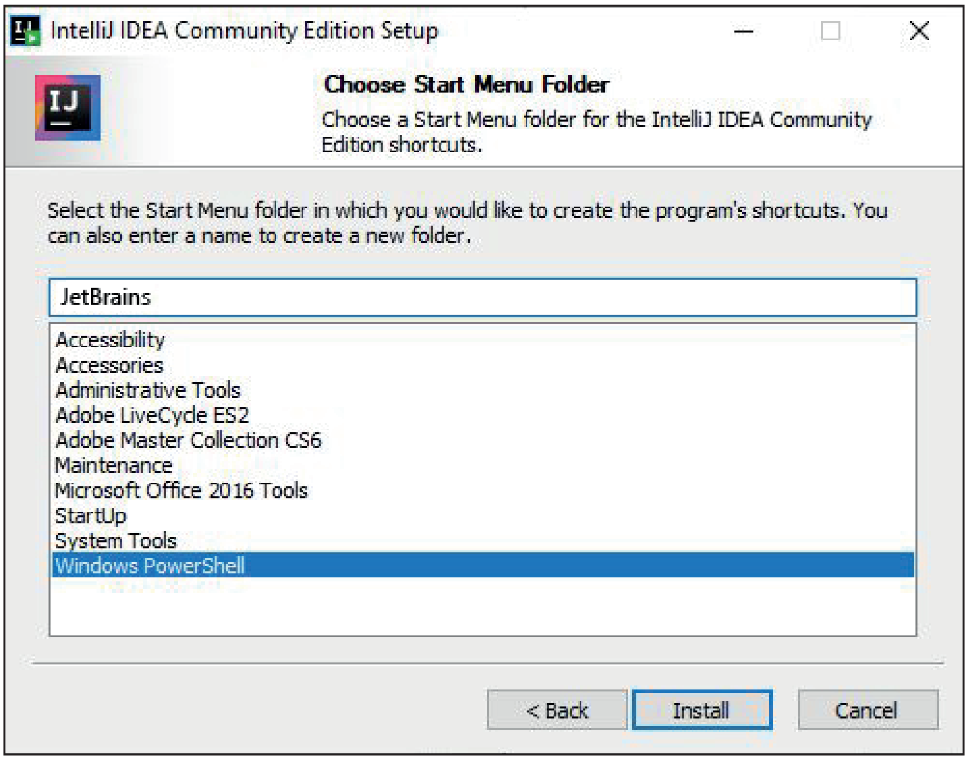
☐ Add "Open Folder as Project"

Create Associations

☐ .java ☐ .groovy ☑ .kt ☑ .kts

☐ Download and install 32-bit JetBrains Runtime

6- Keep the default start menu folder as illustrated in the figure below. Click **Install.**



IntelliJ IDEA Community Edition Setup

Choose Start Menu Folder

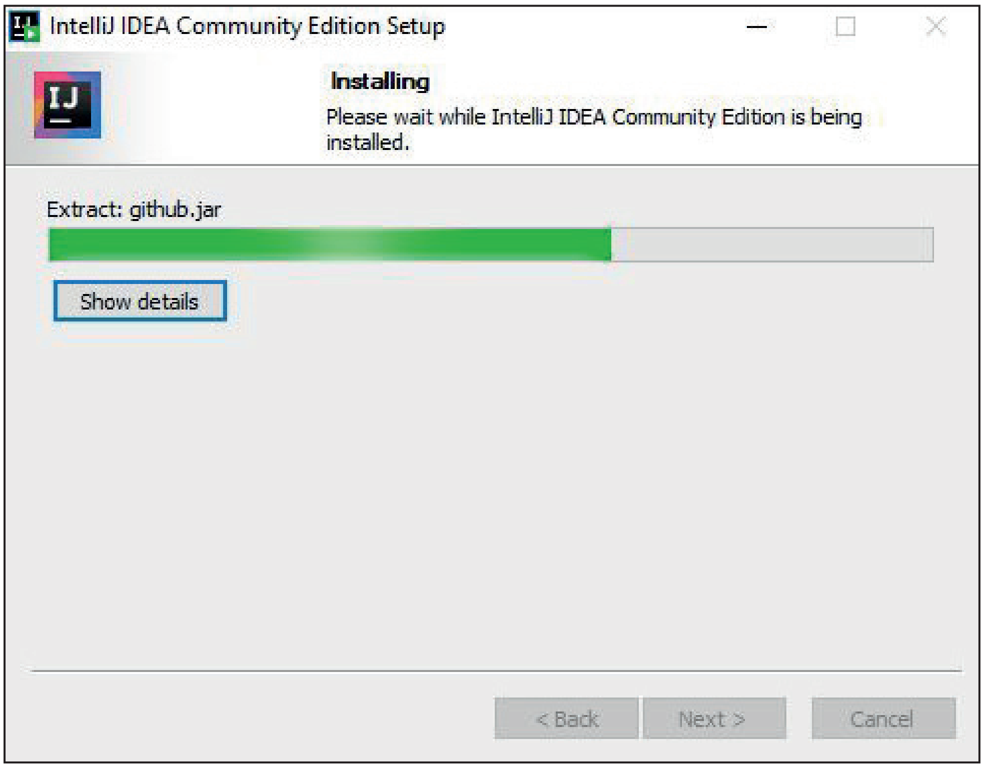
Choose a Start Menu folder for the IntelliJ IDEA Community Edition shortcuts.

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

Accessibility
Accessories
Administrative Tools
Adobe LiveCycle ES2
Adobe Master Collection CS6
Maintenance
Microsoft Office 2016 Tools
StartUp
System Tools
Windows PowerShell

The installation process will start as illustrated in the figure below:

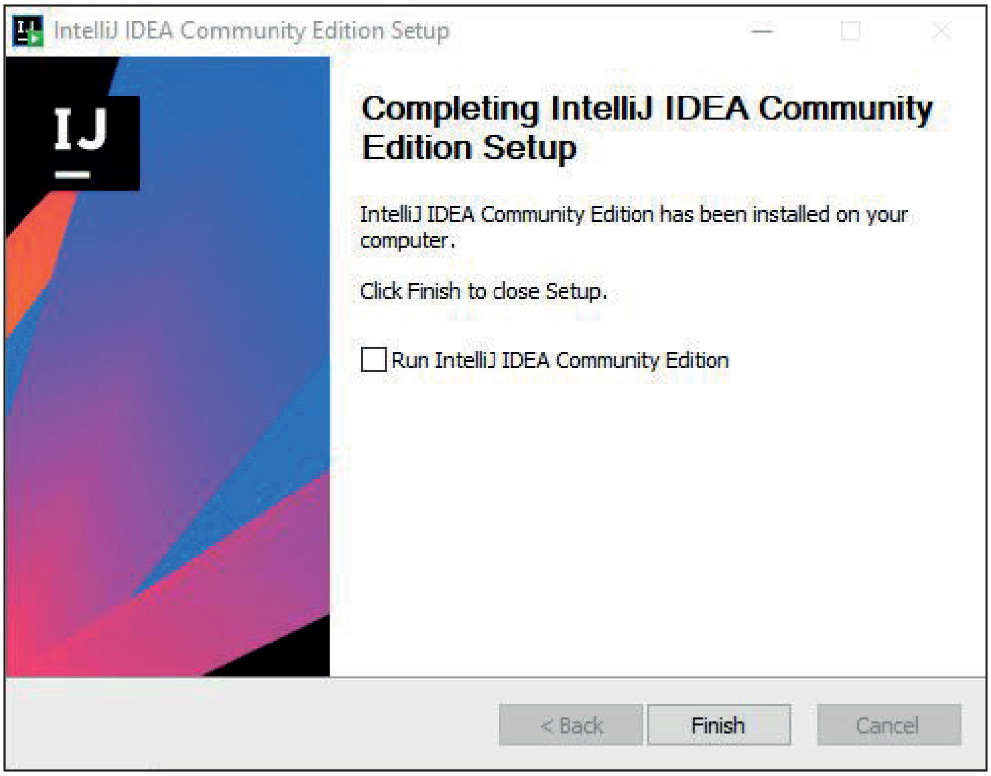

IntelliJ IDEA Community Edition Setup

Installing

Please wait while IntelliJ IDEA Community Edition is being installed.

Extract: github.jar

7- You will get the following figure. Click **Finish.**


IntelliJ IDEA Community Edition Setup

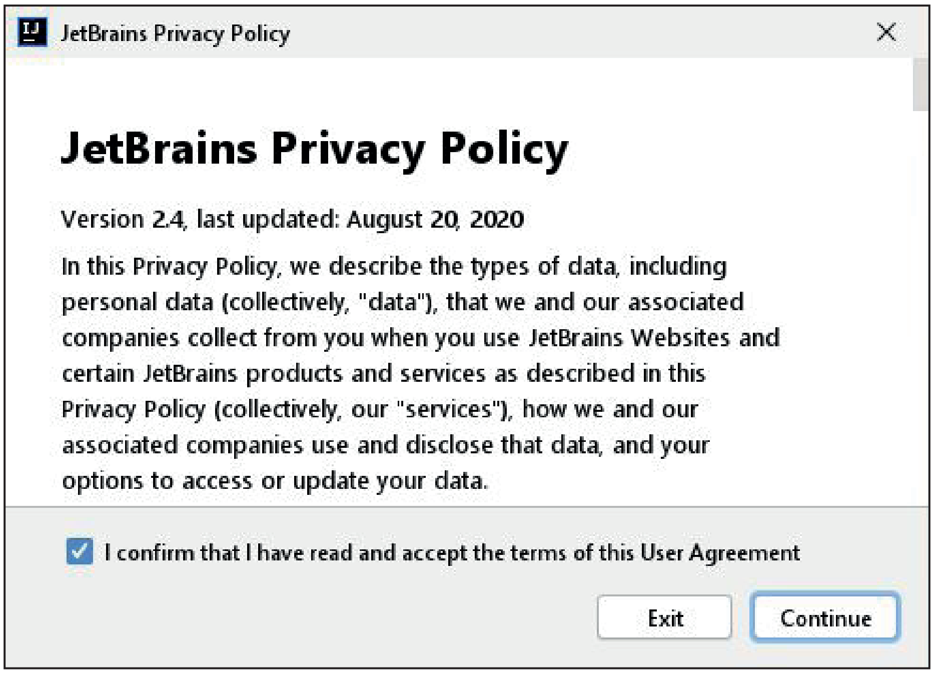
Completing IntelliJ IDEA Community Edition Setup

IntelliJ IDEA Community Edition has been installed on your computer.

Click Finish to close Setup.

☐ Run IntelliJ IDEA Community Edition

8- Double the IntelliJ shortcut at your desktop, and then accept the JetBrains Privacy Policy as illustrated in the figure below. Click **Continue**.



JetBrains Privacy Policy

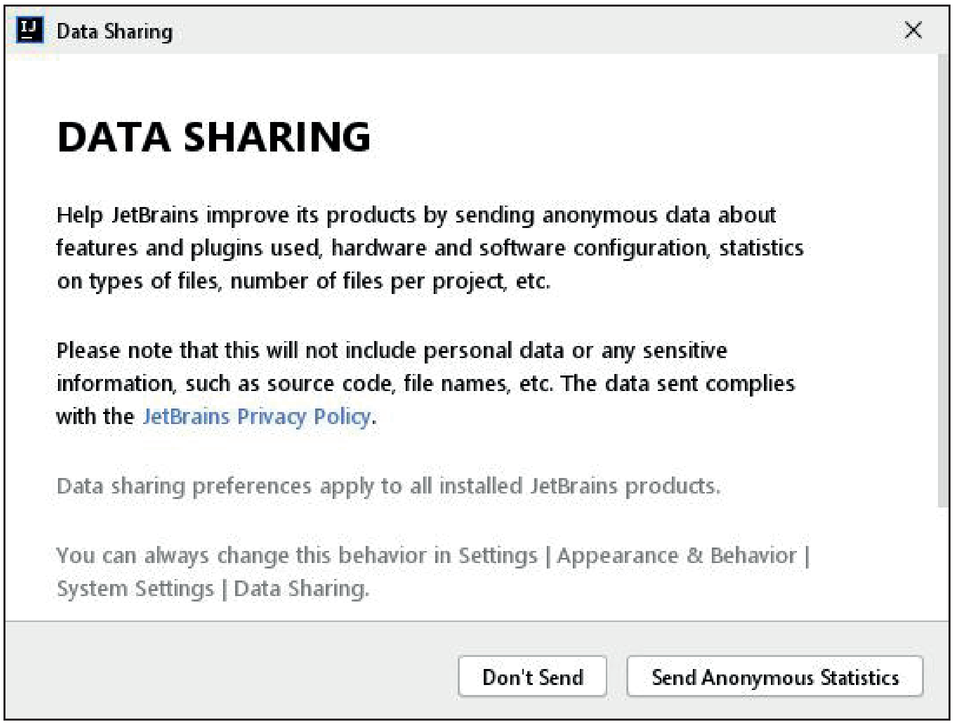
JetBrains Privacy Policy

Version 2.4, last updated: August 20, 2020

In this Privacy Policy, we describe the types of data, including personal data (collectively, "data"), that we and our associated companies collect from you when you use JetBrains Websites and certain JetBrains products and services as described in this Privacy Policy (collectively, our "services"), how we and our associated companies use and disclose that data, and your options to access or update your data.

☑ I confirm that I have read and accept the terms of this User Agreement

9- In this step and as illustrated in the figure below, click **Don’t Send.**



Data Sharing

DATA SHARING

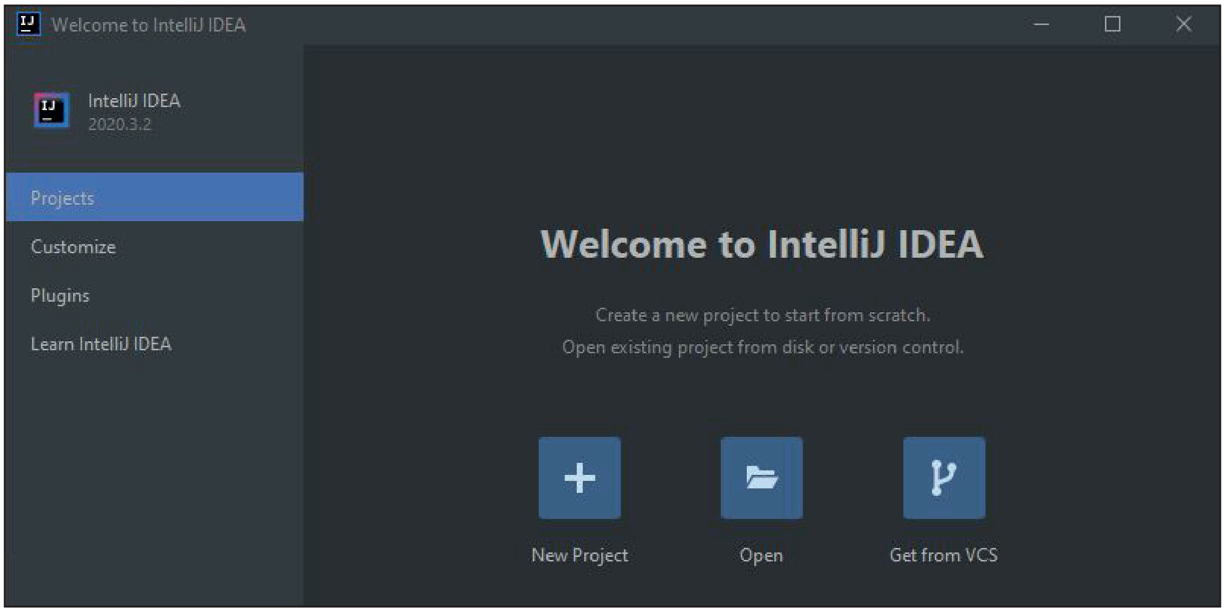
Help JetBrains improve its products by sending anonymous data about features and plugins used, hardware and software configuration, statistics on types of files, number of files per project, etc.

Please note that this will not include personal data or any sensitive information, such as source code, file names, etc. The data sent complies with the JetBrains Privacy Policy.

Data sharing preferences apply to all installed JetBrains products.

You can always change this behavior in Settings | Appearance & Behavior | System Settings | Data Sharing.

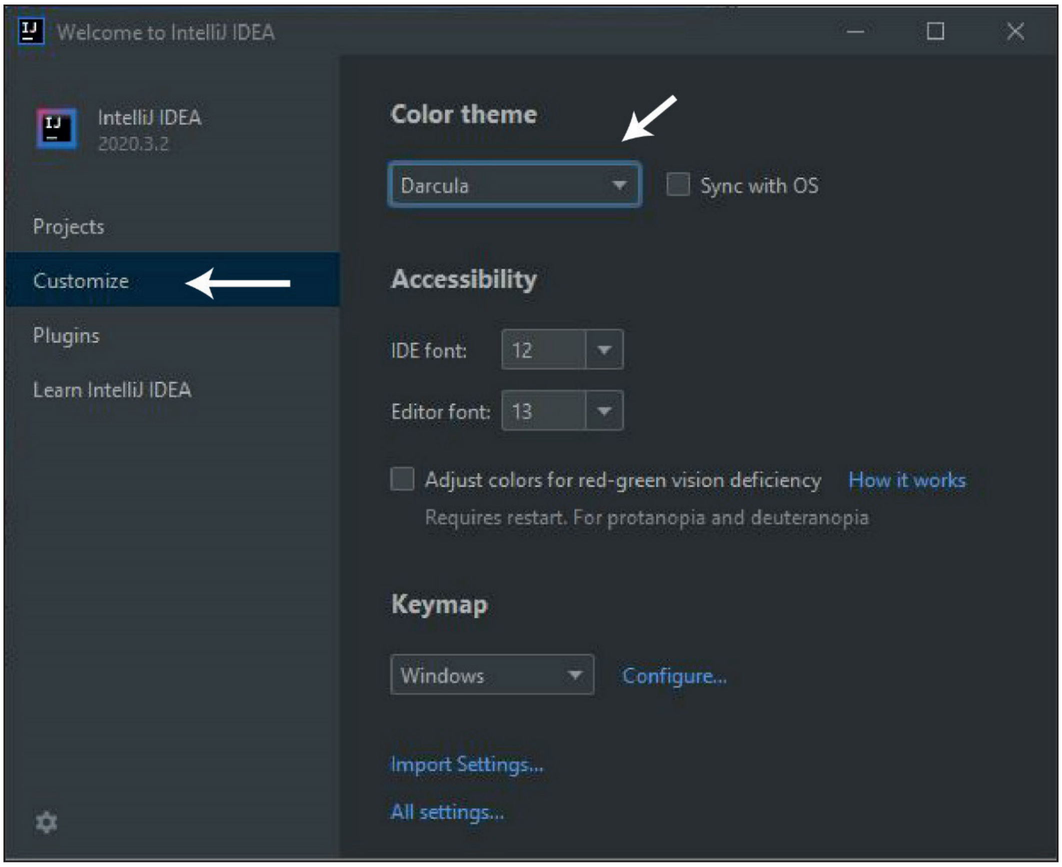
10- In this step, the IntelliJ IDEA will run successfully as illustrated in the figure below:



Welcome to IntelliJ IDEA

Create a new project to start from scratch.
Open existing project from disk or version control.

11- In this step, you may select your IntelliJ IDEA user interface theme. Almost most developers select **Darcula** theme because it is more comfortable for eye; however, we will select **Light** theme for printing considerations. Click **Customize** in the left panel and then select **IntelliJ Light** as illustrated in the figure below:



Color theme

Darcula ☐ Sync with OS

Accessibility

IDE font: 12

Editor font: 13

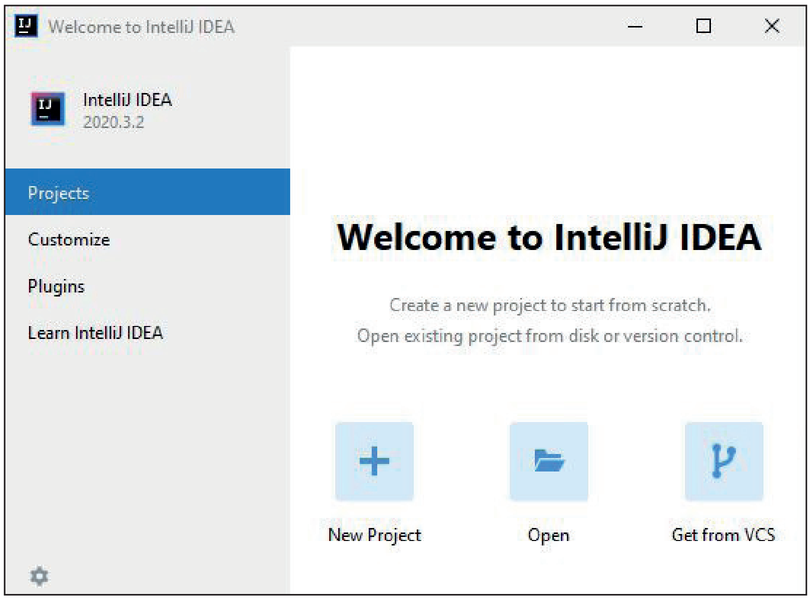
☐ Adjust colors for red-green vision deficiency

Requires restart. For protanopia and deuteranopia

Keymap

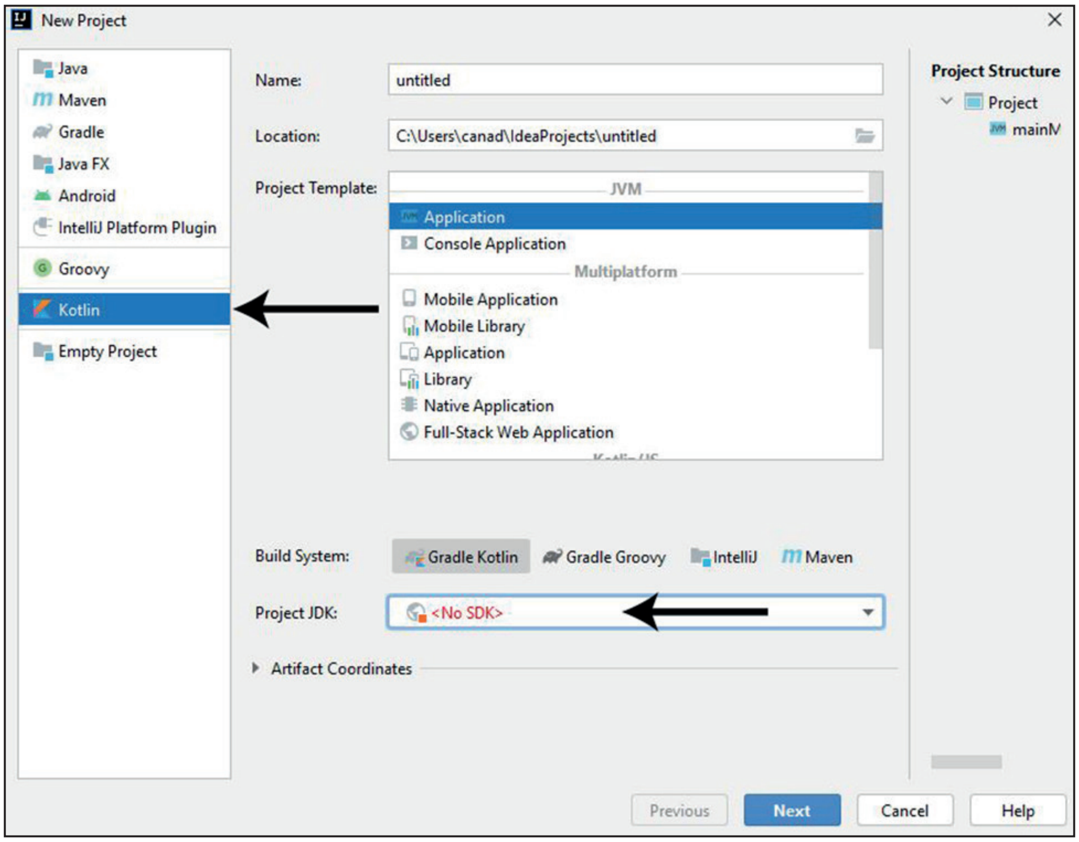
Windows

12- In this step and as illustrated in the figure below, select **Projects** from the left panel, and click **New Project**.



13- Until now, we installed only the Kotlin IDE (IntelliJ IDEA) which will be used to write the Kotlin code. The prerequisites to run any Kotlin code are JRE (Java Runtime Environment) and the JDK (Java Development Kit). You may download and install them as separate files from ORACLE web site or as we will do in the next step within creating our first Kotlin project.

Select **Kotlin**, and for the **Project JDK** drop down list, select the path of the Java JDK or download it.



Name: untitled

Location: C:\Users\canad\IdeaProjects\untitled

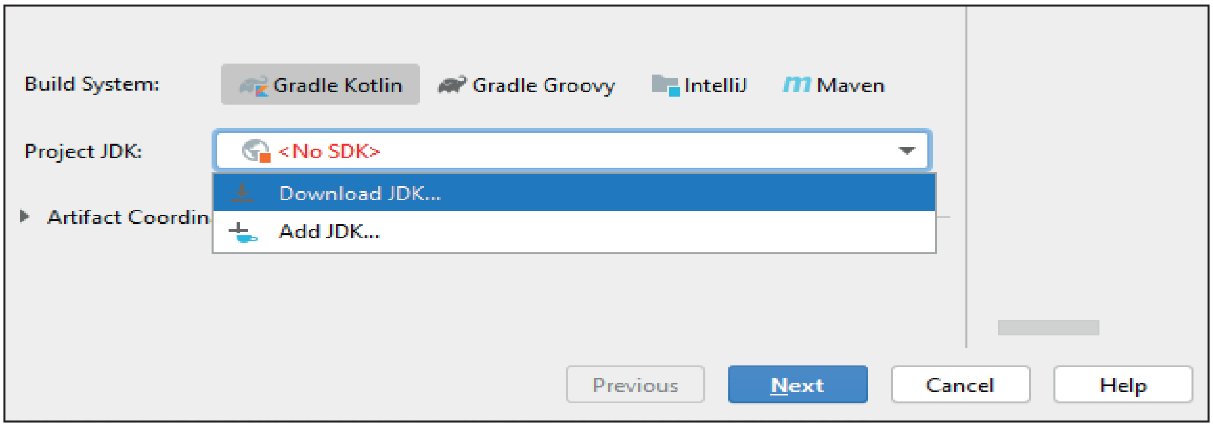
Project Template: Application

Build System: Gradle Kotlin

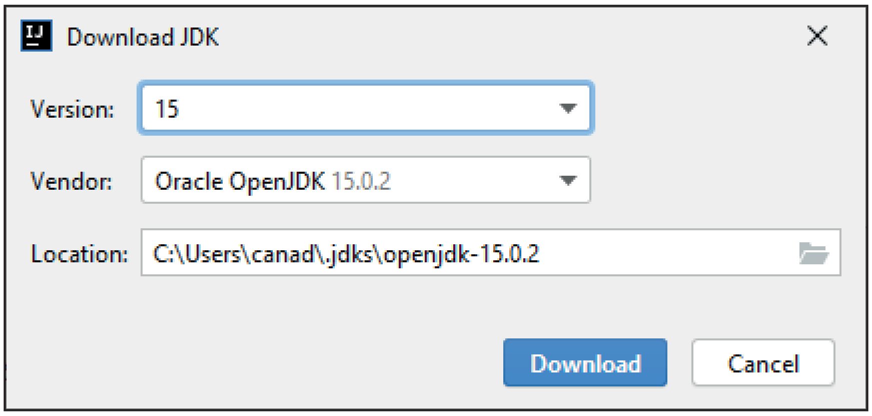
Project JDK: <No SDK>

Note: If the JDK is installed on your computer, but not defined in the IDE, select **Add JDK** and specify the path to the JDK home directory.

Now, because we don't have the Java JDK file before, we will select **Download JDK** choice as illustrated in the figure below:



You will get the pop-up dialog box below. This dialog box displays that your IntelliJ IDE has checked the Oracle web site automatically and found the latest version of Java JDK which is compatible with your operating system. As you see in the figure below, we got the last version of Java JDK at this time and it will be saved on **My Computer**. Click **Download.**

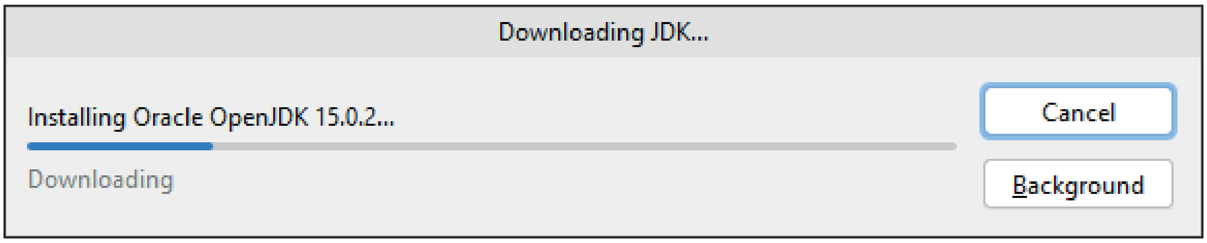

Download JDK

Version: 15

Vendor: Oracle OpenJDK 15.0.2

Location: C:\Users\canad\.jdks\openjdk-15.0.2

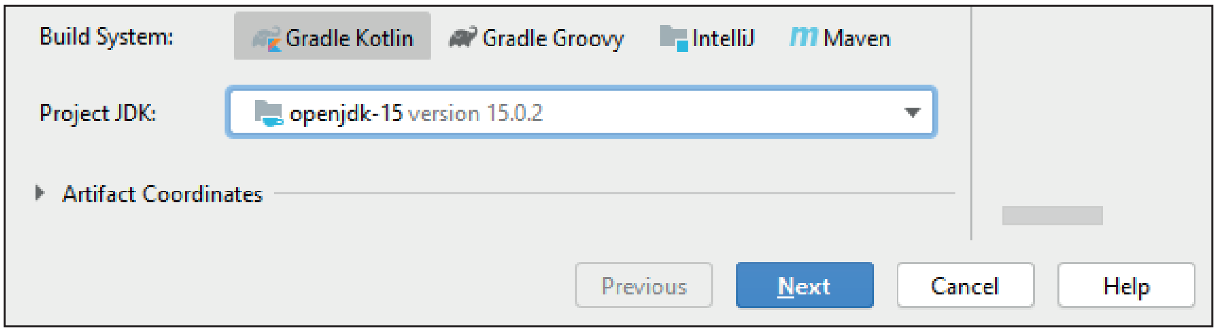
The download process will start as illustrated in the following figure:


Downloading JDK...

Installing Oracle OpenJDK 15.0.2...

Downloading

In this step and as illustrated in the figure below, click **Next.**

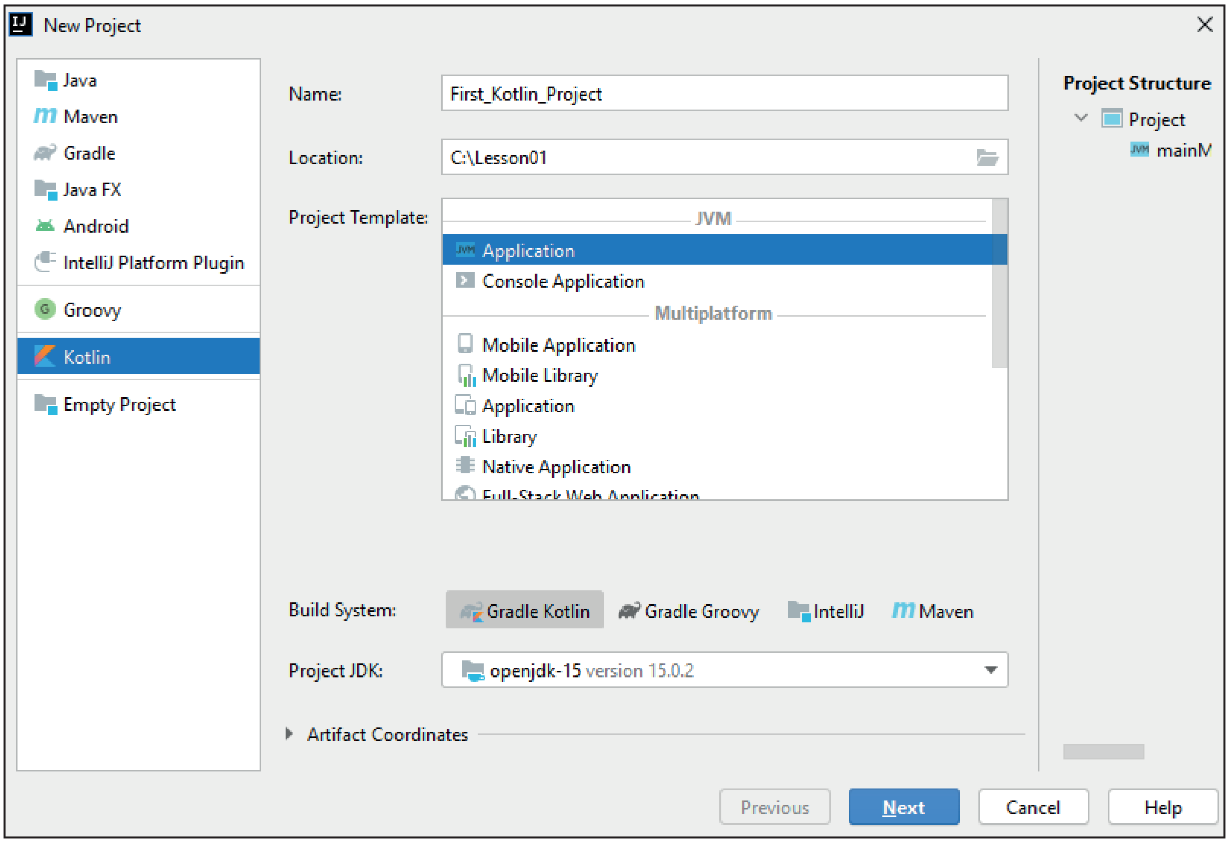

Build System: Gradle Kotlin | Gradle Groovy | IntelliJ | Maven

Project JDK: openjdk-15 version 15.0.2

Artifact Coordinates

14- In this example, we will name the Kotlin project: **First_Kotlin_Project** and save it in a separate folder. Here in this lesson, we will use **Lesson01** folder (C:\Lesson01) to save this new Kotlin project file.
In IntelliJ IDEA, a project helps you organize everything that is necessary for developing your application in a single unit (folder).

Click **Next.**


New Project

Name: First_Kotlin_Project

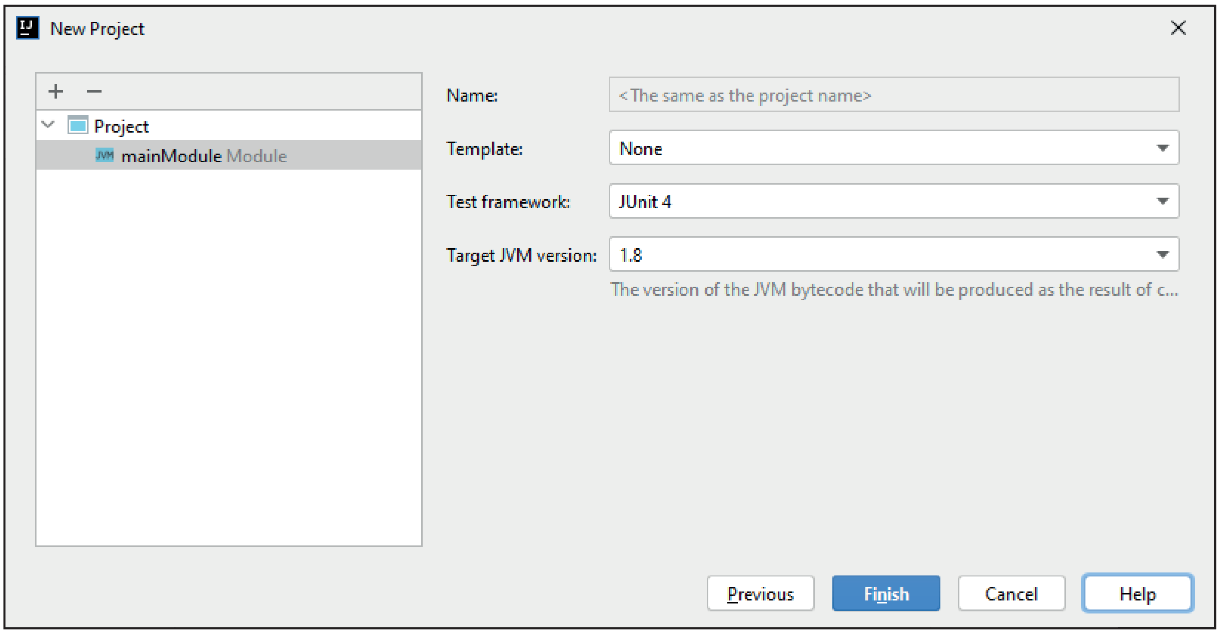
Location: C:\Lesson01

Project Template: Application

Build System: Gradle Kotlin

Project JDK: openjdk-15 version 15.0.2

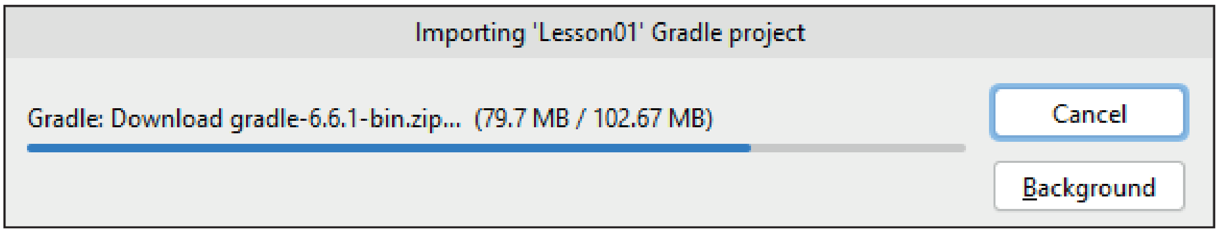
15. In the last step and as illustrated in the figure below, click **Finish**.


New Project

Project
mainModule Module

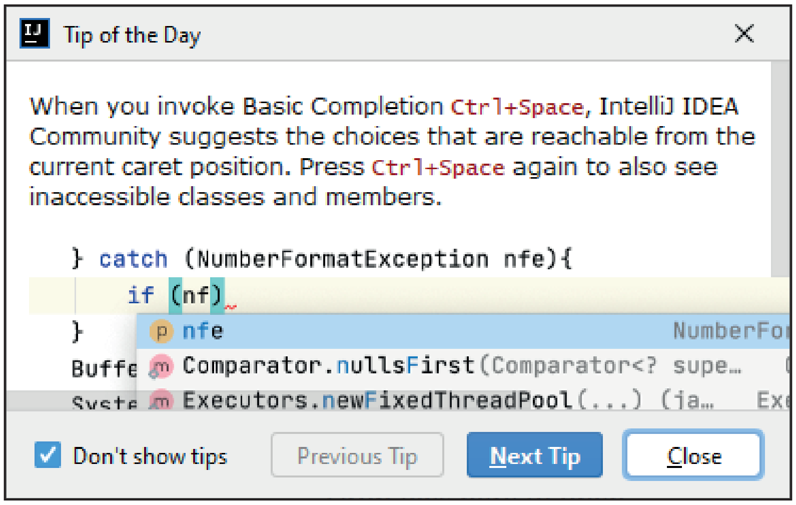
Name: <The same as the project name>
Template: None
Test framework: JUnit 4
Target JVM version: 1.8
The version of the JVM bytecode that will be produced as the result of c...

Once you clicked the **Finish** button in the previous step, the **Gradle** extension download step will start as illustrated in the figure below. In brief, Gradle makes it easy to build common types of project — say Java libraries — by adding a layer of conventions and prebuilt functionality through plugins. Therefore, to run any Kotlin program we need to install the Gradle extension in the IntelliJ IDEA.However, these steps with IntelliJ IDEA will be done automatically.


Importing 'Lesson01' Gradle project

Gradle: Download gradle-6.6.1-bin.zip... (79.7 MB / 102.67 MB)

16- Select **Don't show tips**, then click **Close** as illustrated in the figure below:

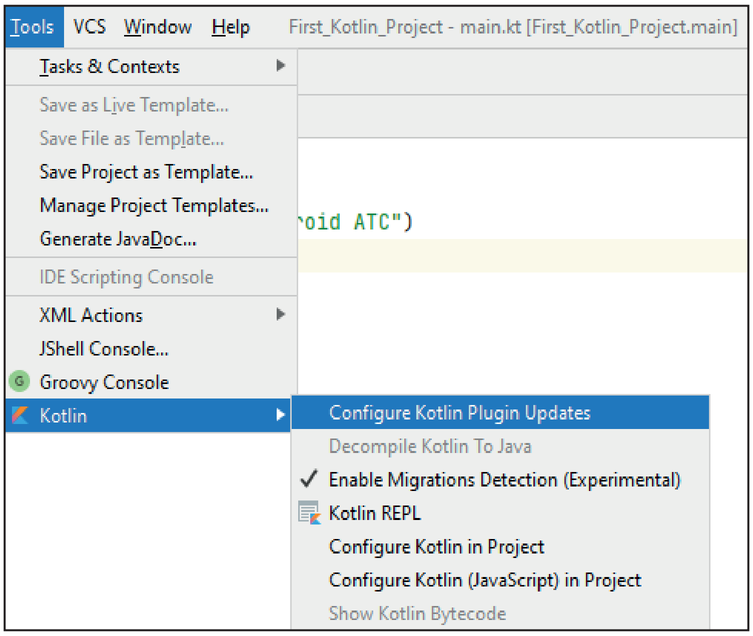

Tip of the Day

When you invoke Basic Completion Ctrl+Space, IntelliJ IDEA Community suggests the choices that are reachable from the current caret position. Press Ctrl+Space again to also see inaccessible classes and members.

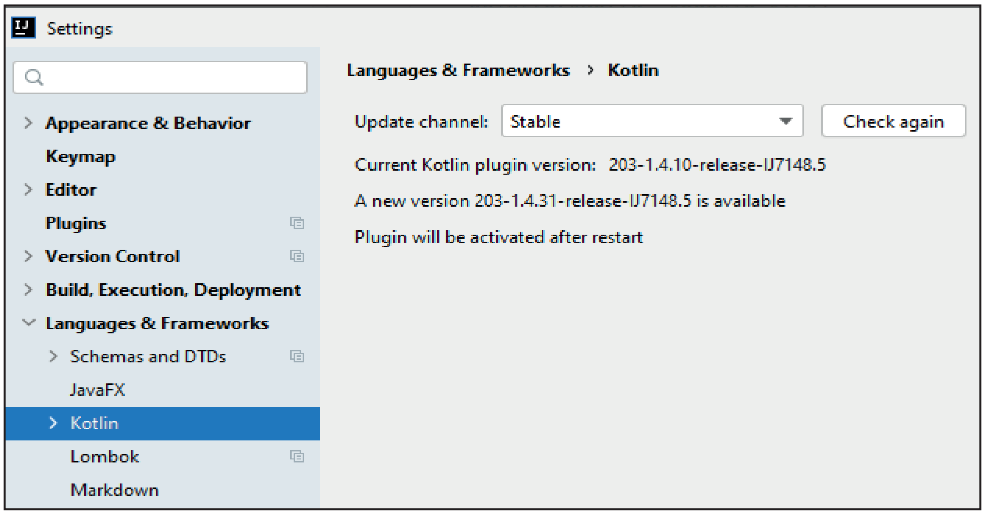
```
} catch (NumberFormatException nfe){
    if (nf)
}
Buffe   nfe                                   NumberFo
Syste   Comparator.nullsFirst(Comparator<? supe...
        Executors.newFixedThreadPool(...) (ja...   Ex
```

☑ Don't show tips

17- Click **Tools → Kotlin → Configure Kotlin Plugin Updates** as illustrated in the figure below:


Tools
VCS
Window
Help
First_Kotlin_Project - main.kt [First_Kotlin_Project.main]
Tasks & Contexts
Save as Live Template...
Save File as Template...
Save Project as Template...
Manage Project Templates...
Generate JavaDoc...
IDE Scripting Console
XML Actions
JShell Console...
Groovy Console
Kotlin
Configure Kotlin Plugin Updates
Decompile Kotlin To Java
Enable Migrations Detection (Experimental)
Kotlin REPL
Configure Kotlin in Project
Configure Kotlin (JavaScript) in Project
Show Kotlin Bytecode

Click **Check again** button to get the latest version of Kotlin plugin, select the new version, and restart the IntelliJ (close & open it again).



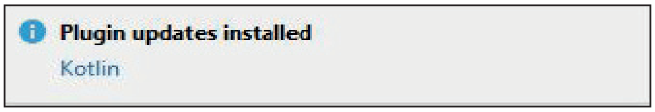
Update channel: Stable

Current Kotlin plugin version: 203-1.4.10-release-IJ7148.5

A new version 203-1.4.31-release-IJ7148.5 is available

Plugin will be activated after restart

You will get the following notification:

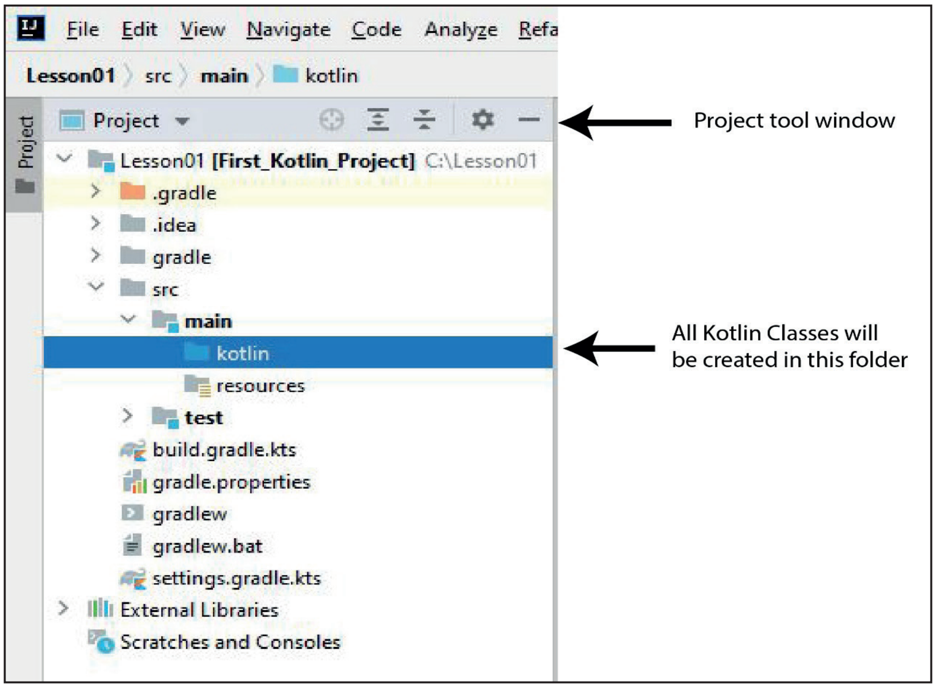


Plugin updates installed

Kotlin

Now, IntelliJ is ready to create your first Kotlin program.

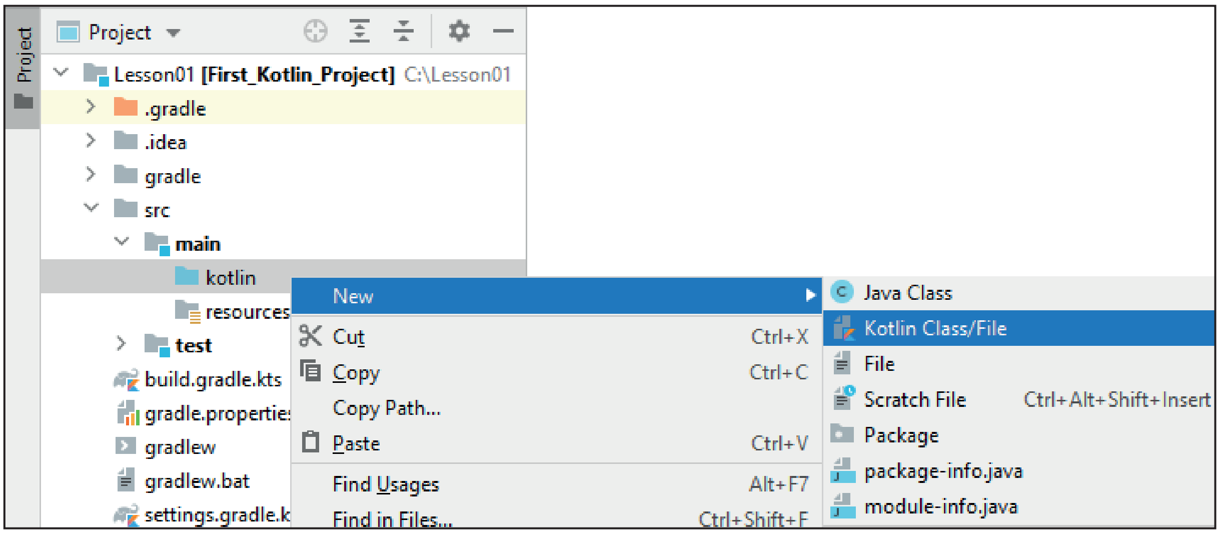
18- In the project console, expand the files as illustrated in the figure below:



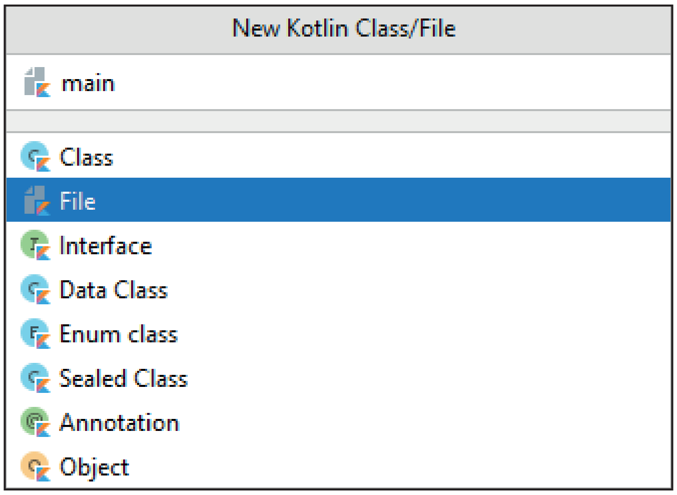
Creating a Kotlin Program

A Kotlin program or Kotlin project consists of a group of classes; each file performs a part of the Kotlin program.

To create a Kotlin file, follow these steps:
1- Right click the **kotlin** folder (Lesson01 → src → main → kotlin), select **New** → **Kotlin Class/File**
As illustrated in the figure below:

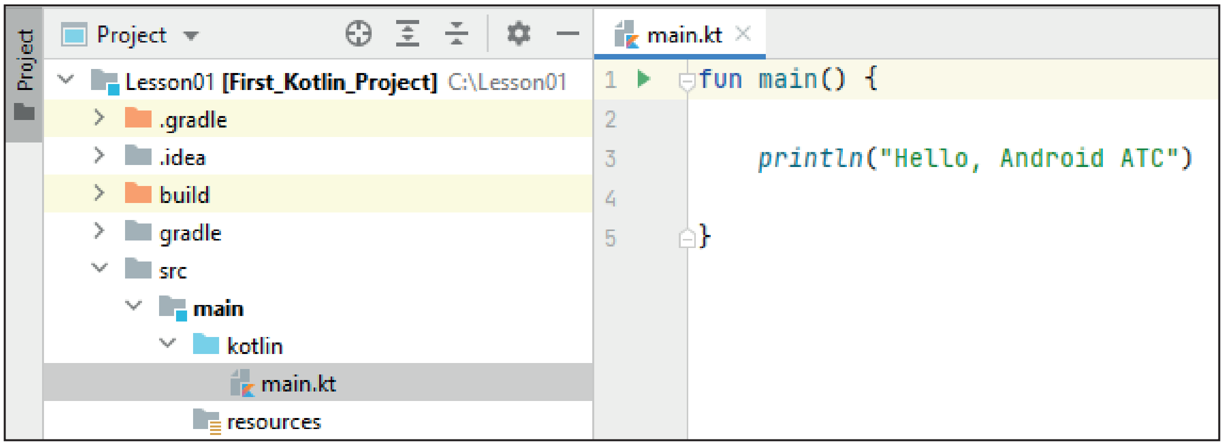


2- You will get the dialog box below. Type the Kotlin file name: **main**, then press **Enter.**


New Kotlin Class/File

main

3- You can type the code below to test how Kotlin code is running. We will explain each part of this code later on.



```
fun main() {

    println("Hello, Android ATC")

}
```

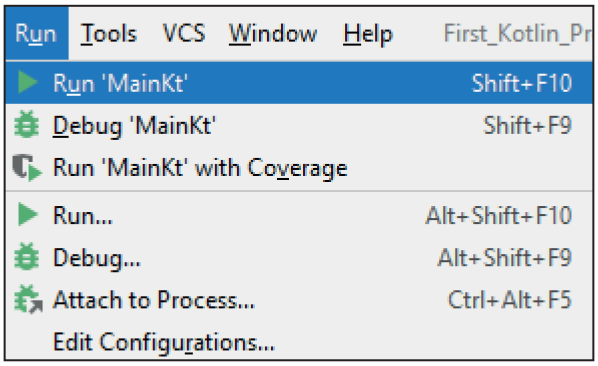
The code is as follows:

```
fun main() {

    println("Hello, Android ATC")

}
```

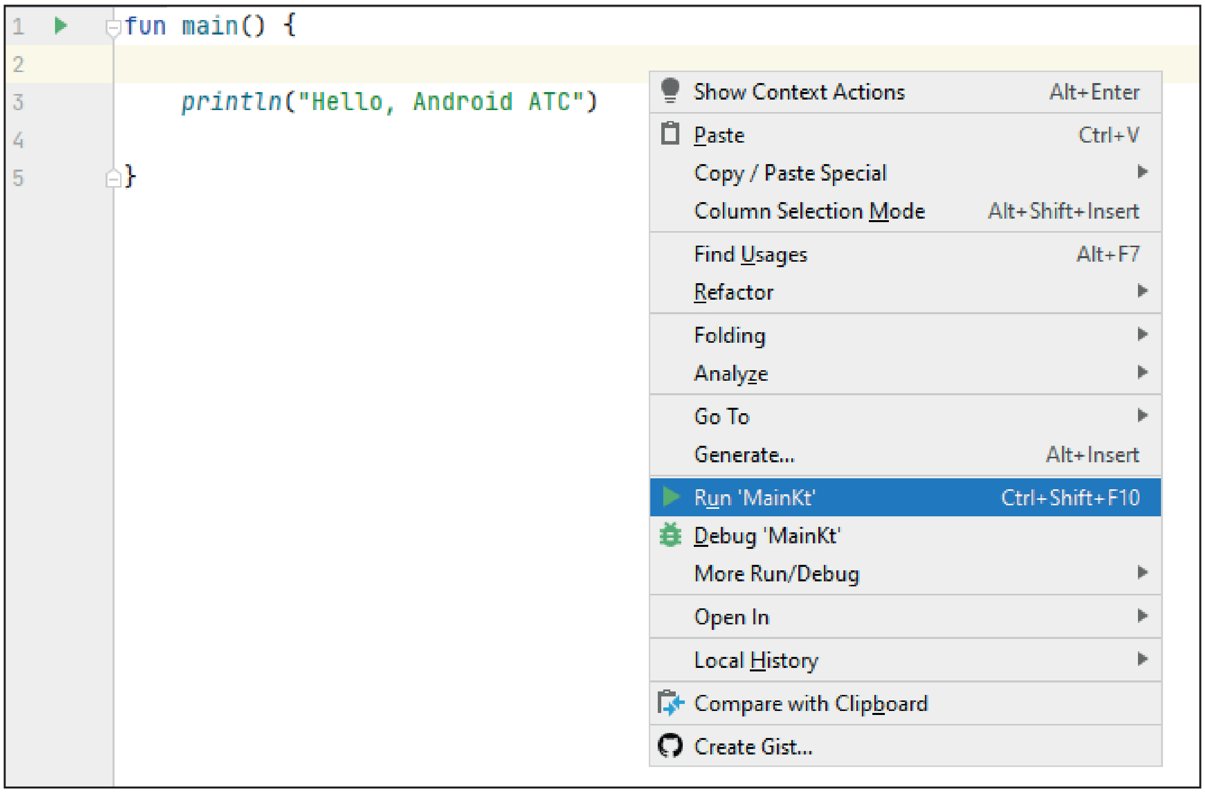
Note: The "`println`" method is used in Kotlin to print text or variable values.

Running a Kotlin Program

You may run your Kotlin code by clicking the **Run** menu, then select **Run** as illustrated in the following figure:

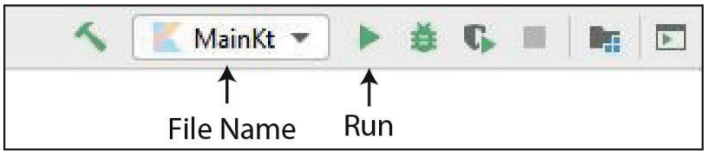


Also, you may run your code through right click on any place on the Kotlin code, and select **Run** from the short cut menu as illustrated in the figure below:

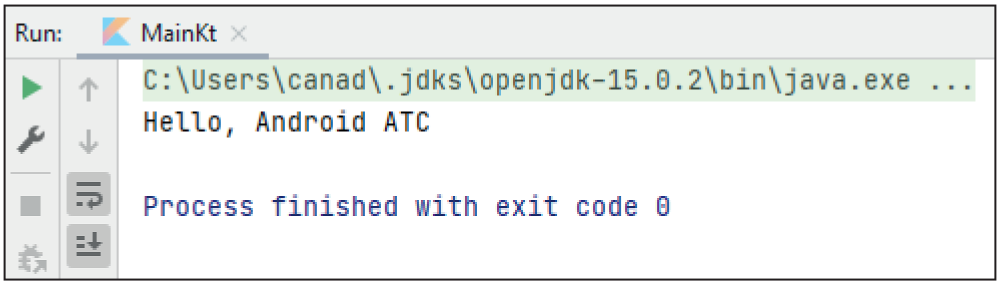


```
fun main() {

    println("Hello, Android ATC")

}
```

Also, you may use the IntelliJ tool bar to run your Kotlin code. Just select your Kotlin file or class name, and then click the run button ▶ as illustrated in the figure below:



The run result will be in the run console as illustrated in the figure below:



```
C:\Users\canad\.jdks\openjdk-15.0.2\bin\java.exe ...
Hello, Android ATC

Process finished with exit code 0
```

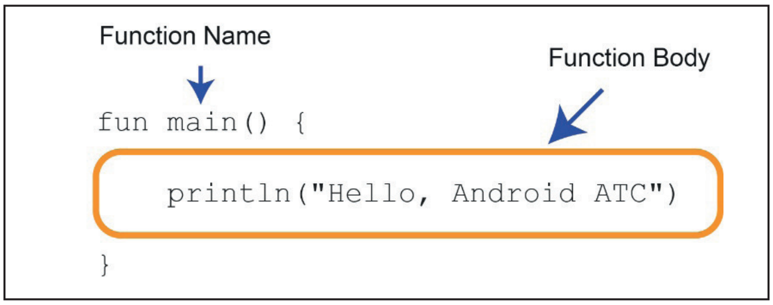
The main( ) function

A function is any close identity which includes a certain code or a collection of statements grouped together to perform an operation.

Every app must have a top-level **`main()`** function, which serves as the entry point to the app.

When you run your Kotlin program, only the code which belongs to the main function body will run while any code outside this function body will not be considered.

The following image displays the **`main()`** function structure:



```
fun main() {
    println("Hello, Android ATC")
}
```

All your Kotlin code must be written inside the **`main()`** function body.

Example:

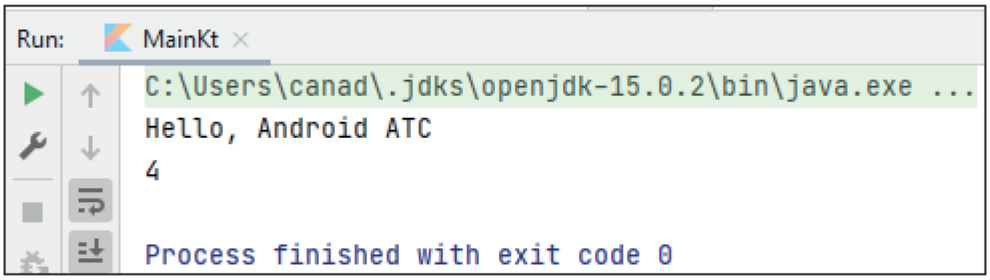
To understand more how Kotlin program works and how easy the language is, we are going to add some variables to the previous example and print their sum.

To add any variable to a Kotlin program, you have to determine the data type of this variable: an integer, a string, a double, etc… or you can just add it without any need to declare the type.

To define a variable in Kotlin program, you must type **val** before the variable letters. The following example gives you an idea of how to add two variables (**x** and **y**) to the previous Kotlin program and print out their sum:

```
val x=1
val y=3

fun main() {

    println("Hello, Android ATC")
    val z=x+y
    println(z)
}
```

Here, the run result will be as follows:



```
Run:   MainKt
C:\Users\canad\.jdks\openjdk-15.0.2\bin\java.exe ...
Hello, Android ATC
4

Process finished with exit code 0
```

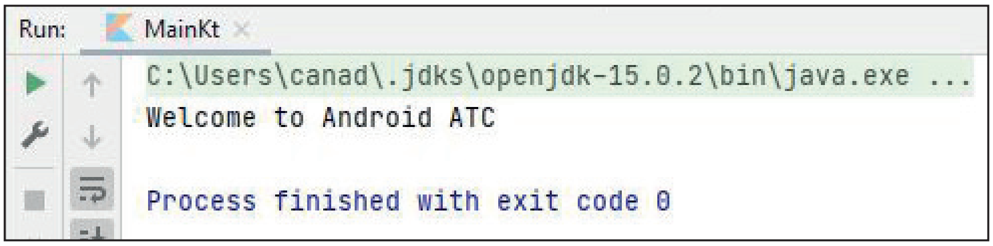
Note: In the previous example, you declared the value of the **x** and **y** outside the main function. Also, you did not declare the data type of these two variables; therefore, Kotlin considered them as integer data type.

Example:

In this example, you will add another function called **`test()`** outside the **`main()`** function body as illustrated in the code below:

```
fun main() {
    println("Welcome to Android ATC")

}

fun test() {
    println("Hello Kotlin Developers")
}
```

When you run this Kotlin code, you will get the same previous result without any effect to the content of the **`test()`** function. The run result will be as follows:

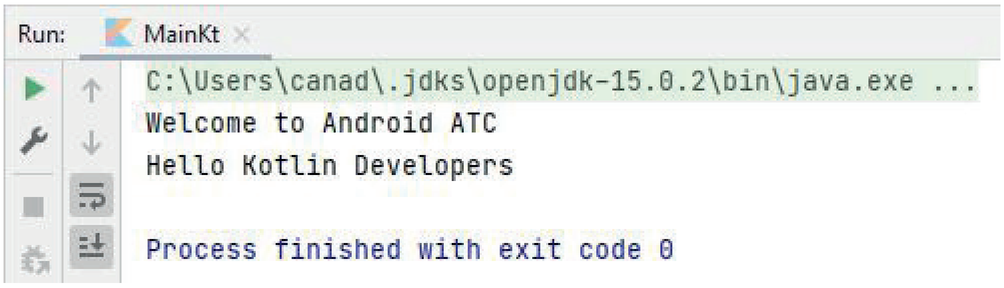


```
Run:   MainKt
C:\Users\canad\.jdks\openjdk-15.0.2\bin\java.exe ...
Welcome to Android ATC

Process finished with exit code 0
```

This means that when you run any Kotlin code, only the codes inside the **`main()`** function body will run. Also, if you want to run any function outside the **`main()`** function body, you must add a reference to this external function inside the **`main()`** function to call its operation to run inside the main function as illustrated in the following code:

```
fun main() {
    println("Welcome to Android ATC")
    test()

}

fun test() {
    println("Hello Kotlin Developers")
}
```

The run result of the code above will be as follows:



```
C:\Users\canad\.jdks\openjdk-15.0.2\bin\java.exe ...
Welcome to Android ATC
Hello Kotlin Developers

Process finished with exit code 0
```

Writing Comments

Comments are used to write short descriptions about different parts of the Kotlin program such as comment about a specific part of Kotlin code, or it would be a comment to the programmer who may change your code in future. The content of your Kotlin comments will not appear to your end users who will run your Kotlin program.

In Kotlin, you may use two types of comments: **line** comments and **block** comments.

Line comments

Any line starting with double forward slash "//" will be considered a comment by the Kotlin compiler. This means that this part will not run or appear to the users of this application because it will remain internal.

Block comments

A block comment is like a line comment, but it includes more than one line and it starts with **/*** and ends with ***/**. These (/* & */) are used to add multiple lines as comments in the code without adding **//** at the beginning of each line.

The following example includes a line and a block comment:

```
/*
Created by Android ATC on 1/10/2021.
 */

val x=1 // here x=1, I did not declare the data type of x
val y=3
fun main() {

    println("Hello, Android ATC")
    val z=x+y
    println(z)
}
```

If you run the code above, you will get the same previous run result. The comments' contents do not appear in the program run result.

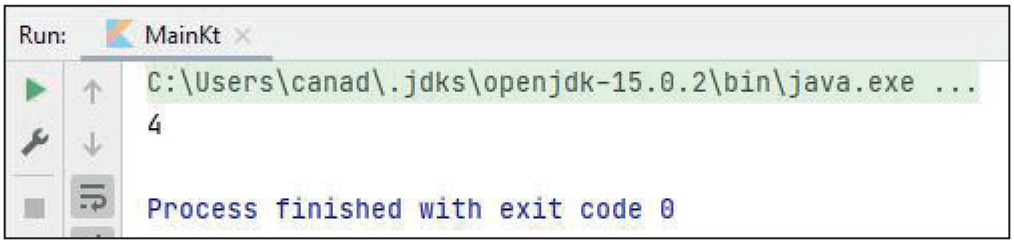
Kotlin Variables

A Kotlin variable is a piece of memory that can contain a data value. Variables are typically used to store information which your Kotlin program needs to do its job. These variables are case sensitive. In Kotlin, we have two types of variables:

1) **Mutable variables:**
   A mutable variable is one whose value can be changed anytime. To declare a mutable variable, we type **var** directly before the variable.
   The following example gives you an idea of how to use a mutable variable:

```

var x=1 // the value of x=1

fun main() {
    x=4 //the value of X has been changed from 1 to 4
    println(x)
}
```

When you run the Kotlin program, you will get the following result for x value:

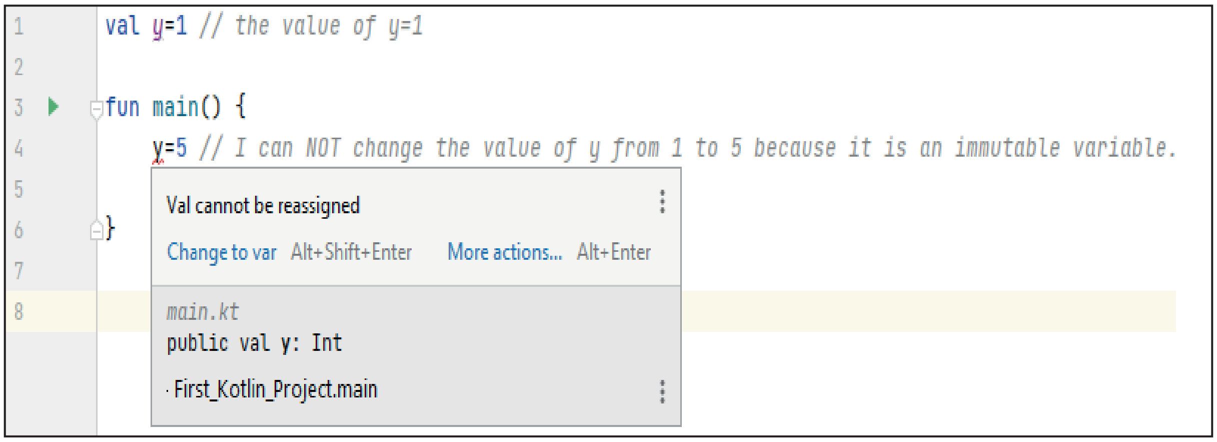


```
C:\Users\canad\.jdks\openjdk-15.0.2\bin\java.exe ...
4

Process finished with exit code 0
```

2) **Immutable variables:**
An immutable variable is a variable whose state, like constants, can't be changed after initialization. If you need to change the value, a new variable should be created. To declare an immutable variable, we type **val** directly before the variable. The following example shows how to use an immutable variable:

```
val y=1 // the value of y=1

fun main() {
    y=5 // I can NOT change the value of y from 1 to 5 because it
is an immutable variable.
    println(y)
}
```

You will get a red underline error under **y** and if you move the mouse pointer over this error underline, you will get a notification which displays that you cannot change the value of **y,** and to do that, you must change the variable type of **y** to **var** as illustrated in the figure below:



```
val y=1 // the value of y=1

fun main() {
    y=5 // I can NOT change the value of y from 1 to 5 because it is an immutable variable.
}
```

```
Val cannot be reassigned
Change to var  Alt+Shift+Enter    More actions...  Alt+Enter
main.kt
public val y: Int
· First_Kotlin_Project.main
```

Kotlin Data Types

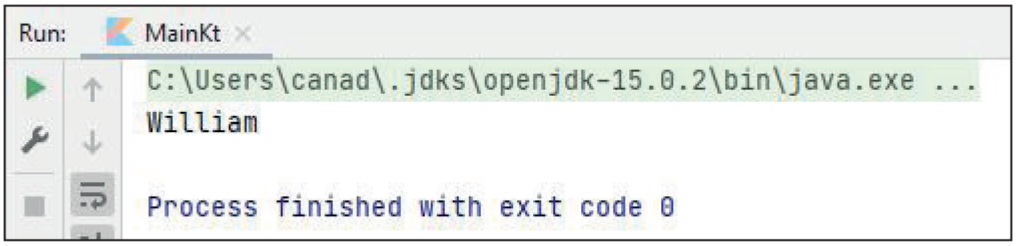
The basic data types used in Kotlin are strings, characters, Booleans, numbers, and arrays. Here is an example with some details about each data type:

1-String:
String data type is used to store words or sentences.

Example: In the code below, the **Name** variable is declared implicitly string data type and its value must be declared between double quotes

```
val Name="William" // Name is implicitly defined to be String

fun main() {

    println(Name)
}
```

When you run the Kotlin program, you will get the following result:



```
C:\Users\canad\.jdks\openjdk-15.0.2\bin\java.exe ...
William

Process finished with exit code 0
```

If you want to declare the data type of the variable name as string, you can write the following code:

```
val Name:String="William"

fun main() {

    println(Name)
}
```

Note: A string value is always enclosed in double quotes.

2- Character

Character data type is used to store a single character such as ‘a’, ‘Z’, ‘&’, ‘$’,’8’ and so on. Characters may be letters or special symbols. Characters are always enclosed in single quotes, while double quotes are for Strings.

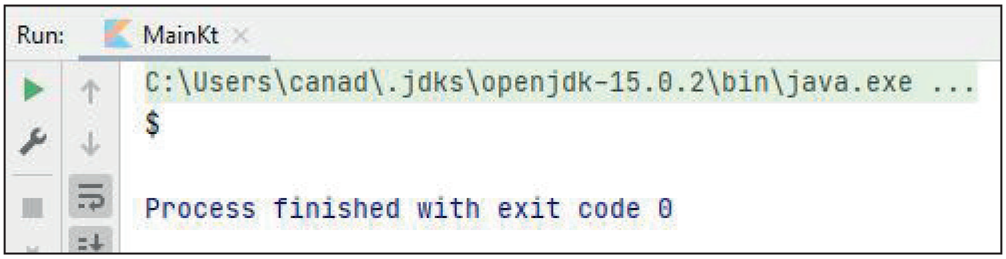
If you want to store the value ‘$’ in a variable x, you should write the following statement:

```
val x:Char='$'

fun main() {

    println(x)
}
```

You can declare the data type char for $ implicitly, as illustrated in the following example:

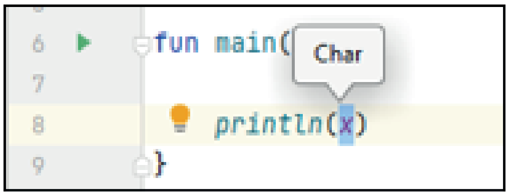
```
val x='$'

//here, because you have used single quotes, the Kotlin compiler
considers it as char; otherwise, if you used double quotes it will
be considered as string.

fun main() {

    println(x)
}
```

When you run the Kotlin program, you will get the following result:



```
C:\Users\canad\.jdks\openjdk-15.0.2\bin\java.exe ...
$

Process finished with exit code 0
```

Note: You can place the cursor inside the variable and press **Ctrl + Shift + P** to see its type. (See illustration below.)



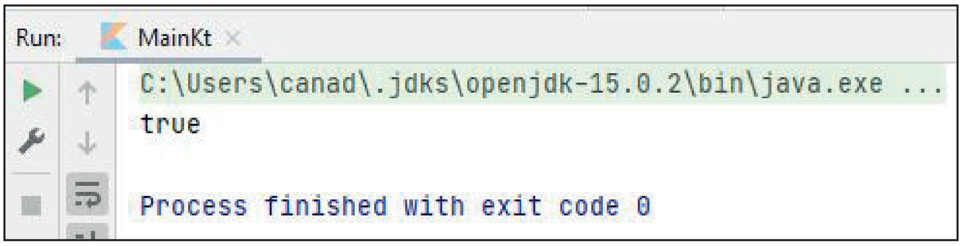
```
fun main(
    println(x)
}
```

3- Booleans

A Boolean data type has two possible values; either true or false. Booleans are used in decision-making statements which you can control in the program work flow.

```
val x:Boolean=true

fun main() {

    println(x)
}
```

Run the Kotlin program and you will get the following result:



```
C:\Users\canad\.jdks\openjdk-15.0.2\bin\java.exe ...
true

Process finished with exit code 0
```

4-Numbers

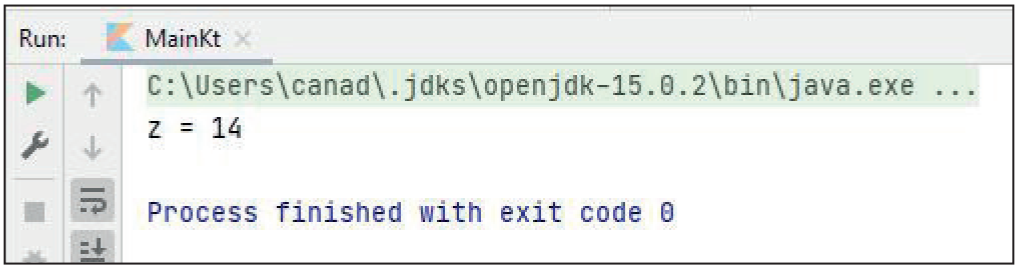
Kotlin provides the following built-in types that represent numbers:

Data Type	Description	Default Value
Byte	The byte data type is an 8-bit signed integer. Its value is from 0 to 255	0
Short	The short data type is a 16-bit signed integer. Its value ranges from - 32768 to 32767	0
Int	The integer data type is a 32-bit signed integer. It has values from -2,147,483,648 to +2,147,483,647	0
Long	The long data type is a 64-bit signed integer.	0L
Float	The float data type is a single-precision 32-bit floating point.	0.0f
Double	The double data type is a double-precision 64-bit floating point.	0.0d

The following example shows how to declare integer variables and use them in a sum formula:

```
var x:Int=9
var y:Int=5

fun main() {
    var z:Int=x+y
    println("z = " + z)
}
```

When you run the Kotlin program, you will get the following result:



```
Run:   MainKt
C:\Users\canad\.jdks\openjdk-15.0.2\bin\java.exe ...
z = 14

Process finished with exit code 0
```

The following code gives the same result because the variables x, y and z are considered implicitly integer variables.

```
var x=9
var y=5

fun main() {
    var z=y+x
    println("z = " + z)
}
```

Conclusion: In Kotlin programming language, you don't need to declare the data type for variables. The IDE considers any variable as Chart if enclosed in single quotes, as a String if in double quotes, and as a number if there are no quotes.

Question: What will IDE consider the value of the variable x in the following code: a Byte, an Integer or a Short?

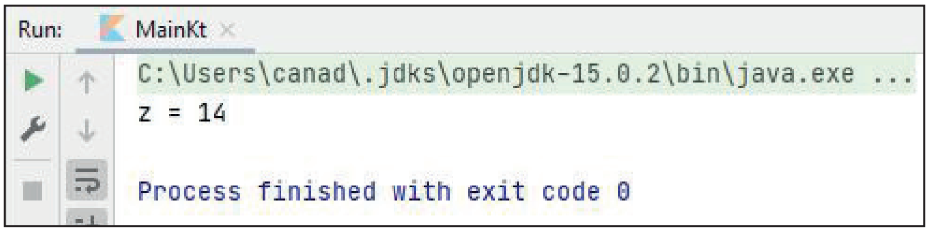
```
var x=9
```

Answer: It will be considered Integer by default, and ,if you want to declare it as Byte, you should do it explicitly as Byte. The example below displays how you can declare a variable as a Byte variable.

Example: The following code displays the sum of two Byte variables:

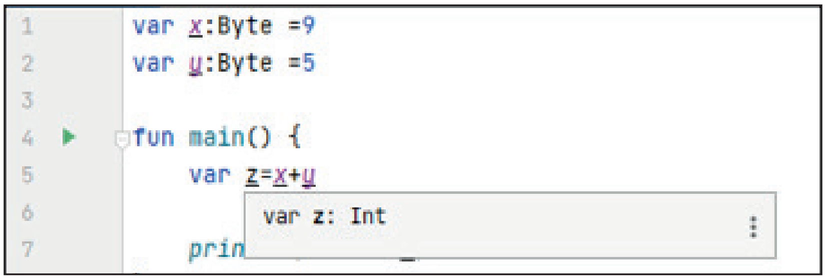
```
var x:Byte =9
var y:Byte =5

fun main() {
    var z=x+y

    println("z = "+z)
}
```

When you run the Kotlin program, you will get the following result:



```
C:\Users\canad\.jdks\openjdk-15.0.2\bin\java.exe ...
z = 14

Process finished with exit code 0
```

Now, what will the complier consider the data type of **z**? An Integer or a Byte?
The answer is: It will consider it as an Integer.



```
var x:Byte =9
var y:Byte =5

fun main() {
    var z=x+y
                var z: Int
    prin
```

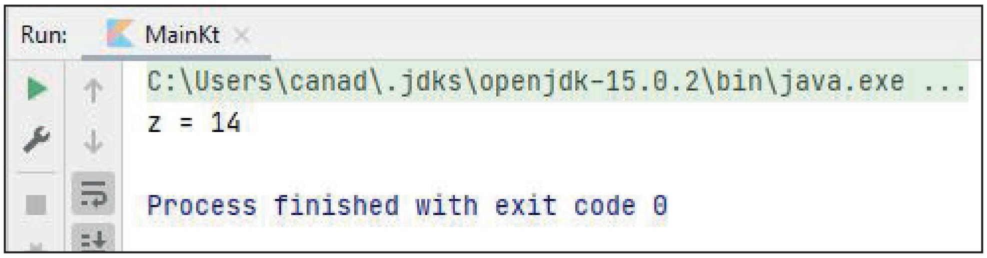
In the previous example, you can use the command `println()` in different ways to make writing in Kotlin program more flexible as follows:

`println(z)` : The output will be **14**
`println("z="+z)` : The output will be **z=14**

`println("z=$z")` : Here, the dollar sign in $z means the value of this variable. The following code shows example of how to use dollar sign with `println` method.

```
var x:Byte =9
var y:Byte =5

fun main() {
    var z=x+y

    println("z = $z")
}
```

The following is the run result of the code above:



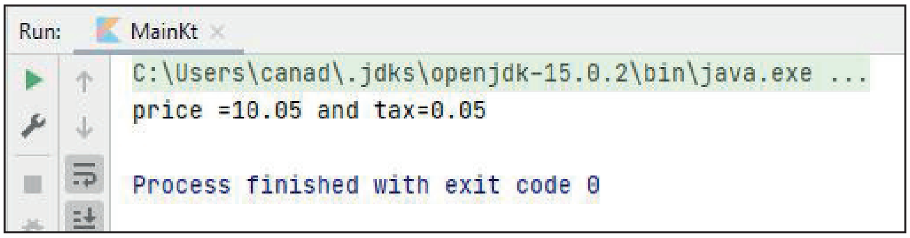
```
C:\Users\canad\.jdks\openjdk-15.0.2\bin\java.exe ...
z = 14

Process finished with exit code 0
```

Using the previous technique is easier and more professional.

You can't get a calculation result with decimal value if you use Integer or Byte variables. Instead of that, you should use Double variables.
The following example displays how you can declare a Double data type both explicitly and implicitly for a variable.

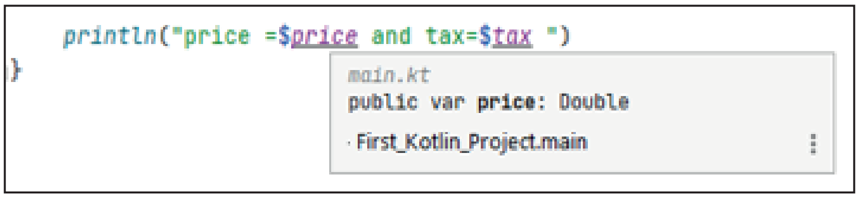
```
var price:Double =10.05 // explicitly Double
var tax = 0.05   // implicitly Double

fun main() {

    println("price =$price and tax=$tax ")
}
```

When you run this Kotlin program you will get the following result:



```
C:\Users\canad\.jdks\openjdk-15.0.2\bin\java.exe ...
price =10.05 and tax=0.05

Process finished with exit code 0
```

If you move the mouse pointer over the $price variable, you will get the following:

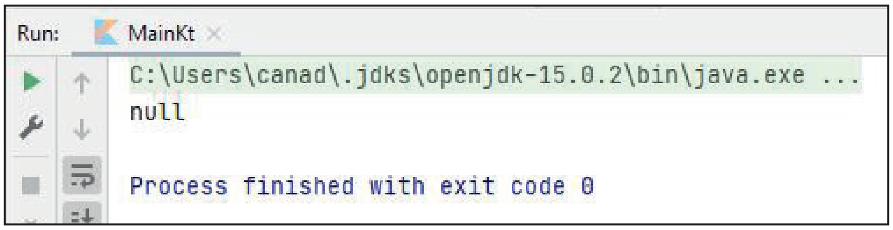


```
    println("price =$price and tax=$tax ")
}
                     main.kt
                     public var price: Double
                     First_Kotlin_Project.main
```

You can also declare a data type for a variable first, and later assign its value from a specific code operation (you will see how that can happen in the next lessons), as shown in the following example:

```
fun main() {
    var First_Name :String?=null // explicitly declared as String
with Null value.

    println(First_Name)
}
```

When you run the code above, you will get the following result:

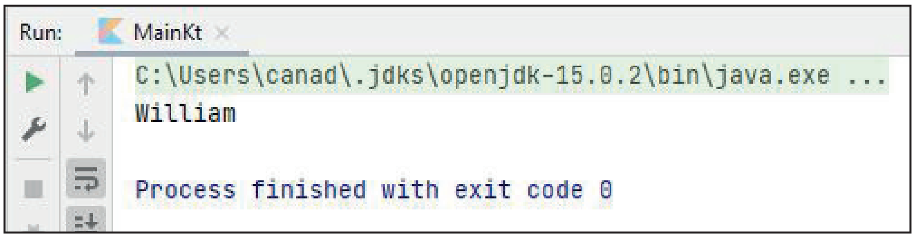


```
C:\Users\canad\.jdks\openjdk-15.0.2\bin\java.exe ...
null

Process finished with exit code 0
```

You may assign a value for this variable (`First_Name`) as illustrated in the following code:

```
fun main() {
    var First_Name : String? =null
    First_Name= "William"

    println(First_Name)
}
```

When you run the Kotlin code above, you will get the following result:



```
C:\Users\canad\.jdks\openjdk-15.0.2\bin\java.exe ...
William

Process finished with exit code 0
```

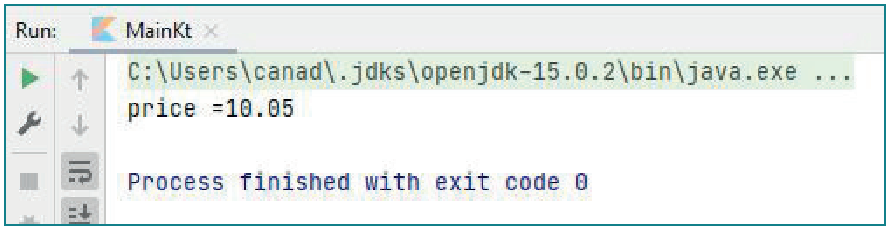
Differences between Double and Float variables:

Most programmers use double data type for variables which are expected to have a fraction value because doubles ensure **higher precision** and they are double-precision (64-bit), while **Floats** are single-precision (32-bit).

If you declare a Float variable, you must add "F" (upper case or lower case) beside the variable value as illustrated in the following code:

```
fun main() {
    var price:Float= 10.05f
    println("price =$price")
}
```

When you run the Kotlin code above, you will get the following result:



```
C:\Users\canad\.jdks\openjdk-15.0.2\bin\java.exe ...
price =10.05

Process finished with exit code 0
```

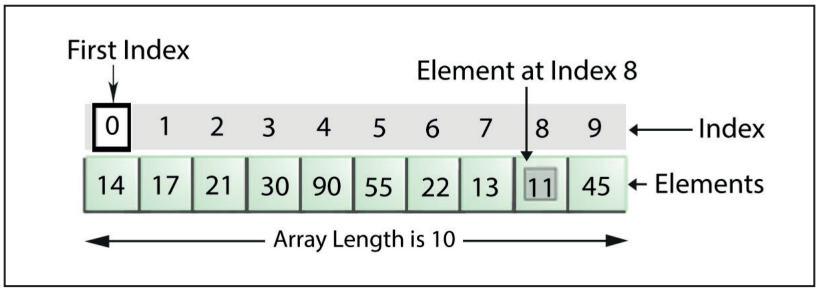
Note: If you assign any fraction value to any variable without declaring it explicitly, as Float, Kotlin compiler will consider it implicitly a Double data type.

5-Array

So far, you have learned about each variable stored as one data item. If you want to store a large number of data items for the same variable, you need to use the array. An array is used to store a group of values, all of which have the same data type. The length of an array is established when the array is created. After creation, its length is fixed.

Much like C, C++, Java, Dart or Kotlin, arrays are indexed numerically on a 0-based system. This means that the first element in the array is at index 0, the second is at index 1, and so on.

For example, the following image displays an array of 10 elements.



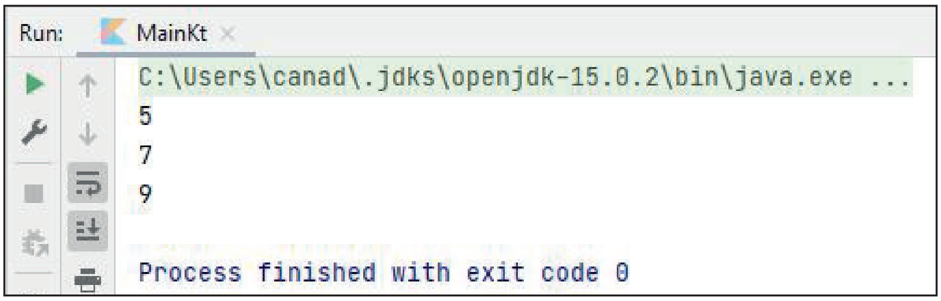
To create an array, we can use a library function `arrayOf()` and pass the item values to it, so that an array of (5, 7, 9) creates an array [5, 7, 9].

The following example explains more how you may use array variable with Kotlin:

Example:

```kotlin
fun main() {

    var x= arrayOf(5, 7, 9)
    println(x[0]) // print the element in index 0
    println(x[1]) // print the element in index 1
    println(x[2]) // print the element in index 2

}
```

When you run the code above, you will get the following result:



```
C:\Users\canad\.jdks\openjdk-15.0.2\bin\java.exe ...
5
7
9

Process finished with exit code 0
```

If you move your mouse pointer over any of these array elements, for example, `x[0]`, you will get a message displaying that the data type of this array element is Integer. It is implicitly Integer.

If you want to declare that all the array elements' values of the array x in the previous example to be explicitly integers, you must use `IntArray` as follows:

```
fun main() {

    var x:IntArray= intArrayOf(5, 7, 9)
    println(x[0]) // print the element in index 0
    println(x[1]) // print the element in index 1
    println(x[2]) // print the element in index 2

}
```

Note: The first letter in `InArray` is uppercase while in `intArrayOf,` the first letter is lowercase.

When you run the Kotlin program, you will get the same previous result:

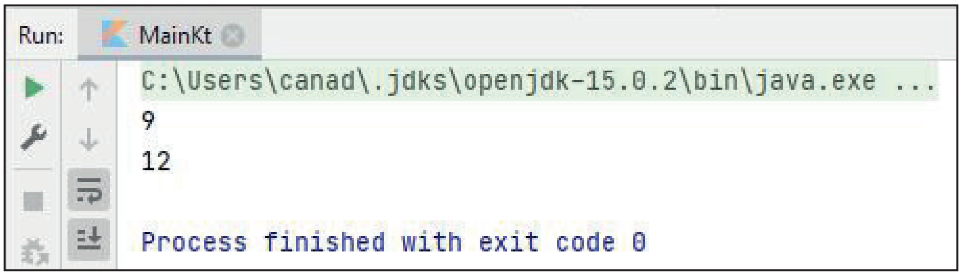
You can also use `ByteArray` for Byte numbers and `ShortArray` for Short numbers.

Example:

The following code displays how you may use some elements of the array in some calculations. See the following:

```
fun main() {

    var x:IntArray= intArrayOf(5, 7, 9)
    println(x[2])
    x[2]=x[0]+x[1]
    println(x[2])

}
```

The run result of the code above is as follows:



```
C:\Users\canad\.jdks\openjdk-15.0.2\bin\java.exe ...
9
12

Process finished with exit code 0
```

This means that the x array now includes the following values: **x** `[5 7 12]`

In addition, you can use the `arrayOf` with other data types such as String or Char as illustrated in the following code:

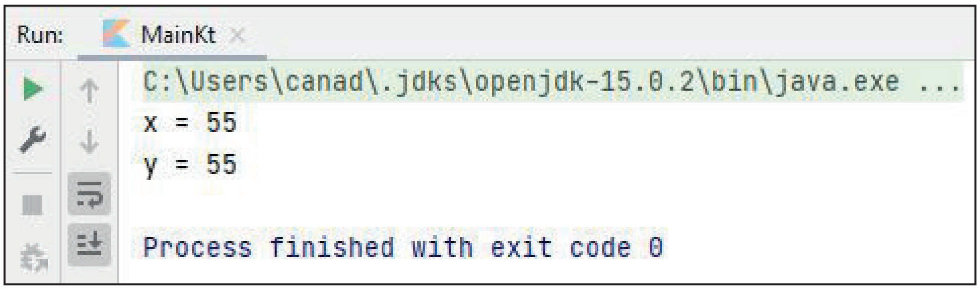
```
fun main() {

    var y= arrayOf(1,2,3) //implicitly Integer
    var Winter = arrayOf("January", "February", "March") //
implicitly String
    var Z= arrayOf('A','B','C') //implicitly Char
}
```

Data Type Conversions

In some cases, you may need to convert a data type for a variable to another data type. For example, changing an Integer variable to a short variable. To do that, you need to use `toShort()` function explicitly to convert the variable to data type short, as is illustrated in the following example:

```
fun main() {

    val x: Int = 55
    val y: Short = x.toShort()
    println("x = $x")
    println("y = $y")
}
```

When you run the Kotlin code above, you will get the following result:



```
C:\Users\canad\.jdks\openjdk-15.0.2\bin\java.exe ...
x = 55
y = 55

Process finished with exit code 0
```

The following is a list of functions in Kotlin used for data type conversions:

- `toByte()`
- `toShort()`
- `toInt()`
- `toLong()`
- `toFloat()`

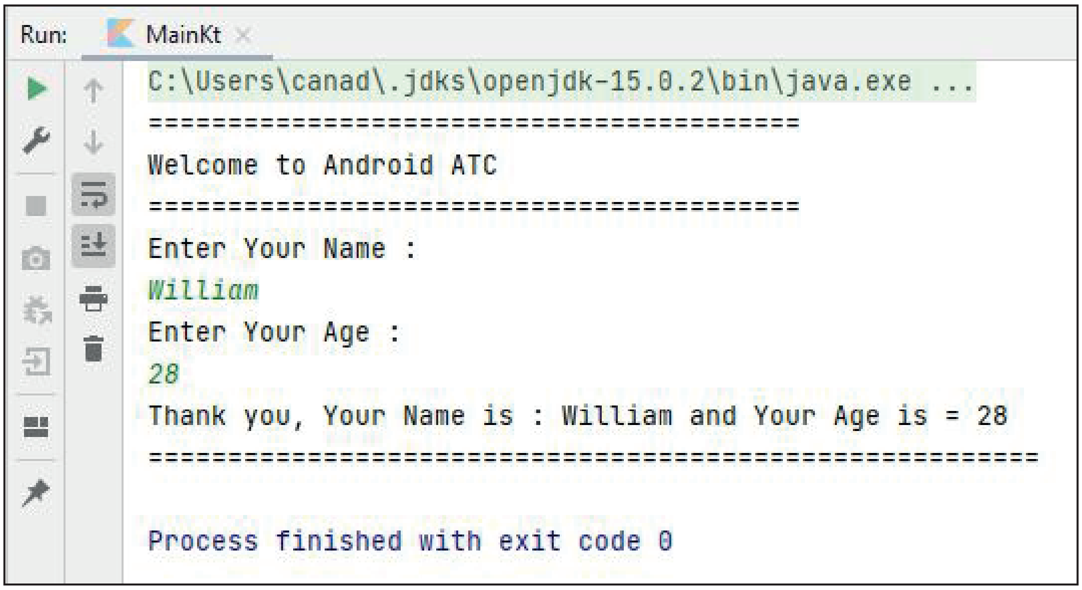
- `toDouble()`
- `toChar()`
- `toString()`

Input of Information to Kotlin Program

`readLine()` function allows the Kotlin program user to enter string values or intercept keyboard input from the console as shown in following example:

```
fun main()  {

    println("=========================================")
    println("Welcome to Android ATC")
    println("=========================================")
    println("Enter Your Name :")
    var x= readLine()

    println("Enter Your Age :")
    var y= readLine()

    println("Thank you, Your Name is : $x and Your Age is = $y")

println("=========================================================")

}
```

When you run the Kotlin program, you will get the following result:

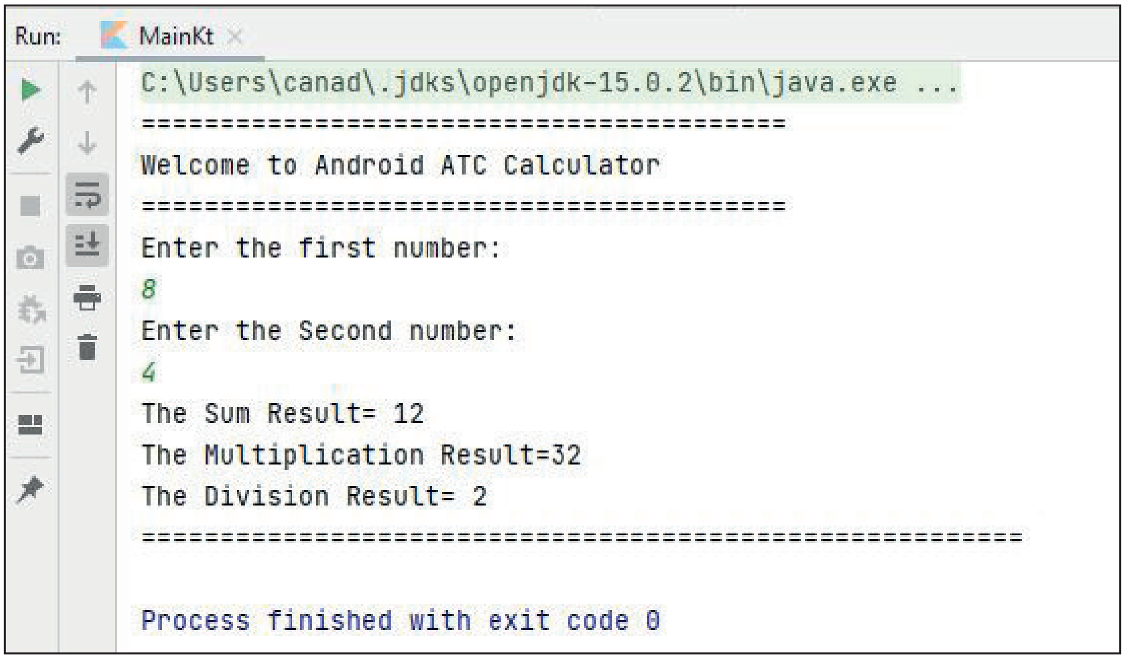


```
C:\Users\canad\.jdks\openjdk-15.0.2\bin\java.exe ...
========================================
Welcome to Android ATC
========================================
Enter Your Name :
William
Enter Your Age :
28
Thank you, Your Name is : William and Your Age is = 28
=======================================================

Process finished with exit code 0
```

To use **`readLine`** `()` to enter integer numbers, you should write this function as illustrated in the following example:

```
fun main() {

    println("=========================================")
    println("Welcome to Android ATC Calculator")
    println("=========================================")
    println("Enter the first number:")
    var x = Integer.valueOf(readLine())
    println("Enter the Second number:")
    var y = Integer.valueOf(readLine())

    var z=x+y
    var a=x*y
    var b=x/y

    println("The Sum Result= $z")
    println("The Multiplication Result=$a")
    println("The Division Result= $b")

    println("=========================================================")

}
```

When you run this Kotlin program (simple calculator), you will get the following result:



```
C:\Users\canad\.jdks\openjdk-15.0.2\bin\java.exe ...
=========================================
Welcome to Android ATC Calculator
=========================================
Enter the first number:
8
Enter the Second number:
4
The Sum Result= 12
The Multiplication Result=32
The Division Result= 2
=========================================================

Process finished with exit code 0
```

The previous example (calculator) included some arithmetic operators such as (+, /, *). The following table includes some Kotlin arithmetic operators and their corresponding functions:

Expression	Function name	Translates to
x + y	Plus	x.plus(y)
x - y	Minus	x.minus(y)
x * y	Times	x.times(y)
x / y	Div	x.div(y)

You will get the same result if you repeat the same previous code using Kotlin arithmetic operators as illustrated in the following code:

```
fun main() {

    println("=========================================")
    println("Welcome to Android ATC Calculator")
    println("=========================================")
    println("Enter the first number:")
    var x = Integer.valueOf(readLine())
    println("Enter the Second number:")
    var y = Integer.valueOf(readLine())

    var z=x.plus(y) // same x+y
    var a=x.times(y) // same x*y
    var b=x.div(y) //same x/y

    println("The Sum Result= $z")
    println("The Multiplication Result= $a")
    println("The Division Result= $b")

println("=========================================================")

}
```

You can also write the above-mentioned arithmetic operators within `priln()` method by doing the following:

```
fun main() {
    println("=========================================")
    println("Welcome to Android ATC Calculator")
    println("=========================================")
    println("Enter the first number:")
    var x = Integer.valueOf(readLine())
    println("Enter the Second number:")
    var y = Integer.valueOf(readLine())


println("The Sum Result= ${x+y}")
    println("The Multiplication Result=${x*y}")
    println("The Division Result= ${x/y}")


println("=====================================================")

}
```

Note: Understanding the use of different techniques to complete arithmetic operations in Kotlin enables you to create a Kotlin program and also to understand programs created by others.

You can use **`readLine`** command, too, to enter integer numbers using **`readLine()`**`!!.`**`toInt()`** statement as illustrated in the following code:

```
fun main() {

    println("=========================================")
    println("Welcome to Android ATC Calculator")
    println("=========================================")
    println("Enter the first number:")
    var x= readLine()!!.toInt()
    println("Enter the Second number:")
    var y = readLine()!!.toInt()


    println("The Sum Result= ${x+y}")
    println("The Multiplication Result=${x*y}")
    println("The Division Result= ${x/y}")


println("=======================================================")

}
```