12

Android Studio Tips and Tricks to Help You during Development

As an Android developer, writing code should not just be the end goal; rather, understanding how to find issues in your applications, using formatting tips to navigate the code base faster, and other skills come in handy.

The programming process includes a lot of collaboration. This can be in peer code review, pair programming, or debugging issues. In these situations, it is handy if you can move fast, for instance, when debugging or formatting code before you submit your pull request. In this chapter, you will learn great Git and Android Studio tips and tricks to help in your day-to-day development.

In this chapter, we'll be covering the following recipes:

- The importance of profiling your Android applications
- Quick Android shortcuts to make your development faster
- JetBrains Toolbox and essential plugins to know
- Debugging your code

- How to extract methods and method parameters
- Understanding Git essentials

Technical requirements

The complete source code for this chapter can be found at **https://github.com/PacktPublishing/Modern-Android-13-Development-Cookbook**.

The importance of profiling your Android applications

In Android, **profiling** is the process of analyzing an application's performance to identify its strengths and weaknesses. Profiling your Android applications is crucial for the following reasons:

- It helps you identify performance bottlenecks such as slow code, memory leaks, and excessive CPU usage. This knowledge can help you optimize your code and make your application run more efficiently.
- It improves the user experience. A poorly performing application can lead to user frustration and negative reviews. By profiling your application and optimizing its performance, you can provide a better user experience, leading to increased user engagement and positive reviews.

- It helps save time and money. It is much easier and less expensive to fix performance issues early on than to try to fix them later when they have become more complex.

Hence, in this recipe, we will explore why profiling your Android applications is essential and look at best practices and tips.

Getting ready

For this recipe, we will be looking at how to use the Profiler to profile our Android application. You do not need to create a new project; you can simply use a pre-existing project to follow along.

How to do it...

Follow these steps to get started with using a Profiler in Android:

1. For this chapter, we are using Android Studio Flamingo 2022.2.1 Patch 1. In your Android Studio, go to **View** | **Tool Windows** | **Profiler** and click the Profiler, which will launch it.

NOTE

To be able to see any activity, you need to start your emulator.

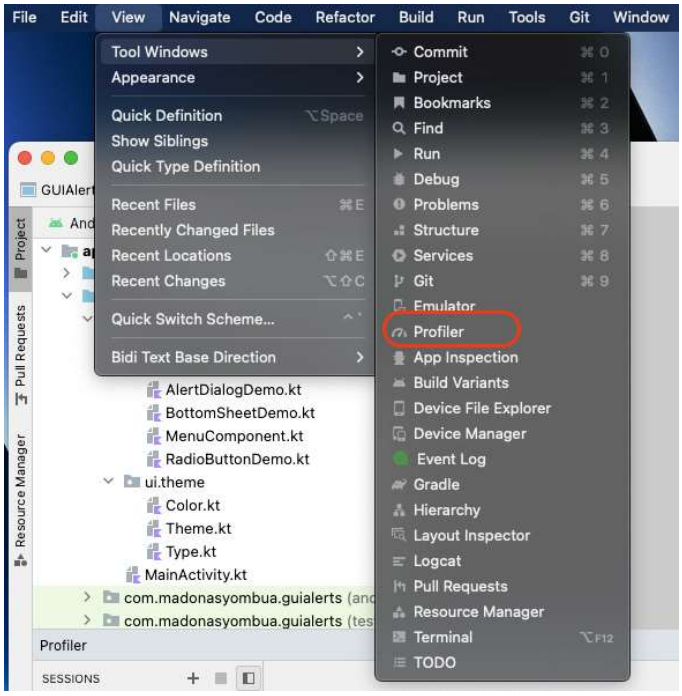

Figure 12.1 – The Profiler in Android Studio

2. You can also navigate to the bottom menu option, near to **App Inspection**; see the green arrow in *Figure 12.2*, which indicates another place you can start a Profiler from. The read arrow indicates the emulator that is attached to visualize the profile.

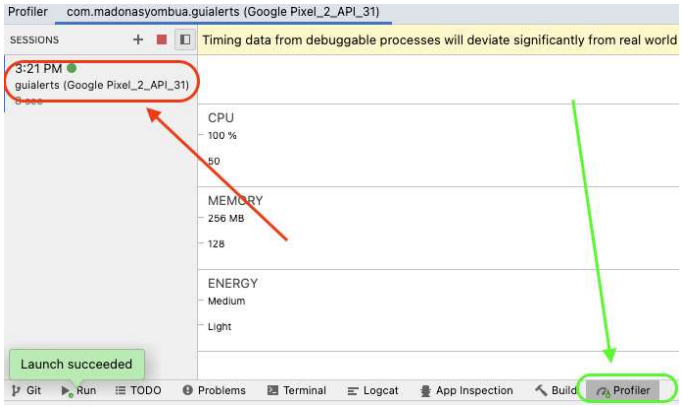

Figure 12.2 – The Profiler started in Android Studio

3. When your Profiler starts running, which means it is attached to your application, you should see **CPU**, **MEMORY**, and **ENERGY**. Depending on your application resources, the

data might vary from what you see in *Figure 12.3*.

Figure 12.3 – The Profiler running

4. You can do a lot, such as simply recording all your method traces, looking at how your resources are utilized, and analyzing the flame chart.

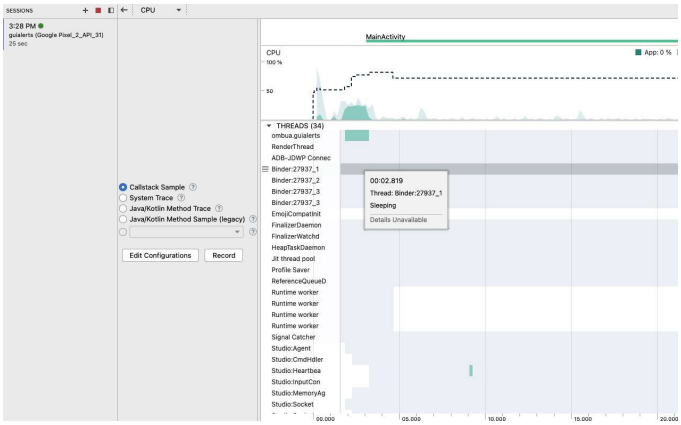

Figure 12.4 – Different ways to utilize the Profiler

5. A CPU flame chart is a type of performance visualization that shows the hierarchical structure of a program's execution over time. It typically includes a timeline at the top of the chart, with function calls represented as rectangles stacked vertically.

Depending on the color, the width of each rectangle represents the duration of the function call, and the rectangle's color represents the CPU usage of that function. The chart allows Android

developers to identify which functions take up the most CPU time quickly and can help inform them where to debug and optimize performance, as shown in *Figure 12.5*.

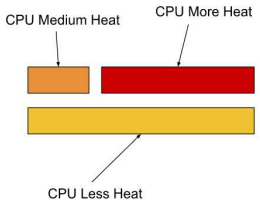

Figure 12.5 – The CPU flame chart

In other words, the application heap is a dedicated, hard, limited memory pool allocated to your app.

NOTE

If your app reaches the heap capacity and tries to allocate any extra memory, you will receive an `OutOfMemoryError` *message.*

6. Finally, a memory leak is a software bug where a program or application repeatedly fails to release memory that it no longer needs, or your garbage collector is not working as expected. This can cause the program to gradually consume more and more memory over time, eventually leading to poor performance or the crashing of your application.

How it works...

An app performs poorly if it responds slowly, has choppy animations, freezes a lot, or consumes a lot of power. Fixing performance problems involves identifying areas where your application does not optimize the use of resources, such as the CPU, memory, graphics, network, or device battery.

Android Studio offers several tools to help developers spot and visualize potential problems:

- The CPU Profiler, which helps track runtime performance issues
- The Memory Profiler, which helps track any memory allocations
- The Network Profiler, which monitors network traffic usage
- The Energy Profiler, which tracks energy usage, which can contribute to battery drain

You can think of profiling in Android through a lens of inspecting, improving, and monitoring your code base. See *Figure 12.6*.

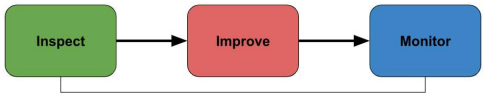

Figure 12.6 – The model for performance source (android.developer.com)

See more...

Learn more about `OutOfMemoryError` by following this link: **https://developer.Android.com/reference/java/lang/OutOfMemoryError**. You can also learn more about profiling by simply using this link: **https://developer.android.com/studio/profile**.

Quick Android shortcuts to make your development faster

Shortcuts can be helpful to developers by making their work quicker and more efficient, allowing them to focus on writing code and solving

problems rather than navigating menus and toolbars. Shortcuts can help automate repetitive tasks such as formatting code, renaming variables, or navigating between files, freeing developers' time and mental energy for more meaningful work.

In addition, when developers use the same shortcuts across different tools and applications, it can help maintain consistency in their workflows and reduce the risk of errors caused by accidentally using the wrong command or tool. Also, for developers with disabilities or physical limitations, using shortcuts can be a more accessible way to interact with software than using a mouse or trackpad.

Getting ready

This isn't really a recipe but a list of useful shortcuts, we will look at widely used shortcuts in Windows and Mac, which are the most popular operating systems used on laptops.

How to do it...

Here are some Android Studio shortcuts in both Mac and Windows operating systems that can help you speed up your workflow:

- Here are some basic navigation shortcuts:
  - **Open class or file**: *Ctrl* + *N* (Windows) or *Cmd* + *O* (Mac)
  - **Find text across project**: *Ctrl* + *Shift* + *F* (Windows) or *Cmd* + *Shift* + *F* (Mac)
  - **Open Recent Files popup**: *Ctrl* + *E* (Windows) or *Cmd* + *E* (Mac)
  - **Search for and execute any action or command**: *Ctrl* + *Shift* + *A* (Windows) or

Cmd + *Shift* + *A* (Mac)

- Code editing shortcuts:
  - **Code completion suggestions**: *Ctrl* + spacebar (Windows and Mac)
  - **Complete current statement**: *Ctrl* + *Shift* + *Enter* (Windows) or *Cmd* + *Shift* + *Enter* (Mac)
  - **Duplicate current line**: *Ctrl* + *D* (Windows) and (Mac) *Cmd* + *D*
  - **Cut current line**: *Ctrl* + *X* (Windows) and (Mac) *Cmd* + *X*
  - **Move current line up or down**: *Ctrl* + *Shift* + up/down (Windows) or *Cmd* + *Shift* + up/down (Mac)
- Refactoring shortcuts:
  - **Extract method from current code block**: *Ctrl* + *Alt* + *M* (Windows) and *Cmd* + *Option* + *M* (Mac)
  - **Extract variable from current code block**: *Ctrl* + *Alt* + *V* (Windows) and *Cmd* + *Option* + *V* (Mac)
  - **Extract field from current code block**: *Ctrl* + *Alt* + *F* (Windows) and *Cmd* + *Option* + *F* (Mac)
  - **Rename class, method, or variable**: *Shift* + *F6* (Windows) and *Fn* + *Shift* +*F6* (Mac)
- Debugging shortcuts:
  - **Step over to the next line of code**: *F8* (Windows and Mac)
  - **Step into the current line of code**: *F7* (Windows and Mac)
  - **Step out of the current method**: *Shift* + *F8* (Windows and Mac)
  - **Toggle breakpoint on the current line of code**: *Ctrl* + *F8* (Windows) or *Cmd* + *F8* (Mac)
- Miscellaneous shortcuts:

- **Run app**: *Ctrl* + *Shift* + *F10* (Windows) or *Cmd* + *Shift* + *F10* (Mac)
- **Debug app**: *Ctrl* + *Shift* + *F9* (Windows) or *Cmd* + *Shift* + *F9* (Mac)

IMPORTANT NOTE

Please note that some shortcuts may differ depending on your specific keyboard layout or operating system preferences. Also, remember that many more Android Studio shortcuts are available, so be sure to explore the Keymap settings to find additional shortcuts that can make your development workflow more efficient.

How it works...

Shortcuts can be a powerful tool for developers looking to streamline their workflows, improve their productivity, and reduce the risk of errors and repetitive strain injuries when developing.

JetBrains Toolbox and essential plugins to know

JetBrains Toolbox is a software management tool that allows developers to manage and install JetBrains IDEs and related tools on their computers. JetBrains is a software company that provides powerful IDEs (namely IntelliJ) for various programming languages, such as Java, Kotlin, Python, Ruby, and JavaScript. In other words, a plugin is simply any class that implements the plugin interface.

Here are some JetBrains Toolbox features and reasons why you should try using it:

- You can easily download and install any JetBrains IDE from Toolbox. It also ensures that you have the latest version of the IDE installed on your computer.
- Toolbox automatically checks for updates and keeps all the installed JetBrains IDEs and plugins up to date, which means if your current Android Studio is not stable, you can revert to a more stable version.
- You can manage your JetBrains licenses and activate/deactivate them from Toolbox.
- Toolbox provides a way to share projects with your team members by creating a shareable link.
- Toolbox integrates with JetBrains services such as JetBrains Account and JetBrains Space.

Getting ready

This isn't really a recipe but a list of useful plugins, we will look at some useful plugins for developers.

How to do it...

Let' go ahead and look at how we can utilize Gradle for our day to day Android Development.

- Gradle is a build automation tool used to build and deploy Android apps. It can help you to manage dependencies, generate APKs, and run tests.
- The ADB plugin provides a graphical user interface for **Android Debug Bridge** (**ADB**), a

command-line tool that can interact with an Android device or emulator.

- Live Templates allows you to insert commonly used code snippets quickly. For example, you can create a live template for a toast message and then simply type in the shortcut and hit the Tab to insert the code. To create a live template, go to **Android Studio** | **Settings** | **Editor** | **Live Templates**.

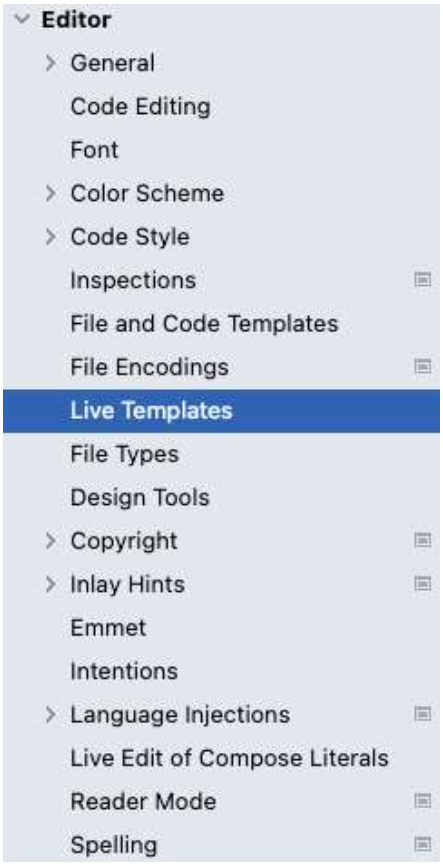

Figure 12.7 – How to access Live Templates

- Android Studio's code completion feature can save you a lot of time. As you type, Android Studio will suggest possible completions for your code. Use the *Tab* key to accept the suggestion.
- A debugger is a powerful tool for finding and fixing bugs in your code. You will learn how to use the debugger to step through your code

and see what's happening at each step in the *Debugging your code* recipe.

- Android Studio's layout editor lets you easily create and modify your app's user interface. You can use the layout editor to drag and drop user interface components onto your layout and easily modify their properties.
- The resource manager allows you to easily manage your app's resources, such as images, strings, and colors. You can use the resource manager to add and modify resources and easily reference them in your code.
- Android Studio supports a variety of plugins that can extend its functionality. You can also easily search for plugins to help you with tasks such as generating code or managing dependencies.
- LeakCanary is a memory leak detection library that can help you identify and fix your app's memory leaks. This helps developers when finding leaks.
- Firebase is a suite of mobile development tools that can be used to add features such as authentication, analytics, and cloud messaging to your app. You can take advantage of this when building your first project as an indie developer.

How it works...

You can easily find Keymap by simply going to **AndroidStudio** | **Setting** | **Keymap** and using the drop-down menu to see what keymaps are available, as shown in *Figure 12.8*.

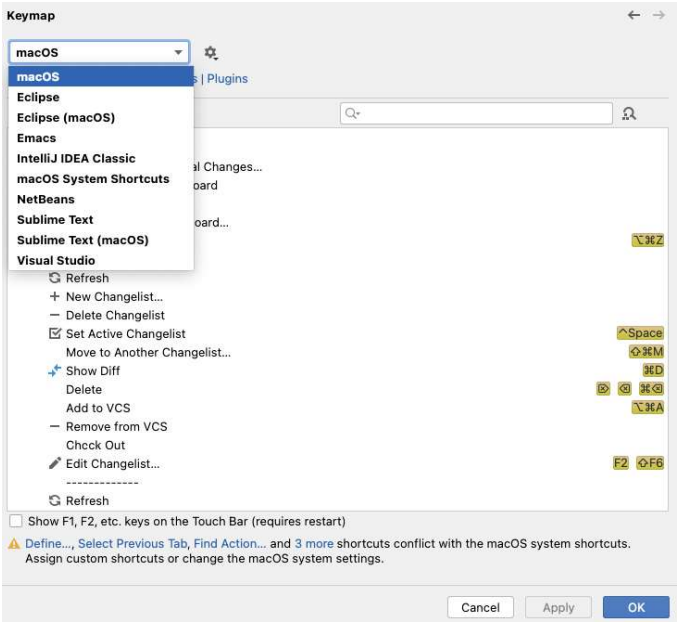

Figure 12.8 – Keymap

IMPORTANT NOTE

Find out what Android Studio's latest release is and what features are offered by following this link: **https://developer.android.com/studio/releases**.

Debugging your code

As an Android developer, debugging is an essential part of the software development process because it helps identify and fix errors in your code. When debugging, you can quickly identify and fix errors or bugs in your code that can cause your application to crash, behave unexpectedly, or produce incorrect results.

In this recipe, we will explore how you can easily add a breakpoint and debug your code.

Getting ready

To get started with this recipe, you need to have a project open and run the project on your emulator. You do not need to create a new project and can use the **`GUIAlert`** project.

How to do it...

We will be trying to debug our code and ensure when we click the items in the menu, we select the correct item. For instance, if we select item 2, when we evaluate the item, we should see the result being **`2`**:

1. First, you need to ensure your app is running; then, click the icon shown in *Figure 12.9*.

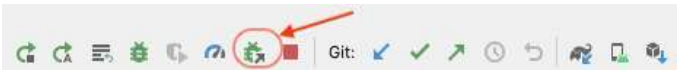

Figure 12.9 – The debugger icon

2. When you click the icon shown in *Figure 12.9*, a pop-up screen will appear, which means you will attach your running application to the debugger.

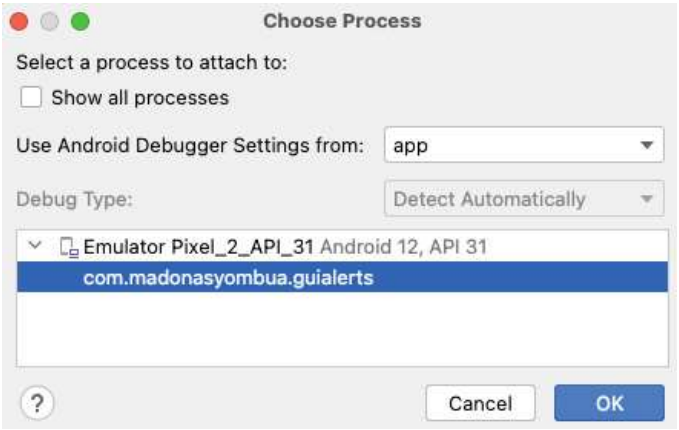

Figure 12.10 – The options on the package to attach the debugger

3. Now, go back to the code base and add breakpoints. You add breakpoints by clicking on

the sidebar on the line number where you wish to test your logic.

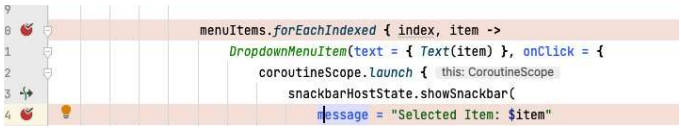

Figure 12.11 – Breakpoints

4. If your app is running when you click on the item, say option 1, the debugger will show an active state, which means the lines we put the breakpoints on were hit. Then, a pop-up window will appear with controls.

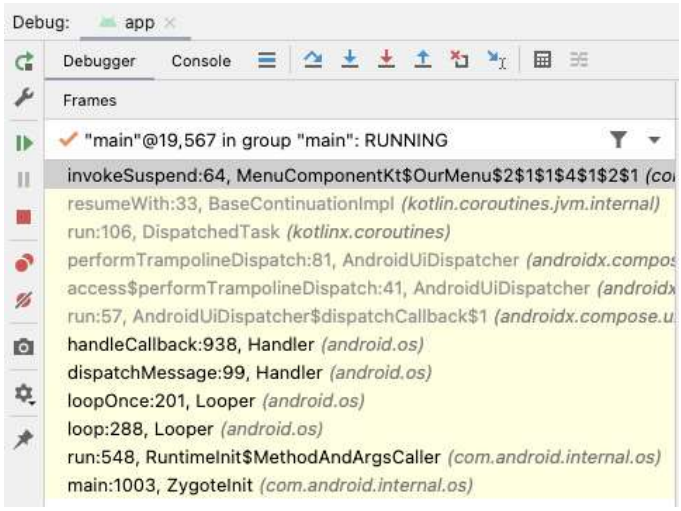

Figure 12.12 – Debug being active

5. You can use the green button on the left to run and the red square button to stop. You can also use **Step Over**, **Step Into**, **Force Step Into**, **Step Out**, **Drop Frame**, **Run to Cursor**, and **Evaluate Expression...**. We will use **Evaluate Expression...** in our example.

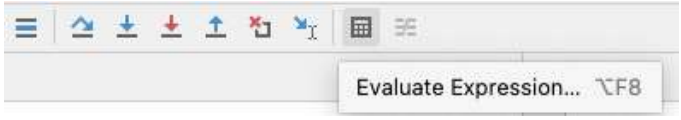

Figure 12.13 – Debug button steps

6. Sometimes, you might have extra breakpoints that might slow down the process. In this

case, you can use the option pointed out by the red arrow in *Figure 12.14* to see all your breakpoints.

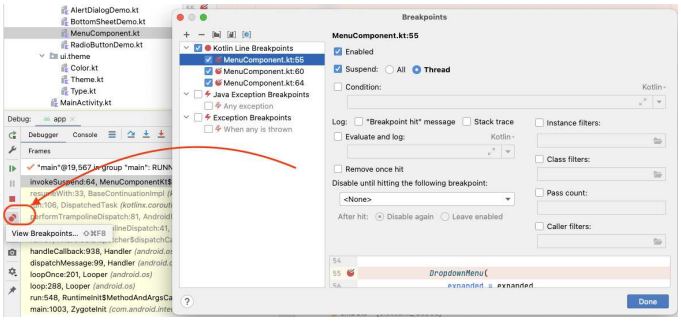

Figure 12.14 – Tracking all your breakpoints

7. Finally, when the app is still in debug mode, open the **Evaluate** section, go back to step 5 of this recipe, and enter `Item`. Based on the current item, you should see the number displayed.

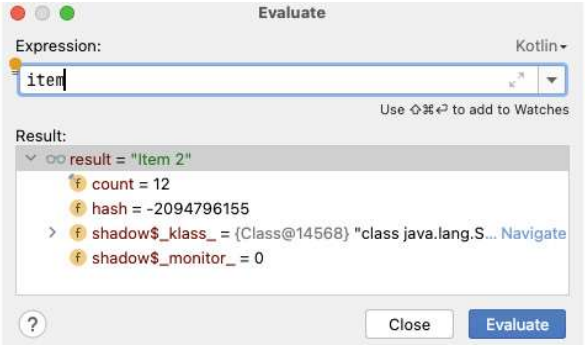

Figure 12.15 – Item is currently selected when we evaluate the breakpoint

How it works...

Android Studio includes a powerful debugger for us developers to use. To debug an application using Android Studio, you need to first build and deploy the app on the device or emulator, then attach the debugger to the running process. This is also a skill that needs to be learned and practiced becoming good at it. Hence, knowing how

you can debug your application, either using logs or breakpoints, comes in handy.

IMPORTANT NOTE

There is more to learn about debugging, and more than one recipe is needed to cover this topic. Follow this link to learn more: **https://developer.android.com/studio/debug**.

How to extract methods and method parameters

Extracting methods and method parameters can add additional imports to your code. This happens because when you extract a method or a parameter, the code that used to be inside the method or parameter is moved to a separate method. If this code relies on other classes or methods that are not already imported into your code, the extraction process may automatically add the necessary import statements to your file.

For example, suppose you have a Kotlin class that contains a method that performs some calculations and returns a result. This method relies on a helper class defined in another package, and you still need to import this class into your code. If you decide to extract the method to a particular method in the same or a different class, the extraction process may add an `import` statement for the helper class so that the code inside the extracted method can reference the helper class.

Similarly, when you extract a method parameter, the extraction process may need to include adding import statements to ensure that any classes or interfaces that are used in the parameter type are correctly resolved.

Getting ready

You do not need to create any project to follow this recipe.

How to do it...

To extract methods and method parameters in Android, you can follow these steps:

1. Open the Kotlin file where you want to extract the methods and parameters.
2. Identify the class containing the methods and parameters you want to extract.
3. Place your cursor inside the class declaration, and right-click to open the context menu.
4. Select the **Refactor** option from the context menu, and then select **Extract** from the submenu.
5. In the **Extract** submenu, you will see options to extract a method or parameter. Select the option that matches the element you want to extract.
6. Follow the prompts in the **Extract** wizard to configure the extraction process. You may need to provide a name for the extracted element, specify the element's scope, or configure other settings depending on the element you are extracting.
7. Once configured in the extraction process, click **Finish** to extract the element from your code.

How it works...

Adding imports during method or parameter extraction is a normal part of the refactoring process and helps ensure that your code remains well organized and easy to maintain.

Understanding Git essentials

This recipe is meant to help any new developers that might have stumbled upon this book. **Git** is a popular version control system that allows developers to manage and track changes to their code base.

Here are some essential concepts to understand:

- A **repository** is a collection of files and folders that Git is tracking. It's also known as a **repo**. This is the most common term.
- A **commit** is a snapshot of the changes made to a repository. Each commit has a unique identifier containing information about the changes made, such as the author, the date, and a message describing the changes.
- A **branch** is a separate line of development that allows developers to work on different features or versions of a project simultaneously. It's like a parallel universe of the repository.
- When developing, **merging** your work refers to combining changes from one branch into another. It's typically used when a feature is complete and ready to be integrated into the main branch.

- A **pull request** is a GitHub feature that allows developers to propose changes to a repository and request that they be merged into the main branch. It includes a description of the changes and any supporting documentation or tests.
- **Cloning** is creating a copy of a repository on your local machine.
- **Pushing** is the process of sending changes from your local machine to a remote repository, such as GitHub or GitLab.
- **Pulling** is the process of downloading changes from a remote repository to your local machine.

By understanding these essential concepts, you can effectively use Git to manage your code base and collaborate with other developers.

Getting ready

We will not follow a recipe here but look at what Git commands you can utilize to make collaboration easier.

How to do it...

Here are some of the most commonly used Git commands:

- To initialize a new Git repository in the current directory. You can simply do the following:

```
$ git init
```

- When you want to add changes to the staging area, you can simply use **`git add`**:

```
$ git add file.txt
```

- When committing changes to the repository, simply use the following:

```
$ git commit -m "message"
```

- Most important, when you start collaborating, is being able to clone the project; you can simply run the following:

```
$ git clone git@github.com:PacktPublishing/Modern-Android-13-Development-Cookbook.git
```

- When you want to pull changes from a remote repository to the local repository, simply use the following:

```
$ git pull origin main
```

- You can also push changes from the local repository to a remote repository by using the following:

```
$ git push origin main
```

- List all the local branches by using **`git branch`**:

```
$ git branch
```

- The following command switches to a different branch:

```
$ git checkout branch_name
```

- Check out a new branch with the following:

```
$ git checkout -b branch_name
```

- Merge changes from one branch into another with the following. Note you can also use **`rebase`**; this is based on the organization's preference:

```
$ git merge branch_name
```

These are just a few of the most commonly used Git commands. Many more Git commands and options are available, so it’s worth exploring the Git documentation to learn more.

How it works...

Git is a distributed version control system allowing users to track code changes over time. Here's a high-level overview of how Git works.

Git doesn't just store the changes you make to your code; it actually stores snapshots of your entire project at different times. Each snapshot represents the state of the project at a specific moment. It stores your code in a tree-like structure, with each project snapshot represented by a commit object.

Each commit object points to the snapshot of the project that it represents and the commit objects that came before it. It also uses a unique pointer called **HEAD** to keep track of the current branch and the most recent commit on that branch.

When you make a new commit, Git updates the HEAD pointer to point to the new commit. In addition, each commit in Git is identified by a unique hash value, which is a 40-character string of letters and numbers. This hash value is generated based on the contents of the commit and the hash values of any previous commits that it points to.

Because Git stores snapshots of your project locally on your computer, you can work offline and still make commits to your project. When you're ready to share your changes, you can push them to a remote repository.

These are just a few of the key concepts behind how Git works. Git is a powerful and flexible tool with many advanced features, so it's worth learning more about how it works.