|  |  |  |
| --- | --- | --- |
|  | **2, LEBUH ACHEH, GEORGE TOWN**  **10300 GEORGE TOWN**  **PULAU PINANG**  **INFORMATION SHEET** | |
| **PROGRAM’S CODE & NAME** | J620-002-4:2020 FRONT-END SOFTWARE DEVELOPMENT | |
| **LEVEL** | FOUR (4) | |
| **COMPETENCY UNIT NO. AND TITLE** | J620-002-4:2020-C04 MOBILE APPLICATION WITH THIRD PARTY API DEVELOPMENT | |
| **WORK ACTIVITIES NO. AND STATEMENT** | 1. CREATE MOBILE APP DESIGN MOCK-UP ELEMENTS. 2. PLAN MOBILE APP DESIGN STRUCTURE. 3. **TRANSFORM MOCK-UP TO MOBILE APP.** 4. INTEGRATE MOBILE APP WITH DATA SOURCE. 5. VERIFY SUCCESSFUL API INTEGRATION 6. VERIFY DEVELOPED MOBILE APP. 7. VERIFY MOBILE APP ACCESSIBLE GLOBALLY. | |
| **CODE NO.** | J620-002-4:2020-C04/IS(4/15) | Page: 1 of17 |

**TITLE**:

**INTEGRATED DEVELOPMENT ENVIRONMENT (IDE)**

**PURPOSE**:

This information sheet is intended to provide insight and knowledge to trainees with regards to the fundamentals of Integrated Development Environment (IDE).

**INFORMATION:**

This information sheet provides useful notes and explanations on fundamental concepts of Integrated Development Environment (IDE).

# **INTEGRATED DEVELOPMENT ENVIRONMENT**

An integrated development environment (IDE) is software for building applications that combines common developer tools into a single graphical user interface (GUI). An IDE typically consists of:

* Source code editor: A text editor that can assist in writing software code with features such as syntax highlighting with visual cues, providing language specific auto-completion, and checking for bugs as code is being written.
* Local build automation: Utilities that automate simple, repeatable tasks as part of creating a local build of the software for use by the developer, like compiling computer source code into binary code, packaging binary code, and running automated tests.
* Debugger: A program for testing other programs that can graphically display the location of a bug in the original code.

## The important of IDEs

An IDE allows developers to start programming new applications quickly because multiple utilities do not need to be manually configured and integrated as part of the setup process. Developers also do not need to spend hours individually learning how to use different tools when every utility is represented in the same workbench. This can be especially useful for onboarding new developers who can rely on an IDE to get up to speed on a team’s standard tools and workflows. In fact, most features of IDEs are meant to save time, like intelligent code completion and automated code generation, which removes the need to type out full character sequences.

Other common IDE features are meant to help developers organize their workflow and solve problems. IDEs parse code as it is written, so bugs caused by human error are identified in real-time. Because utilities are represented by a single GUI, developers can execute actions without switching between applications. Syntax highlighting is also common in most IDEs, which uses visual cues to distinguish grammar in the text editor. Some IDEs additionally include class and object browsers, as well as class hierarchy diagrams for certain languages.

It is possible to develop applications without an IDE, or for each developer to essentially build their own IDE by manually integrating various utilities with a lightweight text editor like Vim or Emacs. For some developers, the benefit of this approach is the ultra-customization and control it offers. In an enterprise context, though, the time saved, environment standardization, and automation features of modern IDEs usually outweigh other considerations.

Today, most enterprise development teams opt for a pre-configured IDE that is best suited to their specific use case, so the question is not whether to adopt an IDE, but rather which IDE to select.

## Popular kinds of IDEs

There are many different technical and business use cases for IDEs, which likewise means there are many proprietary and open-source IDE options on the market. Typically, the most important differentiating characteristics between IDEs are:

1. The number of supported languages: Some IDEs are dedicated to one language, and so are a better match for a specific programming paradigm. IntelliJ, for instance, is known primarily as a Java IDE. Other IDEs have a broad array of supported languages all in one, like the Eclipse IDE which supports Java, XML, Python, and others.
2. Supported operating system(s): A developer’s operating system will constrain which IDEs are viable (unless an IDE is cloud-based), and if the application being developed is intended for an end user with a specific operating system (like Android or iOS), this may be an additional constraint.
3. Automation features: Even though most IDEs include the 3 key features of a text editor, build automation, and debugger, many include support for additional features like refactoring, code search, and continuous integration and continuous deployment (CI/CD) tools.
4. Impact on system performance: An IDE’s memory footprint may be important to consider if a developer wants to run other memory-intensive applications concurrently.
5. Plugins and extensions: Some IDEs include the ability to customize workflows to match a developer’s needs and preferences.

## Mobile development IDEs

Nearly every industry has been affected by the rising popularity of apps designed for smartphones and tablets, leading many companies to develop mobile apps in addition to traditional web apps. One of the key factors in mobile application development is platform choice. For instance, if a new application is intended for use on iOS, Android, and a web page, it may be best to start with an IDE that provides cross-platform support for multiple operating systems.

## Cloud IDEs

IDEs that are provided as a cloud-based Software-as-a-Service (SaaS) provide a number of unique benefits compared to local development environments. For one, as with any SaaS offering, there is no need to download software and configure local environments and dependencies, so developers can start contributing to projects quickly. This also provides a level of standardization across team members’ environments, which can mitigate the common “this works on my machine, why doesn’t it work on yours” problem. Additionally, since the development environment is centrally managed, no code resides on an individual developer’s computer, which can help with intellectual property and security concerns.

The impact of processes on local machines is also different. Processes like running builds and testing suites are typically compute-intensive, which means developers are probably unable to continue using workstations while a process is running. A SaaS IDE can dispatch long-running jobs without monopolizing the compute resources of a local machine. Cloud IDEs are also typically platform agnostic, allowing connection to different cloud vendors.

# **ANDROID STUDIO**

Android Studio is the official Integrated Development Environment (IDE) for Android app development, based on IntelliJ IDEA. On top of IntelliJ's powerful code editor and developer tools, Android Studio offers even more features that enhance your productivity when building Android apps, such as:

* A flexible Gradle-based build system
* A fast and feature-rich emulator
* A unified environment where you can develop for all Android devices.
* Apply Changes to push code and resource changes to your running app without restarting your app.
* Code templates and GitHub integration to help you build common app features and import sample code.
* Extensive testing tools and frameworks
* Lint tools to catch performance, usability, version compatibility, and other problems.
* C++ and NDK support
* Built-in support for Google Cloud Platform, making it easy to integrate Google Cloud Messaging and App Engine

## Project structure

Each project in Android Studio contains one or more modules with source code files and resource files. Types of modules include:

1. Android app modules
2. Library modules
3. Google App Engine modules

By default, Android Studio displays your project files in the Android project view, as shown in figure 1. This view is organized by modules to provide quick access to your project's key source files.

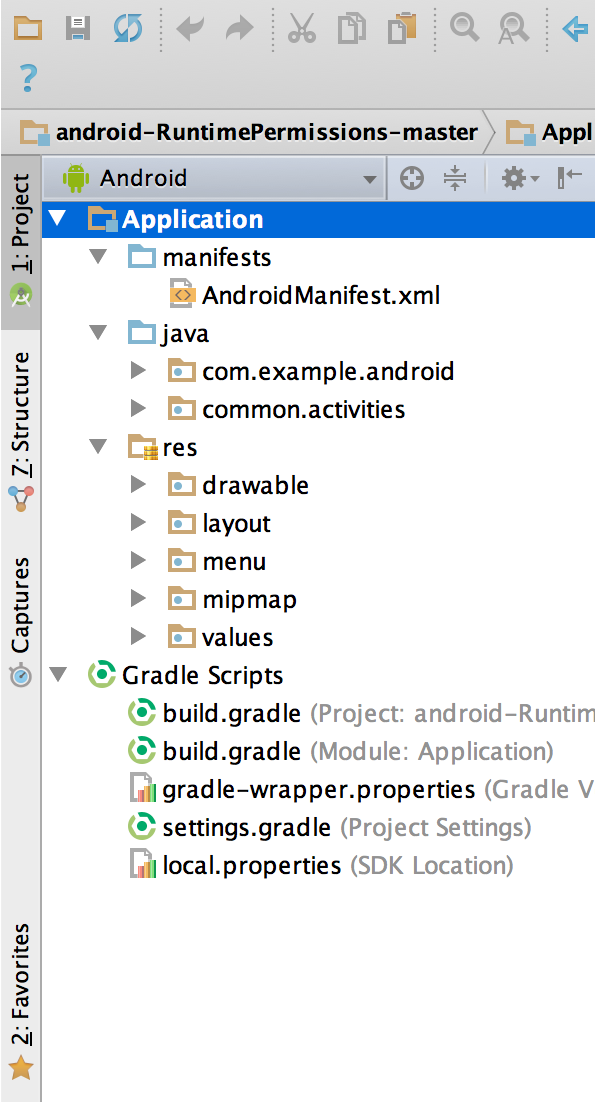


Figure 1: The Project Files In Android View

All the build files are visible at the top level under Gradle Scripts and each app module contains the following folders:

1. manifests: Contains the AndroidManifest.xml file.
2. java: Contains the Java source code files, including JUnit test code.
3. res: Contains all non-code resources, such as XML layouts, UI strings, and bitmap images.

The Android project structure on disk differs from this flattened representation. To see the actual file structure of the project, select Project from the Project dropdown (in figure 1, it is showing as Android).

You can also customize the view of the project files to focus on specific aspects of your app development. For example, selecting the Problems view of your project displays links to the source files containing any recognized coding and syntax errors, such as a missing XML element closing tag in a layout file.

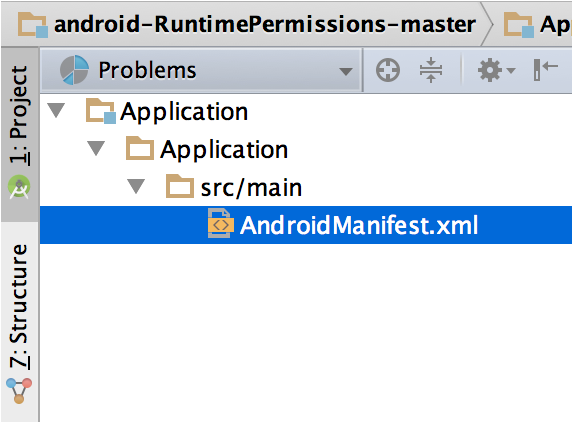


Figure 2: The Project Files In Problems View, Showing A Layout File With A Problem

## The user interface

The Android Studio main window is made up of several logical areas identified in figure 3.

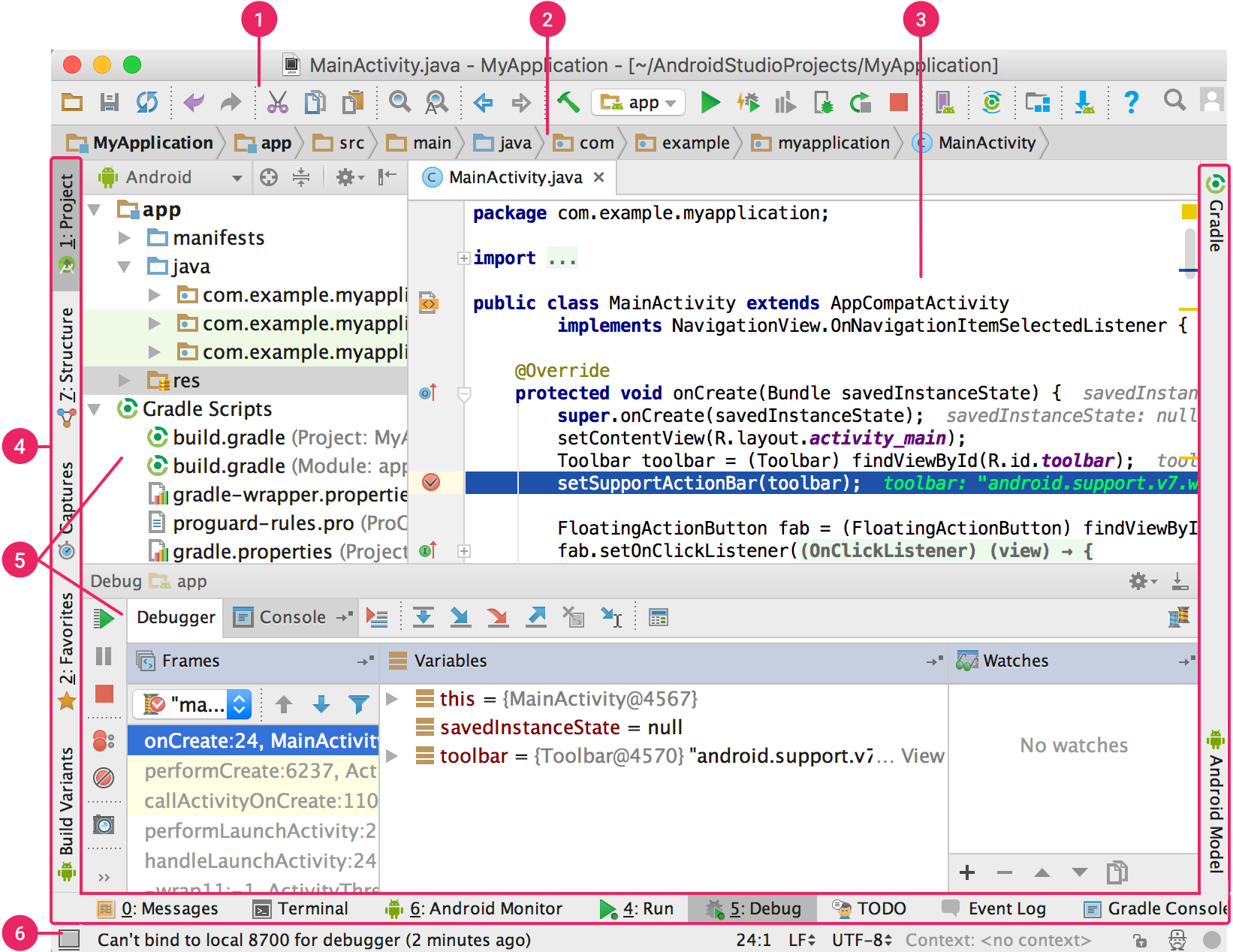


Figure 3: The Android Studio Main Window

1. The toolbar lets you carry out a wide range of actions, including running your app and launching Android tools.
2. The navigation bar helps you navigate through your project and open files for editing. It provides a more compact view of the structure visible in the Project window.
3. The editor window is where you create and modify code. Depending on the current file type, the editor can change. For example, when viewing a layout file, the editor displays the Layout Editor.
4. The tool window bar runs around the outside of the IDE window and contains the buttons that allow you to expand or collapse individual tool windows.
5. The tool windows give you access to specific tasks like project management, search, version control, and more. You can expand them and collapse them.
6. The status bar displays the status of your project and the IDE itself, as well as any warnings or messages.

You can organize the main window to give yourself more screen space by hiding or moving toolbars and tool windows. You can also use keyboard shortcuts to access most IDE features.

At any time, you can search across your source code, databases, actions, elements of the user interface, and so on, by double-pressing the Shift key, or clicking the magnifying glass in the upper right-hand corner of the Android Studio window. This can be very useful if, for example, you are trying to locate a particular IDE action that you have forgotten how to trigger.

## Gradle build system

Android Studio uses Gradle as the foundation of the build system, with more Android-specific capabilities provided by the Android plugin for Gradle. This build system runs as an integrated tool from the Android Studio menu, and independently from the command line. You can use the features of the build system to do the following:

1. Customize, configure, and extend the build process.
2. Create multiple APKs for your app, with different features using the same project and modules.
3. Reuse code and resources across source sets.

By employing the flexibility of Gradle, you can achieve all of this without modifying your app's core source files. Android Studio build files are named build.gradle. They are plain text files that use Groovy syntax to configure the build with elements provided by the Android plugin for Gradle. Each project has one top-level build file for the entire project and separate module-level build files for each module. When you import an existing project, Android Studio automatically generates the necessary build files.

The build system can help you create different versions of the same application from a single project. This is useful when you have both a free version and a paid version of your app, or if you want to distribute multiple APKs for different device configurations on Google Play.

Multiple APK support allows you to efficiently create multiple APKs based on screen density or ABI. For example, you can create separate APKs of an app for the hdpi and mdpi screen densities, while still considering them a single variant and allowing them to share test APK, javac, dx, and ProGuard settings.

Dependencies for your project are specified by name in the build.gradle file. Gradle takes care of finding your dependencies and making them available in your build. You can declare module dependencies, remote binary dependencies, and local binary dependencies in your build.gradle file. Android Studio configures projects to use the Maven Central Repository by default.

## Debugging and profile tools

Android Studio assists you in debugging and improving the performance of your code, including inline debugging and performance analysis tools.

1. Inline debugging

Use inline debugging to enhance your code walk-throughs in the debugger view with inline verification of references, expressions, and variable values. Inline debug information includes:

* Inline variable values
* Referring objects that reference a selected object
* Method return values
* Lambda and operator expressions
* Tooltip values

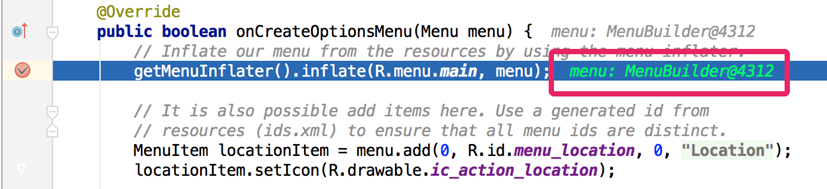


Figure 6. An inline variable value.

1. Performance profilers

Android Studio provides performance profilers so you can more easily track your app’s memory and CPU usage, find deallocated objects, locate memory leaks, optimize graphics performance, and analyse network requests.

1. Heap dump

When you’re profiling memory usage in Android Studio, you can simultaneously initiate garbage collection and dump the Java heap to a heap snapshot in an Android-specific HPROF binary format file. The HPROF viewer displays classes, instances of each class, and a reference tree to help you track memory usage and find memory leaks.

1. Memory Profiler

You can use Memory Profiler to track memory allocation and watch where objects are being allocated when you perform certain actions. Knowing these allocations enables you to optimize your app’s performance and memory use by adjusting the method calls related to those actions.

1. Code inspections

Whenever you compile your program, Android Studio automatically runs configured Lint and other IDE inspections to help you easily identify and correct problems with the structural quality of your code.

The Lint tool checks your Android project source files for potential bugs and optimization improvements for correctness, security, performance, usability, accessibility, and internationalization.

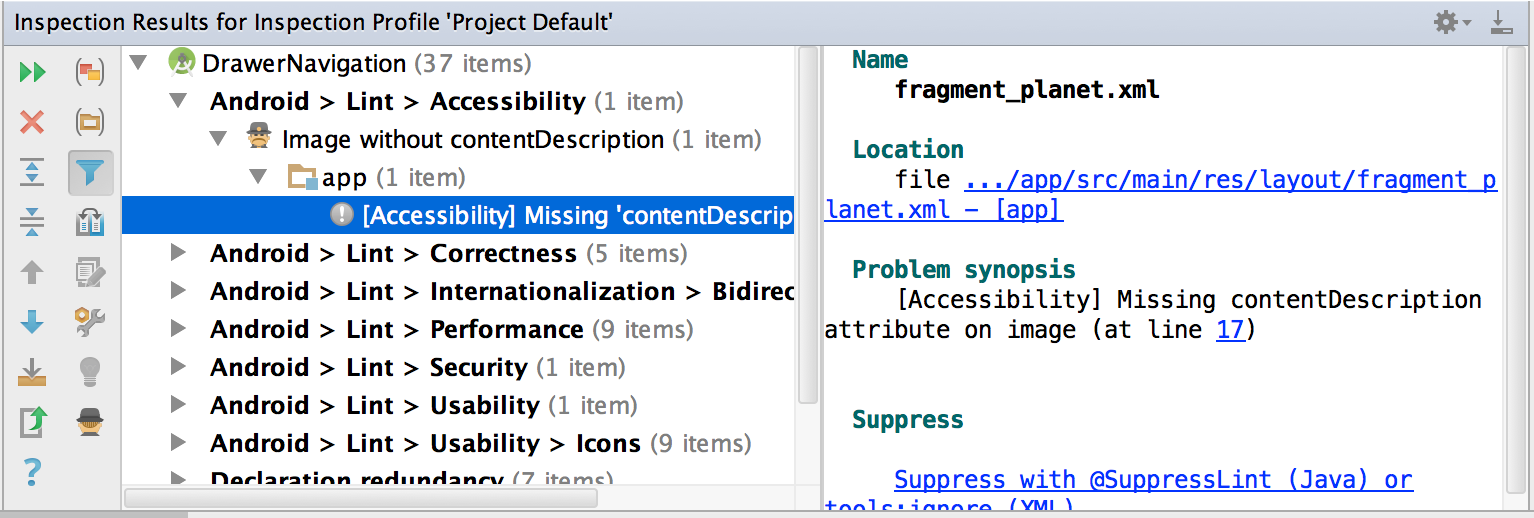


Figure 4: The results of a Lint inspection in Android Studio

In addition to Lint checks, Android Studio also performs IntelliJ code inspections and validates annotations to streamline your coding workflow.

## Configuring Android Studio

Android Studio provides wizards and templates that verify your system requirements, such as the Java Development Kit (JDK) and available RAM, and configure default settings, such as an optimized default Android Virtual Device (AVD) emulation and updated system images. This document describes additional configuration settings you may want to use to customize your use of Android Studio.

Android Studio provides access to two configuration files through the Help menu:

1. studio.vmoptions: Customize options for Studio's Java Virtual Machine (JVM), such as heap size and cache size. Note that on Linux machines this file may be named studio64.vmoptions, depending on your version of Android Studio.
2. idea.properties: Customize Android Studio properties, such as the plugins folder path or maximum supported file size.

The studio.vmoptions file allows you to customize options for Android Studio's JVM. To improve Studio's performance, the most common option to adjust is the maximum heap size, but you can also use the studio.vmoptions file to override other default settings such as initial heap size, cache size, and Java garbage collection switches.

To create a new studio.vmoptions file or to open your existing one, use the following steps:

1. Click Help > Edit Custom VM Options. If you have never edited VM options for Android Studio before, the IDE prompts you to create a new studio.vmoptions file. Click Yes to create the file.
2. The studio.vmoptions file opens in the editor window of Android Studio. Edit the file to add your own customized VM options.

The studio.vmoptions file you create gets added to the default studio.vmoptions file, located in the bin/ directory inside your Android Studio installation folder.

Note that you should never directly edit the studio.vmoptions file found inside the Android Studio program folder. While you can access the file to view Studio's default VM options, editing only your own studio.vmoptions file ensures that you don't override important default settings for Android Studio. Therefore, in your studio.vmoptions file, override only the attributes you care about and allow Android Studio to continue using default values for any attributes you have not changed.

The idea.properties file allows you to customize IDE properties for Android Studio, such as the path to user installed plugins and the maximum file size supported by the IDE. The idea.properties file gets merged with the default properties for the IDE so you can specify just the override properties.

To create a new idea.properties file or to open your existing file, use the following steps:

1. Click Help > Edit Custom Properties. If you have never edited the IDE properties before, Android Studio prompts you to create a new idea.properties file. Click Yes to create the file.
2. The idea.properties file opens in the editor window of Android Studio. Edit the file to add your own customized IDE properties.

If you are running Android Studio on a machine with less than the recommended specifications, you can customize the IDE to improve performance on your machine, as follows:

1. Reduce the maximum heap size available to Android Studio: Reduce the maximum heap size for Android Studio to 512Mb.
2. Update Gradle and the Android plugin for Gradle: Update to the latest versions of Gradle and the Android plugin for Gradle to ensure you are taking advantage of the latest improvements for performance.
3. Enable Power Save Mode: Enabling Power Save Mode turns off a number of memory- and battery-intensive background operations, including error highlighting and on-the-fly inspections, auto popup code completion, and automatic incremental background compilation. To turn on Power Save Mode, click File > Power Save Mode.
4. Disable unnecessary lint checks: To change which lint checks Android Studio runs on your code, do the following:
5. Debug on a physical device: Debugging on an emulator uses more memory than debugging on a physical device, so you can improve overall performance for Android Studio by debugging on a physical device.
6. Click File > Settings (on macOS, Android Studio > Preferences) to open the Settings dialog.
7. In the left pane, expand the Editor section and click Inspections.
8. Click the checkboxes to select or deselect lint checks as appropriate for your project.
9. Click Apply or OK to save your changes.
10. Include only necessary Google Play services as dependencies: Including Google Play Services as dependencies in your project increases the amount of memory necessary. Only include necessary dependencies to improve memory usage and performance. For more information, see Add Google Play Services to Your Project.
11. Reduce the maximum heap size available for Gradle: Gradle's default maximum heap size is 1,536 MB. Reduce the value by overriding the org.gradle.jvmargs property in the gradle.properties file.
12. Do not enable parallel compilation: Android Studio can compile independent modules in parallel, but if you have a low-memory system you should not turn on this feature. To check this setting, do the following:
13. Click File > Settings (on macOS, Android Studio > Preferences) to open the Settings dialog.
14. In the left pane, expand Build, Execution, Deployment and then click Compiler.
15. Ensure that the Compile independent modules in parallel option is unchecked.
16. If you have made a change, click Apply or OK for your change to take effect.

## Updating the IDE and SDK Tools

Once you install Android Studio, it's easy to keep the Android Studio IDE and Android SDK tools up to date with automatic updates and the Android SDK Manager. Android Studio notifies you with a small bubble dialog when an update is available for the IDE, but you can manually check for updates by clicking Help > Check for Update (on Mac, Android Studio > Check for Updates).

Updates for Android Studio are available from the following release channels:

1. Canary channel: These are bleeding-edge releases, updated roughly weekly, and available for download at developer.android.com/studio/preview. In addition to receiving canary versions of Android Studio, you will also receive preview versions of other SDK tools, including the Android Emulator. Although these builds are subject to more bugs, they do get tested and we want to offer them so you can try new features and provide feedback. This channel is not recommended for production development.
2. Dev channel: These are hand-picked canary builds that survived a full round of internal testing.
3. Beta channel: These are release candidates based on stable canary builds, released to get feedback before going into the stable channel.
4. Stable channel: The official stable release that is available for download at developer.android.com/studio.

If you would like to try one of the preview channels (Canary, Dev, or Beta) while still using the Stable build for your production Android projects, you can install both side by side.

To change the update channel for an existing install, proceed as follows:

1. Open the Preferences window by clicking File > Settings (on Mac, Android Studio > Preferences).
2. In the left panel, click Appearance & Behaviour > System Settings > Updates.
3. Be sure that Automatically check for updates is checked, then select a channel from the drop-down list (see figure 1).
4. Click Apply or OK.

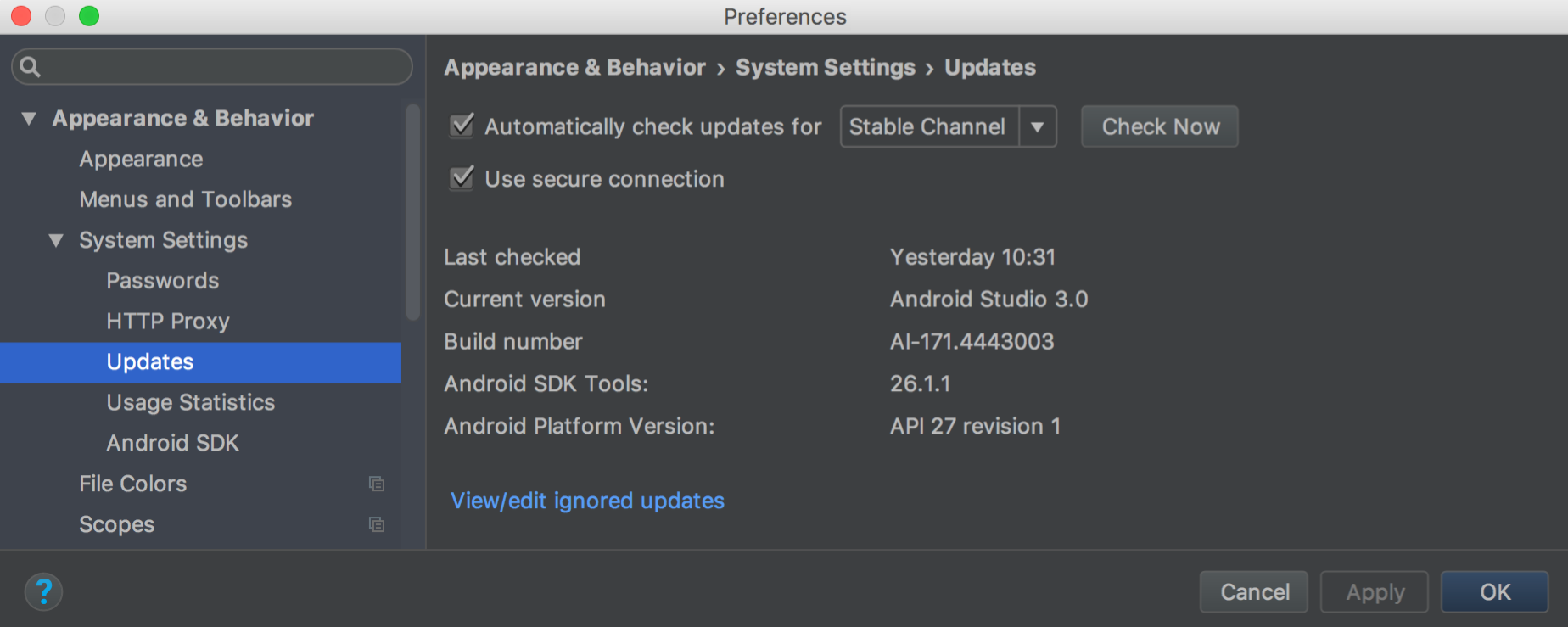


Figure 5: The Android Studio Updates Preferences

**QUESTIONS:**

1. An Integrated Development Environment (IDE) typically consists of:

**Answer:**

* 1. **Code editor.**
  2. **Local build tools.**
  3. **Debugger.**

**REFERENCE:**

1. <https://www.redhat.com/en/topics/middleware/what-is-ide>
2. <https://developer.android.com/studio/intro>
3. <https://developer.android.com/studio/intro/studio-config>