Amper

CS 346 Application Development

https://student.cs.uwaterloo.ca/~cs346

Contents

Introduction . . . 2
- Build Systems . . . 3
- What is Amper? . . . 4

Getting Started . . . 5
- Creating a Project . . . 6
- Project Structure . . . 9
- Running your project . . . 11
- Build tasks . . . 12

Project Templates . . . 13
- JVM Console Application . . . 14
- JVM GUI Application . . . 15
- Android Application . . . 16
- iOS Application . . . 17
- Compose Multiplatform . . . 18

Bibliography . . . 19

Introduction

Build Systems

A build system is a system that manages the process of delivering software. This includes compilation, linking, testing, packaging and other required steps.

- e.g., Ant/Maven for Java; Cargo for Rust; Cmake/Scons/Bazel for C++.

Characteristics of a useful build system:

- It provides consistency in builds and build results.
- It is expressive so that you can define any custom tasks e.g., zip a file.
- You can automate the build process to avoid user errors.
- It integrates with other systems so that you can delegate responsibility e.g., remote test under a different OS.

What is Amper?

Amper is a **new** build system for Kotlin, intended as a lightweight replacement for Gradle. [1]

- It's cross-platform and programming language agnostic.
- It's open source but under active development.
- JetBrains is designing it as a simpler and less error-prone build system than Gradle.

> **Caution**
>
> Amper is still a pre-release product! If you decide to use it, make sure to test it early in your project. JetBrains maintains a Amper channel on Slack that can be helpful [2].

Getting Started

Creating a Project

To setup your code to build with Amper, you need to do the following in IntelliJ:

1. Install the Amper plugin: go to Settings, and make sure to add `Amper` from the Marketplace. Restart IntelliJ if needed.

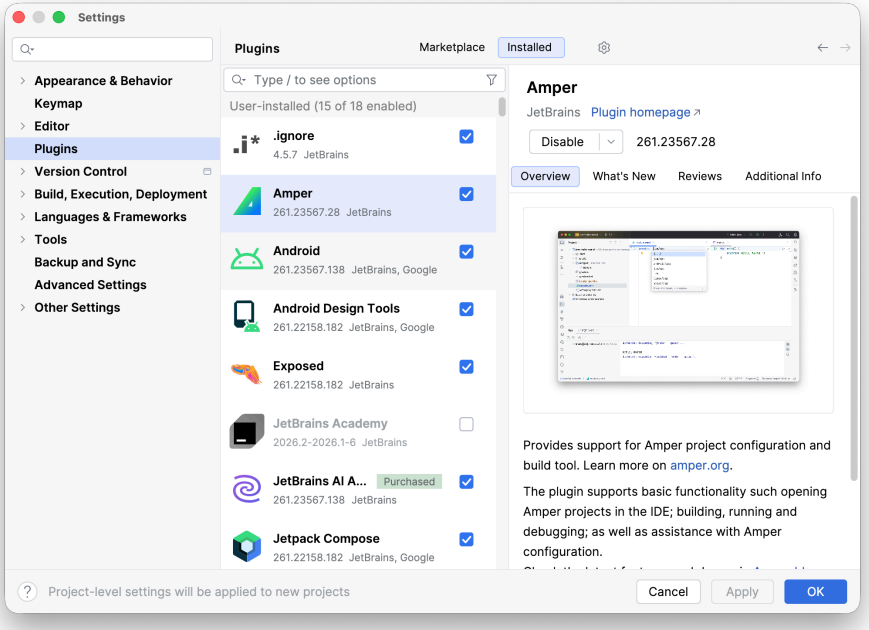

Figure 1: The plugin will update your New-Project options to let you choose Amper as your build system.

Creating a Project

2. Create a project: this is a generated folder structure containing configuration files describing how to build your source code.

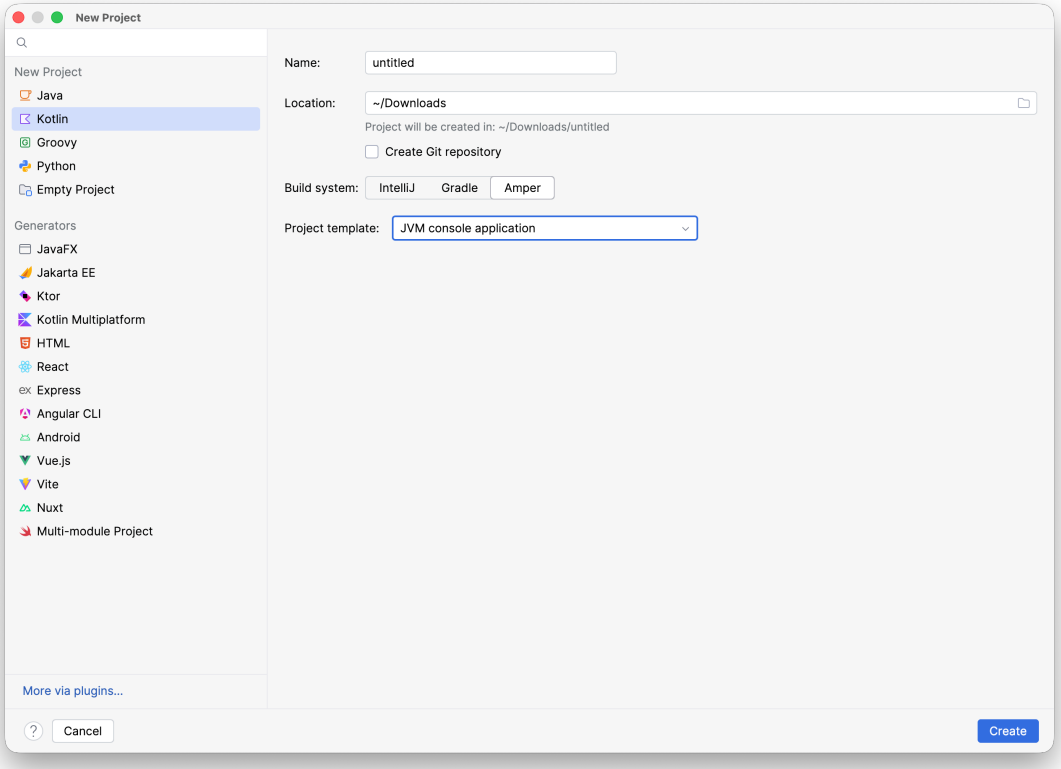

Figure 2: As long as the Amper plugin is installed, you should be able to create an Amper project from the Kotlin project menu.

Creating a Project

Note that your have your choice of `Project template` when creating an Amper project. The following types of projects are useful in this course:

Standalone projects
- Android application (Jetpack Compose)
- iOS Application (Compose Multiplatform)
- JVM GUI application (Compose Multiplatform)
- JVM console

Multi-target projects
- Compose multiplatform

> **Tip**
>
> We will choose `JVM console` for the following slides. Later sections will explore more advanced configuration options.

Project Structure

Compared to Gradle, Amper is very simple to configure and get working. The basic project structure for a `JVM console` project looks something like this.

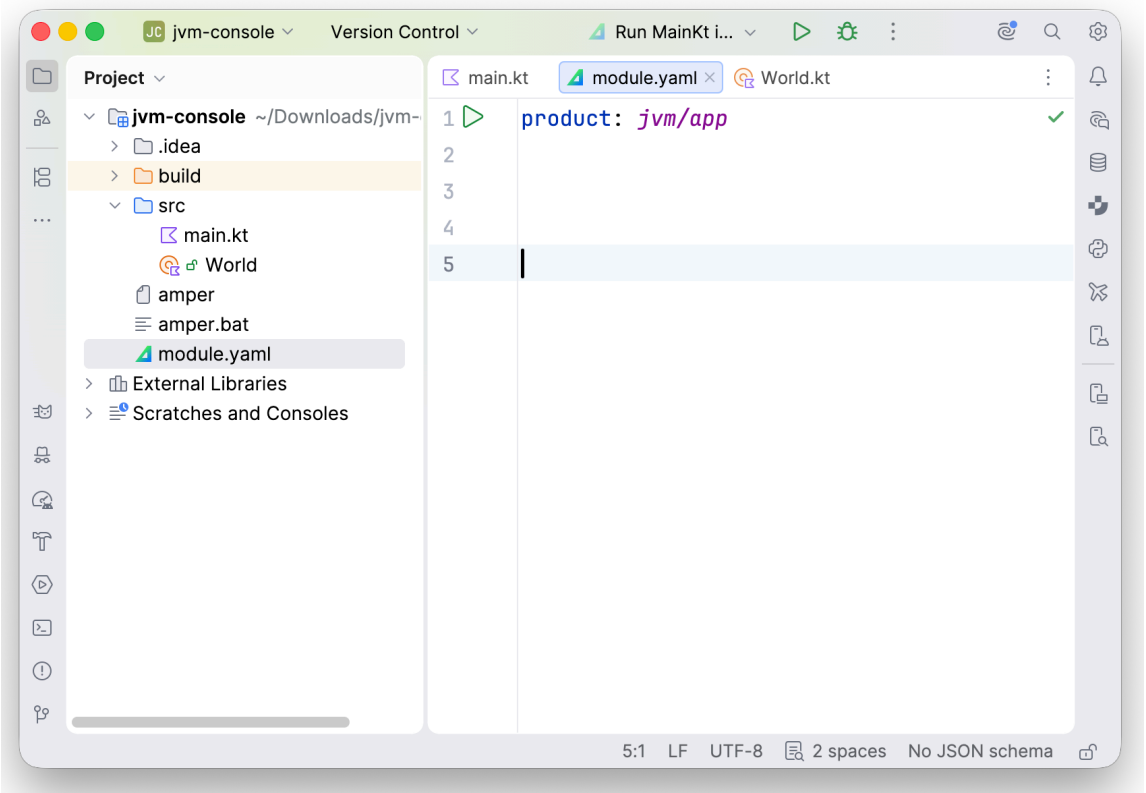

Figure 3: This is a simple `JVM console` application, with a configuration file and a `src` folder.

module.yaml configuration file

The `module.yaml` file contains all of the project configuration information.

Figure 4: The `module.yaml` contains just enough to build the project. A console project is as simple as it gets.

Running your project

To run your project, open the source file, locate the `main` method, and press the `play` button.

Figure 5: This works for most projects.

Alternatively, you can select `Run` > `Run` from the IntelliJ menu (or press the appropriate hotkey e.g., `Ctrl-R`).

> **Note**
>
> If you are working in IntelliJ, it will incrementally compile your code in the background. Running will recompile as-needed before execution.

Build tasks

Amper has a command-line interface, available through the `amper` script that is installed in your project (`amper.bat` for Windows users). The available tasks will be different depending on which project template you selected.

```
$ ./amper
Usage: amper [<options>] <command> [<args>]...

Commands:
  build                 Compile and link all code in the project
  clean                 Remove the projects build output and caches
  clean-shared-caches   Remove Amper caches shared between projects
  generate-completion   Generate a tab-completion script
  init                  Initialize a new Amper project
  package               Package project artifacts for distribution
  publish               Publish modules to a repository
  run                   Run your application
```

Project Templates

JVM Console Application

This is a standalone console application, with no graphical interface included. To add TUI functionality, you would need to add additional libraries like mosaic or kotter.

To build this type of project, choose `JVM console application` from the project template dropdown.

- The target is a console application for Windows, macOS or Linux.
- The starter project contains just the standard library, no external dependencies and no graphical user interface.

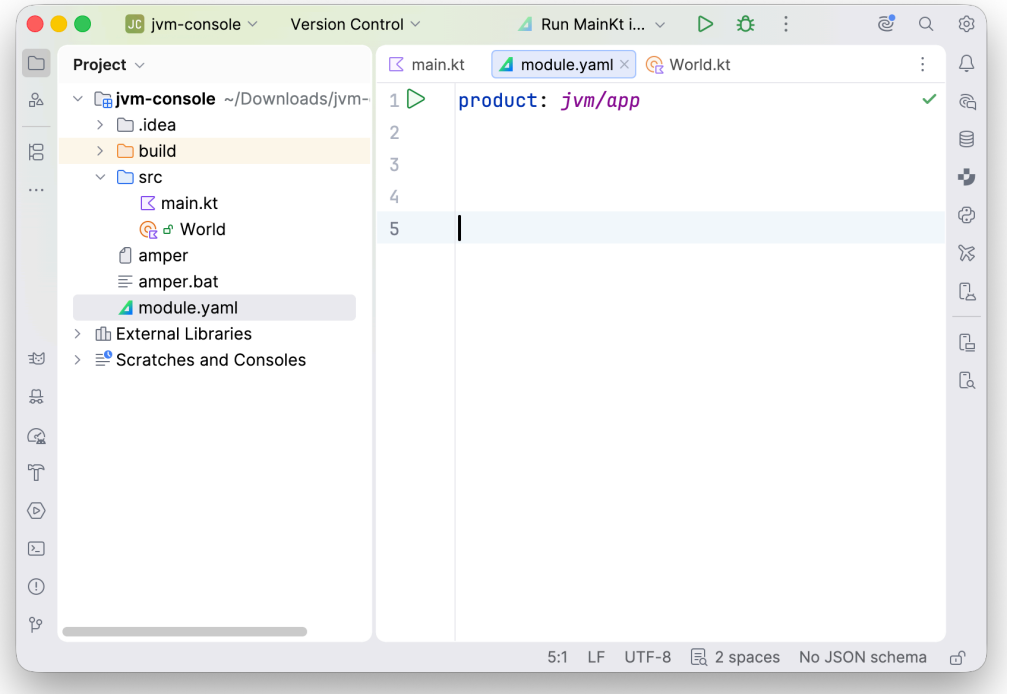

Figure 6: Console applications are the simplest configuration, only needing the `stdlib` to run.

JVM GUI Application

This is a standalone desktop application, which uses `Compose Multiplatform` for the user interface.

To build this type of project, select `JVM GUI Application (Compose Multiplatform)` from the project template dropdown.

- You can target Windows, Linux or macOS with this project.
- The starter project is similar to the console project, but has imports for the Compose framework.
- Note the dependencies in the `module.yaml` file.

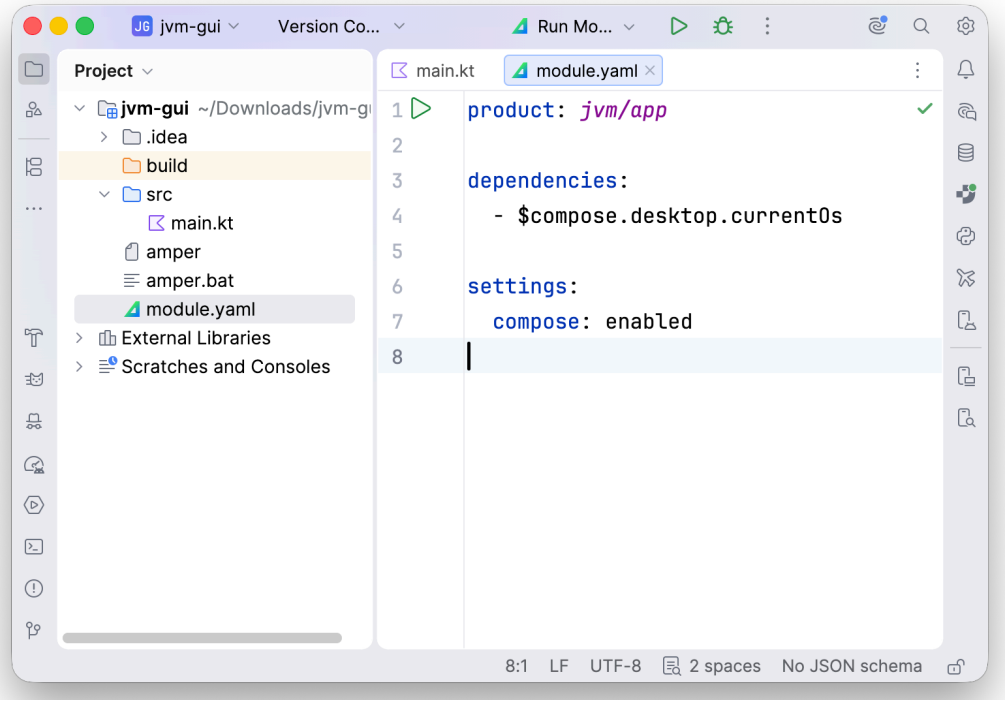

Figure 7: Console applications are the simplest configuration, only needing the `stdlib` to run.

Android Application

This is a standalone Android application, which builds a native graphical application using Jetpack Compose.

To build this type of project, choose `Android application (Jetpack Compose)` from the project template dropdown.

- This builds a standalone Android executable.
- The dependencies include the Android framework and classes.
- There are extra configuration files which we will discuss later (in the Android lecture).

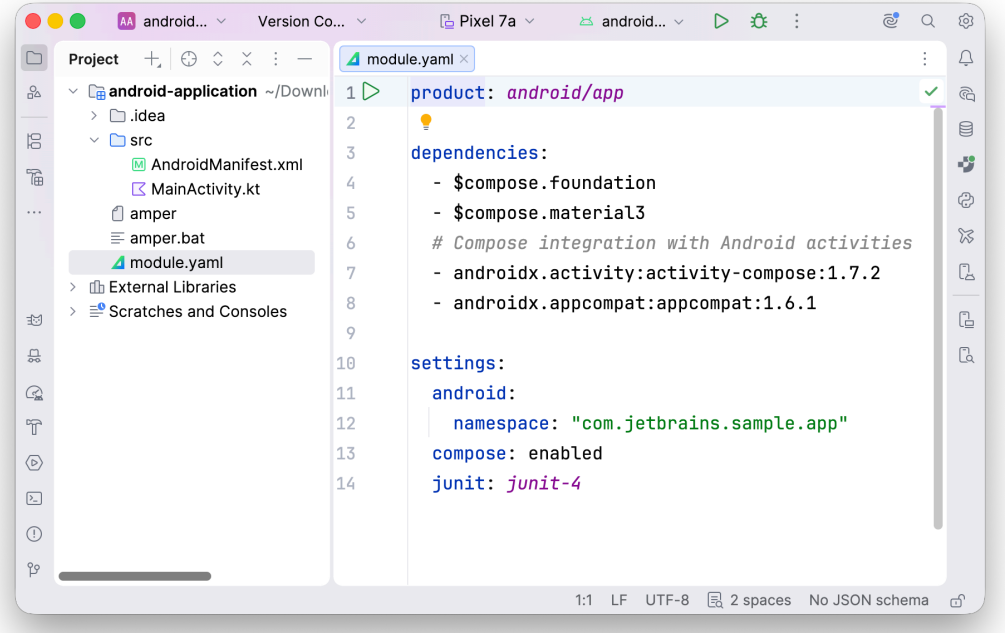

Figure 8: Android applications require more dependencies, and some Android-specific configuration files.

iOS Application

This is a standalone iOS project, that uses KMP and Compose Multiplatform for a user interface.

To build this type, choose `iOS application (Compose Multiplatform)` from the project template dropdown.

- This builds a standalone iOS application from Kotlin code, using XCode and other Apple build tools.
- You need to have XCode and the Command-Line tools installed.
- If you run into XCode build errors, you *may* need to open the `module.xcodeproj` file in XCode and update settings there.

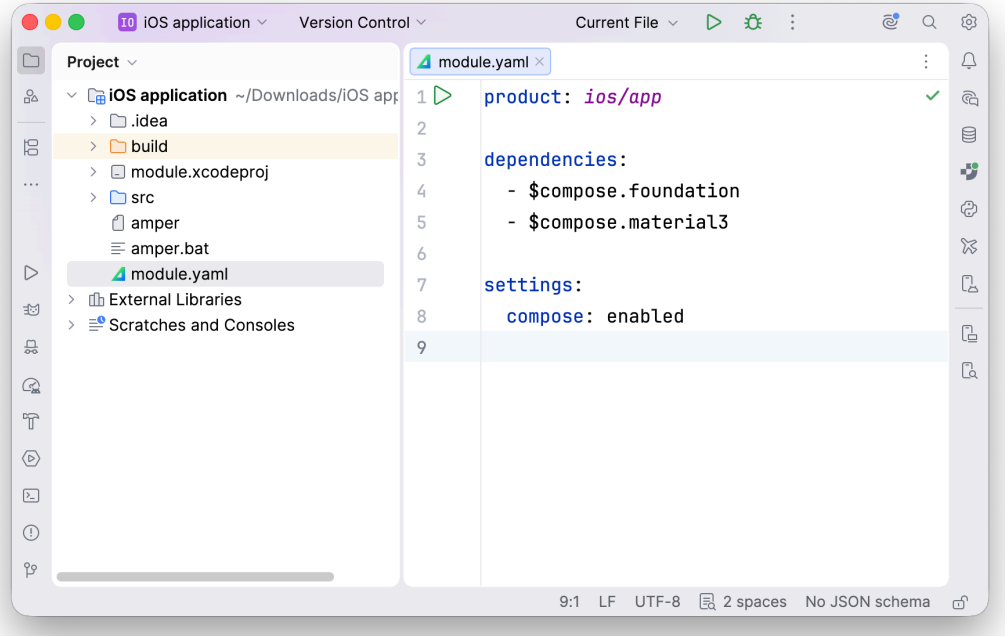

Figure 9: An iOS project uses the XCode build tools to build an iOS executable.

Compose Multiplatform

Our final project is the most flexible and potentially the most complex. This is a multiplatform project, that uses KMP and Compose Multiplatform to target desktop, Android and iOS in a single project.

To build this type, choose `Compose Multiplatform application` from the project template dropdown.

- A top-level folder for each target that contains the entry-point (i.e., distinct for each platform).
- A single `shared` folder where you put code that you want to reuse.
  - `src` is common shared code.
  - `src@platform` is any platform specific implementation code.
- We'll discuss in the KMP lecture.

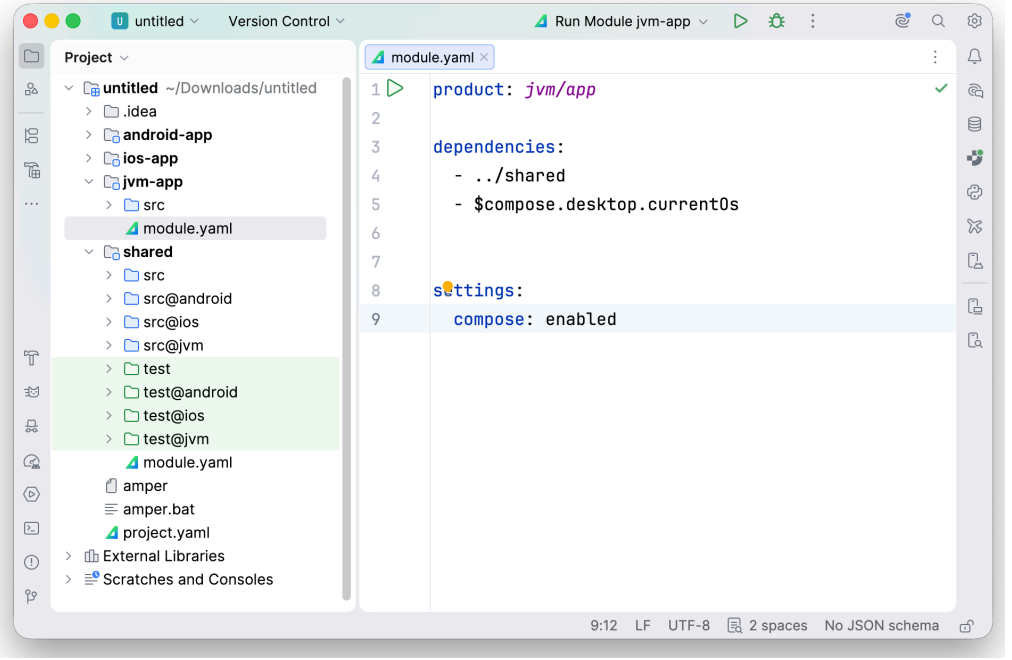

Figure 10: Everything, everywhere, all at once.

Bibliography

[1] JetBrains, “Amper Homepage.” [Online]. Available: https://amper.org/

[2] JetBrains Inc., “Amper Slack Channel.” [Online]. Available: https://slack-chats.kotlinlang.org/c/amper