MOBILE APPLICATION DEVELOPMENT

ANDROID (2017)

LECTURE 22: NDK (PART 1)

SHARED CODE

- Many companies deploy their product on more than one platform.
- Many companies also adapt their old software to new platforms as they appear.
- In an effort to not repeat work, many projects opt to re-use (share) code:
  - Games often reuse nearly 100% of their code across platforms.
  - 'Regular' applications often share significant portions of their code.
- Since each major platform has a set of 'favored' languages and extensive SDKs which are specific to those platforms, how can applications share code?

'WEB APPS' (WEBSITES)

- iOS, Android, and many other operating systems provide a significant amount of support for the idea of 'web apps', or websites which act as native applications.
- On iOS/Android, websites may be 'pinned' to the home screens of devices, showing up with a name and app icon which lets them fit in with the rest of the system.
- If websites are extensively tailored towards this use case, they can look and feel fairly similar to the 'real' applications on those platforms.
- Since these 'web apps' are literally websites, they do not have access to capabilities that many native apps do, and they (usually) must store data online.

CROSS-PLATFORM APPLICATION TOOLKITS

- One common option for sharing code across platforms is to eschew the SDKs for any specific platform and instead use an SDK or toolkit which is specialized at building cross-platform applications.
- A couple popular examples of this are listed below:
  - Xamarin - write code in C#, and either write platform-specific UI or use Xamarin Forms to write shared UI that adapts to each platform.
  - React Native - write code in JavaScript (or variants), write UI which adapts to each platform, and occasionally call out to platform-specific code components.
- These toolkits generally provide good (not great) results while saving a lot of work.

CROSS-PLATFORM NATIVE SDKS

- In situations where sharing UI code or using a comprehensive 3rd-party toolkit is not desirable, developers may opt to write code in a language which all of their target platforms support, and then wrap that code for use on each platform.
- This allows the UI and system interactions for the application to remain fully tailored towards the individual platforms the code runs on, while sharing the 'business logic' of the application in a centralized library.
- Due to its extensive support on virtually all platforms, C is a common choice for an implementation language in cross-platform SDKs. There are also a number of languages which compile to a C-compatible interface and may be used in any situation where C is supported and the language has a stable compiler.

THE NDK

- The NDK, or 'Native Development Kit' is an Android development feature which allows the development of code which interfaces between Java and C/C++.
- C++ is one of the official languages of Android development, and Java provides the JNI (Java Native Interface) which allows Java to communicate with C/C++. Android leverages these features to create the NDK.
- Due to its extensive support on virtually all platforms, C is a common choice for an implementation language in cross-platform SDKs. There are also a number of languages which compile to a C-compatible interface and may be used in any situation where C is supported and the language has a stable compiler.

WHEN TO USE THE NDK

- The NDK is not Android's primary avenue for mobile application development, but it was introduced to provide benefits to developers whose needs do not fit into a purely Java-oriented development environment.
- The use of the NDK should be considered in essentially two cases:
  - Cross-platform applications: When sharing code between platforms is desirable but sharing UI code and system code is not, the NDK is a great choice.
  - Performance: In some (rare) cases, applications may be performance-constrained by the characteristics of the JVM/Java language, and writing performance-critical code in native languages may help speed up operations.

WHEN NOT TO USE THE NDK

- While the NDK allows programmers to use native code, it comes with a set of caveats that limit its usefulness in some situations, meaning that programmers should carefully consider when to use the NDK.
- The use of the NDK should not be considered solely because of:
    - Familiarity: Just because a developer likes C/C++ does not mean they should view the NDK as a 'solution' to getting into Android.
    - Performance(!): The overhead of calls into the JNI means that many tasks which access native code may actually take longer than pure Java solutions. Only consider the NDK to improve performance if you know that moving to native code will provide a significant benefit relative to writing code in Java.

C REFRESHER

- C is a native language, one of the oldest, most widely-used, and most stable languages available. It also is extremely simple from a language standpoint.
- C has a number of primitive types, such as char, int, long, float, double, and bool.
- C also provides pointer types, which are memory addresses that reference the types above. The syntax for a pointer type is type*, where the asterisk after the type name denotes that the value is actually a pointer to something of that type.
- C allows programmers to define more complicated structures than the ones listed above through the use of structs and unions. There is also support for arrays, which are denoted with the syntax of type `variableName[arraySize]`.

C AND JAVA

- The JNI provides access to information about the current Java environment inside of C functions, and can be used to access properties and functions from Java classes in C.
- Java and Kotlin classes may be annotated with special keywords to indicate that certain functions should be linked to implementations in C.
- There are many considerations in C that are not present in Java, such as memory management. To use the NDK correctly, developers should generally make their Java/Kotlin code 'feel' like Java/Kotlin and their C code should act/'feel' like C. This means building memory management on the C side while abstracting it on the Java side, among other concerns.