KivaKit and Java Modules

Pros

- **Improved linkage**—the Java runtime checks module dependencies. If the code checks out, it cannot fail at runtime due to linkage problems (NoClassDefFoundError). If it doesn't check out, the runtime fails early and exits. Classloaders are not isolated, but incompatible JARs cannot "shadow" each other—the runtime fails startup when this happens.
- **Strong encapsulation**—internal packages can be protected from indiscriminate outside access by not exporting them, or with "exports to" and "opens to". This can be used to draw attention to public APIs (versus internal APIs), and to ensure that only public APIs are used, which allows internal packages to be refactored arbitrarily without breaking user code.
- **Classpath compatibility**—non-modular applications can use KivaKit via the classpath, although they require Java 9+ (currently we require Java 17+)
- **Improved performance**—encapsulation increases opportunities for Hotspot to optimize code. JPMS can find classes faster (it doesn't have to search the classpath).

Cons

- **Java 9+ is required** (currently we require Java 17+)
- **Libraries we use lack modularity**
  - Some libraries are not modular and can't be automatic modules either because they contain split packages
  - Some libraries are even incompatible with modular naming conventions
    These issues can take time to resolve, generally by merging code into a single uber-module that contains all of the conflicting code, and assigns a valid module name. We may be able to create a tool to automate some of this work.
- **Javadoc is all or nothing**—Javadoc can only create aggregated documentation indexes if all projects are either modular or non-modular. A mix of the two causes Javadoc to fail. The only solution to this problem is to modularize everything.

Comments

- Developers of KivaKit must ensure that everything is modular
- Users of KivaKit are positively, but not negatively impacted by modularity
- Modularity of libraries has more benefits than modularity of applications
- We can make better and more consistent use of strong encapsulation with internal packages