GAVIN KING

RED HAT

CEYLON SWARM

CEYLON PROJECT

- A relatively new programming language which features:
  - a powerful and extremely elegant static type system
  - built-in modularity
  - support for multiple virtual machine platforms: JVM, Android, JavaScript, Dart
  - powerful multi-language interoperation: Java, JavaScript, Dart
  - excellent tooling: CLI, Eclipse, IntelliJ, Android Studio

TODAY'S TOPIC

- When I was here last time, I talked a lot about the type system (probably the most exciting topic)
- And a little about Ceylon's module system
- This time I'm going to talk about interoperation with Java frameworks with reference to a small demo app based on the WildFly Swarm environment
  - This is a great test case for interop, along with Eclipse, IntelliJ, Android, and other smaller libraries and frameworks

WILDFLY SWARM

- Lets you package your Java EE app and server as a “fat” jar archive
- Offers Java EE APIs without the “container”
  - bundle just the bits of WildFly you’re using, together with your app, and its dependencies, as a single jar file
  - run it using `java -jar`
- Service discovery, failover, integration with Red Hat cloud technologies

ADVANTAGES OF CEYLON ON SWARM

- True null safety, and in general, many more errors detected at compile time
- Union and intersection types
- Tuples
- Type inference and flow-sensitive typing
- Much better support for use of immutability
- Streamlined definition of “model” or “data” classes
- A typesafe metamodel (we’ll see later how this is important!)

OUR DEMO APP

- We want to make the most of Java EE APIs including:
  - JPA for persistence
  - CDI for dependency injection
  - JAX-RS for serving up JSON APIs
  - `transactional`, etc
- And we want to write code using natural Ceylon and Java EE idioms
- https://github.com/DiegoCoronel/ceylon-jboss-swarm/

DO I REALLY NEED ALL THIS?

- No! Not unless you want it!
- There are plenty of other options:
  - `ceylon.dbc` or standalone JPA for persistence
  - `ceylon.json` for producing and parsing and JSON
  - `ceylon.http.server`, Vert.x, many Java web frameworks
  - Guice or Weld (or nothing!) for dependency injection
- If you prefer, Ceylon works with Spring, too

THE CHALLENGE

- JPA, CDI, and JAX-RS are annotation-driven frameworks that work via reflection
- In earlier versions of Ceylon, the mapping to Java wasn't optimized for annotation-driven Java frameworks
  - need JPA converters / JAX-RS adaptors
  - need to explicitly annotate methods `default` (non-final)
  - problems with `generatedValue` and `late`
- Can make it work, but not great for a demo app

THE CHALLENGE

- JPA APIs aren't optimized for usage from Ceylon
  - `setParameter()` methods accept `Object`, and so conversion to Java primitive wrapper types is not automatic
  - operations of `EntityManager` all accept null, but don't know what to do with it
- These issues aren't showstoppers, and you can certainly use JPA APIs directly from Ceylon without difficulty, but it's not quite as comfortable as we would like

THE SOLUTION

- The -ee compiler mode adjusts the mapping to Java so that annotation-driven Java frameworks work more smoothly with Ceylon objects
  - Use direct field access in JPA and JAX-RS
- "ee mode" doesn't affect binary compatibility at all, as the public API of the class isn't affected
  - nor does it affect reflection using Ceylon's metamodel
  - the *only* thing it affects is Java reflection

THE SOLUTION

- The SDK module `ceylon.interop.persistence` is a wrapper for JPA that offers much enhanced type safety for a Ceylon client
- In particular, it has a more typesafe criteria query API that is much less verbose, and doesn't depend on the use of an annotation processor to generate a "model"
  - available in git or in 1.3.3!
- It also solves the little `setParameter()` discomfort

MODULARITY IN CEYLON

- Language level constructs for defining modules, expressing their dependencies, and controlling visibility between modules
- Versioning
- Module archives and module repositories and automatic fetching of dependencies at compilation time and runtime
- Module isolation at runtime
- Interoperation with Maven and npm
- Assembler tools for: Ceylon assembly archives, fat JARs, WARs, WildFly Swarm, Jigsaw mlib, Maven repos, Dart assemblies

SETTING UP A SWARM PROJECT

- Import the Java EE APIs, and SDK module

```
native("jvm")
module jaxrs.example "1.0.0" {
    shared import javax.javaeeapi "7.0";
    import ceylon.interop.persistence "1.3.3-SNAPSHOT";
}
```

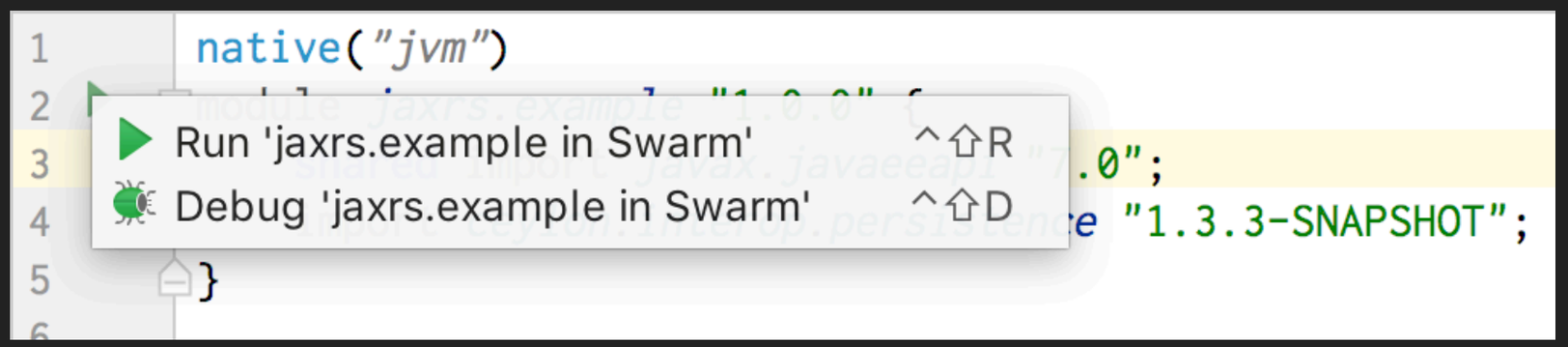
- Override the compile-time JPA API with the Java EE APIs

```
<overrides xmlns="http://www.ceylon-lang.org/xsd/overrides">
    <module module="ceylon.interop.persistence">
        <remove groupId="org.hibernate.javax.persistence"
                artifactId="hibernate-jpa-2.1-api"/>
        <add module="javax.javaeeapi"
             version="7.0"/>
    </module>
</overrides>
```

- Provide `persistence.xml` to configure JPA

SETTING UP A SWARM PROJECT

- That’s everything!
  - no maven, no build scripts, no additional configuration
- As a shortcut I added `ee=true` to my Ceylon `config` to avoid having to specify `-ee` on the command line
- Assemble and run it!



ASSEMBLY WITH SWARM

- The `swarm` plugin for the `ceylon` command assembles a WildFly Swarm fat jar for a given Ceylon module
  - `ceylon plugin install swarm`
  - `ceylon compile`
  - `ceylon swarm --provided-module=javax.javaeeapi jaxrs.example`
  - `java -jar jaxrs.example-1.0.0-swarm.jar`
- The IntelliJ + Eclipse IDEs can do all this for us in one step

ASSEMBLY FOR APPLICATION SERVER

- Alternatively, the `war` plugin for the `ceylon` command assembles a standard Java war for the Ceylon module
  - `ceylon compile`
  - `ceylon war --static-metamodel --provided-module=javax.javaeeapi jaxrs.example`
  - Deploy it to WildFly (or other server)
- No code or project metadata changes required!

MODEL CLASSES

- A model (data) class is declared using JPA annotations

```
 8    shared entity class Employee(name) {
 9
10        generatedValue id
11        shared late Integer id;
12
13        column { length = 50; }
14        shared String name;
15
16        column
17        shared variable Integer? year = null;
18
19    }
```

- That's significantly less noisy than the same code in Java!

REST ENDPOINTS

- A REST endpoint is defined using JAX-RS annotations

```
46      post
47      consumes {"application/json"}
48      produces {"application/json"}
49      shared Employee persist(Employee employee) {
50          service.persist(employee);
51          return employee;
52      }
53
```

- The Java EE services do all the work of mapping our model to the database and to JSON

CRITERIA QUERIES IN JPA

- Java doesn't have a typesafe model of elements belonging to the Java program
- JPA defines a *metamodel* for use with criteria queries, but it must be generated using an annotation processor
- The criteria query API is – overall – highly verbose, quite clumsy to use, and lacking in typesafety, due mainly to limitations of the Java language (no tuples!)

CRITERIA QUERIES IN CEYLON

- Ceylon features a typesafe metamodel built in – it's a bit like Java reflection, but with:
  - typed model objects representing program elements
  - typesafe references to program elements
- I've written a criteria query API that follows the basic design of JPA's API, but is much more typesafe, and is based on Ceylon's metamodel
  - it's much more pleasant to use

CRITERIA QUERIES IN CEYLON

- You can’t tell from a screenshot, but this code fragment is much more typesafe than what can be achieved in Java

```
39      shared List<out Employee> employeesForName(String name) {
40          value crit = entityManager.createCriteria();
41          return
42              let (e = crit.from(`Employee`))
43              crit.where(equal(e.get(`Employee.name`),
44                               crit.parameter(name)))
45                  .select(e)
46                  .getResultList();
47      }
```

- For more complex JPA queries, the additional typesafety helps even more

CONCLUSION

- The resulting code is *extremely* clean and elegant
- The assembly process (`ceylon swarm` command) is a little slower than I would like, but tolerable
- The “ee mode” is not just for Java EE – it works with other reflection-based Java frameworks
- The new criteria query API is way better
- This is a really simple demo app, but the technologies it’s using offer a mountain of really robust functionality

CONCLUSION

- It's a great platform for building microservices
- You should be able to get productive *really* quickly
- The Ceylon language offers so much more that you can grow into
- Next stop: dockerize and deploy to cloud