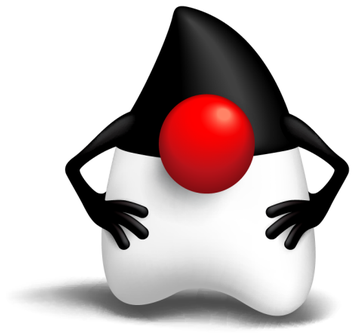
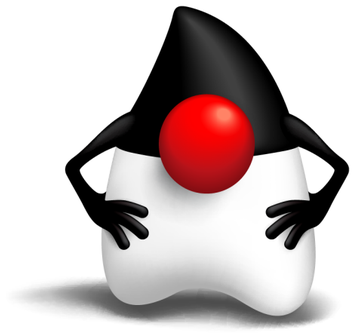
Metadata Features in Java SE 8



Joel Borggrén-Franck
Java Platform Group
Oracle
@joelbf

Metadata Features in Java SE 8



Joel Borggrén-Franck
Java Platform Group
Oracle
@joelbf

First, a message from our lawyers:

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract.

It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Agenda

- Type Annotations
- Repeating Annotations
- Parameter Reflection
- Thoughts from a language hacker/compiler engineer

Annotations

- Java SE 5.0 introduced annotations
- Now an integrated part of the language
  - For example @Override, @Deprecated, @SuppressWarnings
- And the Java ecosystem
  - JPA, Guice, CDI, JAX-RS, JMS

Annotation properties

- Applicability of an annotation can be controlled through @Target
- Retention of an annotation can be specified with @Retention
  - Default is surprisingly Class file
- An annotation can be @Documented, its use showing up in javadoc
- Annotations on classes can be @Inherited

Feature: Type Annotations

- In Java SE 7 annotations can express properties of declarations.

```
@NonNull List<String> f; //Non-null field
```

- How would you express properties of the list elements?

```
class NonNullString extends String {...}
List<NonNullString> f; //List of non-null elements
```

Type Annotations

- To help with this, in Java SE 8 annotations can express properties of type uses:

```
List<@NonNull String> f;
```

```
@Deprecated
class SomeClass { ... }

@Annotated
public void aMethod(@Positive int aParam) {
    @Negative long aVariable;
}

@Foo class ListElement<E> { ... }
```

With Type Annotations

```
new @Interned MyObject();

myString = (@Printable String) myObject;

class UnmodifiableList<T> implements @Readonly List<@Readonly T> {
    void m(List<@NonNull ? extends @NonNull String> x) ...
}

void monitorTemperature() throws @Critical TemperatureException {
    ...
}
```

Type Annotations in core reflection

- Problem: existing methods expose annotations on declarations.

```
class Foo extends @NonNull Bar { ... }

// read annotations on the superclass

Class<?> superClass = Foo.class.getSuperclass();
Annotation a = superClass.getAnnotation(NonNull.class);

// Foo.class.getSuperclass() is:
@ABC @DEF class Bar extends { ... }
```

Type Annotations in core reflection

- New methods expose types

```
class Foo extends @NonNull Bar { ... }

// read annotations on the superclass

AnnotatedType at = Foo.class.getAnnotatedSuperclass();
Annotation a = at.getAnnotation(NonNull.class);
```

Type Annotations in core reflection

- How to “unwrap” parameterized types?

```
List<@NonNull String> f;

Field f = Foo.class.getField(“f”);
AnnotatedType at = f.getAnnotatedType();
if (at instanceof AnnotatedParameterizedType) {
    AnnotatedParameterizedType apt = (AnnotatedParametereizedType) at;
    AnnotatedType[] typeArgs = apt.getAnnotatedActualTypeArguments();
    // typeArgs[0] represents ‘@NonNull String’
    Annotation a typeArgs[0].getAnnotation(NonNull.class);
}
```

Why?

- Annotations on type use is an enabler for static checkers
- Interpreting annotations is the important part
- The Checker framework gives you that interpretation

The Checker framework

- Pluggable type systems
- Lets users develop new useful type checkers
  - Nullness checker that can prove the absence of NPE
  - Lockchecker proves that field access is guarded by lock
  - Regex checker that proves validity of regexps and existence of capture groups

Example: Help with NonNull

```
class BusinessLogic {
    public Entity compute(...) {
        Entity e = (@NonNull Entity) Factory.getEntity();
        //compute compute
        return e;
    }
}
```

The Checker framework

- http://types.cs.washington.edu/checker-framework/
- Example checkers at: http://code.google.com/p/checker-framework/
- We can take the most useful checkers and make them part of Java!

Type Annotations - summary

- The goal is to make it possible extend the type system of Java
  - With for example pluggable checkers from the Checker framework
- Sometimes inference is not enough for the checkers
- Use annotations to help the framework

Feature: Repeating Annotations

Feature: Repeating Annotations

- In Java SE 8 you can now repeat annotations:

```
@Schedule(dayOfMonth="Last")
@Schedule(dayOfWeek="Fri", hour="23")
public void doPeriodicCleanup() { ... }
```

Repeating Annotations

- We added methods to reflection to query for multiple annotations:

```
Method cleanUpMethod = ... ;
Schedule[] schedules =
    cleanupMethod.getAnnotationsByType(Schedule.class);

for (Schedule s : schedules) { ...}
```

Repeating Annotations

- Behind the scenes this is what javac does for you:

```
@Schedule(dayOfMonth="Last"),
@Schedule(dayOfWeek="Fri", hour="23”)
public class MyAction { ... }
```

```
@Schedules(value = {
    @Schedule(dayOfMonth="Last"),
    @Schedule(dayOfWeek="Fri", hour="23”) })
public class MyAction { ... }
```

How to use Repeating Annotations

- Step 1: Create a container annotation

```
public @interface Schedules {
    Schedule[] value;
}
```

- Step 2: Annotate your repeatable annotation, indicating the container

```
@Repeatable(Schedules.class)
public @interface Schedule { ... }
```

Step 2.5

- Your container needs an array valued "`value()`" element with the component type of your repeatable annotation
- All elements non-"`value()`" elements need a default
- The container needs to be retained for at least as long as the contained
- `@Target` for the container must be a subset of `@Target` for the contained
- If contained is `@Documented` the container needs to be `@Documented`

How to use Repeating Annotations

- Step 3: start using the new methods on AnnotatedElement

```
Method cleanUpMethod = ... ;
Schedule[] schedules =
    cleanupMethod.getAnnotationsByType(Schedule.class);

for (Schedule s : schedules) { ...}
```

A real example from our tests

```
@MethodDesc(name = "defaultMethod", modifier = DEFAULT ...)
@MethodDesc(name = "staticMethod", modifier = STATIC ...)
@MethodDesc(name = "method", modifier = ABSTRACT ...)
interface TestIF6 { ... }
```

```
Class<?> typeUnderTest = Class.forName(testTarget);
MethodDesc[] descs = typeUnderTest
    .getAnnotationsByType(MethodDesc.class);

for (MethodDesc desc : descs)
    checkIsFoundByGetMethod(typeUnderTest,
                            desc.name(),
                            argTypes(param));
```

Feature: Parameter Names

Feature: Parameter Names

- In Java SE 7:

```
@Path(“/users”)
class MyResource {
    Response work(@QueryParam(“id”) String id) { ... }
}
```

- In Java SE 8 this could have been written as:

```
@Path(“/users”)
class MyResource {
    Response work(String id) { ... }
}
```

Parameter Names

- Reading parameter names in core reflection:

```
Method workMethod = ...;

Parameter[] parameters = workMethod.getParameters();

for (Parameter p : parameters) { p.getName(); ... }
```

Deep Dive: Language Evolution

```
@Foo @Foo class MyClass { ... }
```

Code assume singular annotations

- Since it wasn't possible to have more than one instance of an annotation per type this is now assumed in lots of code
- Not that hard to find examples of how to upset frameworks, for example:

```
Set<Annotations> set = ...;
for (Annotation a : element.getAnnotations()) {
    // There used to be only one!
    set.add(a);
}
```

Container pattern

- Instead the container pattern was used

```
@Schedule(dayOfMonth="Last"),
@Schedule(dayOfWeek="Fri", hour="23”)
public class MyAction { ... }
```

```
@Schedules(value = {
    @Schedule(dayOfMonth="Last"),
    @Schedule(dayOfWeek="Fri", hour="23”) })
public class MyAction { ... }
```

Compatibility of reflection

- Behavioral compatibility demands that:
  - Repeating annotations are wrapped in a “container” annotation
  - Existing reflection methods treat container annotations as any other annotation
- But the new method `getAnnotationByType(Class)` can look through container annotations

Repeating Annotations

- In order to facilitate the look through in reflection repeatable annotations must nominate their containing type:

```
@Repeatable(Schedules.class)
public @interface Schedule { ... }
```

Deep Dive: Parameter Names

- Parameter names are not used by the JVM
  - If present in the class file at most seen as debug info
  - Not available through Core Reflection - a long-standing “bug”

Issues with Parameter Reflection

- Questions:
  - Should names always be present in the JDK class files?
    - If so, is Oracle committed to their long-term stability?
  - Should names always be present in your own class files?

Issues with Parameter Reflection

- Questions:
  - Should names always be present in the JDK class files? NO
    - If so, is Oracle committed to their long-term stability? N/A
  - Should names always be present in your own class files? NO

The “opt-in” we chose for 8

- We didn’t have the time to get consensus on how to enable method parameter names
- So we made it explicit in the tools
  - Enable parameter name info in tools using a flag like -parameters
- May be revisited in Java SE 9

Deep Dive: Language Evolution

- So you want to change the Java Programming Language?
  - https://blogs.oracle.com/darcy/entry/so_you_want_to_change

- Update JLS
- Implement it in a compiler
- Update essential library support
- Write tests
- Update JVMS
- Update JVM/classfile consumers
- Update JNI
- Update Reflective APIs
- Update serialization support
- Update the javadoc output

Deep Dive: Language Evolution

- Most features requires at least small updates in almost all categories

- Update JLS
- Implement it in a compiler
- Update essential library support
- Write tests
- Update JVMS
- Update JVM/classfile consumers
- Update JNI
- Update Reflective APIs
- Update serialization support
- Update the javadoc output

Essential Library Support

- New enum constants in `java.lang.annotation.ElementType`:
  - TYPE_PARAMETER (Omitted declaration location from Java SE 5)
  - TYPE_USE

Update Reflective APIs

- There is more than one!
  - Core Reflection: `java.lang.Class, java.lang.reflect.*`
  - Language model: `javax.lang.model.{element, type, util}`
  - Annotation Processing: `javax.annotation.processing`
  - Doclet API

j.l.reflect.AnnotatedElement

- New methods in Java SE 8
  - `<T extends Annotation> T getDeclaredAnnotation(Class<T> annotationClass)`
  - `<T extends Annotation> T[] getAnnotationsByType(Class<T> annotationClass)`
  - `<T extends Annotation> T[] getDeclaredAnnotationsByType(Class<T> annotationClass)`
    - The first one can be seen as missing in the 5 API
- Changed to a default method:
  - `boolean isAnnotationPresent(Class<? extends Annotation> annotationClass)`

Adding type use to core reflection

- The current API is focused around declarations
- There is support for generic declarations
  - For example `Class.getField(String name).getGenericType();`
  - Can give you a field of type `List<String>`
  - But the field is tied to a specific (possibly generic) declaration

Adding type use to core reflection

- `Class.getInterfaces()` returns an array of Class objects; per-use annotations can not be stored on Class objects

- Add a suit of sibling getAnnotatedFoo methods for get(Generic)Foo methods in core reflection

  - For example: `Class.getAnnotatedInterfaces()`

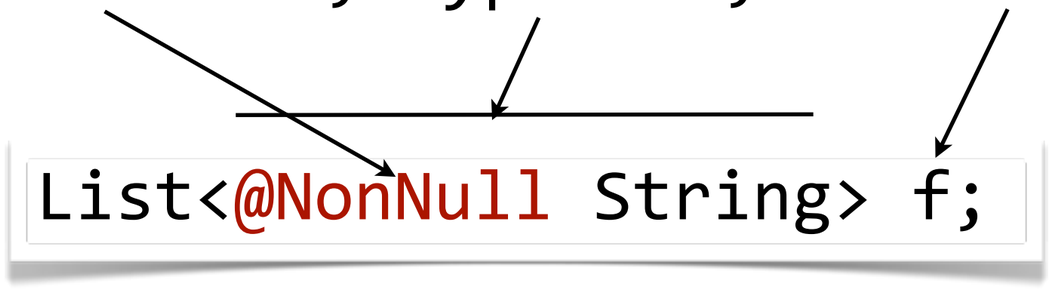
  - returns AnnotatedType

Adding type use to core reflection

- `AnnotatedType`
  - `AnnotatedArrayType`
  - `AnnotatedParameterizedType`
  - `AnnotatedTypeVariable`
  - `AnnotatedWildCardType`

Adding type use to core reflection

- j.l.r.AnnotatedType is conceptually a tuple of (annotations, type-use, declaration)



```
List<@NonNull String> f;
```

- Allows for navigating the declaration that express the uses of the types we are looking at

```
@A List<@B Comparable<@F Object @C [] @D [] @E []>>
```

Acknowledgements

- Maurizio Cimadamore
- Werner Dietl
- Joe Darcy
- Jon Gibbons
- Alex Buckley
- Sonali Goel
- Brian Goetz
- Mike Keith
- Eric McCorkle
- Matherey Nunez
- Bhavesh Patel
- Bill Shannon
- Steve Sides
- Charlie Wang
- Sharon Zakhour

Wrapping up

- Annotations in Java SE 8 are easier to write and discover
- Repeating and type annotations enables better framework
- Parameter reflection fixes a long-standing bug in the JDK