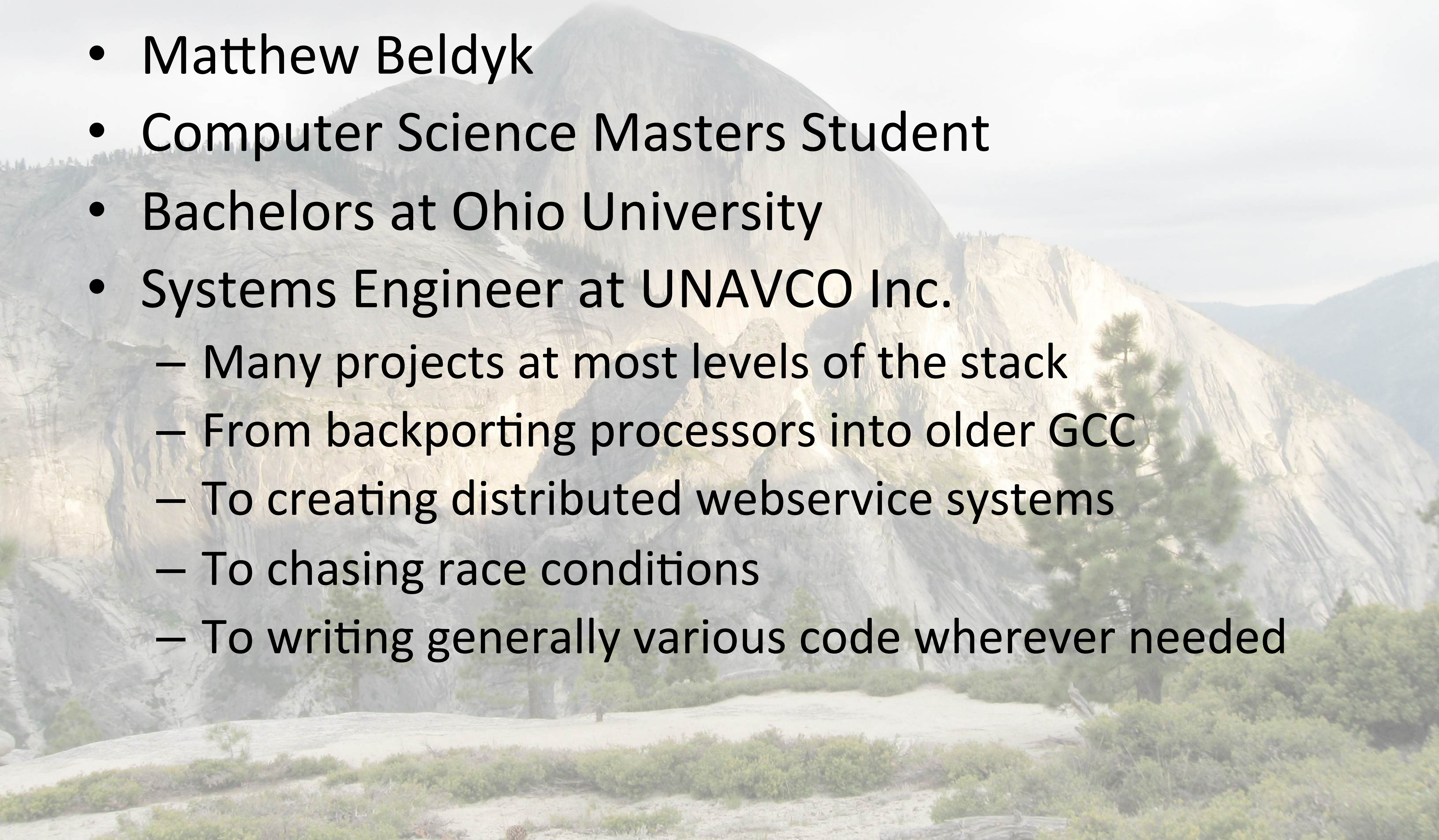
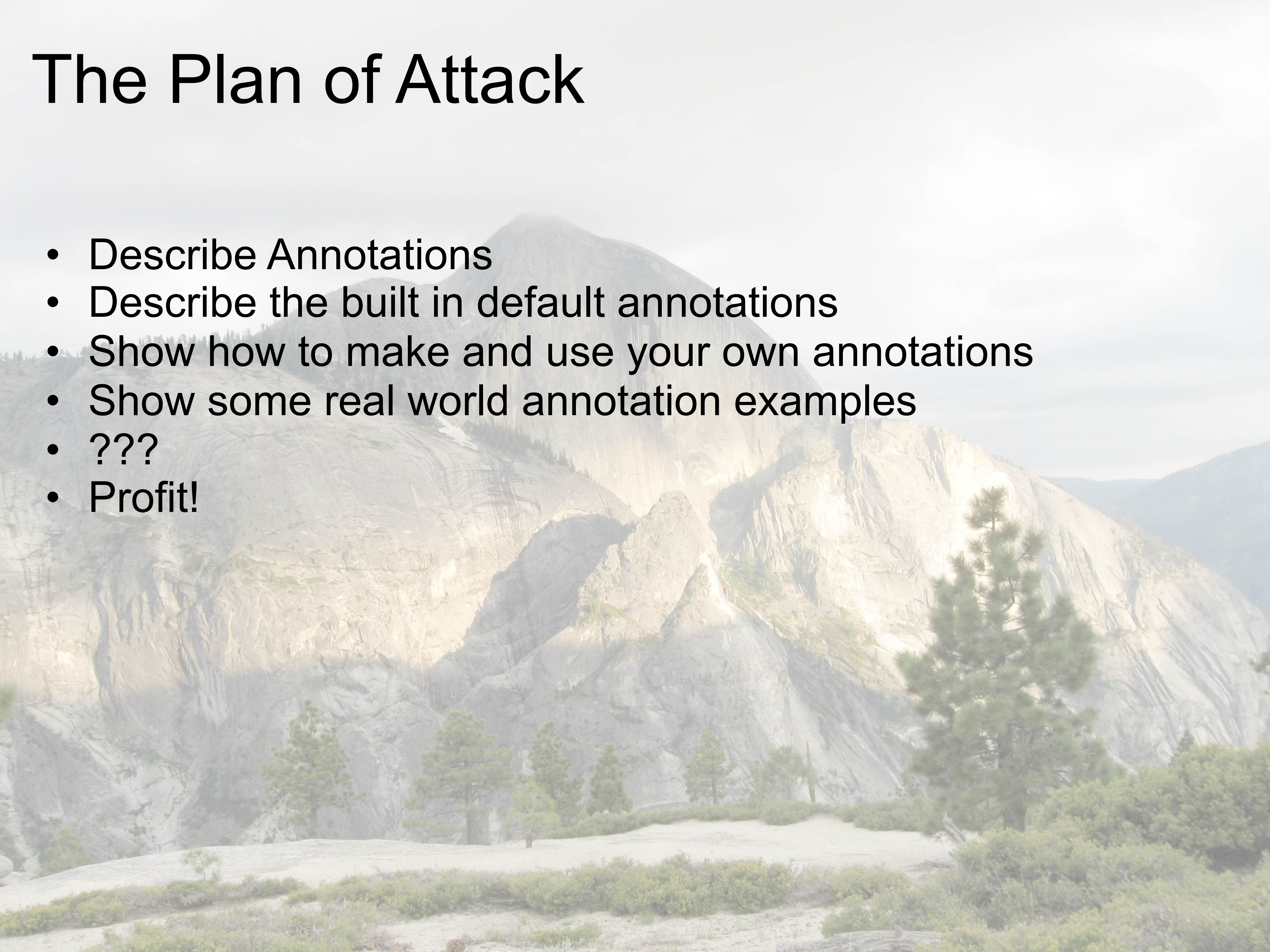
Java @Annotations

Matt Beldyk
CSCI 5448 - Object Oriented Design and Analysis
Spring 2011



About Me

- Matthew Beldyk
- Computer Science Masters Student
- Bachelors at Ohio University
- Systems Engineer at UNAVCO Inc.
  - Many projects at most levels of the stack
  - From backporting processors into older GCC
  - To creating distributed webservice systems
  - To chasing race conditions
  - To writing generally various code wherever needed

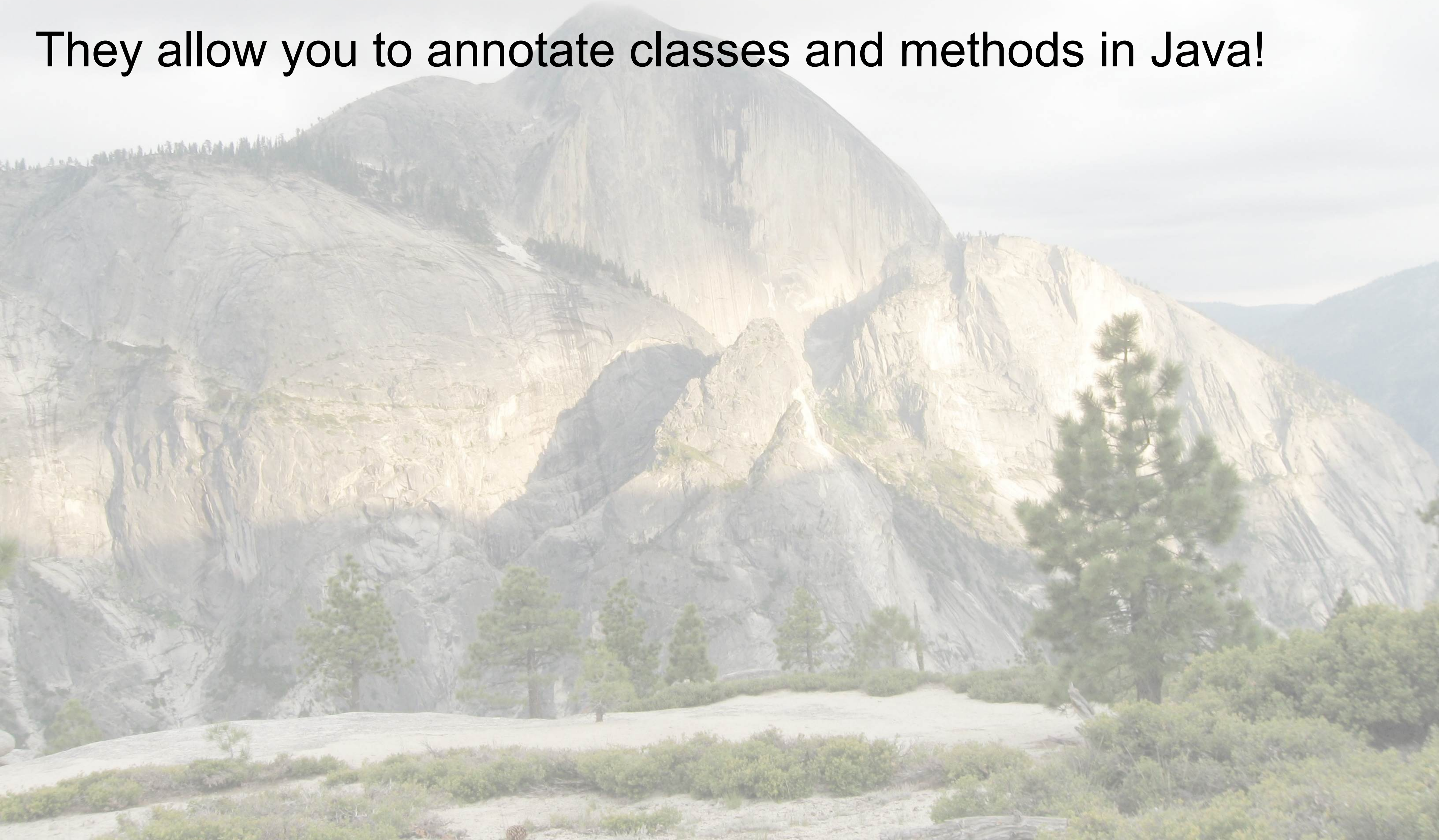


The Plan of Attack

- Describe Annotations
- Describe the built in default annotations
- Show how to make and use your own annotations
- Show some real world annotation examples
- ???
- Profit!

What are Annotations?

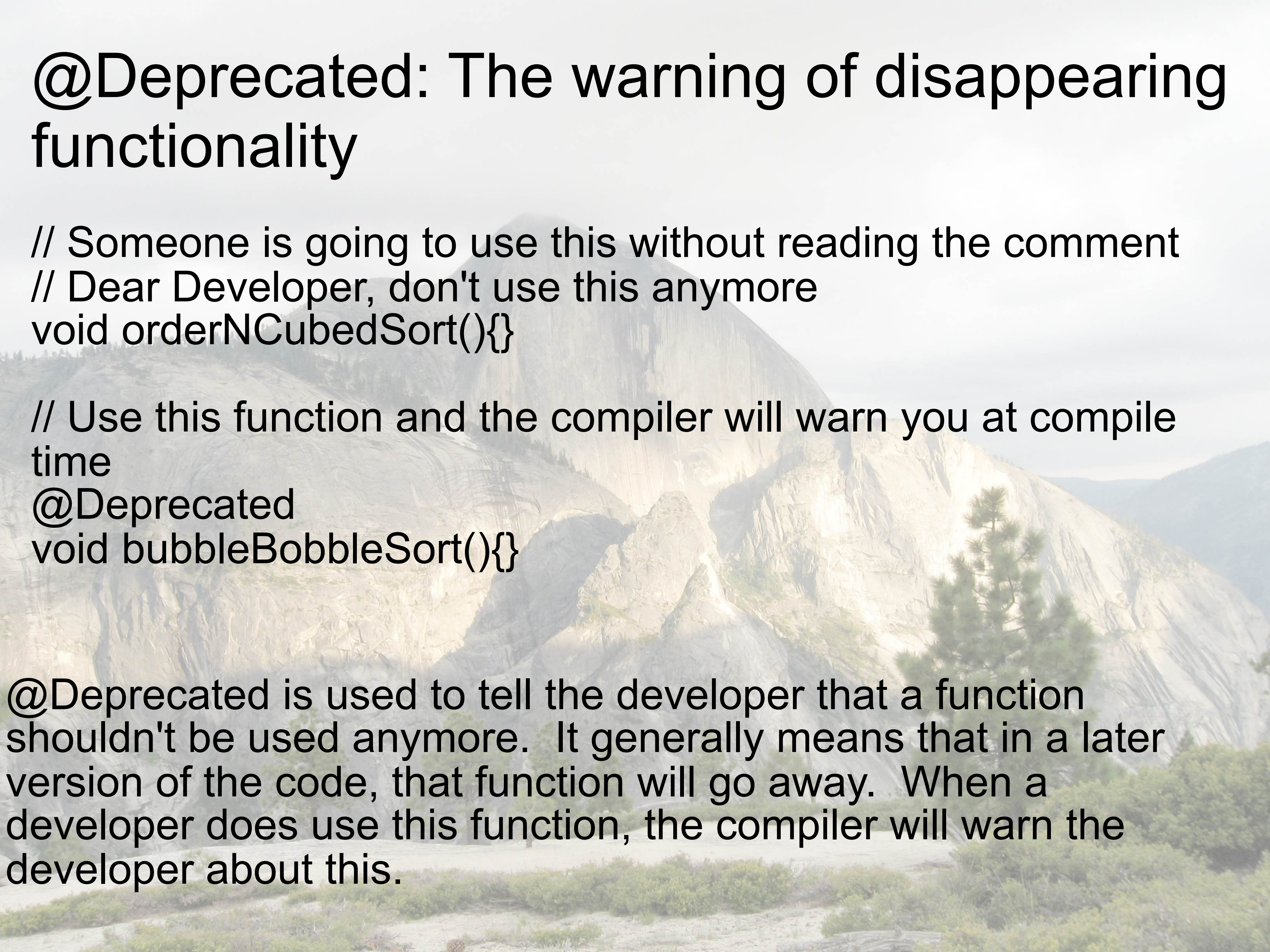
They allow you to annotate classes and methods in Java!



Holy cyclic definition, Batman!

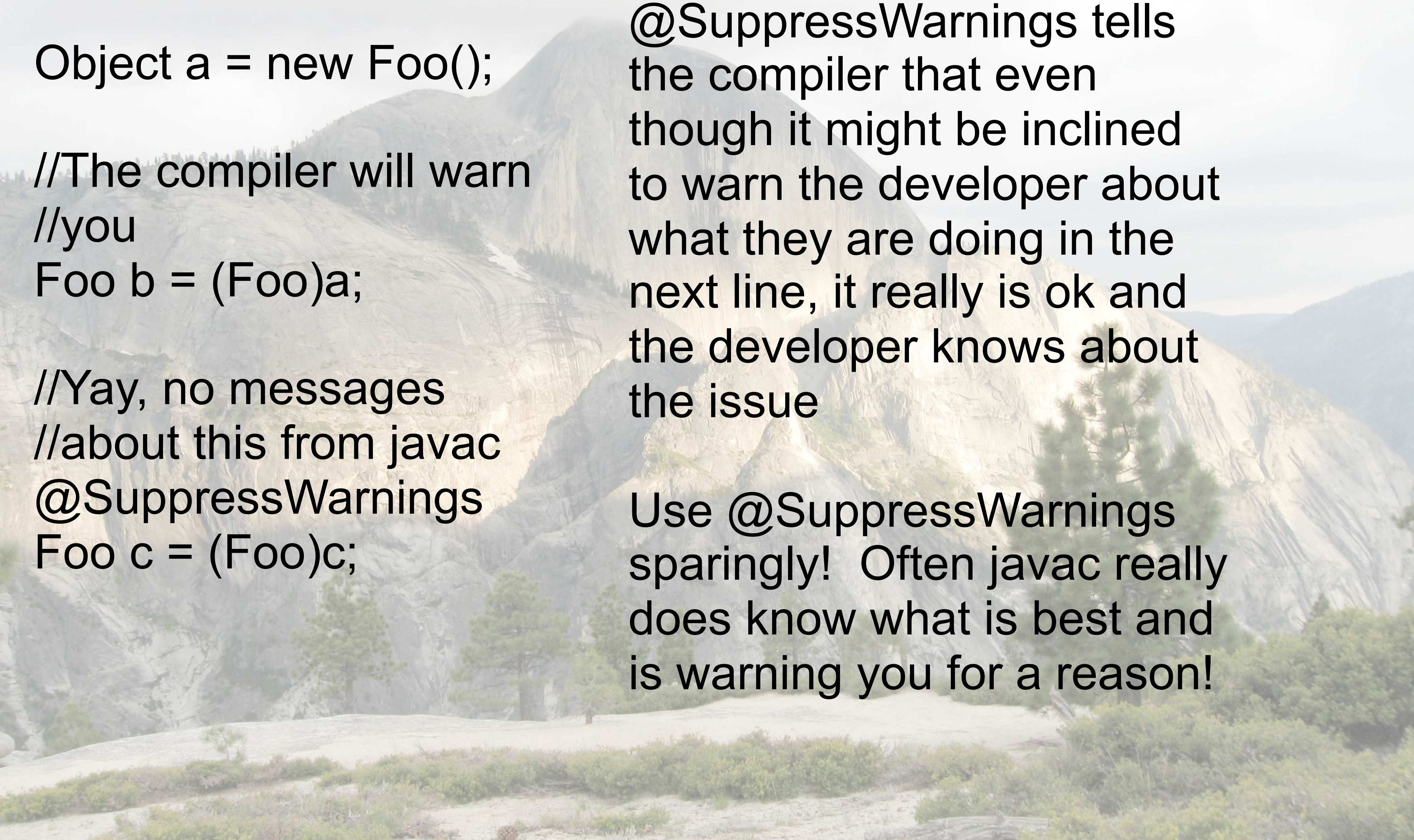
- Introduced in Java 5
- JSR #175: http://www.jcp.org/en/jsr/detail?id=175
- Allow developers to introduce metadata about functions and classes in code
- Better than just a comment
- 3 introduced by default by the JDK
  - @Deprecated
  - @SuppressWarnings
  - @Override
- You can write your own too

@Deprecated: The warning of disappearing functionality



```
// Someone is going to use this without reading the comment
// Dear Developer, don't use this anymore
void orderNCubedSort(){}

// Use this function and the compiler will warn you at compile
time
@Deprecated
void bubbleBobbleSort(){}
```

@Deprecated is used to tell the developer that a function shouldn't be used anymore. It generally means that in a later version of the code, that function will go away. When a developer does use this function, the compiler will warn the developer about this.



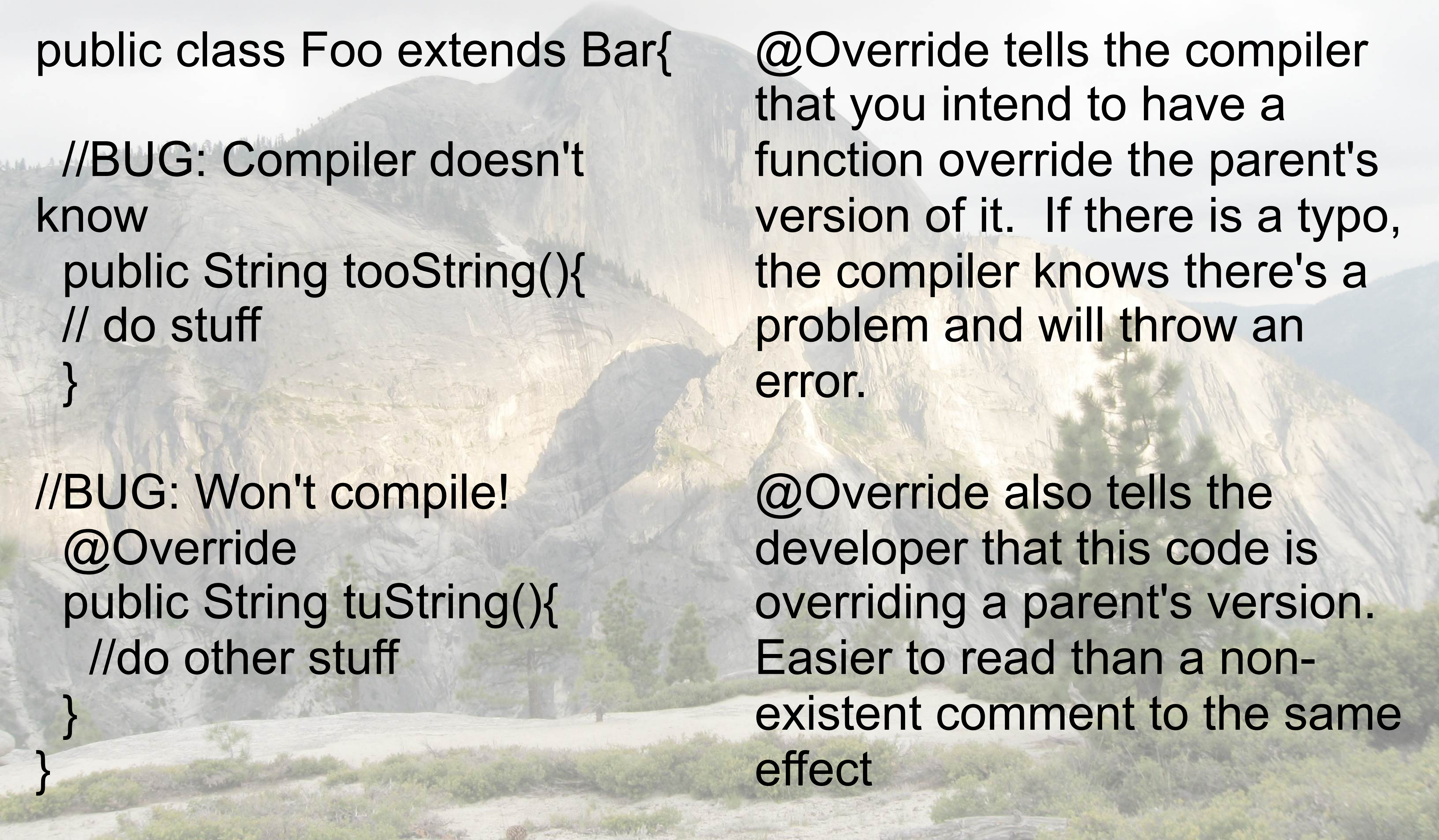
@SuppressWarnings: No really, I know what I'm doing!

```
Object a = new Foo();

//The compiler will warn
//you
Foo b = (Foo)a;

//Yay, no messages
//about this from javac
@SuppressWarnings
Foo c = (Foo)c;
```

@SuppressWarnings tells the compiler that even though it might be inclined to warn the developer about what they are doing in the next line, it really is ok and the developer knows about the issue

Use @SuppressWarnings sparingly! Often javac really does know what is best and is warning you for a reason!

@Override: The class extender's friend



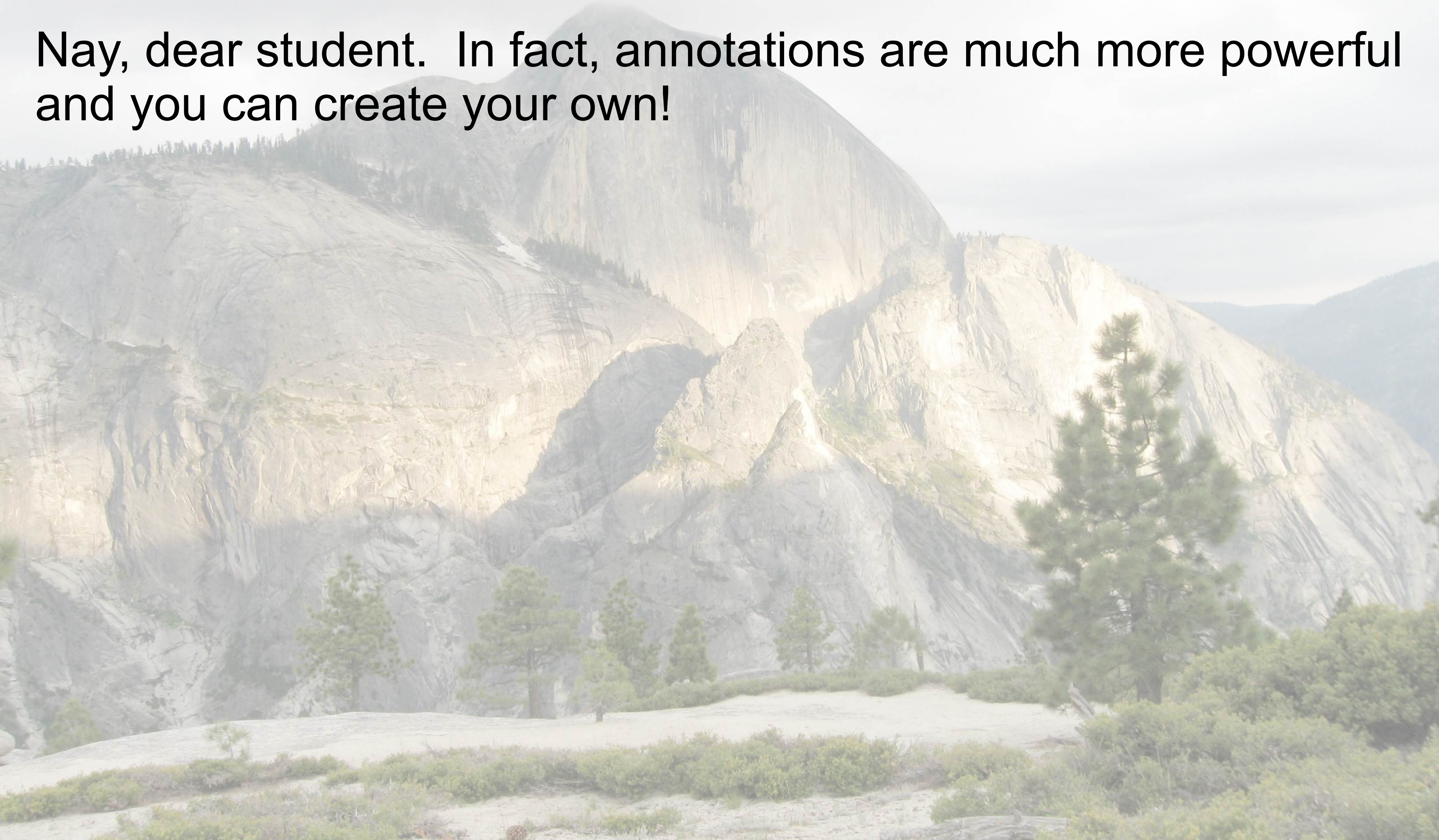
```
public class Foo extends Bar{

  //BUG: Compiler doesn't
know
  public String tooString(){
  // do stuff
  }

//BUG: Won't compile!
  @Override
  public String tuString(){
    //do other stuff
  }
}
```

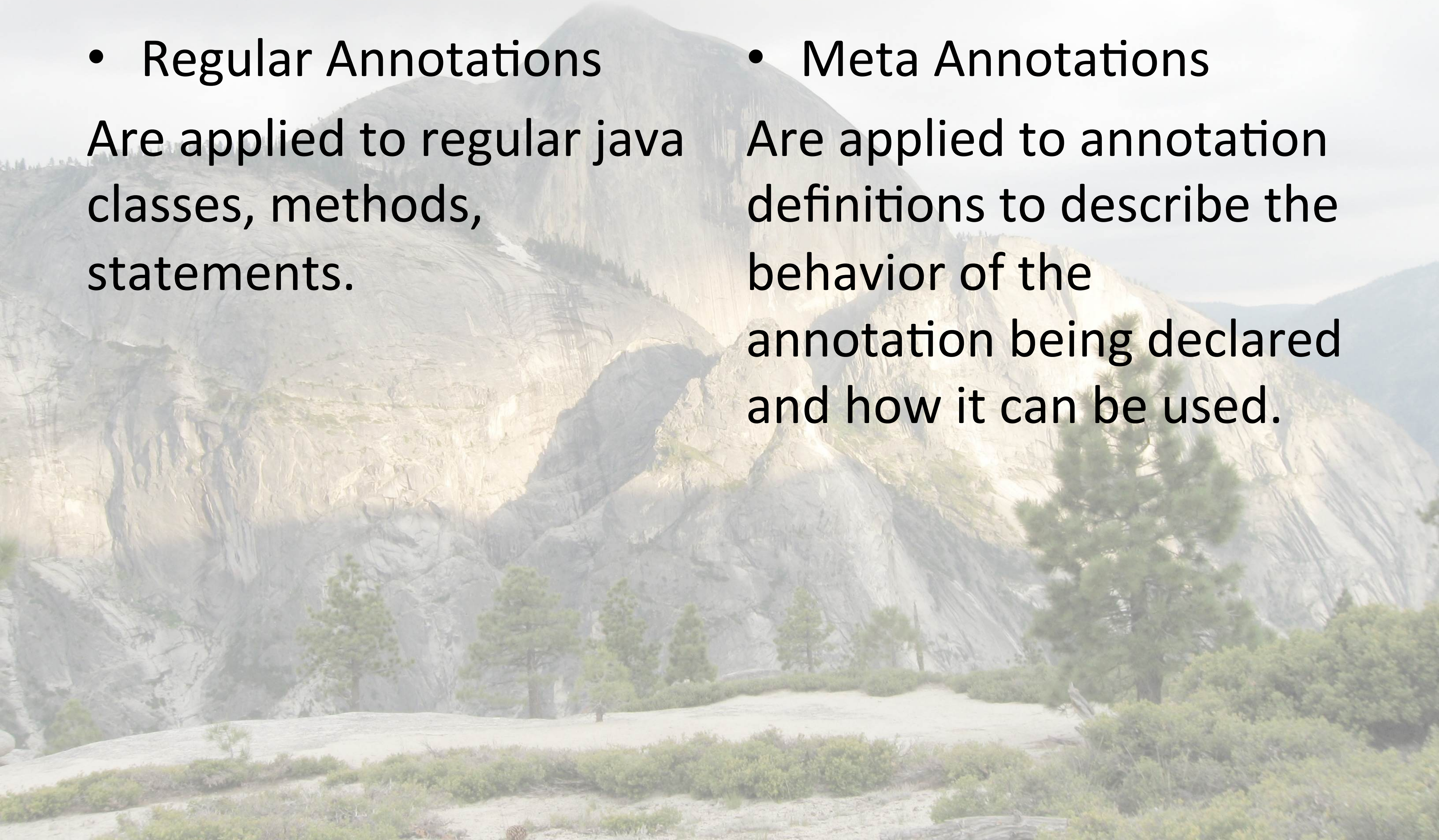
@Override tells the compiler that you intend to have a function override the parent's version of it. If there is a typo, the compiler knows there's a problem and will throw an error.

@Override also tells the developer that this code is overriding a parent's version. Easier to read than a non-existent comment to the same effect

But these just appear to be new keywords?

Nay, dear student. In fact, annotations are much more powerful and you can create your own!





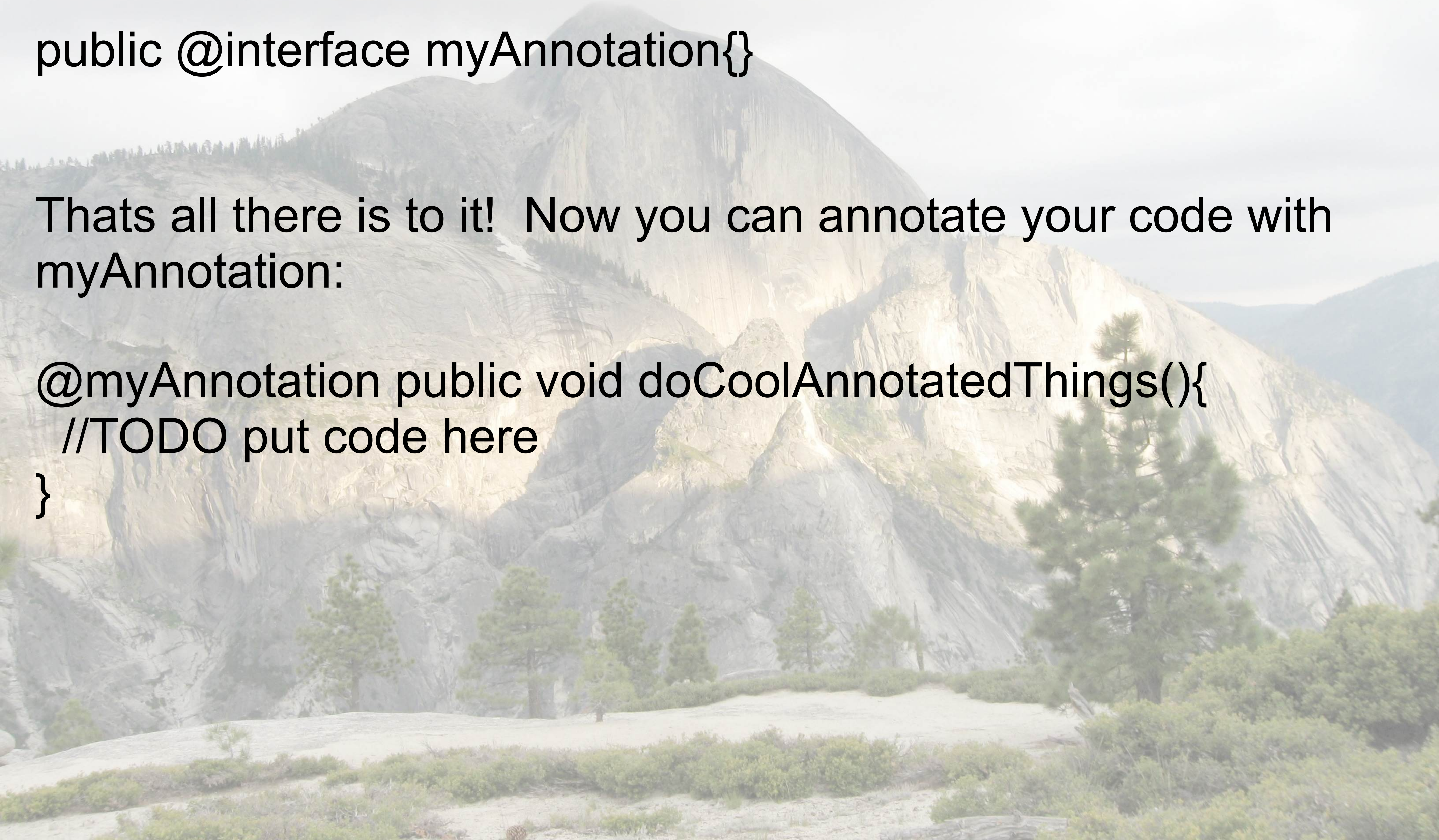
There are two kinds of annotations

- Regular Annotations

Are applied to regular java classes, methods, statements.

- Meta Annotations

Are applied to annotation definitions to describe the behavior of the annotation being declared and how it can be used.

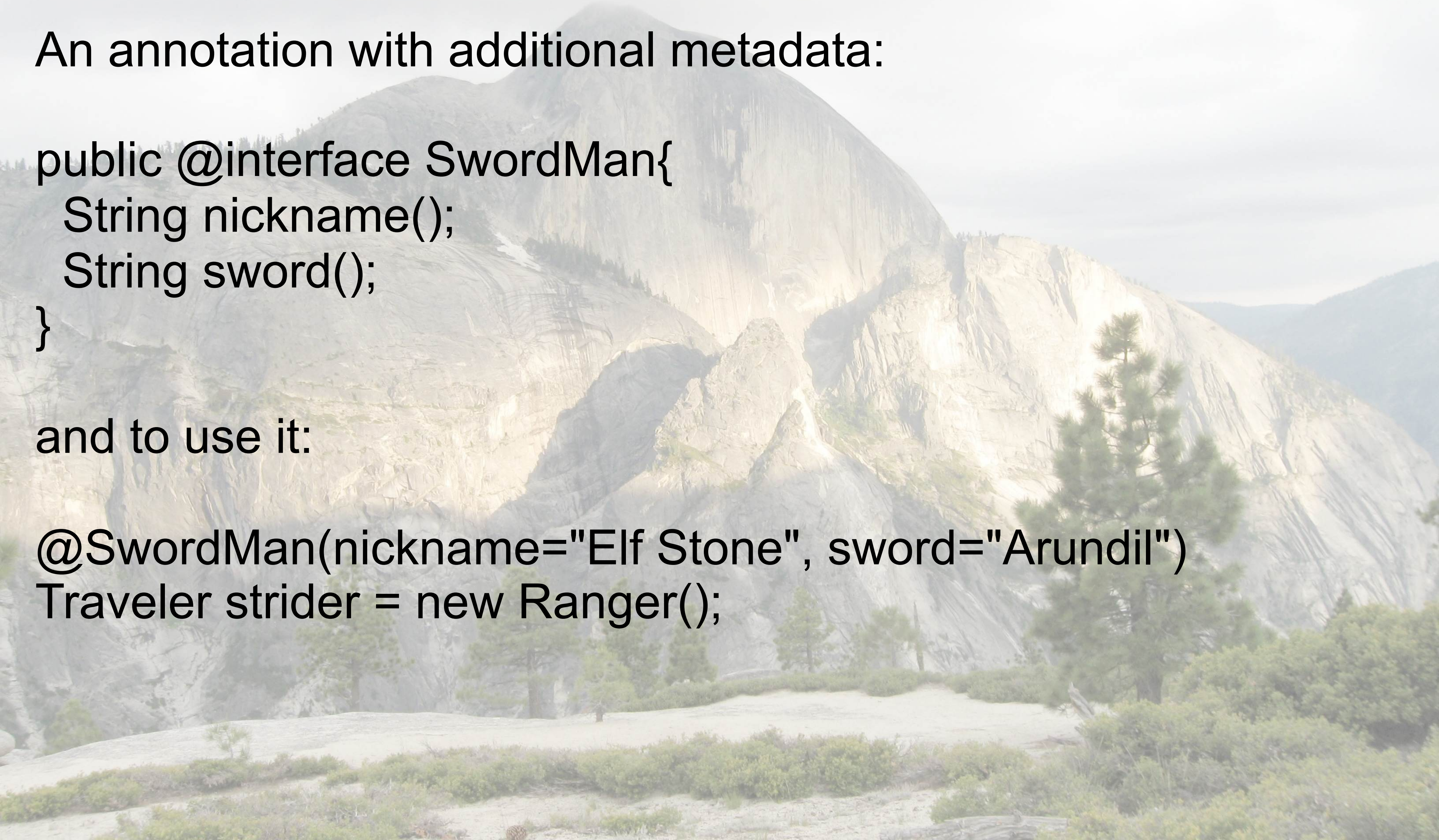


Let's Create our own Annotation

```
public @interface myAnnotation{}
```

Thats all there is to it! Now you can annotate your code with myAnnotation:

```
@myAnnotation public void doCoolAnnotatedThings(){
  //TODO put code here
}
```



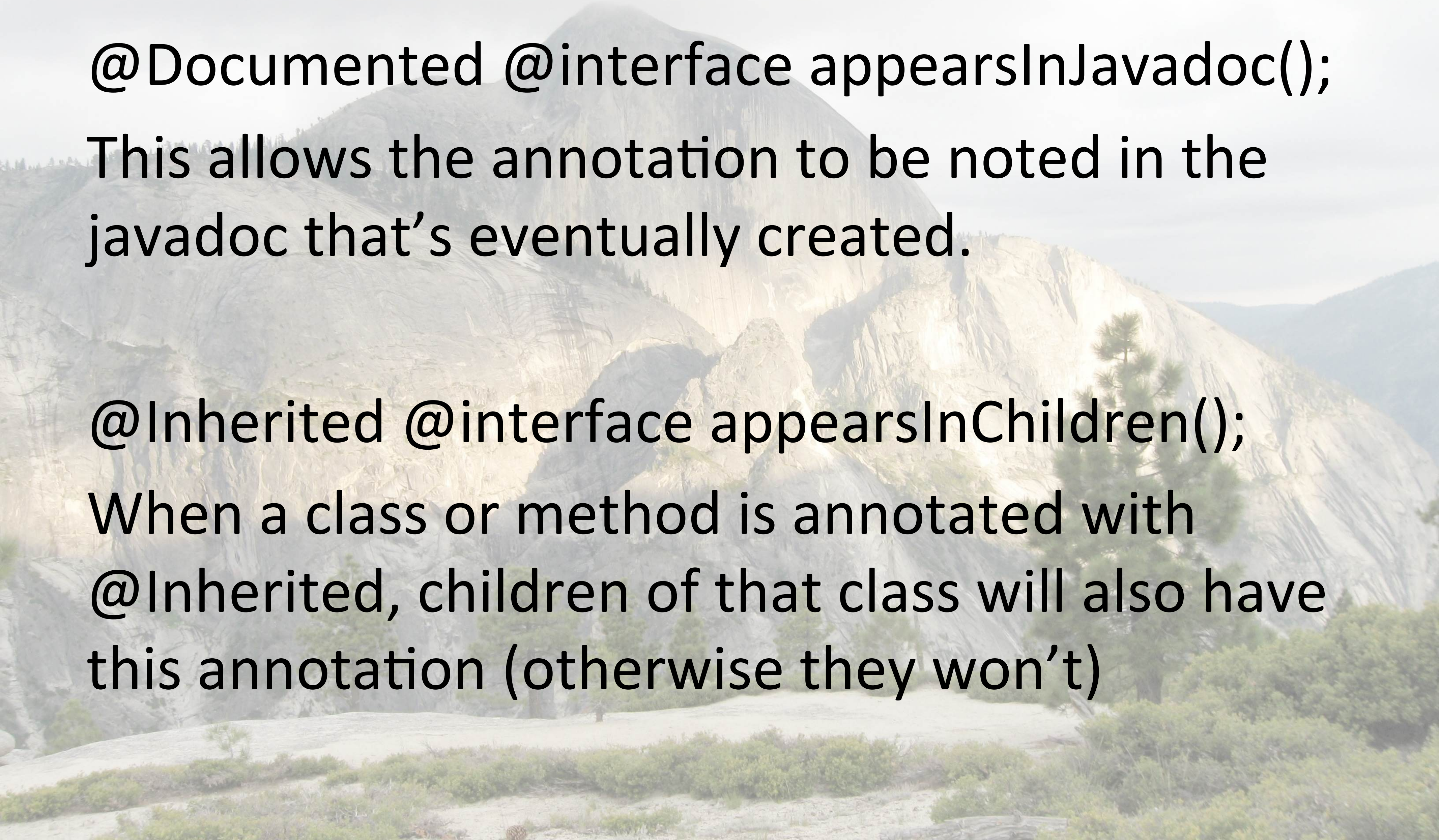
Whoohoo, but that's not very compelling.

An annotation with additional metadata:

```
public @interface SwordMan{
  String nickname();
  String sword();
}
```

and to use it:

```
@SwordMan(nickname="Elf Stone", sword="Arundil")
Traveler strider = new Ranger();
```



A word on Meta Annotations

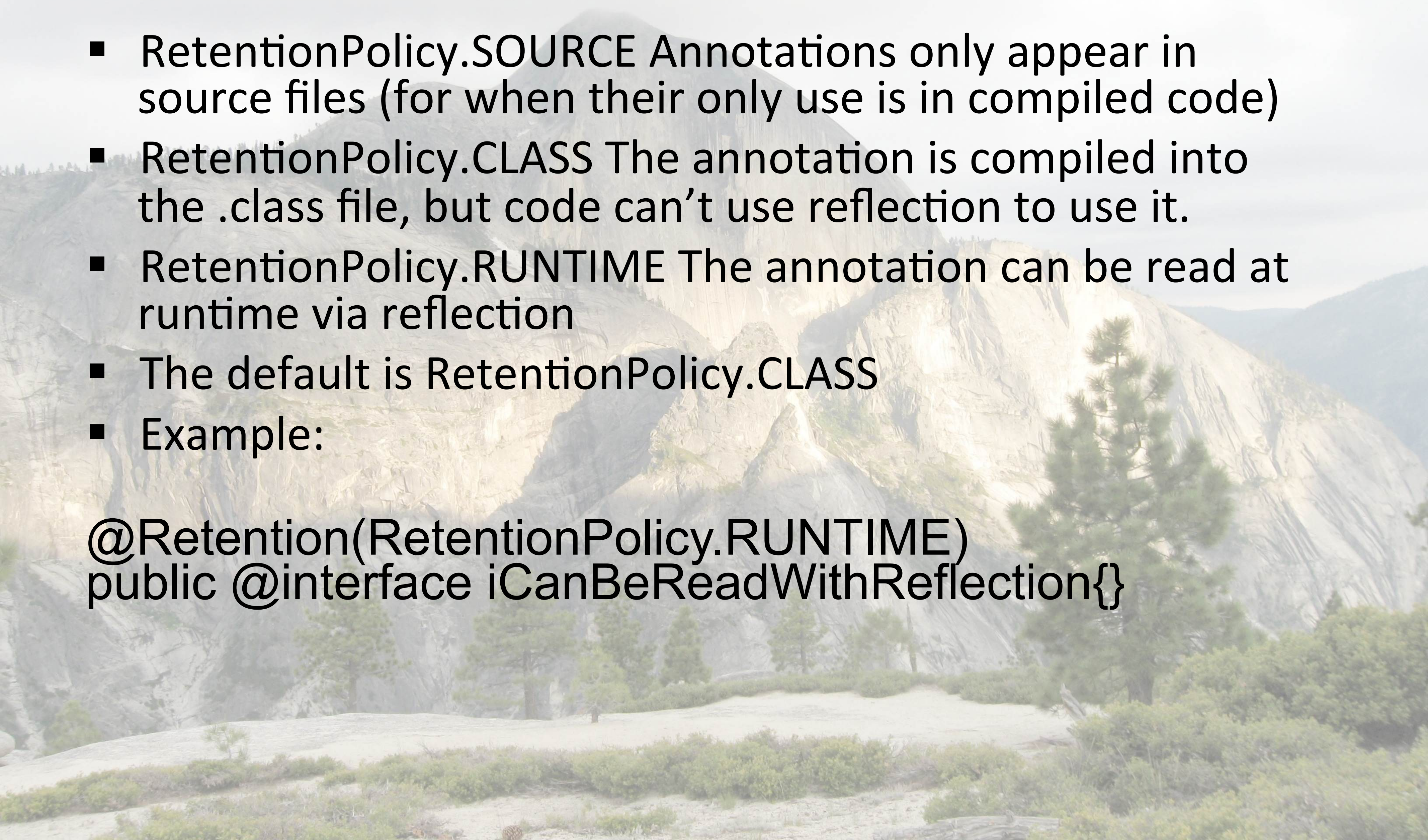
```
@Documented @interface appearsInJavadoc();
```

This allows the annotation to be noted in the javadoc that's eventually created.

```
@Inherited @interface appearsInChildren();
```

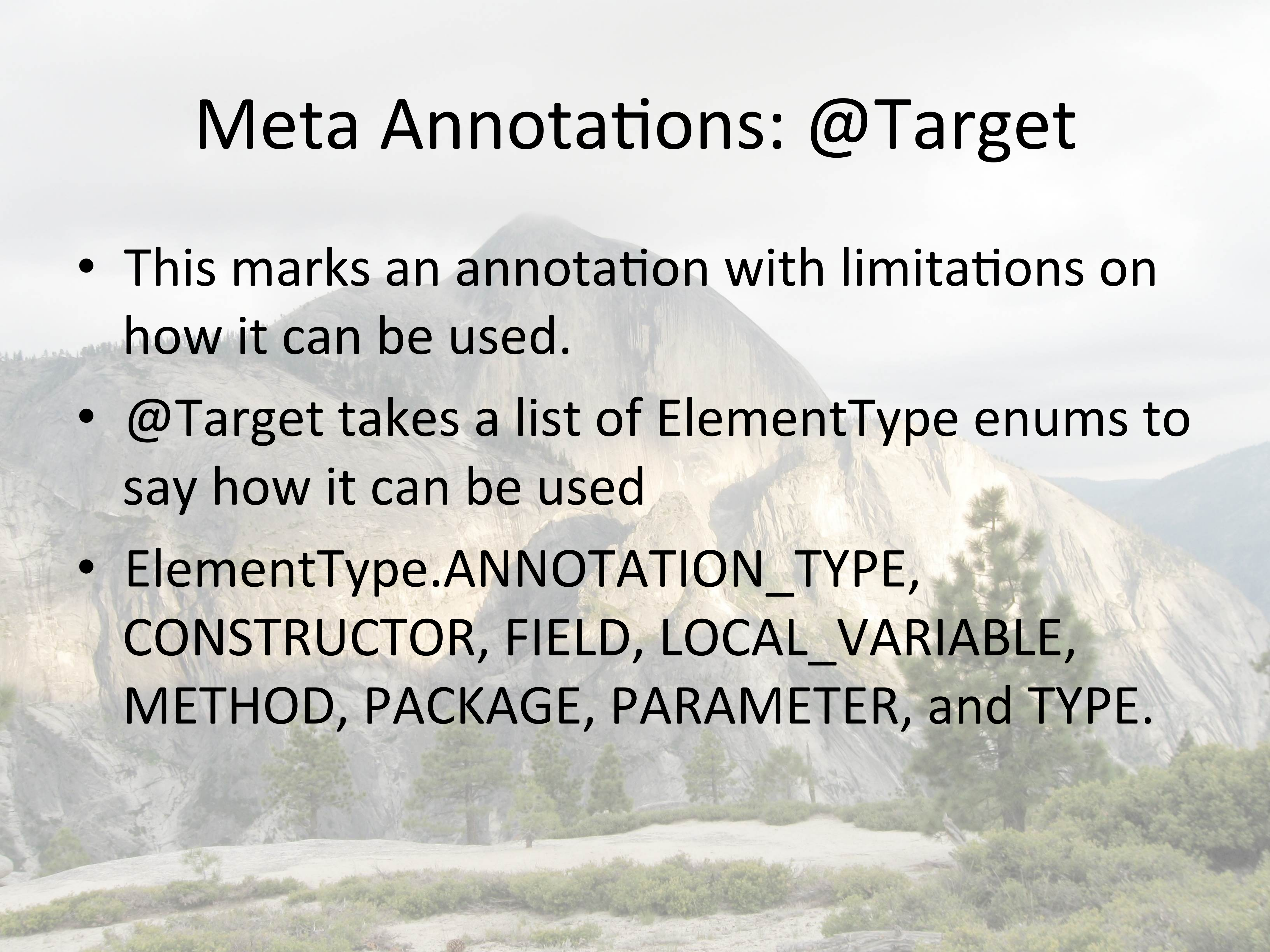
When a class or method is annotated with @Inherited, children of that class will also have this annotation (otherwise they won't)

Meta Annotations: Retention Policies



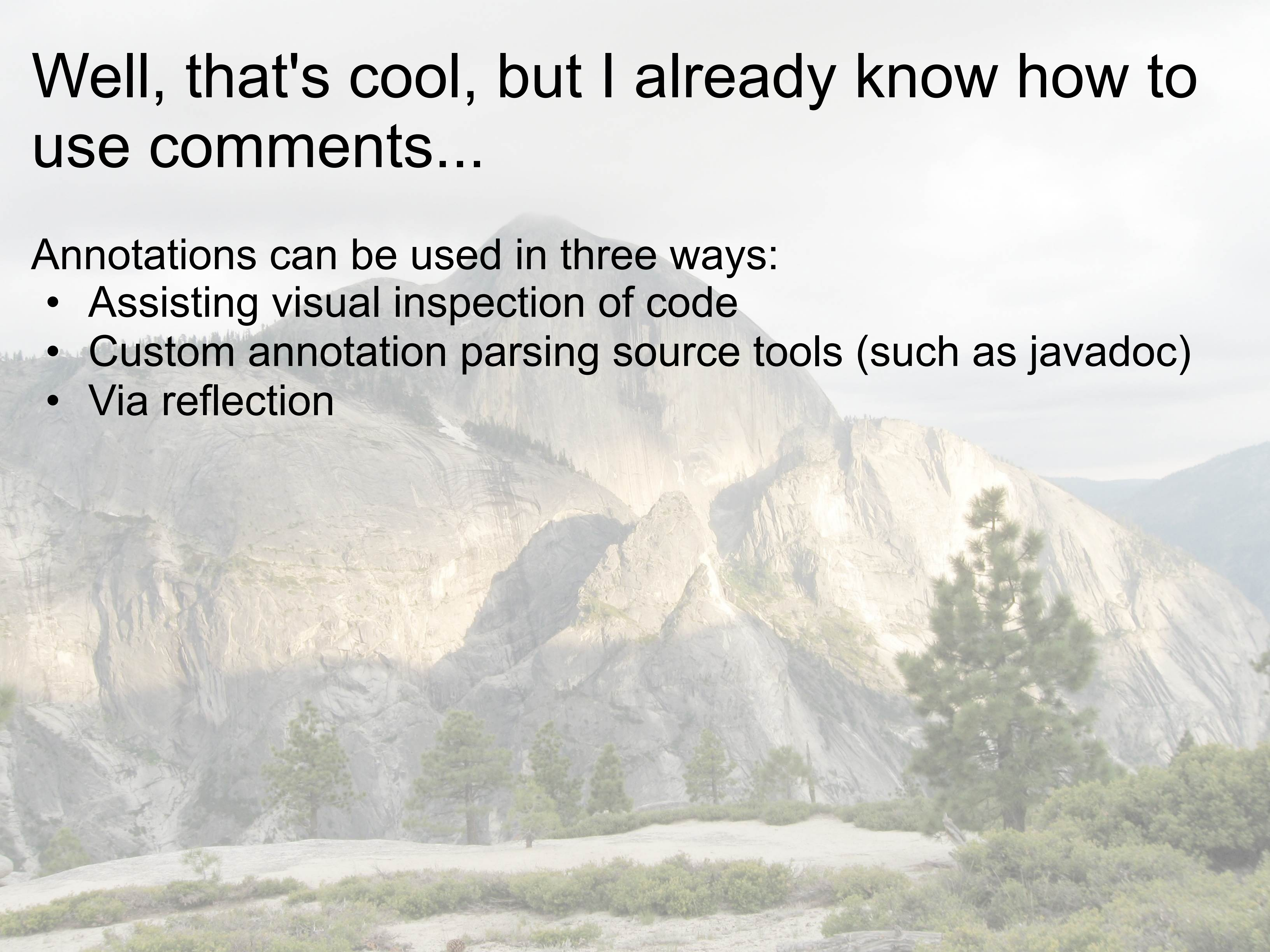
- RetentionPolicy.SOURCE Annotations only appear in source files (for when their only use is in compiled code)
- RetentionPolicy.CLASS The annotation is compiled into the .class file, but code can't use reflection to use it.
- RetentionPolicy.RUNTIME The annotation can be read at runtime via reflection
- The default is RetentionPolicy.CLASS
- Example:

```
@Retention(RetentionPolicy.RUNTIME)
public @interface iCanBeReadWithReflection{}
```



Meta Annotations: @Target

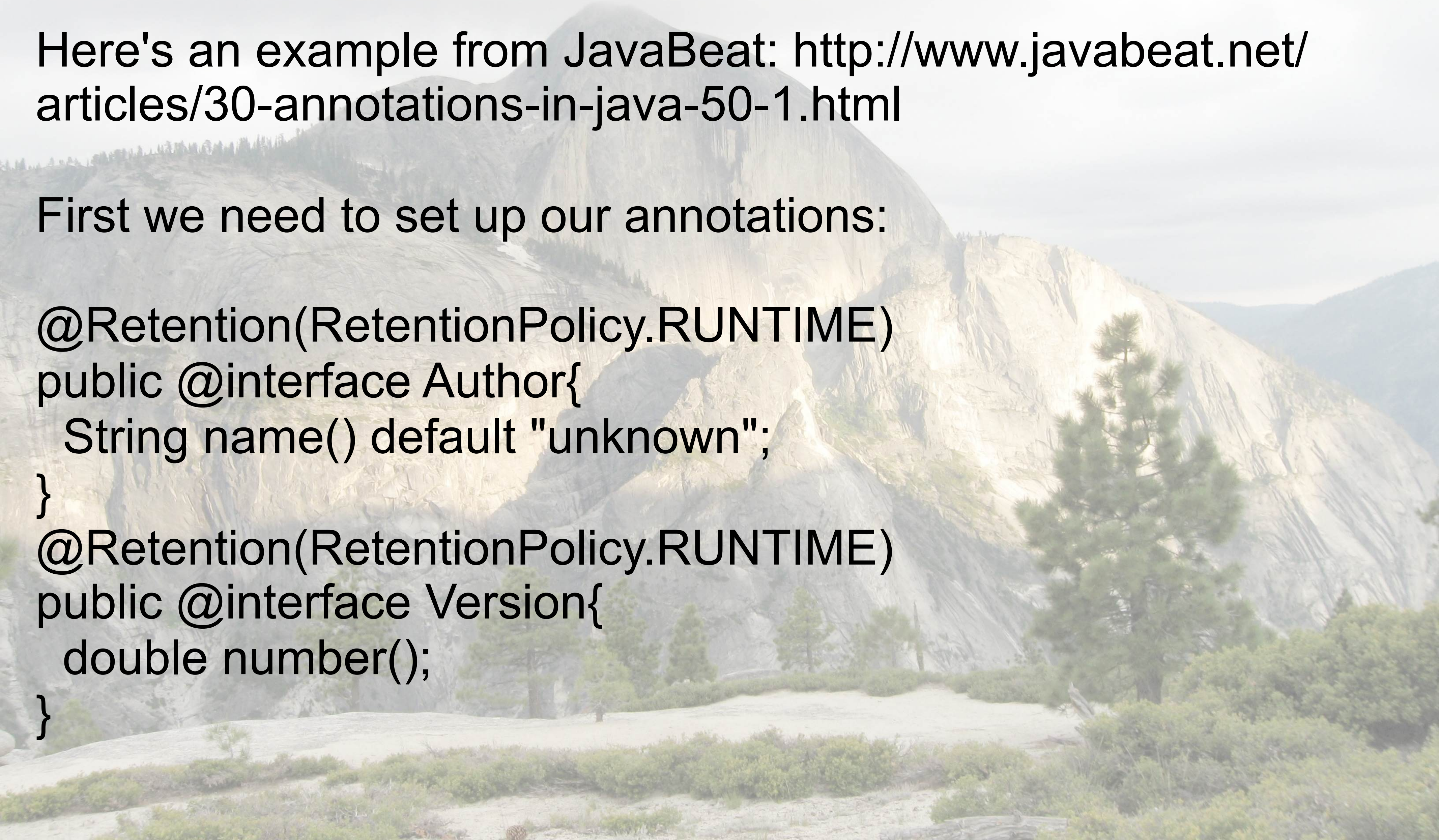
- This marks an annotation with limitations on how it can be used.
- @Target takes a list of ElementType enums to say how it can be used
- ElementType.ANNOTATION_TYPE, CONSTRUCTOR, FIELD, LOCAL_VARIABLE, METHOD, PACKAGE, PARAMETER, and TYPE.



Well, that's cool, but I already know how to use comments...

Annotations can be used in three ways:

- Assisting visual inspection of code
- Custom annotation parsing source tools (such as javadoc)
- Via reflection

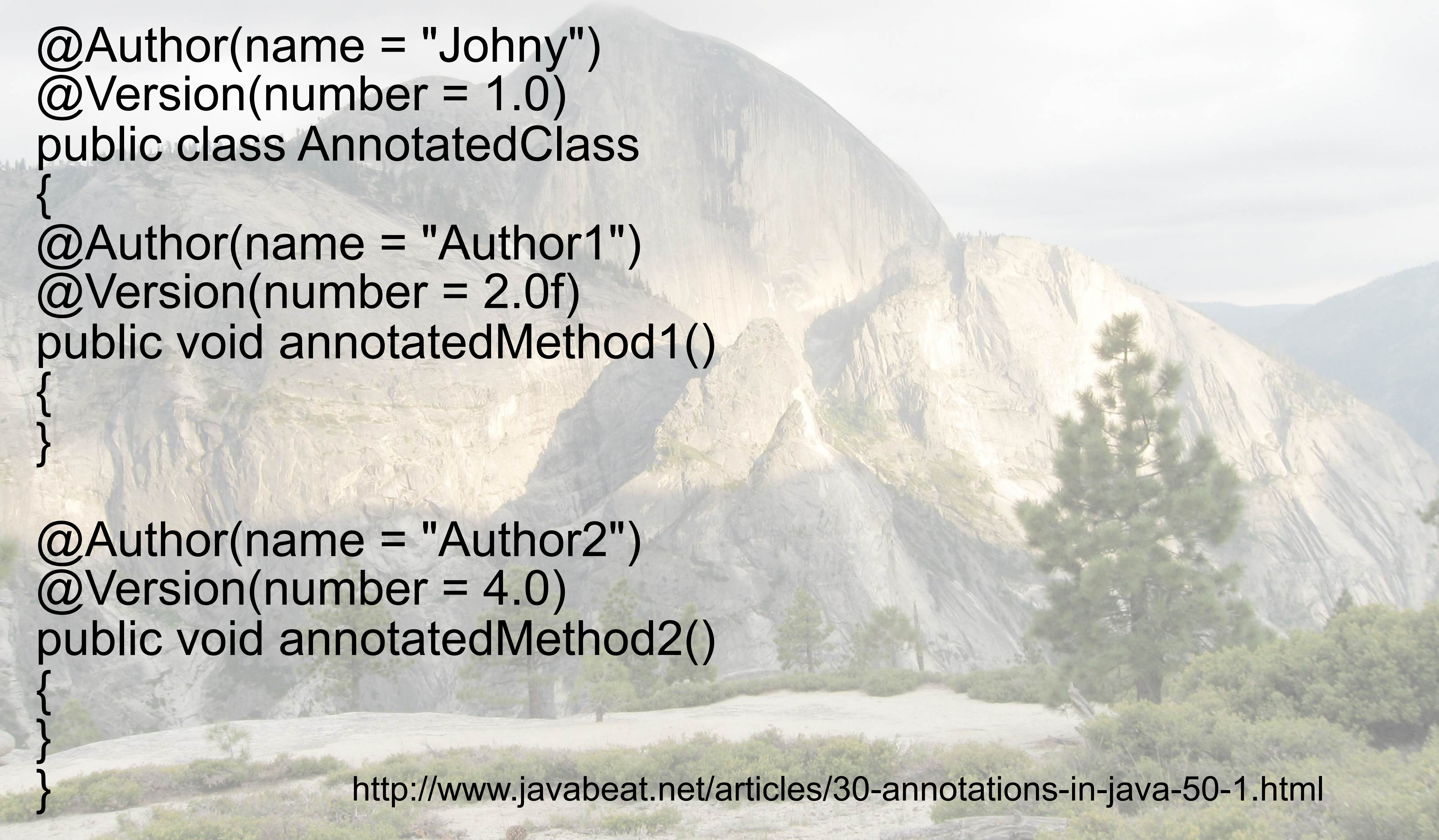


I do like mirrors! Tell me more about reflection.

Here's an example from JavaBeat: http://www.javabeat.net/articles/30-annotations-in-java-50-1.html

First we need to set up our annotations:

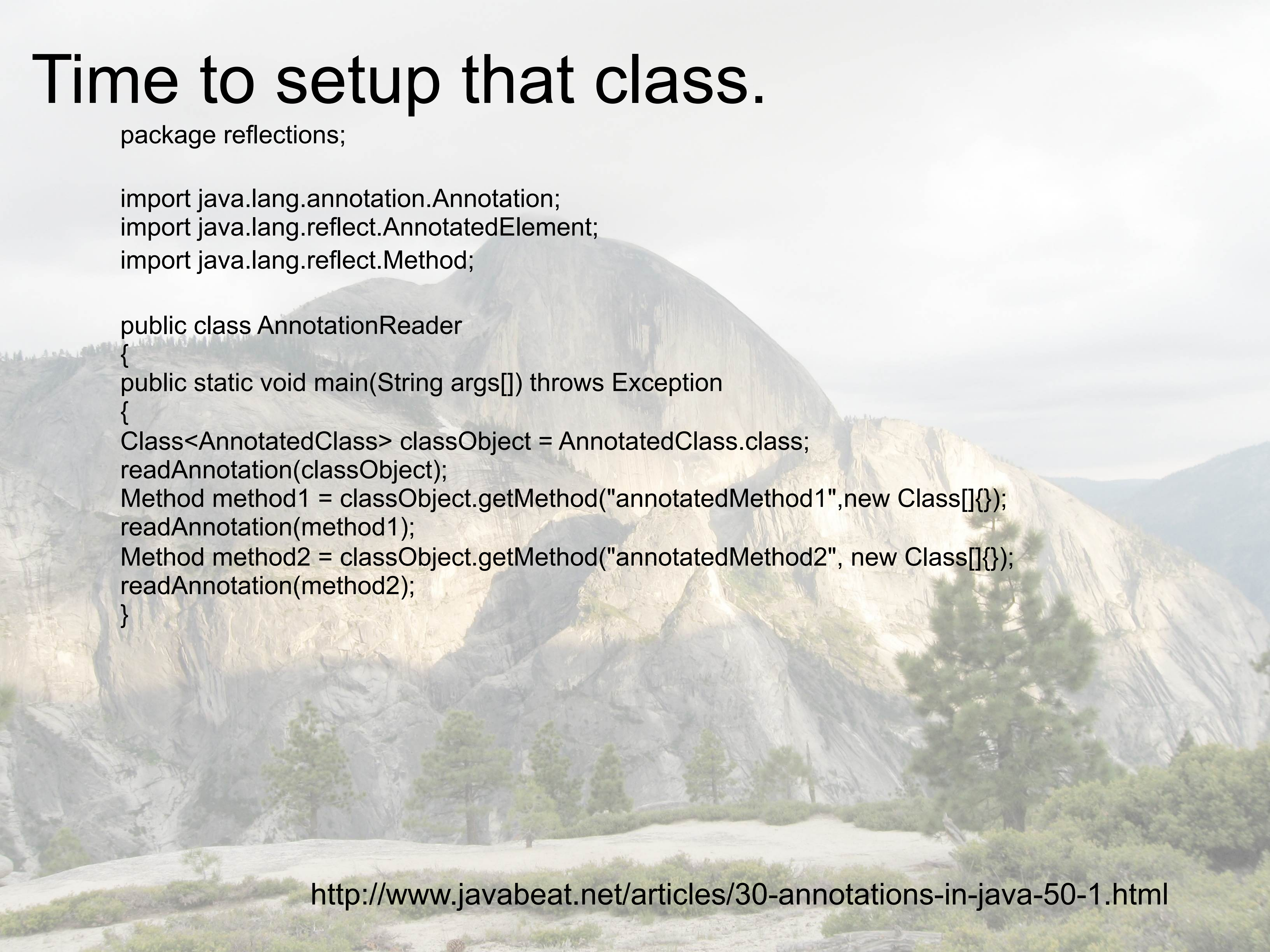
```
@Retention(RetentionPolicy.RUNTIME)
public @interface Author{
  String name() default "unknown";
}
@Retention(RetentionPolicy.RUNTIME)
public @interface Version{
  double number();
}
```



Now we need to apply those annotations to something.

```
@Author(name = "Johny")
@Version(number = 1.0)
public class AnnotatedClass
{
@Author(name = "Author1")
@Version(number = 2.0f)
public void annotatedMethod1()
{
}

@Author(name = "Author2")
@Version(number = 4.0)
public void annotatedMethod2()
{
}
}
```

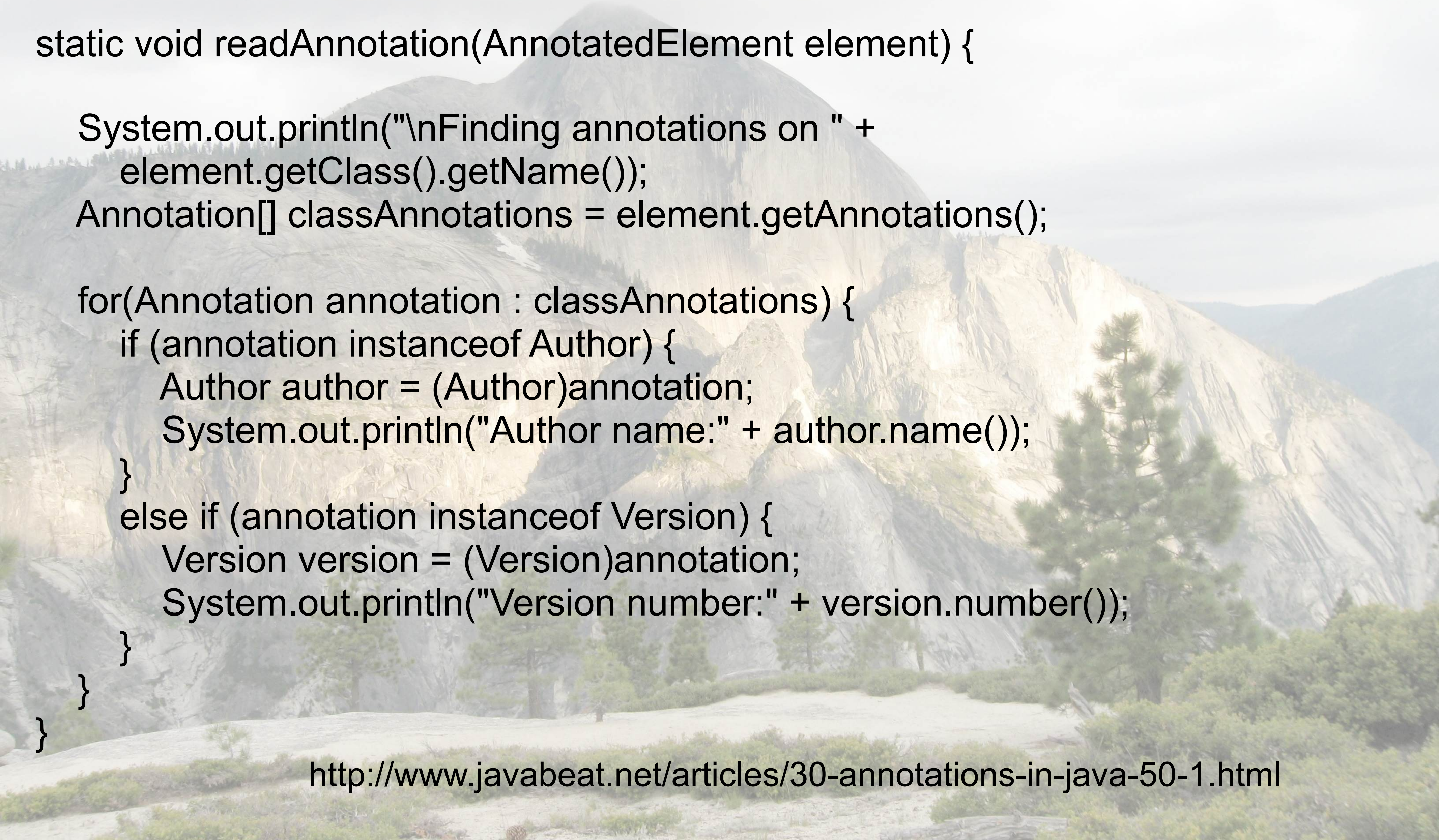
http://www.javabeat.net/articles/30-annotations-in-java-50-1.html



Time to setup that class.

```
package reflections;

import java.lang.annotation.Annotation;
import java.lang.reflect.AnnotatedElement;
import java.lang.reflect.Method;

public class AnnotationReader
{
public static void main(String args[]) throws Exception
{
Class<AnnotatedClass> classObject = AnnotatedClass.class;
readAnnotation(classObject);
Method method1 = classObject.getMethod("annotatedMethod1",new Class[]{});
readAnnotation(method1);
Method method2 = classObject.getMethod("annotatedMethod2", new Class[]{});
readAnnotation(method2);
}
```

http://www.javabeat.net/articles/30-annotations-in-java-50-1.html

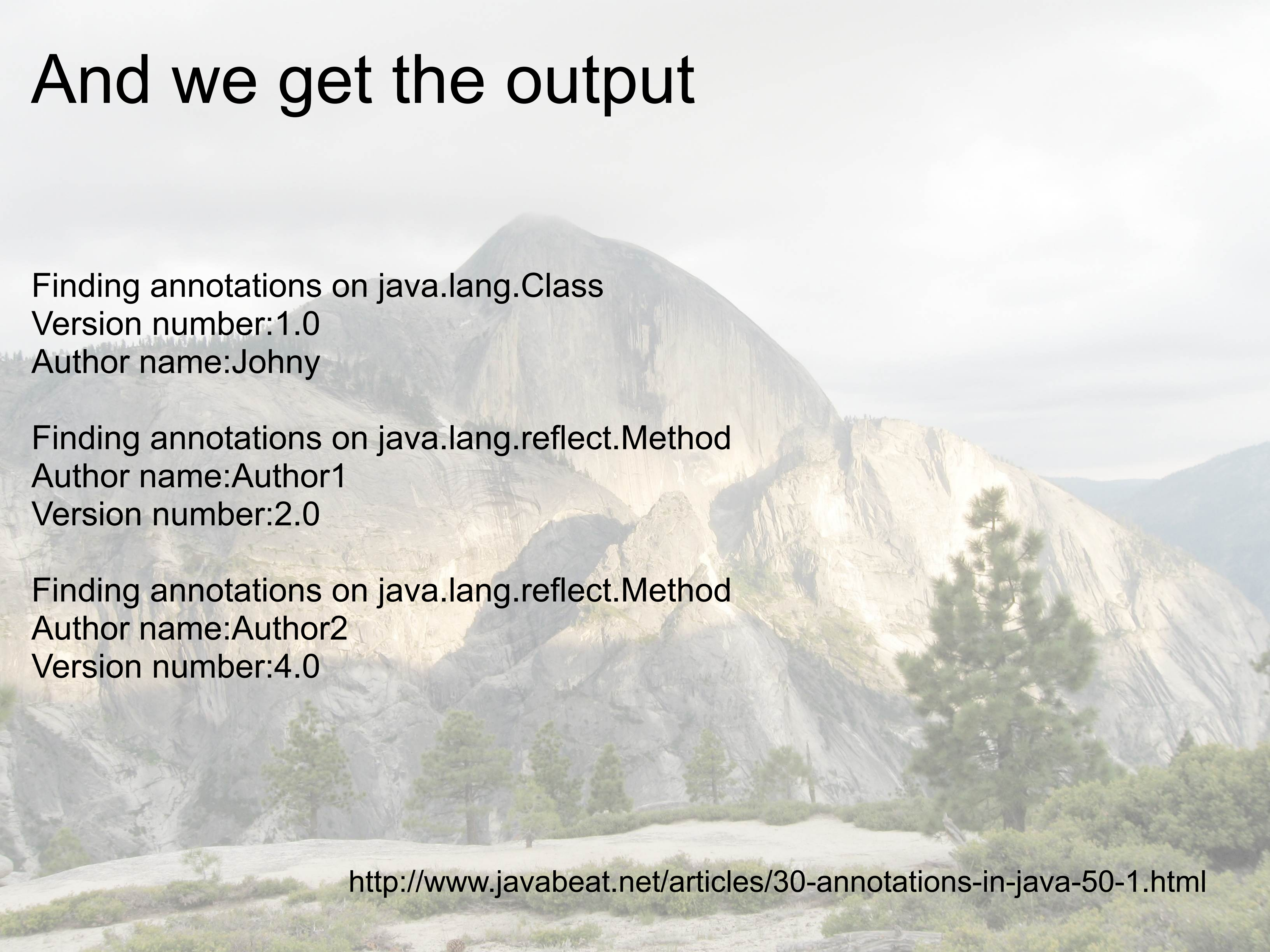


And use those annotations:

```
static void readAnnotation(AnnotatedElement element) {

    System.out.println("\nFinding annotations on " +
        element.getClass().getName());
    Annotation[] classAnnotations = element.getAnnotations();

    for(Annotation annotation : classAnnotations) {
        if (annotation instanceof Author) {
            Author author = (Author)annotation;
            System.out.println("Author name:" + author.name());
        }
        else if (annotation instanceof Version) {
            Version version = (Version)annotation;
            System.out.println("Version number:" + version.number());
        }
    }
}
```

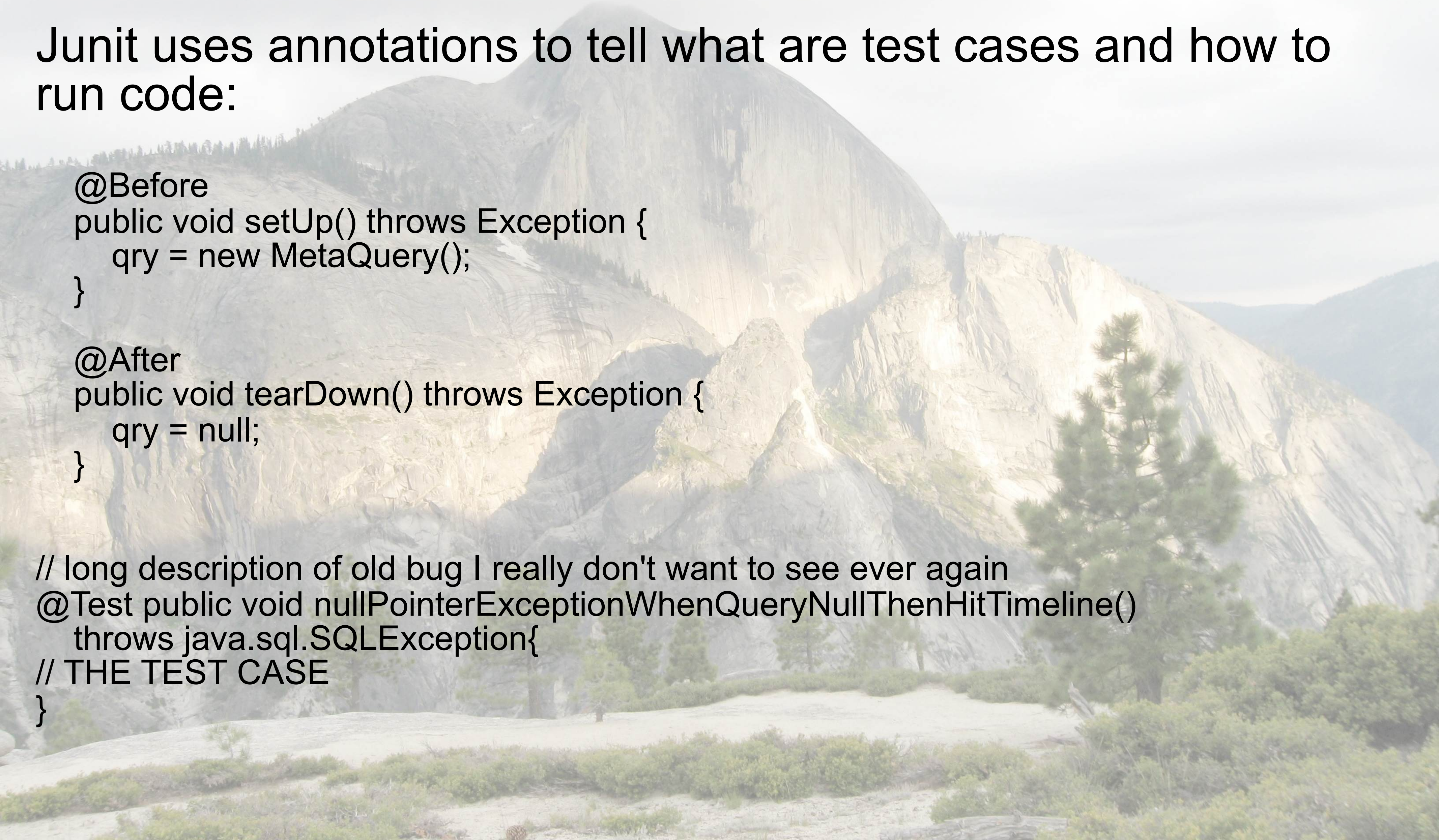
http://www.javabeat.net/articles/30-annotations-in-java-50-1.html

And we get the output



```
Finding annotations on java.lang.Class
Version number:1.0
Author name:Johny

Finding annotations on java.lang.reflect.Method
Author name:Author1
Version number:2.0

Finding annotations on java.lang.reflect.Method
Author name:Author2
Version number:4.0
```

http://www.javabeat.net/articles/30-annotations-in-java-50-1.html



Neat! Can we do things other than just track authors and version numbers?

Junit uses annotations to tell what are test cases and how to run code:

```
    @Before
    public void setUp() throws Exception {
        qry = new MetaQuery();
    }

    @After
    public void tearDown() throws Exception {
        qry = null;
    }


// long description of old bug I really don't want to see ever again
@Test public void nullPointerExceptionWhenQueryNullThenHitTimeline()
    throws java.sql.SQLException{
// THE TEST CASE
}
```

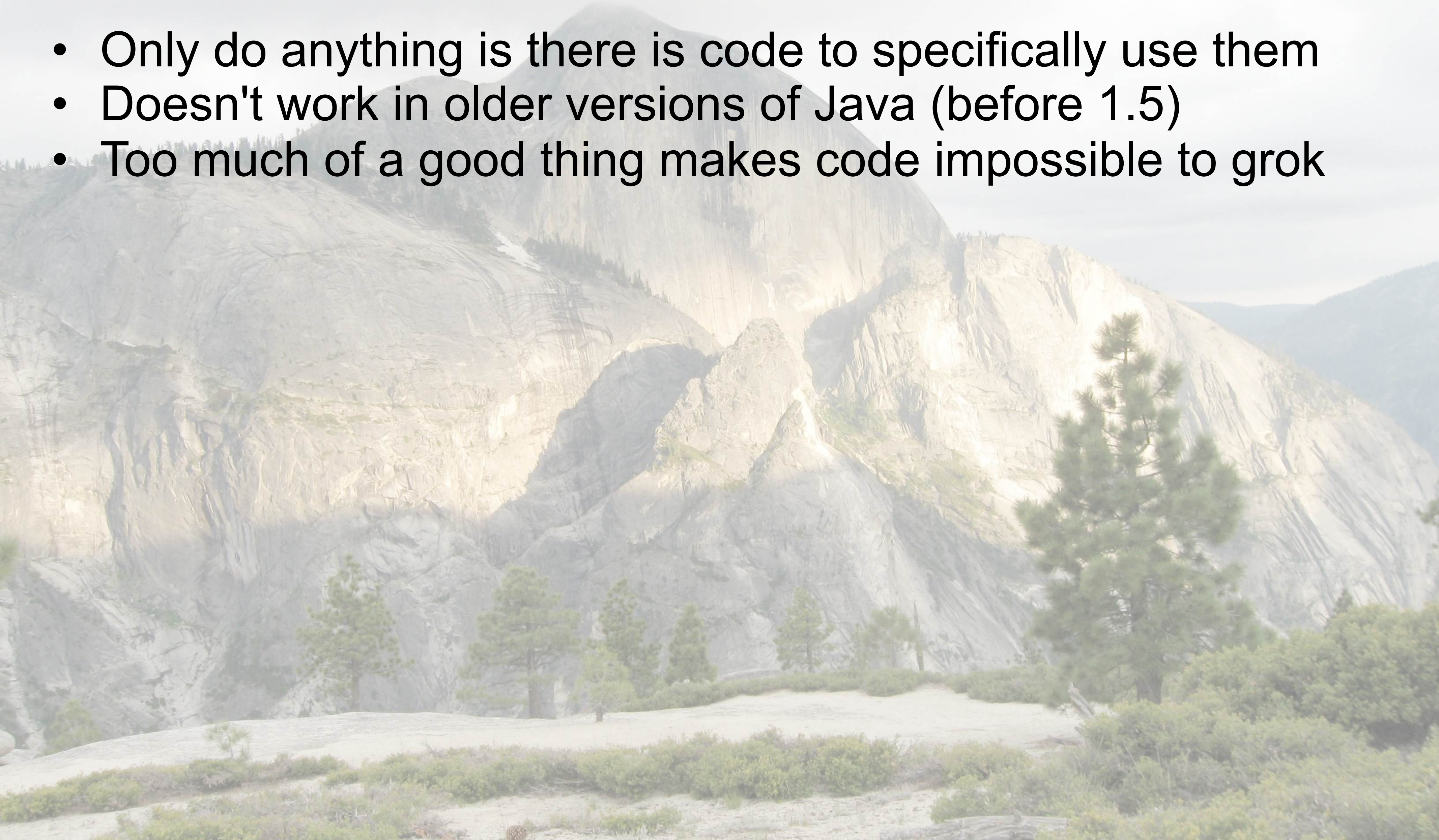
What about all that old hibernate code where we have more xml than code?

```
@Entity
class MedicalHistory implements Serializable {
  @Id @OneToOne
  @JoinColumn(name = "person_id")
  Person patient;
}


@Entity
public class Person implements Serializable {
  @Id @GeneratedValue Integer id;
}
```

See http://docs.jboss.org/hibernate/annotations

Annotation Limitations

- Only do anything is there is code to specifically use them
- Doesn't work in older versions of Java (before 1.5)
- Too much of a good thing makes code impossible to grok



Questions?

More information available

The book "Effective Java" http://www.amazon.com/Effective-Java-2nd-Joshua-Bloch/dp/0321356683

The Hibernate official documentation: http://www.hibernate.org/docs