What the heck are @Annotations?

Annotation

They are metadata that you can write into your Java program. They don't affect the execution of your code directly, but they can be processed by the compiler or at runtime to change the behavior of your code.

You should already have seen couple of common annotations such as @SupressWarnings(param), to get rid of those warning such as unused variables. You have to specify what kind of warnings you want to suppress, "unused" in this case to get rid of unused variable warnings.

Annotations can be added to classes, methods, variables, and even annotations themselves.

Creating custom annotation

To create your own annotation you would do something similar to creating your own class:

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.SOURCE)
public @interface VeryImportant {


}
```

Declaring the annotation is really all you need to create your own annotation, but there are two annotation that you would want to use to custom it further. Those are @Target, and @Retention.

@Target allow you to specify what kind of Java element this annotation is valid to be used for. For example, make it so that you can only apply this annotation to a class, to a method, to a variable, or even to an interface. If you want to specify multiple target you would use an array of ElementType.<Java element>.

```
@Target({ElementType.TYPE, ElementType.METHOD})
@Retention(RetentionPolicy.RUNTIME)
public @interface VeryImportant {
```

```
}
```

Retention policy

Tells Java when to discard the annotations.

@Retention, most of the time you would use RetentionPolicy.RUNTIME which means to keep this annotation around through the running of your program so that other code can inspect this annotation and process it such.

The other two possible value is SOURCE and CLASS. SOURCE will get rid of annotation before the compilation of your code. These are annotation that matter before the program is being compiled such as @SupressWarning. CLASS will keep your annotation through the compilation process, but once your program starts those annotation will be discarded.

Adding parameter to annotation

To add parameter to your annotation you have to declare them inside your annotation definition as a method, even though they are kind of accessed like a field:

```
@Target({ElementType.TYPE, ElementType.METHOD})
@Retention(RetentionPolicy.RUNTIME)
public @interface VeryImportant {
    String why();
}
```

For example, in this annotation we specified for the @VeryImportant annotation it has a required String parameter that you must passed into the annotation when you use it.

You can set up default values for each of the parameter of annotation, but if they don't have default value they are required.

The type of parameter for annotation are limited, they can only be primitives such as String, int, float, boolean, or an array type by adding brackets.

```
@Target({ElementType.TYPE, ElementType.METHOD})
@Retention(RetentionPolicy.RUNTIME)
public @interface VeryImportant {
    String why();
    int random() default 1;
    int[] nums;
}
```

Accessing parameter of annotation

To access the parameter that you have passed to the annotation you have to first retrieve the annotation object by calling `getAnnotation` on the object, method, or variable the thing that you annotated on.

Then you can just access it as a getter method, by calling on the name you have set in the annotation.

```
VeryImportant annotation = myCat.getAnnotation(VeryImportant.class);


annotation.why();
```

Processing annotation

The annotation by themselves, with @Target and @Retention doesn't really do much on their own. It is the code that you write that actually inspects the annotation that gives them special effects.

You can inspect class instances to check whether they have a particular annotation by calling `isAnnotationPresent(Annotation.class)`. For example:

```
Cat myCat = new Cat("Stella");


myCat.getClass().isAnnotationPresent(VeryImportant.class);
```

If the class `Cat` is annotated with the `@VeryImportant` annotation then this method will return `true`, otherwise if it is not annotated with `@VeryImportant` it will return false.

Java reflection

Java reflection is an API that is provided by Java to get metadata about an Java object. Things like getting the list of methods is inspecting the metadata about the Java object.

`getDeclaredMethods()` returns you a list of `Method` objects which you can print out which tells you the method name that this particular object has. In addition, you can also invoke them by calling `.invoke` on each of the `Method` objects.

`getDeclaredFields()` return you a list of `Field` object which are fields that you have inside your object. This is how you would be accessing the annotation that you did on each of the field of your object.

Revision #2
Created 3 February 2023 16:53:03 by Tamarine
Updated 10 March 2023 19:52:26 by Tamarine