val

Finally! Hassle-free final local variables.

`val` was introduced in lombok 0.10.

NEW in Lombok 1.18.22: `val` gets replaced with `final var`.

Overview

You can use `val` as the type of a local variable declaration instead of actually writing the type. When you do this, the type will be inferred from the initializer expression. The local variable will also be made final. This feature works on local variables and on foreach loops only, not on fields. The initializer expression is required.

`val` is actually a 'type' of sorts, and exists as a real class in the `lombok` package. You must import it for val to work (or use `lombok.val` as the type). The existence of this type on a local variable declaration triggers both the adding of the `final` keyword as well as copying the type of the initializing expression which overwrites the 'fake' `val` type.

WARNING: This feature does not currently work in NetBeans.

With Lombok

```java
import java.util.ArrayList;
import java.util.HashMap;
import lombok.val;

public class ValExample {
  public String example() {
    val example = new ArrayList<String>();
    example.add("Hello, World!");
    val foo = example.get(0);
    return foo.toLowerCase();
  }
  
  public void example2() {
    val map = new HashMap<Integer, String>();
    map.put(0, "zero");
    map.put(5, "five");
    for (val entry : map.entrySet()) {
      System.out.printf("%d: %s\n", entry.getKey(), entry.getValue());
    }
  }
}
```

Vanilla Java

```java
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;

public class ValExample {
  public String example() {
    final ArrayList<String> example = new ArrayList<String>();
    example.add("Hello, World!");
    final String foo = example.get(0);
    return foo.toLowerCase();
  }
  
  public void example2() {
    final HashMap<Integer, String> map = new HashMap<Integer, String>();
    map.put(0, "zero");
    map.put(5, "five");
    for (final Map.Entry<Integer, String> entry : map.entrySet()) {
      System.out.printf("%d: %s\n", entry.getKey(), entry.getValue());
    }
  }
}
```

Supported configuration keys:

`lombok.val.flagUsage` = [`warning` | `error`] (default: not set)
Lombok will flag any usage of `val` as a warning or error if configured.

Small print

For compound types, the most common superclass is inferred, not any shared interfaces. For example, `bool ? new HashSet() : new ArrayList()` is an expression with a compound type: The result is both `AbstractCollection` as well as `Serializable`. The type inferred will be `AbstractCollection`, as that is a class, whereas `Serializable` is an interface.

In ambiguous cases, such as when the initializer expression is `null`, `java.lang.Object` is inferred.

credits | Copyright © 2009-2024 The Project Lombok Authors, licensed under the MIT license.