@EqualsAndHashCode

Equality made easy: Generates `hashCode` and `equals` implementations from the fields of your object.

Overview

Any class definition may be annotated with `@EqualsAndHashCode` to let lombok generate implementations of the `equals(Object other)` and `hashCode()` methods. By default, it'll use all non-static, non-transient fields, but you can modify which fields are used (and even specify that the output of various methods is to be used) by marking type members with `@EqualsAndHashCode.Include` or `@EqualsAndHashCode.Exclude`. Alternatively, you can specify exactly which fields or methods you wish to be used by marking them with `@EqualsAndHashCode.Include` and using `@EqualsAndHashCode(onlyExplicitlyIncluded = true)`.

If applying `@EqualsAndHashCode` to a class that extends another, this feature gets a bit trickier. Normally, auto-generating an `equals` and `hashCode` method for such classes is a bad idea, as the superclass also defines fields, which also need equals/hashCode code but this code will not be generated. By setting `callSuper` to *true*, you can include the `equals` and `hashCode` methods of your superclass in the generated methods. For `hashCode`, the result of `super.hashCode()` is included in the hash algorithm, and for `equals`, the generated method will return false if the super implementation thinks it is not equal to the passed in object. Be aware that not all `equals` implementations handle this situation properly. However, lombok-generated `equals` implementations **do** handle this situation properly, so you can safely call your superclass equals if it, too, has a lombok-generated `equals` method. If you have an explicit superclass you are forced to supply some value for `callSuper` to acknowledge that you've considered it; failure to do so results in a warning.

Setting `callSuper` to *true* when you don't extend anything (you extend `java.lang.Object`) is a compile-time error, because it would turn the generated `equals()` and `hashCode()` implementations into having the same behaviour as simply inheriting these methods from `java.lang.Object`: only the same object will be equal to each other and will have the same hashCode. Not setting `callSuper` to *true* when you extend another class generates a warning, because unless the superclass has no (equality-important) fields, lombok cannot generate an implementation for you that takes into account the fields declared by your superclasses. You'll need to write your own implementations, or rely on the `callSuper` chaining facility. You can also use the `lombok.equalsAndHashCode.callSuper` config key.

Note that lombok just defers to the `.equals()` implementation for all objects *except* primitives and arrays. Some well known types have possibly surprising equals implementations. For example, `java.math.BigDecimal` considers scale, i.e. `1E2` is not equal to `100` according to `BigDecimal`'s own `equals` implementation.

CAUTION: It is easy to override the equals behaviour for any field by writing a method that returns a mapped value and annotating it with `@EqualsAndHashCode.Include(replaces = "fieldName")`. For example, to address the `BigDecimal` equality issue, you could write `@EqualsAndHashCode.Include BigDecimal fieldName() { return fieldName.stripTrailingZeros(); }`.

NEW in Lombok 0.10: Unless your class is `final` and extends `java.lang.Object`, lombok generates a `canEqual` method which means JPA proxies can still be equal to their base class, but subclasses that add new state don't break the equals contract. The complicated reasons for why such a method is necessary are explained in this paper: How to Write an Equality Method in Java. If all classes in a hierarchy are a mix of scala case classes and classes with lombok-generated equals methods, all equality will 'just work'. If you need to write your own equals methods, you should *always* override `canEqual` if you change `equals` and `hashCode`.

NEW in Lombok 1.14.0: To put annotations on the `other` parameter of the `equals` (and, if relevant, `canEqual`) method, you can use `onParam=@__({@AnnotationsHere})`. Be careful though! This is an experimental feature. For more details see the documentation on the onX feature.

NEW in Lombok 1.18.16: The result of the generated `hashCode()` can be cached by setting `cacheStrategy` to a value other than `CacheStrategy.NEVER`. *Do not* use this if objects of the annotated class can be modified in any way that would lead to the result of `hashCode()` changing.

With Lombok

```
import lombok.EqualsAndHashCode;

@EqualsAndHashCode
public class EqualsAndHashCodeExample {
  private transient int transientVar = 10;
  private String name;
  private double score;
  @EqualsAndHashCode.Exclude private Shape shape = new Square(5, 10);
  private String[] tags;
  @EqualsAndHashCode.Exclude private int id;

  public String getName() {
    return this.name;
  }

  @EqualsAndHashCode(callSuper=true)
  public static class Square extends Shape {
    private final int width, height;

    public Square(int width, int height) {
      this.width = width;
      this.height = height;
    }
  }
}
```

Vanilla Java

```
import java.util.Arrays;

public class EqualsAndHashCodeExample {
  private transient int transientVar = 10;
  private String name;
  private double score;
  private Shape shape = new Square(5, 10);
  private String[] tags;
  private int id;

  public String getName() {
    return this.name;
  }

  @Override public boolean equals(Object o) {
    if (o == this) return true;
    if (!(o instanceof EqualsAndHashCodeExample)) return false;
    EqualsAndHashCodeExample other = (EqualsAndHashCodeExample) o;
    if (!other.canEqual((Object)this)) return false;
    if (this.getName() == null ? other.getName() != null : !this.getName().equals(other.getName())) return
    if (Double.compare(this.score, other.score) != 0) return false;
    if (!Arrays.deepEquals(this.tags, other.tags)) return false;
    return true;
  }

  @Override public int hashCode() {
    final int PRIME = 59;
    int result = 1;
    final long temp1 = Double.doubleToLongBits(this.score);
    result = (result*PRIME) + (this.name == null ? 43 : this.name.hashCode());
    result = (result*PRIME) + (int)(temp1 ^ (temp1 >>> 32));
    result = (result*PRIME) + Arrays.deepHashCode(this.tags);
    return result;
  }

  protected boolean canEqual(Object other) {
    return other instanceof EqualsAndHashCodeExample;
  }

  public static class Square extends Shape {
    private final int width, height;

    public Square(int width, int height) {
      this.width = width;
      this.height = height;
    }

    @Override public boolean equals(Object o) {
      if (o == this) return true;
      if (!(o instanceof Square)) return false;
      Square other = (Square) o;
      if (!other.canEqual((Object)this)) return false;
      if (!super.equals(o)) return false;
      if (this.width != other.width) return false;
      if (this.height != other.height) return false;
      return true;
    }

    @Override public int hashCode() {
      final int PRIME = 59;
      int result = 1;
      result = (result*PRIME) + super.hashCode();
      result = (result*PRIME) + this.width;
      result = (result*PRIME) + this.height;
      return result;
    }

    protected boolean canEqual(Object other) {
      return other instanceof Square;
    }
  }
}
```

Supported configuration keys:

`lombok.equalsAndHashCode.doNotUseGetters = [ true | false ]` **(default: false)**
If set to `true`, lombok will access fields directly instead of using getters (if available) when generating `equals` and `hashCode` methods. The annotation parameter `'doNotUseGetters'`, if explicitly specified, takes precedence over this setting.
`lombok.equalsAndHashCode.callSuper = [ call | skip | warn ]` **(default: warn)**
If set to `call`, lombok will generate calls to the superclass implementation of `hashCode` and `equals` if your class extends something. If set to `skip` no such calls are generated. The default behaviour is like `skip`, with an additional warning.
`lombok.equalsAndHashCode.flagUsage = [ warning | error ]` **(default: not set)**
Lombok will flag any usage of `@EqualsAndHashCode` as a warning or error if configured.

Small print

Arrays are 'deep' compared/hashCoded, which means that arrays that contain themselves will result in `StackOverflowError`s. However, this behaviour is no different from e.g. `ArrayList`.

You may safely presume that the hashCode implementation used will not change between versions of lombok, however this guarantee is not set in stone; if there's a significant performance improvement to be gained from using an alternate hash algorithm, that will be substituted in a future version.

For the purposes of equality, 2 `NaN` (not a number) values for floats and doubles are considered equal, eventhough 'NaN == NaN' would return false. This is analogous to `java.lang.Double`'s equals method, and is in fact required to ensure that comparing an object to an exact copy of itself returns `true` for equality.

If there is *any* method named either `hashCode` or `equals`, regardless of return type, no methods will be generated, and a warning is emitted instead. These 2 methods need to be in sync with each other, which lombok cannot guarantee unless it generates all the methods, hence you always get a warning if one *or* both of the methods already exist. You can mark any method with `@lombok.experimental.Tolerate` to hide them from lombok.

Attempting to exclude fields that don't exist or would have been excluded anyway (because they are static or transient) results in warnings on the named fields.

If a method is marked for inclusion and it has the same name as a field, it replaces the field (the method is included, the field is excluded).

Prior to lombok 1.16.22, inclusion/exclusion could be done with the `of` and `exclude` parameters of the `@EqualsAndHashCode` annotation. This old-style inclusion mechanism is still supported but will be deprecated in the future.

By default, any variables that start with a $ symbol are excluded automatically. You can only include them by marking them with `@EqualsAndHashCode.Include`.

If a getter exists for a field to be included, it is called instead of using a direct field reference. This behaviour can be suppressed:
`@EqualsAndHashCode(doNotUseGetters = true)`

If you have configured a nullity annotation flavour via `lombok.config` key `lombok.addNullAnnotations`, the parameter of both the generated `equals` method as well as any `canEqual` method is annotated with a nullable annotation. This is required if you use a `@NonNullByDefault` style annotation in combination with strict nullity checking.