@Locked

Pop it and **lock** it! `ReentrantLock`, now with less hassle.

`@Locked` was introduced in lombok v1.20.

Overview

`@Locked` wraps all code in a method into a block that acquires a `java.util.concurrent.locks.ReentrantLock` first, and unlocks it when exiting the method. It is a lot like `@Synchronized`.

You can optionally name a field, which must be a `ReentrantLock`; in that case, lombok locks on that field. Otherwise, the annotation defaults to a field named `$LOCK` (on static methods) / `$lock` (on instance methods), which lombok will generate if the field does not exist yet.

Additionally, there are the `@Locked.Read` and `@Locked.Write` annotations. These use a `java.util.concurrent.locks.ReadWriteLock` (specifically, `ReentrantReadWriteLock`). Methods annotated with `@Locked.Write` will lock on the write lock and methods annotated with `@Locked.Read` will lock on the read lock. When required, a `java.util.concurrent.locks.ReentrantReadWriteLock` is generated.

When using Virtual threads (introduced in Java 20), these locks are recommended compared to what `@Synchronized` does.

With Lombok

```
import lombok.Locked;

public class LockedExample {
  private int value = 0;

  @Locked.Read
  public int getValue() {
    return value;
  }

  @Locked.Write
  public void setValue(int newValue) {
    value = newValue;
  }

  @Locked("baseLock")
  public void foo() {
    System.out.println("bar");
  }
}
```

Vanilla Java

```
public class LockedExample {
  private final ReadWriteLock lock = new ReentrantReadWriteLock();
  private final Lock baseLock = new ReentrantLock();
  private int value = 0;

  public int getValue() {
    this.lock.readLock().lock();
    try {
      return value;
    } finally {
      this.lock.readLock().unlock();
    }
  }

  public void setValue(int newValue) {
    this.lock.writeLock().lock();
    try {
      value = newValue;
    } finally {
      this.lock.writeLock().unlock();
    }
  }

  public void foo() {
    this.baseLock.lock();
    try {
      System.out.println("bar");
    } finally {
      this.baseLock.unlock();
    }
  }
}
```

Supported configuration keys:

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

Small print

Because `@Locked.Read` and `@Locked.Write` use a different type of lock than `@Locked`, these annotations cannot be used on the same lock object without explicitly specifying the name of the field containing the lock object.
The name of the default field of the `@Locked`, `@Locked.Read`, and `@Locked.Write` annotations is the same, so it is not possible to mix the basic `@Locked` annotation with the other two using the default name.