@Cleanup

Automatic resource management: Call your `close()` methods safely with no hassle.

Overview

You can use `@Cleanup` to ensure a given resource is automatically cleaned up before the code execution path exits your current scope. You do this by annotating any local variable declaration with the `@Cleanup` annotation like so:
`@Cleanup InputStream in = new FileInputStream("some/file");`
As a result, at the end of the scope you're in, `in.close()` is called. This call is guaranteed to run by way of a try/finally construct. Look at the example below to see how this works.

If the type of object you'd like to cleanup does not have a `close()` method, but some other no-argument method, you can specify the name of this method like so:
`@Cleanup("dispose") org.eclipse.swt.widgets.CoolBar bar = new CoolBar(parent, 0);`
By default, the cleanup method is presumed to be `close()`. A cleanup method that takes 1 or more arguments cannot be called via `@Cleanup`.

With Lombok

```
import lombok.Cleanup;
import java.io.*;

public class CleanupExample {
  public static void main(String[] args) throws IOException {
    @Cleanup InputStream in = new FileInputStream(args[0]);
    @Cleanup OutputStream out = new FileOutputStream(args[1]);
    byte[] b = new byte[10000];
    while (true) {
      int r = in.read(b);
      if (r == -1) break;
      out.write(b, 0, r);
    }
  }
}
```

Vanilla Java

```
import java.io.*;

public class CleanupExample {
  public static void main(String[] args) throws IOException {
    InputStream in = new FileInputStream(args[0]);
    try {
      OutputStream out = new FileOutputStream(args[1]);
      try {
        byte[] b = new byte[10000];
        while (true) {
          int r = in.read(b);
          if (r == -1) break;
          out.write(b, 0, r);
        }
      } finally {
        if (out != null) {
          out.close();
        }
      }
    } finally {
      if (in != null) {
        in.close();
      }
    }
  }
}
```

Supported configuration keys:

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

Small print

In the finally block, the cleanup method is only called if the given resource is not `null`. However, if you use `delombok` on the code, a call to `lombok.Lombok.preventNullAnalysis(Object o)` is inserted to prevent warnings if static code analysis could determine that a null-check would not be needed. Compilation with `lombok.jar` on the classpath removes that method call, so there is no runtime dependency.

If your code throws an exception, and the cleanup method call that is then triggered also throws an exception, then the original exception is hidden by the exception thrown by the cleanup call. You should *not* rely on this 'feature'. Preferably, lombok would like to generate code so that, if the main body has thrown an exception, any exception thrown by the close call is silently swallowed (but if the main body exited in any other way, exceptions by the close call will not be swallowed). The authors of lombok do not currently know of a feasible way to implement this scheme, but if java updates allow it, or we find a way, we'll fix it.

You do still need to handle any exception that the cleanup method can generate!