

Inspections

[Previous](#) | [Find Usages](#)

[Top](#) | [Quick Start](#)

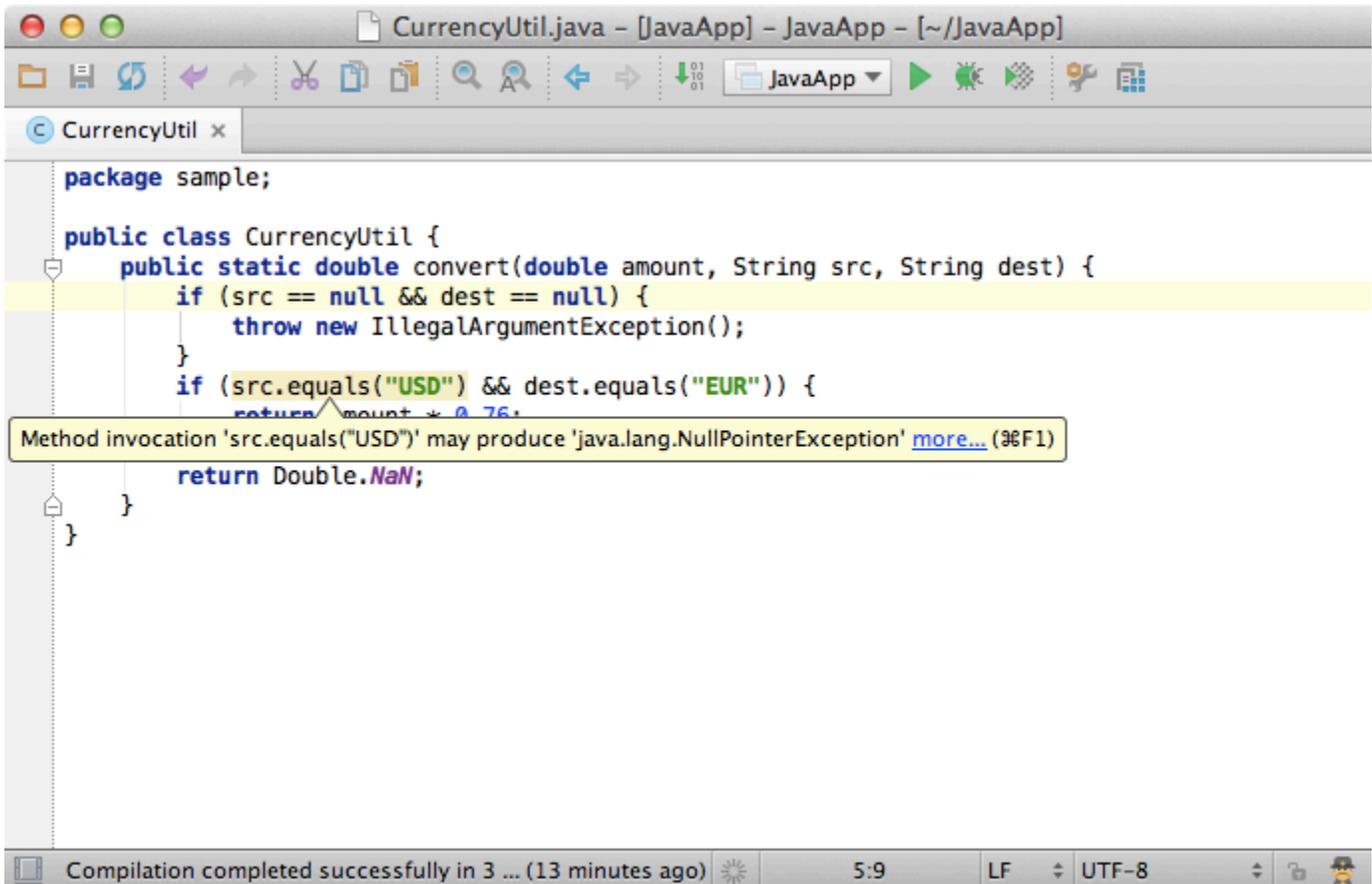
[Next](#) | [Code Style and Formatting](#)

 **Redirection Notice**
This page will redirect to <https://www.jetbrains.com/idea/help/code-inspection.html>.

Inspections are built-in static code analysis tools that help you find probable bugs, locate dead code, detect performance issues and improve the overall code structure.

1. On-the-fly code analysis

Most of the inspections run on-the-fly and display warnings or errors in the editor immediately as you type.



```
package sample;

public class CurrencyUtil {
    public static double convert(double amount, String src, String dest) {
        if (src == null && dest == null) {
            throw new IllegalArgumentException();
        }
        if (src.equals("USD") && dest.equals("EUR")) {
            return amount * 0.76;
        }
        return Double.NaN;
    }
}
```

Method invocation 'src.equals("USD")' may produce 'java.lang.NullPointerException' [more...](#) (⌘F1)

Compilation completed successfully in 3 ... (13 minutes ago) 5:9 LF UTF-8

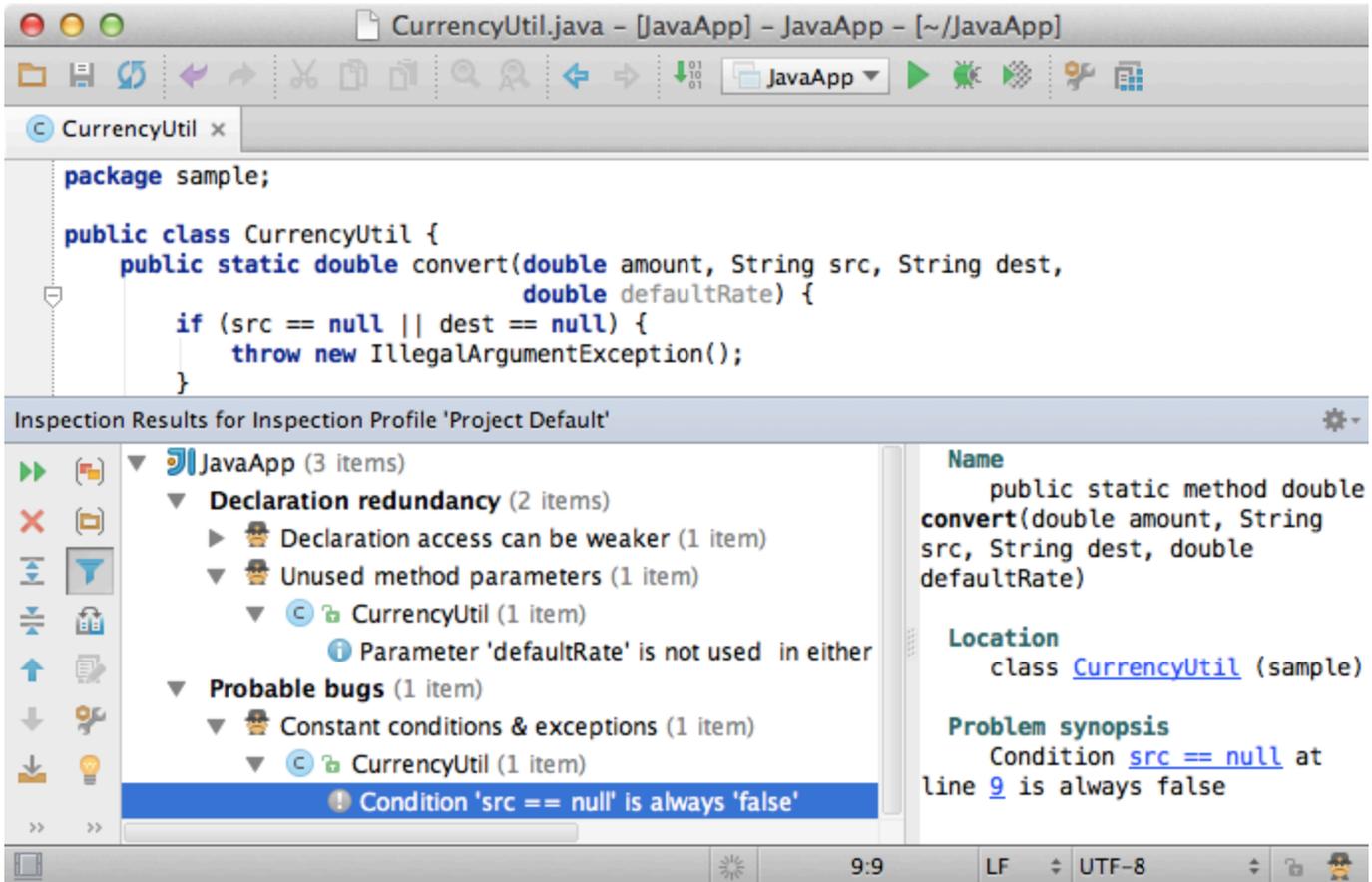
Inspections that are too complex to be run on on-the-fly are available when you perform code analysis for the entire project via `Analyze > Inspect Code` menu, or when you run certain inspection by its name via `Analyze > Run Inspection by Name`.

2. Navigate to the next/previous problem

The editor lets you quickly navigate between the highlighted problems via keyboard shortcuts. Press `F2` to go to the next problem and `Shift+F2` to go to the previous one. You can configure the severity of the problems through which the `F2` key navigates in the `Settings > Editor` and tell it, for example, to always jump to the most serious issue in a file.

3. Run project-wide inspections

To get complete report with the inspection results for an entire project, use the `Analyze > Inspect Code` command.



4. Quick-fixes

Most inspections not only tell you where a problem is, but provide quick-fixes to deal with them right away. Just press Alt + Enter and choose a quick-fix.

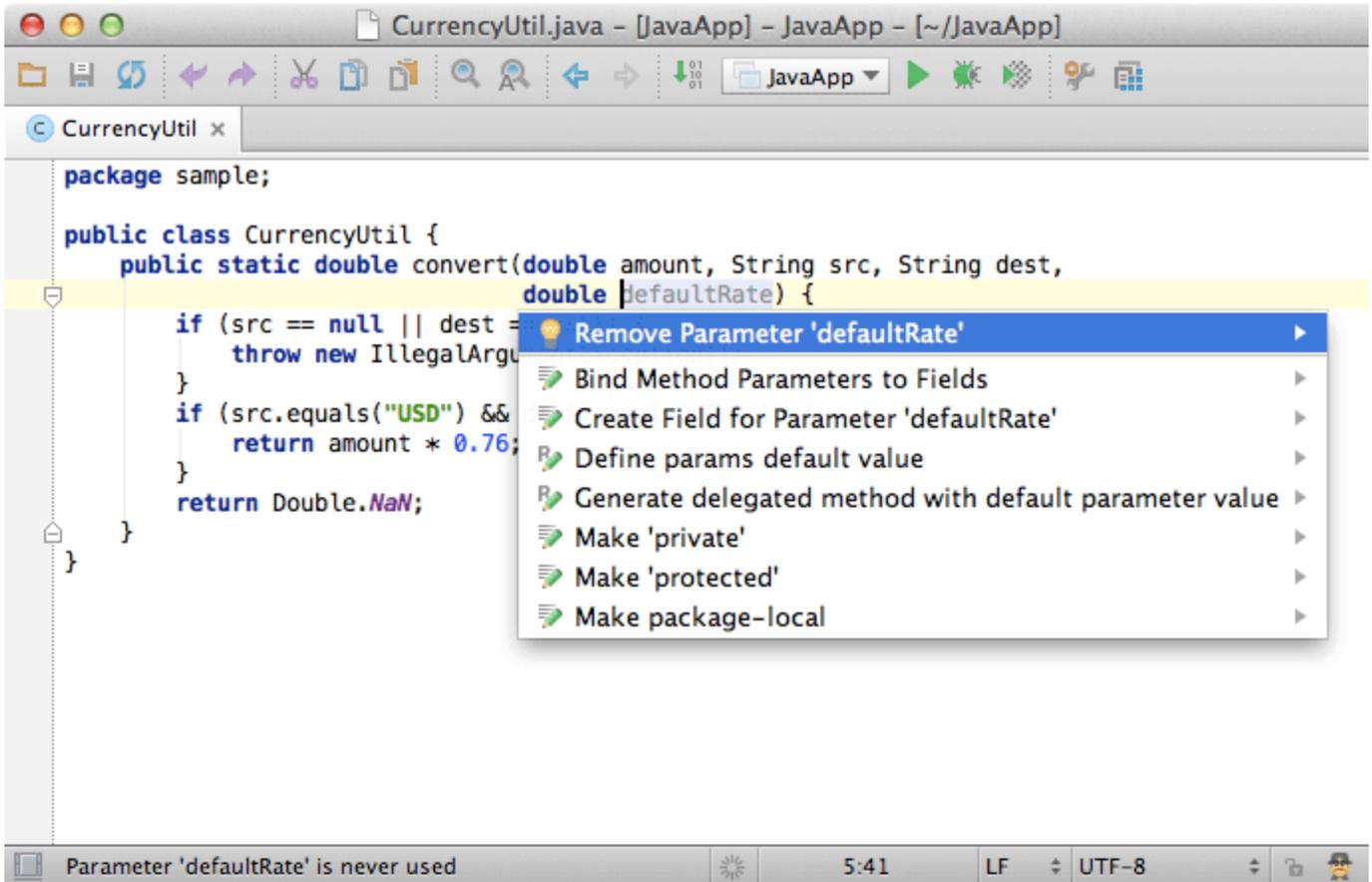
```
package sample;

public class CurrencyUtil {
    public static double convert(double amount, String src, String dest,
                                double defaultRate) {
        if (src == null || dest == null) {
            throw new IllegalArgumentException("Parameter 'defaultRate' is never used");
        }
        if (src.equals("USD") && dest.equals("EUR")) {
            return amount * 0.76;
        }
        return Double.NaN;
    }
}
```

The screenshot shows an IDE window titled "CurrencyUtil.java - [JavaApp] - JavaApp - [~/JavaApp]". The code editor displays the following Java code. A yellow highlight is under the line `double defaultRate` in the method signature. A tooltip points to the `IllegalArgumentException` message, which says "Parameter 'defaultRate' is never used". The status bar at the bottom indicates "Compilation completed successfully in 3 ... (15 minutes ago)", "6:27", "LF", and "UTF-8".

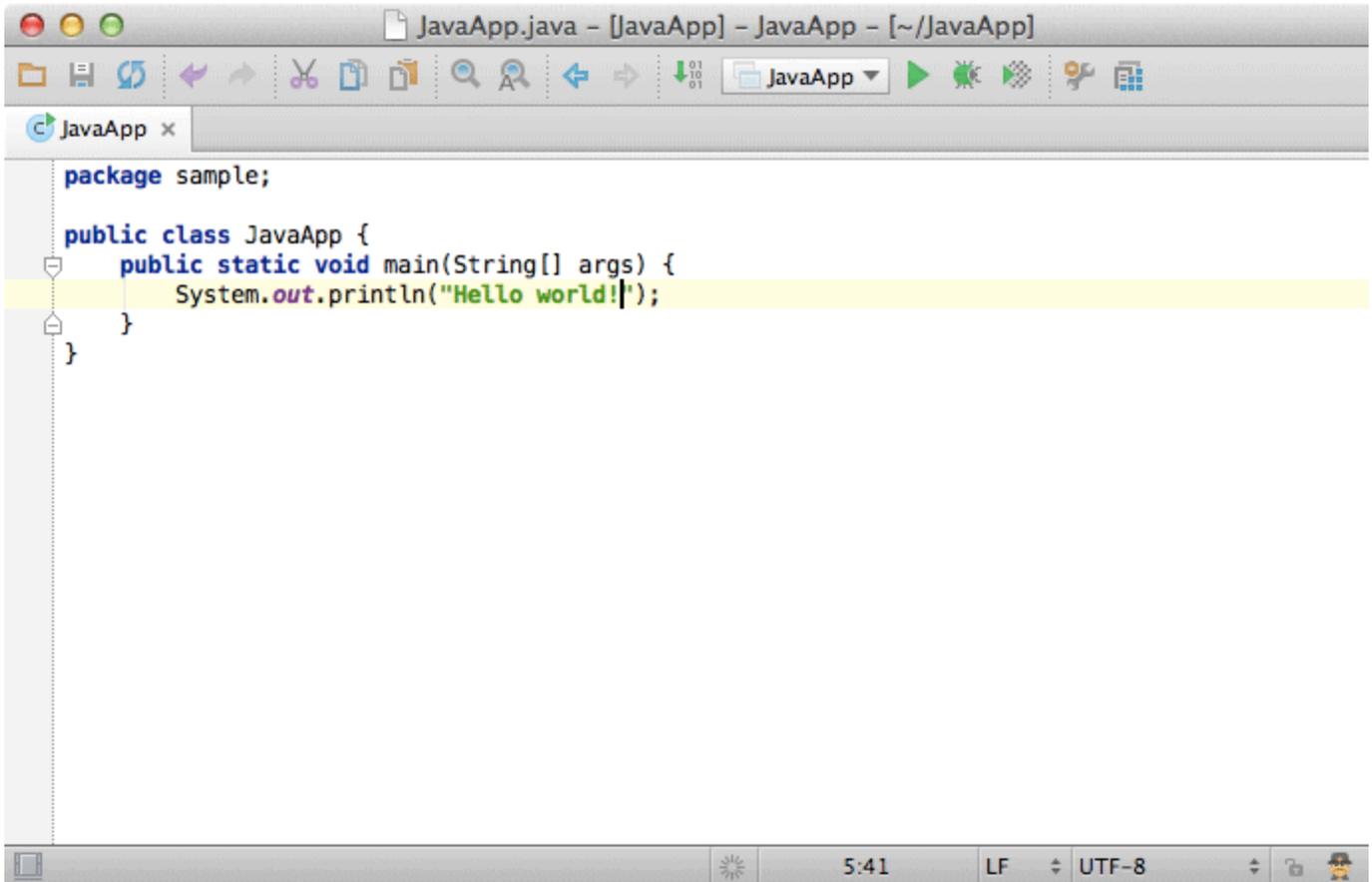
5. Suppress warnings

When you don't want warnings from this or that inspection for a specific statement or method, you can simply suppress them by pressing the right arrow on a quick-fix. Sometimes it may be a better solution than disabling the inspection for an entire project.



6. Run a single inspection

To run a single inspection by its name just press Shift + Alt + Ctrl + I (Shift + Alt + Cmd + I for Mac) shortcut or use Analyze Run Inspection by Name.

A screenshot of an IDE window titled "JavaApp.java - [JavaApp] - JavaApp - [~/JavaApp]". The window contains a Java code snippet with syntax highlighting. The code is as follows:

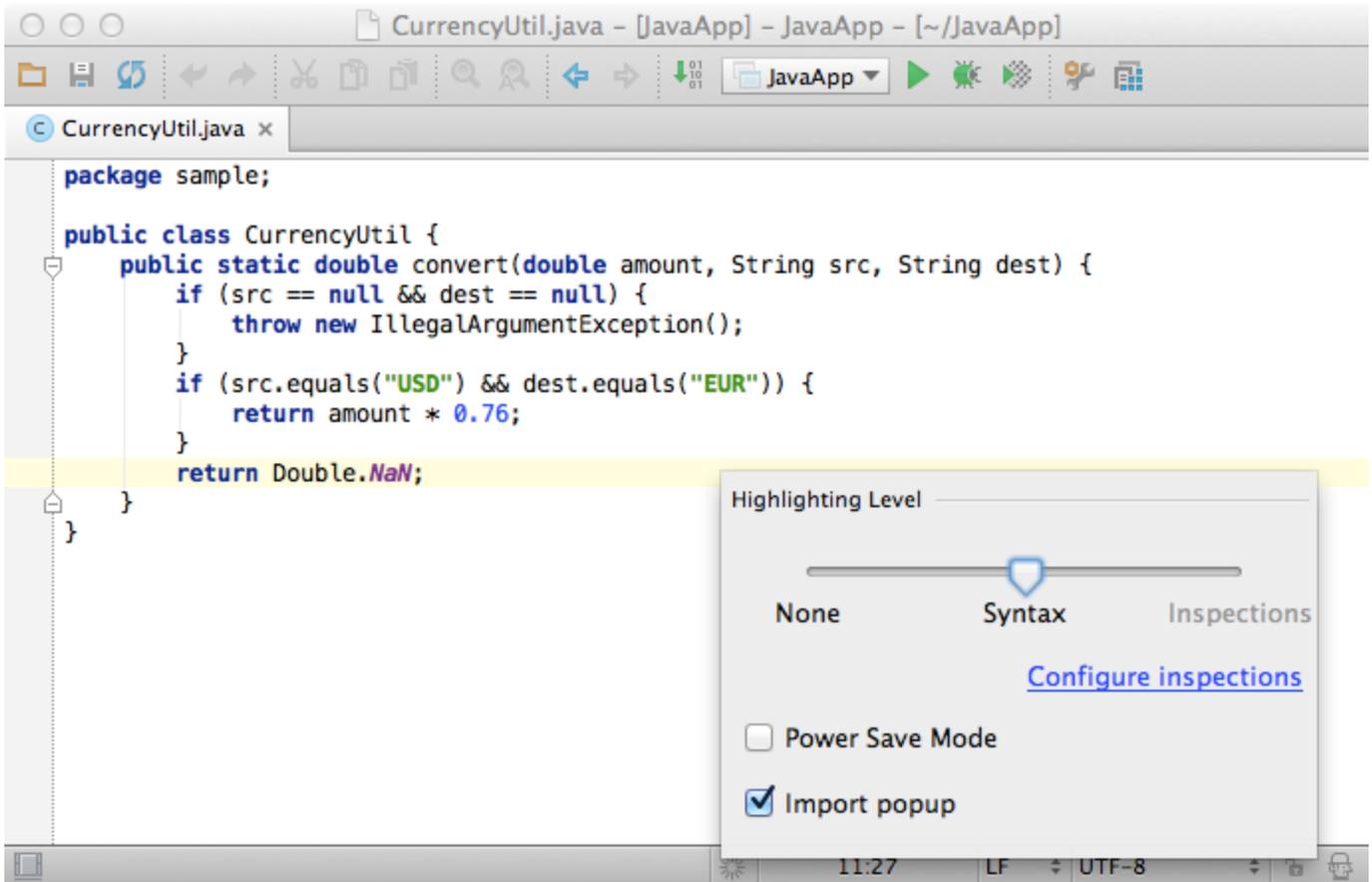
```
package sample;

public class JavaApp {
    public static void main(String[] args) {
        System.out.println("Hello world!");
    }
}
```

The line containing the `System.out.println("Hello world!");` statement is highlighted in yellow. The IDE interface includes a toolbar with various icons for file operations, search, and execution. The status bar at the bottom shows the time "5:41", the line format "LF", and the encoding "UTF-8".

7. Highlighting level and power save mode

Another useful thing is the Hektor icon located on the status bar. It helps you switch highlighting levels: from none to syntax and inspections. At syntax level you don't get any warnings except when code can't be compiled.



The Hektor icon also lets you activate power save mode that helps conserve battery power by disabling highlighting and auto-popups.

8. Inspection profiles

And finally you can change the list of inspections enabled for a project by managing the inspection profiles via Settings. Inspect inspections. If you want to share your inspection profile with your team, enable the Share profile checkbox, and check the inspections project files into VCS.

Inspections

Project Default

Share profile

Add

Copy

Import

Export



Search

Description

- Memory issues
- Method metrics
- Modularization issues
- Naming conventions
- Numeric issues
- OSGi
- Packaging issues
- Pattern Validation
- Performance issues
 - Boolean constructor call
 - Boxing of already boxed value
 - Call to 'Arrays.asList()' with too few arguments
 - Call to 'Collection.toArray()' with zero-length array argument
 - Call to 'intern()' on String constant
 - Call to simple getter from within class
 - Call to simple setter from within class
 - Class initializer may be 'static'

Reports boxing of already boxed values. This is a useless operation since any boxed value will first be auto-unboxed before boxing the value again. If done inside an inner loop such code may cause performance problems.

Options

Severity: Warning

[Previous](#) | [Find Usages](#)

[Top](#) | [Quick Start](#)

[Next](#) | [Code Style and Formatting](#)