


# Code Completion

[Previous](#) | Editor Basics

[Top](#) | Quick Start

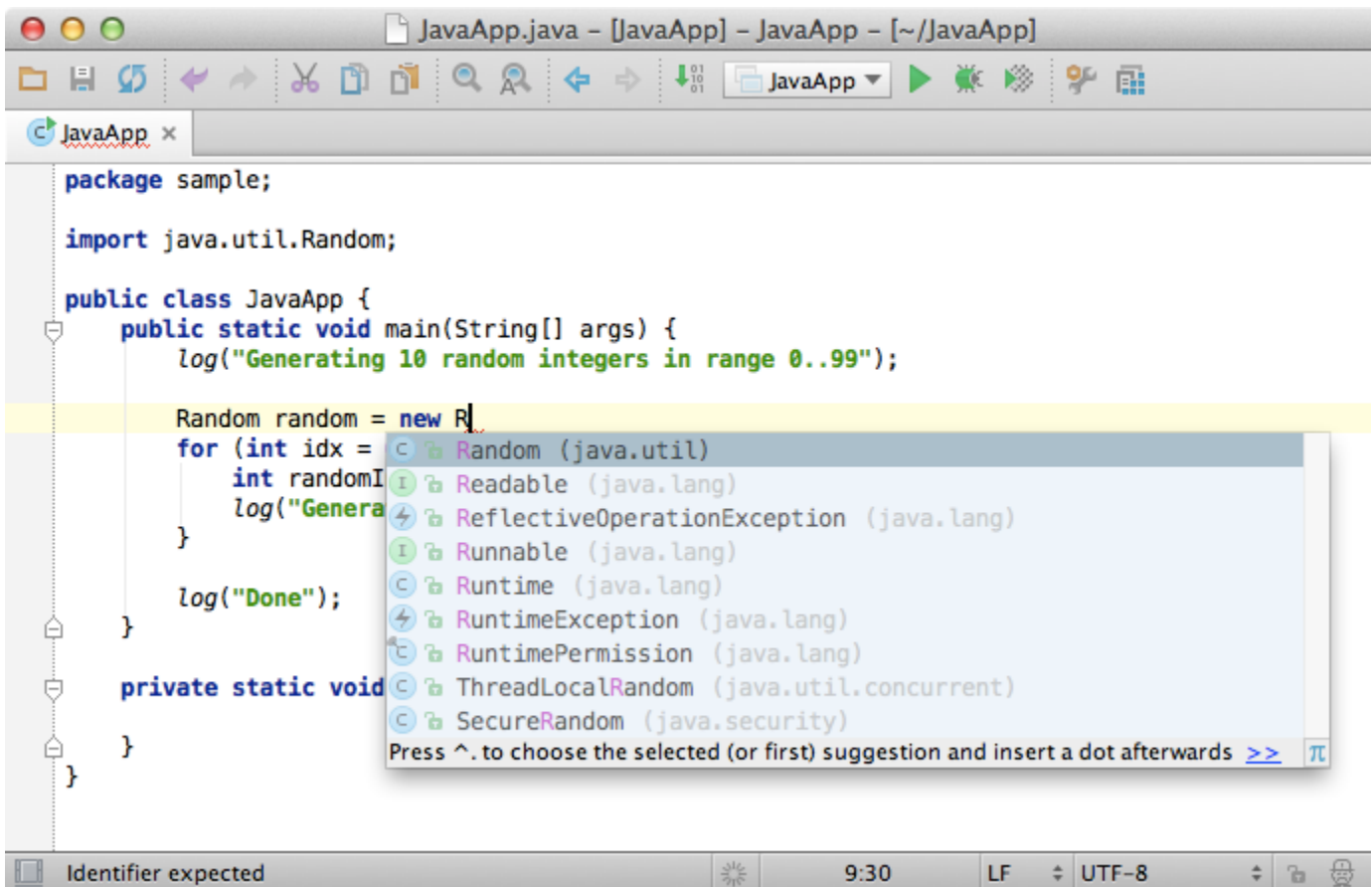
[Next](#) | Navigation

The license could not be verified: License Certificate has expired!

 Focused on developer productivity, IntelliJ IDEA provides powerful features for code completion. However, as there are several code completion features, it is important to learn how they work and when it's best to use each one.

## 1. Basic completion

Basic completion offers most trivial suggestions for variables, types, methods, expressions, etc. It is worth mentioning that IntelliJ IDEA provides Basic completion automatically when you start typing. This means you don't need to press any shortcuts to see suggestions. But if you want to call it explicitly, simply press `Ctrl+Space` shortcut.



Note, when you call Basic completion twice, it shows you more results, including private members.

## 2. Smart completion

Additionally to Basic completion, IntelliJ IDEA provides Smart completion which is much more advanced and comprehensive. Smart completion is aware of the expected type and data flow and offers the option relevant to the context. To call Smart completion, press `Shift+Ctrl+Space`.

```
import java.util.Random;

public class JavaApp {
    public static void main(String[] args) {
        log("Generating 10 random integers in range 0..99");

        Random random = new R
        for (int idx = 0; idx < 10; idx++) {
            int randomInt = random.nextInt();
            log("Generated random integer: " + randomInt);
        }

        log("Done");
    }

    private static void log(String message) {
    }
}
```

'(' or '[' expected

9:30 LF UTF-8

Note that when you call Smart completion twice, it shows you more results, including chains and non-imported static members.

```
package sample;

import java.util.Random;

public class JavaApp {
    public static void main(String[] args) {
        log("Generating 10 random integers in range 0..99");

        Random random = new Random();
        for (int idx = 0; idx < 10; idx++) {
            int randomInt = random.nextInt();
            log("Generated random integer: " + randomInt);
        }

        log("Done");
    }

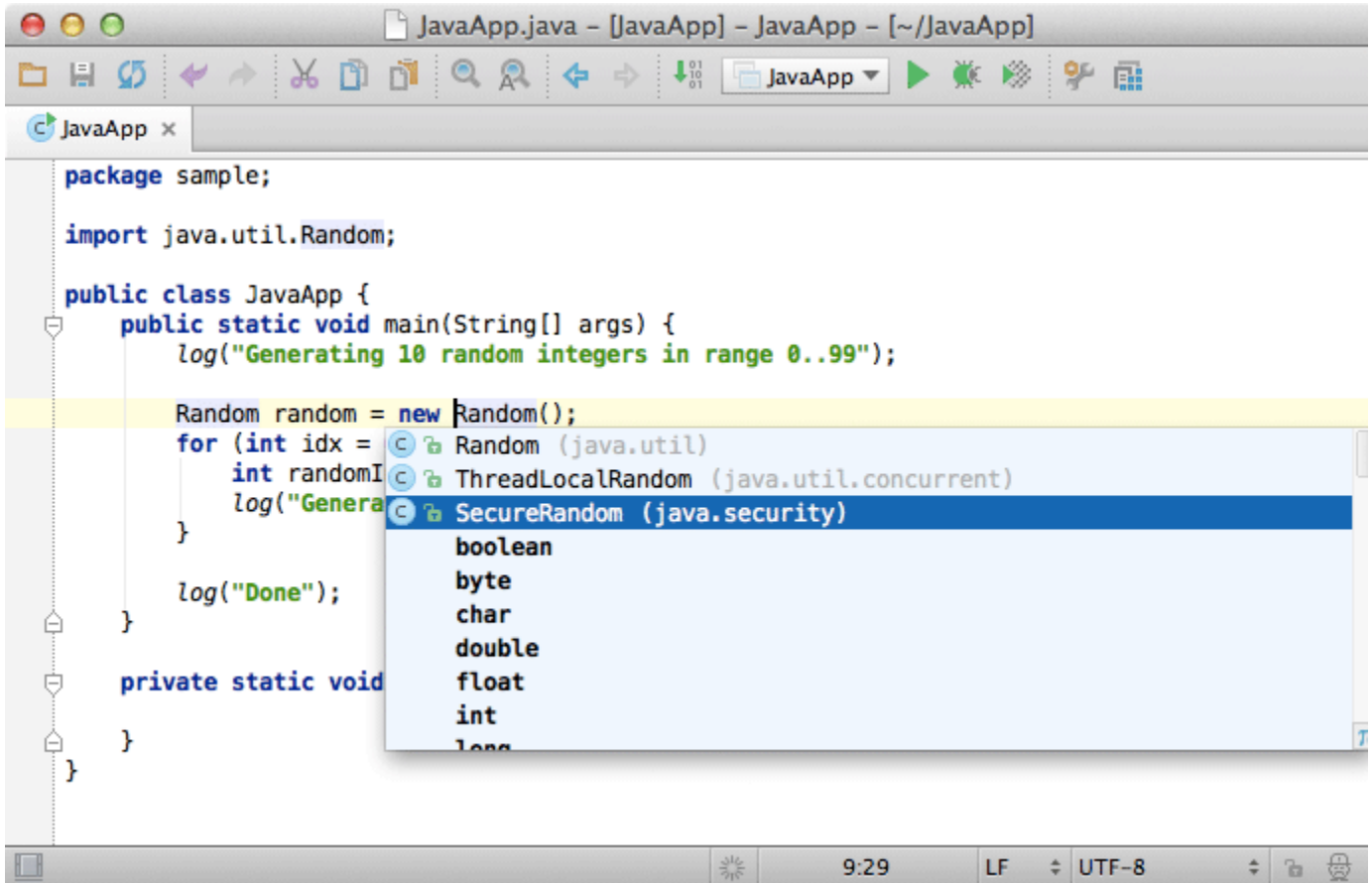
    private static void log(String message) {
    }
}
```

Expression expected

11:29 LF UTF-8

### 3. Completion with Tab

If you select an item from the suggestion list by clicking Tab, it will overwrite the identifier at the caret, instead of just inserting the suggestion. This is helpful if you're editing a part of an identifier, such as a file name.



### 4. Statement completion

Statement completion automatically adds missing parentheses, brackets, braces and adds the necessary formatting. To complete a statement, just press Shift+Ctrl+Enter (Shift+Cmd+Enter for Mac).

```
package sample;

import java.util.Random;

public class JavaApp {
    public static void main(String[] args) {
        log("Generating 10 random integers in range 0..99");

        Random random = new Random();
        for (int idx = 0; idx < 10; idx++) {
            int randomInt = random.nextInt();
            log("Generated: " + randomInt);
        }

        log("Done");
    }

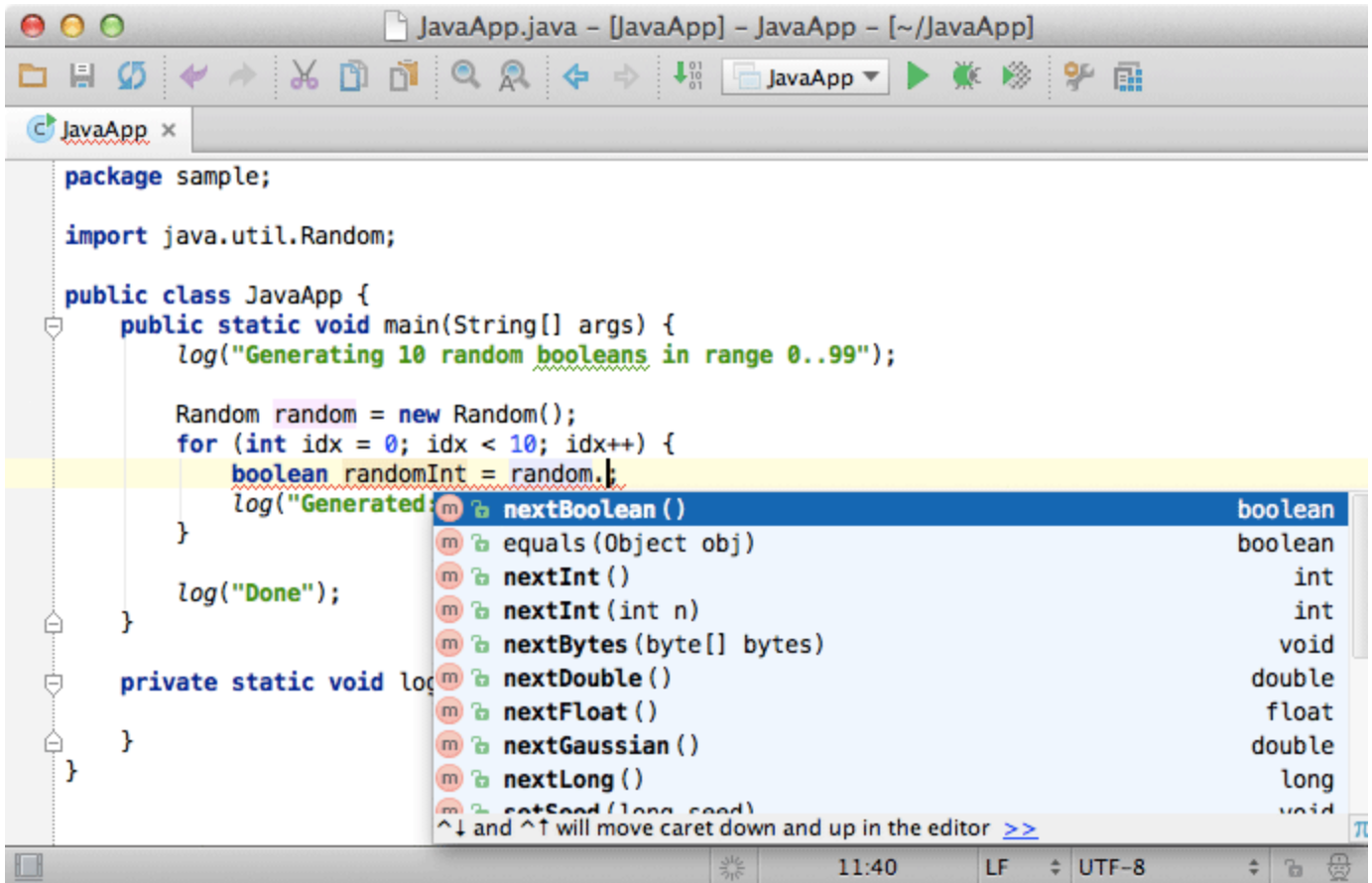
    private static void log(String message)
}
```

'}' expected. '{' or ';' expected.

18:43 LF UTF-8

## 5. Negating completion

If you select a boolean item from the suggestion list with!, IntelliJ IDEA automatically adds the negation operator to the result.



## 6. Middle matching

IntelliJ IDEA also supports so-called Middle matching. It means that you don't necessarily need to type an identifier from the beginning. For example, if you only remember a part of the name, just type it and IntelliJ IDEA will still find the right matches for you.

```
package sample;

import java.util.Random;

public class JavaApp {
    public static void main(String[] args) {
        log("Generating 10 random integers in range 0..99");

        Random random = new Random();
        for (int idx = 0; idx < 10; idx++) {
            int randomInt = random.nextInt();
            log("Generated: "
        }

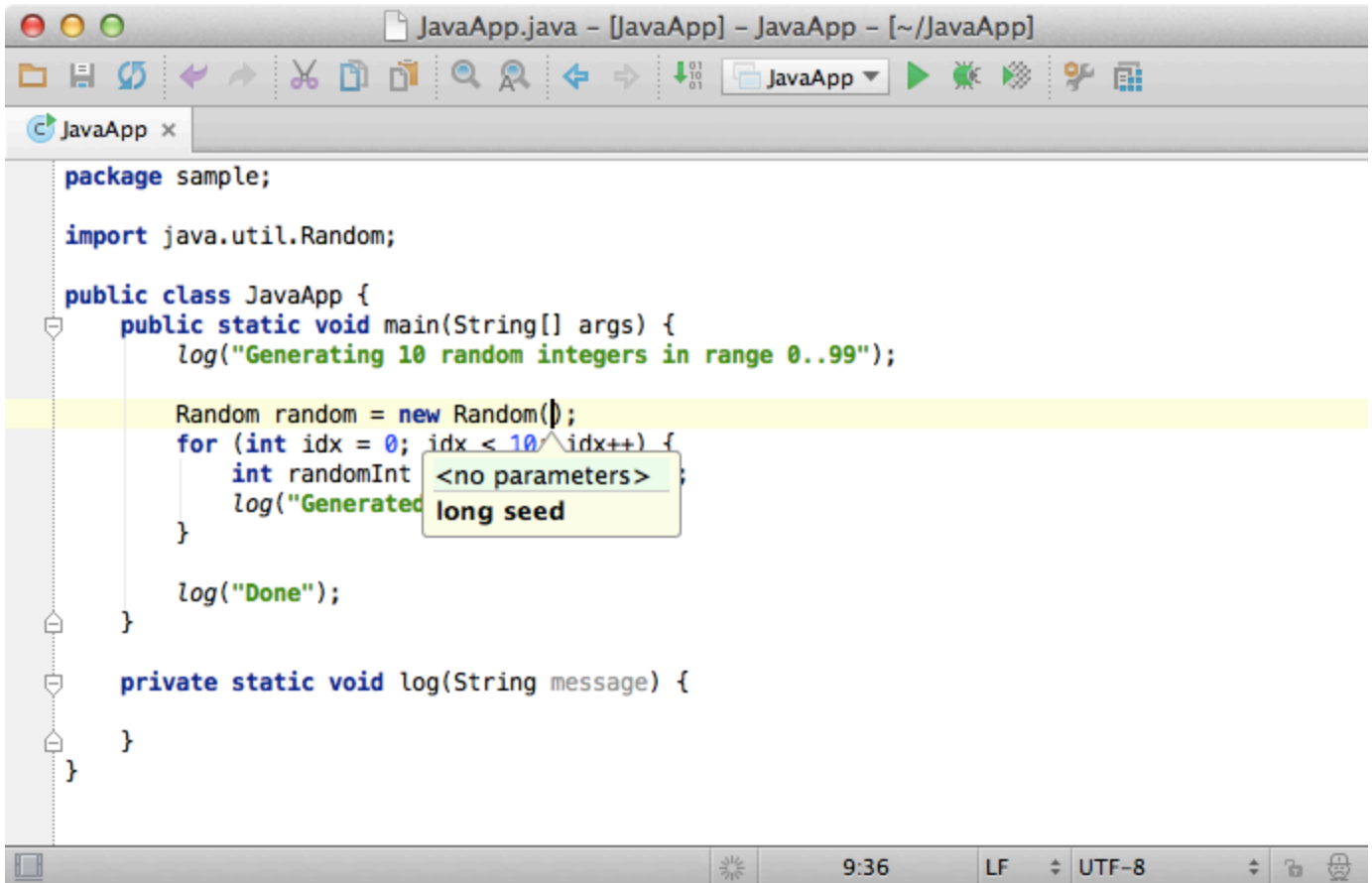
        log("Done");
    }

    private static void log(String message) {
    }
}
```

The screenshot shows the IntelliJ IDEA IDE with a Java file named `JavaApp.java`. The code defines a `sample` package with a `JavaApp` class. The `main` method generates 10 random integers. A parameter info popup is visible over the `nextInt()` call, showing two overloads: `nextInt()` and `nextInt(int n)`, both returning `int`. The status bar at the bottom shows "Unexpected token" and the time is 11:39.

## 7. Parameter info

If you want to see the suggested parameters for any method or constructor, just press `Ctrl+P` (`Cmd+P` for Mac). IntelliJ IDEA shows parameter info for every overloaded method or constructor, and highlights the best match for the parameters already typed. This helps you to choose between overloaded options and compare your input with what's expected.



The screenshot shows an IDE window titled "JavaApp.java - [JavaApp] - JavaApp - [~/JavaApp]". The code is as follows:

```
package sample;

import java.util.Random;

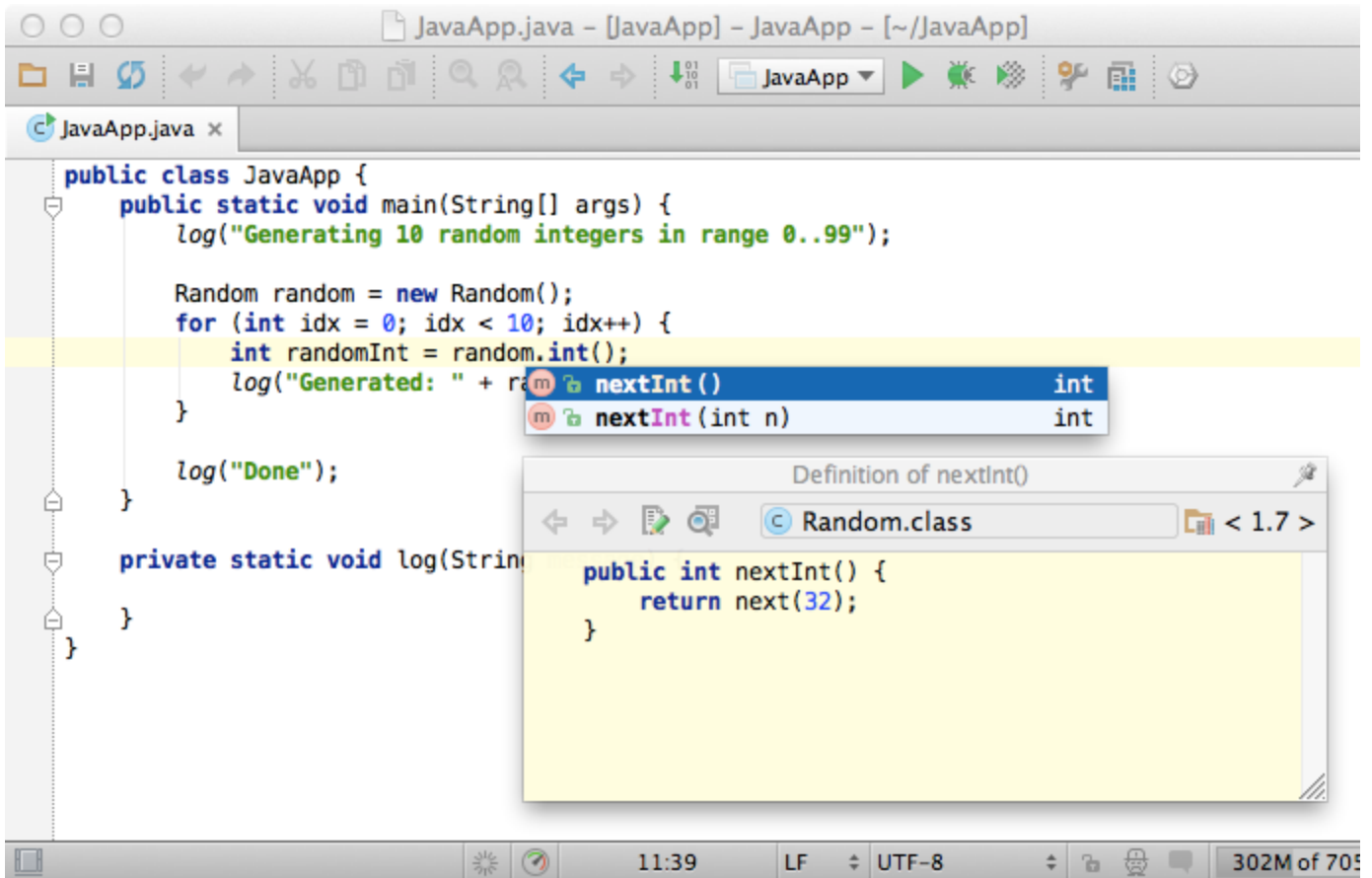
public class JavaApp {
    public static void main(String[] args) {
        log("Generating 10 random integers in range 0..99");
        Random random = new Random();
        for (int idx = 0; idx < 10; idx++) {
            int randomInt = random.nextInt();
            log("Generated " + randomInt);
        }
        log("Done");
    }

    private static void log(String message) {
    }
}
```

A quick popup is visible over the `new Random()` call, showing the signature `<no parameters>` and the text `long seed`. The IDE interface includes a toolbar with icons for file operations, search, and execution, and a status bar at the bottom showing the time 9:36, line length LF, and encoding UTF-8.

## 8. Quick popups

Quick popups such as Quick documentation (via `Ctrl+Q`, or `Ctrl+J` for Mac) or Quick definition (via `Shift+Ctrl+I`, or `Shift+Cmd+I` for Mac) are available when you browse suggestions in code completion.



## 9. Exclude from completion

If you are fed up with some class constantly appearing in the suggestion list, IntelliJ IDEA can exclude this class or even a whole package so you never see it again. Just press `Alt+Enter` on an item and confirm the exclusion. You can always turn it back in `SettingsEditorAuto Import`.



```
package sample;

import java.util.Random;

public class JavaApp {
    public static void main(String[] args) {
        log("Generating 10 random integers in range 0..99");

        Random random = new Ra
        for (int idx = 0; idx < 10; idx++) {
            int randomI = random.nextInt(10);
            log("Generated random integer: " + randomI);
        }

        log("Done");
    }

    private static void log(String message) {
        System.out.println(message);
    }
}
```

Cannot resolve symbol 'Ra'. '(' or '[' expected.

9:31 LF UTF-8

## 10. Completion settings

If you want to change the default settings for completion you can do it via `SettingsEditorCode Completion`.

