FINDING BUGS IN YOUR CODE BASE

Krishnakripa Jayakumar

THE GOAL?

We will look at 2 different ways of finding errors in your codebase.

WHAT ARE THEY?

1. Debugging
2. Static Code Analysis

WHAT IS DEBUGGING?

Debugging is the process of identifying and removing errors or bugs from your code base

DEBUGGING?

- There are different kinds of errors, which you are going to deal with.
- Some of them are easy to catch, like syntax errors, because they are taken care of by the compiler.
- Another easy case is when the error can be quickly identified by looking at the stack trace, which helps you figure out where the error occurred.

DEBUGGING?

- However, there are errors which can be very tricky and take really long to find and fix.
- For example, a subtle logic error, which happened early in the program may not manifest itself until very late, and sometimes it is a real challenge to sort things out.

DEBUGGING?

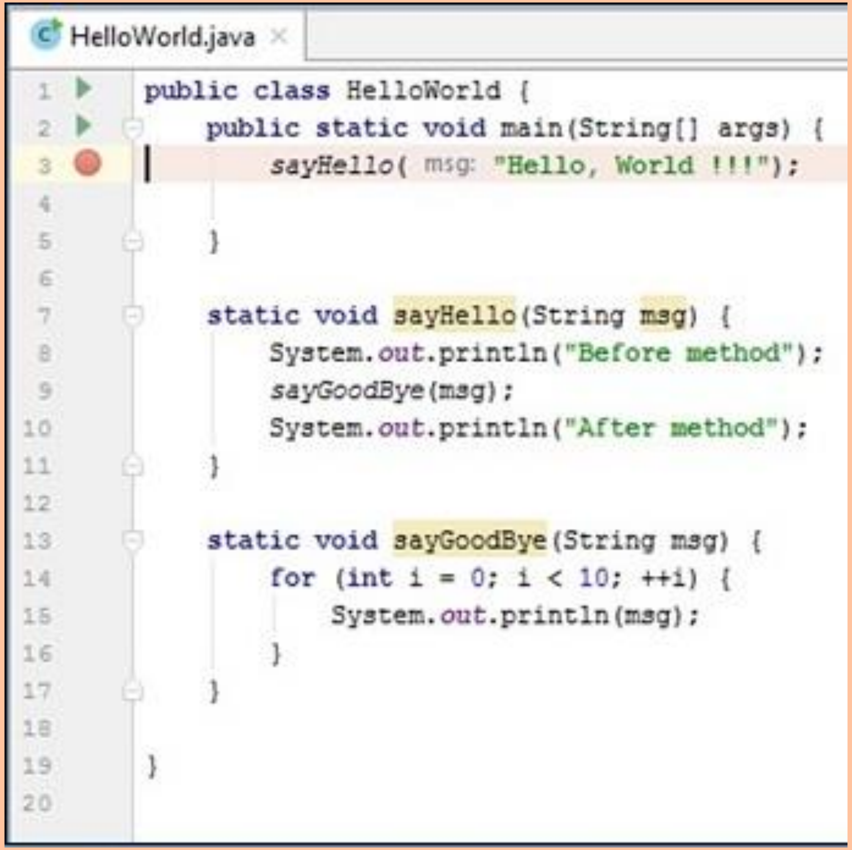
- This is where the debugger is useful.
- The debugger is a powerful tool, which lets you find bugs a lot faster by providing an insight into the internal operations of a program.
- This is possible by pausing the execution and analyzing the state of the program by thorough examination of variables and how they are changed line by line.

DEBUGGING?

- While debugging, you are in full control of the things. In this manual we are covering a basic debugging scenario to get you started.

BREAKPOINTS

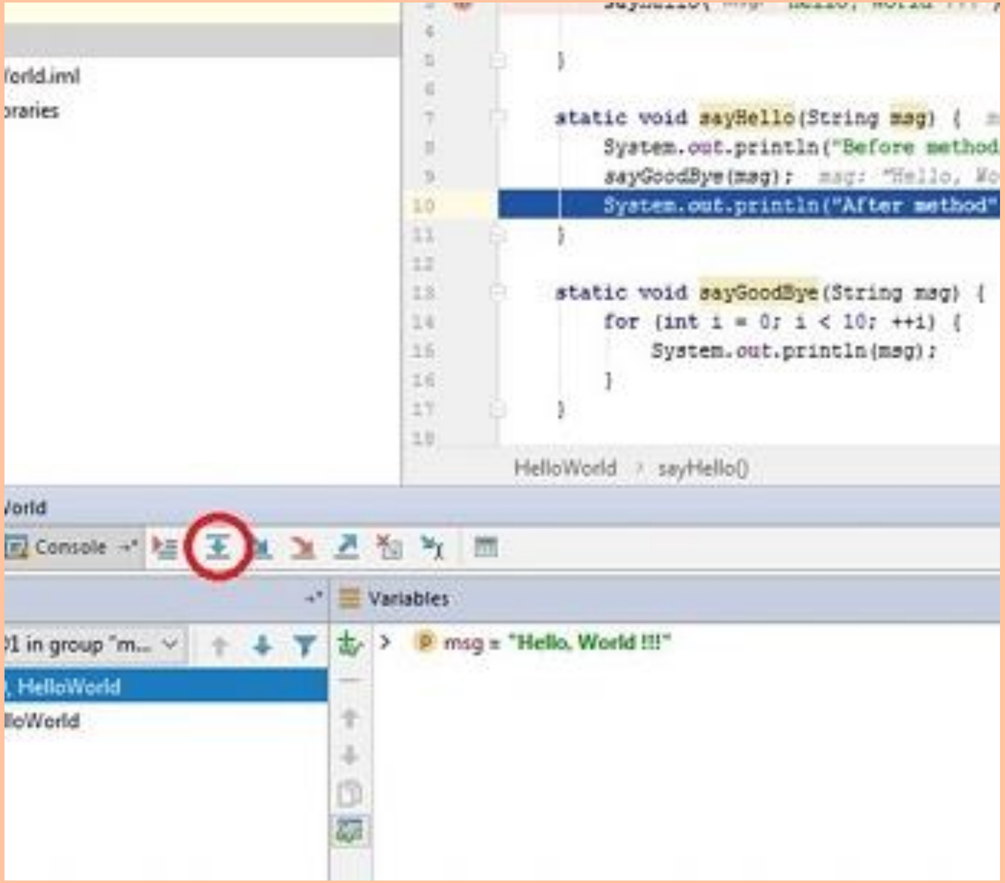
- Breakpoint allows stopping program execution at certain point. Breakpoints can be set by hovering the mouse over the Editor' s gutter area and clicking on it.
- Breakpoints are denoted using red circle symbols. Consider the breakpoint set at line 3.



```
1   public class HelloWorld {
2       public static void main(String[] args) {
3 ●         sayHello( msg: "Hello, World !!!");
4
5       }
6
7       static void sayHello(String msg) {
8           System.out.println("Before method");
9           sayGoodBye(msg);
10          System.out.println("After method");
11      }
12
13      static void sayGoodBye(String msg) {
14          for (int i = 0; i < 10; ++i) {
15              System.out.println(msg);
16          }
17      }
18
19  }
20
```

STEP OVER

- Will step over a given line.
- If the line contains a function the function will be executed and the result returned without debugging each line inside the function.

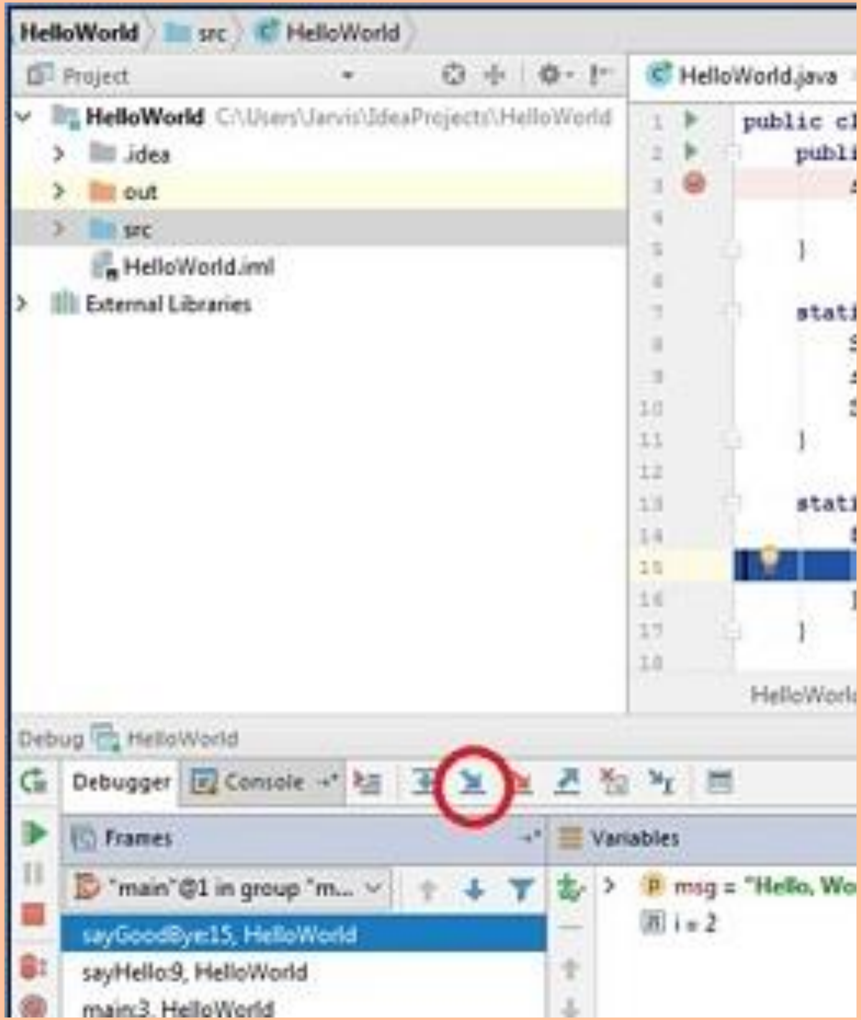


```
    }

    static void sayHello(String msg) {
        System.out.println("Before method
        sayGoodBye(msg);  msg: "Hello, Wo
        System.out.println("After method"
    }

    static void sayGoodBye(String msg) {
        for (int i = 0; i < 10; ++i) {
            System.out.println(msg);
        }
    }
```

msg = "Hello, World !!!"

STEP INTO

- The debugger will enter the function and continue line-by-line debugging there.
- If the line does not contain a function it behaves the same as " step over"



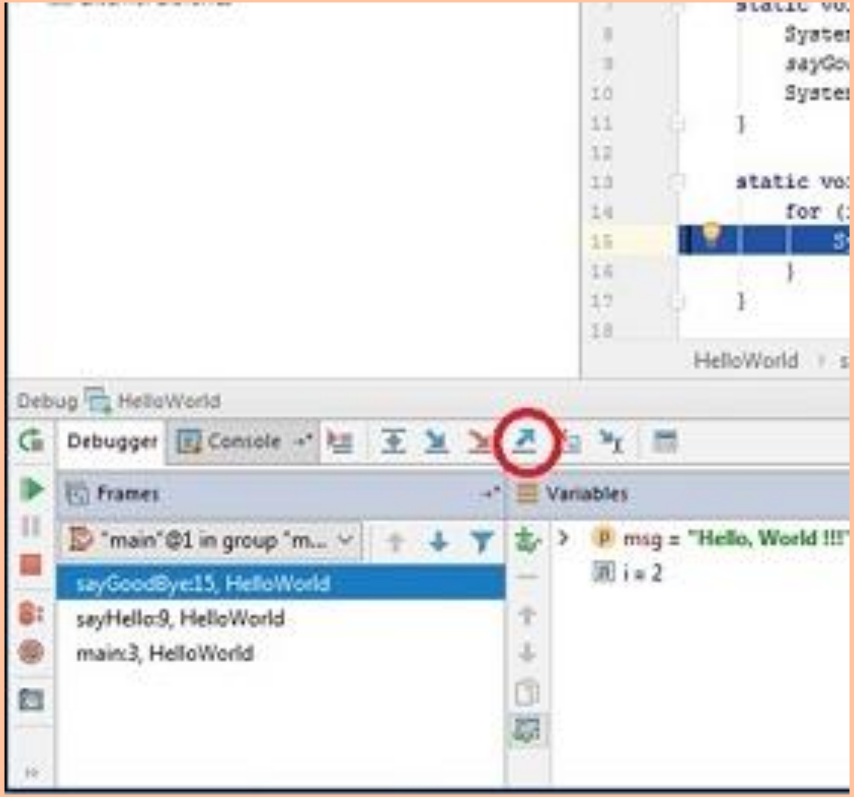
```
public cl
    publi

    }

    stati

    }

    stati

    }
```

sayGoodBye:15, HelloWorld
sayHello:9, HelloWorld
main:3, HelloWorld

msg = "Hello, Wo
i = 2

STEP OUT

- Returns to the line where the current function was called.



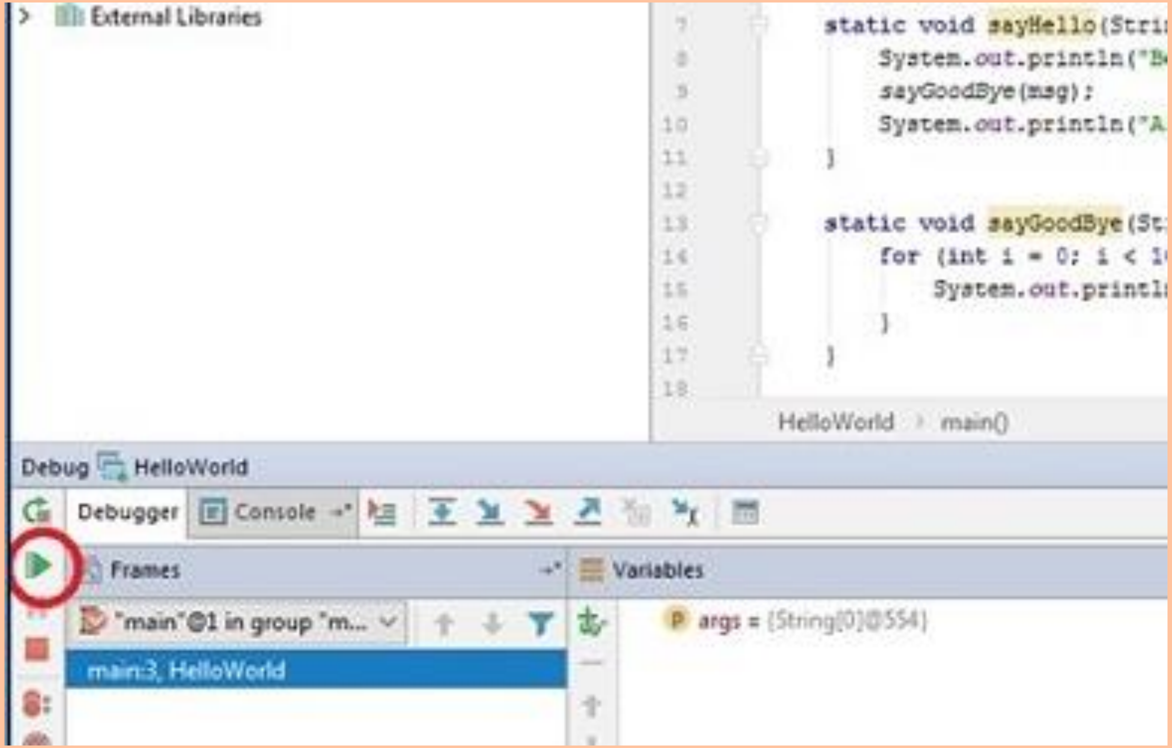
```
    static vo
        Syste
        sayGo
        Syste
    }

    static vo
        for (
            S
        }
    }
```

sayGoodBye:15, HelloWorld
sayHello:9, HelloWorld
main:3, HelloWorld

msg = "Hello, World !!!"
i = 2

RESUME

- Will continue execution until the next breakpoint is reached or the program exits.

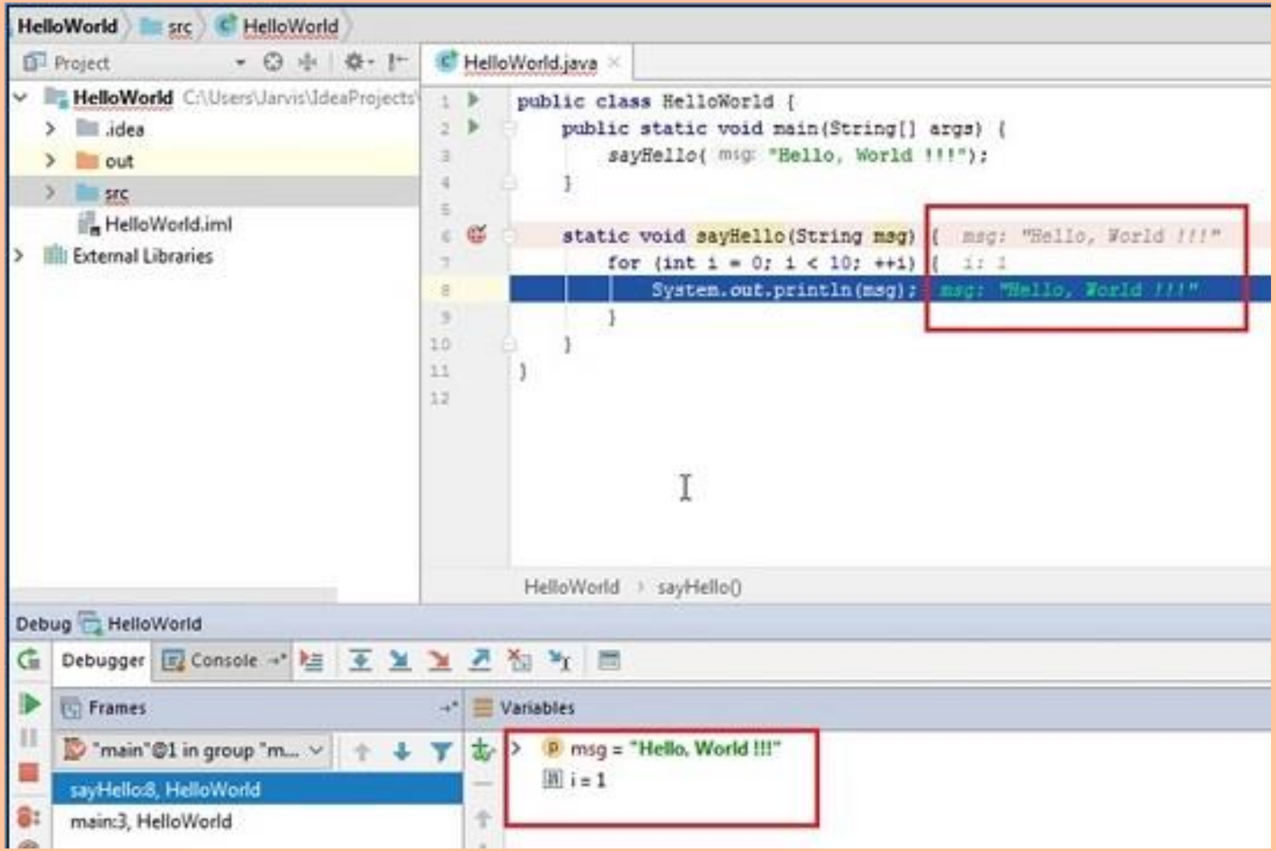


```
    static void sayHello(Stri
        System.out.println("B
        sayGoodBye(msg);
        System.out.println("A
    }

    static void sayGoodBye(St
        for (int i = 0; i < 1
            System.out.printl
        }
    }
```

```
args = {String[0]@554}
```

INSPECTING VARIABLES

- During debugging, IntelliJ shows value of variable in the Editor window itself. We can also view the same information in the Debug window.

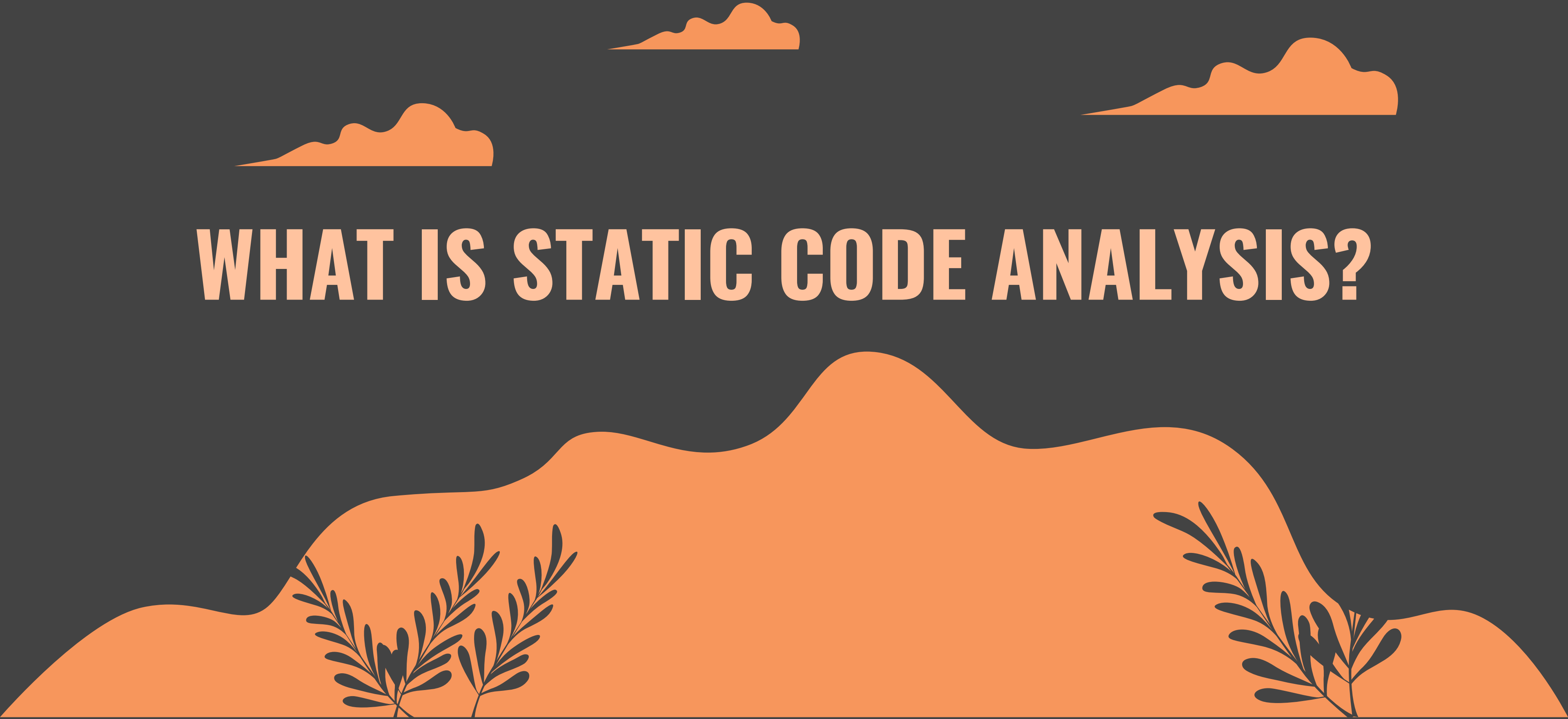


```
public class HelloWorld {
    public static void main(String[] args) {
        sayHello( msg: "Hello, World !!!");
    }

    static void sayHello(String msg) {  msg: "Hello, World !!!"
        for (int i = 0; i < 10; ++i) {  i: 1
            System.out.println(msg);  msg: "Hello, World !!!"
        }
    }
}
```

```
sayHello:8, HelloWorld
main:3, HelloWorld
```

```
> msg = "Hello, World !!!"
  i = 1
```

WHAT IS STATIC CODE ANALYSIS?



Static Code Analysis is the process of checking your program for errors without executing it

WHAT IT'S NOT

TESTING

WHAT IS A STATIC CODE ANALYSIS?

- Static code analyzer looks for patterns, defined to them as rules, which can cause security vulnerability or other code quality problems, necessary for production quality code.

TOOLS FOR STATIC CODE ANALYSIS

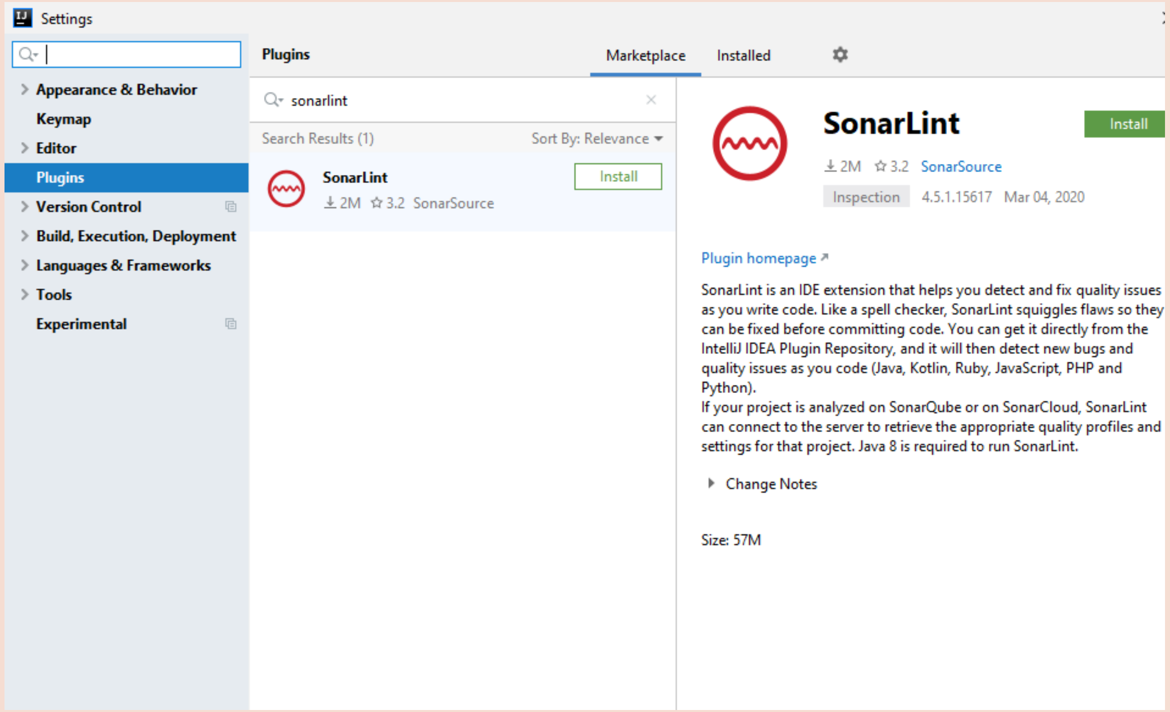
- CheckStyles
- FindBugs
- PMD
- SonarLint

CHOICE OF TOOL – SONARLINT

- SonarLint is an IDE extension that helps you detect and fix quality issues as you write code.
- Like a spell checker, SonarLint squiggles flaws so they can be fixed before committing code.
- You can get it directly from the IntelliJ IDEA Plugin Repository, and it will then detect new bugs and quality issues as you code

Add SonarLint to IntelliJ

1. Go to File -> Settings -> Plugins -> MarketPlace
2. Search for ‘SonarLint’

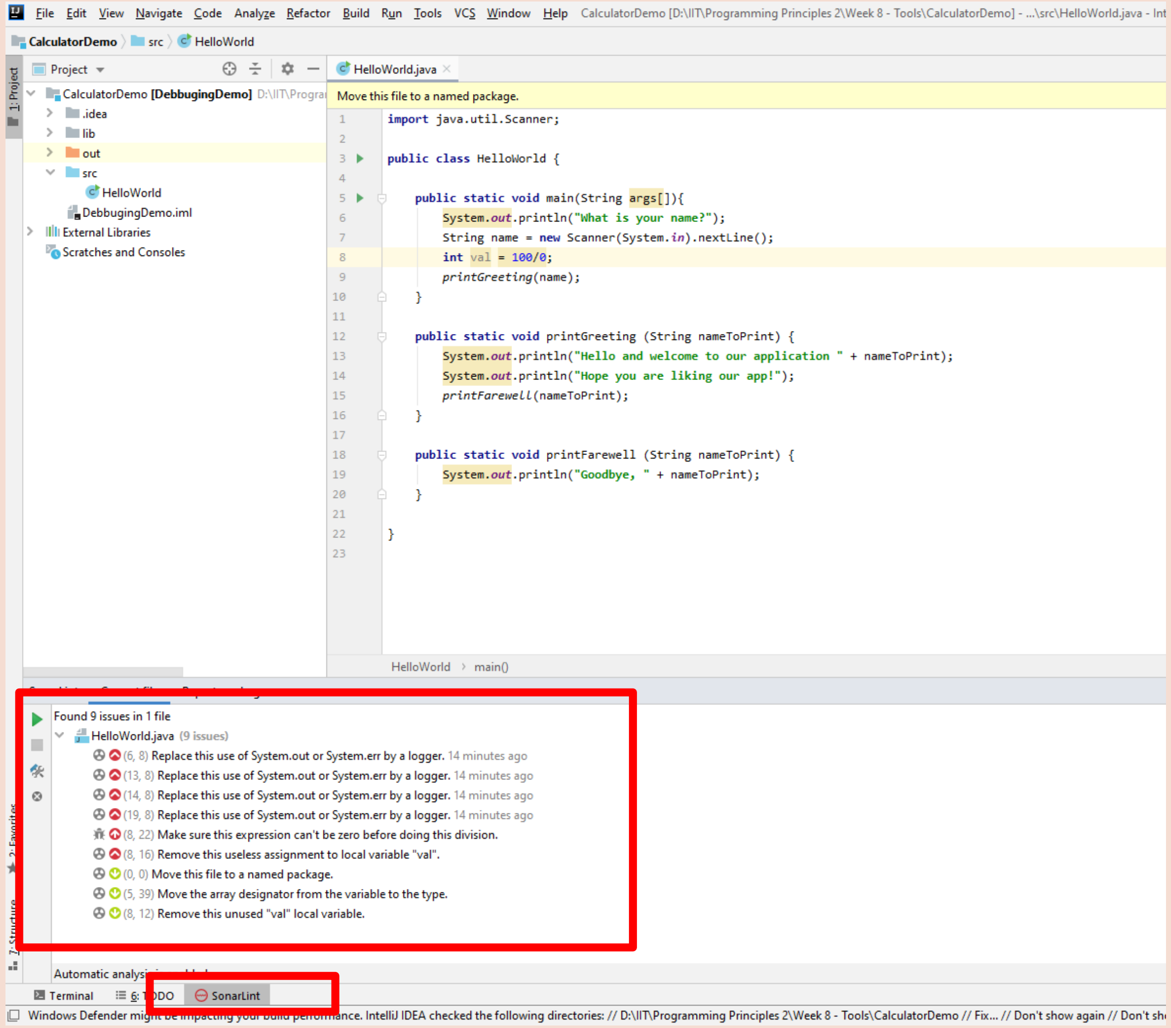


SonarLint is an IDE extension that helps you detect and fix quality issues as you write code. Like a spell checker, SonarLint squiggles flaws so they can be fixed before committing code. You can get it directly from the IntelliJ IDEA Plugin Repository, and it will then detect new bugs and quality issues as you code (Java, Kotlin, Ruby, JavaScript, PHP and Python).
If your project is analyzed on SonarQube or on SonarCloud, SonarLint can connect to the server to retrieve the appropriate quality profiles and settings for that project. Java 8 is required to run SonarLint.

Size: 57M

Add SonarLint to IntelliJ

3. Once installed, restart the IDE
4. Use the SonarLint window to review issues


File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
CalculatorDemo [D:\IIT\Programming Principles 2\Week 8 - Tools\CalculatorDemo] - ...\src\HelloWorld.java - Int
CalculatorDemo
src
HelloWorld
Project
CalculatorDemo [DebbugingDemo] D:\IIT\Progra
.idea
lib
out
src
HelloWorld
DebbugingDemo.iml
External Libraries
Scratches and Consoles
HelloWorld.java
HelloWorld
main()
Terminal
6: TODO
SonarLint

Move this file to a named package.

```java
1   import java.util.Scanner;
2
3   public class HelloWorld {
4
5       public static void main(String args[]){
6           System.out.println("What is your name?");
7           String name = new Scanner(System.in).nextLine();
8           int val = 100/0;
9           printGreeting(name);
10      }
11
12      public static void printGreeting (String nameToPrint) {
13          System.out.println("Hello and welcome to our application " + nameToPrint);
14          System.out.println("Hope you are liking our app!");
15          printFarewell(nameToPrint);
16      }
17
18      public static void printFarewell (String nameToPrint) {
19          System.out.println("Goodbye, " + nameToPrint);
20      }
21
22  }
23
```

```
Found 9 issues in 1 file
HelloWorld.java (9 issues)
    (6, 8) Replace this use of System.out or System.err by a logger. 14 minutes ago
    (13, 8) Replace this use of System.out or System.err by a logger. 14 minutes ago
    (14, 8) Replace this use of System.out or System.err by a logger. 14 minutes ago
    (19, 8) Replace this use of System.out or System.err by a logger. 14 minutes ago
    (8, 22) Make sure this expression can't be zero before doing this division.
    (8, 16) Remove this useless assignment to local variable "val".
    (0, 0) Move this file to a named package.
    (5, 39) Move the array designator from the variable to the type.
    (8, 12) Remove this unused "val" local variable.
```

Automatic analysi

Windows Defender mig ance. IntelliJ IDEA checked the following directories: // D:\IIT\Programming Principles 2\Week 8 - Tools\CalculatorDemo // Fix... // Don't show again // Don't sh