

# Technological foundations of software development

Debug, log, test, profile, analyze your software

ICM – Computer Science Major – Course unit on Technological foundations of computer science

M1 Cyber Physical and Social Systems – Course unit on CPS2 engineering and development, Part 2: Technological foundations of software development

Maxime Lefrançois <https://maxime-lefrancois.info>

online: <https://ci.mines-stetienne.fr/cps2/course/tfsd/>

# Objectives of the session

This session aims to familiarize you with the methods and tools for debugging, logging, testing, profiling, analyzing your software. In particular, we will cover tools for Java, Python, JavaScript.

# Technological foundations of software development

Debug, log, test, profile, analyze your software

Part 1: Big Bang development

ICM – Computer Science Major – Course unit on Technological foundations of computer science

M1 Cyber Physical and Social Systems – Course unit on CPS2 engineering and development, Part 2: Technological foundations of software development

Maxime Lefrançois <https://maxime-lefrancois.info>

online: <https://ci.mines-stetienne.fr/cps2/course/tfsd/>

... write code the old-fashioned way



# ... possibly with some precautions

- try-catch

```
try { ... } catch (Exception ex) { ... }
```

- multcatch

```
try { ... } catch (NullPointerException | IOException ex) { ... }
```

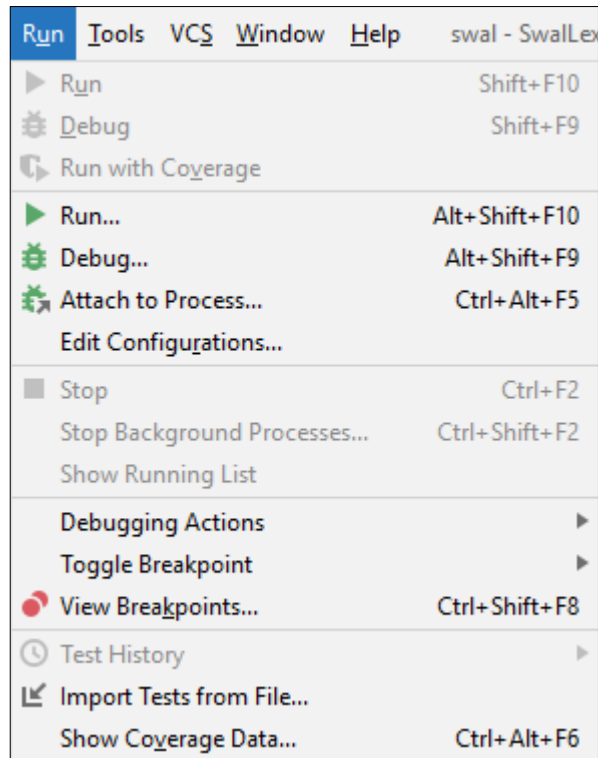
- finally

```
try { ... } catch (Exception ex) { ... } finally { ... }
```

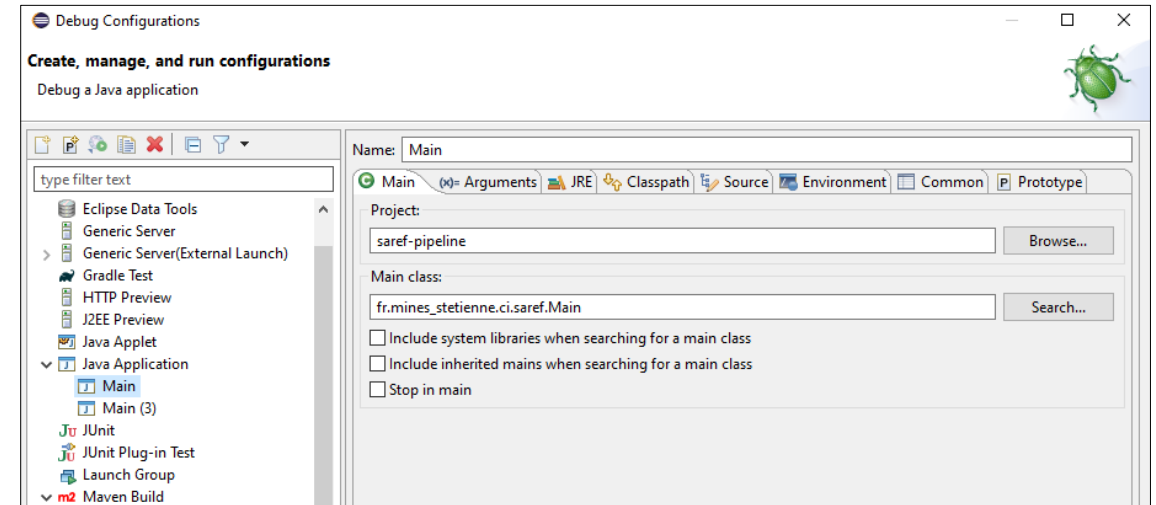
- try with resource:

```
try (InputStream fis = new FileInputStream(file) ) { ... }  
catch (...) { ... }
```

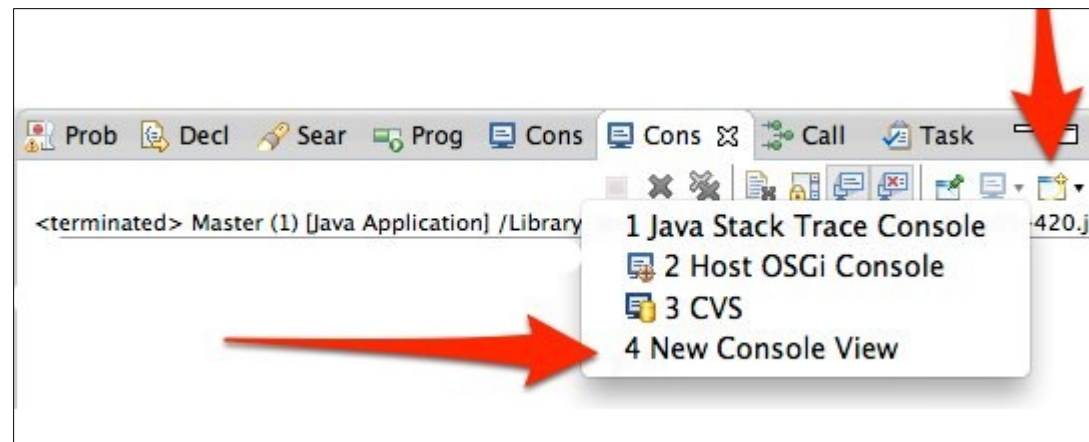
# Run your code...



Run Menu in IntelliJ



Run configuration in Eclipse



Console in Eclipse

... and stumble upon an error



```
Console X
Java Stack Trace Console
Exception in thread "main" java.lang.NullPointerException
    at Main.main(Main.java:9)
```

```
Exception in thread "main" java.lang.IllegalStateException: A book has a null property
    at com.example.myproject.Author.getBookIds(Author.java:38)
    at com.example.myproject.Bootstrap.main(Bootstrap.java:14)
Caused by: java.weblayer.ZzzException
    at com.example.weblayer.Book.getId(WebBook.java:12)
    at com.example.weblayer.Author.getBookIds(WebAuthor.java:38)
Caused by: java.servicelayer.YyyException
    at com.example.servicelayer.Book.getId(BookService.java:220)
    at com.example.servicelayer.Author.getBookIds(AuthorService.java:350)
Caused by: java.componentlayer.NullPointerException
    at com.example.componentlayer.Book.getId(Book.java:22)
    at com.example.componentlayer.Author.getBookIds(Author.java:35)
Caused by: java.lang.daolayer.XxxException
    at com.example.daolayer.Book.getId(BookDao.java:22)
    at com.example.daolayer.Author.getBookIds(AuthorDao.java:35)
... 1 more
```

**Root Cause may  
in the middle**

Stacktrace

# Print values ...

```
try{
    System.out.println("Testing PTree constructor setup");
    PTree ret = new PTree(true);
    System.out.println("PASSED setup test");
    sysout
    return ret;
}
```

↓ Ctrl + Space

```
try{
    System.out.println("Testing PTree constructor setup");
    PTree ret = new PTree(true);
    System.out.println("PASSED setup test");
    System.out.println();
    return ret;
}
```

and we comment ... and we decomment ...

and we start again...



# Technological foundations of software development

Debug, log, test, profile, analyze your software

Part 2: Troubleshoot a bug

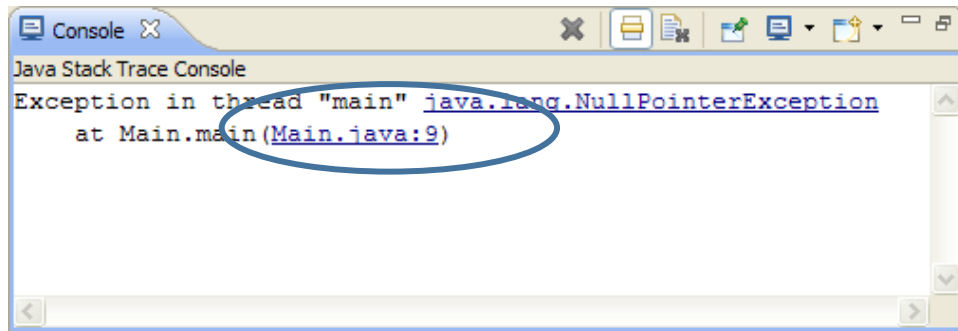
ICM – Computer Science Major – Course unit on Technological foundations of computer science

M1 Cyber Physical and Social Systems – Course unit on CPS2 engineering and development, Part 2: Technological foundations of software development

Maxime Lefrançois <https://maxime-lefrancois.info>

online: <https://ci.mines-stetienne.fr/cps2/course/tfsd/>

... and stumble upon an error



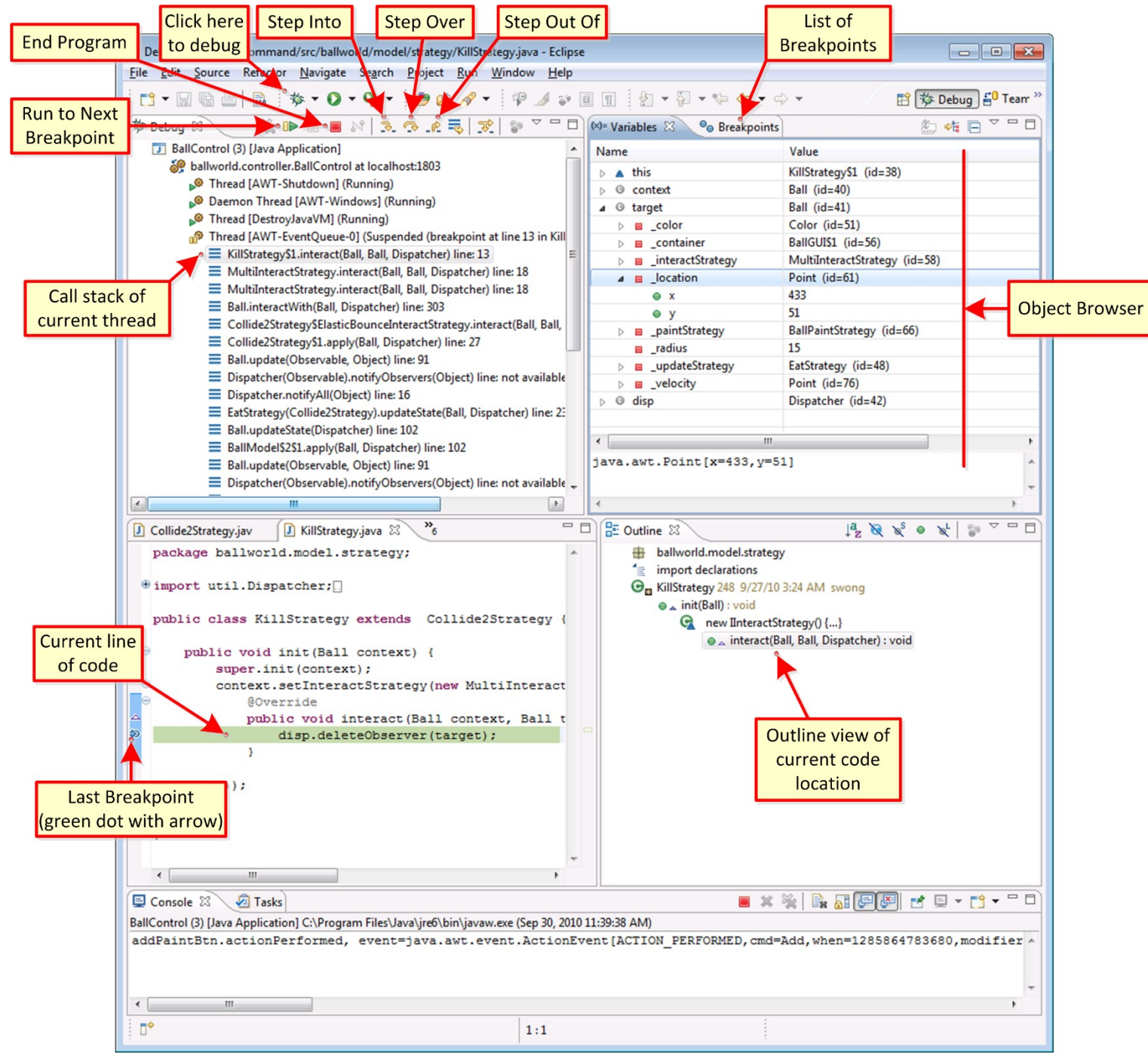
```
Exception in thread "main" java.lang.IllegalStateException: A book has a null property
    at com.example.myproject.Author.getBookIds(Author.java:38)
    at com.example.myproject.Bootstrap.main(Bootstrap.java:14)
Caused by: java.weblayer.ZzzException
    at com.example.weblayer.Book.getId(WebBook.java:12)
    at com.example.weblayer.Author.getBookIds(WebAuthor.java:38)
Caused by: java.servicelayer.YyyException
    at com.example.servicelayer.Book.getId(BookService.java:220)
    at com.example.servicelayer.Author.getBookIds(AuthorService.java:350)
Caused by: java.componentlayer.NullPointerException
    at com.example.componentlayer.Book.getId(Book.java:22)
    at com.example.componentlayer.Author.getBookIds(Author.java:35)
Caused by: java.lang.daolayer.XxxException
    at com.example.daolayer.Book.getId(BookDao.java:22)
    at com.example.daolayer.Author.getBookIds(AuthorDao.java:35)
... 1 more
```

**Root Cause may  
in the middle**

Stacktrace

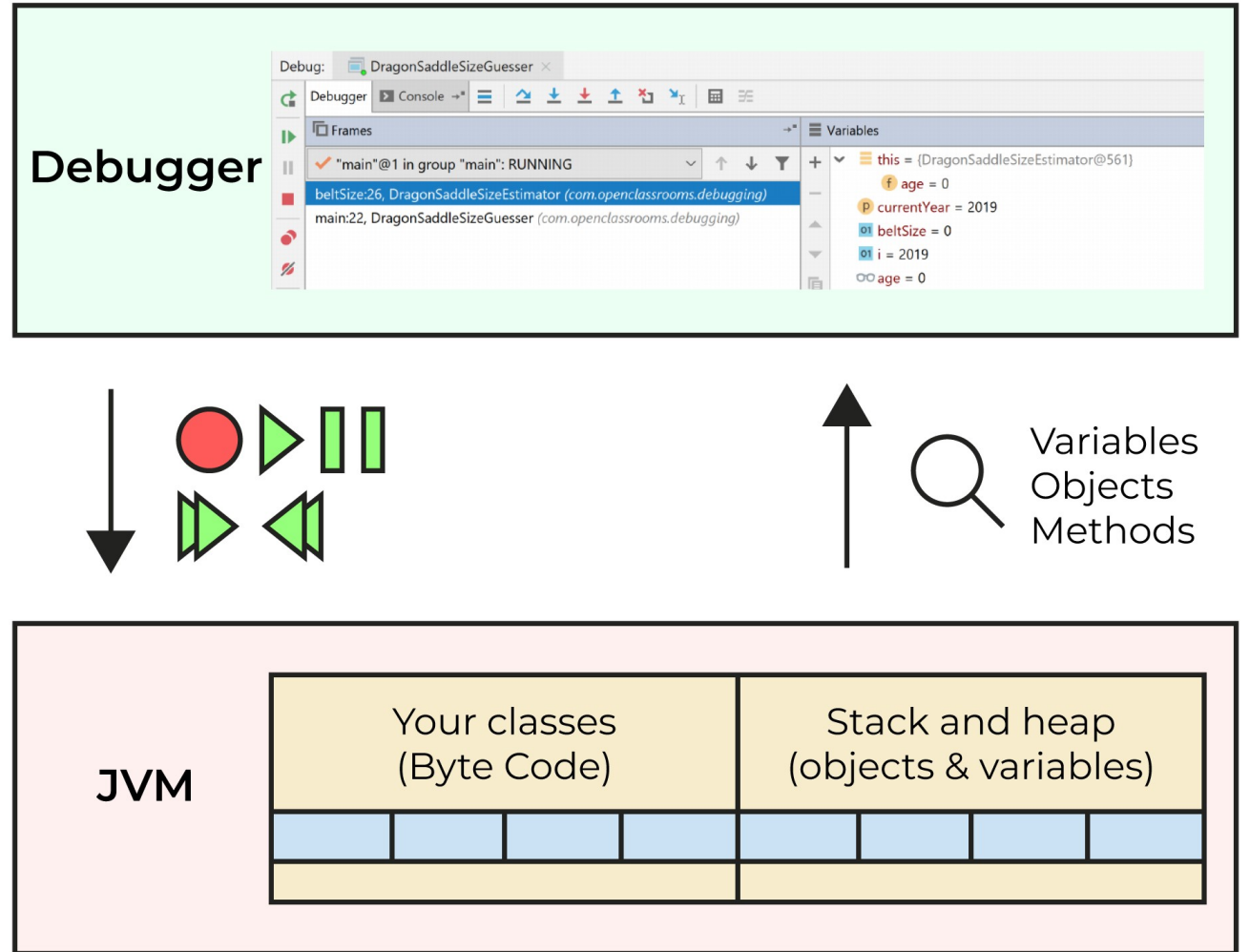
# Debugging

- Controlled execution of the program
- Monitoring and/or modification of the content of variables during execution
- Allows to trace bugs



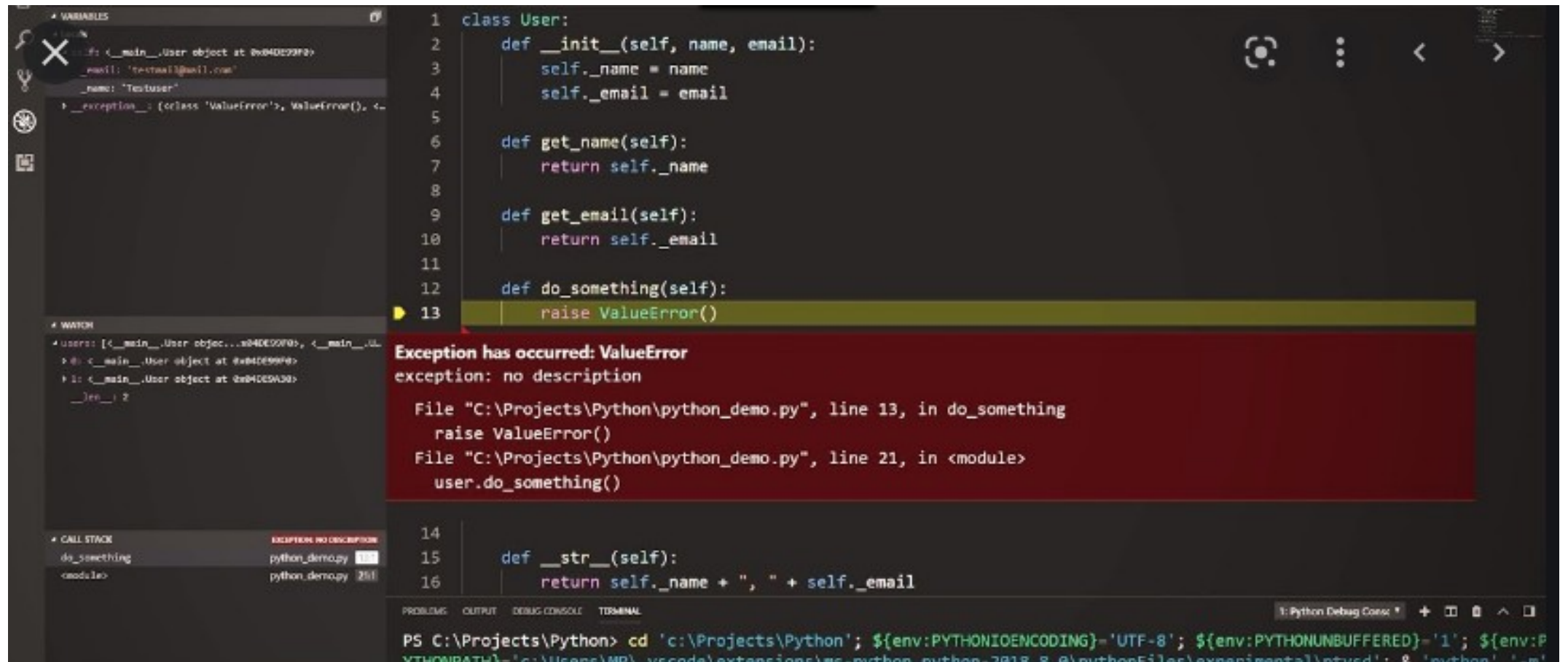
# Debugging

- Uses the **Java Platform Debugger Architecture (JPDA)** framework
- All Java IDE debuggers will have approximately the same functionalities



JPDA

# Debugging with Python



# Debugging with Python

- Python debuggers rely on the standard **pdb** module

The debugger's prompt is `(Pdb)`. Typical usage to run a program under control of the debugger is:

```
>>> import pdb
>>> import mymodule
>>> pdb.run('mymodule.test()')
> <string>(0)?()
(Pdb) continue
> <string>(1)?()
(Pdb) continue
NameError: 'spam'
> <string>(1)?()
(Pdb)
```



# Debugging with node.js

The image displays the Chrome DevTools interface for debugging a Node.js application. On the left, the 'DevTools' sidebar is open, showing the 'Devices' panel with options to 'Discover USB devices' and 'Discover network targets'. Below this, the 'Remote Target' section shows a target named 'hello-name.js' at 'file:///Users/jacopodaelli/dev/JacopoDaelli/hello/hello-name.js'. The main window shows the 'Sources' panel with the file 'hello-name.js' open. The code is as follows:

```
1 (function (exports, require, module, __filename, __dirname) { 'use strict'
2
3 const express = require('express')
4 const app = express()
5
6 const PORT = process.env.PORT || 3000
7
8 function capitalize (str) {
9   const firstLetter = str.charAt(0) // we can check what's inside here
10  return `${firstLetter.toUpperCase()}${str.slice(1)}`
11 }
12
13 app.get('/:name?', (req, res) => { req = IncomingMessage { _readableState:
14   const name = req.params.name ? capitalize(req.params.name) : 'World'
15   res.send(`Hello ${name}!`)
16 })
17
18 app.listen(PORT, () => console.log(`App listening on *:${PORT}`))
19
20 });
```

The code is paused on a breakpoint at line 14, column 16. The right sidebar shows the 'Paused on breakpoint' status, a 'Watch' section with 'No Watch Expressions', a 'Call Stack' section, a 'Scope' section with 'Local' variables (name: undefined, req: IncomingMessage, res: ServerResponse, this: undefined), and a 'Breakpoints' section with two breakpoints: 'hello-name.js:9' and 'hello-name.js:14'.

# Technological foundations of software development

Debug, log, test, profile, analyze your software

Part 3: Smart Logging

ICM – Computer Science Major – Course unit on Technological foundations of computer science

M1 Cyber Physical and Social Systems – Course unit on CPS2 engineering and development, Part 2: Technological foundations of software development

Maxime Lefrançois <https://maxime-lefrancois.info>

online: <https://ci.mines-stetienne.fr/cps2/course/tfsd/>



# Logging

## Print messages with varying degrees of severity

- fatal : messages concerning an unexpected stop of the application
- error : error messages requiring analysis e.g. exceptions thrown
- warn : warning message
- info : information messages for example the start or stop of the application, the connection to a resource, ...
- trace : messages to follow the execution flow
- debug : debugging messages for example to get the value of variables, ...



only during development

# Logging

## Example for Java: log4j framework

```
1 package com.jmdoudoux.test.log4j;
2
3 import org.apache.log4j.Logger;
4
5 public class TestLog4j1 {
6
7     private static Logger logger = Logger.getLogger(TestLog4j1.class);
8
9     public static void main(String[] args) {
10         logger.debug("msg de debogage");
11         logger.info("msg d'information");
12         logger.warn("msg d'avertissement");
13         logger.error("msg d'erreur");
14         logger.fatal("msg d'erreur fatale");
15     }
16 }
```

# Logging

## Exemple for Java: log4j framework (with Maven)

- add `log4j:log4j:jar:1.2.x` in the list of dependencies in the `pom.xml`
- configuration file `src/main/resources/log4j.properties`

```
1 log4j.rootLogger=DEBUG, stdout
2 log4j.appender.stdout=org.apache.log4j.ConsoleAppender
3 log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
4 log4j.appender.stdout.layout.ConversionPattern=%d [%-5p] (%F:%M:%L) %m%n
```

an example of a `log4j.properties` file

- different appenders (console, files, ...)
- choice of the pattern <https://logging.apache.org/log4j/1.2/apidocs/org/apache/log4j/PatternLayout.html>

# Logging

```
1 package com.jmdoudoux.test.log4j;
2
3 import org.apache.log4j.Logger;
4
5 public class TestLog4j1 {
6
7     private static Logger logger = Logger.getLogger(TestLog4j1.class);
8
9     public static void main(String[] args) {
10         logger.debug("msg de debogage");
11         logger.info("msg d'information");
12         logger.warn("msg d'avertissement");
13         logger.error("msg d'erreur");
14         logger.fatal("msg d'erreur fatale");
15     }
16 }
```

```
1 log4j.rootLogger=DEBUG, stdout
2 log4j.appender.stdout=org.apache.log4j.ConsoleAppender
3 log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
4 log4j.appender.stdout.layout.ConversionPattern=%d [%-5p] (%F:%M:%L) %m%n
```

```
2008-06-08 10:16:21,546 [DEBUG] (TestLog4j1.java:main:13) msg de debogage
2008-06-08 10:16:21,546 [INFO ] (TestLog4j1.java:main:14) msg d'information
2008-06-08 10:16:21,546 [WARN ] (TestLog4j1.java:main:15) msg d'avertissement
2008-06-08 10:16:21,546 [ERROR] (TestLog4j1.java:main:16) msg d'erreur
2008-06-08 10:16:21,546 [FATAL] (TestLog4j1.java:main:17) msg d'erreur fatale
```

# Logging

## Limit the cost of logging

```
logger.debug("valeur="+valeur+" , i="+i+" ,next="+next);
```



```
if (logger.isDebugEnabled()) {  
    logger.debug("valeur="+valeur+" , i="+i+" ,next="+next);  
}
```



```
private static Logger LOGGER = Logger.getLogger(MaClasse.class);
```



# Logging

## Same principles for Python

```
FORMAT = '%(asctime)-15s %(clientip)s %(user)-8s %(message)s'
logging.basicConfig(format=FORMAT)
d = {'clientip': '192.168.0.1', 'user': 'fbloggs'}
logger = logging.getLogger('tcpserver')
logger.warning('Protocol problem: %s', 'connection reset', extra=d)
```

would print something like

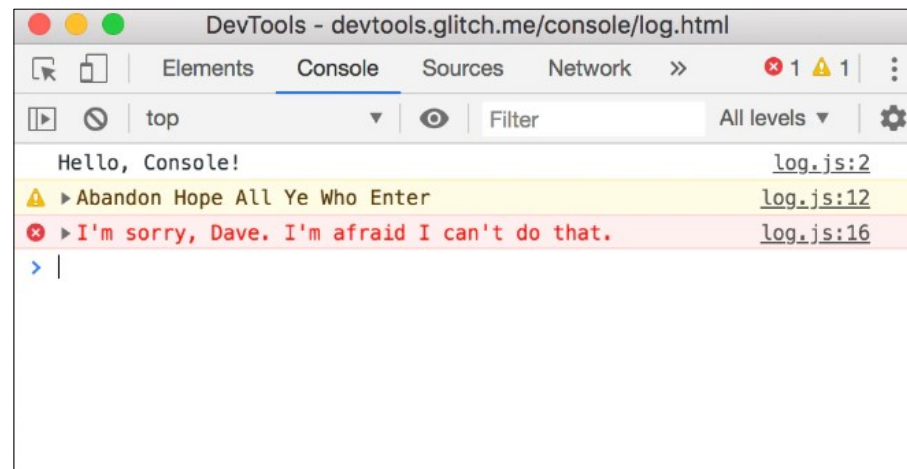
```
2006-02-08 22:20:02,165 192.168.0.1 fbloggs Protocol problem: connection reset
```

| Level    | Numeric value |
|----------|---------------|
| CRITICAL | 50            |
| ERROR    | 40            |
| WARNING  | 30            |
| INFO     | 20            |
| DEBUG    | 10            |
| NOTSET   | 0             |

# Logging

## Same principles for JavaScript

```
console.log('hello world');  
// Prints: hello world, to stdout  
console.log('hello %s', 'world');  
// Prints: hello world, to stdout  
console.error(new Error('Whoops, something bad happened'));  
// Prints error message and stack trace to stderr:  
//   Error: Whoops, something bad happened  
//     at [eval]:5:15  
//     at Script.runInThisContext (node:vm:132:18)  
//     at Object.runInThisContext (node:vm:309:38)  
//     at node:internal/process/execution:77:19  
//     at [eval]-wrapper:6:22  
//     at evalScript (node:internal/process/execution:76:60)  
//     at node:internal/main/eval_string:23:3  
  
const name = 'Will Robinson';  
console.warn(`Danger ${name}! Danger!`);  
// Prints: Danger Will Robinson! Danger!, to stderr
```



<https://developer.chrome.com/docs/devtools/console/log/>

# To read for MCQ test

- Python Basic Logging Tutorial  
<https://docs.python.org/3/howto/logging.html#basic-logging-tutorial>



# Technological foundations of software development

Debug, log, test, profile, analyze your software

Part 4: Writing and running tests

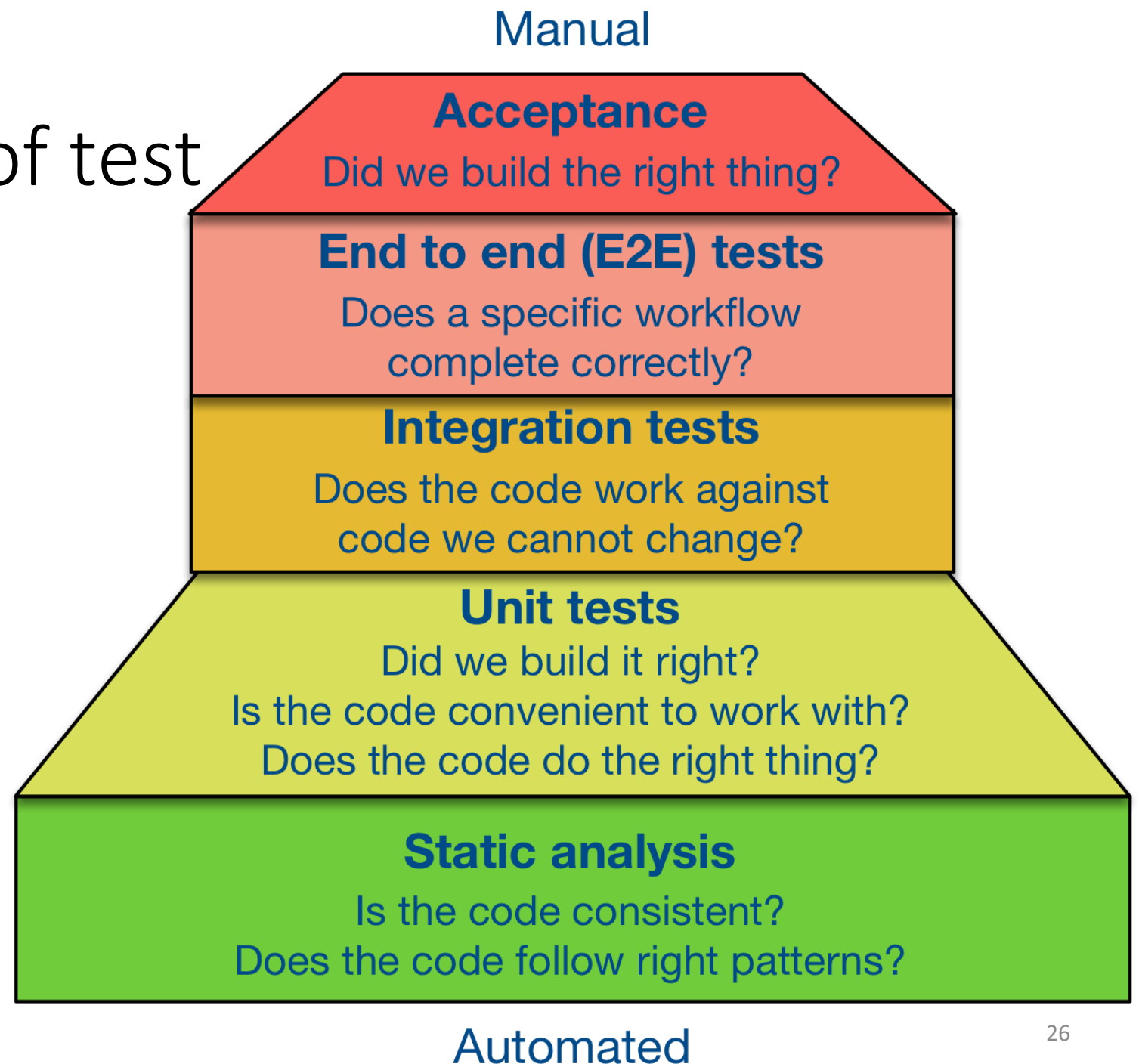
ICM – Computer Science Major – Course unit on Technological foundations of computer science

M1 Cyber Physical and Social Systems – Course unit on CPS2 engineering and development, Part 2: Technological foundations of software development

Maxime Lefrançois <https://maxime-lefrancois.info>

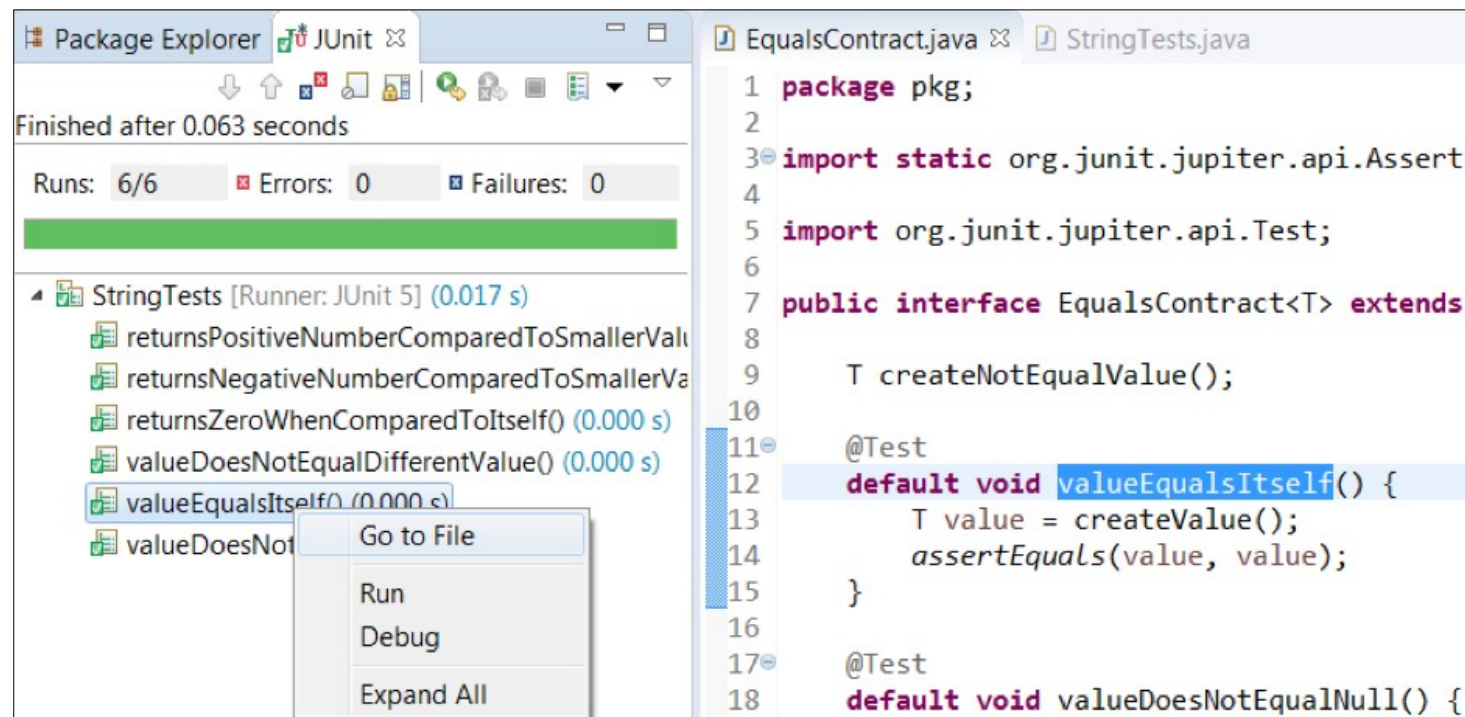
online: <https://ci.mines-stetienne.fr/cps2/course/tfsd/>

# Different types of test



# Unit tests- **JUnit** for Java

- open source framework for the development and execution of automatable unit tests



# Unit tests- **JUnit** for Java

- open source framework for the development and execution of automatable unit tests

```
public class MaClasse {  
  
    public static int calculer(int a, int b) {  
        int res = a + b;  
  
        if (a == 0) {  
            res = b * 2;  
        }  
  
        if (b == 0) {  
            res = a * a;  
        }  
        return res;  
    }  
}
```



```
import junit.framework.*;  
  
public class MaClasseTest extends TestCase {  
  
    public void testCalculer() throws Exception {  
        assertEquals(2, MaClasse.calculer(1, 1));  
    }  
}
```

# Unit tests – Multiple frameworks for Python

- Unit Testing Tools
- Mock Testing Tools
- Fuzz Testing Tools
- Web Testing Tools: Browser simulation, Browser automation
- Acceptance/Business Logic Testing Tools
- GUI Testing Tools
- Source Code Checking Tools
- Code Coverage Tools
- Continuous Integration Tools

# Unit tests – Multiple frameworks for Python

```
"""
This is the "example" module.

The example module supplies one function, factorial().  For example,

>>> factorial(5)
120
"""

def factorial(n):
    """Return the factorial of n, an exact integer >= 0.

    >>> [factorial(n) for n in range(6)]
    [1, 1, 2, 6, 24, 120]
    >>> factorial(30)
    265252859812191058636308480000000
    >>> factorial(-1)
    Traceback (most recent call last):
      ...
    ValueError: n must be >= 0

    Factorials of floats are OK, but the float must be an exact integer:
    >>> factorial(30.1)
    Traceback (most recent call last):
      ...
    ValueError: n must be exact integer
    >>> factorial(30.0)
    26525285981219105863630848000000

    It must also not be ridiculously large:
    >>> factorial(1e100)
    Traceback (most recent call last):
      ...
    OverflowError: n too large
    """

    import math
    if not n >= 0:
        raise ValueError("n must be >= 0")
    if math.floor(n) != n:
        raise ValueError("n must be an integer")
    while n > 1:
        n = n - 1
```

Standard **doctest** module searches for pieces of text that look like interactive Python sessions, and then executes those sessions to verify that they work exactly as shown.

```
if __name__ == "__main__":
    import doctest
    doctest.testmod()
```

you run example.py directly from the command line

```
$ python example.py
$
```

# Unit tests – Multiple frameworks for Python

Standard **unittest** module,

- inspired by Junit,
- similar flavor as major unit testing frameworks in other languages.
- Concepts: *test fixture*, *test case*, *test suite*, *test runner*

```
import unittest

class TestStringMethods(unittest.TestCase):

    def test_upper(self):
        self.assertEqual('foo'.upper(), 'FOO')

    def test_isupper(self):
        self.assertTrue('FOO'.isupper())
        self.assertFalse('Foo'.isupper())

    def test_split(self):
        s = 'hello world'
        self.assertEqual(s.split(), ['hello', 'world'])
        # check that s.split fails when the separator is not a string
        with self.assertRaises(TypeError):
            s.split(2)

if __name__ == '__main__':
    unittest.main()
```

# Unit tests – Multiple frameworks for Python

**pytest** module,

One of the most widely used Python testing frameworks

- Use `assert` statements (simpler than `unittest`)
- Auto-discovery of tests modules and functions
- Modular fixtures for managing small or parametrized long-lived test resources
- Can run **`unittest`** test suites out of the box

```
# content of test_sample.py
def inc(x):
    return x + 1

def test_answer():
    assert inc(3) == 5
```

To execute it:

```
$ pytest
===== test session starts =====
platform linux -- Python 3.x.y, pytest-8.x.y, pluggy-1.x.y
rootdir: /home/sweet/project
collected 1 item

test_sample.py F [100%]

===== FAILURES =====
_____ test_answer _____

    def test_answer():
>     assert inc(3) == 5
E       assert 4 == 5
E       + where 4 = inc(3)

test_sample.py:6: AssertionError
===== short test summary info =====
FAILED test_sample.py::test_answer - assert 4 == 5
===== 1 failed in 0.12s =====
```



# To read for MCQ test

- **doctest** documentation (beginning only)  
<https://docs.python.org/3/library/doctest.html>
- **pytest** getting started guide  
<https://docs.pytest.org/en/stable/getting-started.html>

# Technological foundations of software development

Debug, log, test, profile, analyze your software

Part 5: Profiling the execution of your software

ICM – Computer Science Major – Course unit on Technological foundations of computer science

M1 Cyber Physical and Social Systems – Course unit on CPS2 engineering and development, Part 2: Technological foundations of software development

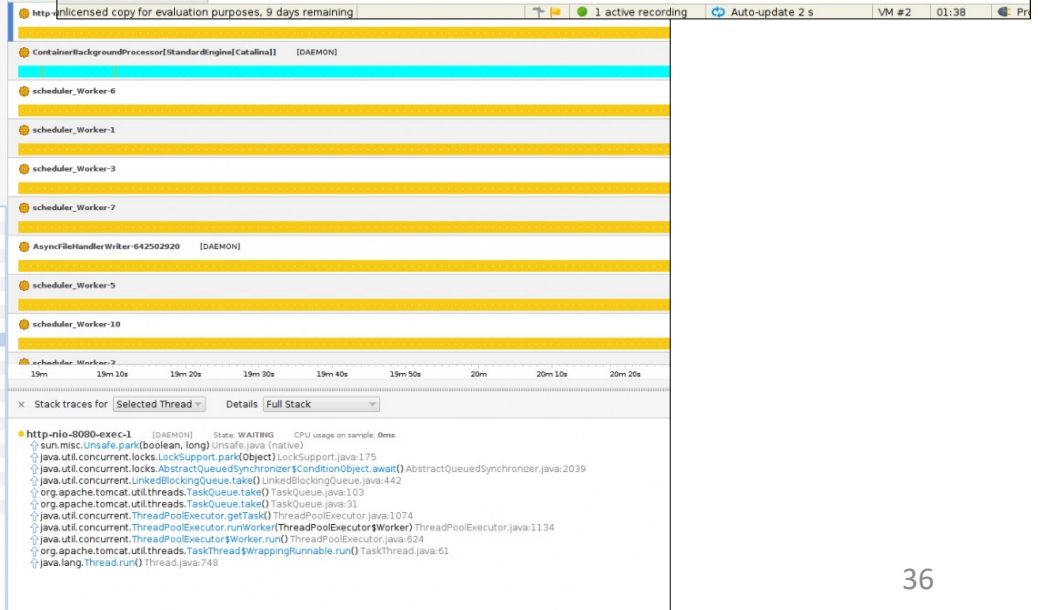
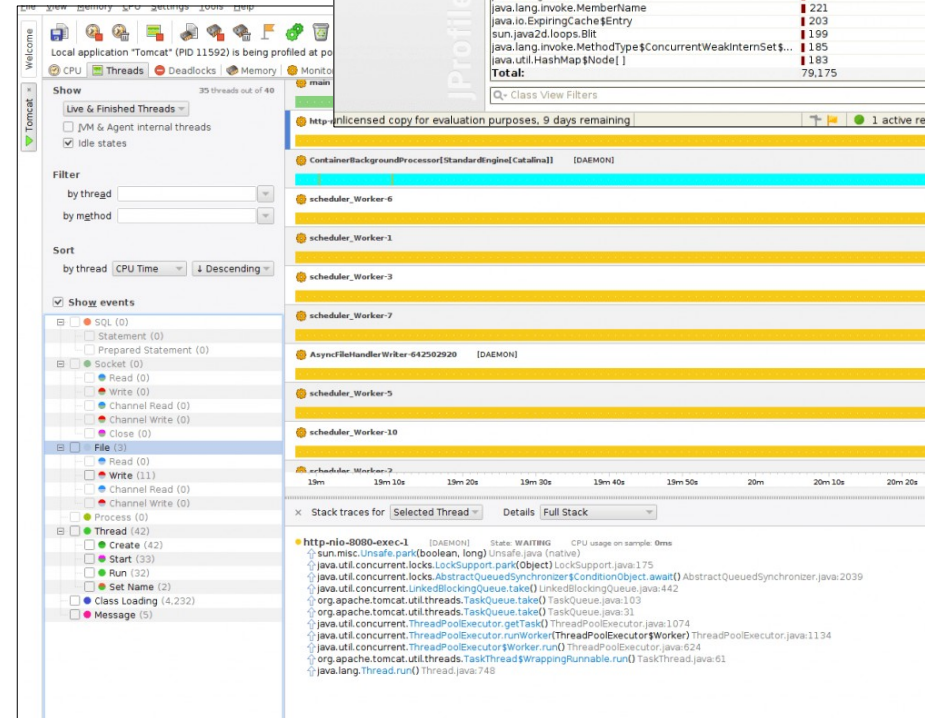
Maxime Lefrançois <https://maxime-lefrancois.info>

online: <https://ci.mines-stetienne.fr/cps2/course/tfsd/>

# Profiling the execution of your software: Why ?

- detect memory leaks
- detect application congestion points
- identify possible improvements

- threads, function calls
- number of objects, garbage collector
- connections to databases

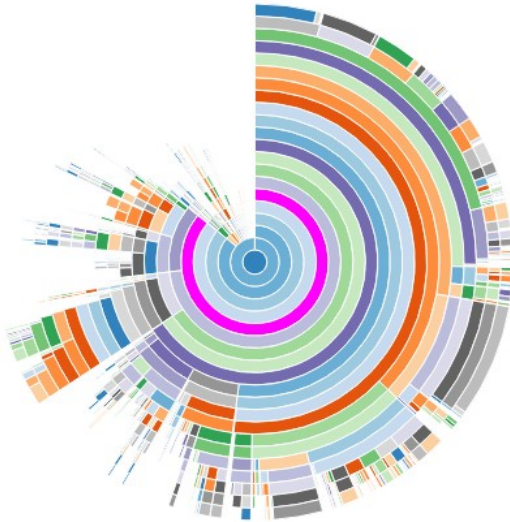


# Python

Reset

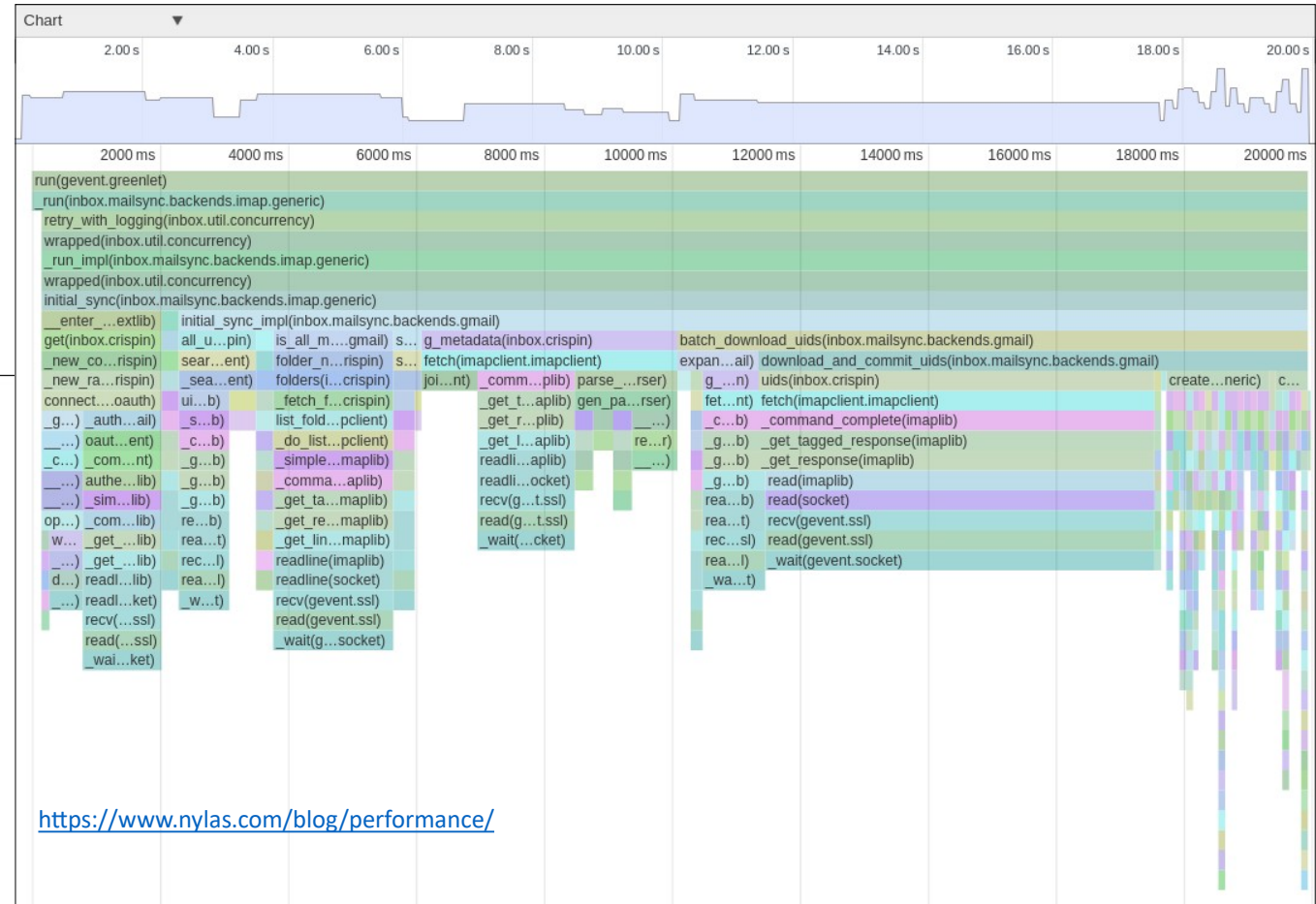
Style: **Sunburst** ▾  
Depth: **20** ▾  
Cutoff: **1/1000** ▾

**Name:**  
compile\_extra(typingctx:Context,  
targetctx:CPUContext, func:function,  
argstuple, return\_type:<unknown>,  
flags:Flags, locals:dict, library:<unknown>)  
**Cumulative Time:**  
1.38 s (99.70 %)  
**File:**  
compiler.py  
**Line:**  
677

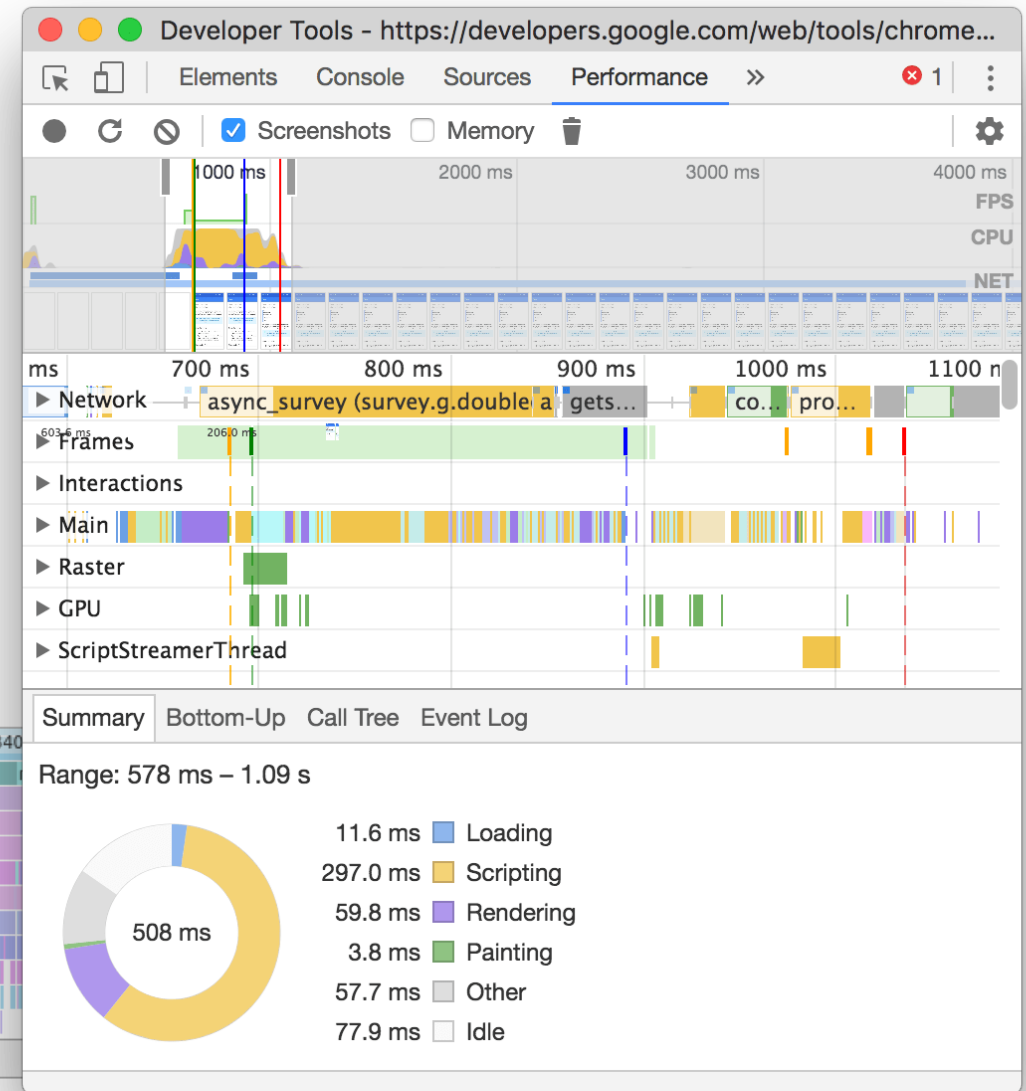
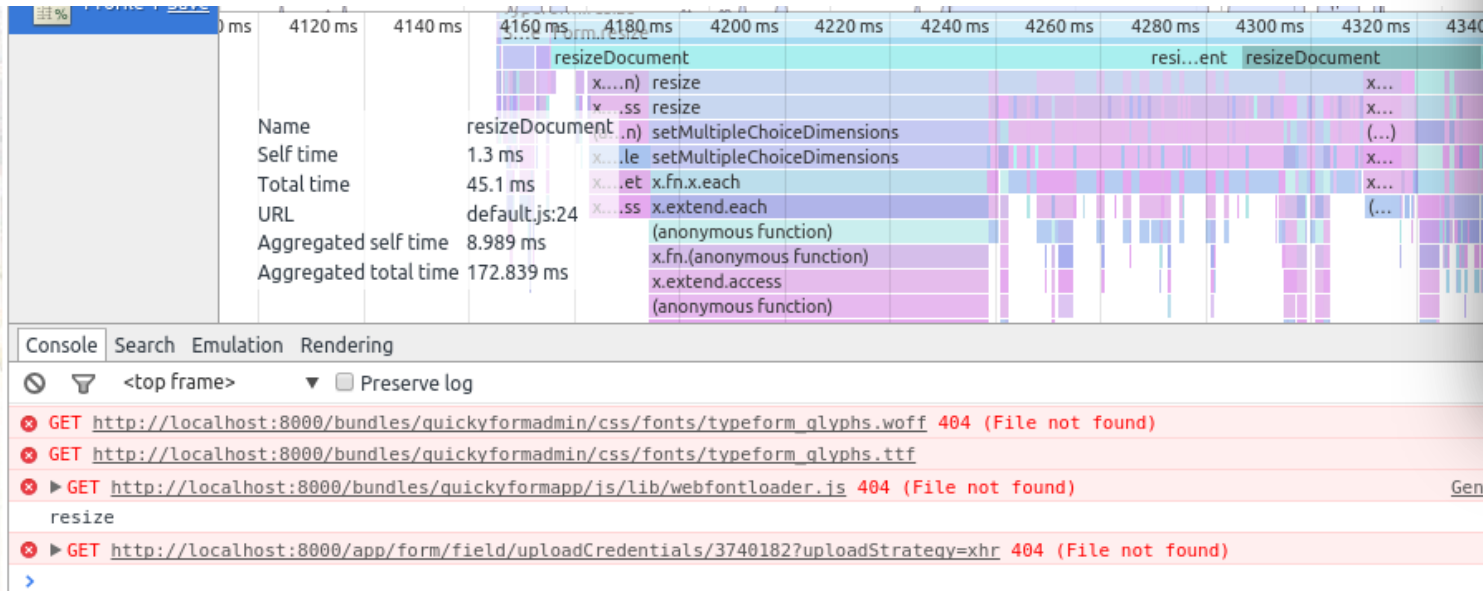


Search:

| ncalls    | tottime | percall   | cumtime | percall   | filename:lineno(function)   |
|-----------|---------|-----------|---------|-----------|---|
| 1851/26   | 0.04616 | 0.001775  | 0.08614 | 0.003313  | sre_compile.py:70(_compile(code:list, pattern:SubPattern, flags:int)) |
| 448/2     | 0.03374 | 0.01687   | 1.379   | 0.6896    | <ipython-input-3-4c49becbf5af>:8(<module>())                          |
| 706/16    | 0.03208 | 0.002005  | 0.07731 | 0.004832  | sre_parse.py:448(_parse(source:Tokenizer, state:Pattern))             |
| 4656/1164 | 0.03073 | 2.64e-05  | 0.03073 | 2.64e-05  | values.py:37(_wrapname(x:str))  |
| 5035/2109 | 0.02544 | 1.206e-05 | 0.02544 | 1.206e-05 | _utils.py:14(NameScope.is_used(name:str))                             |
| 8         | 0.02294 | 0.002868  | 0.04394 | 0.005493  | context.py:258(Context.install_registry(registry:Registry))           |



# JavaScript



# Technological foundations of software development

Debug, log, test, profile, analyze your software

Part 6: Static code analysis

ICM – Computer Science Major – Course unit on Technological foundations of computer science

M1 Cyber Physical and Social Systems – Course unit on CPS2 engineering and development, Part 2: Technological foundations of software development

Maxime Lefrançois <https://maxime-lefrancois.info>

online: <https://ci.mines-stetienne.fr/cps2/course/tfsd/>



# Dynamic vs. static analysis

## **Pros of dynamic analysis**

- is able to detect dependencies that cannot be detected in static analysis. Ex: dynamic dependencies by reflection, dependency injection, polymorphism.
- can collect temporal information.
- processes real input data. During static analysis it is difficult to impossible to know which files will be passed as input, which Web requests will come, which user will click, etc.

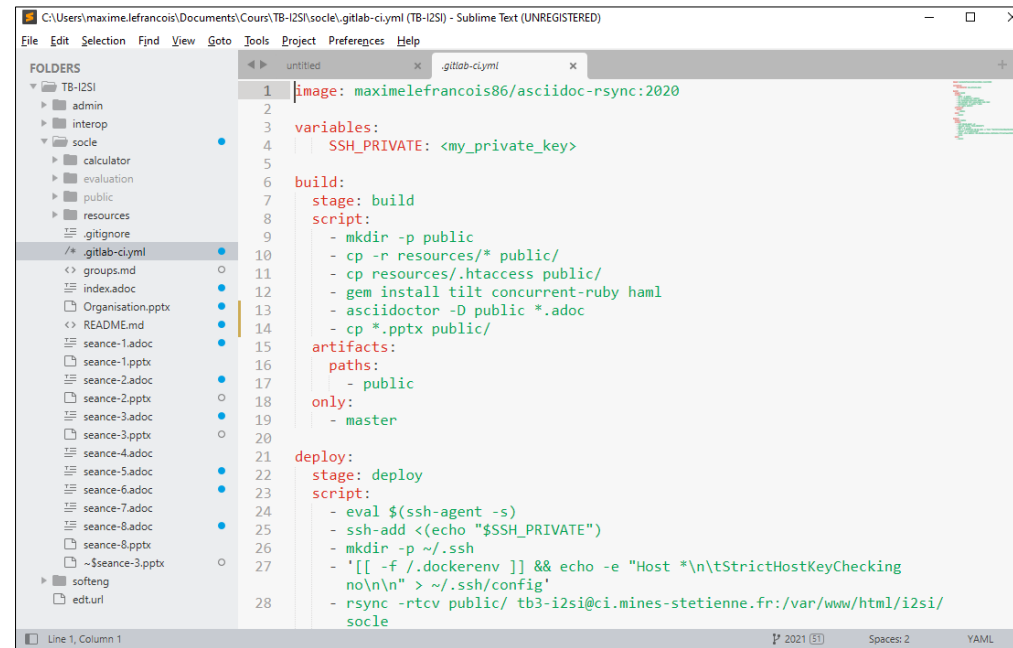
## **Cons of dynamic analysis**

- may have a negative impact on the performance of the application.
- cannot guarantee complete coverage of the source code, as executed, is based on user interaction or automated testing.



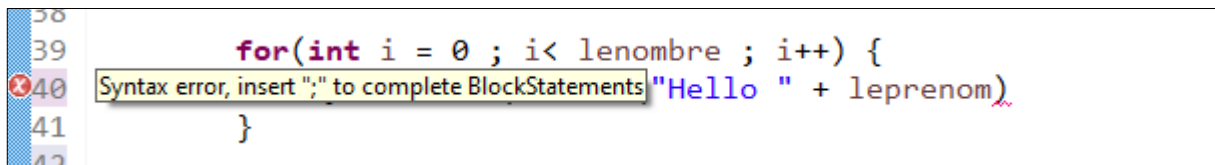
# Static code analysis

- Syntax checking
- Semantic check
- Linter
- Quick fixes



A screenshot of the Sublime Text editor window. The title bar indicates the file path is C:\Users\maxime\Documents\Cours\TB-I2SI\socle\gitlab-ci.yml. The left sidebar shows a file explorer with folders like 'admin', 'interop', 'socle', and 'resources'. The main editor area displays a YAML file with syntax highlighting. The content includes fields for 'image', 'variables' (SSH\_PRIVATE), 'build' (with stage and script), 'artifacts' (with paths and only), and 'deploy' (with stage and script). The status bar at the bottom shows 'Line 1, Column 1', '2021 (S)', 'Spaces: 2', and 'YAML'.

Syntax highlighting with Sublime Text

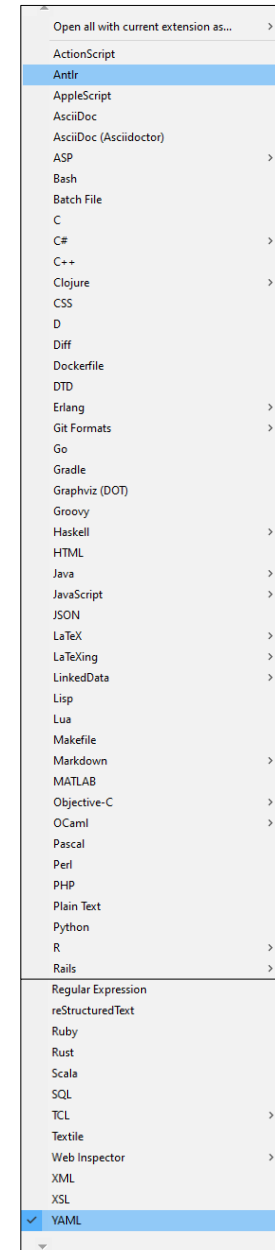


A screenshot of the Eclipse IDE showing a Java code snippet. The code is: 

```
38  
39     for(int i = 0 ; i< lenombre ; i++) {  
40         Syntax error, insert ";" to complete BlockStatements"Hello " + leprenom)  
41     }  
42
```

 A red 'x' icon is next to line 40, and a tooltip message says 'Syntax error, insert ";" to complete BlockStatements'. The status bar at the bottom indicates 'Syntax error, insert ";" to complete BlockStatements'.

Syntactic validation syntaxique with Eclipse



list of available grammars

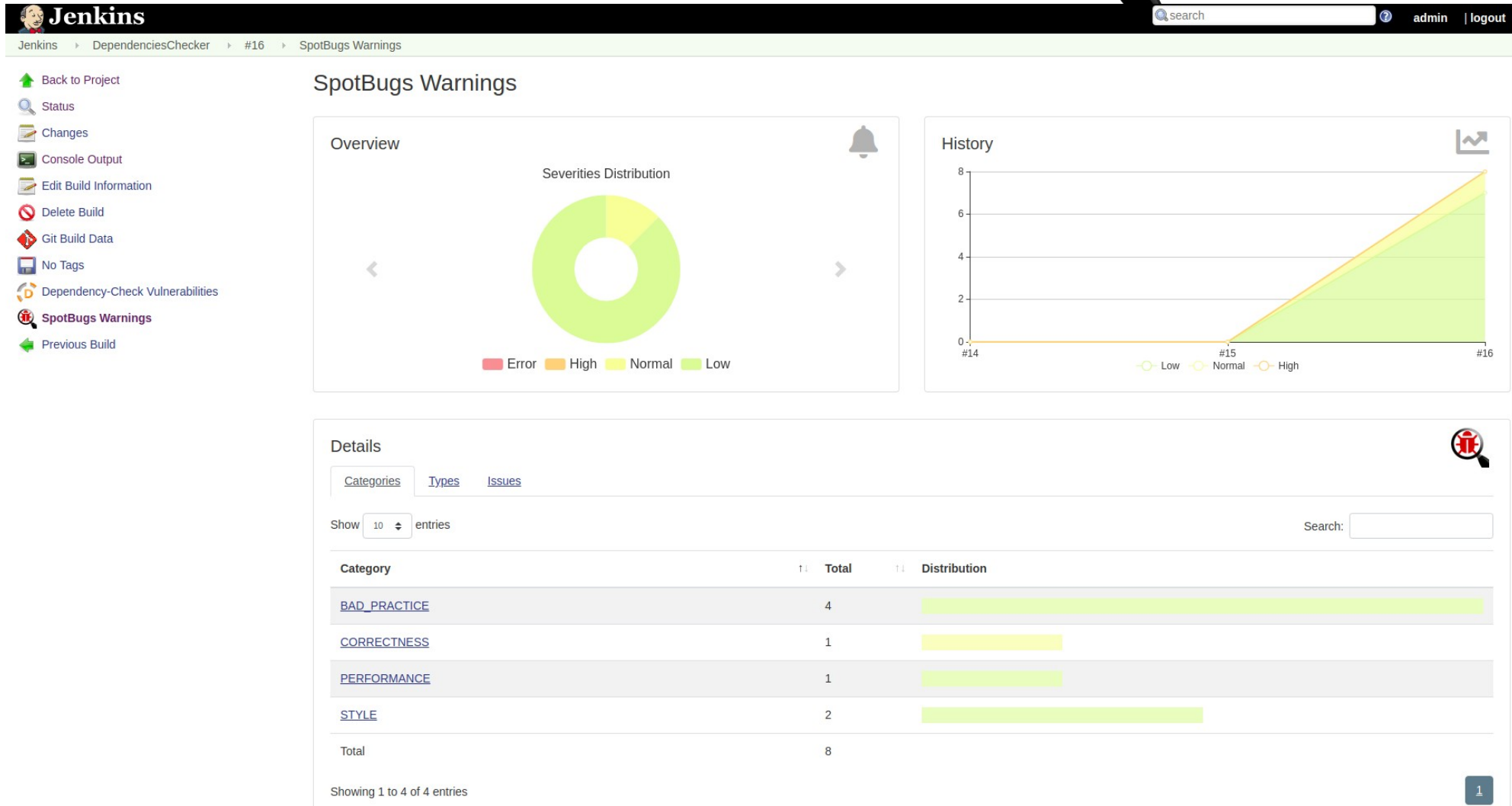
# Static analysis tools

- Example for Java

Java [edit]

| Tool ↕        | Latest release ↕    | Free software ↕                                     | Duplicate code ↕ | Notes   |
|---------------|---------------------|---|------------------|---|
| Checkstyle    | 2020-01-26          | Yes; LGPL   | No               | Besides some static code analysis, it can be used to show violations of a configured coding standard. Duplicate code detection was removed <sup>[10]</sup> from Checkstyle.   |
| Coverity      | 2017-01-19          | No; Proprietary                                     |                  | Coverity is a static analysis and Static Application Security Testing (SAST) platform that finds critical defects and security weaknesses in code as it's written before they become vulnerabilities, crashes, or maintenance issues. |
| Eclipse       | 2017-06-28          | Yes; EPL  | No               | Cross-platform IDE with own set of several hundred code inspections available for analyzing code on-the-fly in the editor and bulk analysis of the whole project. Plugins for Checkstyle, FindBugs, and PMD.                          |
| FindBugs      | 2015-03-06          | Yes; LGPL   |                  | Based on Jakarta BCEL from the University of Maryland. SpotBugs is the spiritual successor of FindBugs, carrying on from the point where it left off with support of its community.   |
| Infer         | 2017-10-19          | Yes; BSD with additional patent clause <sup>?</sup> |                  | Developed by an engineering team at Facebook with open-source contributors. Targets null pointer exceptions, leaks, and thread safety issues.   |
| IntelliJ IDEA | 2021-04-06          | Yes; ASL 2  | Yes              | A leading Java IDE with built-in code inspection and analysis. Plugins for Checkstyle, FindBugs, and PMD.   |
| JArchitect    | 2017-06-11          | No; Proprietary                                     |                  | Simplifies managing a complex code base by analyzing and visualizing code dependencies, defining design rules, doing impact analysis, and by comparing different versions of the code.  |
| Jtest         | 2019-05-21          | No; Proprietary                                     | Yes              | Testing and static code analysis product by Parasoft.   |
| LDRA Testbed  |                     | No; Proprietary                                     |                  | Analysis and testing tool suite.  |
| PMD           | 2020-10-24          | Yes; BSD, ASL 2, LGPL                               | Yes              | A static ruleset based source code analyzer that identifies potential problems.   |
| RIPS          | 2019-01-07          | No; Proprietary                                     |                  | Language-specific source code analysis solution with many integration options for accurate detection of complex security and quality issues.  |
| Semgrep       | 2021-03-16 (0.43.0) | Yes; LGPL v2.1                                      | No               | A static analysis tool that helps expressing code standards and surfacing bugs early. A CI service and a rule library is also available.  |
| Soot          | 2020-10-28          | Yes; LGPL   |                  | A language manipulation and optimization framework consisting of intermediate languages.  |
| Squale        | 2011-05-26          | Yes; LGPL   |                  | A platform to manage software quality.  |
| SourceMeter   | 2016-02-01          | No; Proprietary                                     | Yes              | A platform-independent, command-line static source code analyzer.   |
| ThreadSafe    | 2014-03-28          | No; Proprietary                                     |                  | A static analysis tool focused on finding concurrency bugs.   |

# Static analysis tools- e.g. **SpotBugs**



# Static analysis tools- e.g. SpotBugs

## SpotBugs Maven Plugin

SpotBugs Maven Plugin » Introduction

OVERVIEW

Introduction

Groovydoc

Goals

Usage

FAQ

EXAMPLES

Sample Report

Sample XML xdoc Report

Sample Legacy XML Report

Multi-Module Configuration

Violation Checking Configuration

PROJECT DOCUMENTATION

Project Information

CI Management

Dependency Information

Dependencies

Dependency Convergence

### SpotBugs Maven Plugin

**Please Note - This version is using Spotbugs 4.2.2.**

SpotBugs looks for bugs in Java programs. It is based on the concept of bug patterns. A bug pattern is a code idiom that is likely to be buggy.

- Difficult language features
- Misunderstood API methods
- Misunderstood invariants when code is modified during maintenance
- Garden variety mistakes: typos, use of the wrong boolean operator

SpotBugs uses static analysis to inspect Java bytecode for occurrences of bug patterns. We have found that SpotBugs is generally a good tool for finding bugs, but it does generate warnings that do not indicate real errors. In practice, the rate of false warnings reported by SpotBugs is generally low.

SpotBugs is free software, available under the terms of the Lesser GNU Public License. It is written in Java, and is originally developed by Bill Pugh. It is maintained by Bill Pugh, David Hovemeyer, and a team of volunteers.

SpotBugs uses BCEL to analyze Java bytecode. It uses dom4j for XML manipulation.

This introduction is an excerpt from the Facts Sheet at [SpotBugs home page](#).

To see more documentation about SpotBugs' options, please see the [SpotBugs Manual](#).

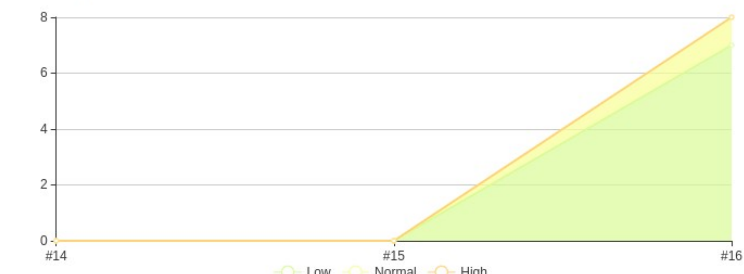
**Please Note**

As of version 3.1.0, you will need to use JDK 8 to run this plugin. This is a requirement imposed by Spotbugs.

search

admin | logout

History



| Severity | #14 | #15 | #16 |
|----------|-----|-----|-----|
| Low      | 0   | 0   | 4   |
| Normal   | 0   | 0   | 4   |
| High     | 0   | 0   | 0   |
| Total    | 0   | 0   | 8   |

Search:

| Category                     | Total | Distribution |
|------------------------------|-------|--------------|
| <a href="#">BAD_PRACTICE</a> | 4     | <div></div>  |
| <a href="#">CORRECTNESS</a>  | 1     | <div></div>  |
| <a href="#">PERFORMANCE</a>  | 1     | <div></div>  |
| <a href="#">STYLE</a>        | 2     | <div></div>  |
| Total                        | 8     |              |

Showing 1 to 4 of 4 entries

1

# Static analysis tools- e.g.



Apache Maven Project  
<http://maven.apache.org/>

Apache / Maven / Plugins / Apache Maven Checkstyle Plugin / Introduction

OVERVIEW

Introduction

Goals

Usage

FAQ

License

Download

Releases History

EXAMPLES

Upgrading Checkstyle at Runtime

Using an Inline Checkstyle Checker Configuration

Using a Custom Checkstyle Checker Configuration

Using Custom Checkstyle Property Expansion Definitions

Using a Suppressions Filter

## Apache Maven Checkstyle Plugin

The Checkstyle Plugin generates a report regarding the code style used by the developers. For this version of the plugin uses Checkstyle 8.29 by default and requires Java 8. But you can upgrade to a newer version of Checkstyle.

The plugin can be configured in the project's POM. Predefined rulesets available for use in the configuration.

### Goals Overview

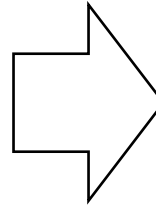
The Checkstyle Plugin has three goals:

- `checkstyle:checkstyle` is a reporting goal that performs Checkstyle analysis and generates a report.
- `checkstyle:checkstyle-aggregate` is a reporting goal that performs Checkstyle analysis and aggregates the results of multiple runs.
- `checkstyle:check` is a goal that performs Checkstyle analysis and outputs violations or a count of violations.

### Major Version Upgrade to version 3.0.0

Please note that the following parameters have been completely removed from the plugin configuration:

- `sourceDirectory`: use `sourceDirectories` instead;
- `testSourceDirectory`: use `testSourceDirectories` instead.



Files

| Files                      | T | W | E  |
|----------------------------|---|---|----|
| com/maventest/Howdy.java   | 0 | 0 | 12 |
| com/maventest/App.java     | 0 | 0 | 10 |
| com/maventest/package.html | 0 | 0 | 1  |

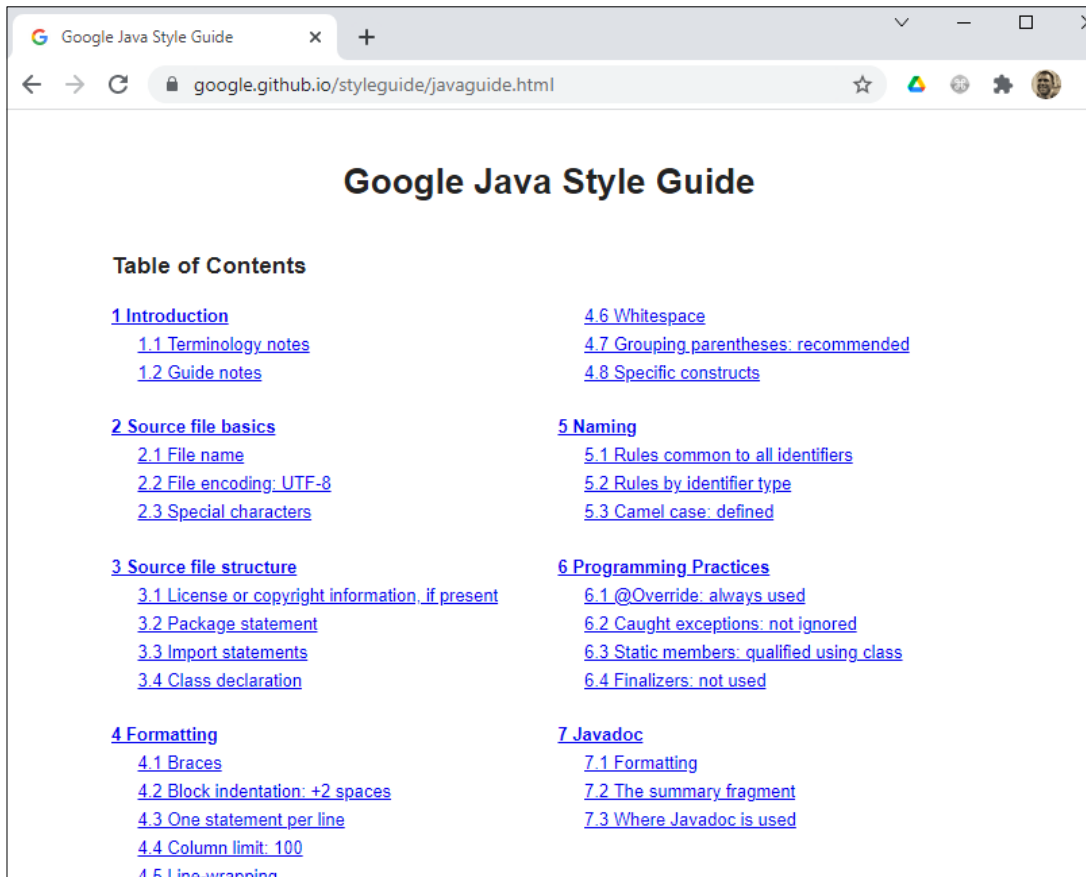
Details

com/maventest/Howdy.java

| Violation | Message  | Line |
|-----------|--|------|
| ✖         | File does not end with a newline.                                | 0    |
| ✖         | Missing a Javadoc comment.                                       | 3    |
| ✖         | Utility classes should not have a public or default constructor. | 3    |
| ✖         | Line contains a tab character.                                   | 5    |
| ✖         | Missing a Javadoc comment.                                       | 5    |
| ✖         | Parameter args should be final.                                  | 5    |
| ✖         | Line contains a tab character.                                   | 6    |
| ✖         | Line contains a tab character.                                   | 7    |
| ✖         | Line contains a tab character.                                   | 9    |
| ✖         | Missing a Javadoc comment.                                       | 9    |
| ✖         | Line contains a tab character.                                   | 10   |
| ✖         | Line contains a tab character.                                   | 11   |

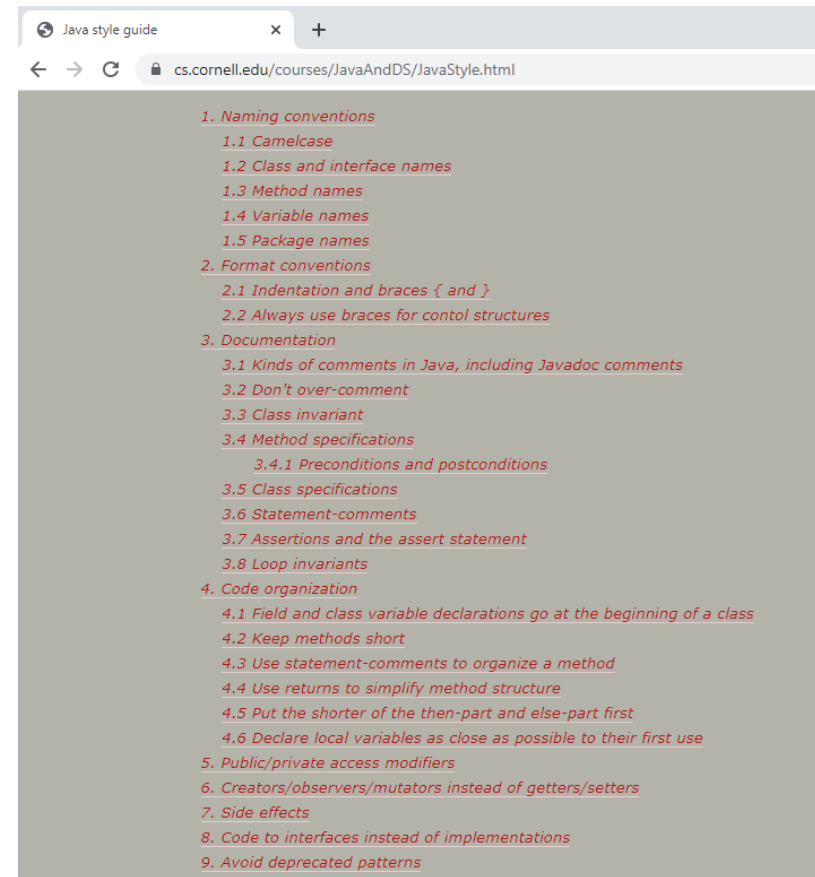
# Develop with style !

*"There are two ways of constructing a software design. One way is to make it so simple that there are obviously no deficiencies. The other way is to make it so complicated that there are no obvious deficiencies." - C.A.R. Hoare.*



A screenshot of the Google Java Style Guide website. The browser tab is titled "Google Java Style Guide" and the address bar shows "google.github.io/styleguide/javaguide.html". The page has a white background with a black title "Google Java Style Guide". Below the title is a "Table of Contents" section with a list of links organized into numbered sections from 1 to 7. The links are in blue text.

| Table of Contents  |   |
|--|---|
| <a href="#">1 Introduction</a>                                   | <a href="#">4.6 Whitespace</a>                            |
| <a href="#">1.1 Terminology notes</a>                            | <a href="#">4.7 Grouping parentheses: recommended</a>     |
| <a href="#">1.2 Guide notes</a>                                  | <a href="#">4.8 Specific constructs</a>                   |
| <a href="#">2 Source file basics</a>                             | <a href="#">5 Naming</a>                                  |
| <a href="#">2.1 File name</a>                                    | <a href="#">5.1 Rules common to all identifiers</a>       |
| <a href="#">2.2 File encoding: UTF-8</a>                         | <a href="#">5.2 Rules by identifier type</a>              |
| <a href="#">2.3 Special characters</a>                           | <a href="#">5.3 Camel case: defined</a>                   |
| <a href="#">3 Source file structure</a>                          | <a href="#">6 Programming Practices</a>                   |
| <a href="#">3.1 License or copyright information, if present</a> | <a href="#">6.1 @Override: always used</a>                |
| <a href="#">3.2 Package statement</a>                            | <a href="#">6.2 Caught exceptions: not ignored</a>        |
| <a href="#">3.3 Import statements</a>                            | <a href="#">6.3 Static members: qualified using class</a> |
| <a href="#">3.4 Class declaration</a>                            | <a href="#">6.4 Finalizers: not used</a>                  |
| <a href="#">4 Formatting</a>                                     | <a href="#">7 Javadoc</a>                                 |
| <a href="#">4.1 Braces</a>                                       | <a href="#">7.1 Formatting</a>                            |
| <a href="#">4.2 Block indentation: +2 spaces</a>                 | <a href="#">7.2 The summary fragment</a>                  |
| <a href="#">4.3 One statement per line</a>                       | <a href="#">7.3 Where Javadoc is used</a>                 |
| <a href="#">4.4 Column limit: 100</a>                            |   |
| <a href="#">4.5 Line-wrapping</a>                                |   |



A screenshot of the Java style guide website. The browser tab is titled "Java style guide" and the address bar shows "cs.cornell.edu/courses/JavaAndDS/JavaStyle.html". The page has a light gray background with a list of links organized into numbered sections from 1 to 9. The links are in red text.

|  |
|--|
| <a href="#">1. Naming conventions</a>  |
| <a href="#">1.1 Camelcase</a>  |
| <a href="#">1.2 Class and interface names</a>  |
| <a href="#">1.3 Method names</a>   |
| <a href="#">1.4 Variable names</a>   |
| <a href="#">1.5 Package names</a>  |
| <a href="#">2. Format conventions</a>  |
| <a href="#">2.1 Indentation and braces { and }</a>                                       |
| <a href="#">2.2 Always use braces for control structures</a>                             |
| <a href="#">3. Documentation</a>   |
| <a href="#">3.1 Kinds of comments in Java, including Javadoc comments</a>                |
| <a href="#">3.2 Don't over-comment</a>   |
| <a href="#">3.3 Class invariant</a>  |
| <a href="#">3.4 Method specifications</a>  |
| <a href="#">3.4.1 Preconditions and postconditions</a>                                   |
| <a href="#">3.5 Class specifications</a>   |
| <a href="#">3.6 Statement-comments</a>   |
| <a href="#">3.7 Assertions and the assert statement</a>                                  |
| <a href="#">3.8 Loop invariants</a>  |
| <a href="#">4. Code organization</a>   |
| <a href="#">4.1 Field and class variable declarations go at the beginning of a class</a> |
| <a href="#">4.2 Keep methods short</a>   |
| <a href="#">4.3 Use statement-comments to organize a method</a>                          |
| <a href="#">4.4 Use returns to simplify method structure</a>                             |
| <a href="#">4.5 Put the shorter of the then-part and else-part first</a>                 |
| <a href="#">4.6 Declare local variables as close as possible to their first use</a>      |
| <a href="#">5. Public/private access modifiers</a>                                       |
| <a href="#">6. Creators/observers/mutators instead of getters/setters</a>                |
| <a href="#">7. Side effects</a>  |
| <a href="#">8. Code to interfaces instead of implementations</a>                         |
| <a href="#">9. Avoid deprecated patterns</a>   |

# Technological foundations of software development

Debug, log, test, profile, analyze your software

Part 7: Integration in platforms

ICM – Computer Science Major – Course unit on Technological foundations of computer science

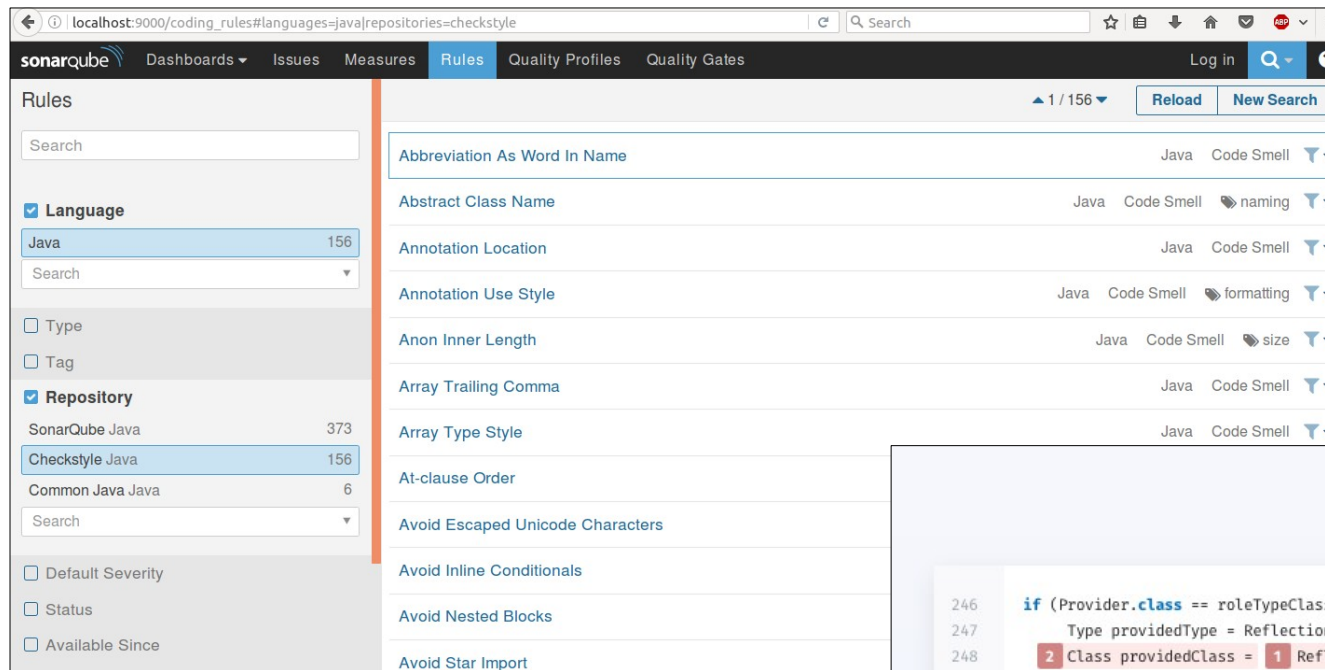
M1 Cyber Physical and Social Systems – Course unit on CPS2 engineering and development, Part 2: Technological foundations of software development

Maxime Lefrançois <https://maxime-lefrancois.info>

online: <https://ci.mines-stetienne.fr/cps2/course/tfsd/>



# Code quality and security analysis platform





# Technological foundations of software development

Debug, log, test, profile, analyze your software

ICM – Computer Science Major – Course unit on Technological foundations of computer science

M1 Cyber Physical and Social Systems – Course unit on CPS2 engineering and development, Part 2: Technological foundations of software development

Maxime Lefrançois <https://maxime-lefrancois.info>

online: <https://ci.mines-stetienne.fr/cps2/course/tfsd/>

# ... Your turn

Complete the TODO section:

[https://ci.mines-stetienne.fr/cps2/course/tfsd/course-5.html#\\_todos](https://ci.mines-stetienne.fr/cps2/course/tfsd/course-5.html#_todos)