PG4200: Algorithms And Data Structures

Lesson 01: Arrays, Lists, and Unit Tests

Prof. Andrea Arcuri

About Me

Prof. Andrea Arcuri

[simula . research laboratory]
- by thinking constantly about it

WesternGeco

telenor

uni.lu
UNIVERSITÉ DU LUXEMBOURG

Contact

- "*Discussion Forum*" on Canvas
- For announcements, and questions of general interest for the whole class
- Use the discussion forum **instead of** sending me emails
  - If you send me a private email/message, I will tell you to post it on the Discussion Forum. However, if I am busy (as most of the time…), *I might just ignore your message…*

Course Info

- 12 lessons, once a weak

- Class 1-9: *Foundation*, algorithms and data structures that all of you will need to know if you are going to work as a developer/programmer/engineer/etc.

- Class 10-12: *Advanced*, interesting and important topics, but that not all of you will need in your daily jobs

- Check each week to see if changes in schedule (time and room)

Foundation

1. Arrays/Lists/Tests
2. Stacks/Queues
3. Runtime analysis and Sorting
4. Recursion and TDD
5. Tree Maps
6. Hash Maps
7. Streams
8. Graphs
9. Regular Expressions

Advanced

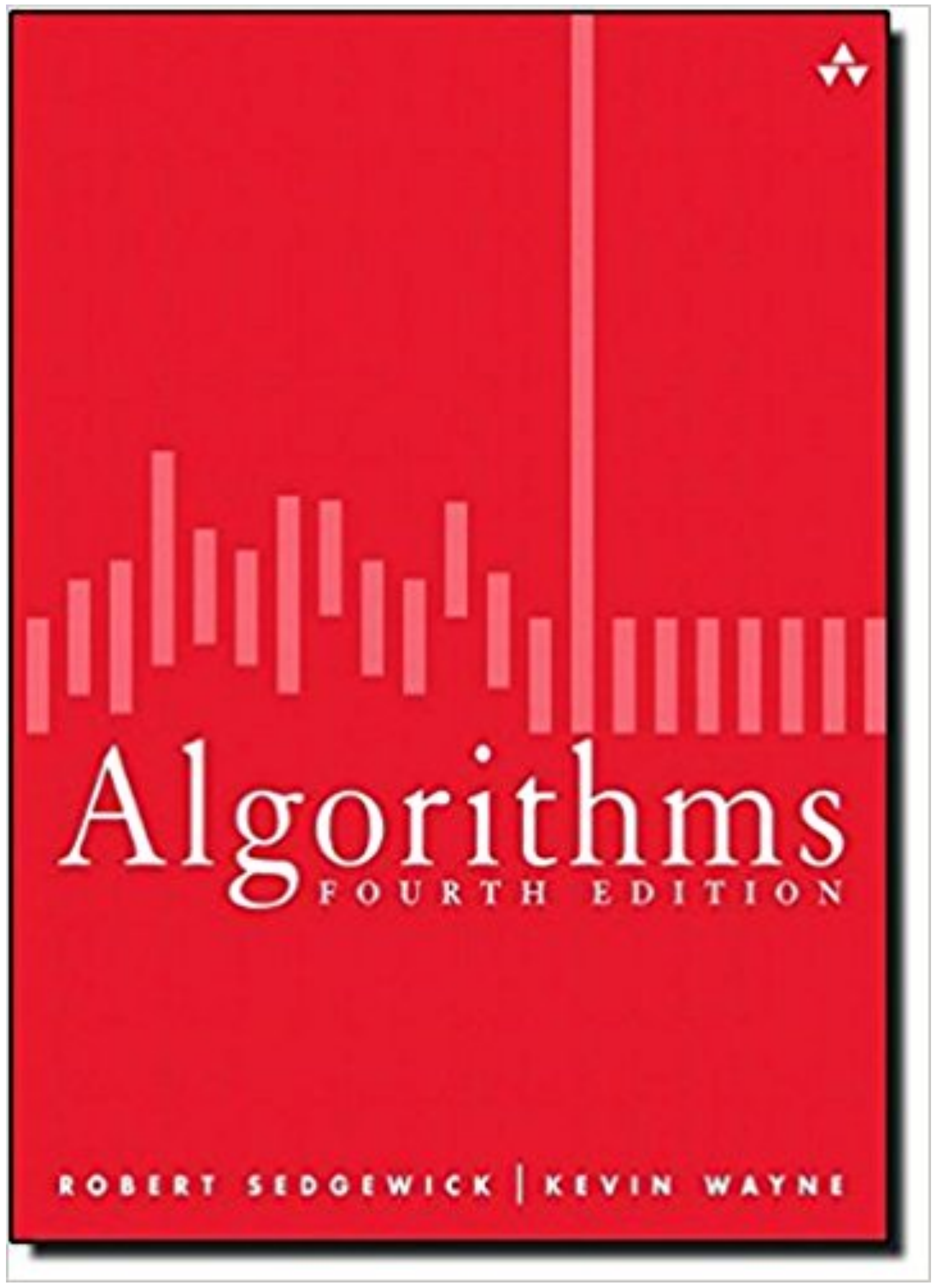
10. Optimization Algorithms
11. Evolutionary Algorithms
12. Data Compression

Class Structure

- “Usually” 2+2
  - 2-3 hours of lecture: code and slides
  - 1-2 hours in which you should do exercises and get help
- IMPORTANT: the 1-2 hours after lecture is not only for exercises. If you are falling behind, or you need some more revision, you can ask for my help on anything related to coding

Course Material

- Algorithms (4th Edition)
- We **actually use it** in the course, so *you should really try to get a copy*
    - and likely you ll need it also outside of this course
- Note, however, that there are plenty of resources on internet to learn Algorithms



Git Repository

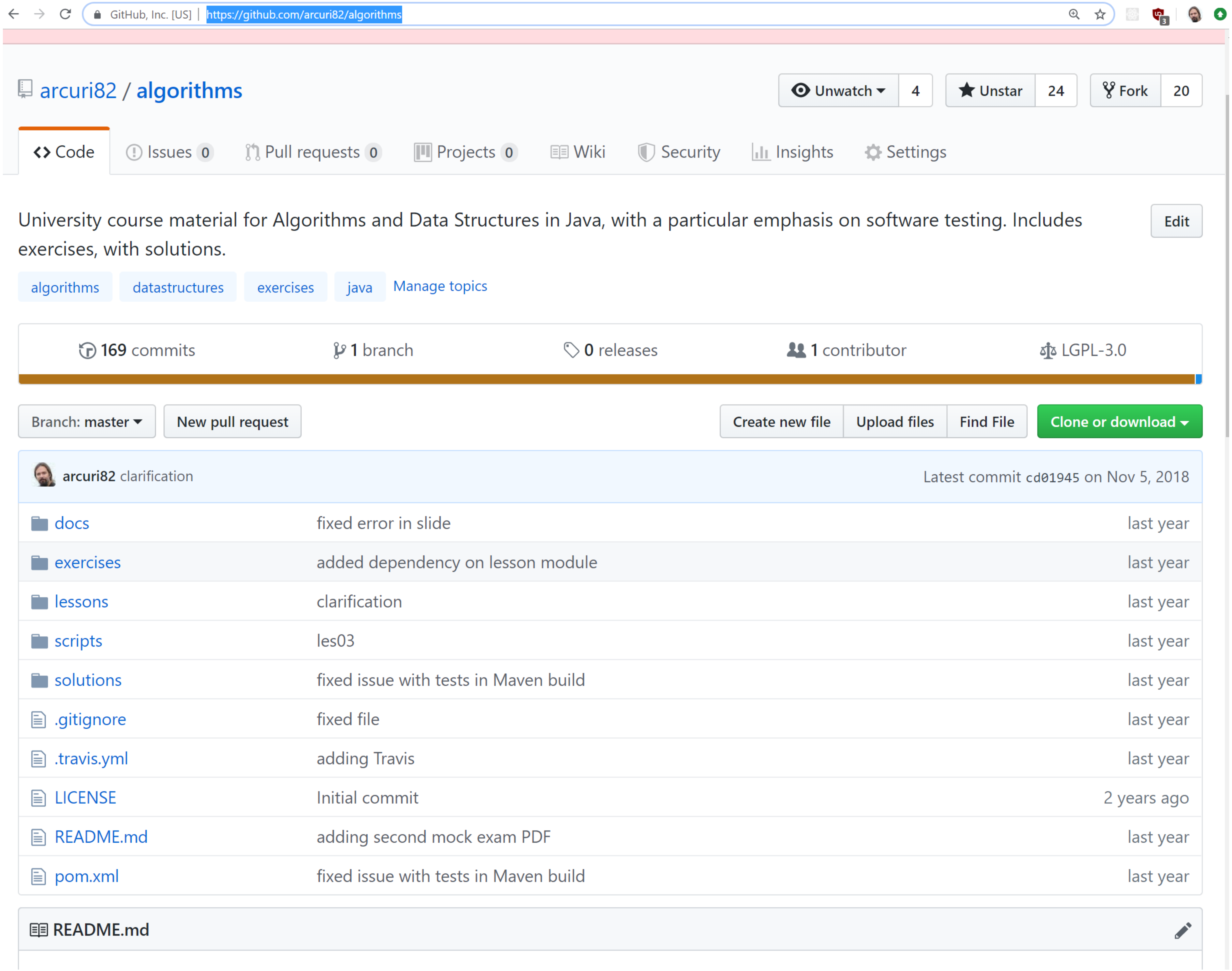
- https://github.com/arcuri82/algorithms
- Note: pull often, as new material and corrections can be added during the course
- If you add code (eg working on exercises), recall to do a Git "*rebase*" or "*stash*" before pulling (which otherwise might fail)

Git

- *Git* is a tool to share code among different developers in the same project
- Also useful for single developers to keep track of changes, and automatically have backups on remote servers
- You should have already seen the details of *Git* in other courses…
- … but I am using *Git* to handle all the teaching material in this course
- Note: usage of *Git* will **NOT** be part of the exam…

GitHub

- Currently the main server repository for hosting open-source projects
    - Before, the main one was *SourceForge*
- *GitHub* provides a website in which projects can be browsed
- Projects on *GitHub* are handled with Git
- *GitHub* is most famous/used, but there are others as well
    - eg, *BitBucket* and *GitLab*


https://github.com/arcuri82/algorithms

arcuri82 / algorithms

Unwatch 4 | Unstar 24 | Fork 20

Code | Issues 0 | Pull requests 0 | Projects 0 | Wiki | Security | Insights | Settings

University course material for Algorithms and Data Structures in Java, with a particular emphasis on software testing. Includes exercises, with solutions.

algorithms datastructures exercises java Manage topics

169 commits | 1 branch | 0 releases | 1 contributor | LGPL-3.0

Branch: master | New pull request | Create new file | Upload files | Find File | Clone or download

arcuri82 clarification — Latest commit cd01945 on Nov 5, 2018

docs	fixed error in slide	last year
exercises	added dependency on lesson module	last year
lessons	clarification	last year
scripts	les03	last year
solutions	fixed issue with tests in Maven build	last year
.gitignore	fixed file	last year
.travis.yml	adding Travis	last year
LICENSE	Initial commit	2 years ago
README.md	adding second mock exam PDF	last year
pom.xml	fixed issue with tests in Maven build	last year

README.md

Git: What You Need To Do

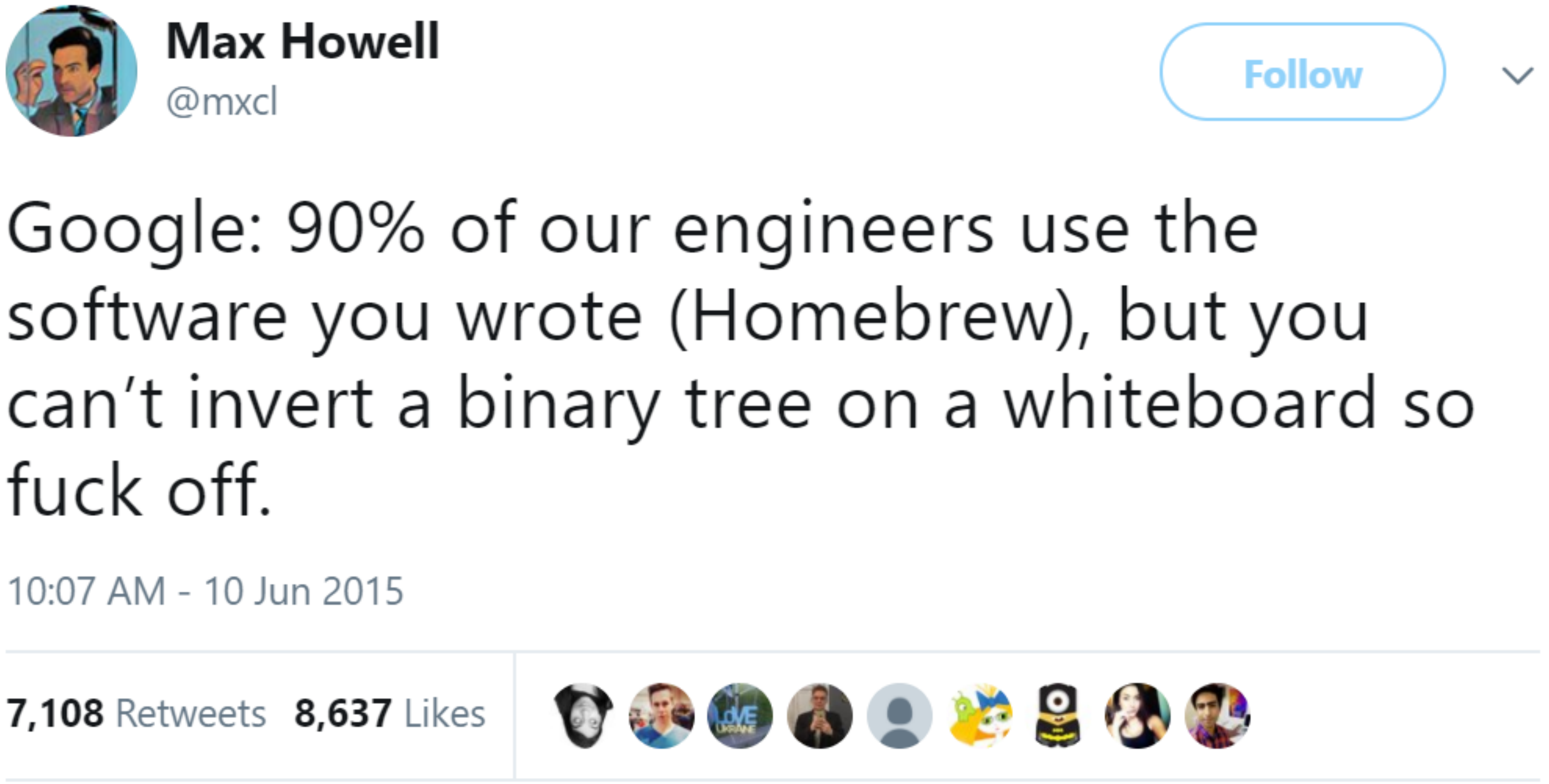
- Install *Git*, if you don’t have it yet
- **git clone https://github.com/arcuri82/algorithms.git**
    - clone the repository on your local machine
- **git pull**
    - update your local copy with the latest changes in the repository
- Those commands can be run from a terminal, or from your IDE (eg, IntelliJ)

Why Studying Algorithms?

- Algorithms and data structures are the foundation of programming
    - ie, the base building blocks
- Impact on all fields of engineering and science
    - internet, computer graphics, social networks, biology, physics, etc.
- In this course, we will not build whole applications (eg, web or mobile), but rather concentrate on the building blocks to enable it in the next courses

Like it or not…

- … used in practically most programs you will write
- … algorithms and data structures are very common exercises in job interviews
    - Especially for juniors straight out of university
    - Don’t be surprised to be asked to write a stack or a queue class on a whiteboard…
    - … or other advanced algorithms



Max Howell
@mxcl

Google: 90% of our engineers use the software you wrote (Homebrew), but you can’t invert a binary tree on a whiteboard so fuck off.

10:07 AM - 10 Jun 2015

7,108 Retweets 8,637 Likes

Math

- (un)fortunately there is **math** involved…
- Math: can tell you **WHY** a particular algorithm or data structure performs in a certain way
  - As an engineer, you need to make conscious decisions about what you use
- I like math, but, in contrast to lecturers of previous years or in other universities, I put more emphasis on the programming side…

Coding

- This course is heavily based on coding
- There are going to be slides, but in class we will spend most of the time going through source code
- Slides will often just be a quick overview of what we will cover in the code

Necessary Tools

- Java 11 JDK
  - use *OpenJDK*, eg downloaded from https://adoptopenjdk.net/
- *Git*
- *IntelliJ Ultimate Edition*
  - you might want to install *JetBrains Toolbox* first
  - anyway, any other IDE would do, eg *Eclipse* and *NetBeans*

Java

- In this course, **Java** is used as programming language for the examples and exercises
- The concepts of Algorithms do apply to **any** programming language, and this is **NOT** a course on Java

Why Java?

- Need *object-oriented* language that is **strongly typed**
- **Java:** one of the most popular languages, and you have already seen it in previous courses
- **Kotlin**: great language (my favorite), but too advanced
- **C#:** would had been a great choice as well
- **C++**: good choice, but can get tricky when dealing with memory allocation issues and OS dependent
- **JavaScript**: HELL NO!!! There is a limit to sadism…
- **Python**: not statically typed
- **Go**: no *Generic* types

If You Skip Class…

- Usually acceptable that a student skips 1-2 classes
- You are supposed to attend, although no strict checks
- If you skip too many classes, it is **YOUR** responsibility to catch up and find out what done in class
    - you are adults, after all…

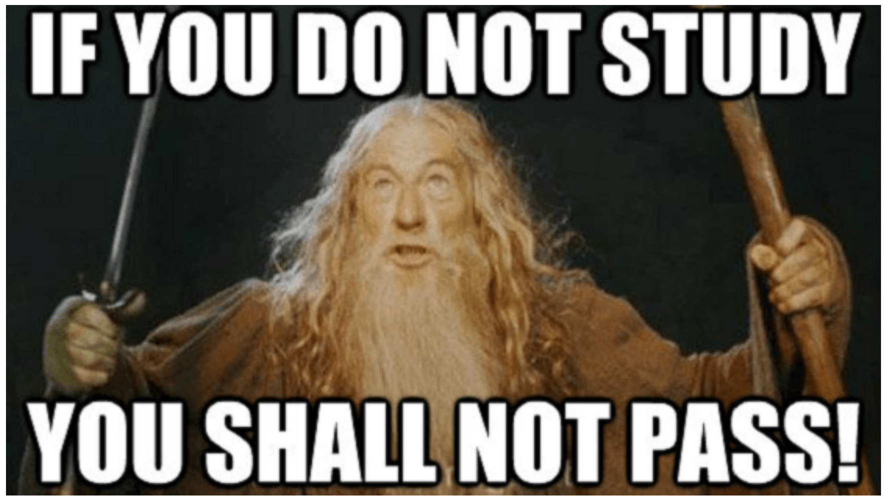
Exams

- 3 hour written exam
- Expect around 10 questions/exercises
  - Based on slides and all code in the repository
  - Typically only 1 question from the Advanced Topics
  - Note: this might change… but if so, you will be told before the exam
  - See mock exams in the Git repository
- Expect theoretical questions and also the writing of code from scratch (at least 1, but no more than 50%)
- The exam is based on what covered in the Git repository
  - the book just gives you extra info and clarifications
  - exam is based on code in the Git repository, NOT the one of the book

Code In The Exam

- There are 12 classes which you need to know by heart, and be able to write from scratch
  - *MyLinkedList, MyStackLinkedList, MyQueueArray, BubbleSort, InsertionSort, MergeSort, QuickSort, MyMapBinarySearchTree, MyHashMapWithLists, MyStreamSupport, UndirectedGraph, TextSearchKMP*
  - In the exercises, you will be asked to write them on paper
  - You can expect 1-3 of them ending up in the exam
- Note: you can still get questions from any of the code in the repository
  - but *usually* in those cases it is just to complete the code from a starting snippet, or find bugs in them

Difficulty



- This is a **difficult** course, more difficult than what you might be used to and expect
- Not uncommon that *many* students **fail** the exam
- You need to study **each week**, and do **all** the exercises
- Forget trying to learn it in just a couple of weeks/days before the exam…
    - I keep saying it every time, but students do not believe me, and then are surprised when they get an F…

Typical Exam Results

- **40%** score for **E**, and **90%** for **A**
- But that would usually mean **60%-80%** of students get an **F**, and top grade is a **C**, as *most students underestimate this course*
  - eg, naively believe that can start studying just few days/weeks before the exam
- **Rescaling**: usually *not failing* more than **50%** of students, and top scores get an **A**
  - eg, typically after rescaling, **25%** for **E**, and **75%** for **A**
  - Rescaling does **NOT** apply to "*continuation*" exams
- *Strongly suggest to have a chat with students that have taken this course before*

Arrays and Lists

Containers

- When writing programs, need to manipulate data
  - e.g., adding songs to a playlist
  - e.g., adding an item on a shopping cart like on Amazon
- *Adding*, *removing* and *searching* for elements
- Different *data structures* with different properties
- In this course, we will see **Arrays**, **Lists**, **Maps**, **Sets** and **Graphs**

Arrays

```
int[] array = new int[3];
array[0] = 5;
array[1] = 3;
```

[0]	[1]	[2]
5	3	0

- Easy, direct access to all elements
- Possible issues when deleting elements (ie holes)
- Fixed size, decided at creation
    - If you create it for 10 elements, but then you need 11, you would need to create a new array
- Arrays are low-level constructs of Java language

Lists

- Conceptually like arrays, but no fixed size
    - ie, you can add as many elements as you want, as long as you have enough memory
- Ordered sequence of elements, from index 0 to N-1
    - can have duplicates
- Lists (and all data structures will see in this course) are Java objects, and not treated specially like arrays

List Object Operations

- size(): return how many elements are contained
- get(index): return the value at position index
- add(index, value): add value at position index
  - existing value in that position is right-shifted (with all following elements) to do not lose it
  - add(value) just add at the end of the list
- deleted(index): remove element at position index
  - all following elements are left-shifted to cover the hole

List Implementations

- 2 main ways to "*implement*" a list object
- *ArrayList*: internally storing an array
  - All operations mapped to an internal array
  - Special: need to create new internal array and move over old data when full. But this can be done automatically when new element is added, without the client knowing about it
- *LinkedList*: each element has its own node object
  - nodes are connected with object pointers/links

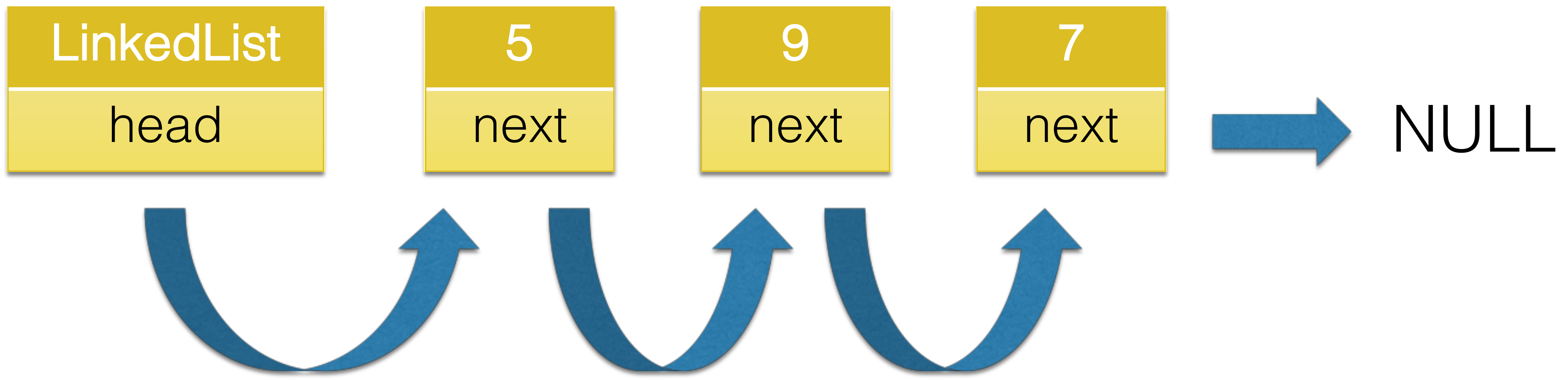
ArrayList

```
List list = new ArrayList(5);
list.add(5);
list.add(9);
list.add(7);
assert list.size() == 3;
```

[0]	[1]	[2]	[3]	[4]
5	9	7		

- In this example, internally the ArrayList would use an array of size 5
- But only 3 elements are contained in the "*list*"

LinkedList



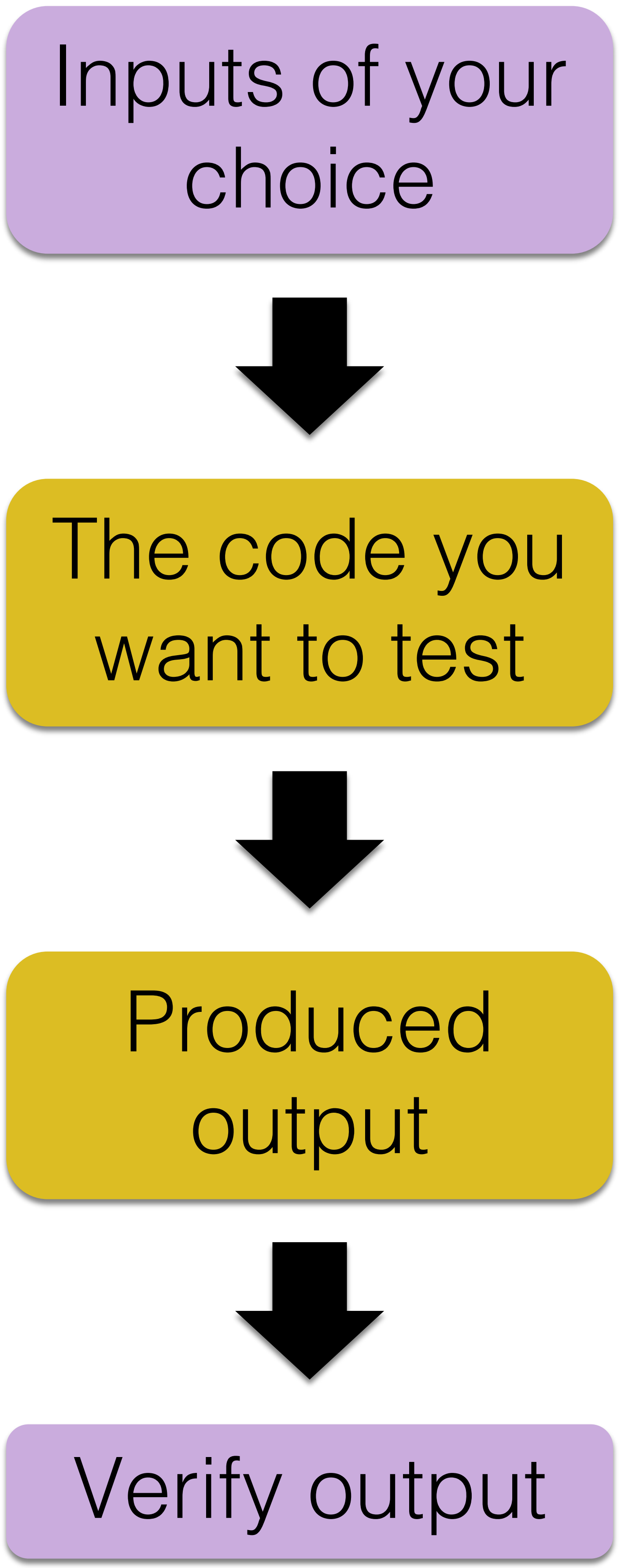
- A *node* for each element (they are objects)
- *Links* from node to node (eg, in a *next* field)
- *LinkedList* object only contains a *head* link to first node in the list
- Dynamic, not fixed size
- Accessing elements more difficult, as need to traverse the links
- Going into more details on *links* and *memory* in the *next class*
    - reason is that you first need to learn about *Stacks*
    - so in this class we only start to see a subset of functionalities for linked lists

Unit Testing

Bugs

- Software has bugs, ie, errors/mistakes
- Not just students, but also professional engineers with decades of experiences make mistakes, quite often…
    - not necessarily because they are bad, but just that code nowadays can become very, very complex
- You want to check if the code you write is actually doing what it is supposed to do

Testing



- Cannot guarantee the code is correct, but can increase your confidence in it
- You want the checking of your code to be automated
- In each test case, you **verify** the **output** generated when you run the **code** with the **inputs** of your choice

Writing Unit Tests

- Using a library called **JUnit**
  - Note: how to configure *Maven* to import third-party libraries is not part of this course (and so not on the exam), but you can ask me in the breaks if you are curious (for some of you, we will dig into its low level details in Enterprise Programming 1 next semester)
- Regular code in “*src/main/java*” folder
- Test code in “*src/test/java*” folder
- A test class is just a Java class with *@ annotations*
- A test class for a class called *Foo.java* will be called *FooTest.java*, in the same package

Main @ Annotations

- **@Test**: mark a method as a test
- **@BeforeEach**: execute method before each test
- **@BeforeAll**: execute method once before any of the tests is started
- **@AfterEach**, **@AfterAll**: same, but after the tests
- **@Disable**: temporarily disable a test, which is not going to be run

Assertions

- When you have an output, you need to *verify* if correct
- Extra code (assertion methods) that throws an error if the output is not equal to the expected one
- *assertEquals(expected, output)*
  - throw error if *output* variable is not equal to the *expected* one
- *assertTrue(condition)*
  - throw error if *condition* is false
- *assertNotNull(output)*
  - throw error if *output* is null

Test Example

Mark method as a test

```
@Test
public void testBase(){

    int[] array = {1, 2, 3};

    int res = ArrayExample.sum(array);

    assertEquals(6, res);
}
```

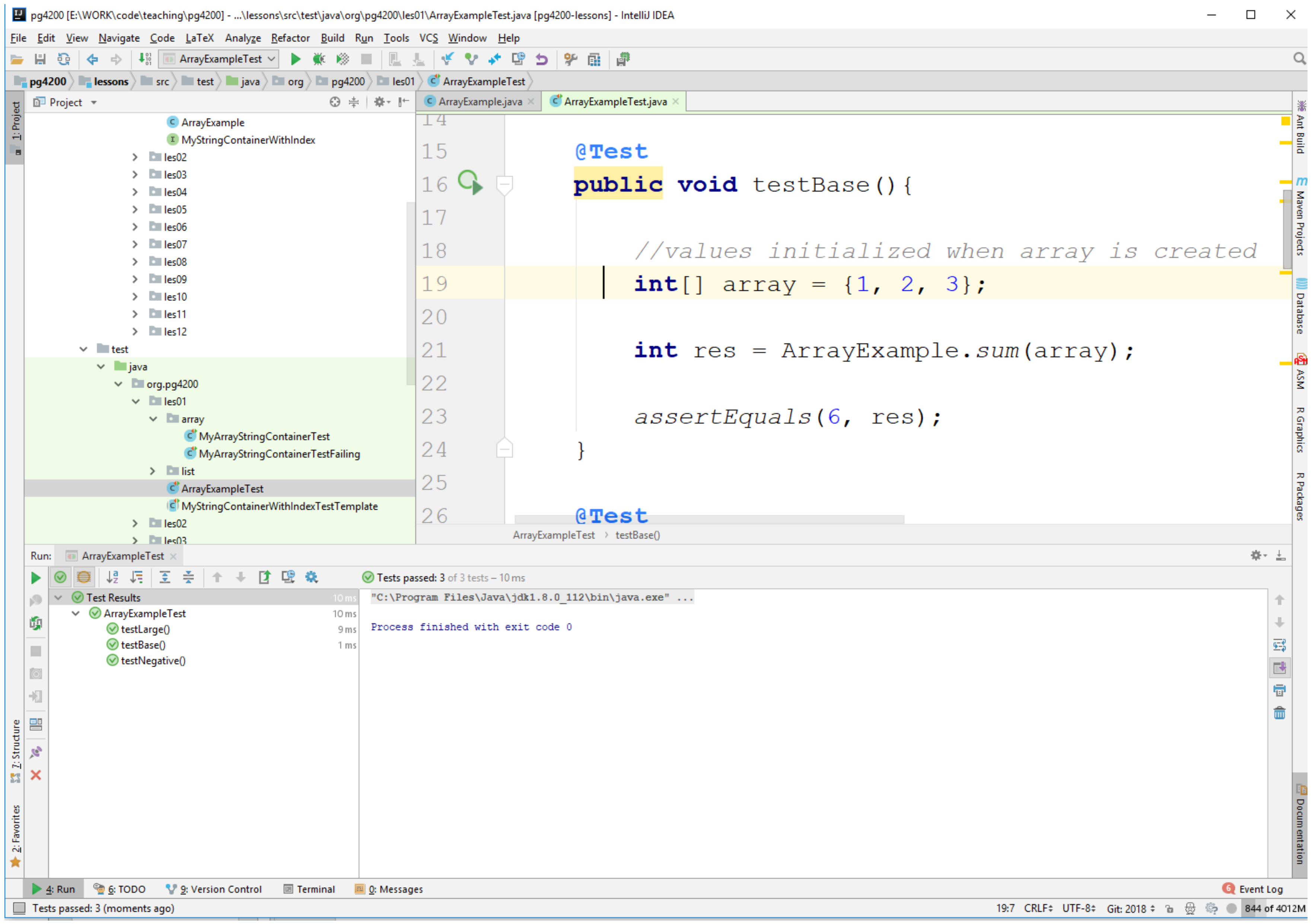
Input data

Code execution

Verify output

Running a Test

- Right-click, and choose "*Run <ClassName>*"
- Can also use "*Debug*" and "*Run With Coverage*"



pg4200 [E:\WORK\code\teaching\pg4200] - ...\lessons\src\test\java\org\pg4200\les01\ArrayExampleTest.java [pg4200-lessons] - IntelliJ IDEA

```java
    @Test
    public void testBase(){

        //values initialized when array is created
        int[] array = {1, 2, 3};

        int res = ArrayExample.sum(array);

        assertEquals(6, res);
    }

    @Test
```

Run: ArrayExampleTest

Tests passed: 3 of 3 tests – 10 ms

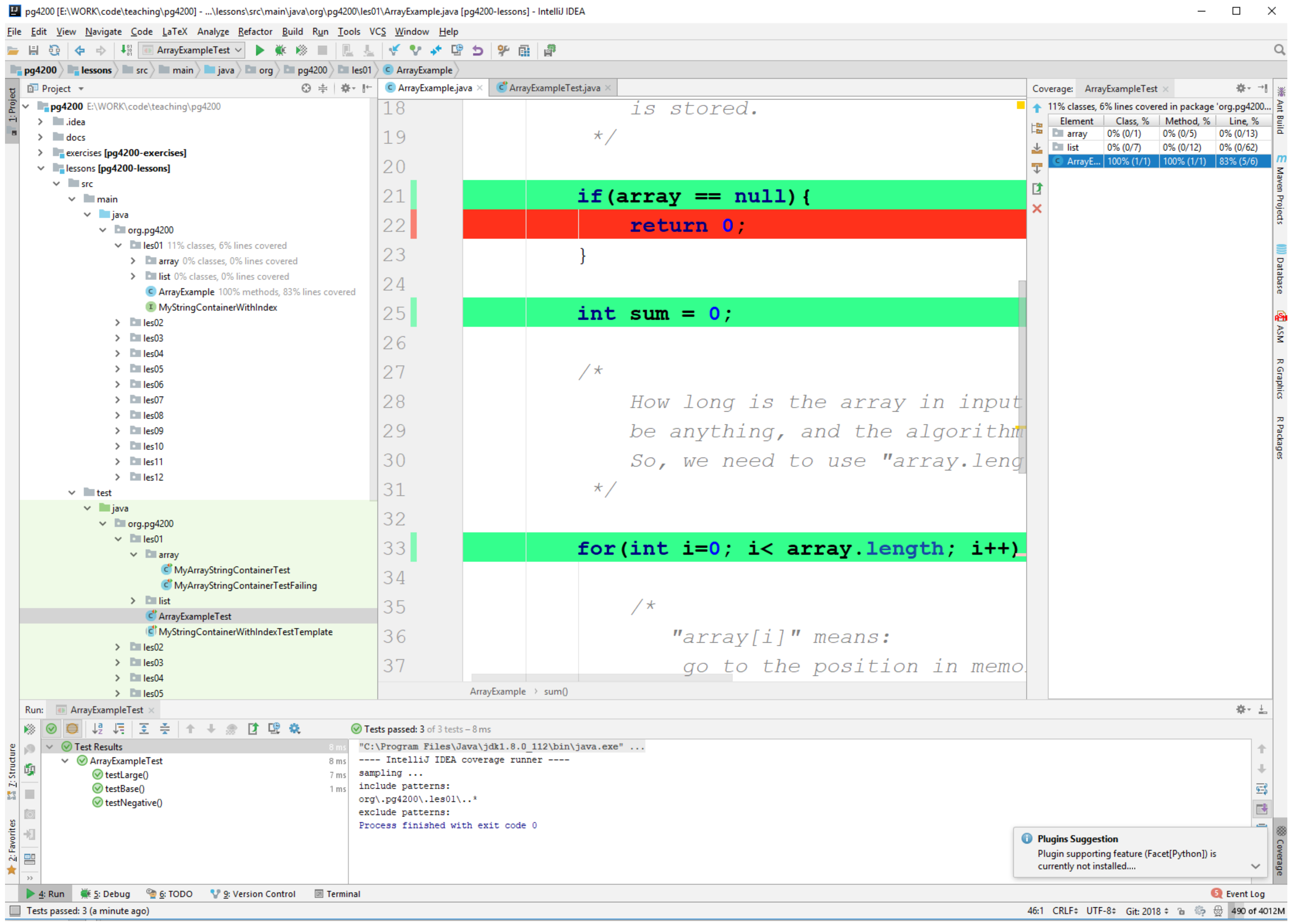
```
Test Results                10 ms
  ArrayExampleTest          10 ms
    testLarge()              9 ms
    testBase()               1 ms
    testNegative()
```

```
"C:\Program Files\Java\jdk1.8.0_112\bin\java.exe" ...

Process finished with exit code 0
```

Tests passed: 3 (moments ago)

Run With Coverage



pg4200 [E:\WORK\code\teaching\pg4200] - ...\lessons\src\main\java\org\pg4200\les01\ArrayExample.java [pg4200-lessons] - IntelliJ IDEA

```
pg4200 E:\WORK\code\teaching\pg4200
  .idea
  docs
  exercises [pg4200-exercises]
  lessons [pg4200-lessons]
    src
      main
        java
          org.pg4200
            les01 11% classes, 6% lines covered
              array 0% classes, 0% lines covered
              list 0% classes, 0% lines covered
              ArrayExample 100% methods, 83% lines covered
              MyStringContainerWithIndex
            les02
            les03
            les04
            les05
            les06
            les07
            les08
            les09
            les10
            les11
            les12
      test
        java
          org.pg4200
            les01
              array
                MyArrayStringContainerTest
                MyArrayStringContainerTestFailing
              list
              ArrayExampleTest
              MyStringContainerWithIndexTestTemplate
            les02
            les03
            les04
            les05
```

```java
            is stored.
         */

        if(array == null){
            return 0;
        }

        int sum = 0;

        /*
            How long is the array in input
            be anything, and the algorithm
            So, we need to use "array.leng
         */

        for(int i=0; i< array.length; i++)

            /*
                "array[i]" means:
                  go to the position in memo
```

ArrayExample › sum()

Coverage: ArrayExampleTest

11% classes, 6% lines covered in package 'org.pg4200...

Element	Class, %	Method, %	Line, %
array	0% (0/1)	0% (0/5)	0% (0/13)
list	0% (0/7)	0% (0/12)	0% (0/62)
ArrayE...	100% (1/1)	100% (1/1)	83% (5/6)

Run: ArrayExampleTest

Tests passed: 3 of 3 tests – 8 ms

```
Test Results            8 ms
  ArrayExampleTest      8 ms
    testLarge()         7 ms
    testBase()          1 ms
    testNegative()
```

```
"C:\Program Files\Java\jdk1.8.0_112\bin\java.exe" ...
---- IntelliJ IDEA coverage runner ----
sampling ...
include patterns:
org\.pg4200\.les01\..*
exclude patterns:
Process finished with exit code 0
```

Plugins Suggestion
Plugin supporting feature (Facet[Python]) is currently not installed....

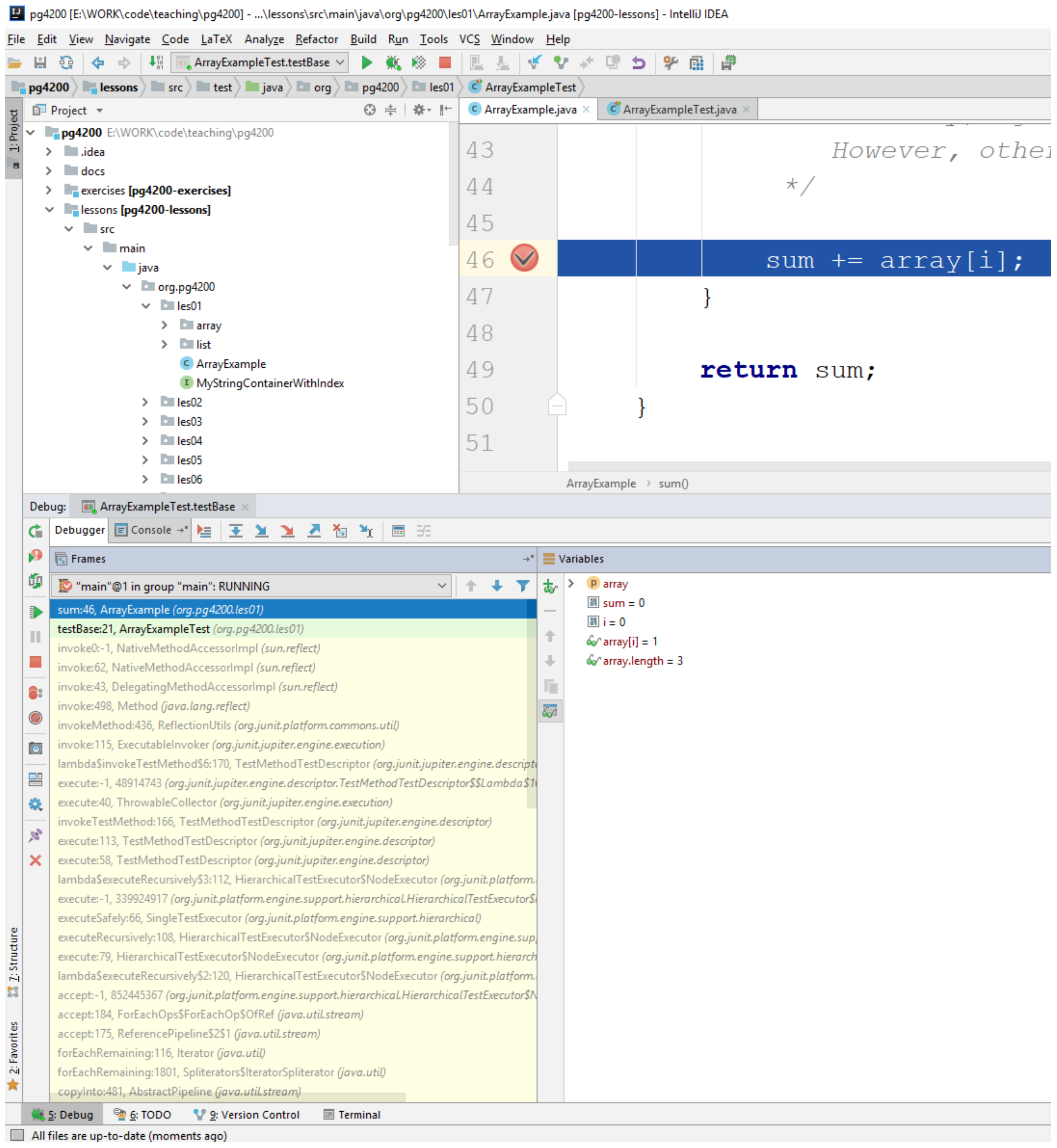
Tests passed: 3 (a minute ago)

46:1 CRLF÷ UTF-8÷ Git: 2018 ÷ 490 of 4012M

File Edit View Navigate Code LaTeX Analyze Refactor Build Run Tools VCS Window Help
ArrayExampleTest
pg4200 › lessons › src › main › java › org › pg4200 › les01 › ArrayExample
Project
ArrayExample.java
ArrayExampleTest.java
4: Run
5: Debug
6: TODO
9: Version Control
Terminal
Event Log

- Can tell you how much of the code is executed
- Eg, 83% in this case
- Code that is never executed by a test, might have bugs

Debugging



File Edit View Navigate Code LaTeX Analyze Refactor Build Run Tools VCS Window Help
ArrayExampleTest.testBase
pg4200 lessons src test java org pg4200 les01 ArrayExampleTest
Project
ArrayExample.java
ArrayExampleTest.java
Debug: ArrayExampleTest.testBase
Debugger
Console
Frames
Variables
5: Debug
6: TODO
9: Version Control
Terminal
1: Project
7: Structure
2: Favorites

pg4200 [E:\WORK\code\teaching\pg4200] - ...\lessons\src\main\java\org\pg4200\les01\ArrayExample.java [pg4200-lessons] - IntelliJ IDEA

```
pg4200 E:\WORK\code\teaching\pg4200
  .idea
  docs
  exercises [pg4200-exercises]
  lessons [pg4200-lessons]
    src
      main
        java
          org.pg4200
            les01
              array
              list
              ArrayExample
              MyStringContainerWithIndex
            les02
            les03
            les04
            les05
            les06
```

```java
                However, othe
            */

            sum += array[i];
        }

        return sum;
    }

```

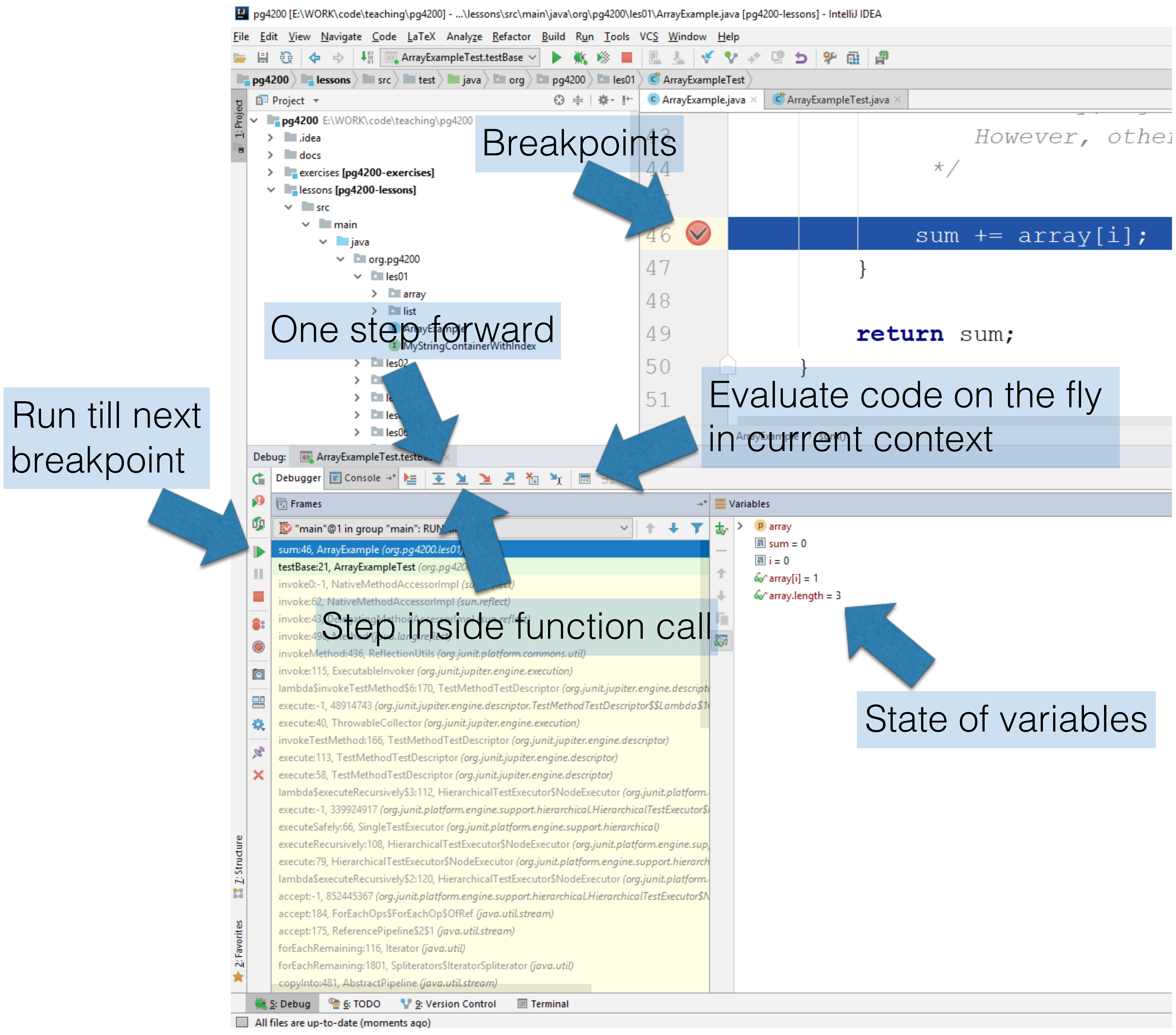
ArrayExample › sum()

```
"main"@1 in group "main": RUNNING
sum:46, ArrayExample (org.pg4200.les01)
testBase:21, ArrayExampleTest (org.pg4200.les01)
invoke0:-1, NativeMethodAccessorImpl (sun.reflect)
invoke:62, NativeMethodAccessorImpl (sun.reflect)
invoke:43, DelegatingMethodAccessorImpl (sun.reflect)
invoke:498, Method (java.lang.reflect)
invokeMethod:436, ReflectionUtils (org.junit.platform.commons.util)
invoke:115, ExecutableInvoker (org.junit.jupiter.engine.execution)
lambda$invokeTestMethod$6:170, TestMethodTestDescriptor (org.junit.jupiter.engine.descript
execute:-1, 48914743 (org.junit.jupiter.engine.descriptor.TestMethodTestDescriptor$$Lambda$1
execute:40, ThrowableCollector (org.junit.jupiter.engine.execution)
invokeTestMethod:166, TestMethodTestDescriptor (org.junit.jupiter.engine.descriptor)
execute:113, TestMethodTestDescriptor (org.junit.jupiter.engine.descriptor)
execute:58, TestMethodTestDescriptor (org.junit.jupiter.engine.descriptor)
lambda$executeRecursively$3:112, HierarchicalTestExecutor$NodeExecutor (org.junit.platform.
execute:-1, 339924917 (org.junit.platform.engine.support.hierarchical.HierarchicalTestExecutor$
executeSafely:66, SingleTestExecutor (org.junit.platform.engine.support.hierarchical)
executeRecursively:108, HierarchicalTestExecutor$NodeExecutor (org.junit.platform.engine.sup
execute:79, HierarchicalTestExecutor$NodeExecutor (org.junit.platform.engine.support.hierarch
lambda$executeRecursively$2:120, HierarchicalTestExecutor$NodeExecutor (org.junit.platform.
accept:-1, 852445367 (org.junit.platform.engine.support.hierarchical.HierarchicalTestExecutor$N
accept:184, ForEachOps$ForEachOp$OfRef (java.util.stream)
accept:175, ReferencePipeline$2$1 (java.util.stream)
forEachRemaining:116, Iterator (java.util)
forEachRemaining:1801, Spliterators$IteratorSpliterator (java.util)
copyInto:481, AbstractPipeline (java.util.stream)
```

```
> array
  sum = 0
  i = 0
  array[i] = 1
  array.length = 3
```

All files are up-to-date (moments ago)

- VERY IMPORTANT
- Can put “*break points*”
- Execute one step at a time
- Inspect status of all variables, at each step
- Easier to understand with live demo


Breakpoints
One step forward
Run till next breakpoint
Evaluate code on the fly in current context
Step inside function call
State of variables
pg4200 [E:\WORK\code\teaching\pg4200] - ...\lessons\src\main\java\org\pg4200\les01\ArrayExample.java [pg4200-lessons] - IntelliJ IDEA
File Edit View Navigate Code LaTeX Analyze Refactor Build Run Tools VCS Window Help
ArrayExampleTest.testBase
pg4200 lessons src test java org pg4200 les01 ArrayExampleTest
Project
ArrayExample.java
ArrayExampleTest.java
pg4200 E:\WORK\code\teaching\pg4200
.idea
docs
exercises [pg4200-exercises]
lessons [pg4200-lessons]
src
main
java
org.pg4200
les01
array
list
MyStringContainerWithIndex
les02
Debug: ArrayExampleTest.testBase
Debugger
Console
Frames
Variables
5: Debug
6: TODO
9: Version Control
Terminal
All files are up-to-date (moments ago)

```
         However, other
       */

            sum += array[i];
        }

        return sum;
    }
```

```
"main"@1 in group "main": RUN...
sum:46, ArrayExample (org.pg4200.les01)
testBase:21, ArrayExampleTest (org.pg420
invoke0:-1, NativeMethodAccessorImpl (sun.reflect)
invoke:62, NativeMethodAccessorImpl (sun.reflect)
invoke:43, DelegatingMethodAccessorImpl (sun.reflect)
invoke:498, Method (java.lang.reflect)
invokeMethod:436, ReflectionUtils (org.junit.platform.commons.util)
invoke:115, ExecutableInvoker (org.junit.jupiter.engine.execution)
lambda$invokeTestMethod$6:170, TestMethodTestDescriptor (org.junit.jupiter.engine.descript
execute:-1, 48914743 (org.junit.jupiter.engine.descriptor.TestMethodTestDescriptor$$Lambda$1
execute:40, ThrowableCollector (org.junit.jupiter.engine.execution)
invokeTestMethod:166, TestMethodTestDescriptor (org.junit.jupiter.engine.descriptor)
execute:113, TestMethodTestDescriptor (org.junit.jupiter.engine.descriptor)
execute:58, TestMethodTestDescriptor (org.junit.jupiter.engine.descriptor)
lambda$executeRecursively$3:112, HierarchicalTestExecutor$NodeExecutor (org.junit.platform.
execute:-1, 339924917 (org.junit.platform.engine.support.hierarchical.HierarchicalTestExecutor$
executeSafely:66, SingleTestExecutor (org.junit.platform.engine.support.hierarchical)
executeRecursively:108, HierarchicalTestExecutor$NodeExecutor (org.junit.platform.engine.sup
execute:79, HierarchicalTestExecutor$NodeExecutor (org.junit.platform.engine.support.hierarch
lambda$executeRecursively$2:120, HierarchicalTestExecutor$NodeExecutor (org.junit.platform.
accept:-1, 852445367 (org.junit.platform.engine.support.hierarchical.HierarchicalTestExecutor$N
accept:184, ForEachOps$ForEachOp$OfRef (java.util.stream)
accept:175, ReferencePipeline$2$1 (java.util.stream)
forEachRemaining:116, Iterator (java.util)
forEachRemaining:1801, Spliterators$IteratorSpliterator (java.util)
copyInto:481, AbstractPipeline (java.util.stream)
```

```
> array
  sum = 0
  i = 0
  array[i] = 1
  array.length = 3
```

Homework

- Study Book Chapter 1.1 and 1.2
- Study code in the *org.pg4200.les01* package
- Do exercises in *exercises/ex01*
- Extra: do exercises in the book