Ca' Foscari University of Venice

Save the date! 16/11/2024 Google DevFest @ Campus

- Organized already last year and the year before
  - https://gdg.community.dev/events/details/google-gdg-venezia-presents-devfest-triveneto-2022/
  - https://www.devfest-triveneto.it/
- Part of the activities of this course
- 15 enterprises will be at the event
  - Open for interviews for stages, jobs, etc etc…
- More details will follow in about a month

Recap

- Everyday as users we rely on some technologies we do not know at all
- The interface of the technology is clear
- The details of the technology are hidden
  - And we should not care at all about them!
- The same must happen for software
  - We rely on code written by others (libraries) using some interface
- … or do you want to build up an app starting from the operating system???

Ca' Foscari University of Venice

Java

Object oriented programming, module 1

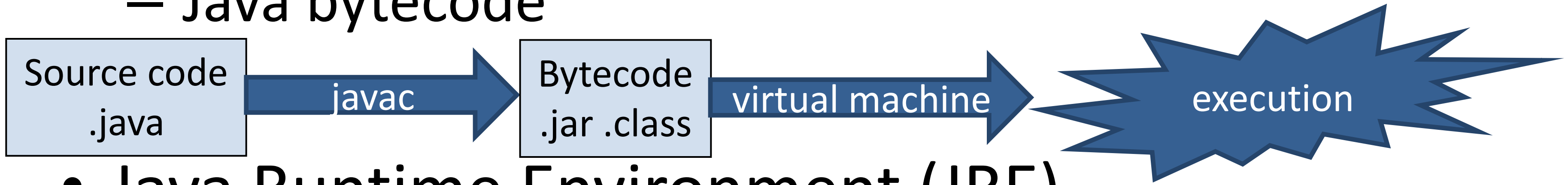
Pietro Ferrara

pietro.ferrara@unive.it

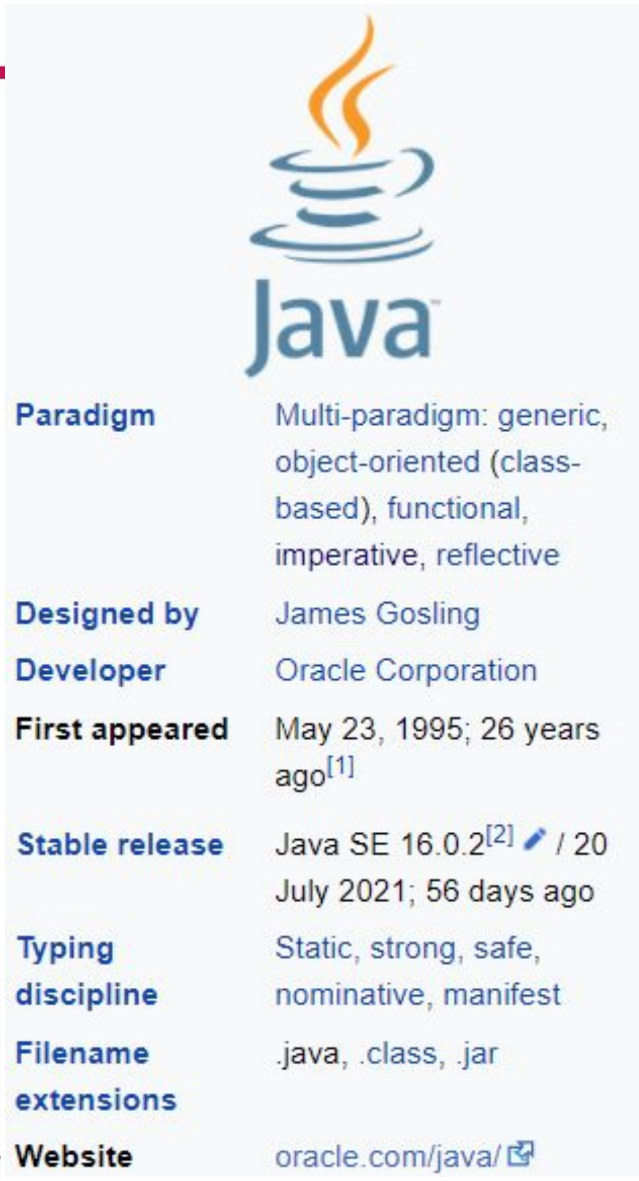
Ca' Foscari University of Venice

Java?

- Object oriented!
- Huge community
  - And even more libraries
- Portable:
  - Java bytecode



- Java Runtime Environment (JRE)
  - Virtual machine
- Java Development Kit (JDK)
  - JRE+compiler+…



Paradigm	Multi-paradigm: generic, object-oriented (class-based), functional, imperative, reflective
Designed by	James Gosling
Developer	Oracle Corporation
First appeared	May 23, 1995; 26 years ago[1]
Stable release	Java SE 16.0.2[2] / 20 July 2021; 56 days ago
Typing discipline	Static, strong, safe, nominative, manifest
Filename extensions	.java, .class, .jar
Website	oracle.com/java/

Wikipedia
The Free Encyclopedia

Ca' Foscari University of Venice

Java bytecode

- Java bytecode is:
  - a machine-independent low-level language
  - object-oriented
  - garbage-collection-based
- Its execution state is composed by:
  - a stack of frames (one per method call) containing:
    - a pool of local variables holding values
    - an operand stack of values
  - a memory holding objects
- We can see it with various tools
  - https://github.com/ingokegel/jclasslib/releases
  - https://plugins.jetbrains.com/plugin/9248-jclasslib-bytecode-viewer

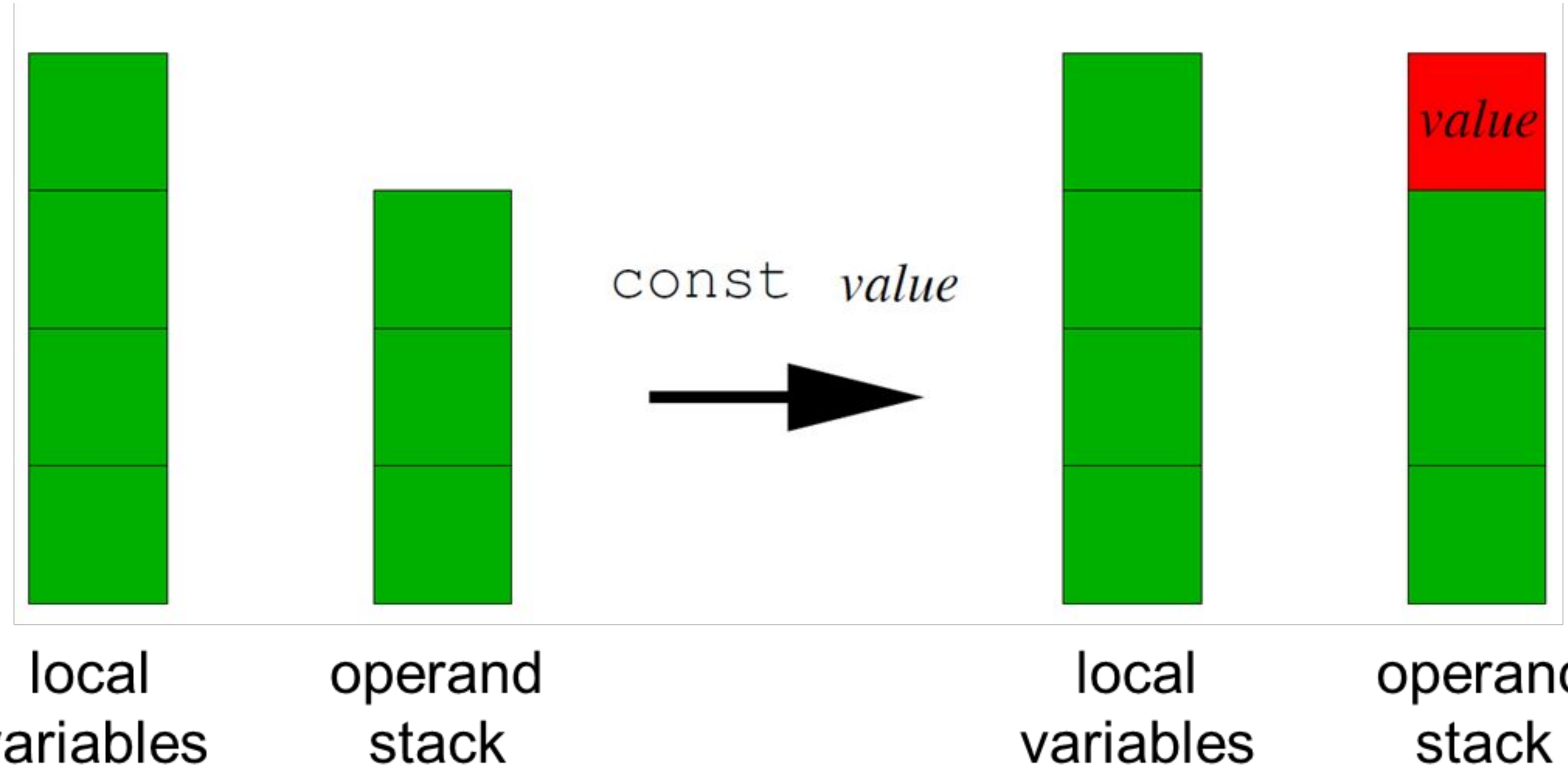
Ca' Foscari University of Venice

Specification

- https://docs.oracle.com/javase/specs/jvms/se7/html/jvms-6.html
  - Hundreds of statements
  - Language mostly stable
    - Only one statement (invokedynamic) added in ~30 years
- We can divide it into few main categories:
  - load or store local variables
  - read or write heap locations
  - invoke methods
  - perform arithmetic operations
  - check conditions on values
- Some examples follow

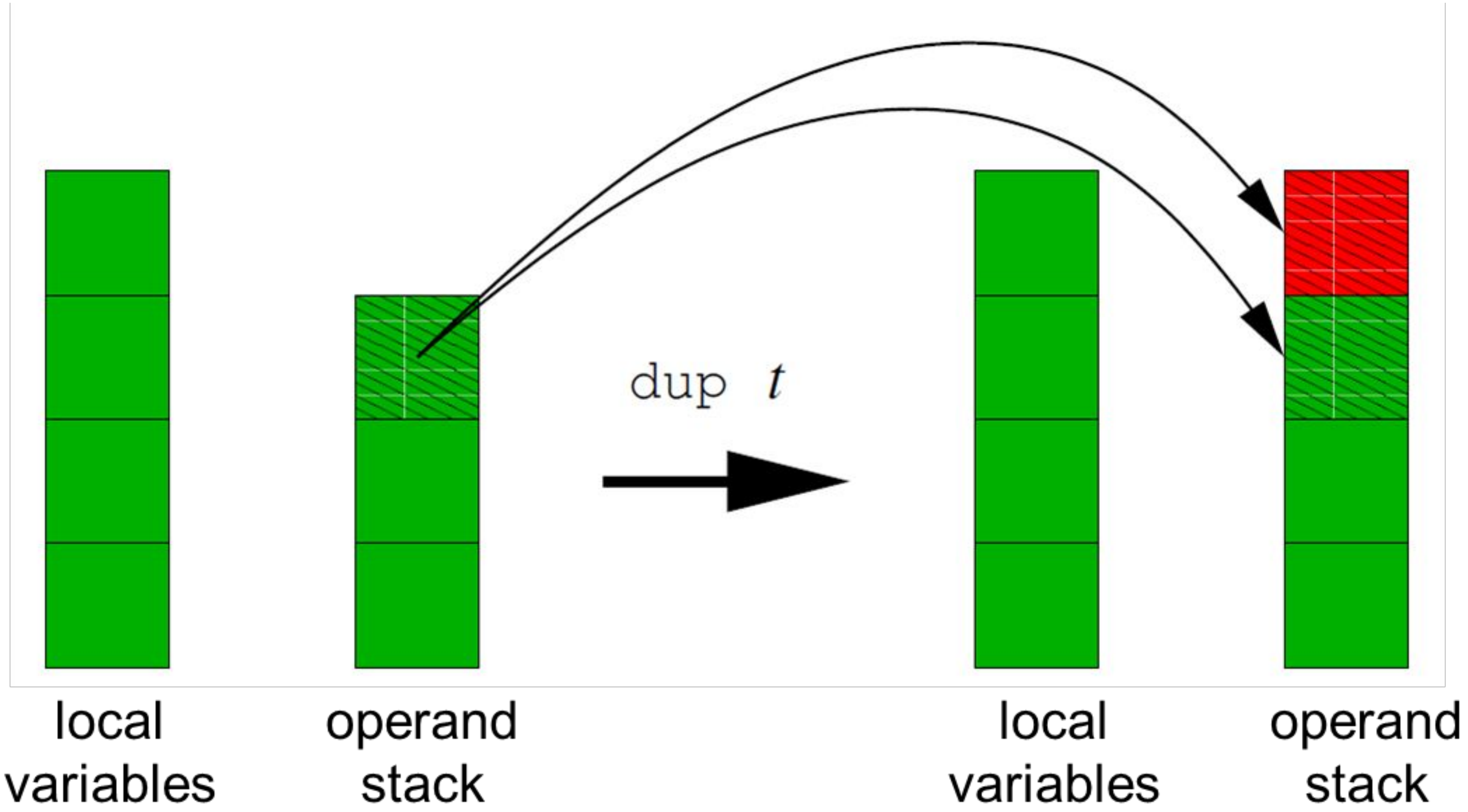
Ca' Foscari University of Venice

const



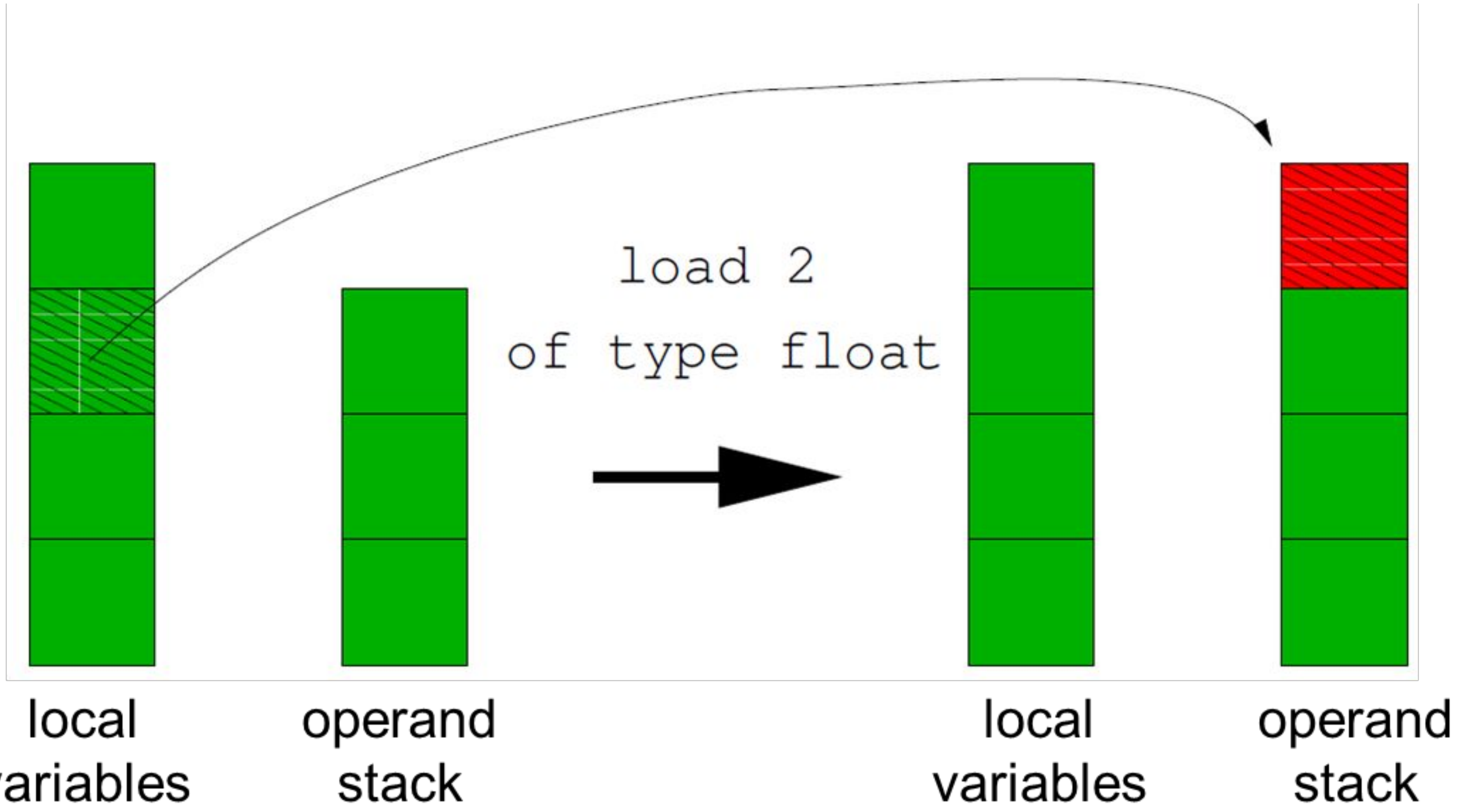
Ca' Foscari University of Venice

dup



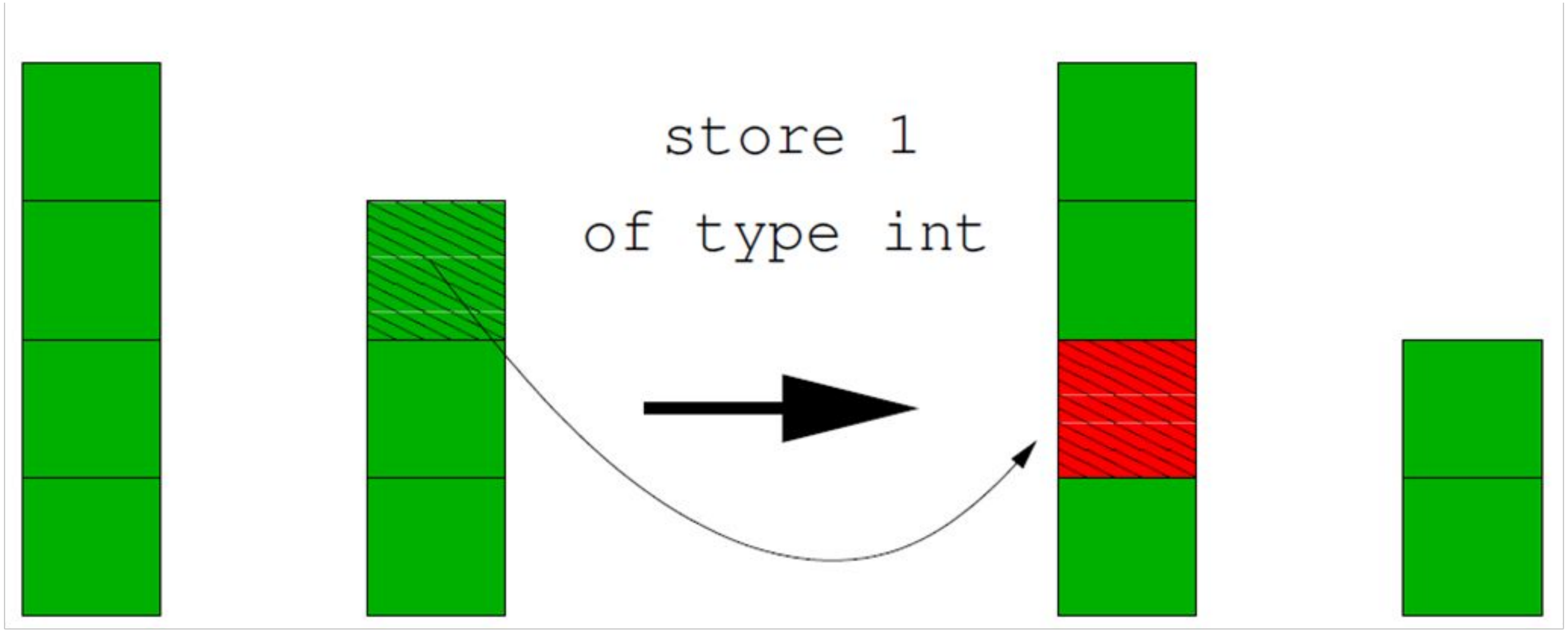
Ca' Foscari University of Venice

load



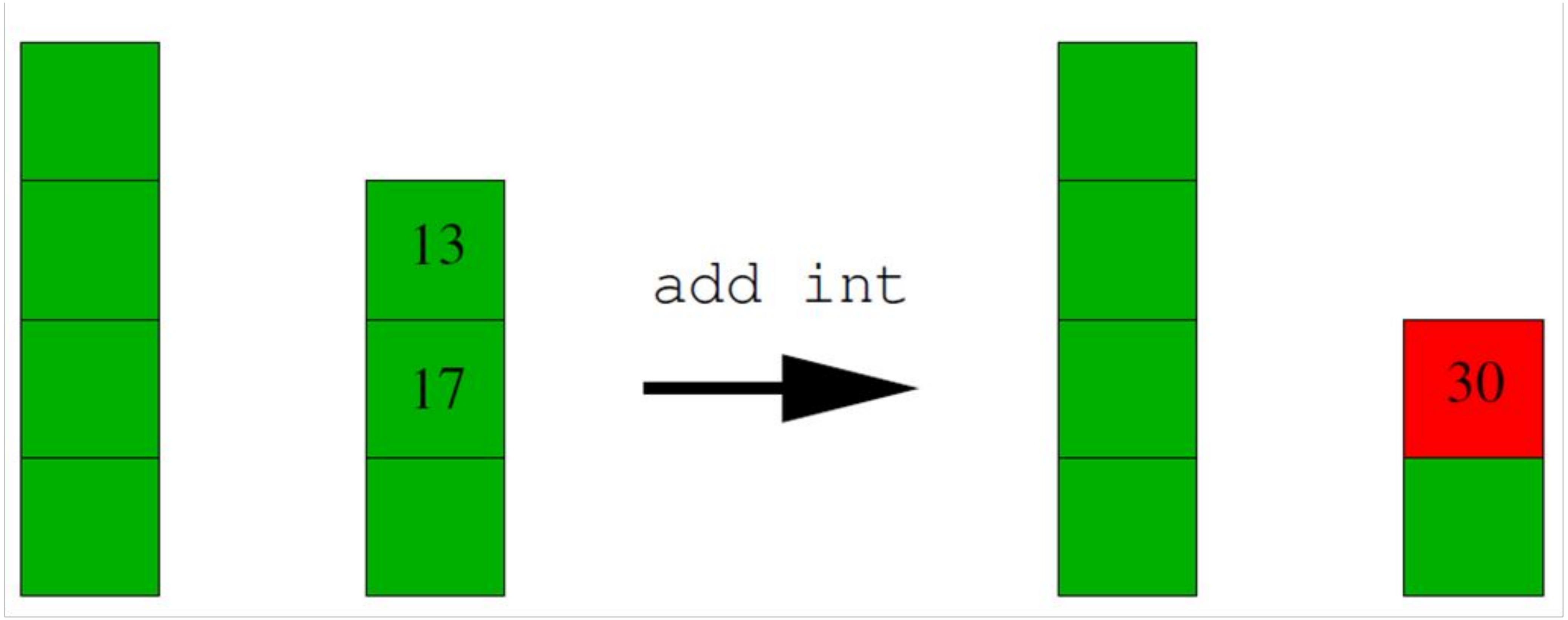
Ca' Foscari University of Venice

store



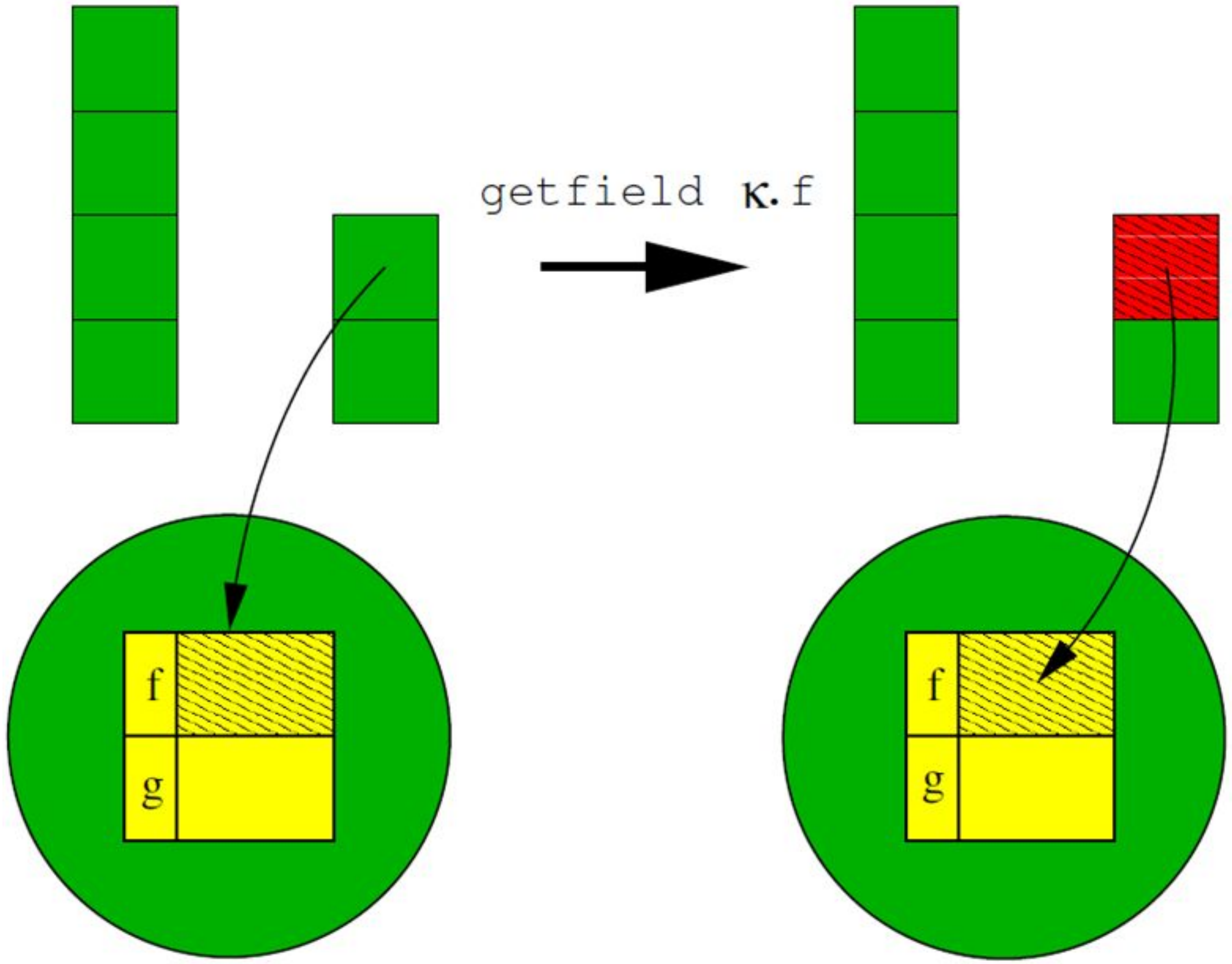
local variables operand stack local variables operand stack

add



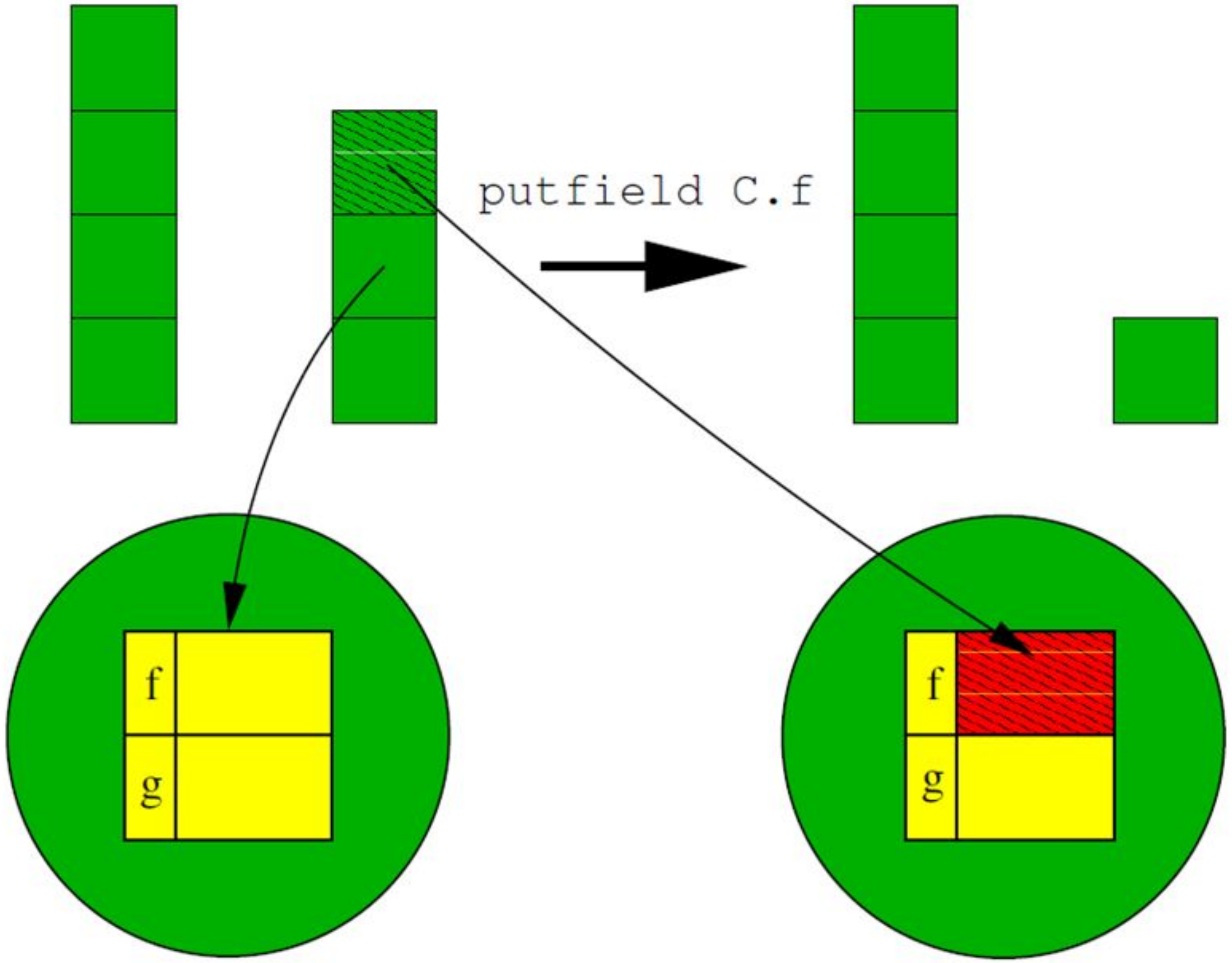
local variables operand stack local variables operand stack

getfield



Ca' Foscari University of Venice

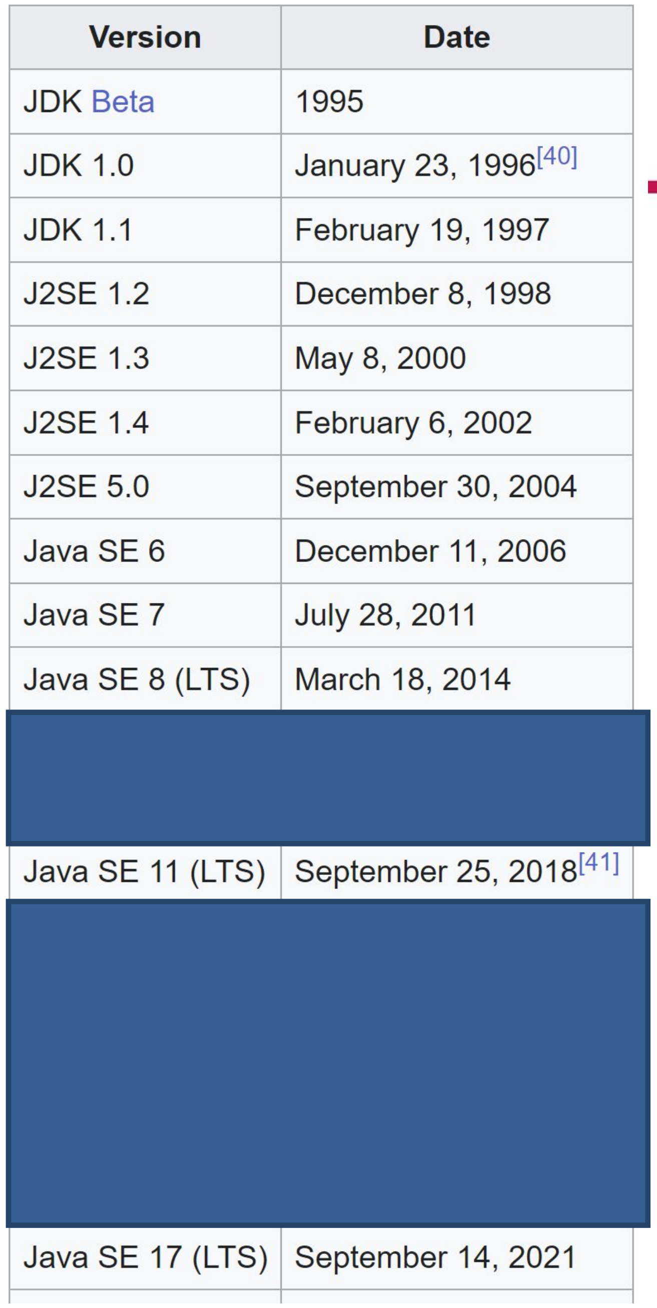
putfield



Ca' Foscari University of Venice

Java history

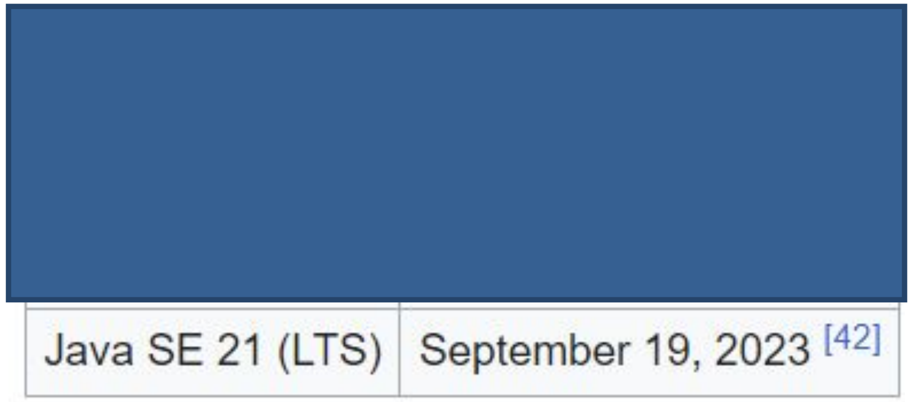
- Originally developed by James Gosling at Sun
- January 2010: Oracle acquires Sun
- Starting from Java 9, 1 out of 6 versions is LTS
  - LTS = long term support
  - Java 9, 10, 12, 13, 14, 15 no longer supported!
  - Java 11, 17, 23, … are LTS
- Oracle is one (out of many) provider of JDK and JRE
  - Current owner of the official implementation
  - OpenJDK is another implementation



Version	Date
JDK Beta	1995
JDK 1.0	January 23, 1996[40]
JDK 1.1	February 19, 1997
J2SE 1.2	December 8, 1998
J2SE 1.3	May 8, 2000
J2SE 1.4	February 6, 2002
J2SE 5.0	September 30, 2004
Java SE 6	December 11, 2006
Java SE 7	July 28, 2011
Java SE 8 (LTS)	March 18, 2014
[redacted]	[redacted]
Java SE 11 (LTS)	September 25, 2018[41]
[redacted]	[redacted]
Java SE 17 (LTS)	September 14, 2021

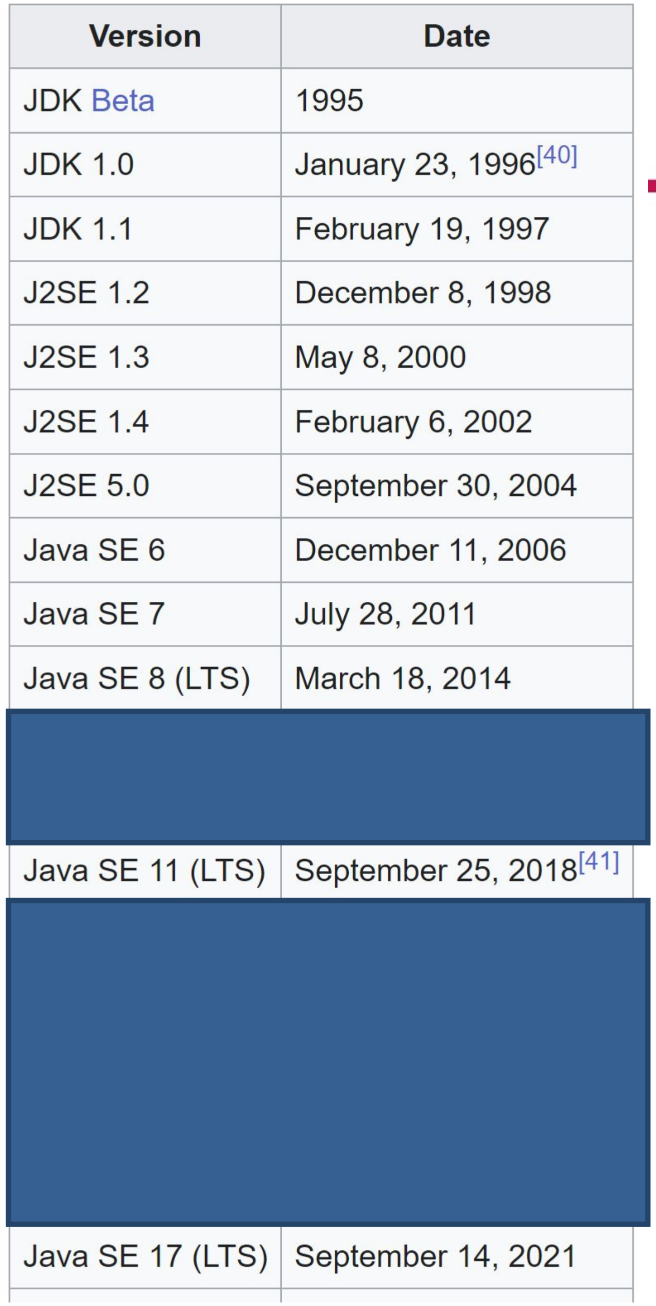
Java history

- Starting from Java 9, 1 out of 6 versions is LTS
  - LTS = long term support
  - Java 9, 10, 12, 13, 14, 15 no longer supported!
  - Java 11, 17, 23, … are LTS
- Ah, no, we were joking!
  - Java 21 is a LTS
  - Released 3 days ago!



[redacted]

Java SE 21 (LTS)	September 19, 2023 [42]



Version	Date
JDK Beta	1995
JDK 1.0	January 23, 1996[40]
JDK 1.1	February 19, 1997
J2SE 1.2	December 8, 1998
J2SE 1.3	May 8, 2000
J2SE 1.4	February 6, 2002
J2SE 5.0	September 30, 2004
Java SE 6	December 11, 2006
Java SE 7	July 28, 2011
Java SE 8 (LTS)	March 18, 2014
[redacted]	[redacted]
Java SE 11 (LTS)	September 25, 2018[41]
[redacted]	[redacted]
Java SE 17 (LTS)	September 14, 2021

Ca' Foscari University of Venice

Java versions

Versions	Date	Features
1.0	1996	
1.1	1997	Inner classes, partial reflection
1.2	1998	Collection, full reflection
1.3	2000	Remote Method Invocation
1.4	2002	Assert keyword, first XML support
5	2004	**Generics**, varargs, enumerations, multithreaded Java memory model
6	2006	Performance improvements, second XML support
7	2011	Improved IO libraries
8	2014	**Lambda expressions**
11	2018	Modules
16	2021	Records
21	2023	Pattern matching for switch

- Full details at https://en.wikipedia.org/wiki/Java_version_history

Ca' Foscari University of Venice

Java vs. C

Java	C
• Imperative	• Imperative
• Object-oriented	• Procedural
• Interpreted	• Compiled
• High-level	• Low-level

The basic blocks (e.g., assignments, if-then-else, while loops, etc…) are the same (imperative). Java provides different (and I would say more expressive) primitives to structure your code. Java does not allow to freely access the memory through arbitrary pointers. C is more efficient, and part of the Java libraries are written in C (!), or in another "low-level" languages (native code).

Ca' Foscari University of Venice

And your first Java program is…

- Hello World!

- Definitely not the shortest Hello World program!
  - https://towardsdatascience.com/how-to-print-hello-world-in-top-12-most-popular-programming-languages-736d49c6c61c
- Java requires to structure your code in a quite fixed way
  - Class, method, statement (and much more!)

Ca' Foscari University of Venice

Why YAPL? 1/2

- YAPL = Yet Another Programming Language!
- C: limited extension and adaptation -> limited **code reuse**

```
typedef struct {
  int edgeLength;
} Square;

typedef struct {
  int edge1Length;
  int edge2Length;
} Rectangle;
```

```
int getAreaOfSquare(Square* s) {
  return s -> edgeLength * s -> edgeLength;
}

int getAreaOfRectangle(Rectangle * r) {
  return r -> edge1Length * r -> edge2Length;
}
```

```
typedef struct {
  int edgeLength;
  int height;
} Rhombus;
```

```
int getAreaOfRhombus(Rhombus* r) {
  return s -> edgeLength * s -> height;
}
```

Ca' Foscari University of Venice

Why YAPL? 2/2

- What about a Quadrilateral to represent all of them?

```
typedef struct {
  enum { S, Re, Rh} kind;
  union  {
    Square* s;
    Rectangle* r;
    Rhombus* r;
  } u;
} Quadrilateral;
```

```
int getAreaOfQuadrilateral(Quadrilateral* s) {
  switch(s -> kind) {
    case Square : return getAreaofSquare( s -> u.s);
    case Rectangle: …;
    case Rhombus: …;
  }
}
```

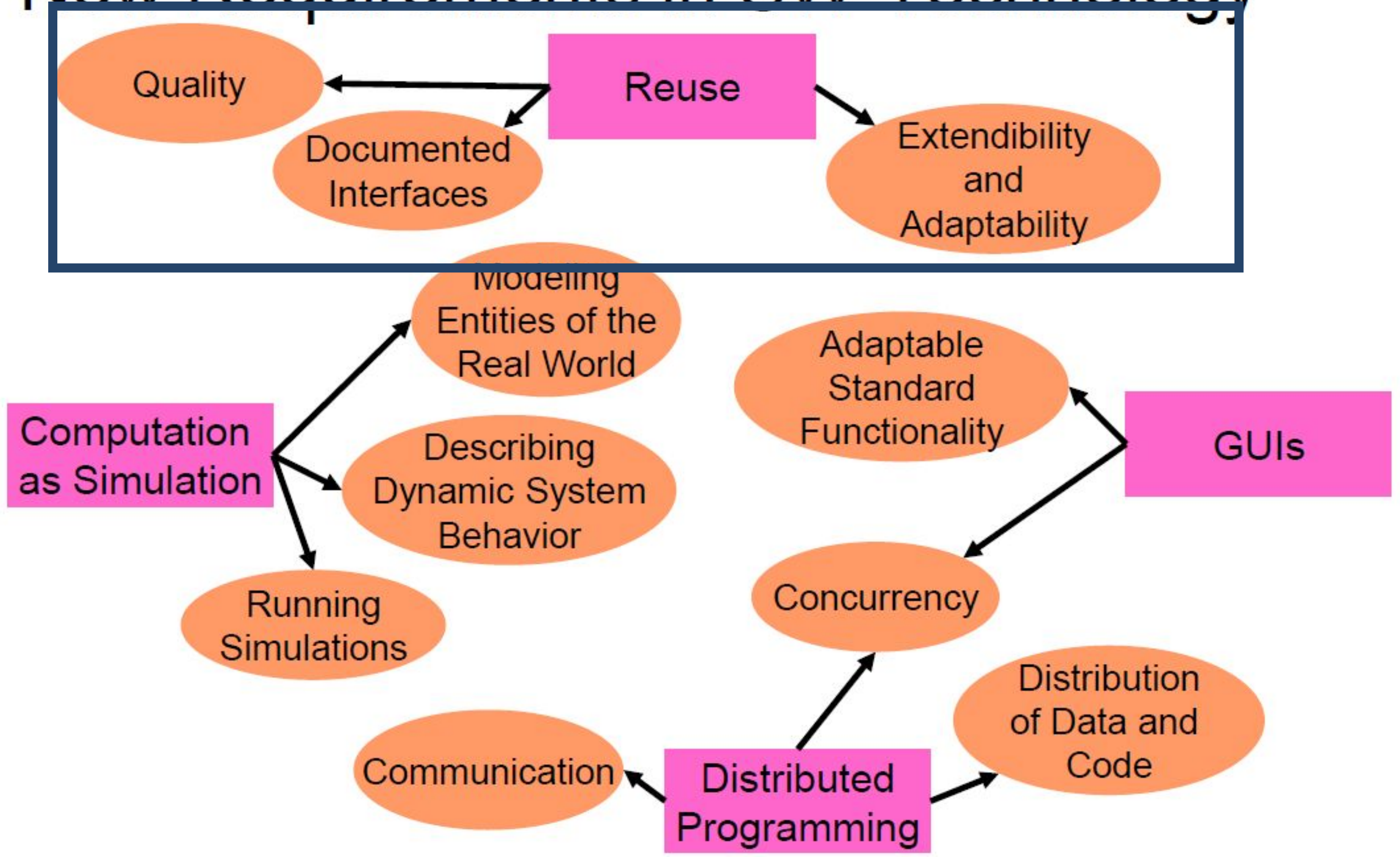
- Add Trapezoid?
  1. Create the Trapezoid struct and the method to compute the area
  2. Add Trepezoid to enum and union of Quadrilateral
  3. Consider case Trapezoid in getAreaOfQuadrilateral

Ca' Foscari University of Venice

Limit of imperative languages

- No support for extension and adaptation
  - Where adaptation often requires to modify existing code
- One extension requires to touch different parts of the code
  - Really hard to maintain in the long term
- A lot of duplicated code
  - Bad practice
  - Difficult to maintain all the copies
  - A modification in a piece of duplicated code is not propagated to all the copies of the code!

New Requirements in SW-Technology


Quality
Reuse
Documented Interfaces
Extendibility and Adaptability
Modeling Entities of the Real World
Adaptable Standard Functionality
Computation as Simulation
Describing Dynamic System Behavior
GUIs
Running Simulations
Concurrency
Distribution of Data and Code
Communication
Distributed Programming

https://www.pm.inf.ethz.ch/education/courses/COOP.html

Ca' Foscari University of Venice

Goal of OO programming languages

- Improve code reuse
  - Allow a clean code structure through encapsulation
    - Hide information of software units that should not be visible from outside
  - Allow to extend and specialize existing code through inheritance
  - Allow to develop reusable algorithms through classification, polymorphism and dynamic method binding
- Main outcome: a programming language that allows to modularly reason on software capsules
  - Advantages: a lot of well documented and easy to use libraries
  - Weaknesses: efficiency and conciseness

Ca' Foscari University of Venice

Textbook

- Lecture notes: Appendix A

- Arnold&others:
  - Hello World:
    - Section 1.1 (but please DO NOT look to the rest of the first chapter!)
    - Section 2.10 about main method