CSUN
CALIFORNIA
STATE UNIVERSITY
NORTHRIDGE

COMP110

Java

DR JEFF
SOFTWARE
INDIE APP DEVELOPER

Rev. 1-19-22

Intro to Algorithms & Programming

LECTURES

Part 1

Dr Jeff Drobman

Index

KEY

DATA
STRUCTURE
LOGIC
OTHER

Software

Models

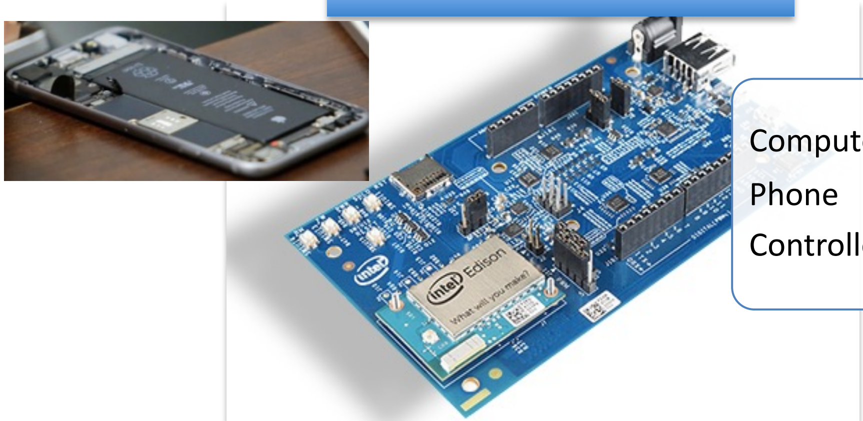
CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE

COMP110

Digital Systems

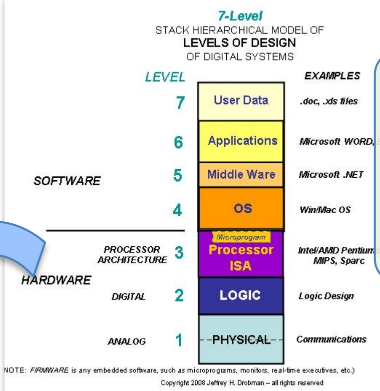
DR JEFF SOFTWARE
INDIE APP DEVELOPER
©2016-2022
Jeff Drobman

Hardware



Computer
Phone
Controller

Software



7-Level
STACK HIERARCHICAL MODEL OF
LEVELS OF DESIGN
OF DIGITAL SYSTEMS

		LEVEL		EXAMPLES
SOFTWARE		7	User Data	.doc, .xls files
		6	Applications	Microsoft WORD, Excel
		5	Middle Ware	Microsoft .NET
		4	OS	Win/Mac OS
HARDWARE	PROCESSOR ARCHITECTURE	3	Microprogram Processor ISA	Intel/AMD Pentiums MIPS, Sparc
	DIGITAL	2	LOGIC	Logic Design
	ANALOG	1	PHYSICAL	Communications

NOTE: *FIRMWARE* is any embedded software, such as microprograms, monitors, real-time executives, etc.)

Copyright 2008 Jeffrey H. Drobman – all rights reserved

Layers
Levels
Composition

- CODE
- DATA

Computer **Engineering**

- **Architecture**
- Physical design

External Storage

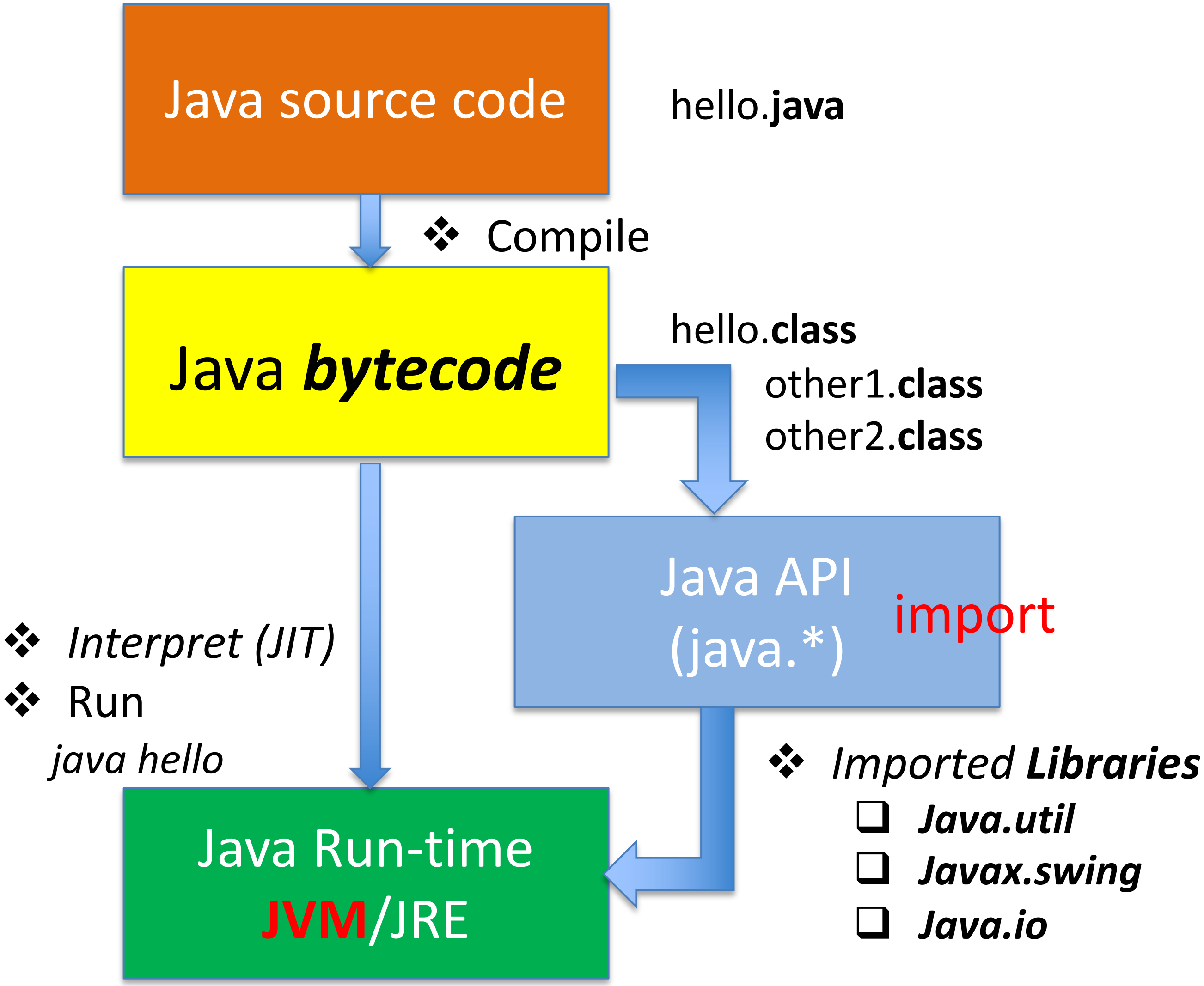
Ultrastar® He
Helium Hard Disk Drive
6TB
7Stac™ & HelioSeal™ Featuring technology
HGST
A Western Digital company

Hard disk
Flash disk
Flash
EEPROM

Computer **Science**

- **Algorithms**
- Theory

Running Java



Java source code

hello.**java**

- ❖ Compile

Java ***bytecode***

hello.**class**
other1.**class**
other2.**class**

Java API
(java.*)

import

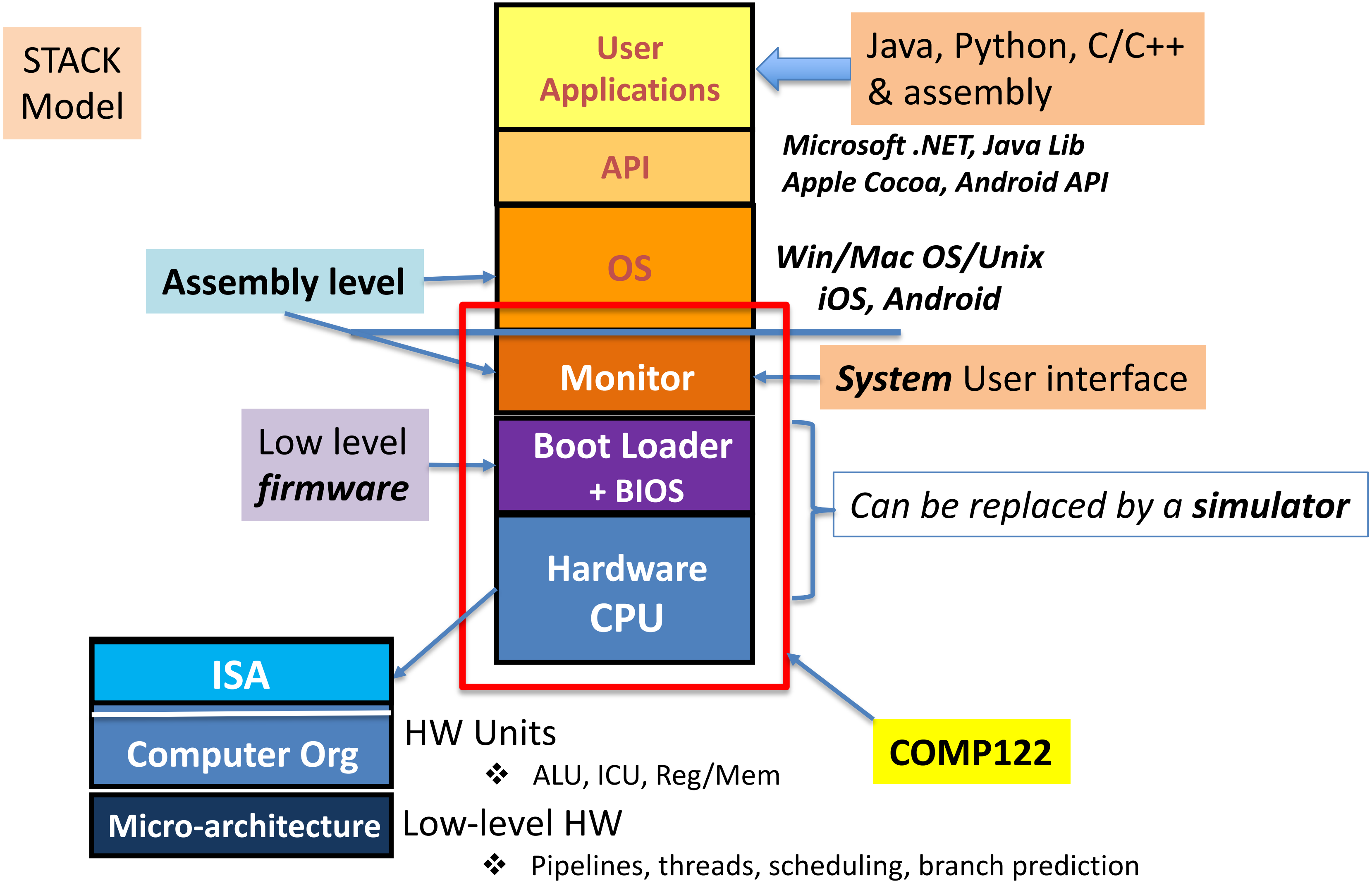
- ❖ *Interpret (JIT)*
- ❖ Run

java hello

Java Run-time
JVM/JRE

- ❖ *Imported* ***Libraries***
  - ❑ ***Java.util***
  - ❑ ***Javax.swing***
  - ❑ ***Java.io***

Hardware/Software *Low Level*



HW Units

- ALU, ICU, Reg/Mem

Low-level HW

- Pipelines, threads, scheduling, branch prediction

Rev Aug 2021

Baseline Instruction Set

Computation

- ALU
  - ADD
  - SUB
  - AND
  - OR
  - XOR
  - NOT
- MULT/DIV [opt]
- BIT
  - SET/CLR
  - TEST
- COMPARE
  - CMP
- SHIFT
  - SHIFT (A, L)
  - ROTATE

Memory

- Reg-Reg
  - MOV
- Reg-Mem
  - LOAD
  - STORE
  - MOV

RISC

- Mem-Mem
  - MOV

CISC

- Stack
  - PUSH
  - POP

Program Control

- JUMP
  - JUMP/GOTO
- BRANCH
  - BRA
  - BRCC
  - LOOP
- CALL
  - CALL/CALR/JAL
  - RET/RETFIE
- NOP

I/O

- I/O
  - IN
  - OUT

OLD

- Mem Mapped
  - MOV PORT
  - LOAD/STORE

NEW

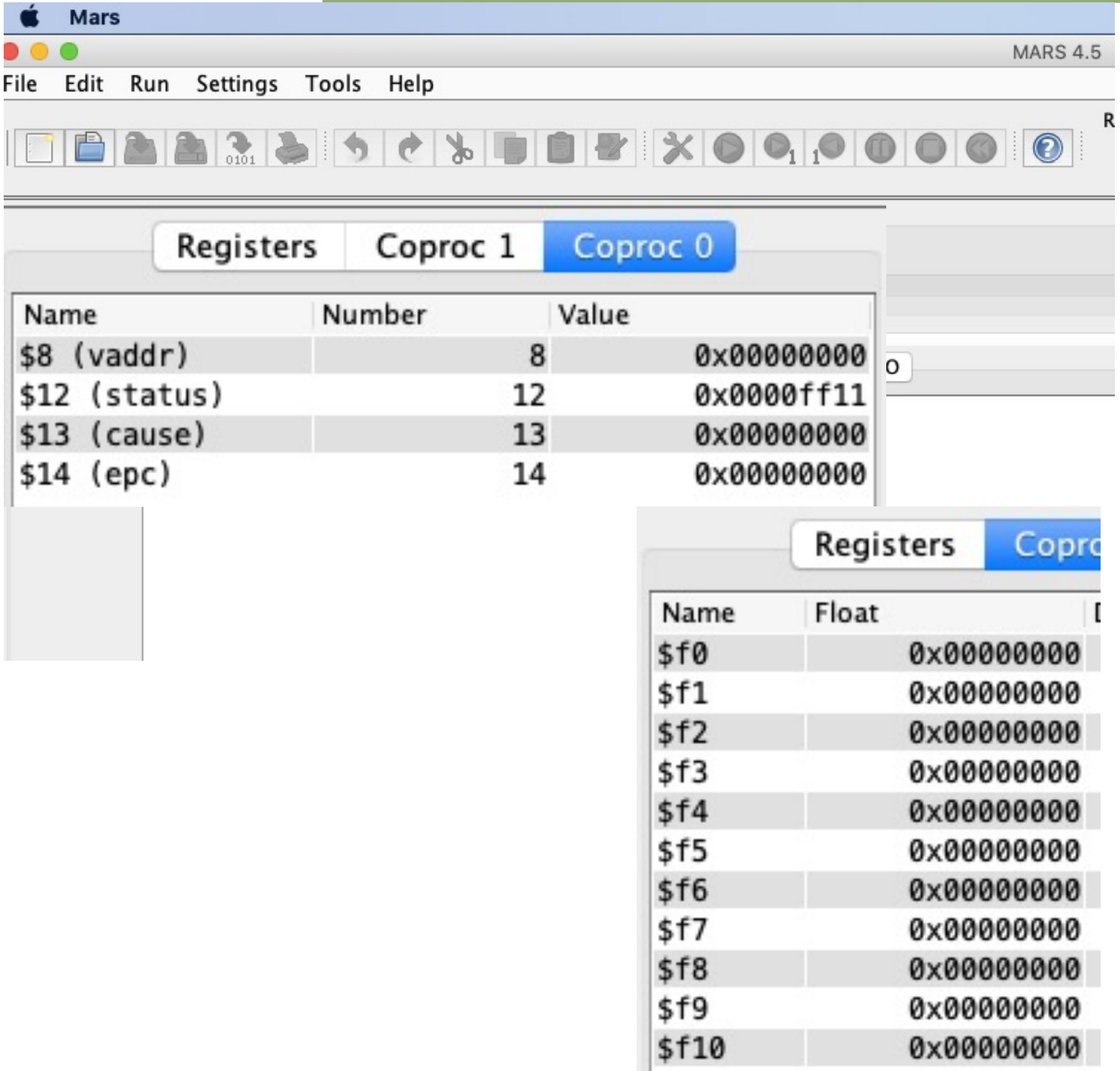
System Control

- Reset
  - RESET
- Power
  - SLEEP/HALT

MARS

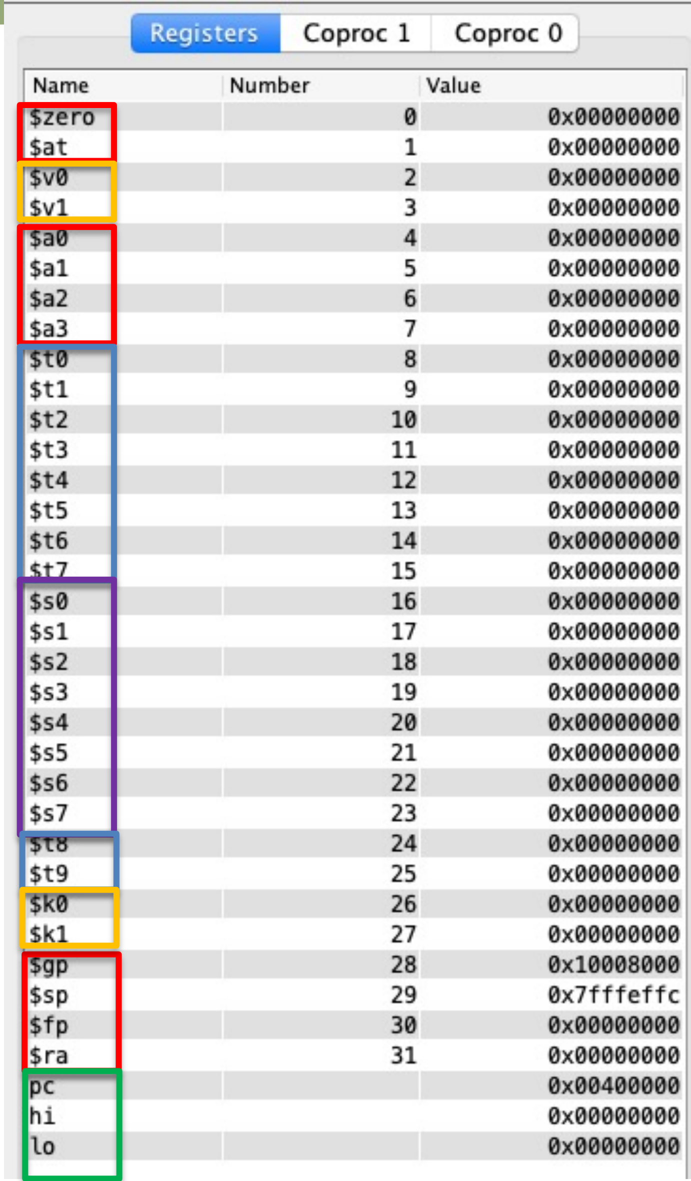
MARS (MIPS Assembler and Runtime Simulator)

Registers



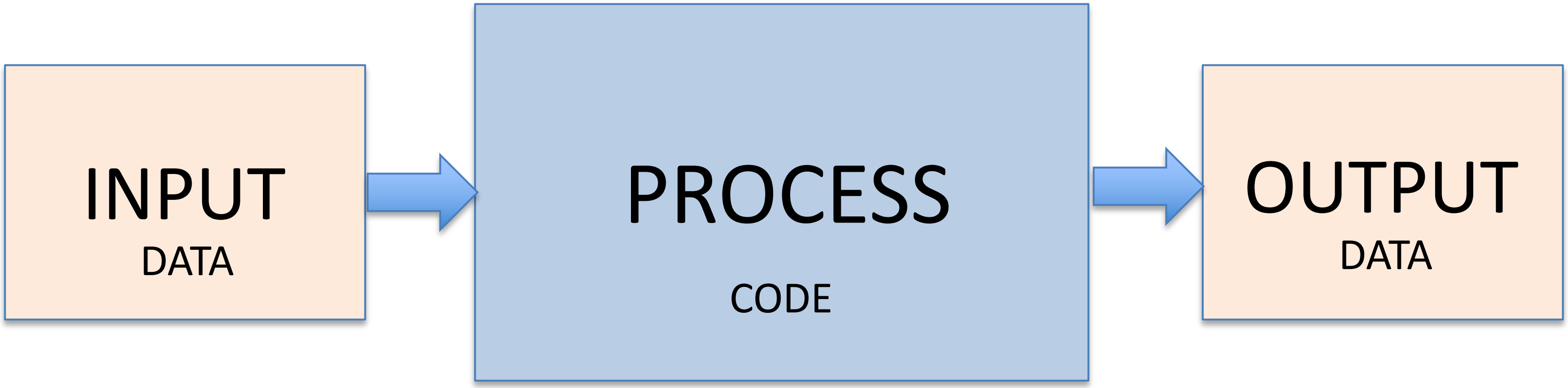
Name	Number	Value
$8 (vaddr)	8	0x00000000
$12 (status)	12	0x0000ff11
$13 (cause)	13	0x00000000
$14 (epc)	14	0x00000000

Name	Float
$f0	0x00000000
$f1	0x00000000
$f2	0x00000000
$f3	0x00000000
$f4	0x00000000
$f5	0x00000000
$f6	0x00000000
$f7	0x00000000
$f8	0x00000000
$f9	0x00000000
$f10	0x00000000



Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x00000000
$s1	17	0x00000000
$s2	18	0x00000000
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00400000
hi		0x00000000
lo		0x00000000

IPO Model



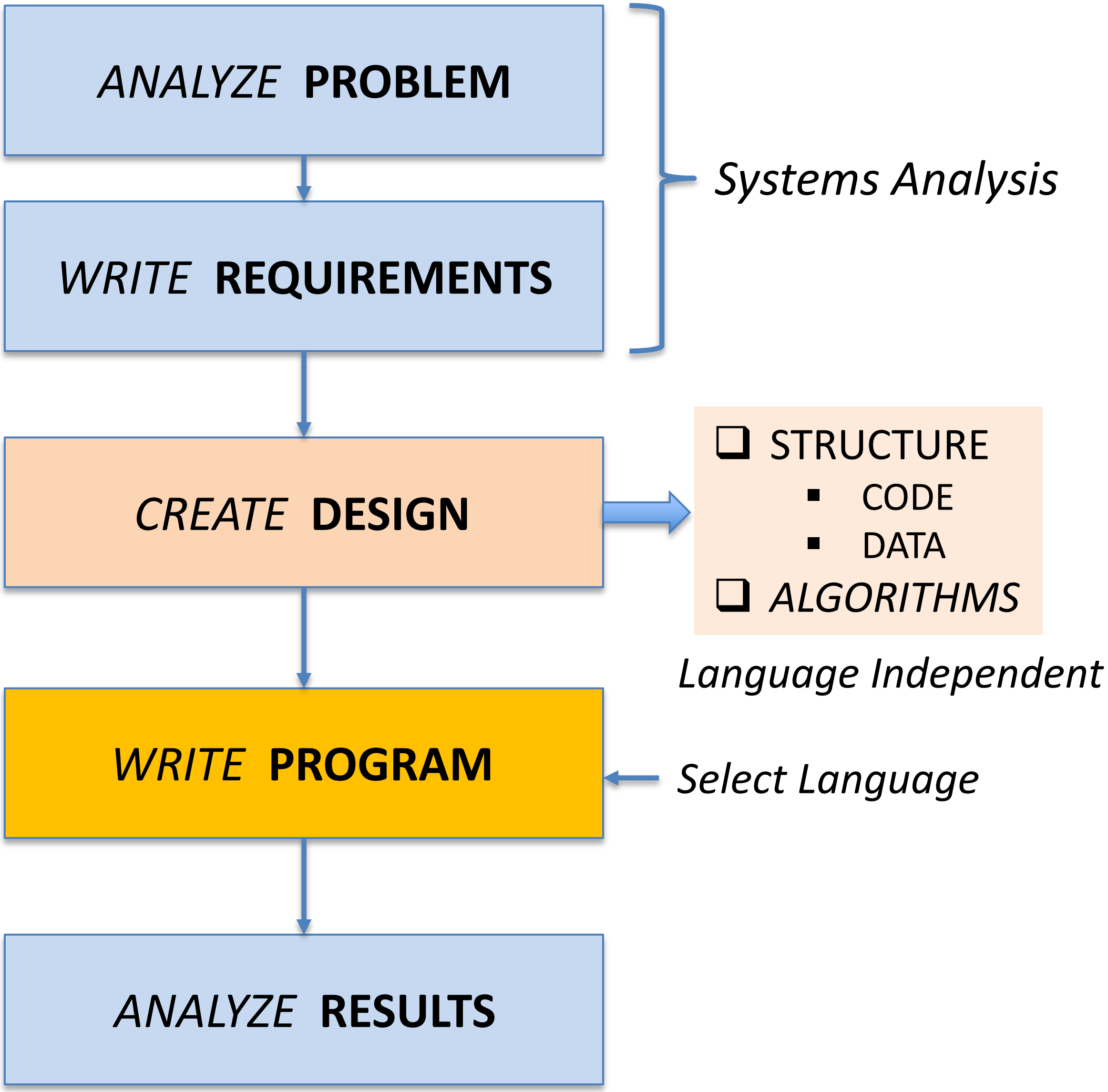
INPUT
DATA

PROCESS
CODE

OUTPUT
DATA

Software Development

Development Procedure
(similar to **SDLC**)



ANALYZE **PROBLEM**

WRITE **REQUIREMENTS**

Systems Analysis

CREATE **DESIGN**

□ STRUCTURE
- CODE
- DATA

□ *ALGORITHMS*

Language Independent

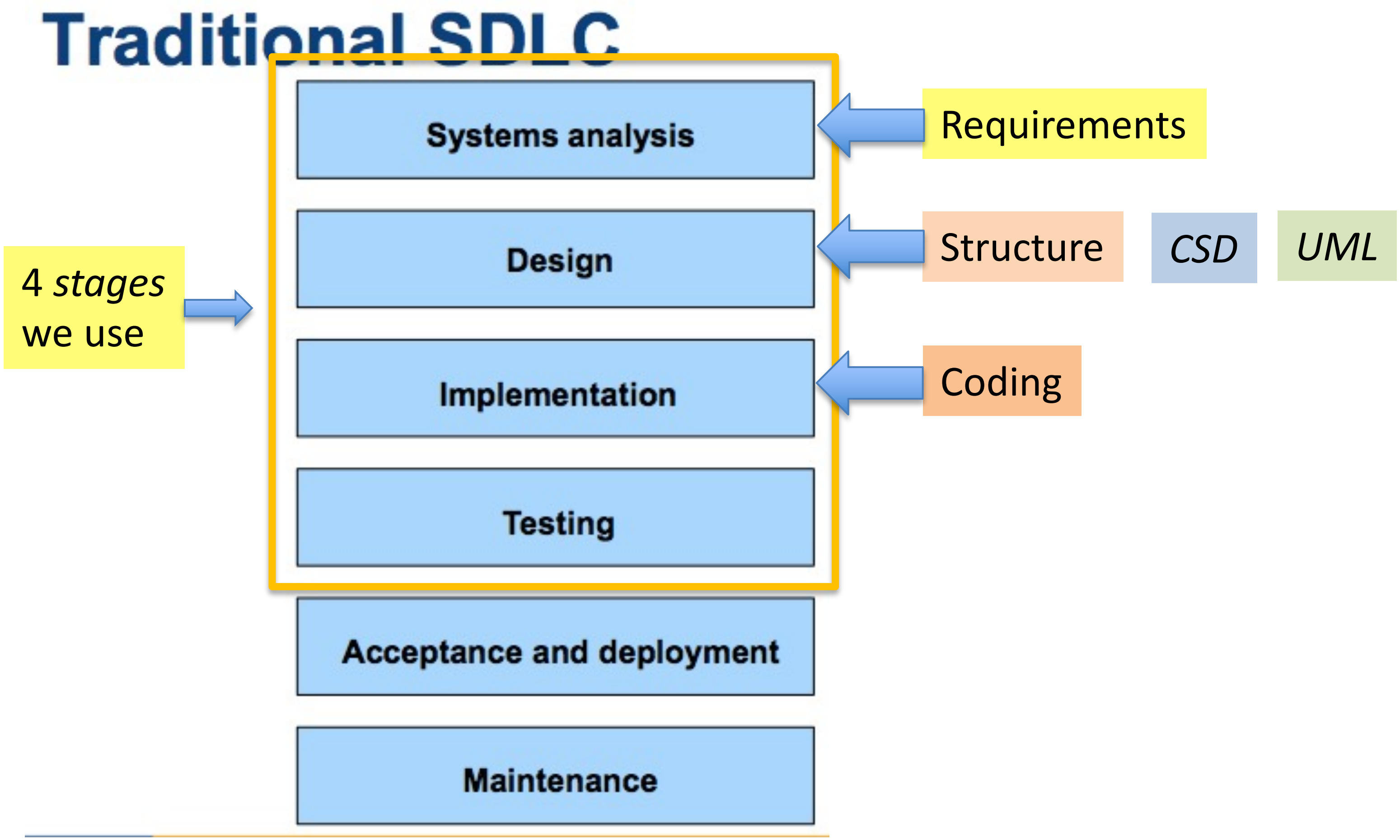
WRITE **PROGRAM**

Select Language

ANALYZE **RESULTS**

SDLC

SOFTWARE DEVELOPMENT LIFE CYCLE



Traditional SDLC

- Systems analysis
- Design
- Implementation
- Testing
- Acceptance and deployment
- Maintenance

4 stages we use

Requirements

Structure *CSD* *UML*

Coding

SDLC

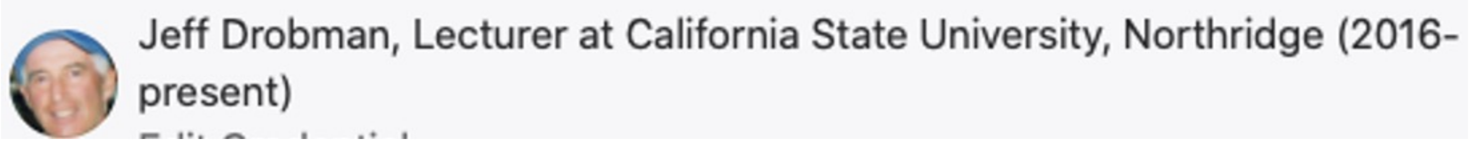
Software Development Life Cycle

- *Software Engineering*

- 6 Stages (we use the 1st four)

- **Requirements**
- **Design**
- **Implementation**
  - Coding
- **Testing**
- Deployment
- Maintenance

Software Engineering



Jeff Drobman, Lecturer at California State University, Northridge (2016-present)

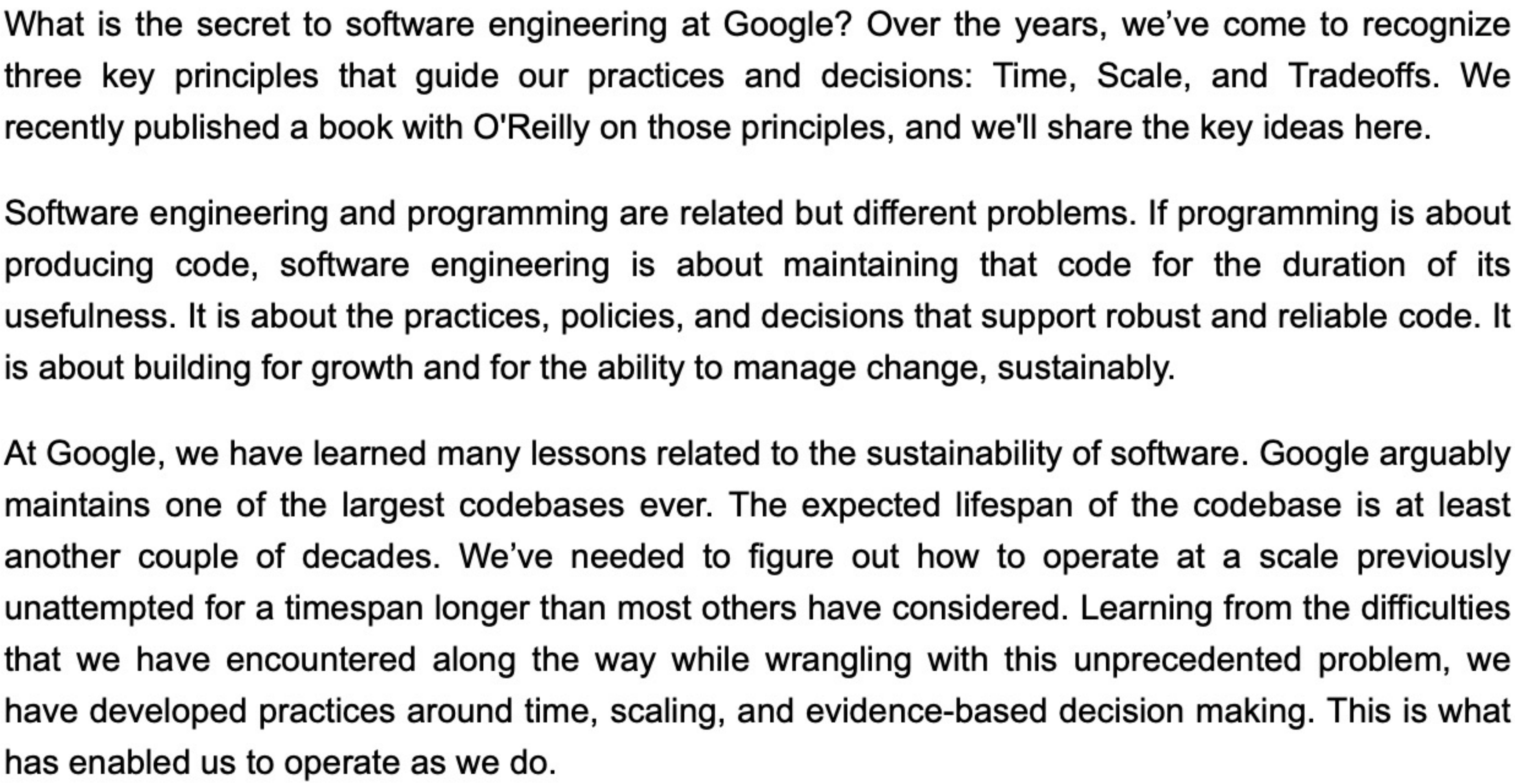
Software Engineering is a specialty of *computer science*, and it uses engineering style disciplines in the construction of correct and robust programs. models such as "SDLC" and "IPO" are the key pillars, along with OOP and its forebear, "structured programming". Also included is "safety engineering" for mission critical applications, and it may include "proof of correctness". These are concepts and models independent of implementation language.

As a *Software Engineer* you will be required to become proficient in several programming languages, plus "design patterns", in addition to the concepts and models.

Most engineering schools only offer computer science at the undergrad level, while offering software engineering at the master's level. CSUN offers a BS in computer science plus an MS in software engineering, as well as in computer science.

Software Engr. at Google

ACM



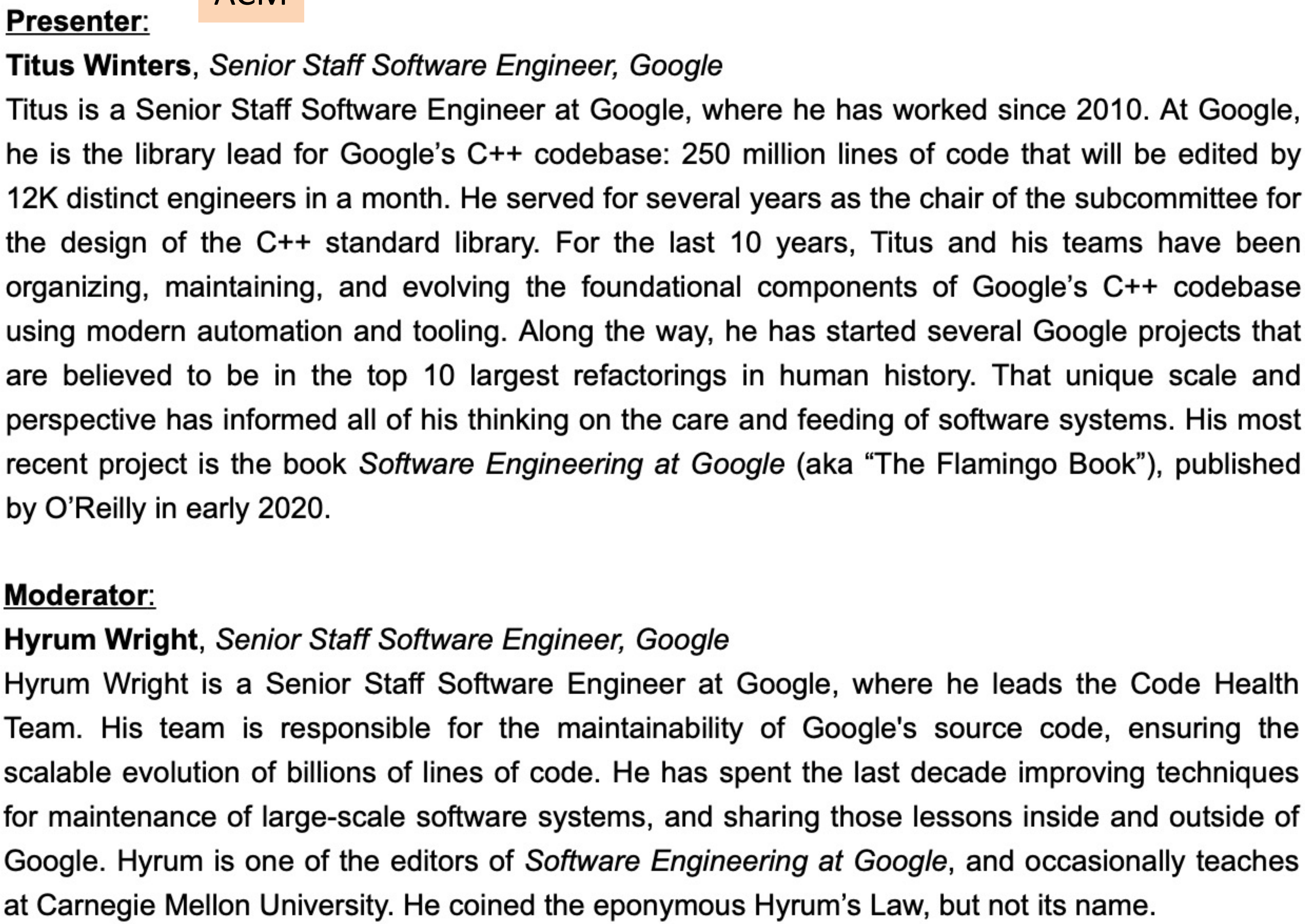
What is the secret to software engineering at Google? Over the years, we've come to recognize three key principles that guide our practices and decisions: Time, Scale, and Tradeoffs. We recently published a book with O'Reilly on those principles, and we'll share the key ideas here.

Software engineering and programming are related but different problems. If programming is about producing code, software engineering is about maintaining that code for the duration of its usefulness. It is about the practices, policies, and decisions that support robust and reliable code. It is about building for growth and for the ability to manage change, sustainably.

At Google, we have learned many lessons related to the sustainability of software. Google arguably maintains one of the largest codebases ever. The expected lifespan of the codebase is at least another couple of decades. We've needed to figure out how to operate at a scale previously unattempted for a timespan longer than most others have considered. Learning from the difficulties that we have encountered along the way while wrangling with this unprecedented problem, we have developed practices around time, scaling, and evidence-based decision making. This is what has enabled us to operate as we do.

Software Engr. at Google

ACM



Presenter:

Titus Winters, *Senior Staff Software Engineer, Google*

Titus is a Senior Staff Software Engineer at Google, where he has worked since 2010. At Google, he is the library lead for Google's C++ codebase: 250 million lines of code that will be edited by 12K distinct engineers in a month. He served for several years as the chair of the subcommittee for the design of the C++ standard library. For the last 10 years, Titus and his teams have been organizing, maintaining, and evolving the foundational components of Google's C++ codebase using modern automation and tooling. Along the way, he has started several Google projects that are believed to be in the top 10 largest refactorings in human history. That unique scale and perspective has informed all of his thinking on the care and feeding of software systems. His most recent project is the book *Software Engineering at Google* (aka "The Flamingo Book"), published by O'Reilly in early 2020.

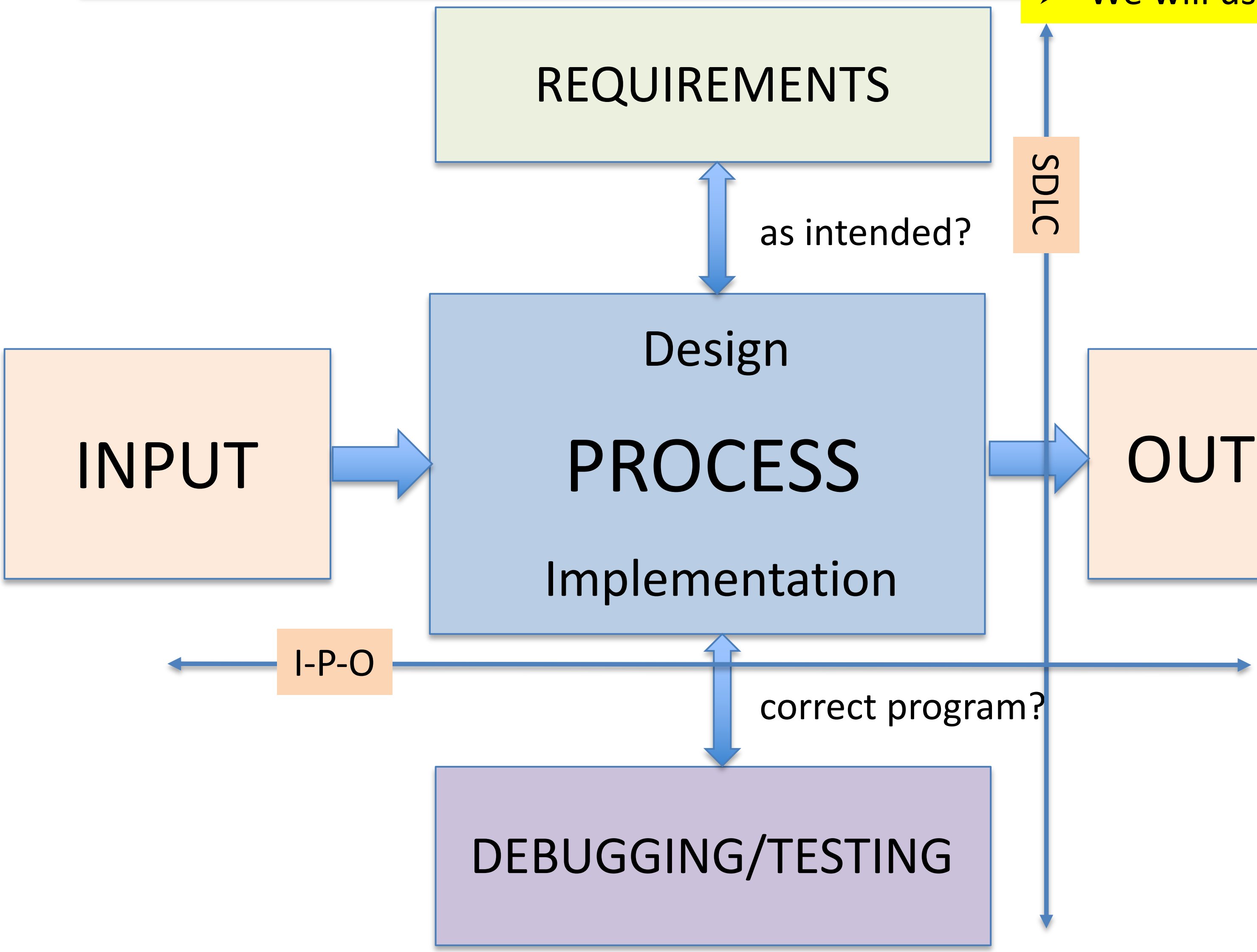
Moderator:

Hyrum Wright, *Senior Staff Software Engineer, Google*

Hyrum Wright is a Senior Staff Software Engineer at Google, where he leads the Code Health Team. His team is responsible for the maintainability of Google's source code, ensuring the scalable evolution of billions of lines of code. He has spent the last decade improving techniques for maintenance of large-scale software systems, and sharing those lessons inside and outside of Google. Hyrum is one of the editors of *Software Engineering at Google*, and occasionally teaches at Carnegie Mellon University. He coined the eponymous Hyrum's Law, but not its name.

SDLC + In-Process-Out

- We will use this model



REQUIREMENTS

as intended?

SDLC

INPUT

Design

PROCESS

Implementation

OUTPUT

I-P-O

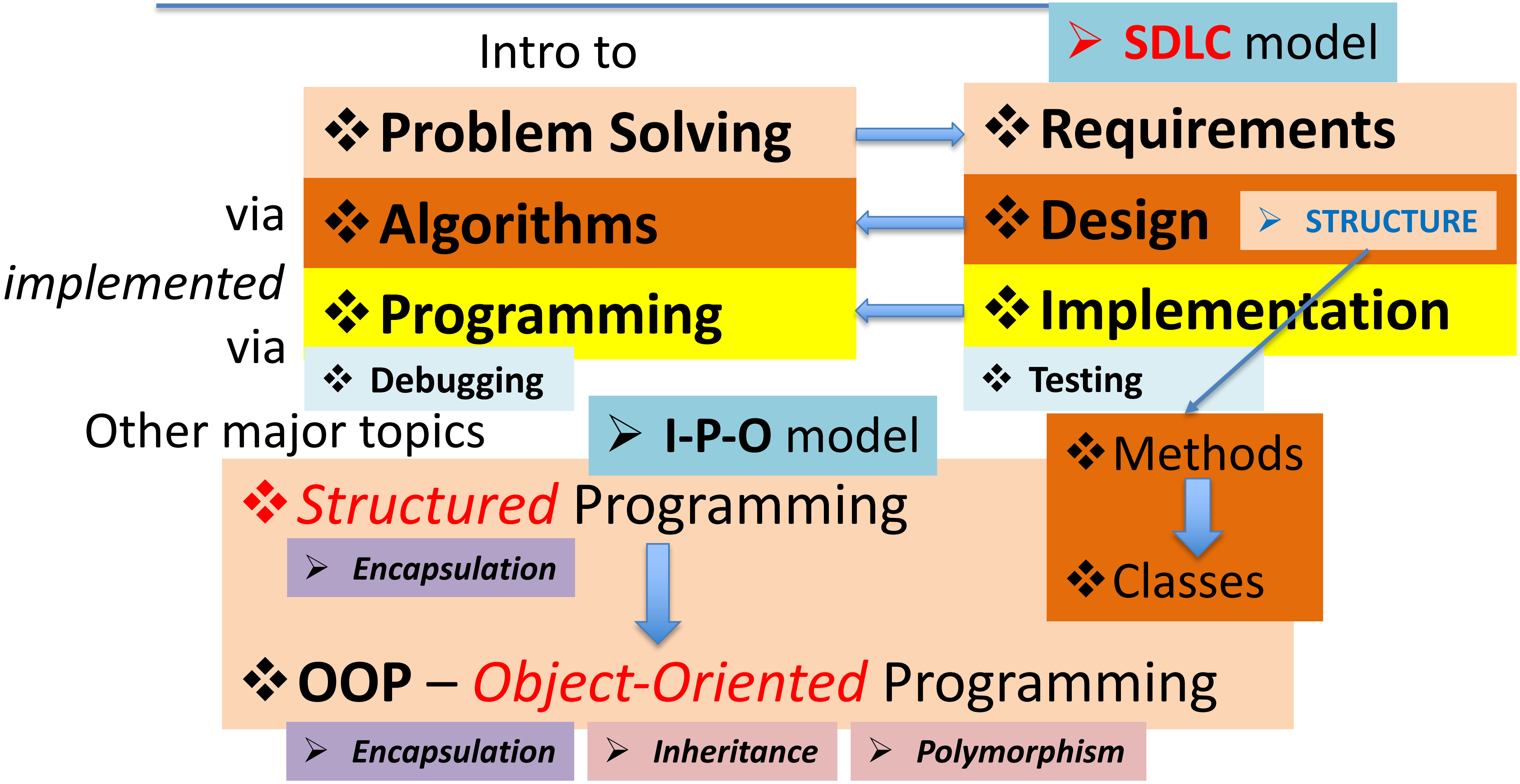
correct program?

DEBUGGING/TESTING

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP110

Course Mapping

DR JEFF SOFTWARE
INDIE APP DEVELOPER



Intro to

- ❖ **Problem Solving**

via

- ❖ **Algorithms**

implemented via

- ❖ **Programming**
  - ❖ **Debugging**

➢ **SDLC** model

- ❖ **Requirements**
- ❖ **Design**
  - ➢ **STRUCTURE**
- ❖ **Implementation**
  - ❖ **Testing**

Other major topics

➢ **I-P-O** model

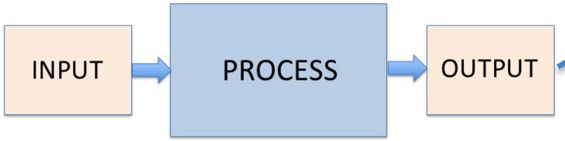
- ❖ *Structured* Programming
  - ➢ ***Encapsulation***
- ❖ **OOP** – *Object-Oriented* Programming
  - ➢ ***Encapsulation***
  - ➢ ***Inheritance***
  - ➢ ***Polymorphism***

- ❖ Methods
- ❖ Classes

Programming Languages

- ❖ Using **JAVA**
- ❖ Compare to **C, C++**

Sandwich Model



- Bread (encapsulation)
  - ❑ Classes
  - ❑ Methods

➢ STRUCTURE

- Greens (passing data)
  - ❑ I/O
  - ❑ Parameter passing

➢ Input/Output

- Cheese (blocks/constructs)
  - ❑ Conditional structures
    - IF-THEN-ELSE
    - SWITCH-CASE
  - ❑ Loops

➢ CONTROL

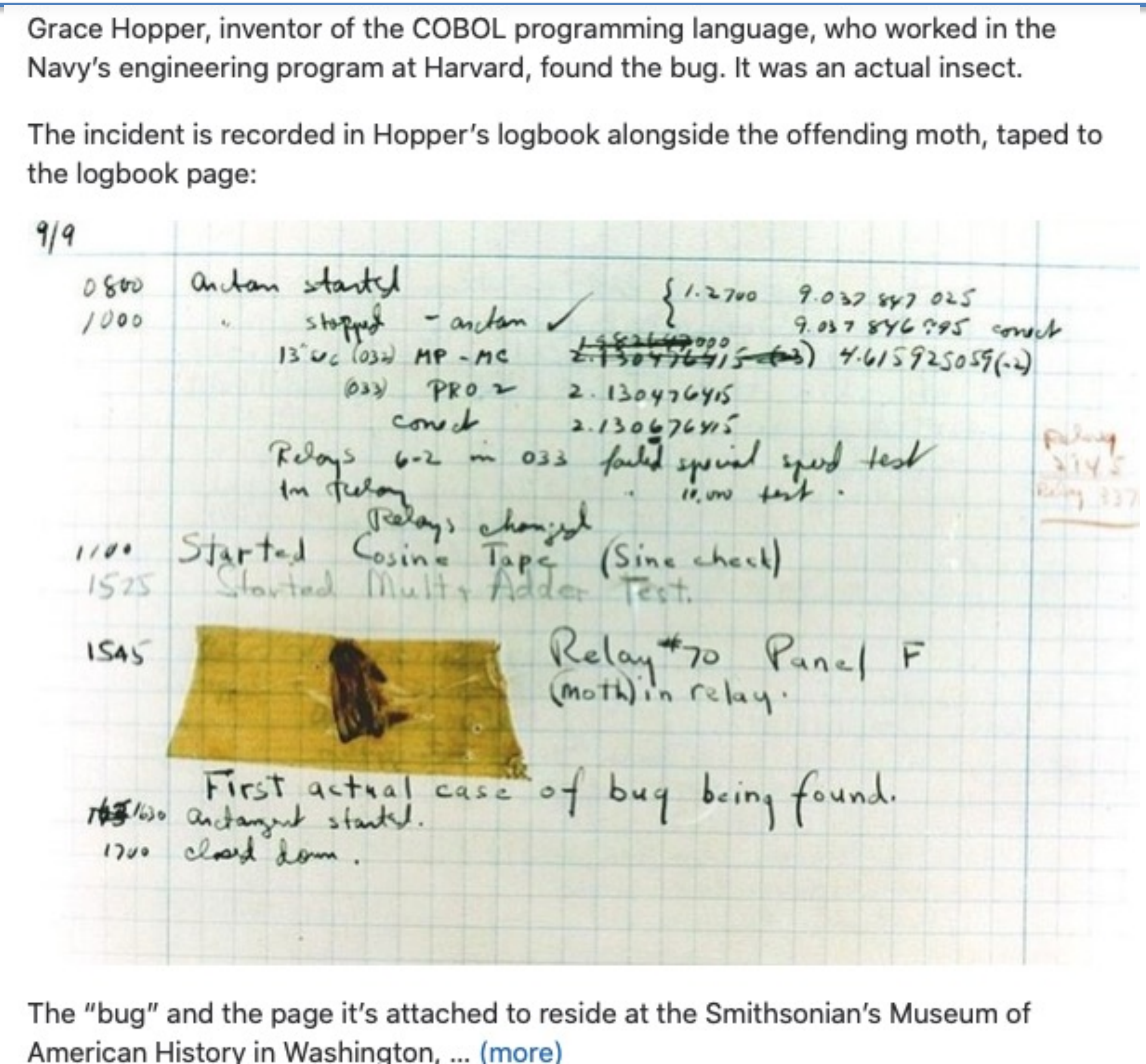
- MEAT (statements→gist)
  - ❑ Arrays
  - ❑ Strings
  - ❑ Expressions
    - Arithmetic
    - Logical

➢ DATA

Java

Debugging

Debugging



Grace Hopper, inventor of the COBOL programming language, who worked in the Navy's engineering program at Harvard, found the bug. It was an actual insect.

The incident is recorded in Hopper's logbook alongside the offending moth, taped to the logbook page:

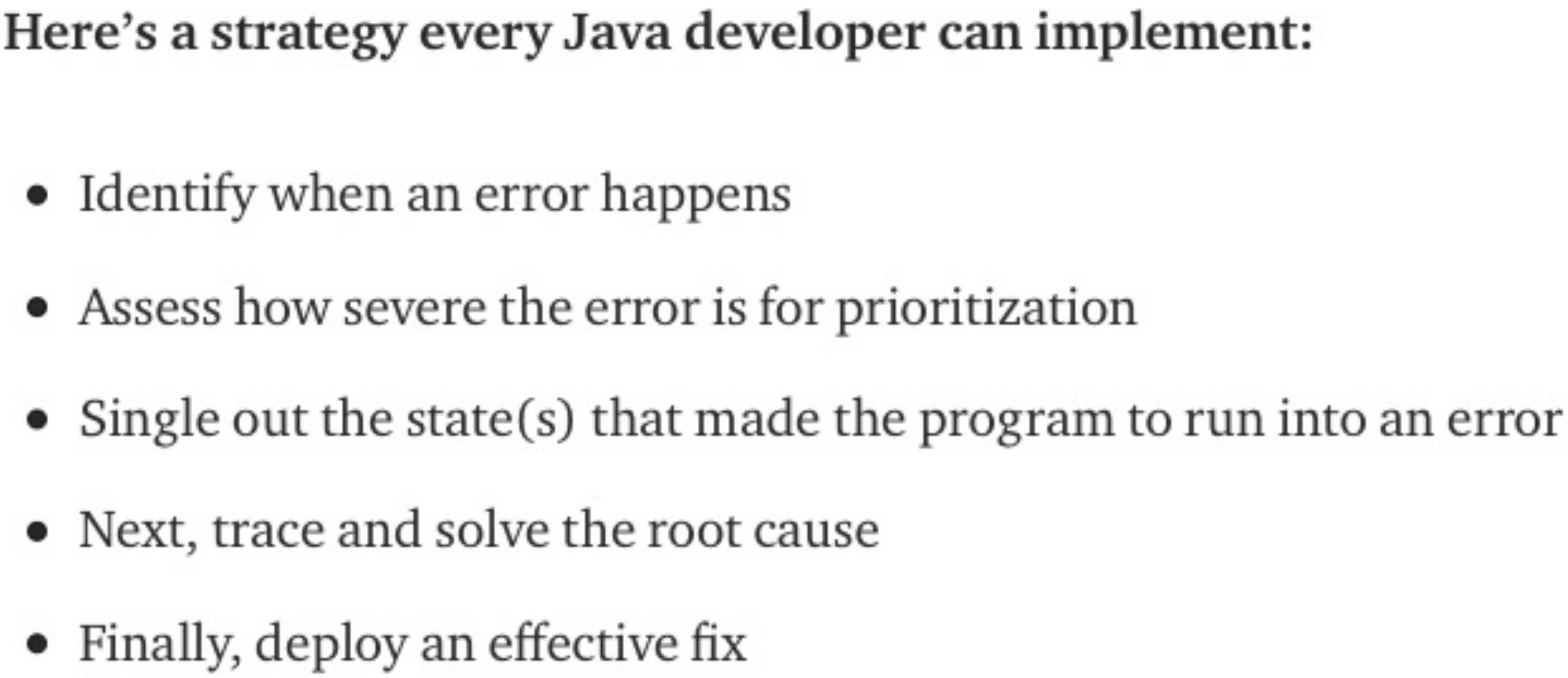
9/9

0800 Anctan started
1000 " stopped - anctan ✓ {1.2700 9.037 847 025
9.037 846 995 conct
13" uc (032) MP - MC ~~1.982147000~~ ~~2.130476415~~ (~~3~~) 4.615925059(-2)
(033) PRO 2 2.130476415
conct 2.130676415
Relays 6-2 in 033 failed special speed test
In relay 11,000 test.
Relays changed
1100 Started Cosine Tape (Sine check)
1525 Started Mult + Adder Test.
1545 Relay #70 Panel F
(moth) in relay.
First actual case of bug being found.
~~1~~1630 Antangent started.
1700 closed down.

Relay 2145
Relay 337

The "bug" and the page it's attached to reside at the Smithsonian's Museum of American History in Washington, ... (more)

Debugging

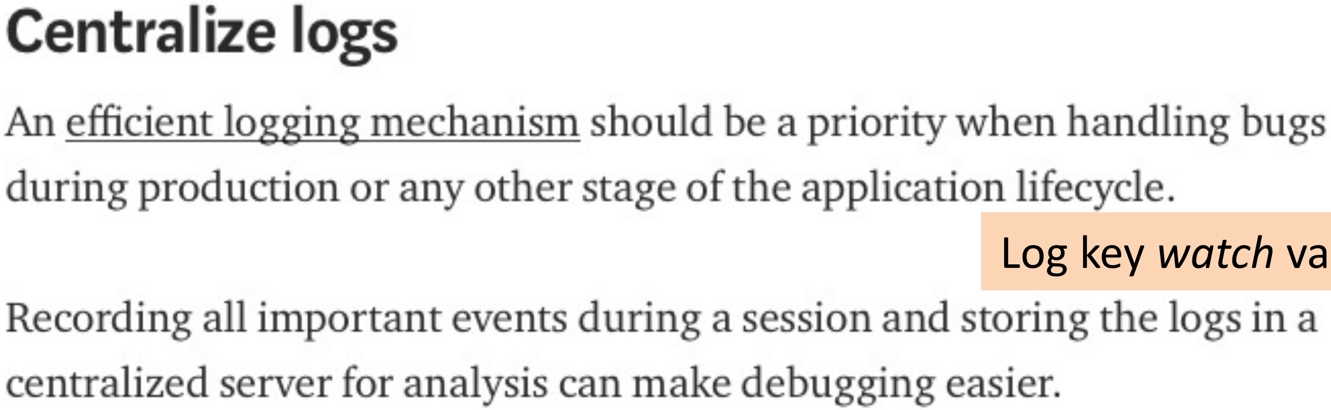


Here’s a strategy every Java developer can implement:

- Identify when an error happens
- Assess how severe the error is for prioritization
- Single out the state(s) that made the program to run into an error
- Next, trace and solve the root cause
- Finally, deploy an effective fix

Debugging

Console Logs

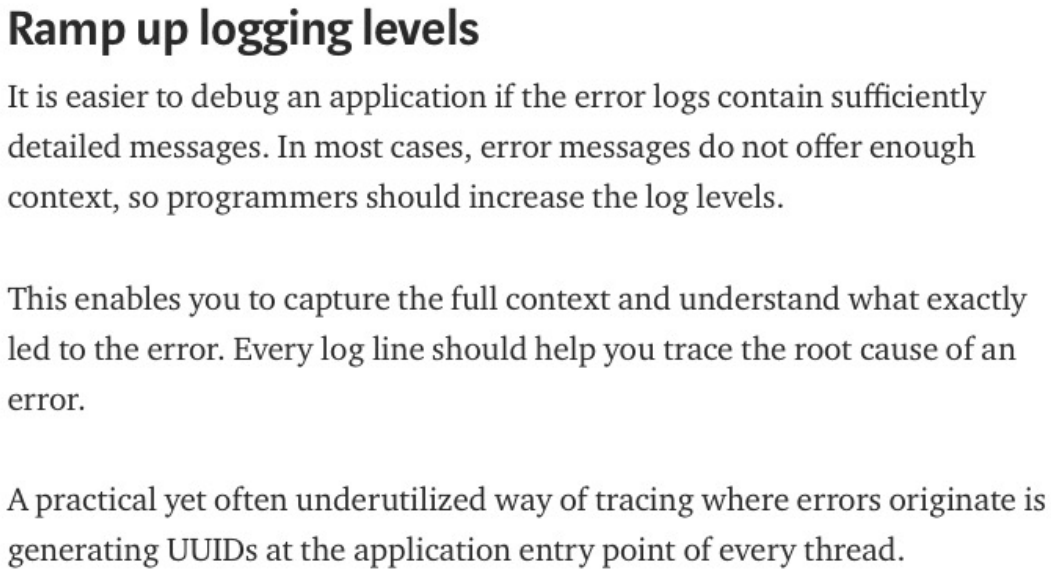


Centralize logs

An efficient logging mechanism should be a priority when handling bugs during production or any other stage of the application lifecycle.

Recording all important events during a session and storing the logs in a centralized server for analysis can make debugging easier.

Log key *watch* values



Ramp up logging levels

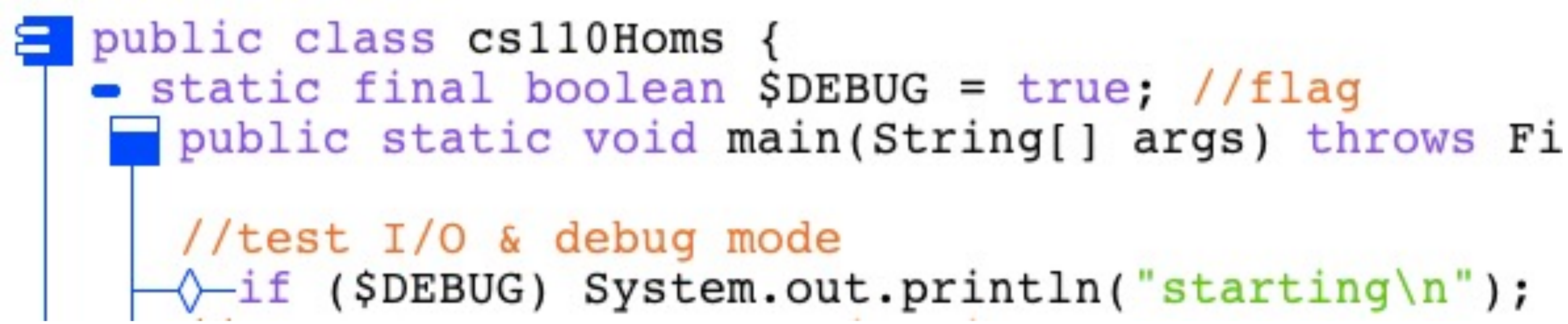
It is easier to debug an application if the error logs contain sufficiently detailed messages. In most cases, error messages do not offer enough context, so programmers should increase the log levels.

This enables you to capture the full context and understand what exactly led to the error. Every log line should help you trace the root cause of an error.

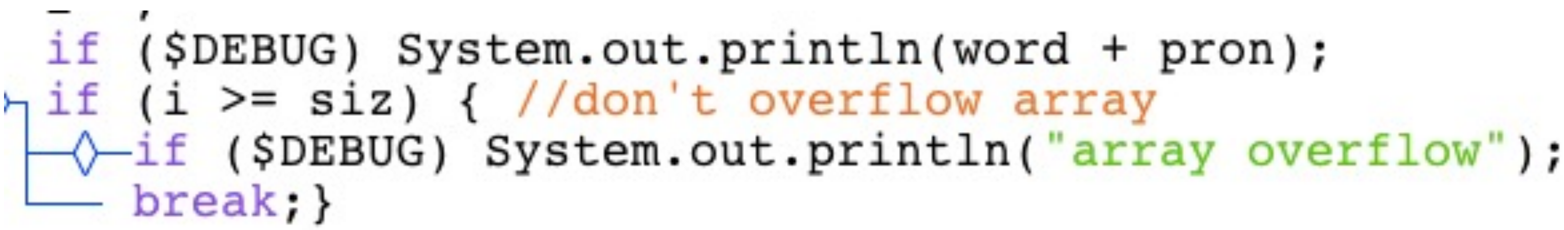
A practical yet often underutilized way of tracing where errors originate is generating UUIDs at the application entry point of every thread.

details

Debugging



```
public class cs110Homs {
  static final boolean $DEBUG = true; //flag
   public static void main(String[] args) throws Fi

      //test I/O & debug mode
      if ($DEBUG) System.out.println("starting\n");
```

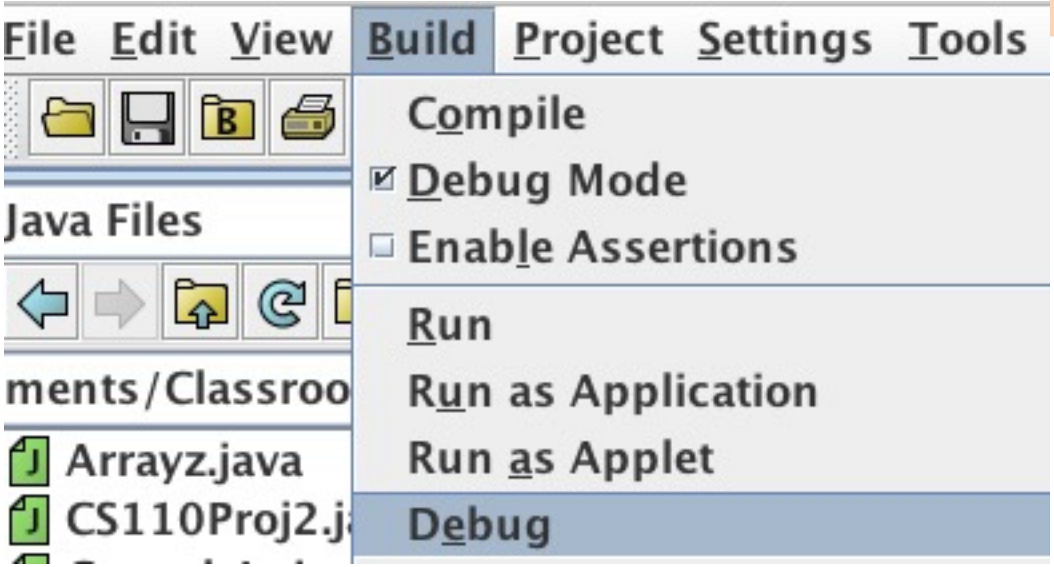


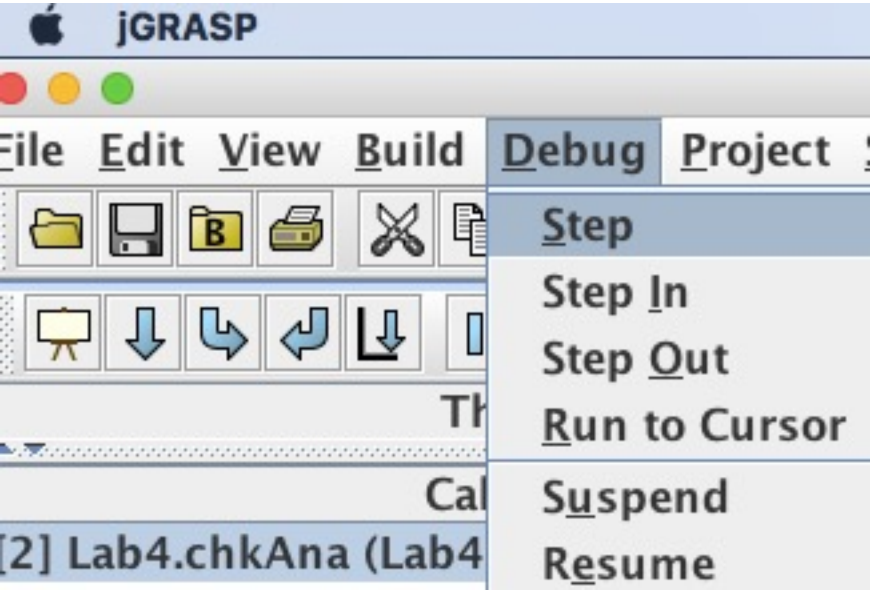
```
 if ($DEBUG) System.out.println(word + pron);
 if (i >= siz) { //don't overflow array
    if ($DEBUG) System.out.println("array overflow");
    break;}
```

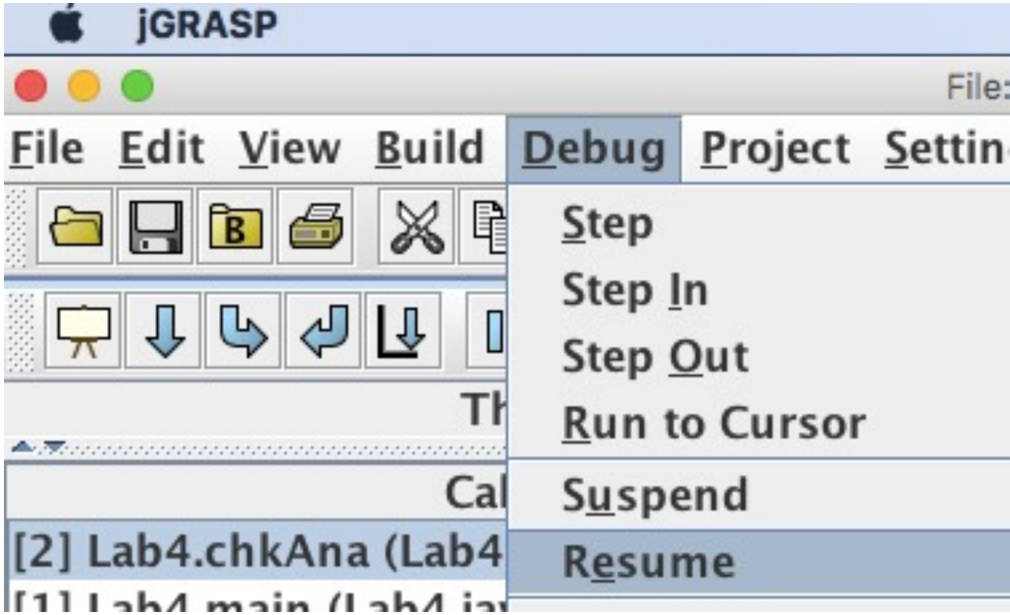
try using “Debug mode” in jGrasp

Debugging

try using “Debug mode” in jGrasp

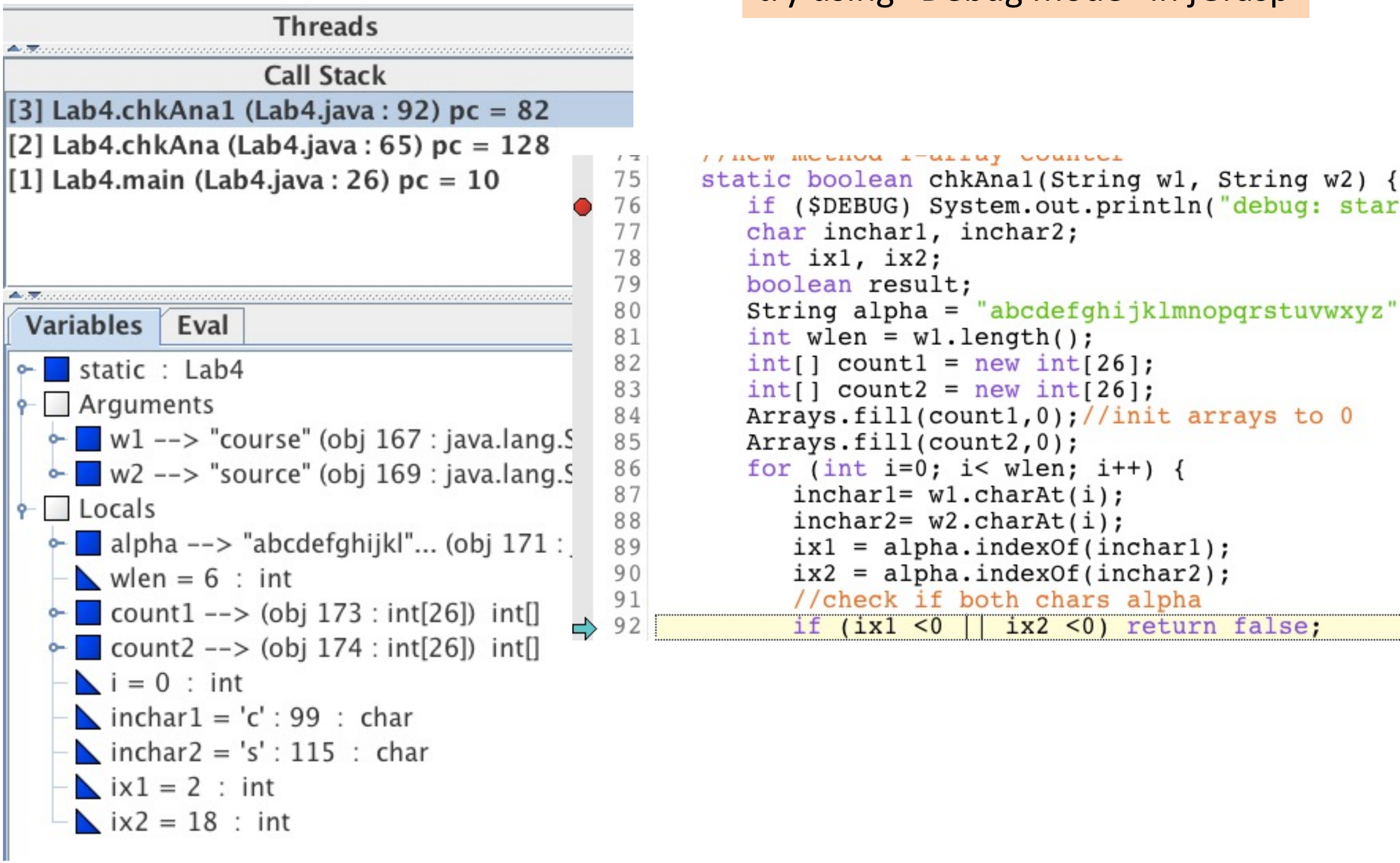






Debugging

try using “Debug mode” in jGrasp



Threads

Call Stack

[3] Lab4.chkAna1 (Lab4.java : 92) pc = 82
[2] Lab4.chkAna (Lab4.java : 65) pc = 128
[1] Lab4.main (Lab4.java : 26) pc = 10

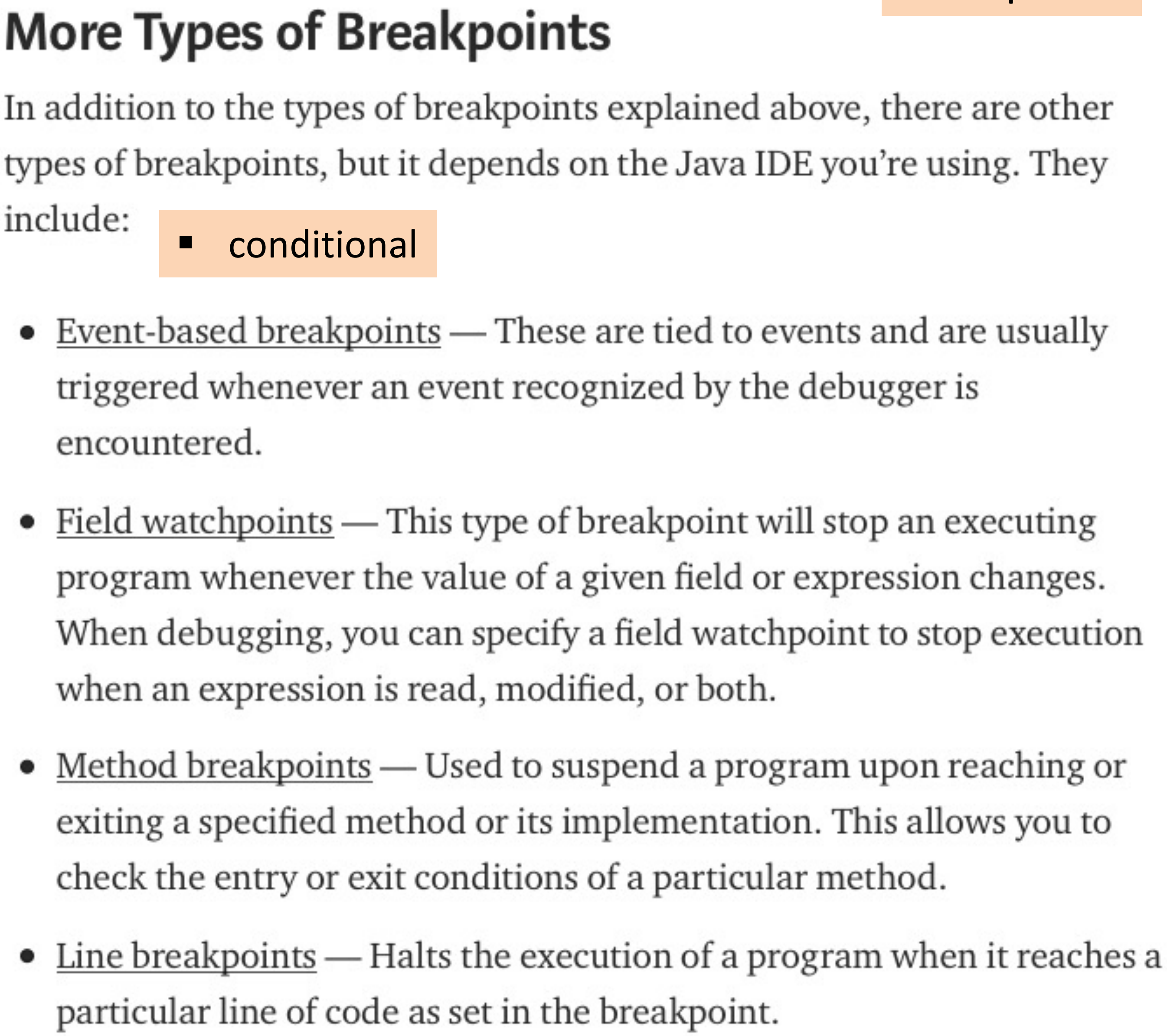
Variables | Eval

```
static : Lab4
Arguments
   w1 --> "course" (obj 167 : java.lang.S
   w2 --> "source" (obj 169 : java.lang.S
Locals
   alpha --> "abcdefghijkl"... (obj 171 : 
   wlen = 6 : int
   count1 --> (obj 173 : int[26]) int[]
   count2 --> (obj 174 : int[26]) int[]
   i = 0 : int
   inchar1 = 'c' : 99 : char
   inchar2 = 's' : 115 : char
   ix1 = 2 : int
   ix2 = 18 : int
```

```
   //new method 1-array counter
   static boolean chkAna1(String w1, String w2) {
      if ($DEBUG) System.out.println("debug: star
      char inchar1, inchar2;
      int ix1, ix2;
      boolean result;
      String alpha = "abcdefghijklmnopqrstuvwxyz"
      int wlen = w1.length();
      int[] count1 = new int[26];
      int[] count2 = new int[26];
      Arrays.fill(count1,0);//init arrays to 0
      Arrays.fill(count2,0);
      for (int i=0; i< wlen; i++) {
         inchar1= w1.charAt(i);
         inchar2= w2.charAt(i);
         ix1 = alpha.indexOf(inchar1);
         ix2 = alpha.indexOf(inchar2);
         //check if both chars alpha
         if (ix1 <0 || ix2 <0) return false;
```

Debugging

Breakpoints



More Types of Breakpoints

In addition to the types of breakpoints explained above, there are other types of breakpoints, but it depends on the Java IDE you're using. They include:

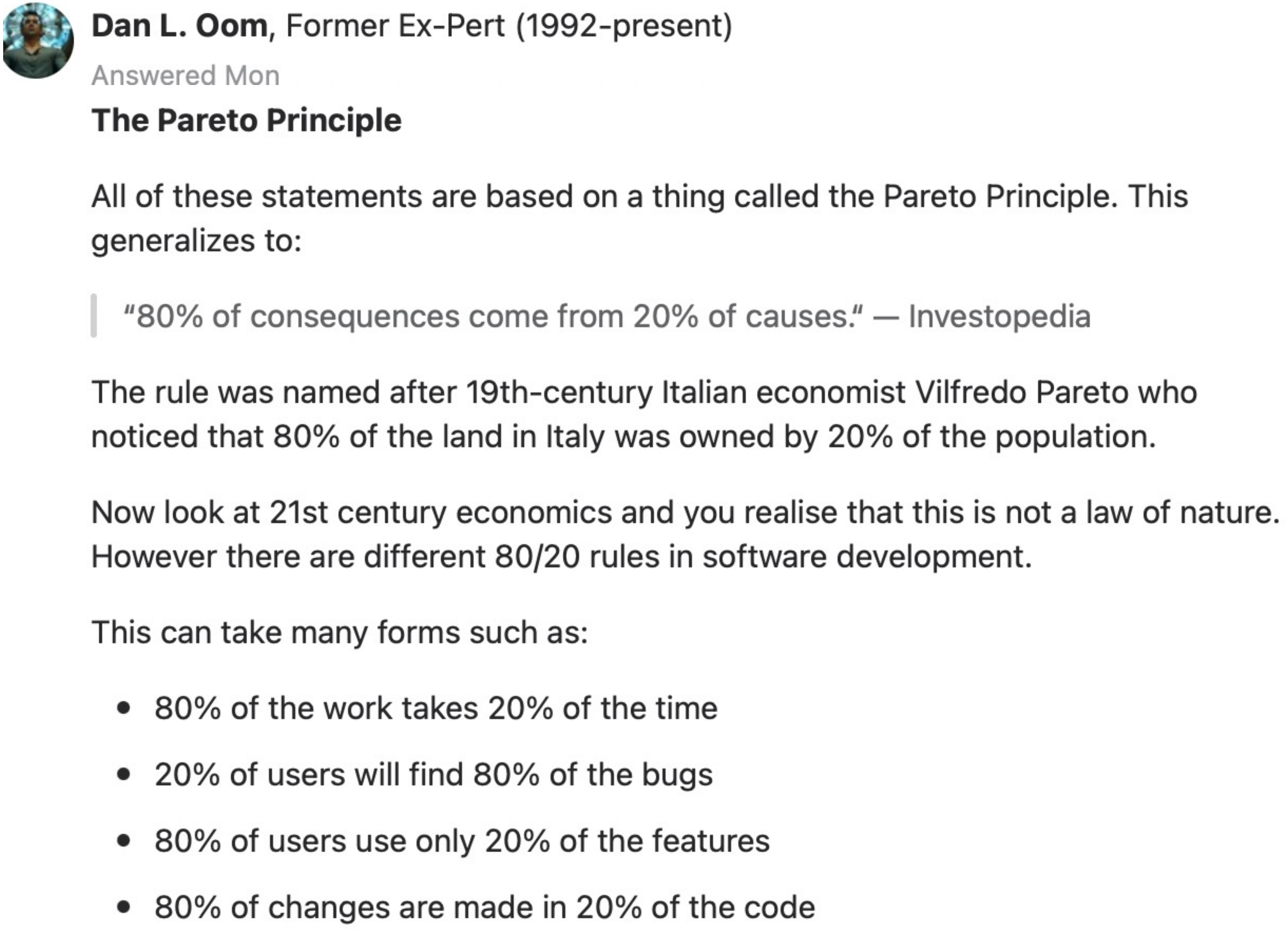
- conditional

- Event-based breakpoints — These are tied to events and are usually triggered whenever an event recognized by the debugger is encountered.
- Field watchpoints — This type of breakpoint will stop an executing program whenever the value of a given field or expression changes. When debugging, you can specify a field watchpoint to stop execution when an expression is read, modified, or both.
- Method breakpoints — Used to suspend a program upon reaching or exiting a specified method or its implementation. This allows you to check the entry or exit conditions of a particular method.
- Line breakpoints — Halts the execution of a program when it reaches a particular line of code as set in the breakpoint.

usual

Pareto in Code Practice

Quora



Dan L. Oom, Former Ex-Pert (1992-present)

Answered Mon

The Pareto Principle

All of these statements are based on a thing called the Pareto Principle. This generalizes to:

> "80% of consequences come from 20% of causes." — Investopedia

The rule was named after 19th-century Italian economist Vilfredo Pareto who noticed that 80% of the land in Italy was owned by 20% of the population.

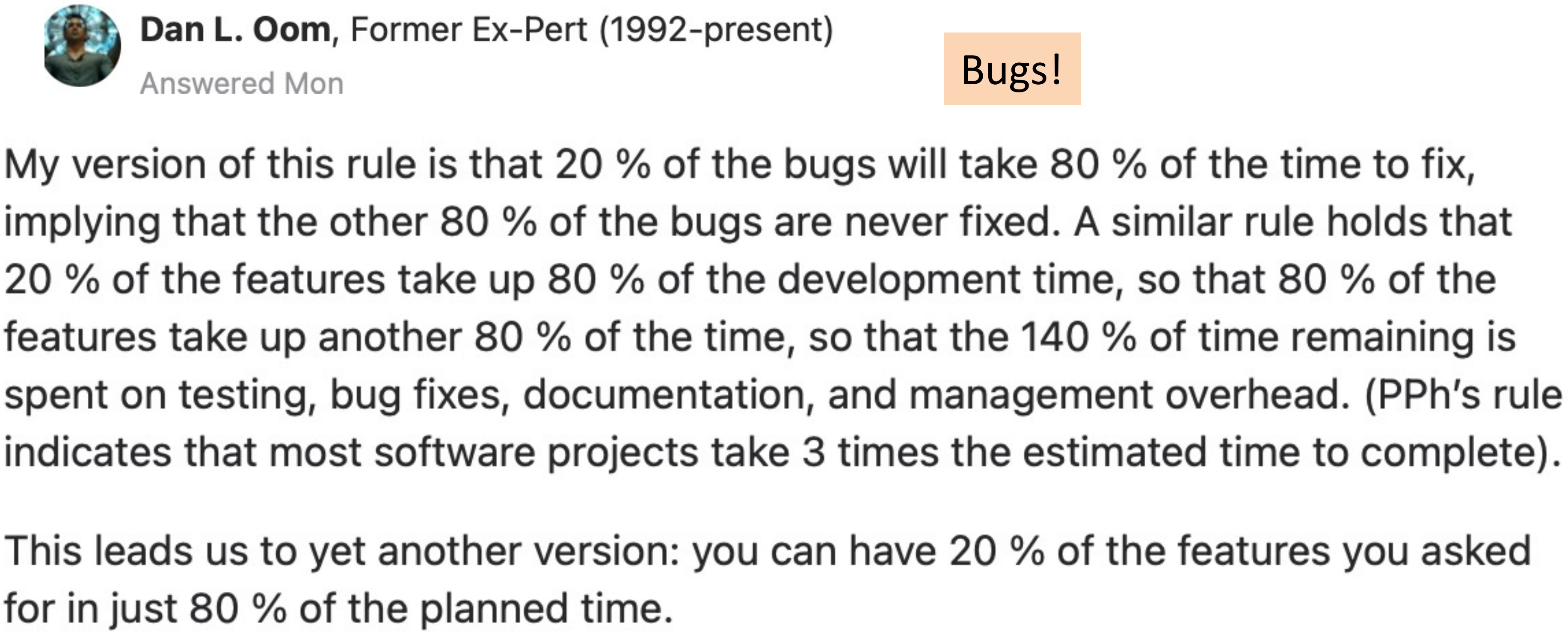
Now look at 21st century economics and you realise that this is not a law of nature. However there are different 80/20 rules in software development.

This can take many forms such as:

- 80% of the work takes 20% of the time
- 20% of users will find 80% of the bugs
- 80% of users use only 20% of the features
- 80% of changes are made in 20% of the code

Pareto in Code Practice

Quora



Dan L. Oom, Former Ex-Pert (1992-present)

Answered Mon

My version of this rule is that 20 % of the bugs will take 80 % of the time to fix, implying that the other 80 % of the bugs are never fixed. A similar rule holds that 20 % of the features take up 80 % of the development time, so that 80 % of the features take up another 80 % of the time, so that the 140 % of time remaining is spent on testing, bug fixes, documentation, and management overhead. (PPh's rule indicates that most software projects take 3 times the estimated time to complete).

This leads us to yet another version: you can have 20 % of the features you asked for in just 80 % of the planned time.

Bugs!

Java

Intro to

Structure

Code + Data

Software

❖ **Code** structure

- ❑ Macro
  - ▪ Classes
  - ▪ Methods
- ❑ Control (Micro)
  - ▪ *Conditional* blocks
    - If
    - Switch - Case
  - ▪ Loops
    - For
    - While

❖ **Data *structure***

- ❑ In Code
  - ▪ **Arrays**
  - ▪ Array-Lists
  - ▪ Enums
  - ▪ Collections
- ❑ In Files
  - ▪ **CSV files**
  - ▪ Databases

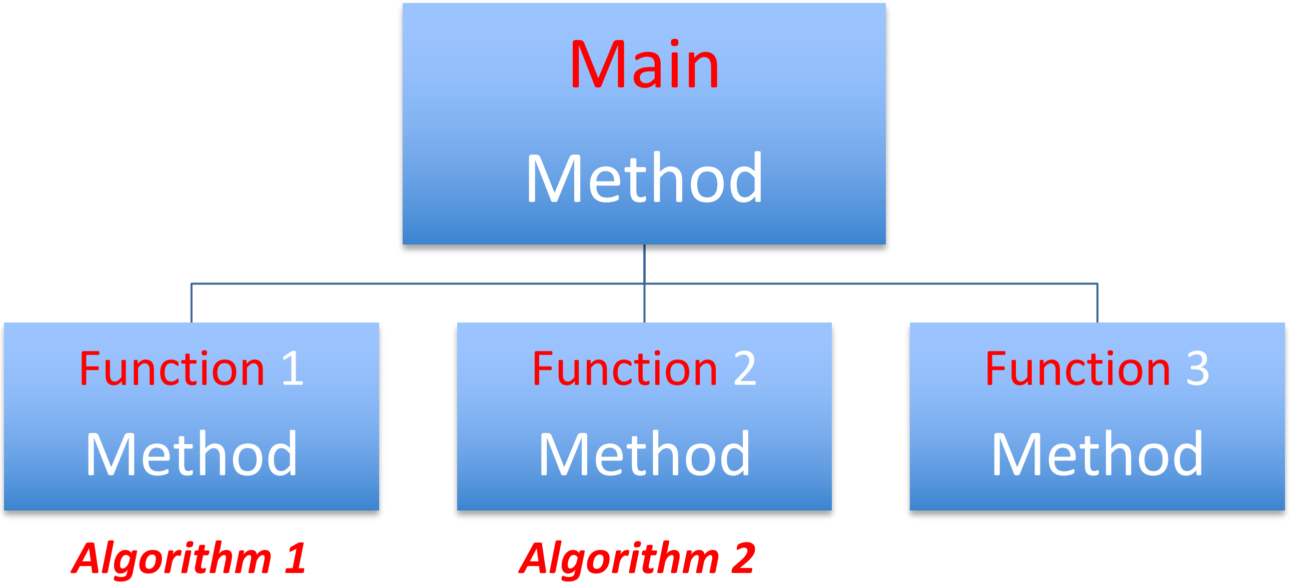
❖ **Data *types***

- ❑ Numeric
  - ▪ Integer
  - ▪ Floating-point
- ❑ Non-Numeric
  - ▪ Logic (Boolean)
  - ▪ Characters
  - ▪ Strings

Code Structure

Tree structure

Structure Mapped to ***Requirements***



- Not all ***Functions*** are implemented by ***Algorithms***

Code Structure (Macro)

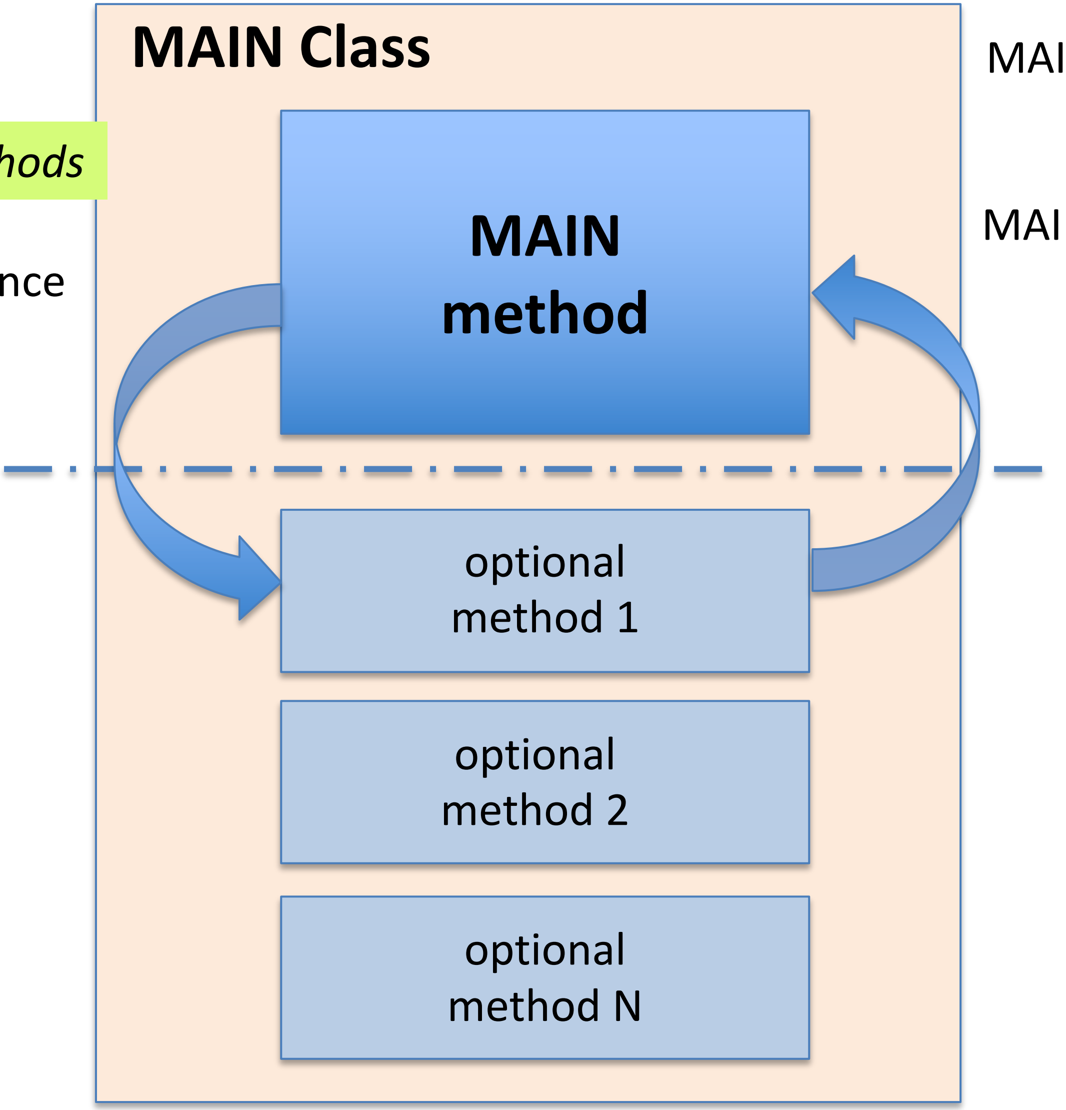
Java ⇔ OOP

OOP
Structures

Classes/methods

Execution is
by *call* sequence

Minimum
required



MAIN *Class* = any name

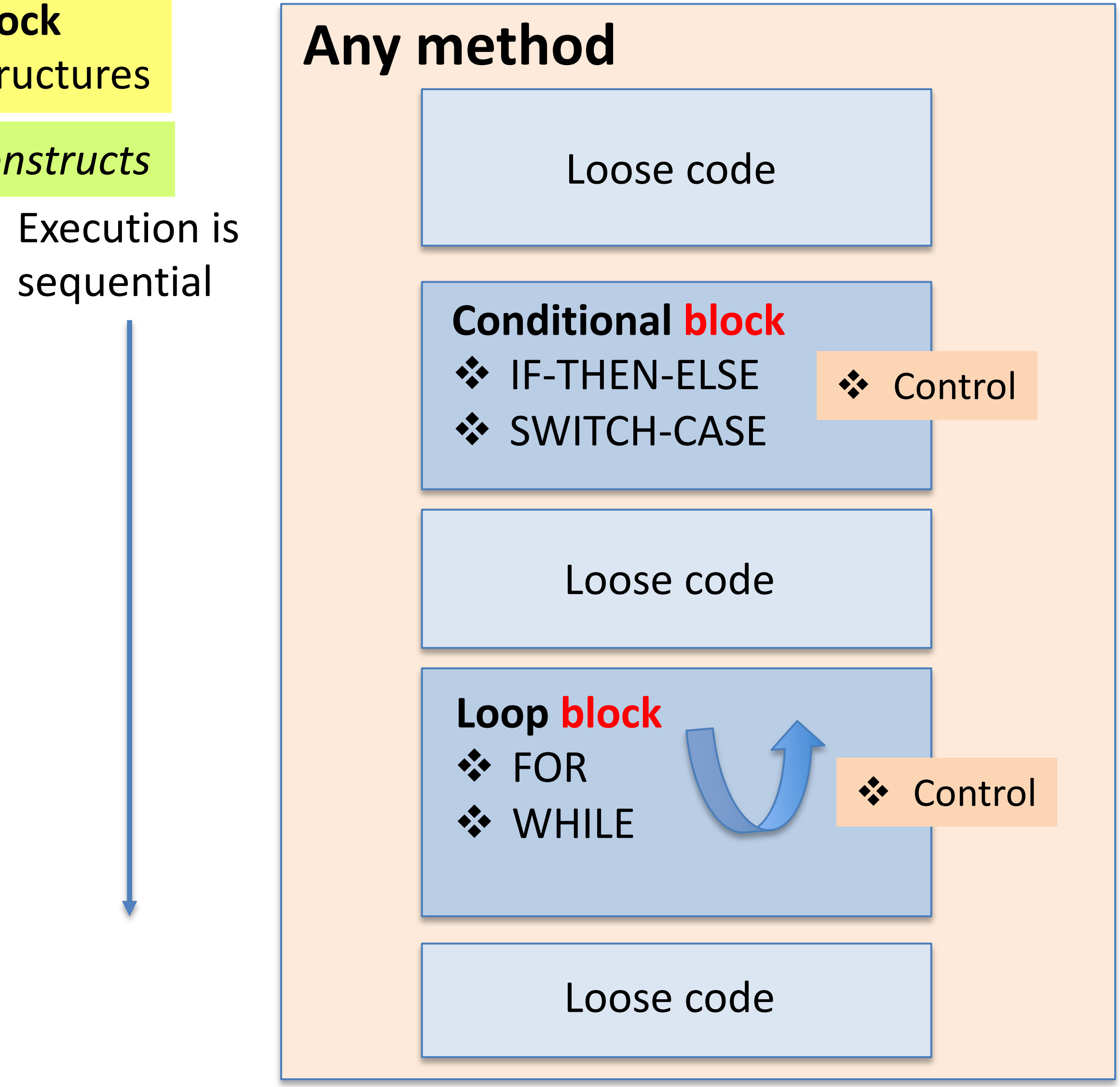
MAIN *Method* = “main”

Code Structure (Micro)

Java ⇔ OOP

Block Structures

Constructs



Execution is sequential

Any method

Loose code

Conditional block
- ❖ IF-THEN-ELSE
- ❖ SWITCH-CASE

❖ Control

Loose code

Loop block
- ❖ FOR
- ❖ WHILE

❖ Control

Loose code

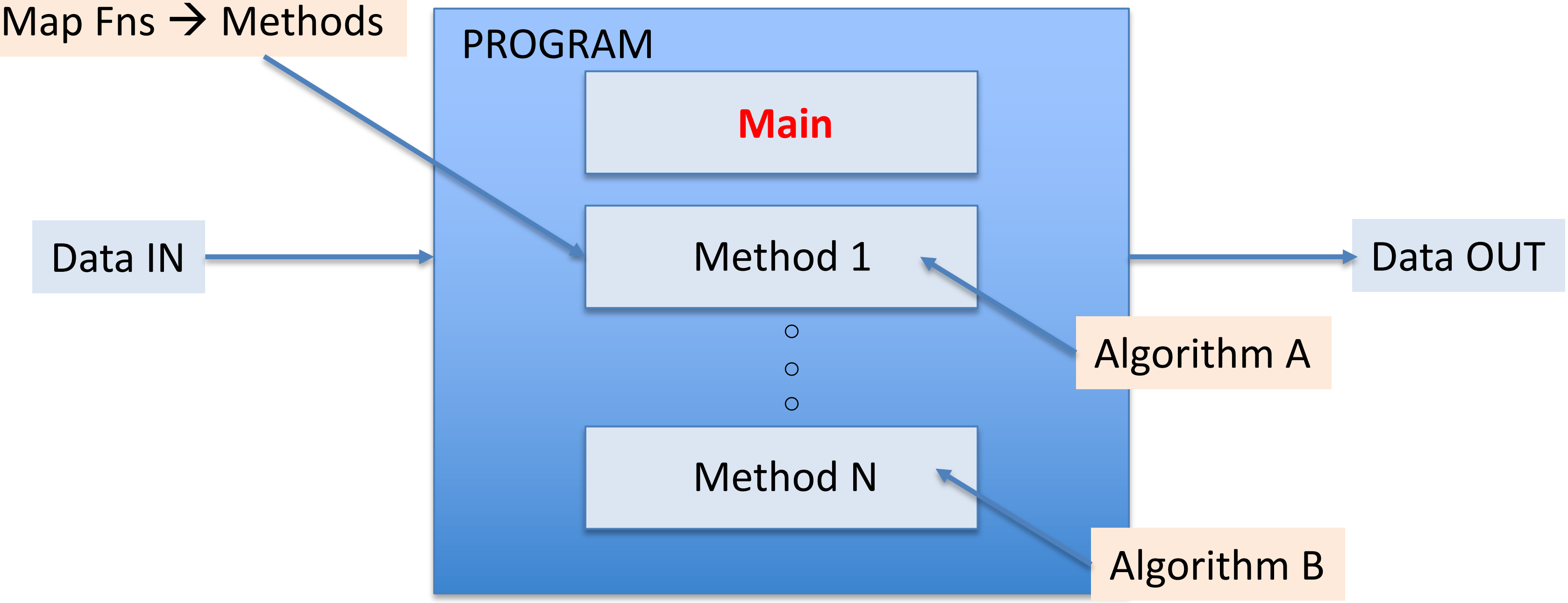
Programs & Algorithms

Nested Structure

Requirements

Procedure → Process

- ➢ Map Fns → Methods



PROGRAM

Main

Method 1

○
○
○

Method N

Data IN

Data OUT

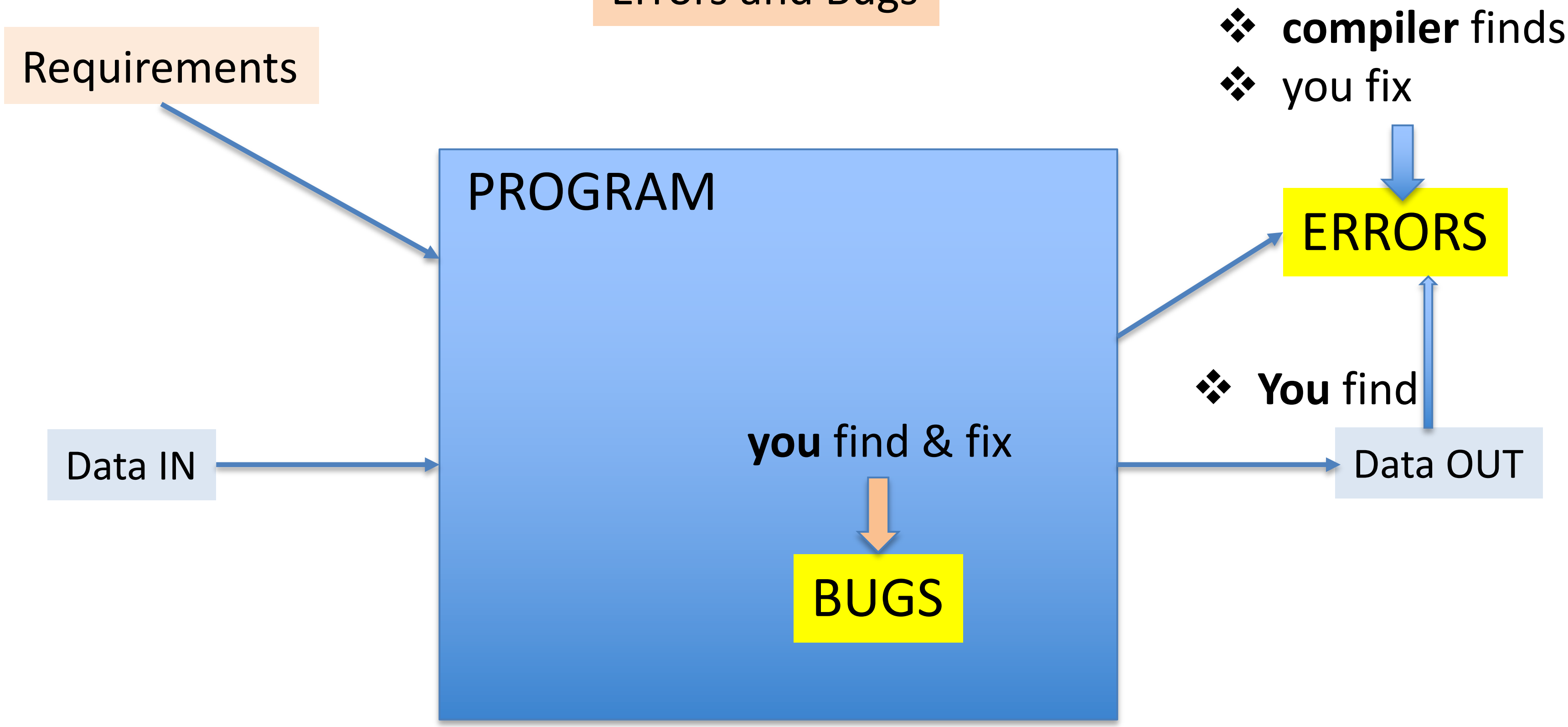
Algorithm A

Algorithm B

- ➢ 1 Algorithm per *function* or *method*

Program Debugging

Errors and Bugs



❖ **compiler** finds

❖ you fix

Requirements

PROGRAM

ERRORS

❖ **You** find

Data IN

you find & fix

Data OUT

BUGS

- ❖ *ERRORS*
  - ❑ syntax → compiler
  - ❑ semantic/logic → debug
  - ❑ exceptions → handlers

- ❖ *BUGS*
  - ❑ *DEBUG*
  - ❑ *in situ* → IF-THEN
  - ❑ exception handlers

Java

Reference

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP110

Meta-language: BNF

DR JEFF SOFTWARE
INDIE APP DEVELOPER

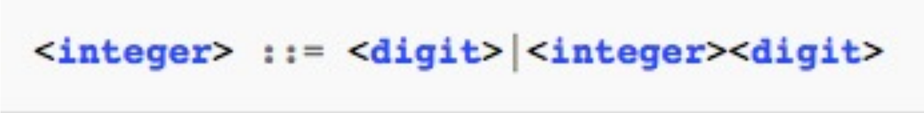
❖BNF = Bachus-Naur Form

<item to be replaced>
[optional item]
::= → "equals" (is defined as)
<item 1 | item 2> → "OR"
<item 1 …> → ellipsis (any number more)
<1..N> → 1 "to" N

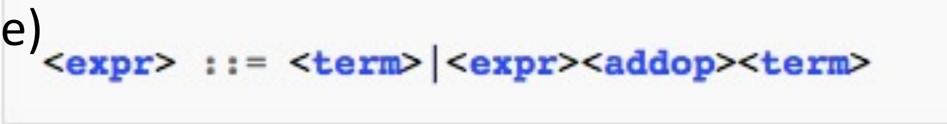
Backus–Naur Form

From Wikipedia, the free encyclopedia

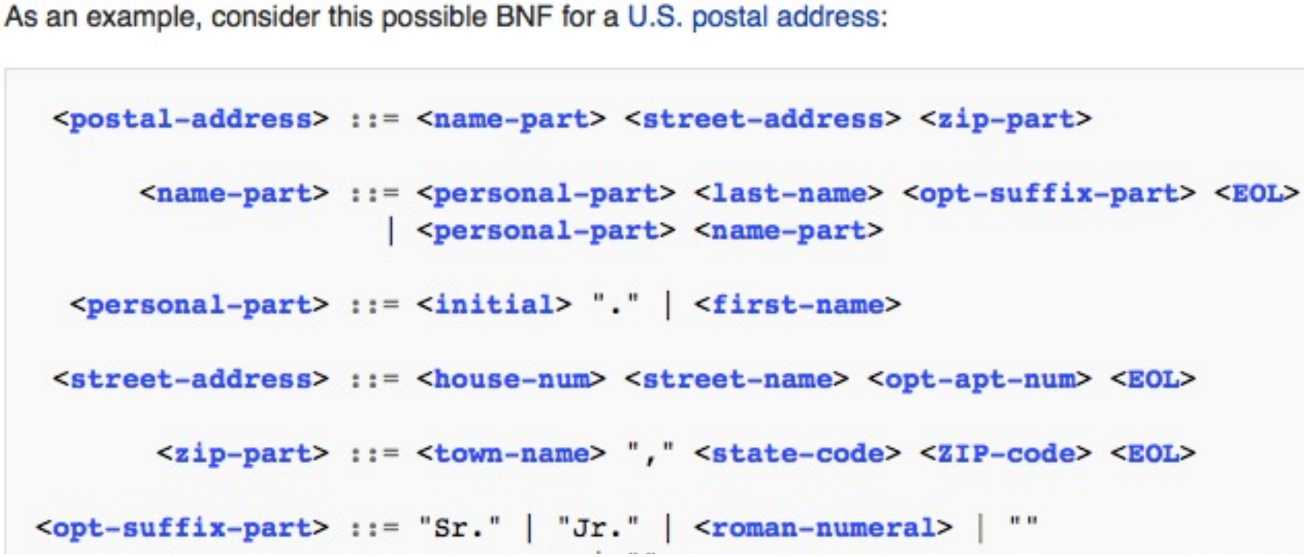
Examples



```
<integer> ::= <digit>|<integer><digit>
```

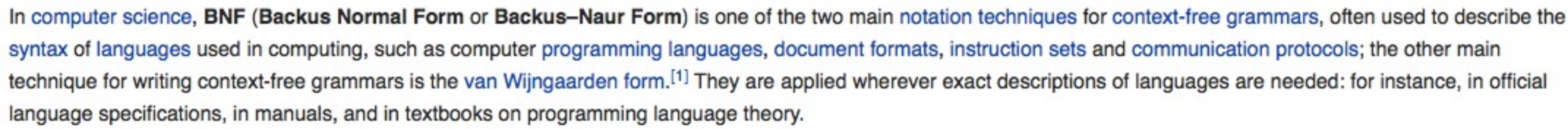


```
<expr> ::= <term>|<expr><addop><term>
```



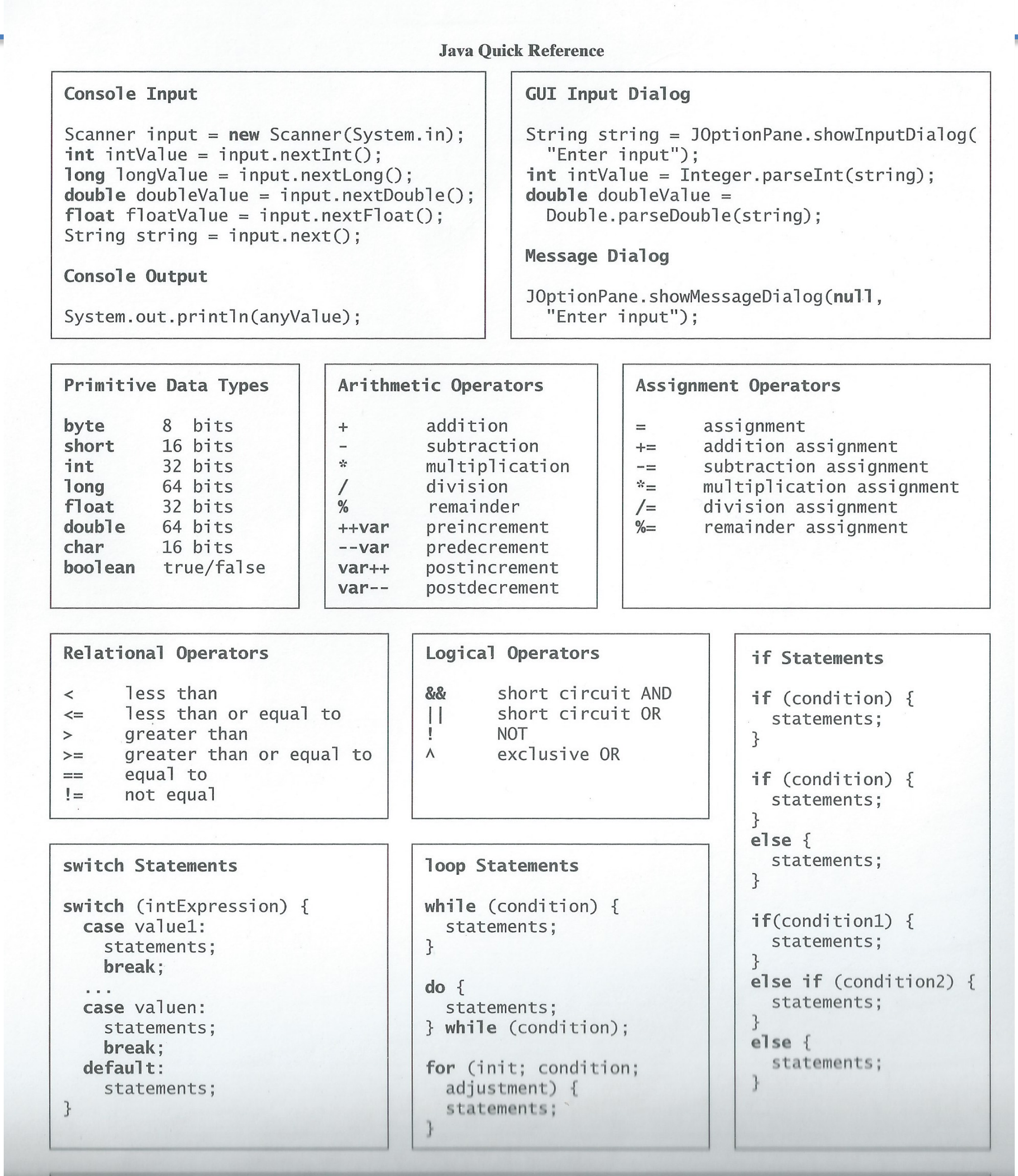
As an example, consider this possible BNF for a U.S. postal address:

```
 <postal-address> ::= <name-part> <street-address> <zip-part>

      <name-part> ::= <personal-part> <last-name> <opt-suffix-part> <EOL>
                    | <personal-part> <name-part>

  <personal-part> ::= <initial> "." | <first-name>

 <street-address> ::= <house-num> <street-name> <opt-apt-num> <EOL>

       <zip-part> ::= <town-name> "," <state-code> <ZIP-code> <EOL>

<opt-suffix-part> ::= "Sr." | "Jr." | <roman-numeral> | ""
```



In computer science, **BNF** (**Backus Normal Form** or **Backus–Naur Form**) is one of the two main notation techniques for context-free grammars, often used to describe the syntax of languages used in computing, such as computer programming languages, document formats, instruction sets and communication protocols; the other main technique for writing context-free grammars is the van Wijngaarden form.[1] They are applied wherever exact descriptions of languages are needed: for instance, in official language specifications, in manuals, and in textbooks on programming language theory.

Java Ref Pg 1



Java Quick Reference

Console Input

```
Scanner input = new Scanner(System.in);
int intValue = input.nextInt();
long longValue = input.nextLong();
double doubleValue = input.nextDouble();
float floatValue = input.nextFloat();
String string = input.next();
```

Console Output

```
System.out.println(anyValue);
```

GUI Input Dialog

```
String string = JOptionPane.showInputDialog(
  "Enter input");
int intValue = Integer.parseInt(string);
double doubleValue =
  Double.parseDouble(string);
```

Message Dialog

```
JOptionPane.showMessageDialog(null,
  "Enter input");
```

Primitive Data Types

byte	8 bits
short	16 bits
int	32 bits
long	64 bits
float	32 bits
double	64 bits
char	16 bits
boolean	true/false

Arithmetic Operators

+	addition
-	subtraction
*	multiplication
/	division
%	remainder
++var	preincrement
--var	predecrement
var++	postincrement
var--	postdecrement

Assignment Operators

=	assignment
+=	addition assignment
-=	subtraction assignment
*=	multiplication assignment
/=	division assignment
%=	remainder assignment

Relational Operators

<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	equal to
!=	not equal

Logical Operators

&&	short circuit AND
||	short circuit OR
!	NOT
^	exclusive OR

if Statements

```
if (condition) {
  statements;
}

if (condition) {
  statements;
}
else {
  statements;
}

if(condition1) {
  statements;
}
else if (condition2) {
  statements;
}
else {
  statements;
}
```

switch Statements

```
switch (intExpression) {
  case value1:
    statements;
    break;
  ...
  case valuen:
    statements;
    break;
  default:
    statements;
}
```

loop Statements

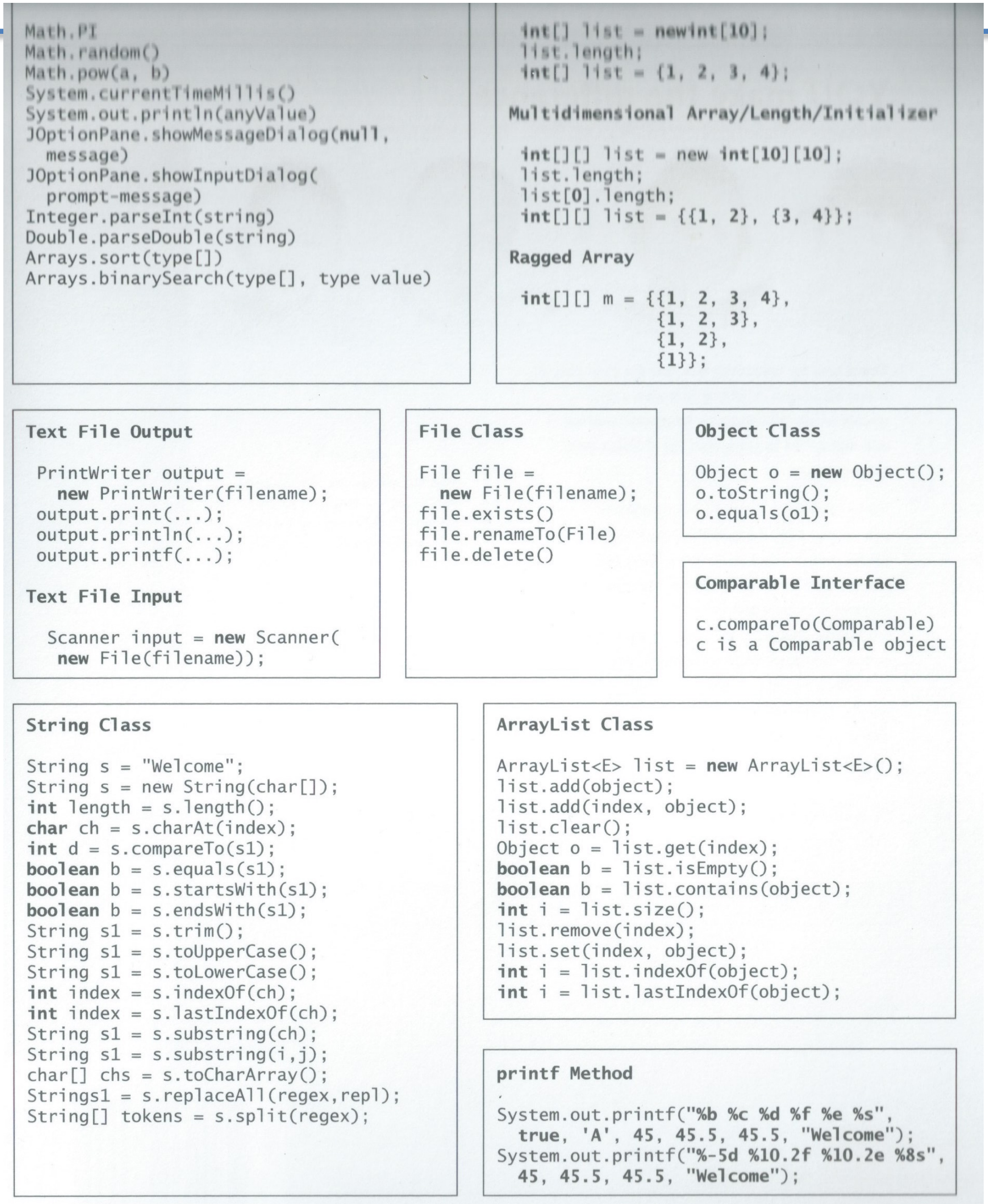
```
while (condition) {
  statements;
}

do {
  statements;
} while (condition);

for (init; condition;
  adjustment) {
  statements;
}
```

CSUN
CALIFORNIA
STATE UNIVERSITY
NORTHRIDGE

COMP110

Java Ref Pg 2

DR JEFF SOFTWARE
INDIE APP DEVELOPER
©2016-2022
Jeff Drobman



```
Math.PI
Math.random()
Math.pow(a, b)
System.currentTimeMillis()
System.out.println(anyValue)
JOptionPane.showMessageDialog(null,
  message)
JOptionPane.showInputDialog(
  prompt-message)
Integer.parseInt(string)
Double.parseDouble(string)
Arrays.sort(type[])
Arrays.binarySearch(type[], type value)
```

```
 int[] list = newint[10];
 list.length;
 int[] list = {1, 2, 3, 4};

Multidimensional Array/Length/Initializer

 int[][] list = new int[10][10];
 list.length;
 list[0].length;
 int[][] list = {{1, 2}, {3, 4}};

Ragged Array

 int[][] m = {{1, 2, 3, 4},
              {1, 2, 3},
              {1, 2},
              {1}};
```

```
Text File Output

 PrintWriter output =
   new PrintWriter(filename);
 output.print(...);
 output.println(...);
 output.printf(...);

Text File Input

  Scanner input = new Scanner(
   new File(filename));
```

```
File Class

File file =
  new File(filename);
file.exists()
file.renameTo(File)
file.delete()
```

```
Object Class

Object o = new Object();
o.toString();
o.equals(o1);
```

```
Comparable Interface

c.compareTo(Comparable)
c is a Comparable object
```

```
String Class

String s = "Welcome";
String s = new String(char[]);
int length = s.length();
char ch = s.charAt(index);
int d = s.compareTo(s1);
boolean b = s.equals(s1);
boolean b = s.startsWith(s1);
boolean b = s.endsWith(s1);
String s1 = s.trim();
String s1 = s.toUpperCase();
String s1 = s.toLowerCase();
int index = s.indexOf(ch);
int index = s.lastIndexOf(ch);
String s1 = s.substring(ch);
String s1 = s.substring(i,j);
char[] chs = s.toCharArray();
Strings1 = s.replaceAll(regex,repl);
String[] tokens = s.split(regex);
```

```
ArrayList Class

ArrayList<E> list = new ArrayList<E>();
list.add(object);
list.add(index, object);
list.clear();
Object o = list.get(index);
boolean b = list.isEmpty();
boolean b = list.contains(object);
int i = list.size();
list.remove(index);
list.set(index, object);
int i = list.indexOf(object);
int i = list.lastIndexOf(object);
```

```
printf Method

System.out.printf("%b %c %d %f %e %s",
  true, 'A', 45, 45.5, 45.5, "Welcome");
System.out.printf("%-5d %10.2f %10.2e %8s",
  45, 45.5, 45.5, "Welcome");
```

Software – Data

Data Codes

Number Codes

❖Invented/Artificial

- ❑ Signaling
  - ➢ Smoke signals
  - ➢ Drums
  - ➢ Semaphores
- ❑ Communications
  - ➢ Morse code
  - ➢ Hollerith code (punch cards)
  - ➢ Paper tape codes
  - ➢ Encryption/cypher codes
  - ➢ **ASCII code** (also **EBCDIC**)

❖Natural

- ❑ DNA – Genetic code
  - ➢ Base-4 {A,C,G,T)
- ❑ Fibonacci sequence
  - ➢ Shell growth
  - ➢ Leaf growth

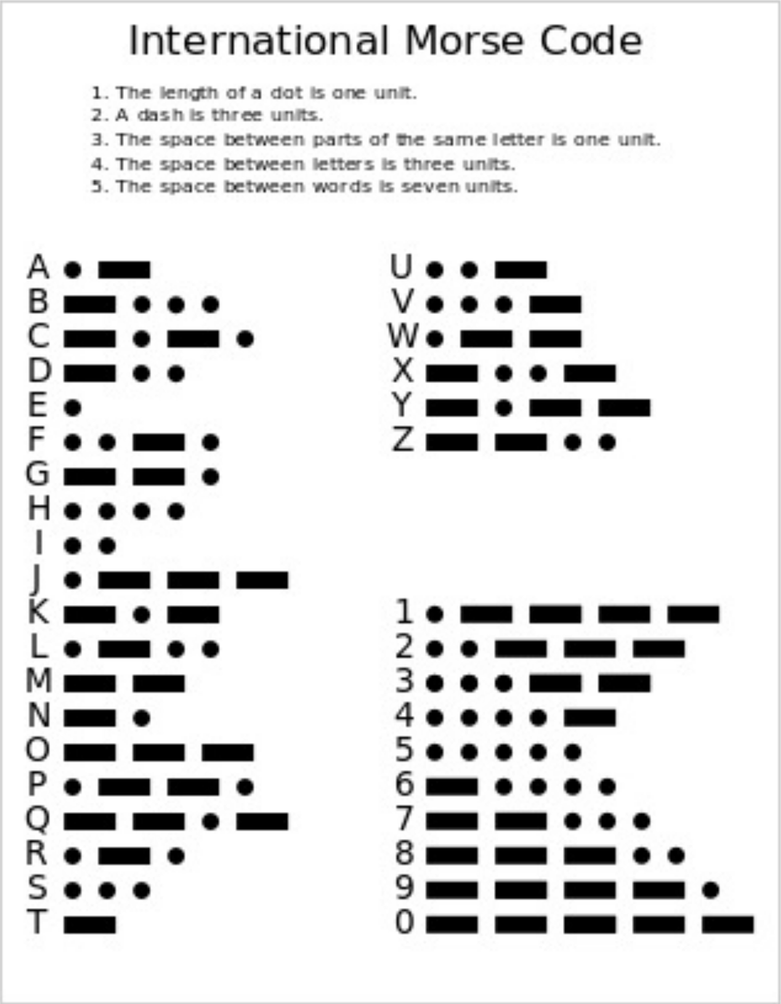
Telegraph: Morse Code

Base 2 = {dot, dash}

Each letter is a 1 to 4-bit character

1st Digital Code

1836-1844
by Samuel F.B. Morse et al.



International Morse Code

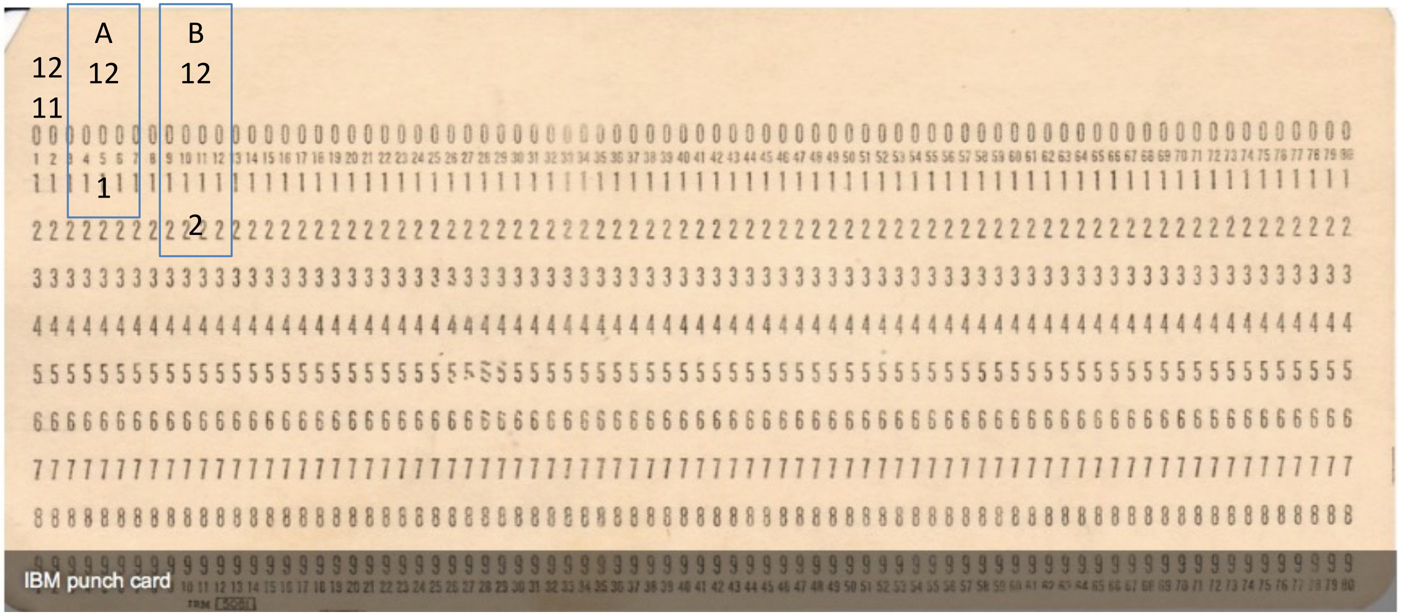
1. The length of a dot is one unit.
2. A dash is three units.
3. The space between parts of the same letter is one unit.
4. The space between letters is three units.
5. The space between words is seven units.

A	•▬	U	••▬
B	▬•••	V	•••▬
C	▬•▬•	W	•▬▬
D	▬••	X	▬••▬
E	•	Y	▬•▬▬
F	••▬•	Z	▬▬••
G	▬▬•		
H	••••		
I	••		
J	•▬▬▬		
K	▬•▬	1	•▬▬▬▬
L	•▬••	2	••▬▬▬
M	▬▬	3	•••▬▬
N	▬•	4	••••▬
O	▬▬▬	5	•••••
P	•▬▬•	6	▬••••
Q	▬▬•▬	7	▬▬•••
R	•▬•	8	▬▬▬••
S	•••	9	▬▬▬▬•
T	▬	0	▬▬▬▬▬

Chart of the Morse code letters and numerals.[1]

A typical "straight key". This U.S. model, known as the J-38, was manufactured in huge quantities during World War II, and remains in widespread use today. In a straight key, the signal is "on" when the knob is pressed, and "off" when it is released. Length and timing of the dots and dashes are entirely controlled by the telegraphist.

Punchcards



Invented by Herman Hollerith for 1890 census

ASCII Codes- Letters

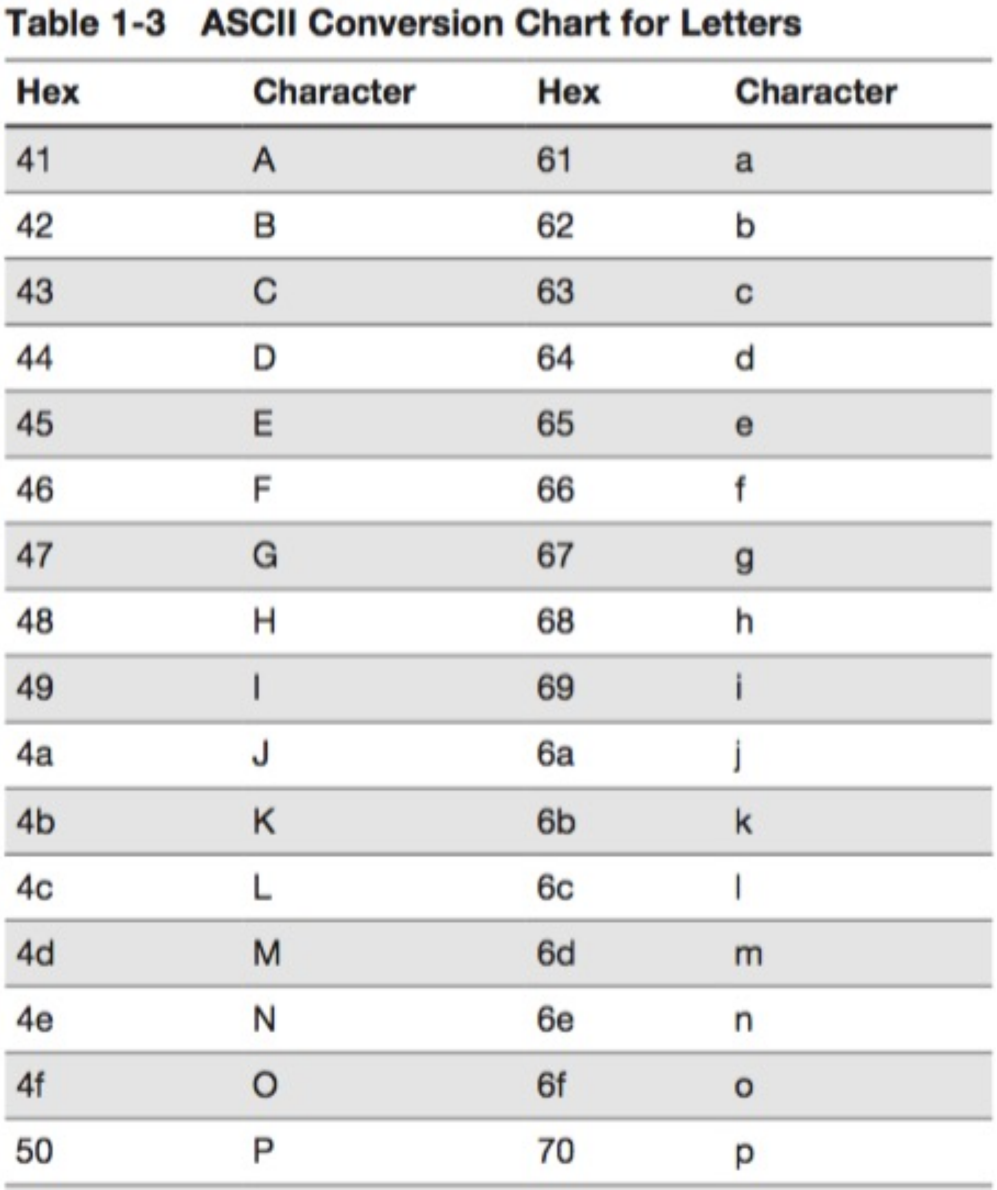
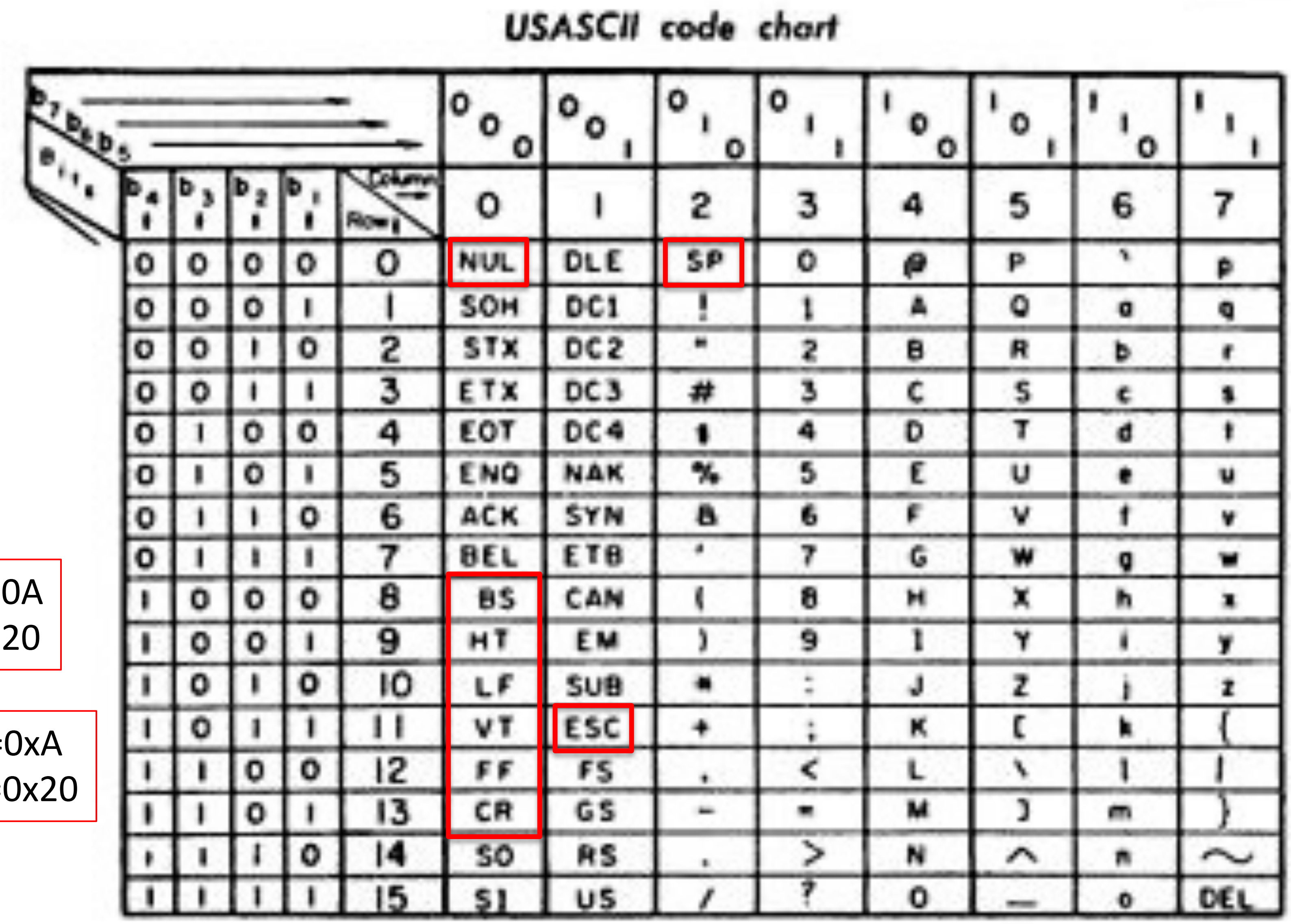


Table 1-3 ASCII Conversion Chart for Letters

Hex	Character	Hex	Character
41	A	61	a
42	B	62	b
43	C	63	c
44	D	64	d
45	E	65	e
46	F	66	f
47	G	67	g
48	H	68	h
49	I	69	i
4a	J	6a	j
4b	K	6b	k
4c	L	6c	l
4d	M	6d	m
4e	N	6e	n
4f	O	6f	o
50	P	70	p

1963

ASCII Codes- 7-bit



USASCII code chart

b7 b6 b5					0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b4	b3	b2	b1	Column / Row	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	|
1	1	0	1	13	CR	GS	-	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	_	o	DEL

\n=\u000A
sp=\u0020

char ch=0xA
char sp=0x20

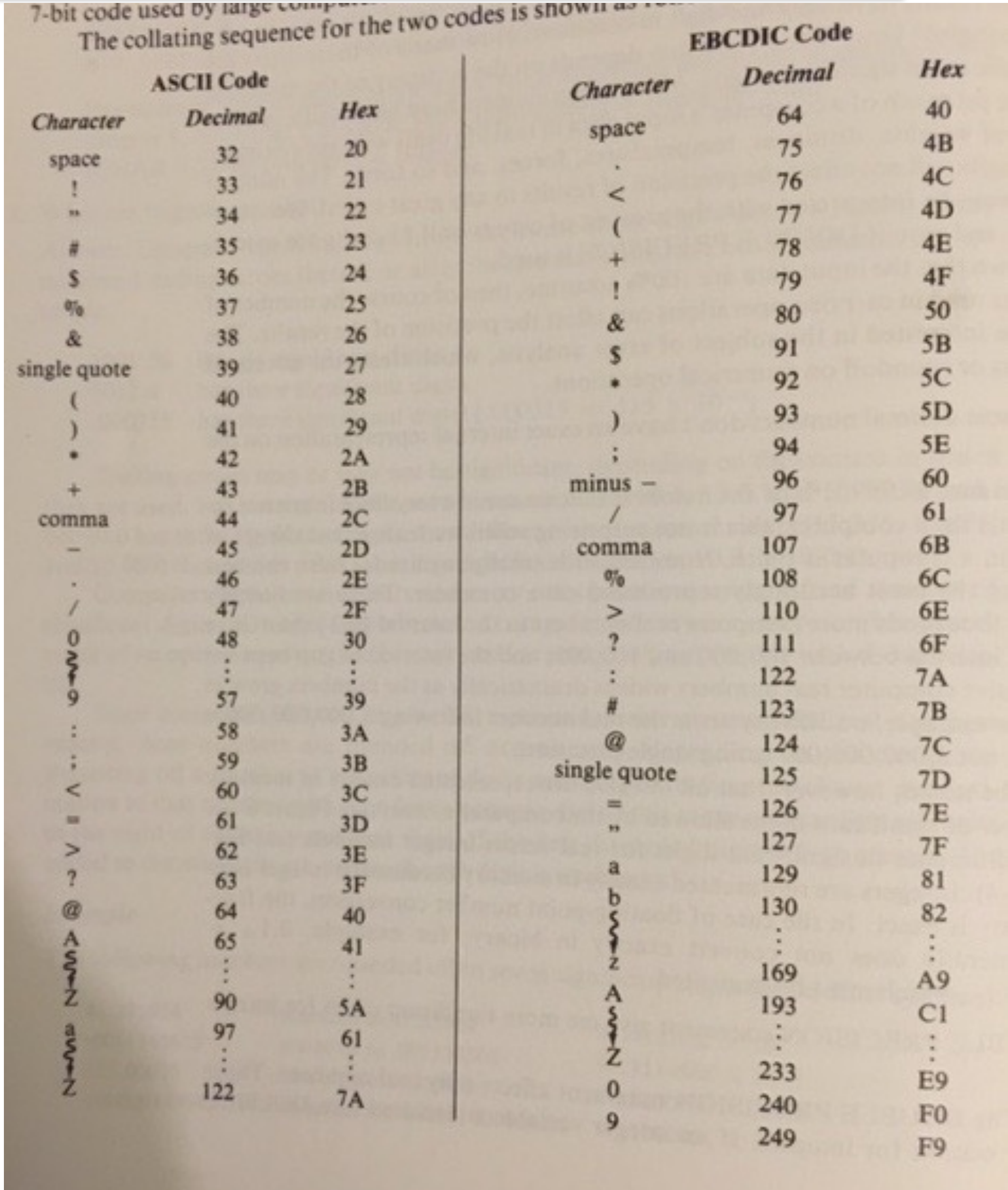
IANA encourages use of the name "US-ASCII" for Internet uses of ASCII

COMP110

IBM EBCDIC

EBCDIC

Extended **Binary Coded Decimal**



7-bit code used by large compu
The collating sequence for the two codes is shown as fo

ASCII Code

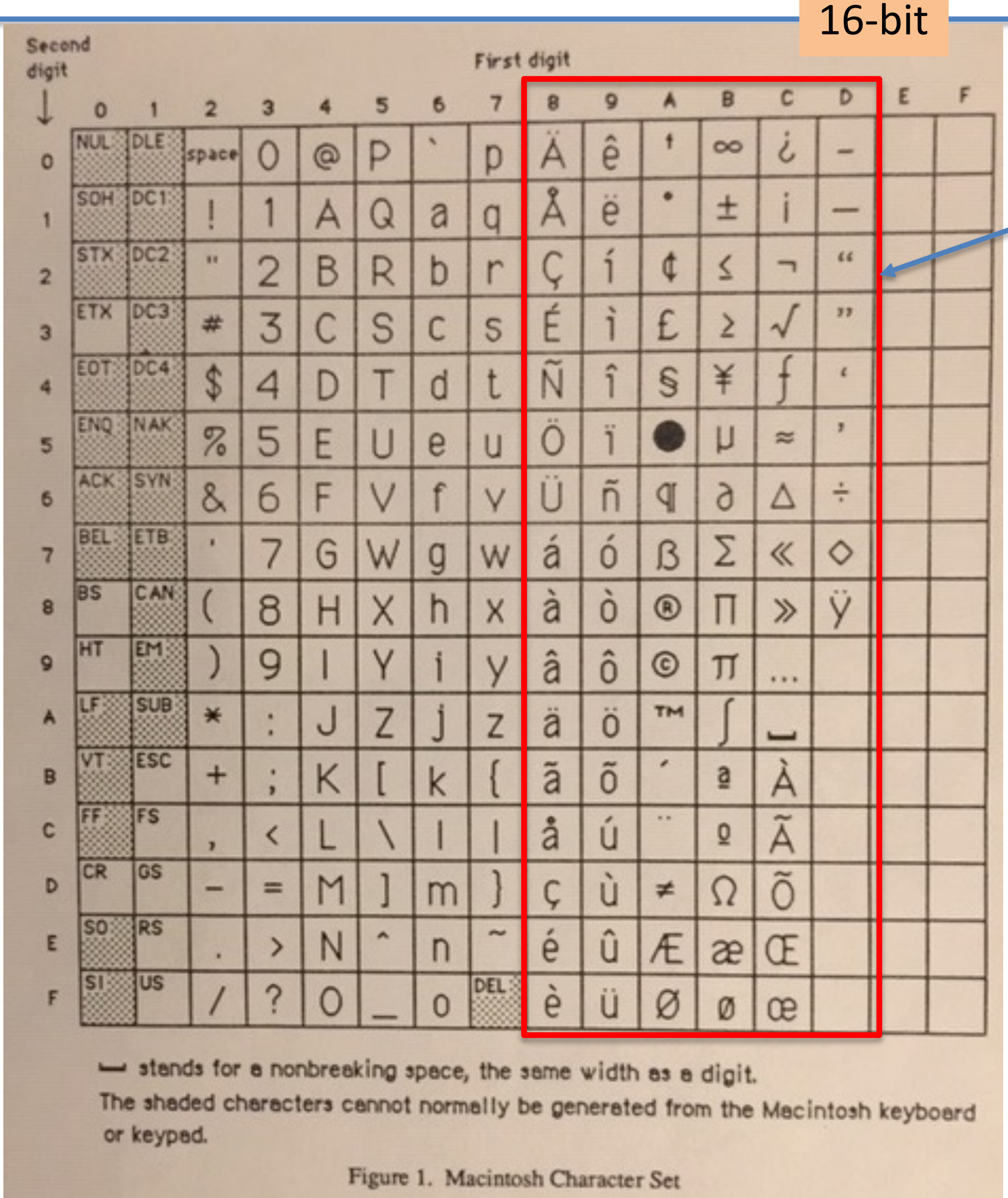
Character	*Decimal*	*Hex*
space	32	20
!	33	21
"	34	22
#	35	23
$	36	24
%	37	25
&	38	26
single quote	39	27
(	40	28
)	41	29
*	42	2A
+	43	2B
comma	44	2C
–	45	2D
.	46	2E
/	47	2F
0	48	30
⋮	⋮	⋮
9	57	39
:	58	3A
;	59	3B
<	60	3C
=	61	3D
>	62	3E
?	63	3F
@	64	40
A	65	41
⋮	⋮	⋮
Z	90	5A
a	97	61
⋮	⋮	⋮
z	122	7A

EBCDIC Code

Character	*Decimal*	*Hex*
space	64	40
.	75	4B
<	76	4C
(	77	4D
+	78	4E
!	79	4F
&	80	50
$	91	5B
*	92	5C
)	93	5D
;	94	5E
minus –	96	60
/	97	61
comma	107	6B
%	108	6C
>	110	6E
?	111	6F
:	122	7A
#	123	7B
@	124	7C
single quote	125	7D
=	126	7E
"	127	7F
a	129	81
b	130	82
⋮	⋮	⋮
z	169	A9
A	193	C1
⋮	⋮	⋮
Z	233	E9
0	240	F0
9	249	F9

Old Mac Char Codes

16-bit



First digit

Second digit ↓	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	DLE	space	0	@	P	`	p	Ä	ê	†	∞	¿	–		
1	SOH	DC1	!	1	A	Q	a	q	Å	ë	•	±	¡	—		
2	STX	DC2	"	2	B	R	b	r	Ç	í	¢	≤	¬	“		
3	ETX	DC3	#	3	C	S	c	s	É	ì	£	≥	√	”		
4	EOT	DC4	$	4	D	T	d	t	Ñ	î	§	¥	ƒ	‘		
5	ENQ	NAK	%	5	E	U	e	u	Ö	ï	●	µ	≈	’		
6	ACK	SYN	&	6	F	V	f	v	Ü	ñ	¶	∂	∆	÷		
7	BEL	ETB	'	7	G	W	g	w	á	ó	ß	Σ	«	◇		
8	BS	CAN	(	8	H	X	h	x	à	ò	®	Π	»	ÿ		
9	HT	EM	)	9	I	Y	i	y	â	ô	©	π	…			
A	LF	SUB	*	:	J	Z	j	z	ä	ö	™	∫	⎵			
B	VT	ESC	+	;	K	[	k	{	ã	õ	´	ª	À			
C	FF	FS	,	<	L	\	l	|	å	ú	¨	º	Ã			
D	CR	GS	–	=	M	]	m	}	ç	ù	≠	Ω	Õ			
E	SO	RS	.	>	N	^	n	~	é	û	Æ	æ	Œ			
F	SI	US	/	?	O	_	o	DEL	è	ü	Ø	ø	œ			

⎵ stands for a nonbreaking space, the same width as a digit.
The shaded characters cannot normally be generated from the Macintosh keyboard or keypad.

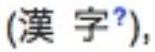
Figure 1. Macintosh Character Set

unique
special chars

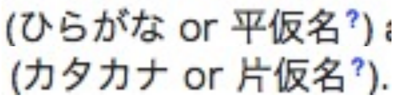
Unicode – 16-Bit

UTF-16

- ❖ 7 LSB are same codes as for **ASCII**
- ❖ 9 MSB add 2^{16}=65,536 - 128 new codes
- ❖ Japanese character sets

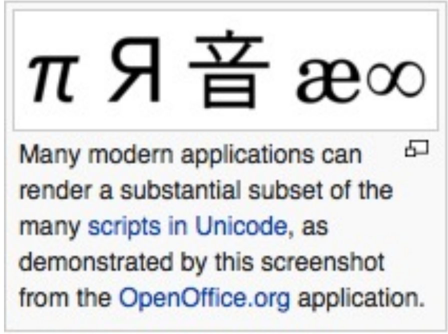

(漢字?),

  - ➢ *Kanji* uses same 5000 characters as base Chinese
  - ➢ *Hiragana/Katakana* uses

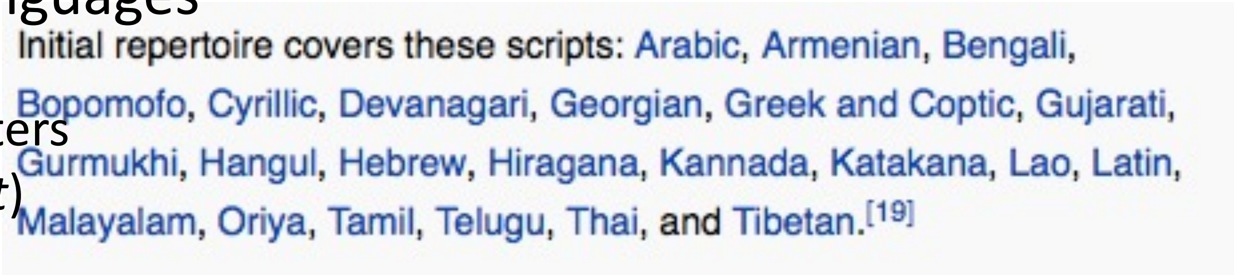

(ひらがな or 平仮名?) a
(カタカナ or 片仮名?).

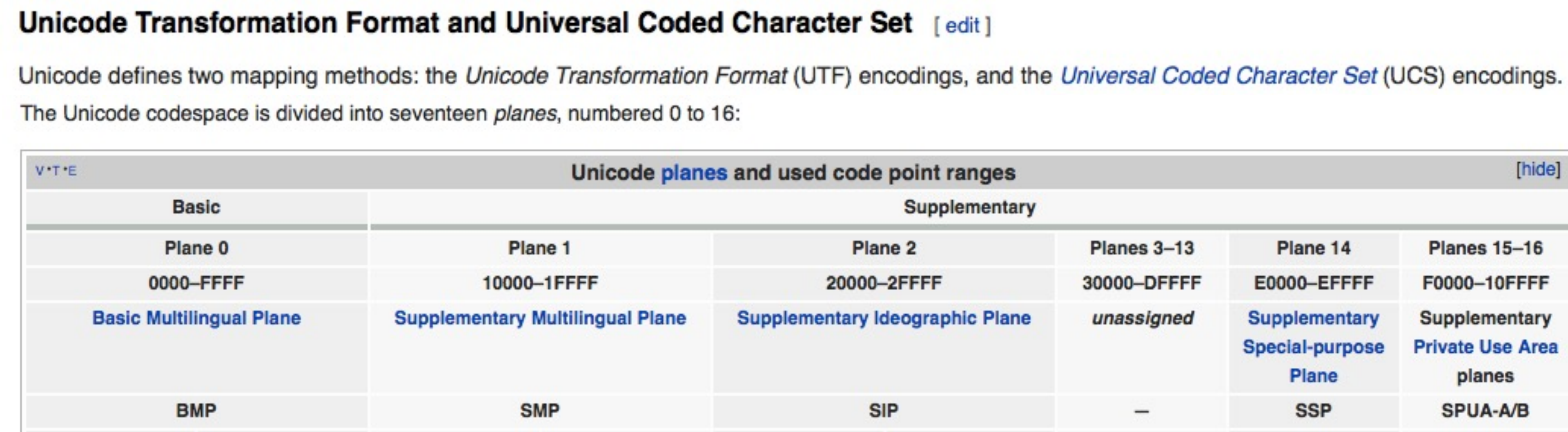
- ❖ Other foreign languages
  - ❑ alphabets
  - ❑ special characters

(vis-à-vis, *oomlaut*)


π Я 音 æ∞

Many modern applications can render a substantial subset of the many scripts in Unicode, as demonstrated by this screenshot from the OpenOffice.org application.


Initial repertoire covers these scripts: Arabic, Armenian, Bengali, Bopomofo, Cyrillic, Devanagari, Georgian, Greek and Coptic, Gujarati, Gurmukhi, Hangul, Hebrew, Hiragana, Kannada, Katakana, Lao, Latin, Malayalam, Oriya, Tamil, Telugu, Thai, and Tibetan.[19]

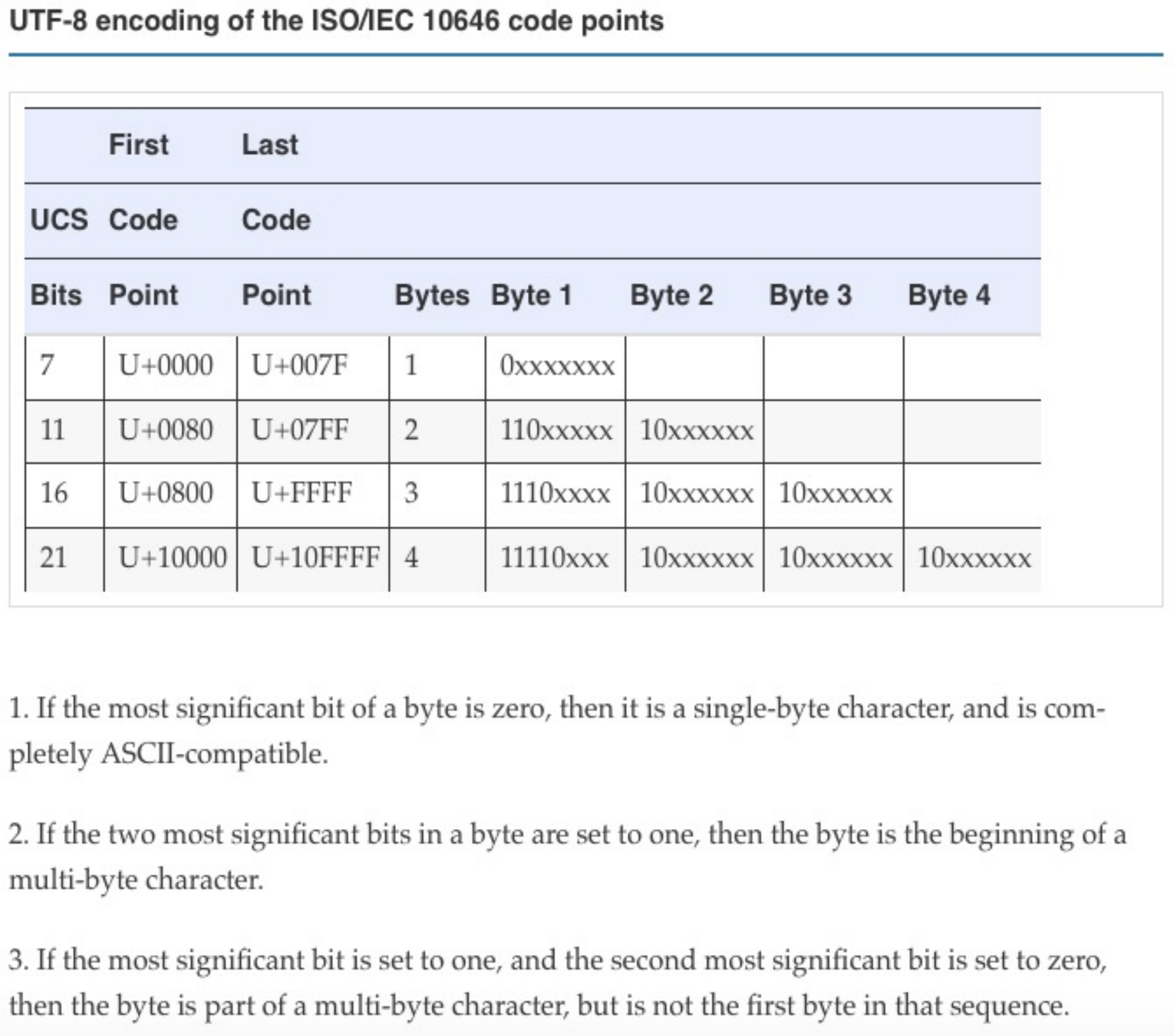

Unicode Transformation Format and Universal Coded Character Set [edit]

Unicode defines two mapping methods: the *Unicode Transformation Format* (UTF) encodings, and the *Universal Coded Character Set* (UCS) encodings. The Unicode codespace is divided into seventeen *planes*, numbered 0 to 16:

v·t·e **Unicode planes and used code point ranges** [hide]

Basic	Supplementary				
Plane 0	Plane 1	Plane 2	Planes 3–13	Plane 14	Planes 15–16
0000–FFFF	10000–1FFFF	20000–2FFFF	30000–DFFFF	E0000–EFFFF	F0000–10FFFF
Basic Multilingual Plane	Supplementary Multilingual Plane	Supplementary Ideographic Plane	*unassigned*	Supplementary Special-purpose Plane	Supplementary Private Use Area planes
BMP	SMP	SIP	–	SSP	SPUA-A/B

UTF-8 Variable Codes



UTF-8 encoding of the ISO/IEC 10646 code points

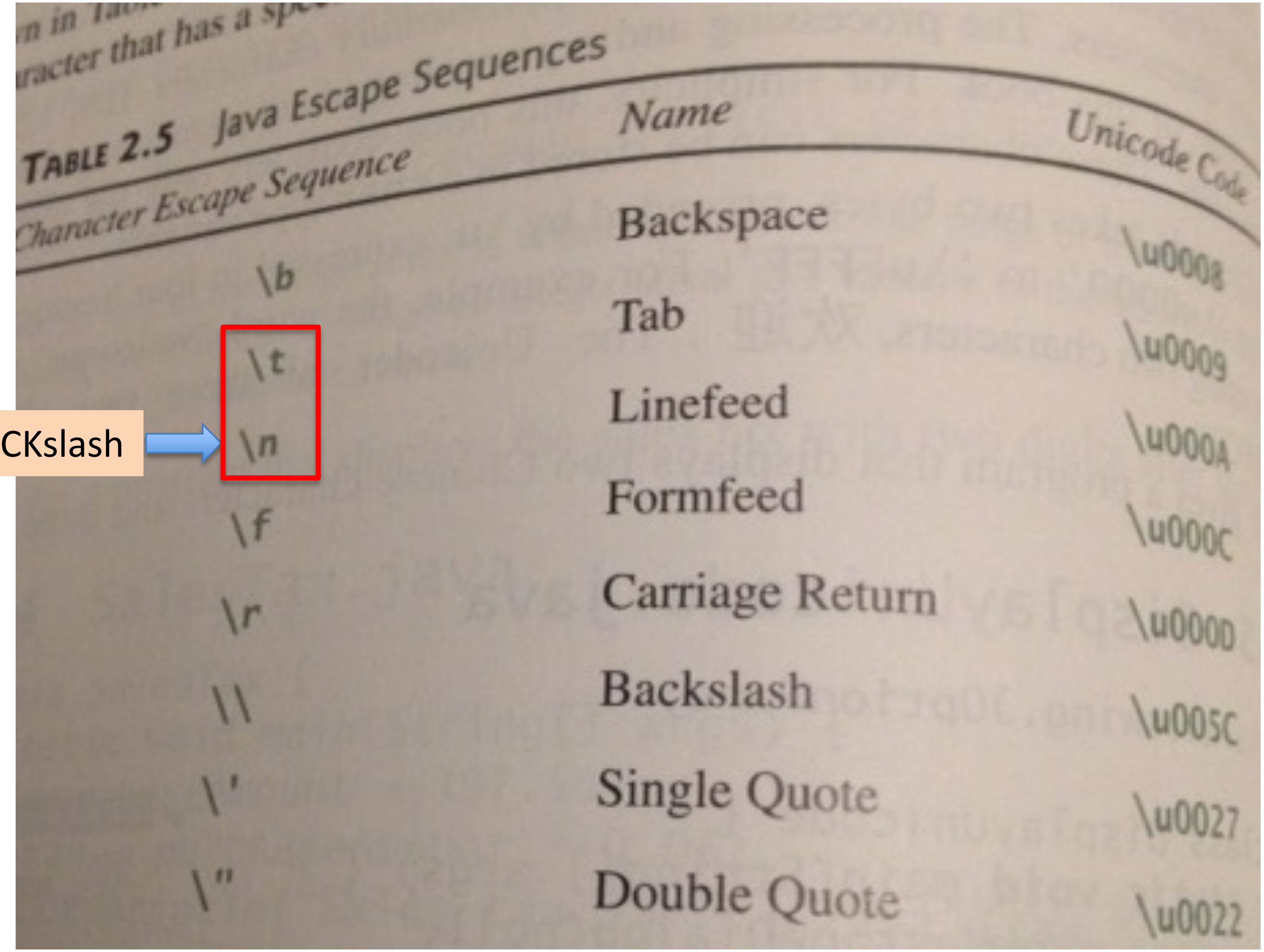
	First	Last					
UCS	Code	Code					
Bits	Point	Point	Bytes	Byte 1	Byte 2	Byte 3	Byte 4
7	U+0000	U+007F	1	0xxxxxxx			
11	U+0080	U+07FF	2	110xxxxx	10xxxxxx		
16	U+0800	U+FFFF	3	1110xxxx	10xxxxxx	10xxxxxx	
21	U+10000	U+10FFFF	4	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

1. If the most significant bit of a byte is zero, then it is a single-byte character, and is completely ASCII-compatible.

2. If the two most significant bits in a byte are set to one, then the byte is the beginning of a multi-byte character.

3. If the most significant bit is set to one, and the second most significant bit is set to zero, then the byte is part of a multi-byte character, but is not the first byte in that sequence.

Escape Characters



n in Tab
racter that has a spe

TABLE 2.5 *Java Escape Sequences*

Character Escape Sequence	*Name*	*Unicode Code*
\b	Backspace	\u0008
\t	Tab	\u0009
\n	Linefeed	\u000A
\f	Formfeed	\u000C
\r	Carriage Return	\u000D
\\	Backslash	\u005C
\'	Single Quote	\u0027
\"	Double Quote	\u0022

- BACKslash

Chapter 2

- Identifiers
- Keywords/RESERVED Words
- Number Representation
- Variables/Constants
- Data types
- Logic
- I/O
- Strings
- Date/Time

Identifiers

Identifier = Name

where

- each character is in {a-z, A-Z, 0-9, _, $}
- 1st character is in {a-z, A-Z, _, $} (no digits)
- not a *reserved word*
- not in {true, false, null} (*literals*)
- any length

Symbol Table

Symbol	address
xyz	0x1FE5 87C4
$X2	0x1FE5 87C8
_K3$	0x1FE5 87CD

examples

- any$
- anyStringXY
- any_string
- any_num012
- A1_xyz
- $special
- _special

Naming Conventions

examples

- anyStringWillDo
- JOptionsPane
- _special
- $DEBUG
- old style (FORTRAN):
  - {I .. N} integers
  - {X .. Z} float

Variables vs Constants

❖Variables

- Assign values any time, many times
- *Mutable*
- Naming: lowercase (“camelCase”)
- Examples
  - int x = 1; //initialized
  - x = y + 13;
  - x = z – 2;

❖Constants

- Assign value only ONE time
- *Immutable*
- Naming: UPPER_CASE
- Examples
  - final double PI = 3.14159;
  - final int MIN = 3;
  - final boolean $DEBUG = true;
  - final boolean $FLAG = false;

Keywords/Reserved Words

JAVA

Appendix A

- abstract*
- assert
- boolean
- break
- byte
- case
- catch
- char
- class
- const
- continue
- default*
- do
- double
- else
- enum
- extends
- for
- final*
- finally
- float
- goto
- if
- implements
- import
- instanceof
- int
- interface
- long
- native*
- new
- package
- private*
- protected*
- public*
- return
- short
- static*
- strictfp*
- super
- switch
- synchronized
- this
- throw
- throws
- transient
- try
- void
- volatile
- while

true, false, null are reserved *literals*

*class modifiers

Number Systems

POSITIONAL REPRESENTATION

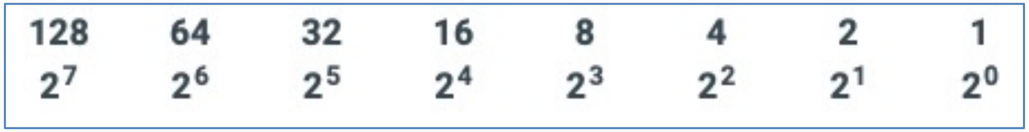
$$N = \mathrm{SUM}_i\,(n_i * R^i)$$

Notes:

1. R = Radix
2. i = -infinity to +infinity
3. i >= 0 for integers

binary

radix = **2**: <-- $2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$ with n = {0, 1}



128	64	32	16	8	4	2	1
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Byte = {0..255} $2^8 = 256$

decimal

radix = **10**: <-- $10^5\ 10^4\ 10^3\ 10^2\ 10^1\ 10^0$ with n = {0 .. 9}

hexadecimal

radix = **16**: <-- $16^5\ 16^4\ 16^3\ 16^2\ 16^1\ 16^0$ with n = {0 .. 9, A, B, C, D, E, F}

binary *hexadecimal*

0)	0		8)	1000	
1)	1		9)	1001	
2)	10		10)	1010	A
3)	11		11)	1011	B
4)	100	+8	12)	1100	C
5)	101	-->	13)	1101	D
6)	110		14)	1110	E
7)	111		15)	1111	F

Numbers

- Integers

123456789

int x = 123.567
assigns 123 to x

- Java ***truncates*** Integers
  - to *Round* add 0.5

- Fixed-point*

12345.12345

Pi = 3.14159

*NOT a supported ***type*** in any HLL; must use Floating-point

- Floating-point

1.234512345**e+4**

Pi = 3.14159**e0**

- Java ***rounds*** floats

Accuracy vs. *Precision*

no. digits correct

no. digits stored/displayed

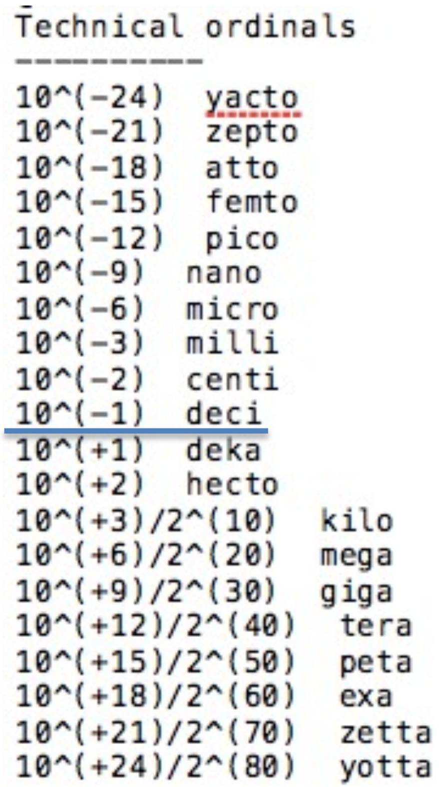
CSUN
CALIFORNIA
STATE UNIVERSITY
NORTHRIDGE

COMP110

Ordinals

DR JEFF SOFTWARE
INDIE APP DEVELOPER

Powers of 2 <> 10: 10:3



```
Technical ordinals
----------
10^(-24)   yacto
10^(-21)   zepto
10^(-18)   atto
10^(-15)   femto
10^(-12)   pico
10^(-9)   nano
10^(-6)   micro
10^(-3)   milli
10^(-2)   centi
10^(-1)   deci
10^(+1)   deka
10^(+2)   hecto
10^(+3)/2^(10)   kilo
10^(+6)/2^(20)   mega
10^(+9)/2^(30)   giga
10^(+12)/2^(40)   tera
10^(+15)/2^(50)   peta
10^(+18)/2^(60)   exa
10^(+21)/2^(70)   zetta
10^(+24)/2^(80)   yotta
```

Gazillions

10^(+6) million
10^(+9) billion
10^(+12) trillion
10^(+15) quadrillion
10^(+18) quintillion
10^(+21) sexillion
10^(+24) septillion
10^(+27) octillion
10^(+30) nonillion
10^(+33) decillion
10^(+36) undecillion
10^(+39) duodecillion
10^(+42) tredecillion
10^(+45) quattuordecillion
10^(+48) quindecillion
10^(+51) sexdecillion
10^(+54) septendecillion
10^(+57) octodecillion
10^(+60) novemdecillion
10^(+63) vigintillion
10^(+100) googol
10^(+303) centillion
10^(10^(+100))
googolplex

Ordinal	Power of 2	Power of 10	Actual
1K	2^{10}	10^{3}	1024
1M	2^{20}	10^{6}	1,048,576
1G	2^{30}	10^{9}	1.074×10^{9}
1T	2^{40}	10^{12}	1.0995×10^{12}

Name	2^n	M/G	Actual
byte	2^{8}	--	256
word	2^{16}	64K	65,536
integer	2^{32}	4B	4.3×10^{9}
double	2^{64}	16 Q	1.84×10^{19}
IPv6	2^{128}	340 uD	3.4×10^{38}

Declaring Data Types

- Standard Data Types
  - □ Numerics

    ```
    byte xb;
    short s, p;
    int n, m;
    long nn, mm;
    ```

  - □ Non-Numerics

    ```
    char xch;
    boolean flag1, flag2;
    ```

➢ **String** is a *Class* not a *Type*

In VB

```
‘**standard variables
   Dim i, j, k, l, m, n As Byte
   Dim const1 As Byte = 1
   Dim u, v, w, x, y, z As Int16
   Dim ss, tt, uu, vv, ww, xx, yy, zz As Single
   Dim wstr, xstr, ystr, zstr, xxstr, alertst, srchstr, matchstr As String
   Dim err1 As Boolean = False, err2 As Boolean = False
   Dim vobj, wobj, xobj, yobj, zobj As Object
```

Data Types – C, Java

C

❖Integers (16-bit)

- unsigned int (0 to 65,535)
- signed int (-32,768 to +32,767)

❖Char string/byte (8-bit)

ASCII-8
UNICODE-16

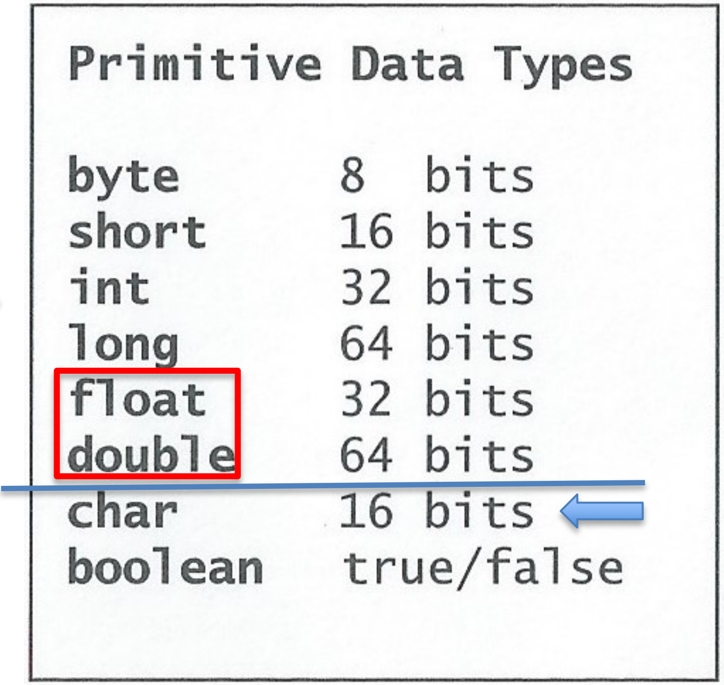
- unsigned char (0 to 255)
- signed char (-128 to +127)

❖Floating Point ("Real") (32/64-bit)

- Float (32-bit single precision: +-N x 10^64)
- Double (64-bit double precision: +-N x 10^256)

1.123456789 E+64

Java

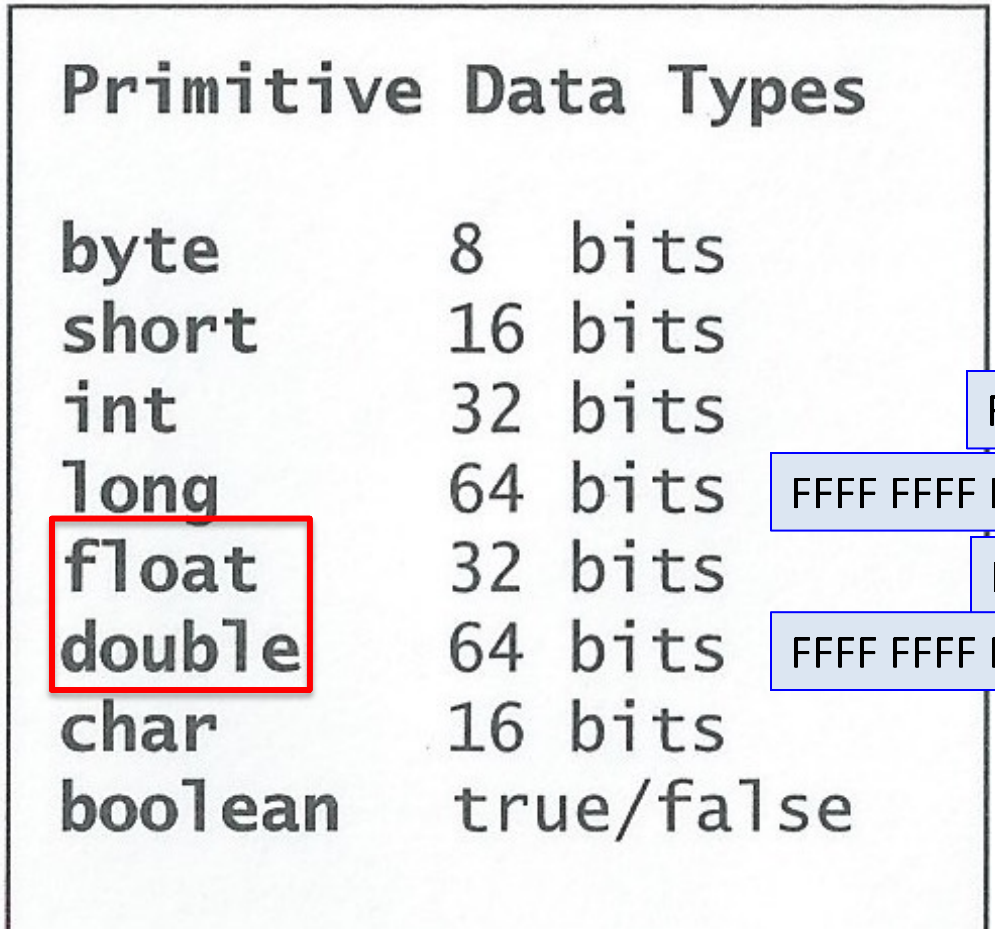


Primitive Data Types

byte	8 bits
short	16 bits
int	32 bits
long	64 bits
float	32 bits
double	64 bits
char	16 bits
boolean	true/false

Data Types – Java

Java



Primitive Data Types

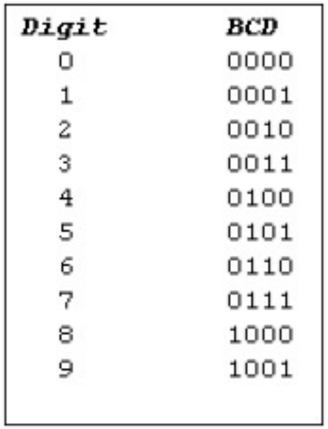
❖ all signed

byte	8 bits	FF	-128 to +127
short	16 bits	FFFF	-32,768 to +32,767
int	32 bits	FFFF FFFF	-2,147,483,648 to +2.15B
long	64 bits	FFFF FFFF FFFF FFFF	-1.845x10^{19} to +X(-1)
float	32 bits	FFFF FFFF	0 to +-3.4x10^{38}
double	64 bits	FFFF FFFF FFFF FFFF	0 to +-1.8x10^{308}
char	16 bits	FFFF	0 to 65,535
boolean	true/false	FF	0 or -1 {00, FF}

❖ unsigned

Data Types– Assembly Level

- Hex: 0x5EC46A or 5EC46AH or just 5EC46A
- Binary: B ’001100101001’
- Decimal: D ’123456’ or just 123456
- BCD: packed = [D,D] un-packed = [0,D] [0,D]
- ASCII: A ’TeXt234’
- Signed numbers (16-bit example)
  - Unsigned (addresses) → 0 to 65,535 MMMMM
  - Sign & Magnitude → -32,767 to +32,767 [+0, -0] S MMMM
  - Two’s Complement → -32,768 to +32,767 [0 only] S [2^{16}-] MMMM
  - Sign extension SSSSSSSSS [2^N-] MMMM



Digit	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

CSUN
CALIFORNIA
STATE UNIVERSITY
NORTHRIDGE

COMP110

Floating Point

DR JEFF
SOFTWARE
INDIE APP DEVELOPER

ALL *REAL* NUMBERS

Java

Integers:
Div by 0 → ArithmeticException

1.123456 x10^nnn

mantissa   characteristic (exponent)

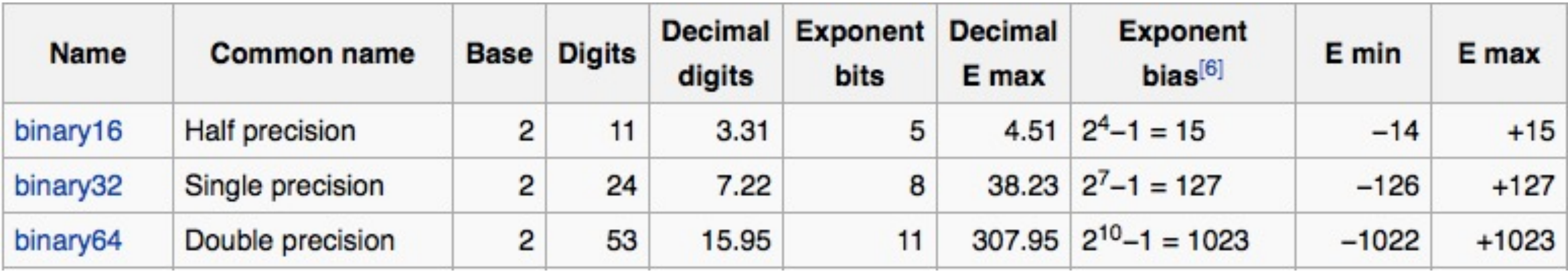
SPECIAL VALUES

- ❑ POSITIVE_INFINITY (=X/0)
- ❑ NEGATIVE_INFINITY (=-X/0)
- ❑ **NaN** (=0/0 and many more)

RANGES	Large (>>1)	Small (<<1)
SINGLE	3.403 E**+38** (2^{128})	1.4 E**-45**
DOUBLE	1.798 E**+308**	4.9 E**-324**

Positive & Negative

IEEE 754



Name	Common name	Base	Digits	Decimal digits	Exponent bits	Decimal E max	Exponent bias[6]	E min	E max
binary16	Half precision	2	11	3.31	5	4.51	$2^4-1 = 15$	−14	+15
binary32	Single precision	2	24	7.22	8	38.23	$2^7-1 = 127$	−126	+127
binary64	Double precision	2	53	15.95	11	307.95	$2^{10}-1 = 1023$	−1022	+1023

CSUN CALIFORNIA STATE UNIVERSITY NORTHRIDGE

COMP110

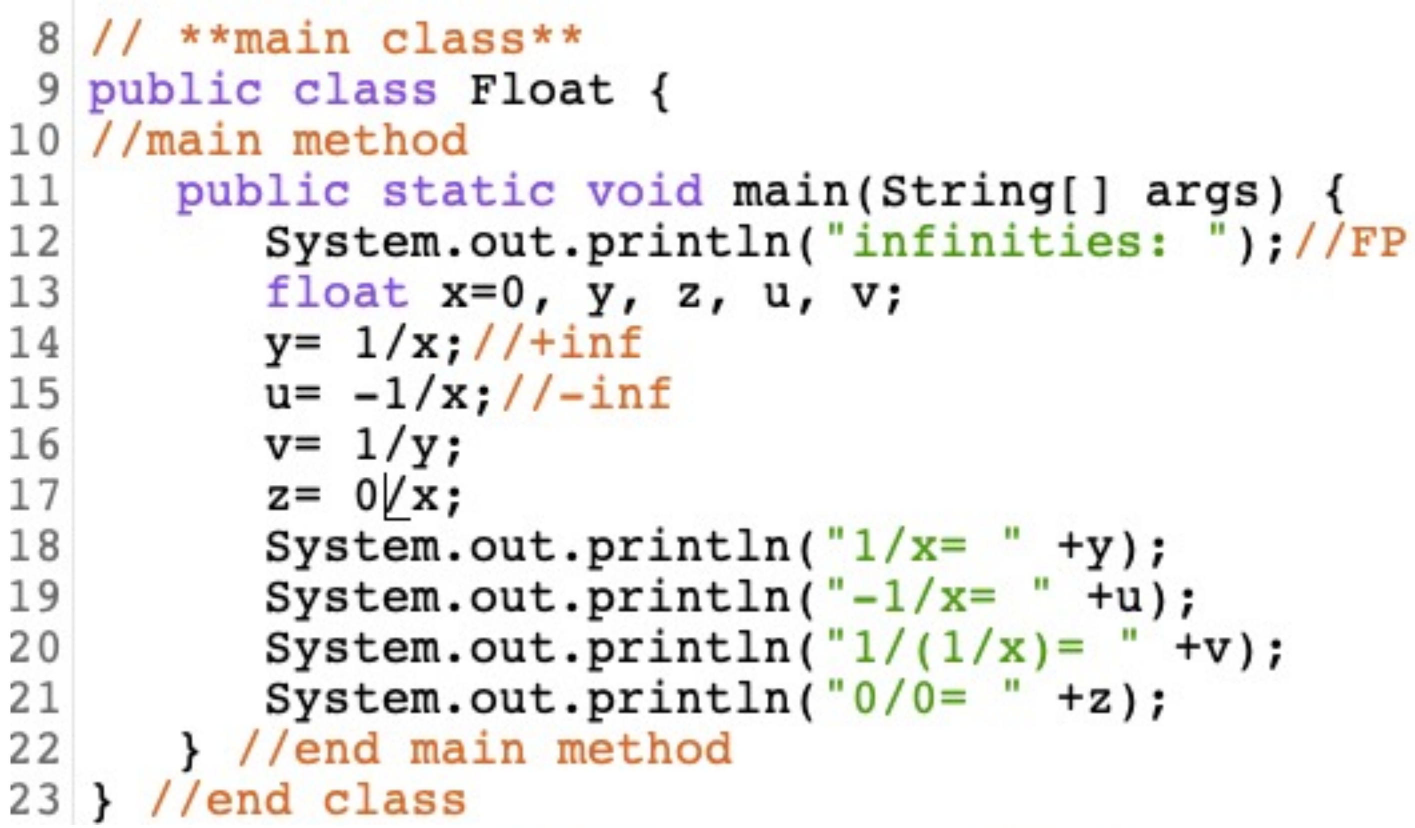
Floating Point

DR JEFF SOFTWARE INDIE APP DEVELOPER

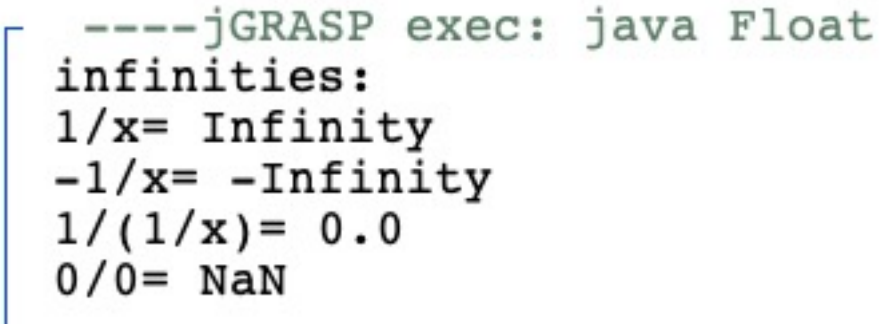
ALL *REAL* NUMBERS

IEEE 754

Java



```
// **main class**
public class Float {
//main method
    public static void main(String[] args) {
        System.out.println("infinities: ");//FP
        float x=0, y, z, u, v;
        y= 1/x;//+inf
        u= -1/x;//-inf
        v= 1/y;
        z= 0/x;
        System.out.println("1/x= " +y);
        System.out.println("-1/x= " +u);
        System.out.println("1/(1/x)= " +v);
        System.out.println("0/0= " +z);
    } //end main method
} //end class
```



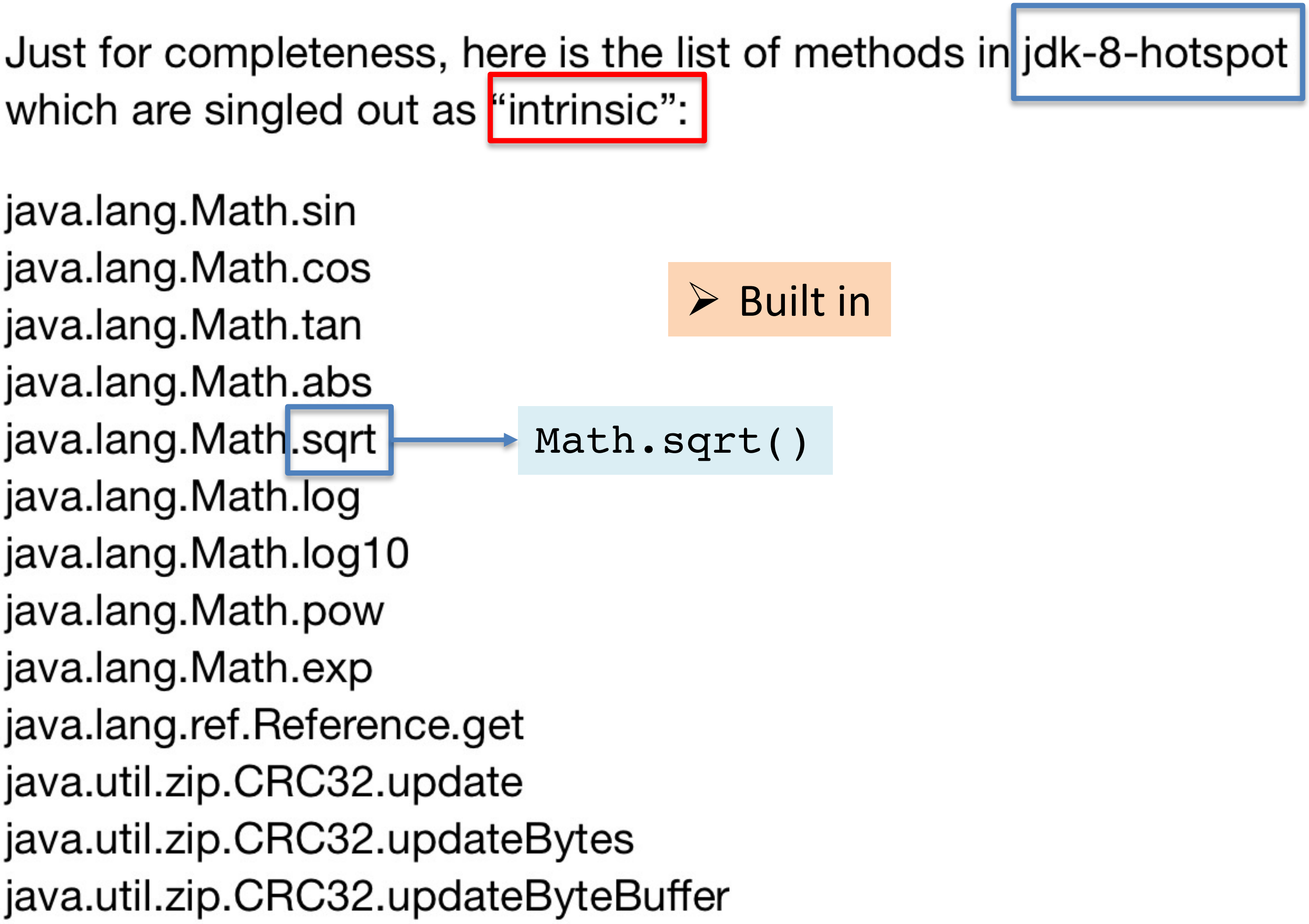
```
 ----jGRASP exec: java Float
infinities:
1/x= Infinity
-1/x= -Infinity
1/(1/x)= 0.0
0/0= NaN
```

SPECIAL VALUES

- ❑ POSITIVE_INFINITY (=X/0)
- ❑ NEGATIVE_INFINITY (=-X/0)
- ❑ **NaN** (=0/0 and many more)

Floating Point Math Functions

Transcendentals as Intrinsics



Just for completeness, here is the list of methods in jdk-8-hotspot which are singled out as “intrinsic”:

java.lang.Math.sin
java.lang.Math.cos
java.lang.Math.tan
java.lang.Math.abs
java.lang.Math.sqrt
java.lang.Math.log
java.lang.Math.log10
java.lang.Math.pow
java.lang.Math.exp
java.lang.ref.Reference.get
java.util.zip.CRC32.update
java.util.zip.CRC32.updateBytes
java.util.zip.CRC32.updateByteBuffer

- Built in

`Math.sqrt()`

Literals – Numeric, String

Java

- Decimal
  - int i = 0;
  - long k = 123456789
  - double xx = 3.141592653
  - float x = 1.23456**e-21**
- Hexadecimal
  - int hx = **0x**1F8C
  - long hxyz = **0x**1F8C5D1B
- String
  - char cx = ‘a’
  - **String** sx = “anytext”

❖ String is a ***class*** with ***methods*** (built-in)

Operations – Numeric

Java

Symbol	Operation	Example	Result
+	addition	34 + 1	35
-	subtraction	34.0 – 0.1	33.9
*	multiplication	300 * 30	9000
/	division	1.0 / 2.0	0.5
%	remainder (mod residue)	20 % 3	2

1 / 2 → 0

Operations – Shorthand

Java

Symbol	Operation	Example	Result
+=	addition	x += 8	x ← x + 8
-=	subtraction	x -= 8	x ← x - 8
***=**	multiplication	x *= 8	x ← x * 8
/=	division	x /= 8	x ← x / 8
%=	remainder (mod residue)	x %= 8	x ← x % 8

every programming language uses these

Operations – Incr/Decr

Java

Symbol	Operation	Example	Result

++var	PRE incr	y = (++x) + 2	x += 1
			y ← x + 2

var++	POST incr	y = (x++) + 2	y ← x + 2
			x += 1

--var	PRE decr	y = (--x) + 2	x -= 1
			y ← x + 2

var--	POST decr	y = (x--) + 2	y ← x + 2
			x -= 1

most programming languages uses these

Section

Type Conversion

Type Conversions

Numeric

- Java *truncates* Integers
  - to *Round* add 0.5

5/9 → 0
5/9 + 0.5 → 1.0555 → 1

- Expressions
  - mixed types resolve to highest precision operand
    - Double > Float > Long > Int > Short > Byte

5.0/9 → 0.5555
5/9 → 0

- Casting
  - ❑ Implicit

int i = 1.23 → 1
int i = 1.23e+12 → *error*
float x = 1.23 → 1.23
byte x = 128 → *error*

  - ❑ Explicit

float f = 1.23 → 1.23e0
int i = f + 1.23 → *error?*
int i = (int) f → 1
int i = (int) 1.23e+12 → *error*
long i = (int) 1.23e+12 → 1,230,000,000,000

can use:
123f
123d

- Strings – String.method (shown later)

Non-Numeric

Data Type Conversions

Mixed type expressions

X (O) Y

A (O) B | (O) C | (O) D | (O) E

Left to → Right

➢ Double > Float > Long > Int > Short > Byte

- ❖ **Numerics**: UP convert
- ❖ **String**: convert Numeric to String → `“” + x`
  - ➢ String to Numeric → `Integer.parseInt(str)`
- ❖ **Char**: treat as Numeric (use ASCII code/ Unicode)

CSUN
CALIFORNIA
STATE UNIVERSITY
NORTHRIDGE

COMP110

Operation Precedence

DR JEFF
SOFTWARE
INDIE APP DEVELOPER

zyBook

Expression Evaluation

arithmetic

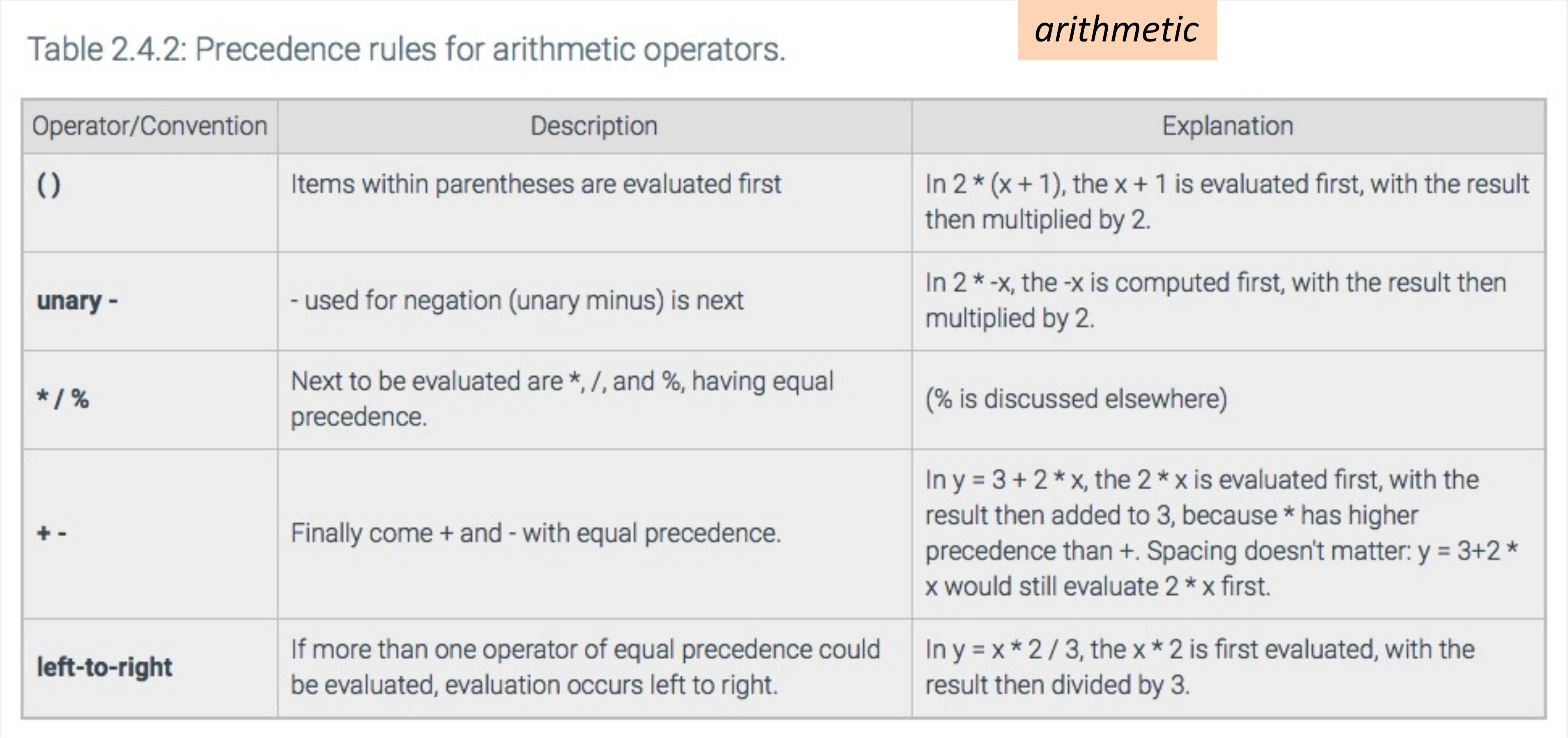


Table 2.4.2: Precedence rules for arithmetic operators.

Operator/Convention	Description	Explanation
( )	Items within parentheses are evaluated first	In 2 * (x + 1), the x + 1 is evaluated first, with the result then multiplied by 2.
unary -	- used for negation (unary minus) is next	In 2 * -x, the -x is computed first, with the result then multiplied by 2.
*** / %**	Next to be evaluated are *, /, and %, having equal precedence.	(% is discussed elsewhere)
+ -	Finally come + and - with equal precedence.	In y = 3 + 2 * x, the 2 * x is evaluated first, with the result then added to 3, because * has higher precedence than +. Spacing doesn't matter: y = 3+2 * x would still evaluate 2 * x first.
left-to-right	If more than one operator of equal precedence could be evaluated, evaluation occurs left to right.	In y = x * 2 / 3, the x * 2 is first evaluated, with the result then divided by 3.

Operation Precedence

Liang

Expression Evaluation

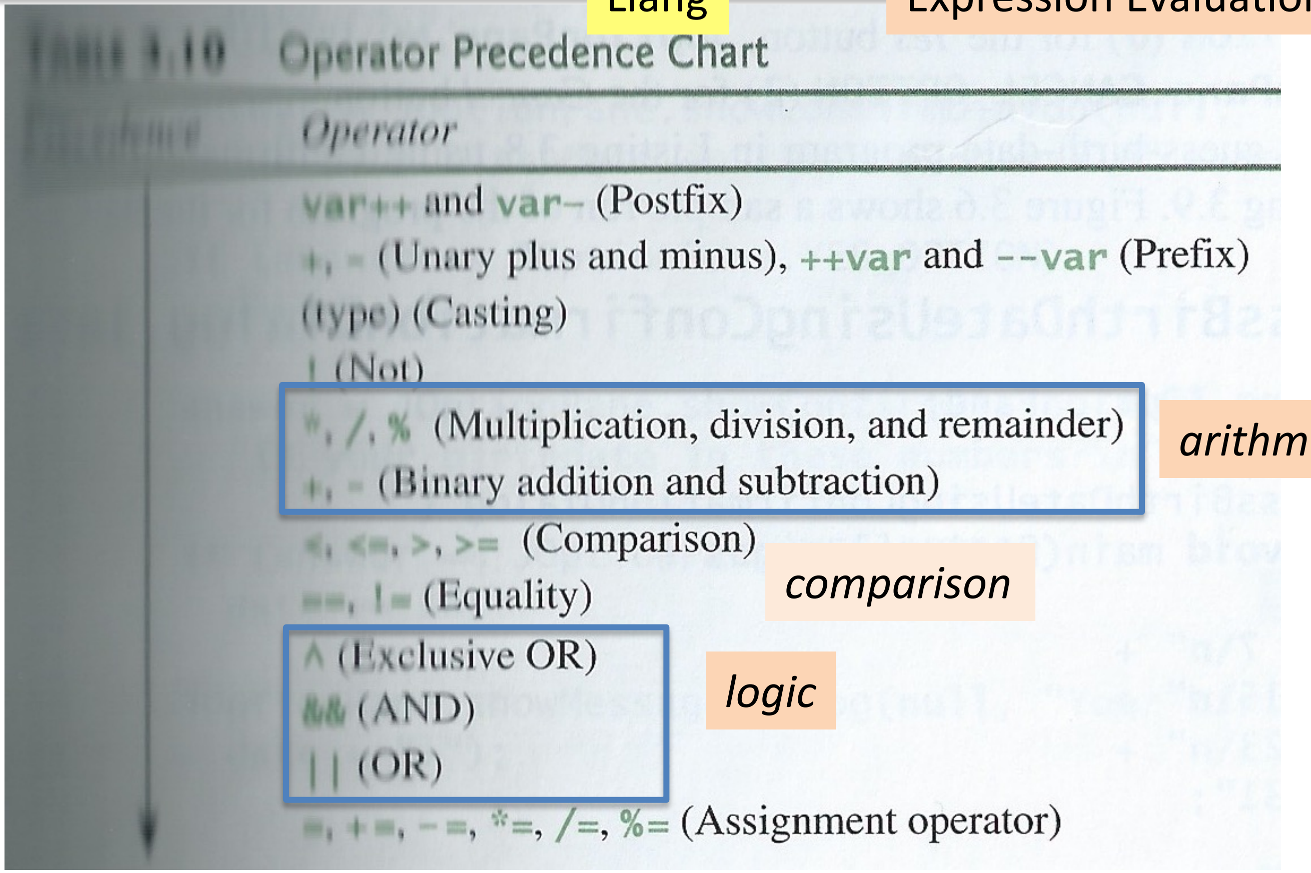


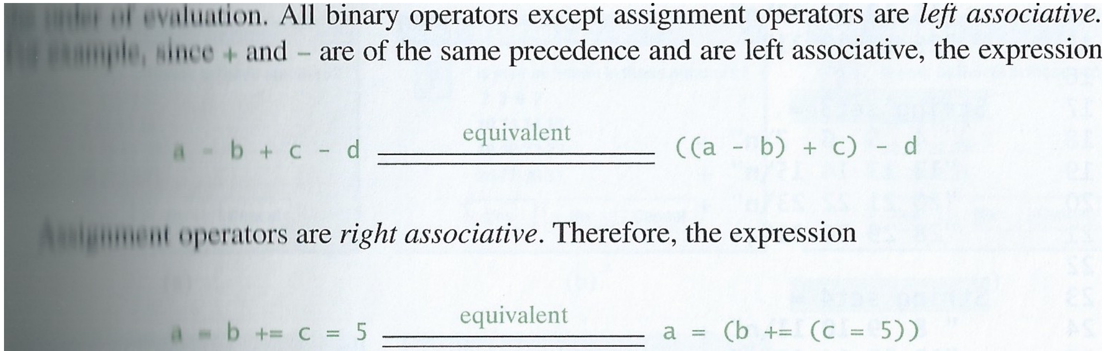
TABLE 3.10 Operator Precedence Chart

Precedence	Operator
	`var++` and `var--` (Postfix)
	`+`, `-` (Unary plus and minus), `++var` and `--var` (Prefix)
	(type) (Casting)
	`!` (Not)
	`*`, `/`, `%` (Multiplication, division, and remainder)
	`+`, `-` (Binary addition and subtraction)
	`<`, `<=`, `>`, `>=` (Comparison)
	`==`, `!=` (Equality)
	`^` (Exclusive OR)
	`&&` (AND)
	`||` (OR)
	`=`, `+=`, `-=`, `*=`, `/=`, `%=` (Assignment operator)

arithmetic

comparison

logic



order of evaluation. All binary operators except assignment operators are *left associative*. For example, since + and – are of the same precedence and are left associative, the expression

```
a - b + c - d  equivalent  ((a - b) + c) - d
```

Assignment operators are *right associative*. Therefore, the expression

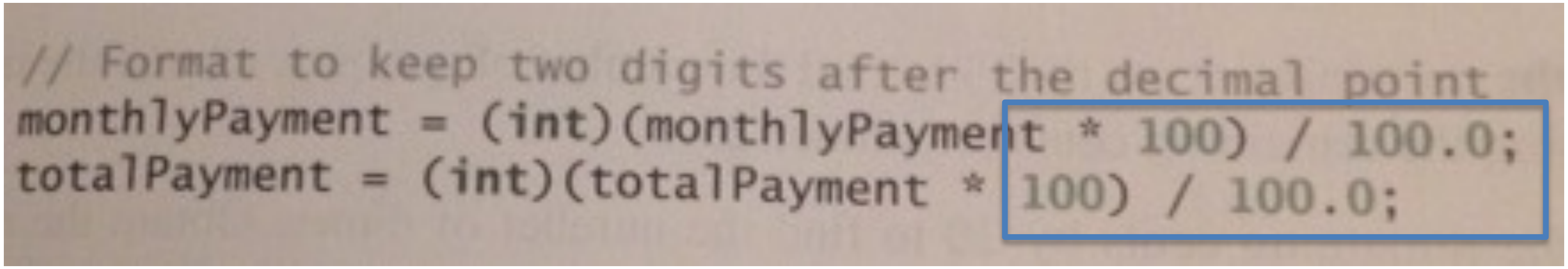
```
a = b += c = 5  equivalent  a = (b += (c = 5))
```

Section

Formatted Output

Formatted Precision

123.45



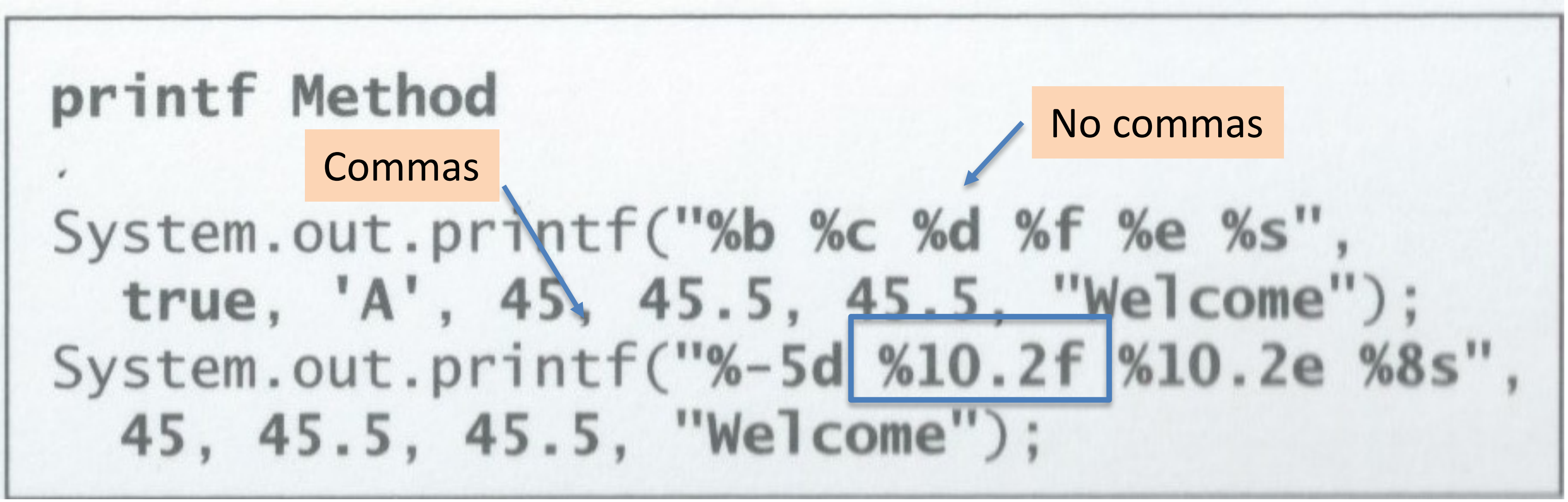
```
// Format to keep two digits after the decimal point
monthlyPayment = (int)(monthlyPayment * 100) / 100.0;
totalPayment = (int)(totalPayment * 100) / 100.0;
```

(int) (x * 100) / 100**.0**

123.456

(int) (x * 1000) / 1000**.0**

Formatted Output



printf Method

```
System.out.printf("%b %c %d %f %e %s",
   true, 'A', 45, 45.5, 45.5, "Welcome");
System.out.printf("%-5d %10.2f %10.2e %8s",
   45, 45.5, 45.5, "Welcome");
```

Commas

No commas

%n.m f → can be placed anywhere in a **string**

❖ format specifiers (%) are related to arguments left to right

```
String fstr = “temperature = %10.2f”;
System.out.printf (fstr, celcius);
```

Chapter 3

- Logic: Operators & Expressions
- *Selections*
  - ❑ IF-THEN-ELSE
  - ❑ Switch-Case

CSUN
CALIFORNIA
STATE UNIVERSITY
NORTHRIDGE
COMP110

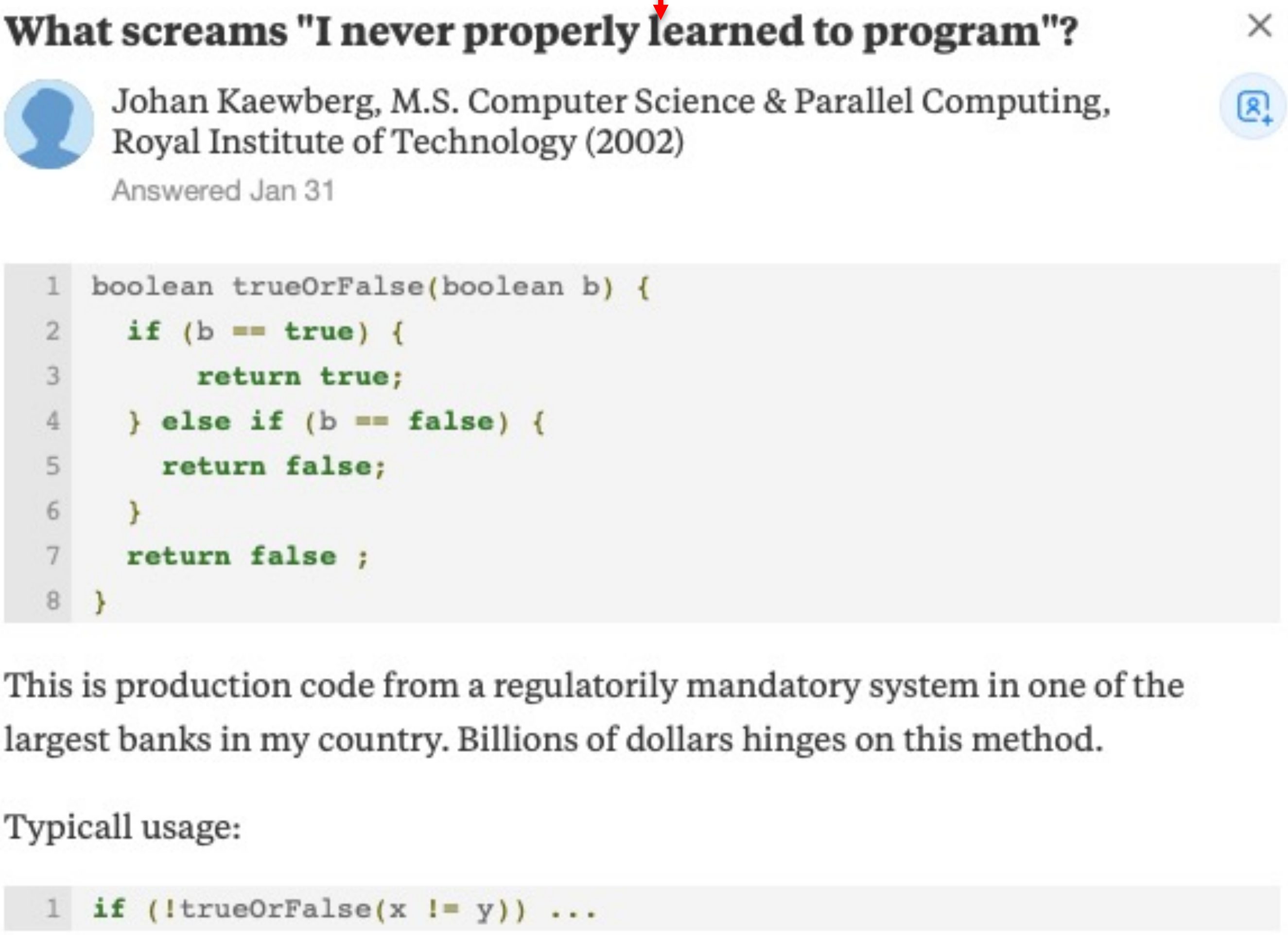
DR JEFF SOFTWARE
INDIE APP DEVELOPER
©2016-2022
Jeff Drobman

Boolean

true or false

`boolean x = true;`

~~`if (x) == true {`~~ → `if (x) {`



What screams "I never properly learned to program"?

Johan Kaewberg, M.S. Computer Science & Parallel Computing, Royal Institute of Technology (2002)

Answered Jan 31

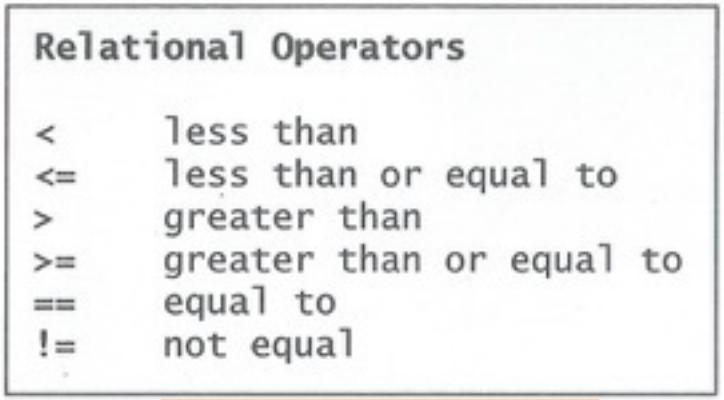
```
boolean trueOrFalse(boolean b) {
  if (b == true) {
      return true;
  } else if (b == false) {
    return false;
  }
  return false ;
}
```

This is production code from a regulatorily mandatory system in one of the largest banks in my country. Billions of dollas hinges on this method.

Typicall usage:

```
if (!trueOrFalse(x != y)) ...
```

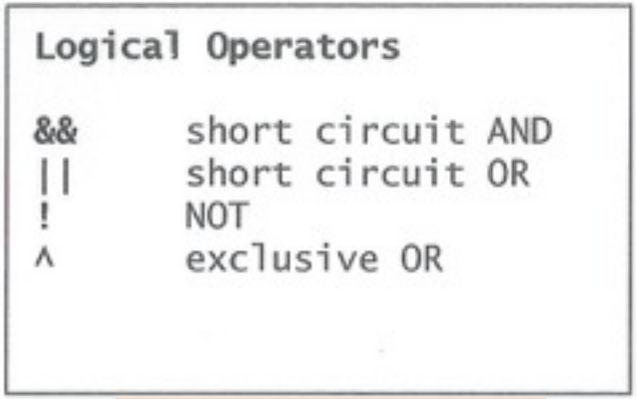
Logic



Relational Operators

<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	equal to
!=	not equal

any type → boolean



Logical Operators

&&	short circuit AND
||	short circuit OR
!	NOT
^	exclusive OR

boolean → boolean

if (x <= y+3) && x > 2 || _FLAG == true

_FLAG == true ⇔ _FLAG
_FLAG == false ⇔ ! _FLAG

- AND has 2 uses:
  1) Mask (1 lets in)
  2) Filter (0 keeps out)
- XOR has 2 uses:
  1) Bit complement/toggle
  2) Bit equal

Truth Tables

OR

X	Y	X || Y
0	0	0
0	1	1
1	0	1
1	1	1

INclusive

AND

X	Y	X && Y
0	0	0
0	1	0
1	0	0
1	1	1

XOR

	X	Y	X ^ Y
pass	0	0	0
	0	1	1
flip	1	0	1
	1	1	0
	control	data	

EXclusive

- ❖ AND has 2 uses:
  1) Mask (1 lets in)
  2) Filter (0 keeps out)
- ❖ XOR has 2 uses:
  1) Bit complement/toggle
  2) Bit equal

Bit-wise Operations

Appendix G p. 751 *NEW*

Operator	Name	Example	Result
&	Bitwise AND	11101 & 00111	00101
|	Bitwise OR	00010 | 11000	11010
^	Bitwise XOR	00111 ^ 11111	11000
~	1's complement	00111100	11000011
<<	Left shift (*2^n)	10101010 << 2	10101000
>>	Right shift, arith SE	10101011 >> 2	11101010
>>>	Right shift, logical	10101011 >> 2	00101010

bit flip

*2^n

Integer types only

Section

Control Structures

Control

- Control Flow
  - Confined to structures

- Control Structures
  - IF-THEN-ELSE
  - LOOPS
    - **FOR** (iteration)
    - **WHILE**
    - DO-WHILE
  - Subroutines/Functions → *Methods*

Control Structures:

Conditional & Loops

VB

- ❖IF-THEN-ELSE

```
IF I <= N THEN
    <statement>
ELSEIF x <> y
    <statement>
ELSE
    <statement>
ENDIF
```

- ❖Case (Switch)
  - ✧ Integer values
  - ✧ Discrete objects

```
Select N
    Case is 1: <statement>
    Case is 2: <statement>
    Case is 3: <statement>
End Select
```

- ❖Loops
  - ✧ FOR (count)
  - ✧ (DO) WHILE (switch ON)

```
FOR I = 1 to N
    <statements>
Next
```

```
DO WHILE  I <= N
    <statements>
    I += 1
Continue
```

Conditional: *IF-THEN-ELSE*

```
IF I <= N THEN
    <statements>
ELSEIF x <> y
     <statements>
ELSE
     <statements>
ENDIF
```

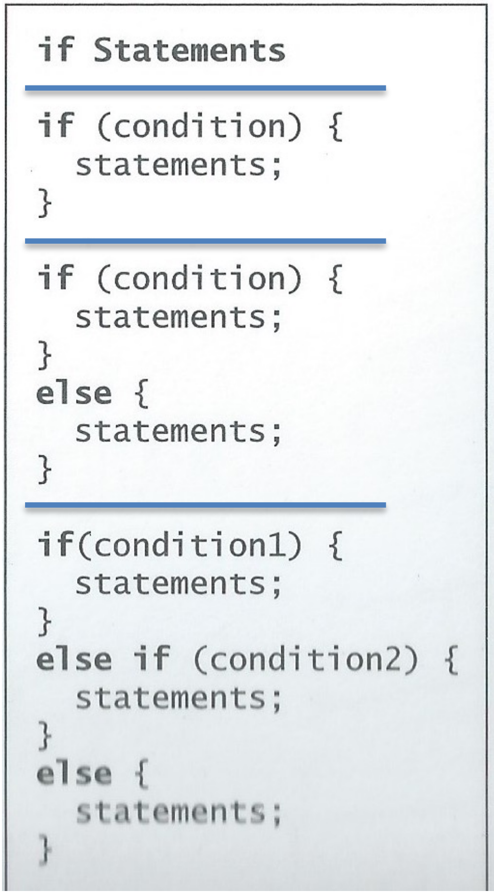
VB

```
if (I <= N)  {
    <statements>}
elseif (x <> y)  {
     <statements>}
else {
     <statements>
}
```

C/C++

```
if (I <= N)
    <statement>
elseif (x <> y)
     <statement>
else {
     <statements>
}
```

- ❖ *Cascaded*
- ❖ May be *nested*
    - ➢ (but bad practice)



if Statements

```
if (condition) {
   statements;
}
```

```
if (condition) {
   statements;
}
else {
   statements;
}
```

```
if(condition1) {
   statements;
}
else if (condition2) {
   statements;
}
else {
   statements;
}
```

Java

Example: *IF-THEN-ELSE*

```
if (num==1) {
 System.out.println (“Ordinary number = “+num);
}
else if (num==2) {
 System.out.println (“Special number = “+num);
}
else {
 System.out.println (“Illegal number = ”+num);
}
```

--OR--

```
String str;
if (num==1) {str=“Ordinary”;}
else if (num==2) {str=“Special”;}
else {str=“Illegal”;}
System.out.println (str+” number = ”+num);
```

single use of print

Conditional: *Expressions*

```
if (x > 0)
  y = 0;
else
    y = -1;
```

⟺

```
                    T     F
y = (x > 0) ? 0 : -1;
```

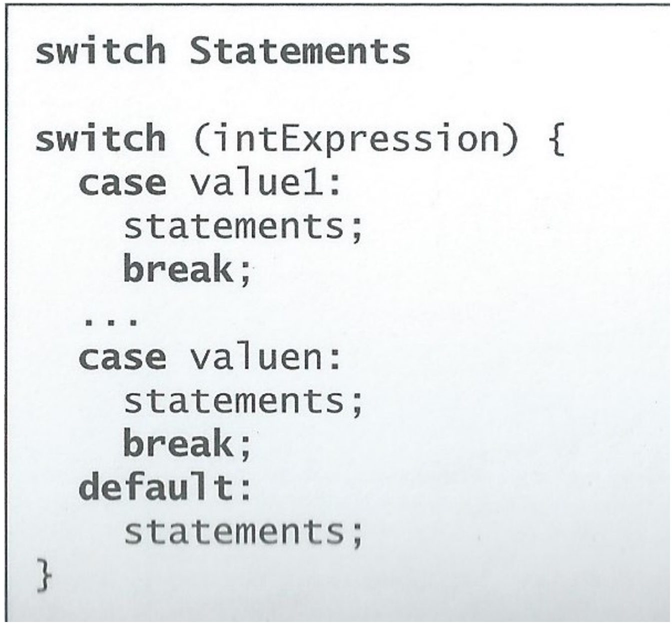
Conditional: *CASE/SWITCH*

```
SELECT N
   Case is 1: <statements>
   Case is 2: <statements>
   Case is 3: <statements>
   Case else: <statements>
END SELECT
```

VB

```
switch (N) {
   Case 1: <statements>
           break;
   Case 2: <statements>
           break;
   Case 3: <statements>
}
```

C/C++



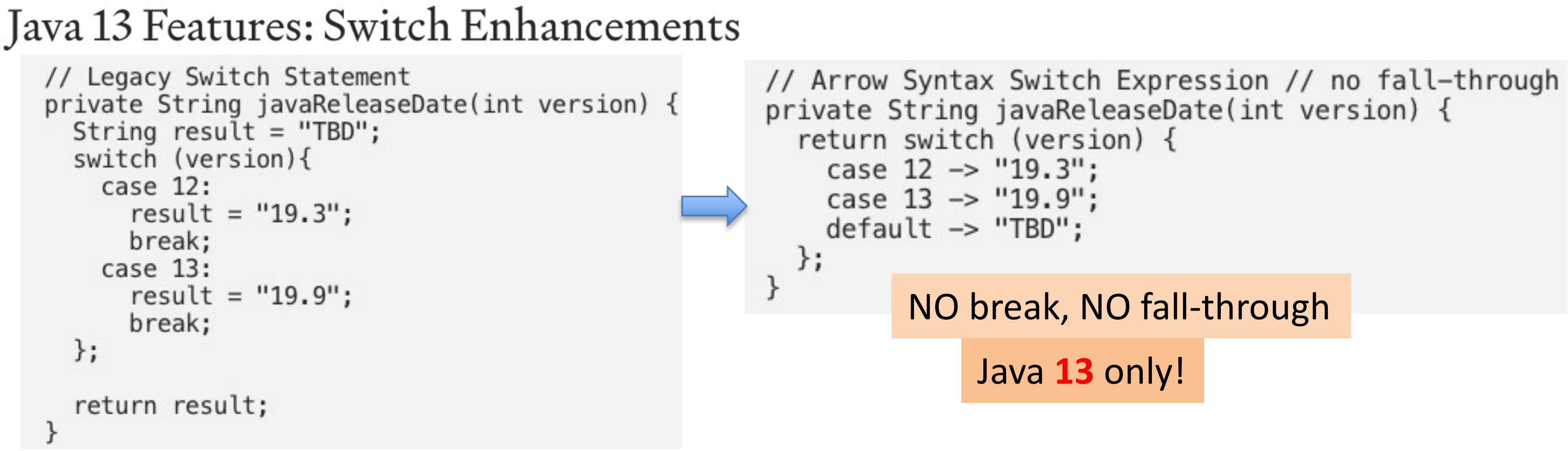
```
switch Statements

switch (intExpression) {
   case value1:
      statements;
      break;
   ...
   case valuen:
      statements;
      break;
   default:
      statements;
}
```

❖ *fall-through* if no “break”

Java

Examples: *CASE/SWITCH*

```
switch (Num) {
   case 1:
         str = “Ordinary”;
         break;
   case 2:
         str = “Special”;
         break;
   default:
         str = “Illegal”;
}
System.out.println (str + ” number = ” + num);
```



Java 13 Features: Switch Enhancements

```
// Legacy Switch Statement
private String javaReleaseDate(int version) {
  String result = "TBD";
  switch (version){
    case 12:
      result = "19.3";
      break;
    case 13:
      result = "19.9";
      break;
  };

  return result;
}
```

```
// Arrow Syntax Switch Expression // no fall-through
private String javaReleaseDate(int version) {
  return switch (version) {
    case 12 -> "19.3";
    case 13 -> "19.9";
    default -> "TBD";
  };
}
```

NO break, NO fall-through

Java 13 only!

Example: *Strings*

```
if (nam==“Joe”) {
 System.out.println (“Joe, do this.”);
}
else if (nam==“Mary”) {
 System.out.println (“Mary, do something else.”);
}
else {  //Bob
 System.out.println (“Bob, do Joe’s job.”);
}
```

`if (nam == “Joe”)`

does NOT work!

```
String str;
if (nam.equals(“Joe”)) {str=“do this”;}
else if (nam.equals(“Mary”)) {str=“do something else”;}
else {str=“do Joe’s job”;}
System.out.println (nam+”, “+str+”.”);
```

equalsIgnoreCase

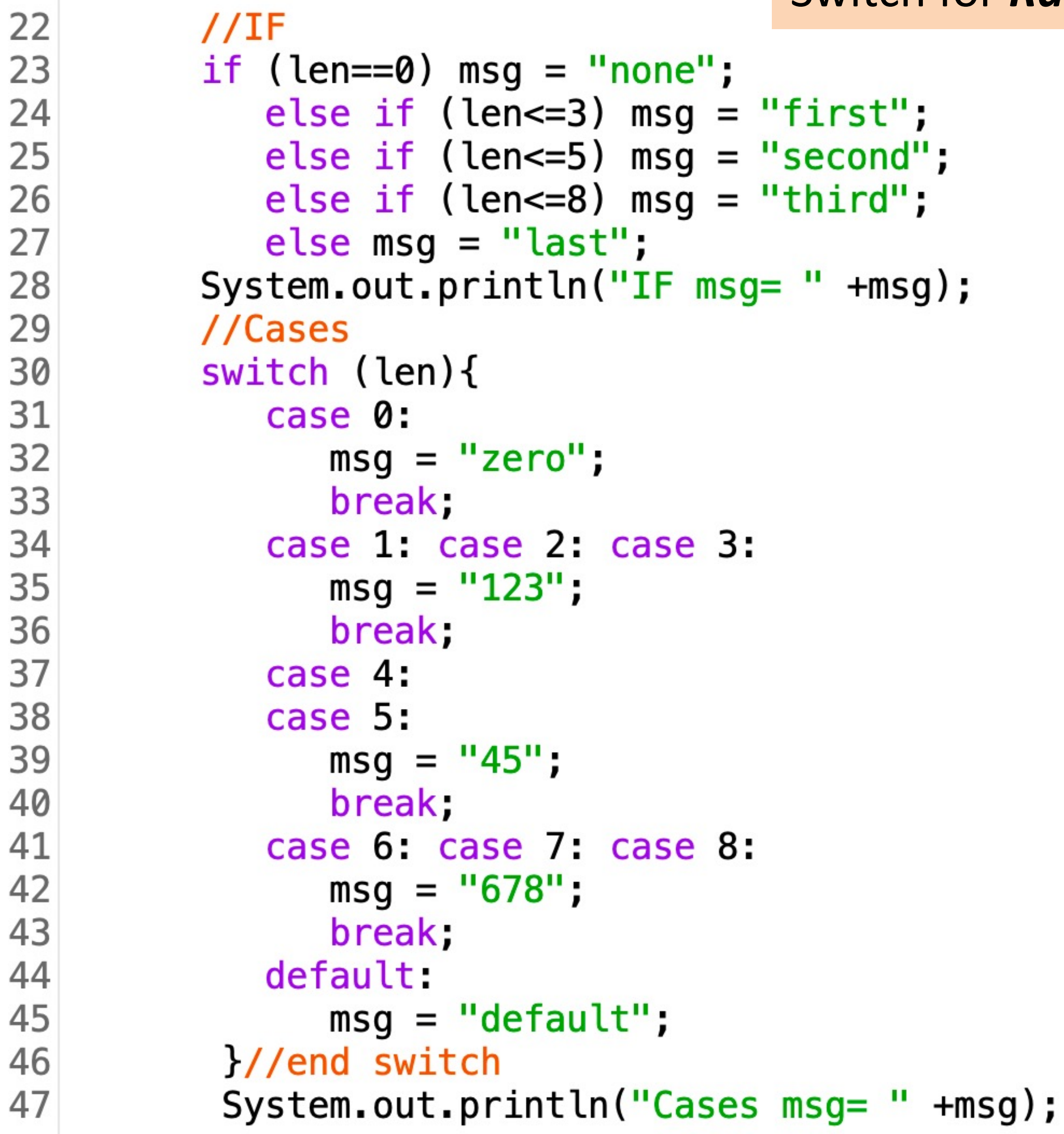
Example: *CASE/SWITCH*

- ❖ Switch can also be a String or Object

```
String name = “Joe”;
switch (name) {
   case “Joe”:  //Joe
        str = “do this”;
        break;
   case “Mary”:  //Mary
       str = “do something else”;
       break;
   default:  //Bob or other
       str = “Bob, do Joe’s job”;
}
System.out.println (nam + ”, “ + str + ”.”);
```

CASE/SWITCH Ranges

Switch for ***Ranges***



```
//IF
if (len==0) msg = "none";
   else if (len<=3) msg = "first";
   else if (len<=5) msg = "second";
   else if (len<=8) msg = "third";
   else msg = "last";
System.out.println("IF msg= " +msg);
//Cases
switch (len){
   case 0:
      msg = "zero";
      break;
   case 1: case 2: case 3:
      msg = "123";
      break;
   case 4:
   case 5:
      msg = "45";
      break;
   case 6: case 7: case 8:
      msg = "678";
      break;
   default:
      msg = "default";
 }//end switch
 System.out.println("Cases msg= " +msg);
```

Loops

- Loops
  - For
  - While
  - Do-While

Code Blocks – *Loops*

```
DO WHILE (I <= N)
    <statements>
    I += 1
CONTINUE
```

```
FOR I = 1 to N
    <statements>
NEXT
```

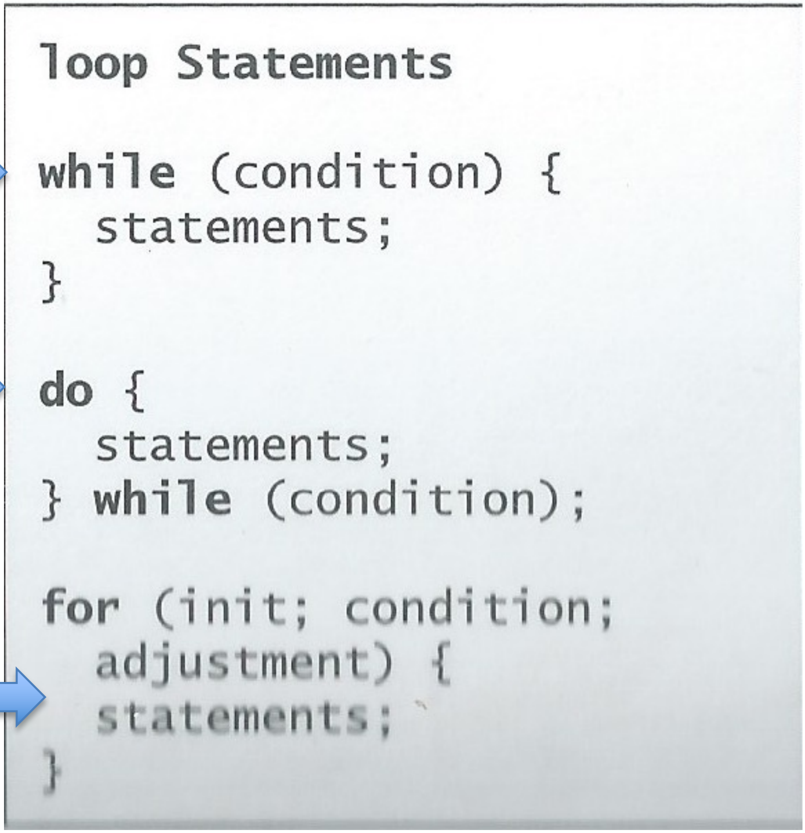
VB

```
while (I <= N)  {
    <statements>
    I ++
}
```

```
do  {
    <statements>
    I ++
} while (I <= N);
```

```
for (I = 1; I<N; I++) {
    <statements>
}
```

C



loop Statements

```
while (condition) {
   statements;
}

do {
   statements;
} while (condition);

for (init; condition;
   adjustment) {
   statements;
}
```

Java

while (1) { }

embedded apps

For Loops

```
for (i = 1; i<=N; i++) {
    <statements>
}
```

- ❖ executes N times
- ❖ i = **1 to N**
- ❖ i *post* increments

```
for (i = 0; i<N; i++) {
    <statements>
}
```

- ❖ executes N times
- ❖ i = **0 to N-1**
- ❖ i *post* increments

```
for (i = 0; i<=N; ++i) {
    <statements>
}
```

- ❖ executes N times
- ❖ i = **1 to N**
- ❖ i *pre* increments

```
for (i = 0; i<=N; ++i) {
    <statements>
if (<cond>) break;
}
```

- ❖ break → EXIT early

Continuous Loops

```
while(<cond>) {
```

or use this

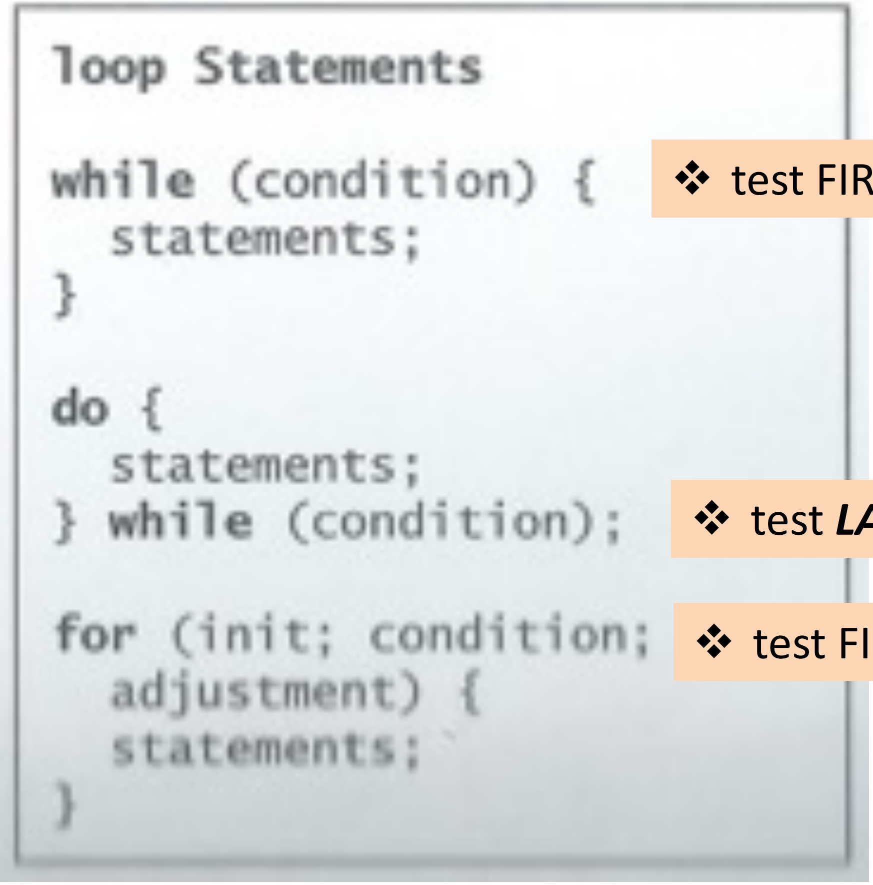
```
while(true) {
```

•
•
•

```
if (<cond>) break;
}
```

Project:
Thermostat

Loop Conditions



```
loop Statements

while (condition) {
   statements;
}

do {
   statements;
} while (condition);

for (init; condition;
   adjustment) {
   statements;
}
```

- ❖ test FIRST
- ❖ break → EXIT early
- ❖ test ***LAST***
- ❖ test FIRST

Loops: *Continue*

```
while ( i <= N) {
    <statements>
    if ( x > m) break;
}
```

```
while ( i <= N && x <= m) {
    <statements>
}
```

not break

!(x > m) == (<= m)

- break
- continue

```
while ( i <= N) {
    <statements>
    if ( x > m) continue;
    <statements>
}
```

```
while ( i <= N) {
    <statements>
    if ( x <= m)   {
    <statements>
    }
}
```

not continue

Loops: *While, Do*

```
while (x <= m) {
    <statements>
    x = y + z;
    if ( x > m) break;
    <statements>
}
```

→

```
while (true) {
    <statements>
    x = y + z;
    if ( x > m) break;
    <statements>
}
```

```
while (x <= m) {
    <statements>
    x = y + z;
    if ( x > m) break;
}
```

→

```
do {
    <statements>
    x = y + z;
}  while (x <= m);
```

Strings

Strings

as **Class**

Special Char + String Literals

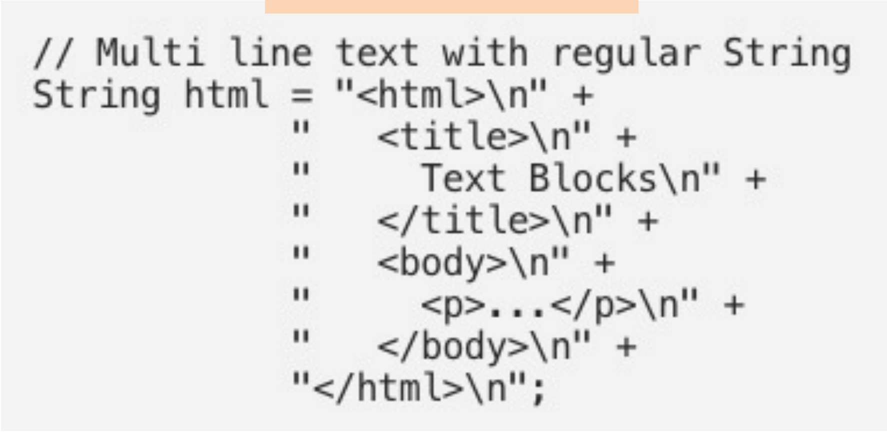
may declare variables for these:

space	char sp = ‘ ‘; or char sp = \u0020;
2x sp	String sp2 = sp + sp; //etc for 2..n
?	char qmark = ‘?’;
@	char at = ‘@’; //etc for others

Null → null (already a *reserved* literal)

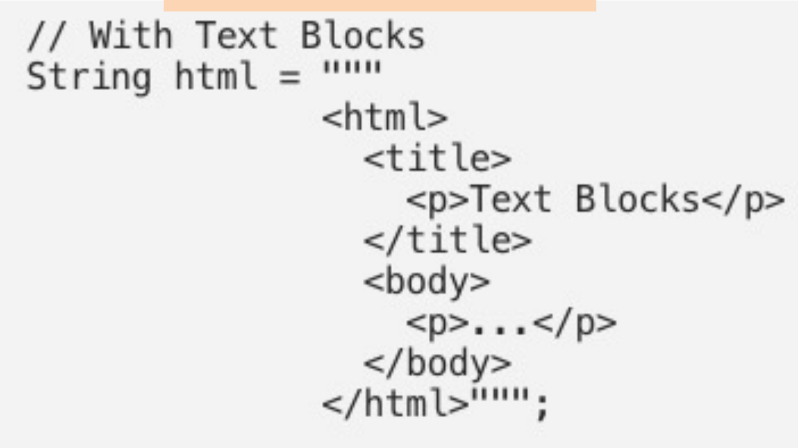
Java 13 Features:
and Text Blocks

Multi-line literals



```
// Multi line text with regular String
String html = "<html>\n" +
            "   <title>\n" +
            "     Text Blocks\n" +
            "   </title>\n" +
            "   <body>\n" +
            "     <p>...</p>\n" +
            "   </body>\n" +
            "</html>\n";
```

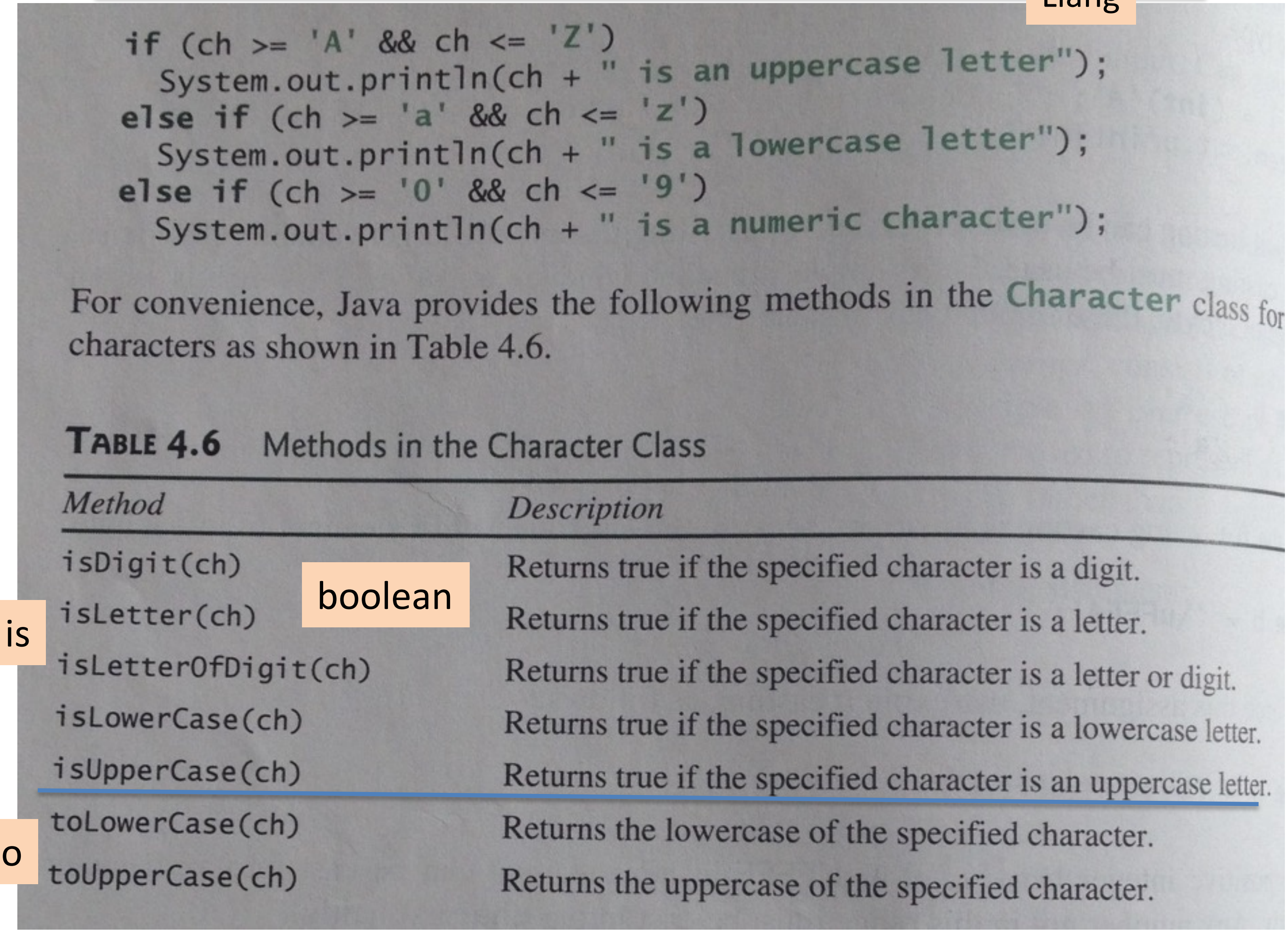
New **Text Blocks** “””



```
// With Text Blocks
String html = """
              <html>
                <title>
                  <p>Text Blocks</p>
                </title>
                <body>
                  <p>...</p>
                </body>
              </html>""";
```

Char Methods

Liang



```
if (ch >= 'A' && ch <= 'Z')
  System.out.println(ch + " is an uppercase letter");
else if (ch >= 'a' && ch <= 'z')
  System.out.println(ch + " is a lowercase letter");
else if (ch >= '0' && ch <= '9')
  System.out.println(ch + " is a numeric character");
```

For convenience, Java provides the following methods in the Character class for characters as shown in Table 4.6.

TABLE 4.6 Methods in the Character Class

Method	*Description*
isDigit(ch)	Returns true if the specified character is a digit.
isLetter(ch)	Returns true if the specified character is a letter.
isLetterOfDigit(ch)	Returns true if the specified character is a letter or digit.
isLowerCase(ch)	Returns true if the specified character is a lowercase letter.
isUpperCase(ch)	Returns true if the specified character is an uppercase letter.
toLowerCase(ch)	Returns the lowercase of the specified character.
toUpperCase(ch)	Returns the uppercase of the specified character.

boolean

is

to

Char Methods

zyBook

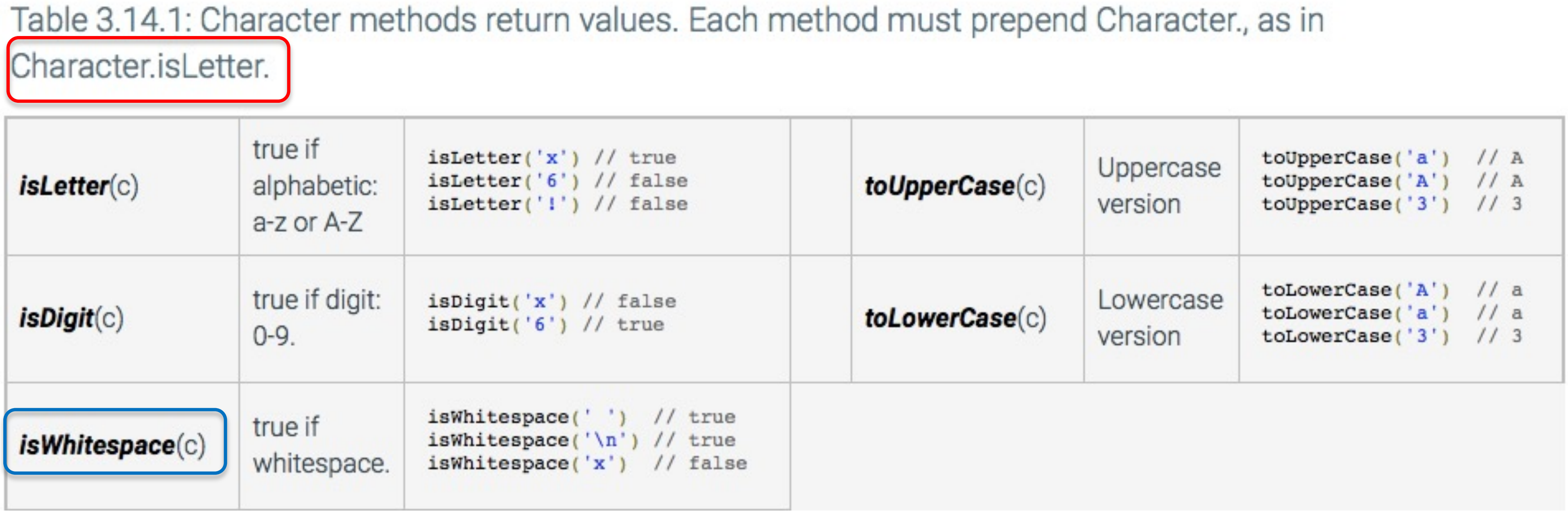


Table 3.14.1: Character methods return values. Each method must prepend Character., as in Character.isLetter.

isLetter(c)	true if alphabetic: a-z or A-Z	`isLetter('x') // true`
		`isLetter('6') // false`
		`isLetter('!') // false`
				toUpperCase(c)	Uppercase version	`toUpperCase('a')  // A`
						`toUpperCase('A')  // A`
						`toUpperCase('3')  // 3`

isDigit(c)	true if digit: 0-9.	`isDigit('x') // false`
		`isDigit('6') // true`
				toLowerCase(c)	Lowercase version	`toLowerCase('A')  // a`
						`toLowerCase('a')  // a`
						`toLowerCase('3')  // 3`

isWhitespace(c)	true if whitespace.	`isWhitespace(' ')  // true`
		`isWhitespace('\n') // true`
		`isWhitespace('x')  // false`				

Char Methods Example

zyBook

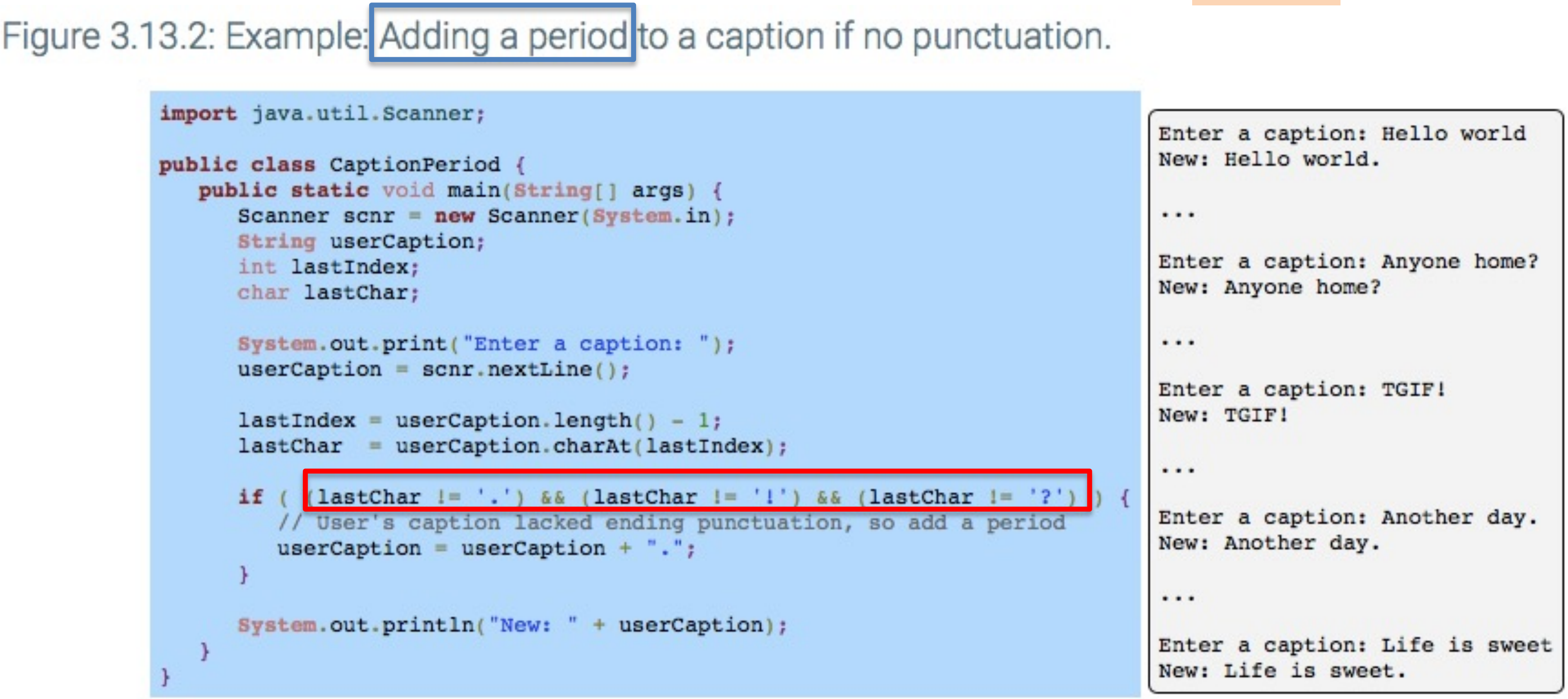


Figure 3.13.2: Example: Adding a period to a caption if no punctuation.

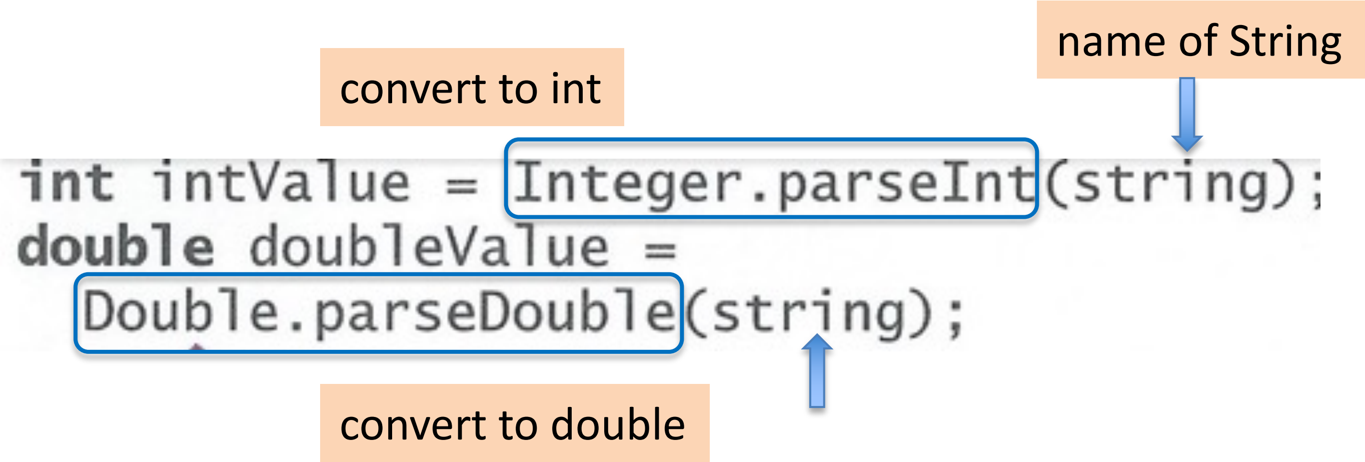
```java
import java.util.Scanner;

public class CaptionPeriod {
   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      String userCaption;
      int lastIndex;
      char lastChar;

      System.out.print("Enter a caption: ");
      userCaption = scnr.nextLine();

      lastIndex = userCaption.length() - 1;
      lastChar  = userCaption.charAt(lastIndex);

      if ( (lastChar != '.') && (lastChar != '!') && (lastChar != '?') ) {
         // User's caption lacked ending punctuation, so add a period
         userCaption = userCaption + ".";
      }

      System.out.println("New: " + userCaption);
   }
}
```

```
Enter a caption: Hello world
New: Hello world.

...

Enter a caption: Anyone home?
New: Anyone home?

...

Enter a caption: TGIF!
New: TGIF!

...

Enter a caption: Another day.
New: Another day.

...

Enter a caption: Life is sweet
New: Life is sweet.
```

Convert String to Numeric

- convert <string> to <Type>

<Type>.parse<Type>(<string>)



name of String

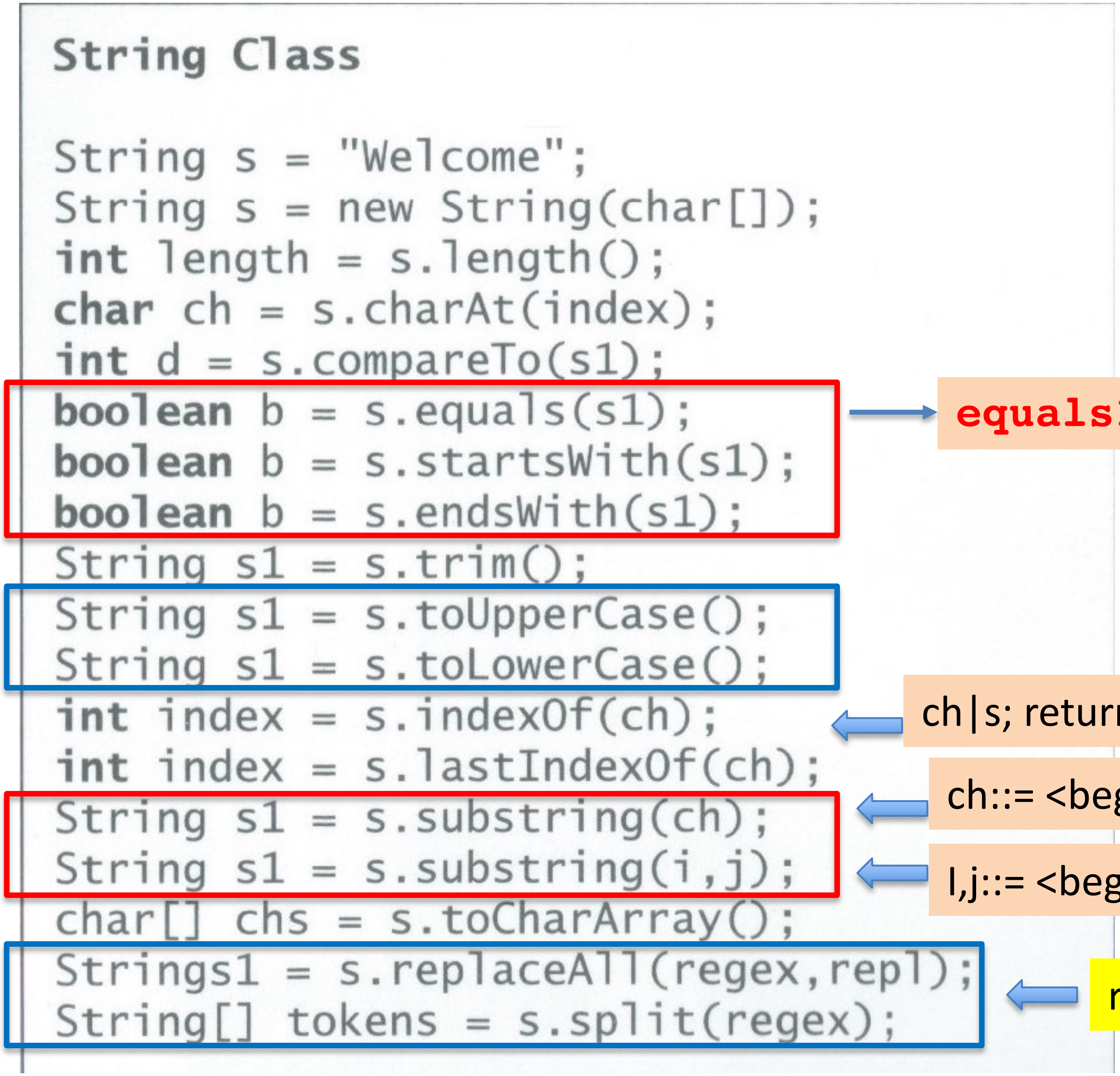
convert to int

```
int intValue = Integer.parseInt(string);
double doubleValue =
   Double.parseDouble(string);
```

convert to double

String Methods

Liang



```
String Class

String s = "Welcome";
String s = new String(char[]);
int length = s.length();
char ch = s.charAt(index);
int d = s.compareTo(s1);
boolean b = s.equals(s1);
boolean b = s.startsWith(s1);
boolean b = s.endsWith(s1);
String s1 = s.trim();
String s1 = s.toUpperCase();
String s1 = s.toLowerCase();
int index = s.indexOf(ch);
int index = s.lastIndexOf(ch);
String s1 = s.substring(ch);
String s1 = s.substring(i,j);
char[] chs = s.toCharArray();
Strings1 = s.replaceAll(regex,repl);
String[] tokens = s.split(regex);
```

equalsIgnoreCase

ch|s; returns (0-n | -1)

ch::= <beginIndex>

I,j::= <beginIndex, endIndex>

regex

String Compare

Liang

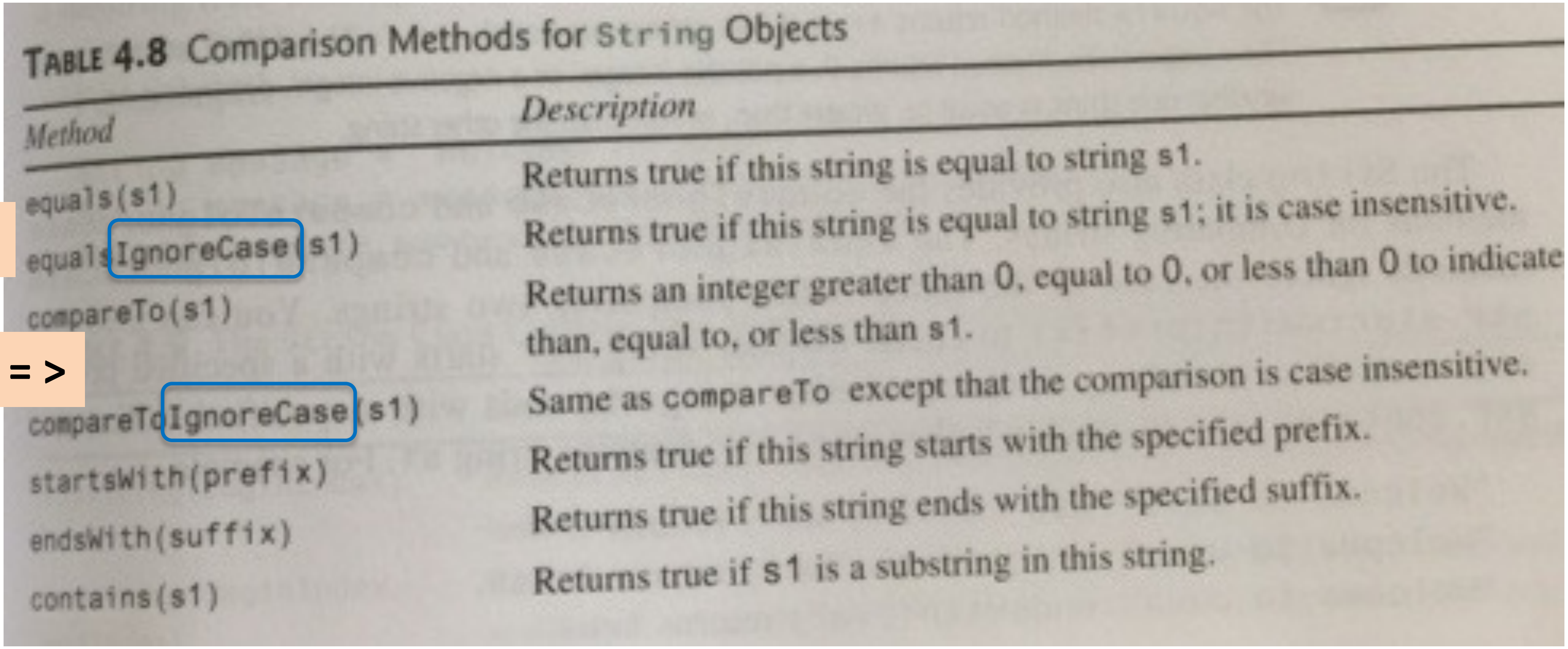


TABLE 4.8 Comparison Methods for `String` Objects

Method	*Description*
`equals(s1)`	Returns true if this string is equal to string `s1`.
`equalsIgnoreCase(s1)`	Returns true if this string is equal to string `s1`; it is case insensitive.
`compareTo(s1)`	Returns an integer greater than 0, equal to 0, or less than 0 to indicate than, equal to, or less than `s1`.
`compareToIgnoreCase(s1)`	Same as `compareTo` except that the comparison is case insensitive.
`startsWith(prefix)`	Returns true if this string starts with the specified prefix.
`endsWith(suffix)`	Returns true if this string ends with the specified suffix.
`contains(s1)`	Returns true if `s1` is a substring in this string.

=

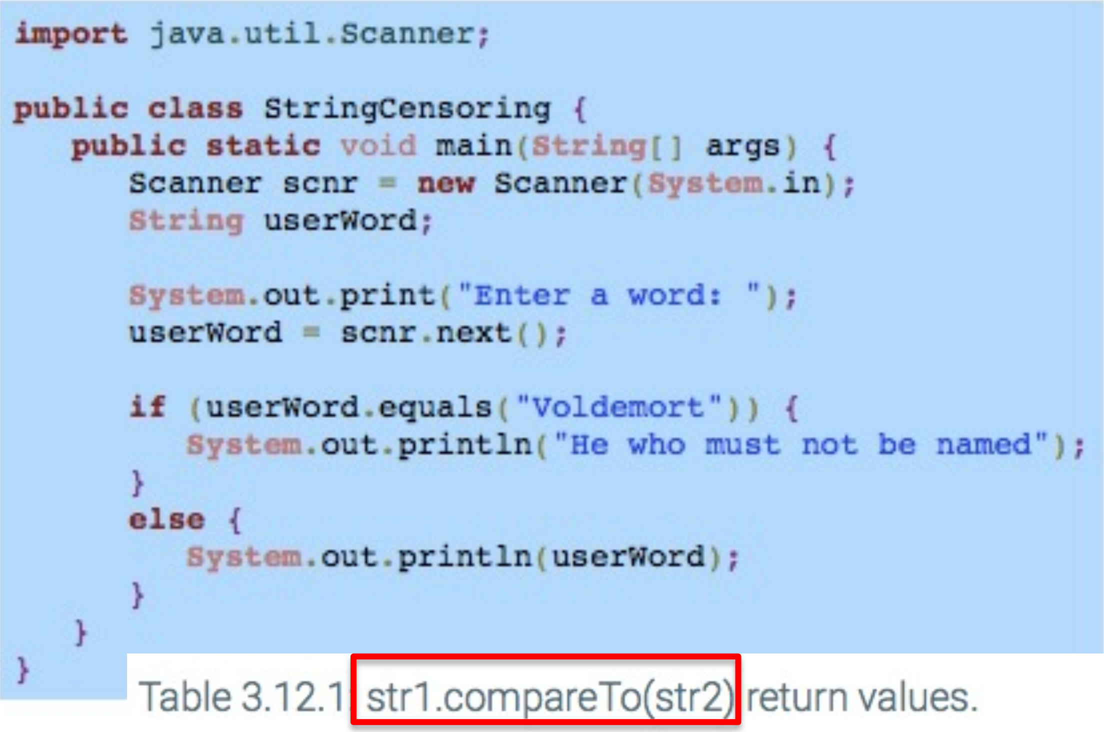
< = >

String Compare

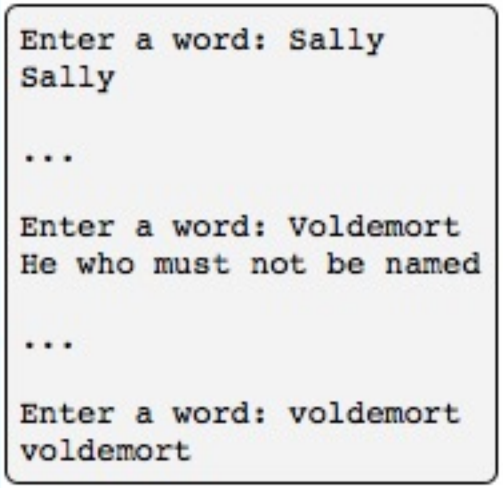
zyBook

Figure 3.12.1: String equality example: Censoring.

=



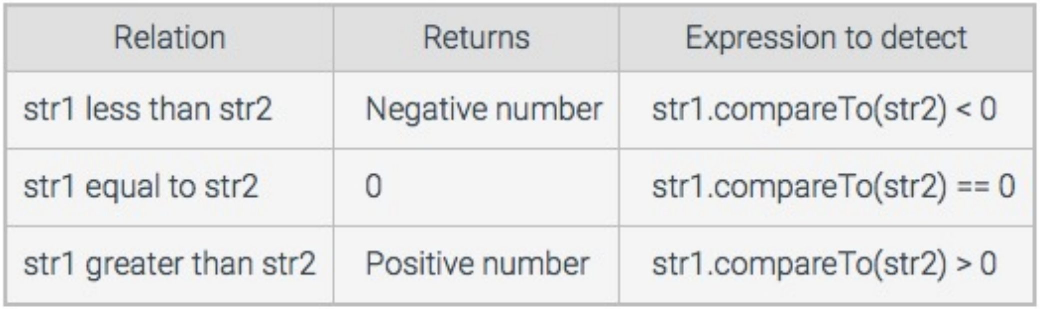
```
import java.util.Scanner;

public class StringCensoring {
   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      String userWord;

      System.out.print("Enter a word: ");
      userWord = scnr.next();

      if (userWord.equals("Voldemort")) {
         System.out.println("He who must not be named");
      }
      else {
         System.out.println(userWord);
      }
   }
}
```



```
Enter a word: Sally
Sally

...

Enter a word: Voldemort
He who must not be named

...

Enter a word: voldemort
voldemort
```

Table 3.12.1: str1.compareTo(str2) return values.

< = >

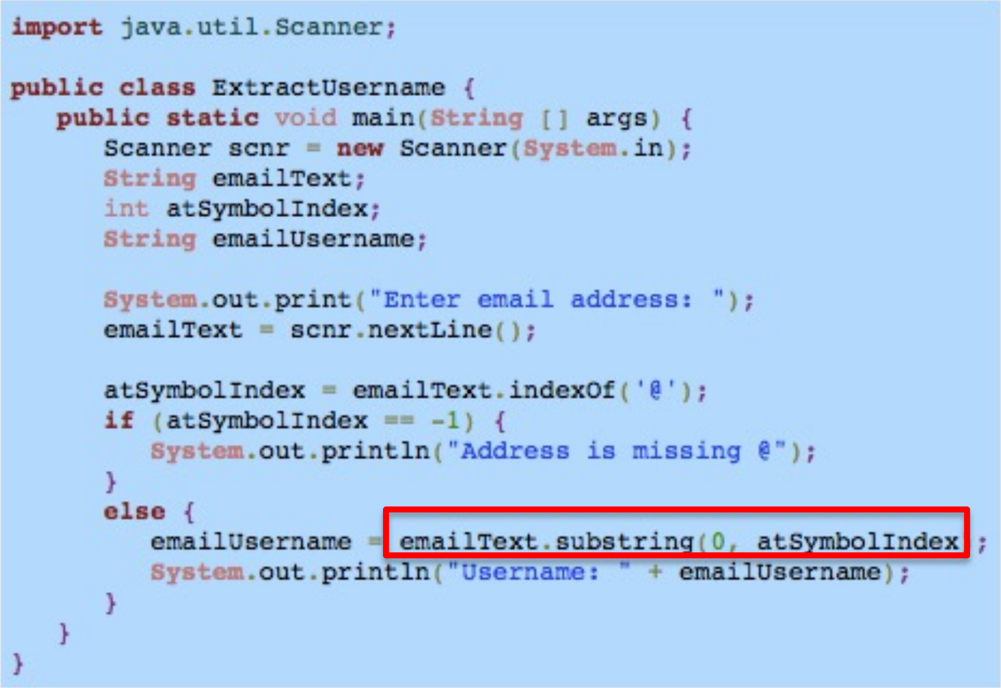


Relation	Returns	Expression to detect
str1 less than str2	Negative number	str1.compareTo(str2) < 0
str1 equal to str2	0	str1.compareTo(str2) == 0
str1 greater than str2	Positive number	str1.compareTo(str2) > 0

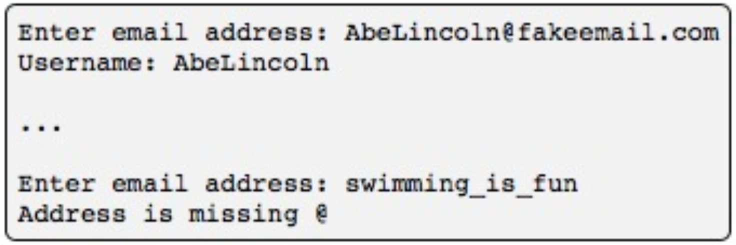
SubString Example

zyBook

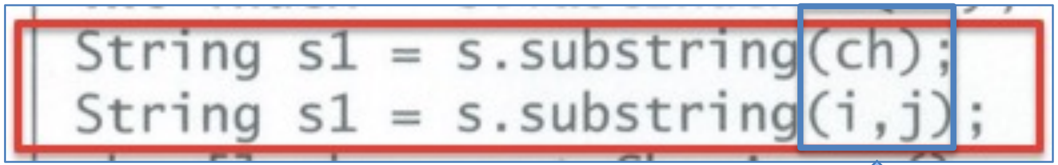
Figure 3.15.1: Example: Get username from email address.



```
import java.util.Scanner;

public class ExtractUsername {
   public static void main(String [] args) {
      Scanner scnr = new Scanner(System.in);
      String emailText;
      int atSymbolIndex;
      String emailUsername;

      System.out.print("Enter email address: ");
      emailText = scnr.nextLine();

      atSymbolIndex = emailText.indexOf('@');
      if (atSymbolIndex == -1) {
         System.out.println("Address is missing @");
      }
      else {
         emailUsername = emailText.substring(0, atSymbolIndex);
         System.out.println("Username: " + emailUsername);
      }
   }
}
```



```
Enter email address: AbeLincoln@fakeemail.com
Username: AbeLincoln

...

Enter email address: swimming_is_fun
Address is missing @
```



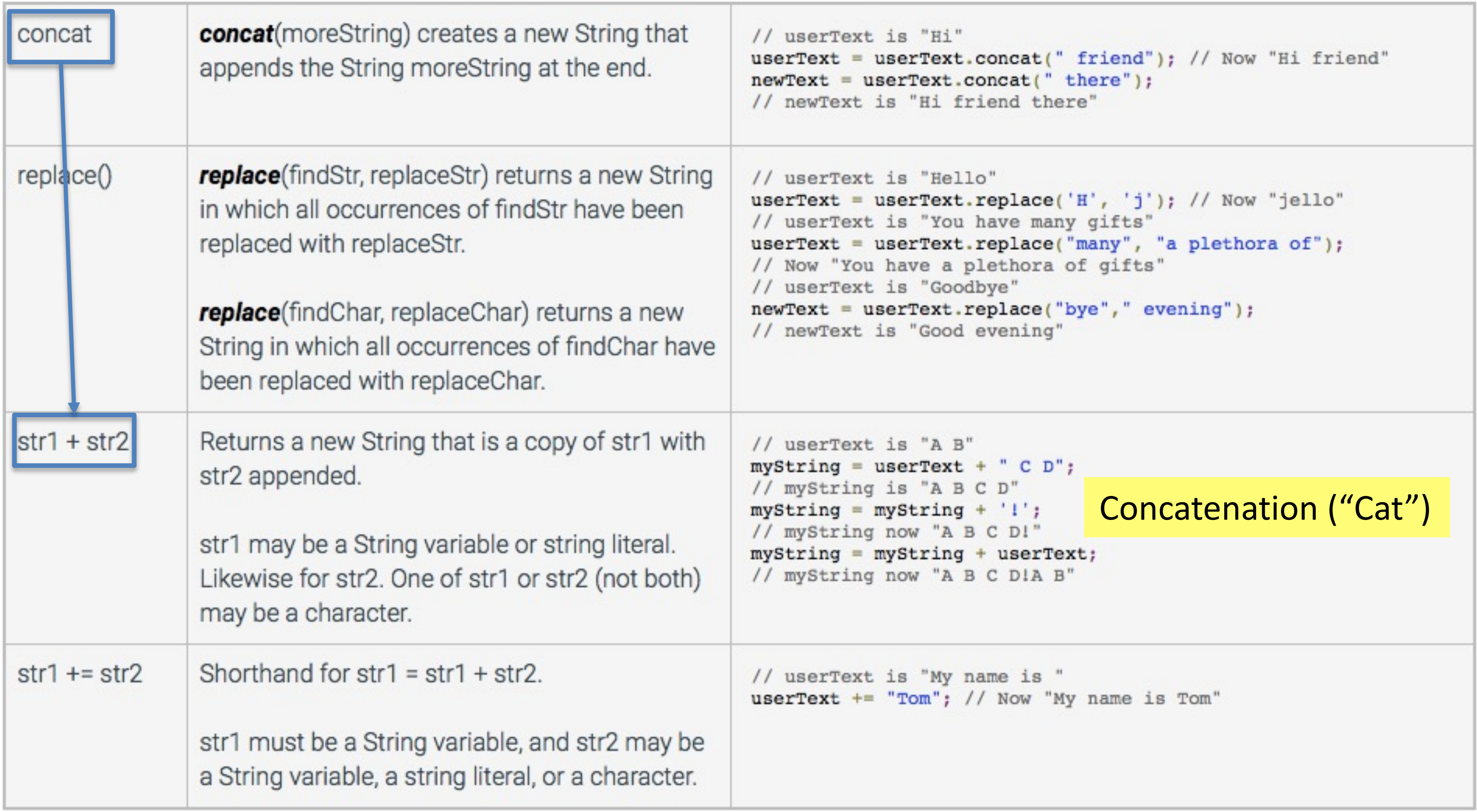
```
String s1 = s.substring(ch);
String s1 = s.substring(i,j);
```

2 *signatures*

String Modify Methods

zyBook

Table 3.15.2: String modify methods, invoked as myString.concat(moreString). Each returns a new String of the appropriate length.



concat	***concat***(moreString) creates a new String that appends the String moreString at the end.	// userText is "Hi"
		userText = userText.concat(" friend"); // Now "Hi friend"
		newText = userText.concat(" there");
		// newText is "Hi friend there"

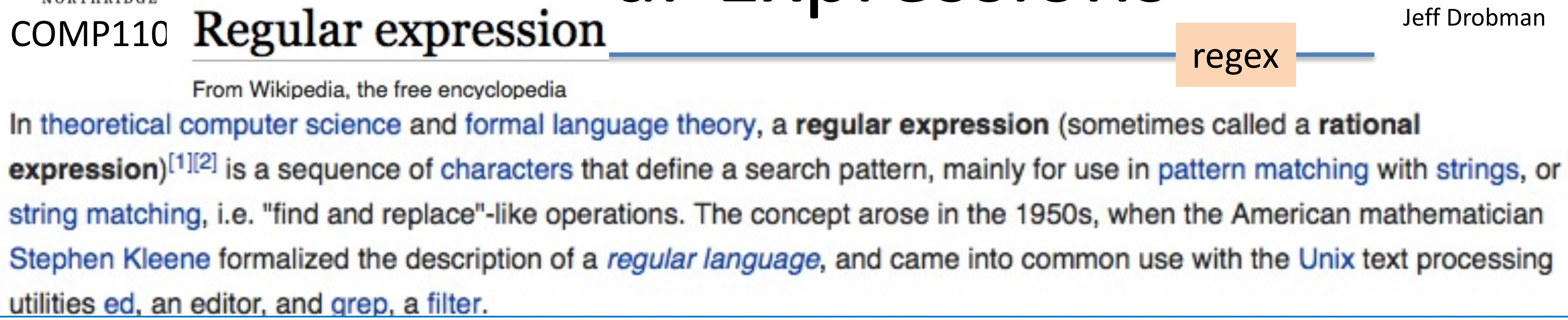
replace()	***replace***(findStr, replaceStr) returns a new String in which all occurrences of findStr have been replaced with replaceStr.
	replace(findChar, replaceChar) returns a new String in which all occurrences of findChar have been replaced with replaceChar.
		// userText is "Hello"
		userText = userText.replace('H', 'j'); // Now "jello"
		// userText is "You have many gifts"
		userText = userText.replace("many", "a plethora of");
		// Now "You have a plethora of gifts"
		// userText is "Goodbye"
		newText = userText.replace("bye"," evening");
		// newText is "Good evening"

str1 + str2	Returns a new String that is a copy of str1 with str2 appended.
	str1 may be a String variable or string literal. Likewise for str2. One of str1 or str2 (not both) may be a character.
		// userText is "A B"
		myString = userText + " C D";
		// myString is "A B C D"
		myString = myString + '!';
		// myString now "A B C D!"
		myString = myString + userText;
		// myString now "A B C D!A B"

str1 += str2	Shorthand for str1 = str1 + str2. str1 must be a String variable, and str2 may be a String variable, a string literal, or a character.	// userText is "My name is " userText += "Tom"; // Now "My name is Tom"

Concatenation ("Cat")

Regular Expressions



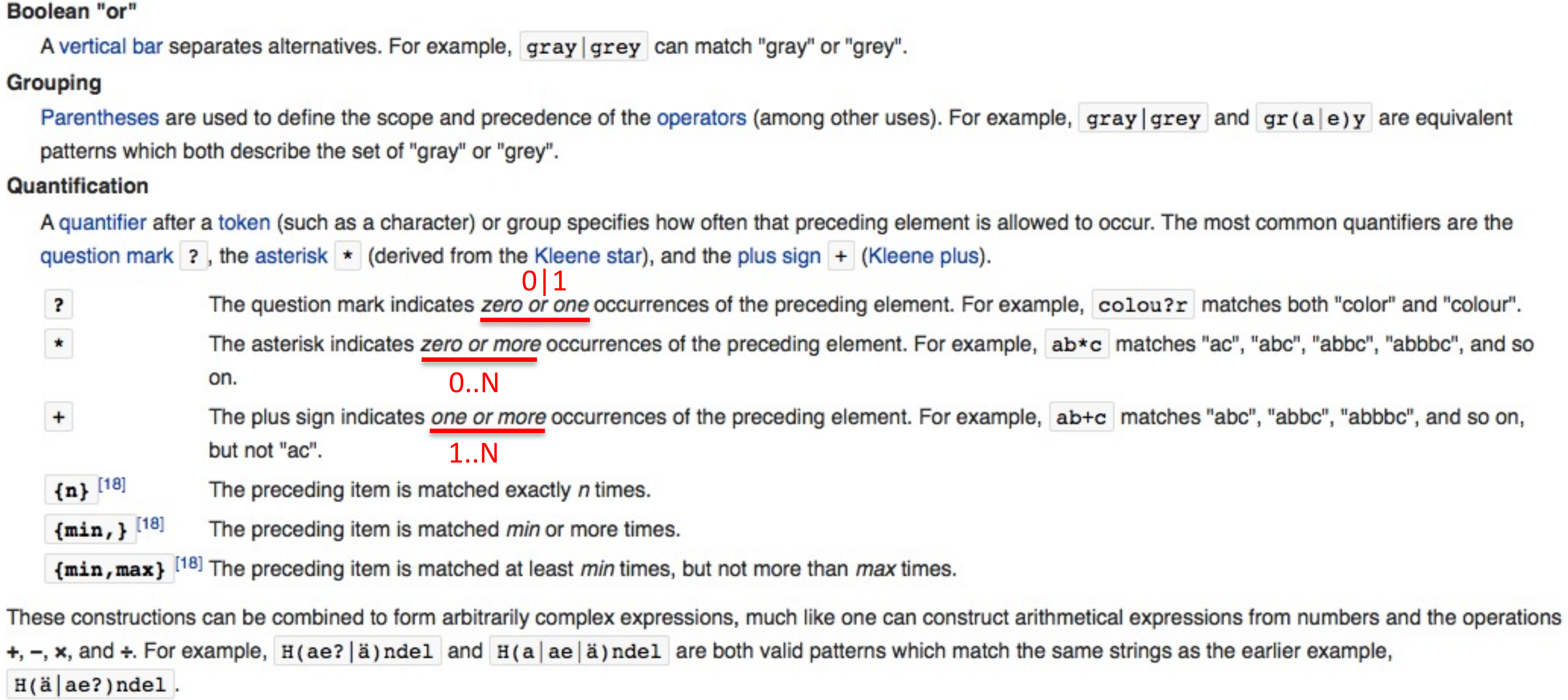
Regular expression

regex

From Wikipedia, the free encyclopedia

In theoretical computer science and formal language theory, a **regular expression** (sometimes called a **rational expression**)[1][2] is a sequence of characters that define a search pattern, mainly for use in pattern matching with strings, or string matching, i.e. "find and replace"-like operations. The concept arose in the 1950s, when the American mathematician Stephen Kleene formalized the description of a *regular language*, and came into common use with the Unix text processing utilities ed, an editor, and grep, a filter.

A regular expression, often called a **pattern**, is an expression used to specify a set of strings required for a particular purpose.



Boolean "or"

A vertical bar separates alternatives. For example, `gray|grey` can match "gray" or "grey".

Grouping

Parentheses are used to define the scope and precedence of the operators (among other uses). For example, `gray|grey` and `gr(a|e)y` are equivalent patterns which both describe the set of "gray" or "grey".

Quantification

A quantifier after a token (such as a character) or group specifies how often that preceding element is allowed to occur. The most common quantifiers are the question mark `?`, the asterisk `*` (derived from the Kleene star), and the plus sign `+` (Kleene plus).

`?`	The question mark indicates *zero or one* occurrences of the preceding element. For example, `colou?r` matches both "color" and "colour". 0|1
`*`	The asterisk indicates *zero or more* occurrences of the preceding element. For example, `ab*c` matches "ac", "abc", "abbc", "abbbc", and so on. 0..N
`+`	The plus sign indicates *one or more* occurrences of the preceding element. For example, `ab+c` matches "abc", "abbc", "abbbc", and so on, but not "ac". 1..N
`{n}`[18]	The preceding item is matched exactly *n* times.
`{min,}`[18]	The preceding item is matched *min* or more times.
`{min,max}`[18]	The preceding item is matched at least *min* times, but not more than *max* times.

These constructions can be combined to form arbitrarily complex expressions, much like one can construct arithmetical expressions from numbers and the operations +, −, ×, and ÷. For example, `H(ae?|ä)ndel` and `H(a|ae|ä)ndel` are both valid patterns which match the same strings as the earlier example, `H(ä|ae?)ndel`.

Regex in Java

- ref: **Liang** Appendix H, Table H.1

.	any single character (different than ‘?’)
[abc]	a, b or c (short for ‘a|b|c’)
[^abc]	NOT a, b or c (not any in the list)
[a-z]	a, b, c, …, z (LC letter)
[^a-z]	NOT a, b, c, …, z (not LC letter)
p?	0 or 1 x pattern *p*
p*	>=0 x pattern *p*
p+	>= 1 x pattern *p*

specials: (compare to escapes\)

[a-z] ⇔ \w && \D && [^_]

\d	digit	\D	NON digit
\w	*word* char	\W	NON => ([a-z] | [A-Z] | [0-9] | _)
\s	space	\S	NON space (actually “whitespace” to include others)

ix = str.indexOf(sp); ⇦ find spaces

where
char sp = ‘ ‘; or
char sp = \u0020;

Search Alpha Via Regex

```
String alpha = “abcdefghijklmnopqrstuvwxyz”;
```

```
if(alpha.indexOf(chx.toLower))
            ⇕
if(chx.matches(“[a-z] | [A-Z]”)
```

Matching Strings

- *boolean* expressions = true|false

ref: Appendix H, Table H.1

<string>.matches (“<regex>”)

examples
“Java is fun”.matches(“Java.*”)
“Java is fun”.matches(“Java.+”)
“Java fun”.matches(“Java\\s.?w?.?”)
“fun fun fun”.matches(“fun{3}”)

replaceAll(regex: <regex>, replacement: <repl_string>: <in_string>)
replaceFirst(regex: <regex>, replacement: <repl_string>: <in_string>)
split(regex: <regex>): <from_string>[ ])

filter out non-alphas

```
str.replaceAll([^a-z], empty);
```

```
if(!chx.matches(“[a-z] | [A-Z}”) chx=empty;
```

String Handling

VB

❖ STRING Data Type (VB)

Arrays vs. Substrings

```
'**STRING variables
Dim wstr, xstr, ystr, zstr, xxstr, alertst, srchstr, matchstr As String

'**STRING expressions, operators, functions
xstr &= ystr & "123" 'concatenate
zstr = substring(xstr, 1, 4)  'substrings (index, length)
wstr = substring(xstr, 5)  'substrings (index)

'**Strings are an option for arrays:  string indices ⇔ array indices
'string of 10 characters vs. array of 10 elements:  example for Y/N votes
Dim xstr As String, xarr(9) As String
xstr = "YNNYYNYYNN"
xarr(0) = "Y" : xarr(1) = "N" : xarr(2) = "N" : xarr(3) = "Y" : xarr(4) = "Y"
Xarr(5) = "N" : xarr(6) = "Y" : xarr(7) = "Y" : xarr(8) = "N" : xarr(9) = "N"

'load array of 10 from string
For I = 0 To 9
    Xarr(I) = substring(xstr,I,1)
Next
```

Methods

Methods

Methods

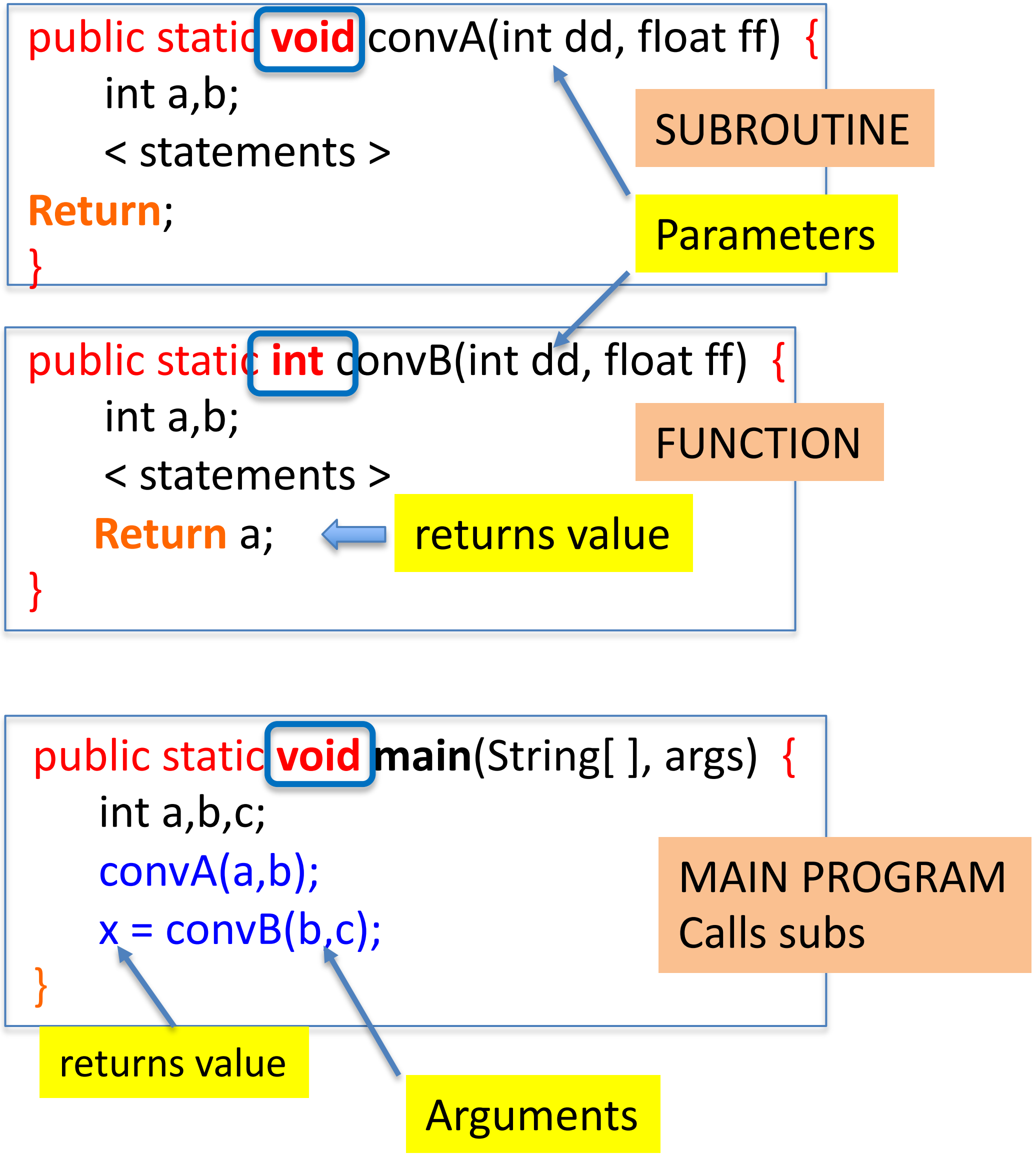
Midterm

```
[public][static] <type><name>(<parm list>) {
```

```
<parm list> ::= [<type><name>, <type><name>, … ]
```

```
[<type> may include arrays:
      int[] A, int[]B, int[] C
```

Methods in Java



```
public static void convA(int dd, float ff)  {
        int a,b;
        < statements >
Return;
}
```

SUBROUTINE

Parameters

```
public static int convB(int dd, float ff)  {
        int a,b;
        < statements >
      Return a;
}
```

FUNCTION

returns value

```
public static void main(String[ ], args)  {
        int a,b,c;
        convA(a,b);
        x = convB(b,c);
}
```

MAIN PROGRAM
Calls subs

returns value

Arguments

- *Signature*
- Call
- Parameter passing
  - By ***Value***
  - By *Reference**
- Return
  - Sub -> *void*
  - Function -> *value*

Arguments → *Parameters*

- 1 to 1
- **match type**
- *Java passes objects *by value* via ref to objects

Methods: Lab 3 Example

Fahr ⬄ Celc

```
public static void main(String[ ], args)  {
      <statements>
      if (F2C) temp = FtoC(fahr);
      else temp = CtoF(celc);
}
```

```
public static double FtoC(double ftemp)  {
      double ctemp = (ftemp-32) *5/9.0;
      return ctemp;
}
```

```
public static double CtoF(double ctemp)  {
      double ftemp = ctemp *9/5.0 + 32;
      return ftemp;
}
```

- *Signature*
- Call
- Parameter passing
  - By ***Value***
- Return
  - Sub -> *void*
  - Function -> *value*

- F2C
  - □ Global var (or)
  - □ Parameter

Methods: Lab 4 Example

Lab 4: Pals + Anagrams

```
public static void main(String[ ], args)  {
      <statements>
      boolean palFlag = isPal(inStr);
      boolean anaFlag = isAna(inStr);
}
```

```
public static boolean isPal(String strParm)  {
      boolean result = true;
      <statements>
      return result;
}
```

```
public static boolean isAna(String strParm)  {
      boolean result = true;
      <statements>
      return result;
}
```

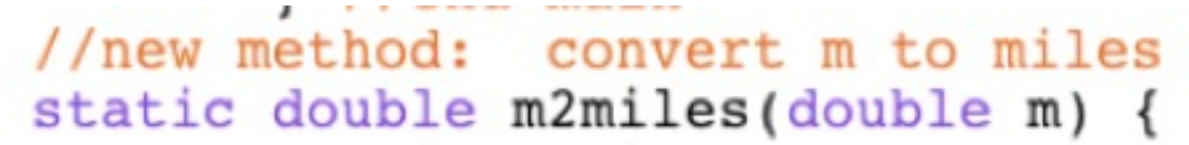
- *Signature*
- Call
- Parameter passing
  - By ***Value***
- Return
  - Sub -> *void*
  - Function -> *value*

- Flags

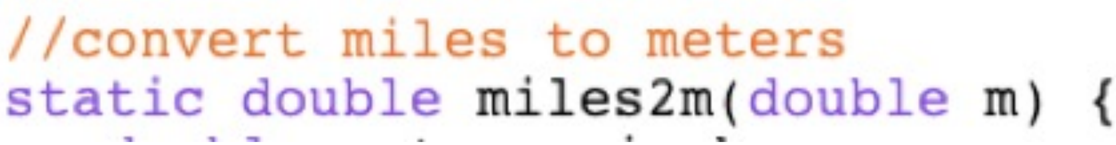
Methods

Midterm

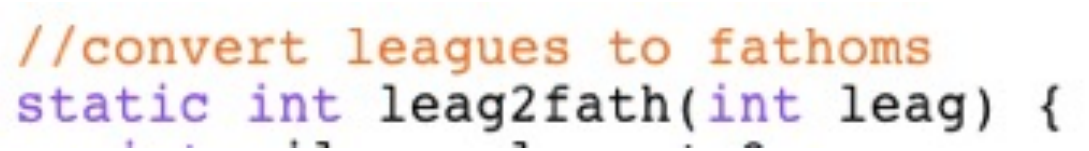
```
[public][static] <type><name>(<parm list>) {
```



```
//new method:  convert m to miles
static double m2miles(double m) {
```



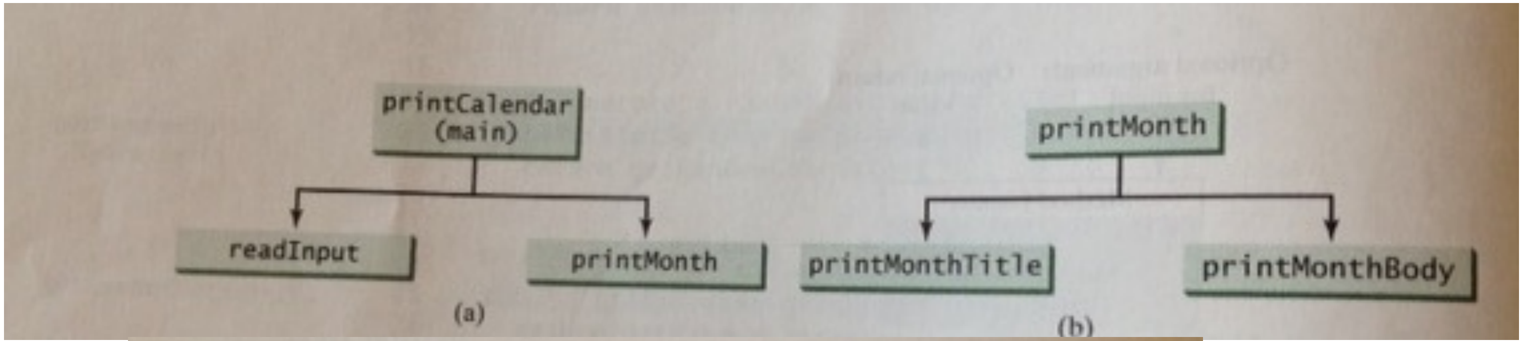
```
//convert miles to meters
static double miles2m(double m) {
```

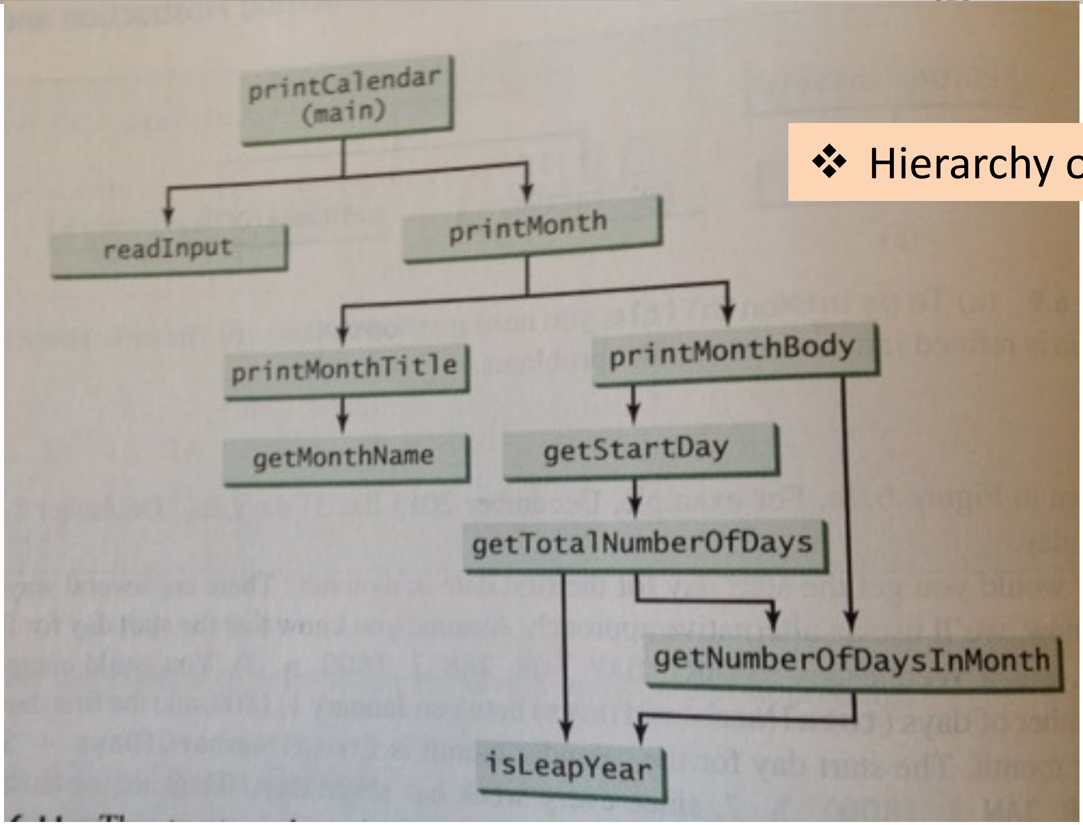


```
//convert leagues to fathoms
static int leag2fath(int leag) {
```

Methods: Structuring

Calendar App Example



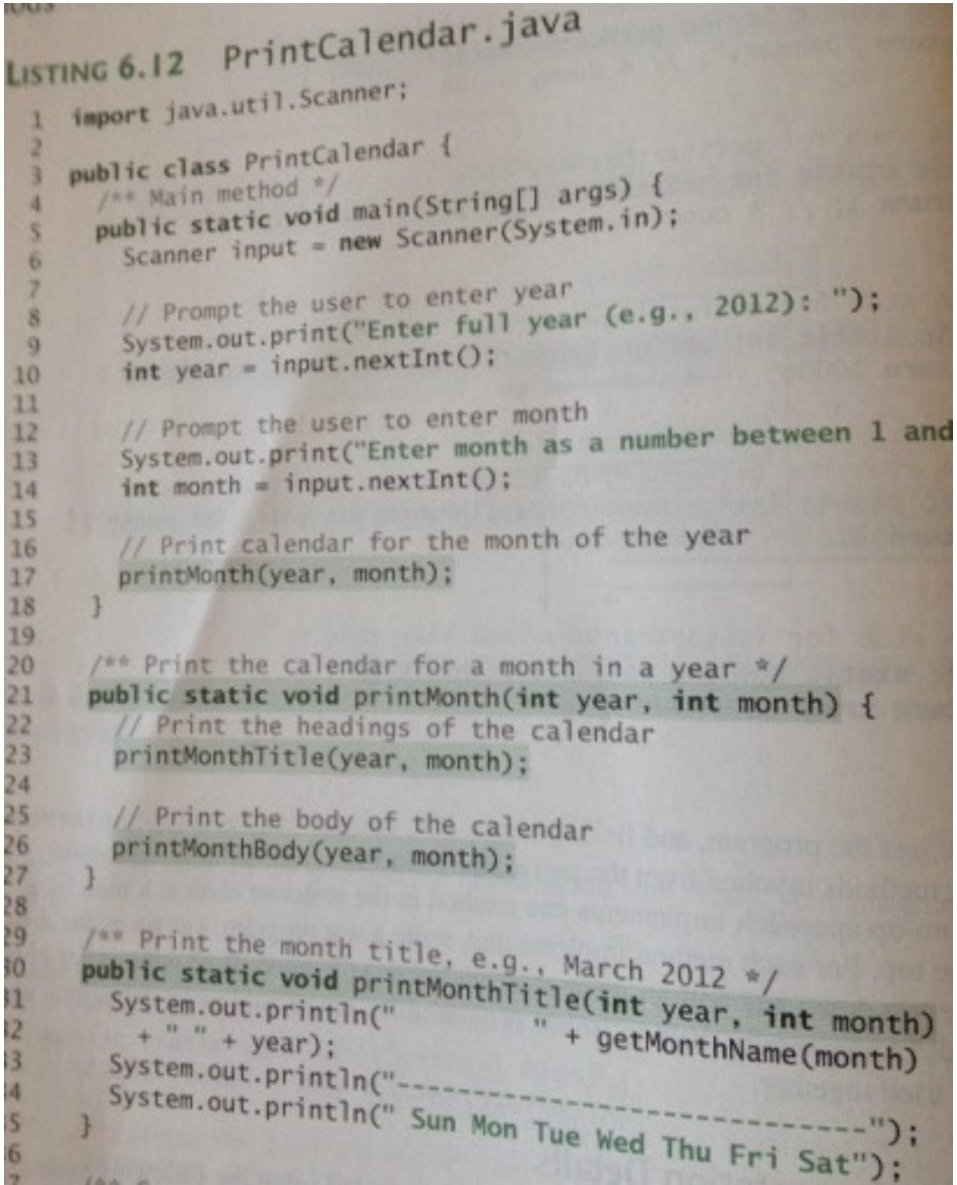


❖ Hierarchy of Methods

Methods: Structuring

Ch 6

Calendar App Example



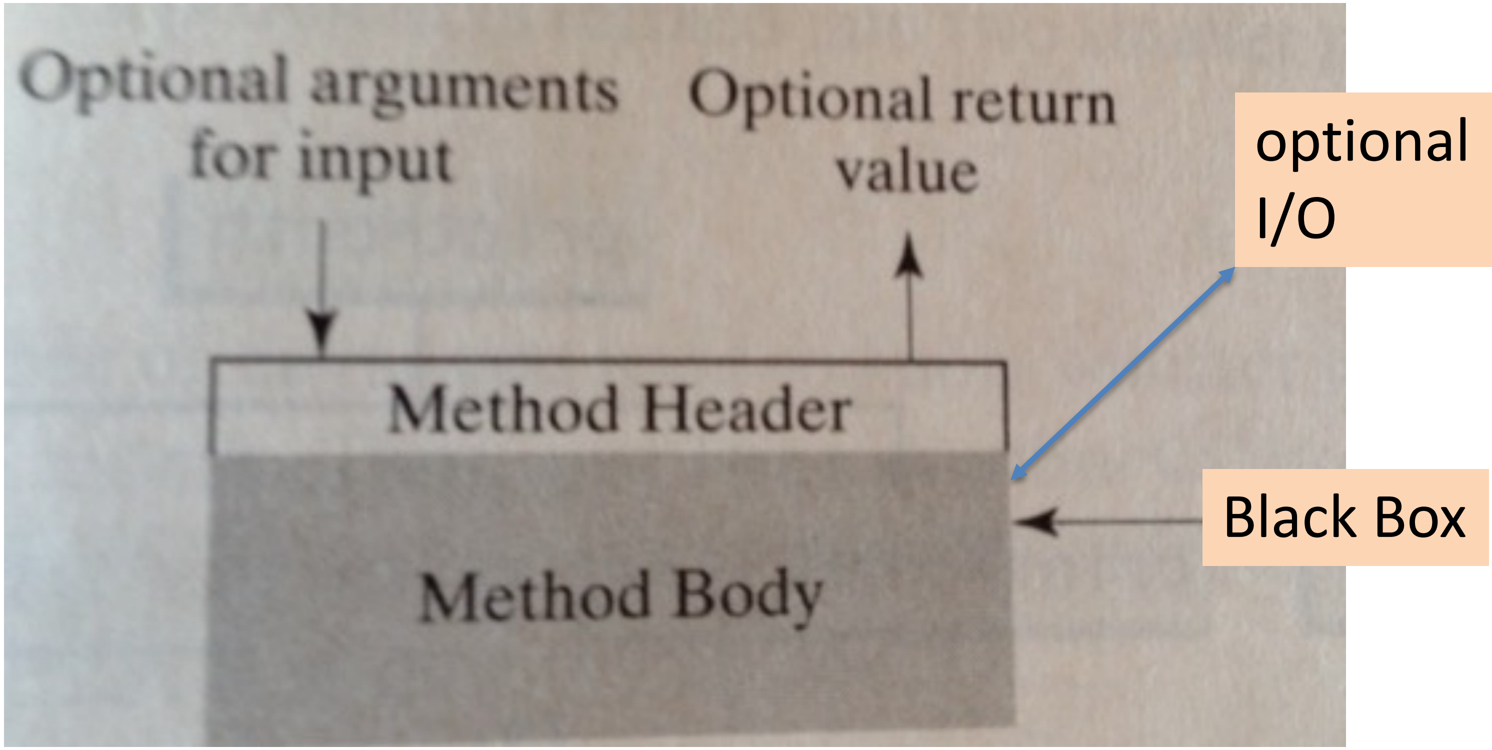
LISTING 6.12 PrintCalendar.java

```
import java.util.Scanner;

public class PrintCalendar {
  /** Main method */
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);

    // Prompt the user to enter year
    System.out.print("Enter full year (e.g., 2012): ");
    int year = input.nextInt();

    // Prompt the user to enter month
    System.out.print("Enter month as a number between 1 and
    int month = input.nextInt();

    // Print calendar for the month of the year
    printMonth(year, month);
  }

  /** Print the calendar for a month in a year */
  public static void printMonth(int year, int month) {
    // Print the headings of the calendar
    printMonthTitle(year, month);

    // Print the body of the calendar
    printMonthBody(year, month);
  }

  /** Print the month title, e.g., March 2012 */
  public static void printMonthTitle(int year, int month)
    System.out.println("         " + getMonthName(month)
      + " " + year);
    System.out.println("-----------------------------");
    System.out.println(" Sun Mon Tue Wed Thu Fri Sat");
  }
```

Abstraction/Encapsulation

Ch 6

Hiding



optional I/O

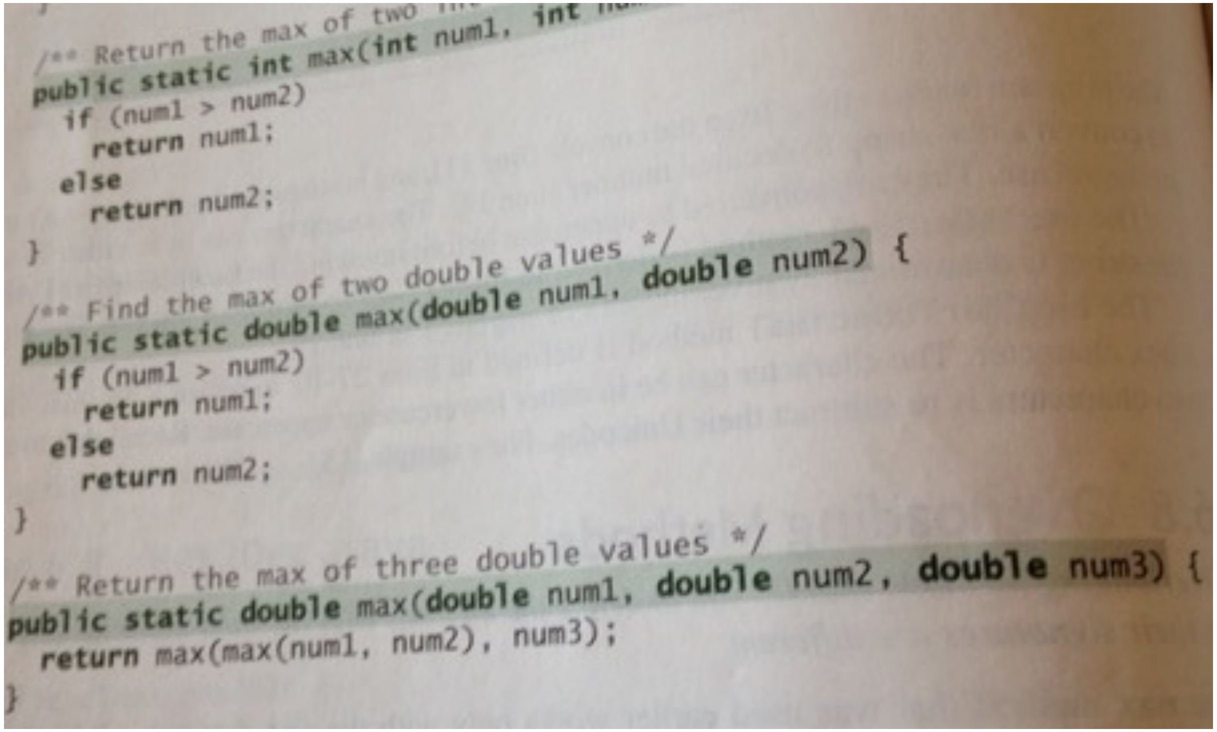
Black Box

Methods: *Overloading*

Multiple Instances with DIFFERENT SIGNATURES

<type> <name> (parm list)

int myMeth( )
int myMeth(int p1)
int myMeth(int p1, int p2)
int myMeth(float p1, float p2)



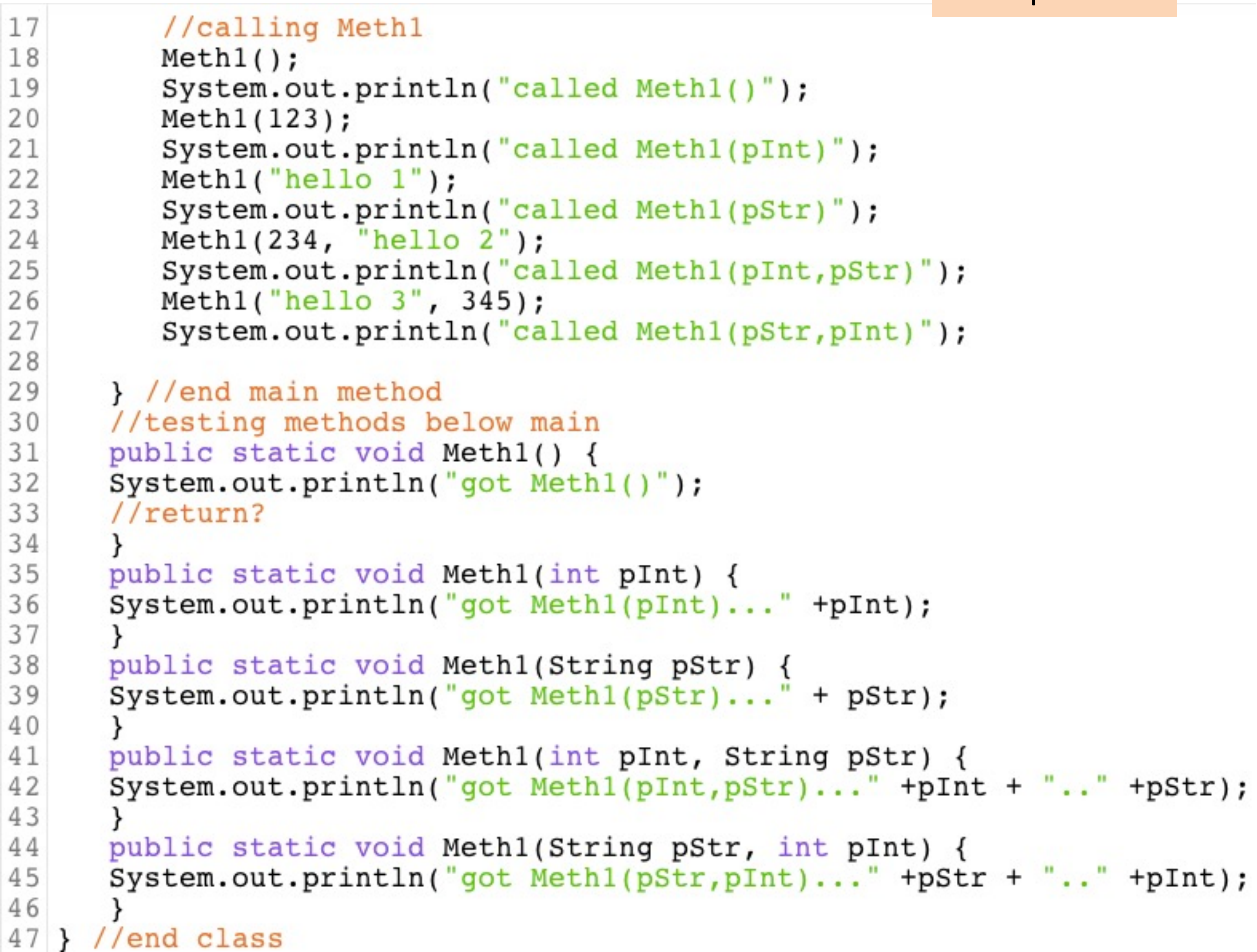
```
/** Return the max of two in
public static int max(int num1, int nu
  if (num1 > num2)
    return num1;
  else
    return num2;
}

/** Find the max of two double values */
public static double max(double num1, double num2) {
  if (num1 > num2)
    return num1;
  else
    return num2;
}

/** Return the max of three double values */
public static double max(double num1, double num2, double num3) {
  return max(max(num1, num2), num3);
}
```

CSUN
CALIFORNIA
STATE UNIVERSITY
NORTHRIDGE
COMP110

Methods: *Overloading*

DR JEFF
SOFTWARE
INDIE APP DEVELOPER
©2016-2022
Jeff Drobman

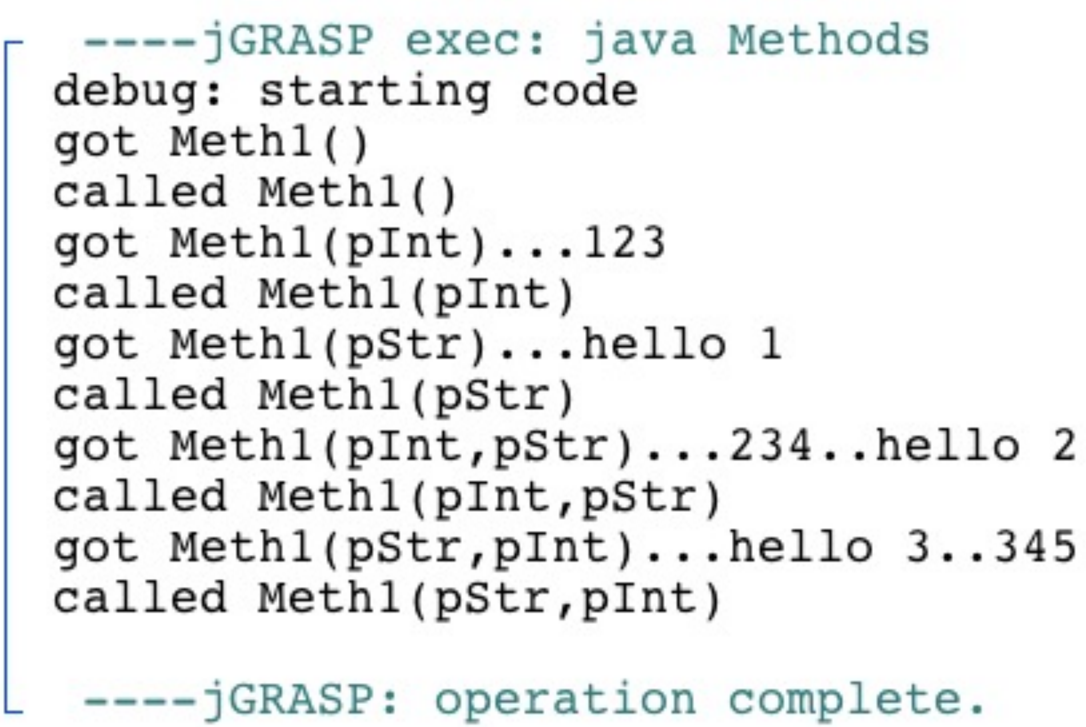
Example code



```
      //calling Meth1
      Meth1();
      System.out.println("called Meth1()");
      Meth1(123);
      System.out.println("called Meth1(pInt)");
      Meth1("hello 1");
      System.out.println("called Meth1(pStr)");
      Meth1(234, "hello 2");
      System.out.println("called Meth1(pInt,pStr)");
      Meth1("hello 3", 345);
      System.out.println("called Meth1(pStr,pInt)");

   } //end main method
   //testing methods below main
   public static void Meth1() {
   System.out.println("got Meth1()");
   //return?
   }
   public static void Meth1(int pInt) {
   System.out.println("got Meth1(pInt)..." +pInt);
   }
   public static void Meth1(String pStr) {
   System.out.println("got Meth1(pStr)..." + pStr);
   }
   public static void Meth1(int pInt, String pStr) {
   System.out.println("got Meth1(pInt,pStr)..." +pInt + ".." +pStr);
   }
   public static void Meth1(String pStr, int pInt) {
   System.out.println("got Meth1(pStr,pInt)..." +pStr + ".." +pInt);
   }
} //end class
```

Methods: *Overloading*

Example code

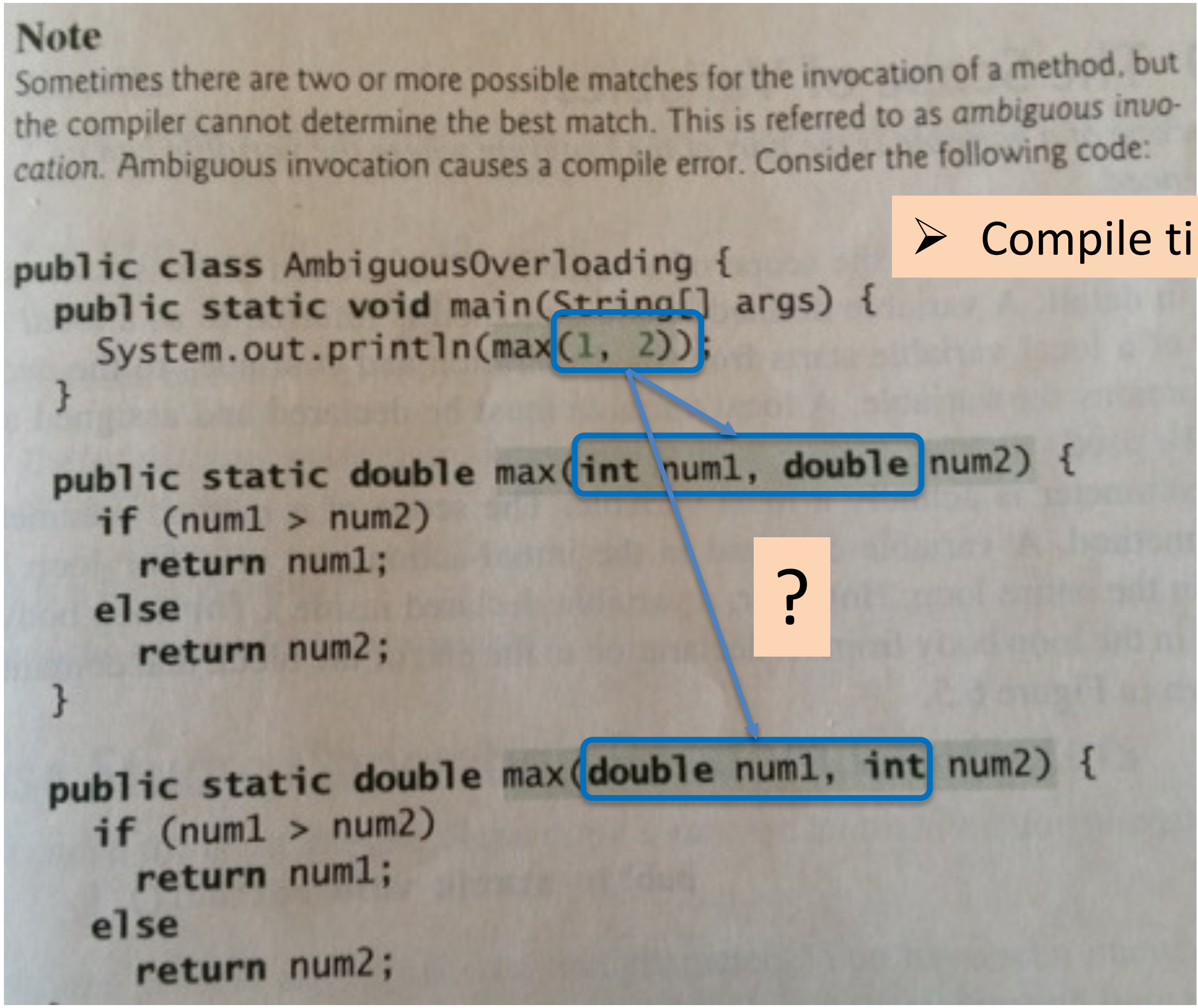


```
 ----jGRASP exec: java Methods
debug: starting code
got Meth1()
called Meth1()
got Meth1(pInt)...123
called Meth1(pInt)
got Meth1(pStr)...hello 1
called Meth1(pStr)
got Meth1(pInt,pStr)...234..hello 2
called Meth1(pInt,pStr)
got Meth1(pStr,pInt)...hello 3..345
called Meth1(pStr,pInt)

 ----jGRASP: operation complete.
```

Methods: Overloading

Ch 6

Ambiguous Overloading



Note

Sometimes there are two or more possible matches for the invocation of a method, but the compiler cannot determine the best match. This is referred to as *ambiguous invocation*. Ambiguous invocation causes a compile error. Consider the following code:

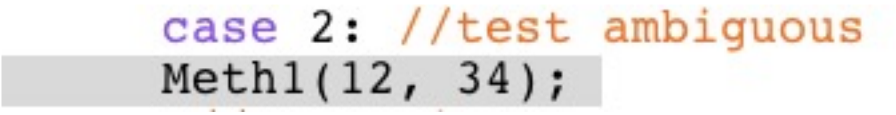
```
public class AmbiguousOverloading {
  public static void main(String[] args) {
    System.out.println(max(1, 2));
  }

  public static double max(int num1, double num2) {
    if (num1 > num2)
      return num1;
    else
      return num2;
  }

  public static double max(double num1, int num2) {
    if (num1 > num2)
      return num1;
    else
      return num2;
```

?

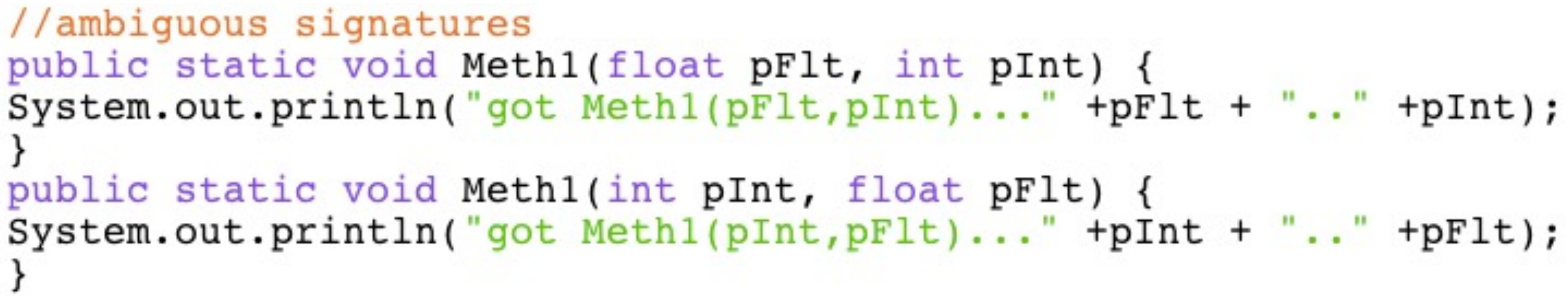
- Compile time error

Ambiguous Overloading

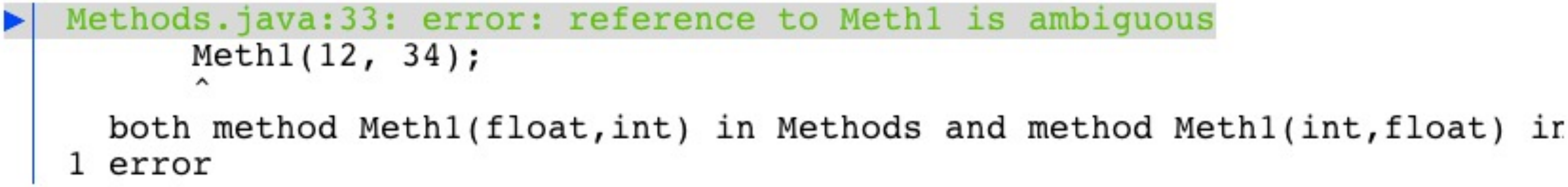
Ambiguous Example



```
     case 2: //test ambiguous
     Meth1(12, 34);
```



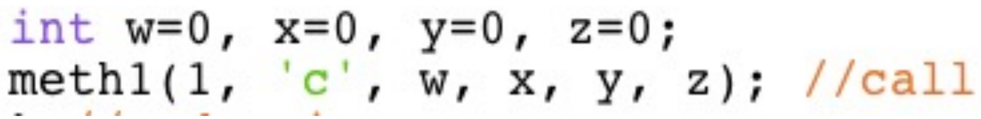
```
//ambiguous signatures
public static void Meth1(float pFlt, int pInt) {
System.out.println("got Meth1(pFlt,pInt)..." +pFlt + ".." +pInt);
}
public static void Meth1(int pInt, float pFlt) {
System.out.println("got Meth1(pInt,pFlt)..." +pInt + ".." +pFlt);
}
```



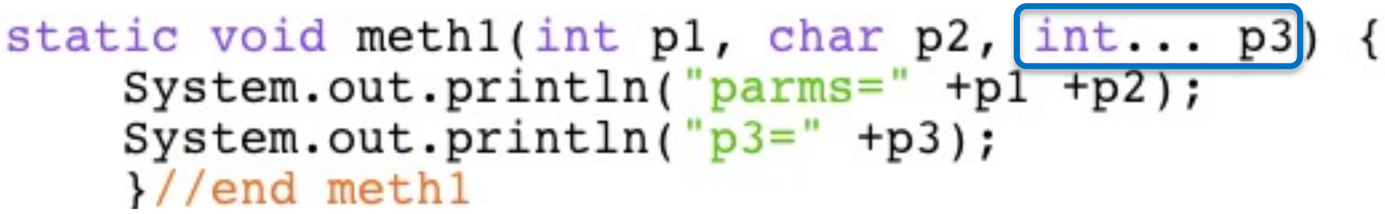
```
Methods.java:33: error: reference to Meth1 is ambiguous
        Meth1(12, 34);
        ^
  both method Meth1(float,int) in Methods and method Meth1(int,float) ir
1 error
```

Parameter Overloading

Parameter Overloading



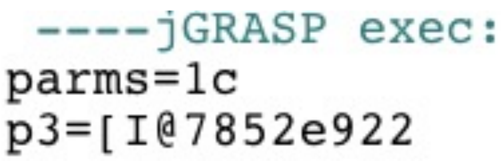
```
int w=0, x=0, y=0, z=0;
meth1(1, 'c', w, x, y, z); //call
```



```
static void meth1(int p1, char p2, int... p3) {
    System.out.println("parms=" +p1 +p2);
    System.out.println("p3=" +p3);
}//end meth1
```

static void meth1(int p1, char p2, int**... p3**) {

→ produces array ”p3” as int[4] with values passed as w=0, x=0, y=0, z=0

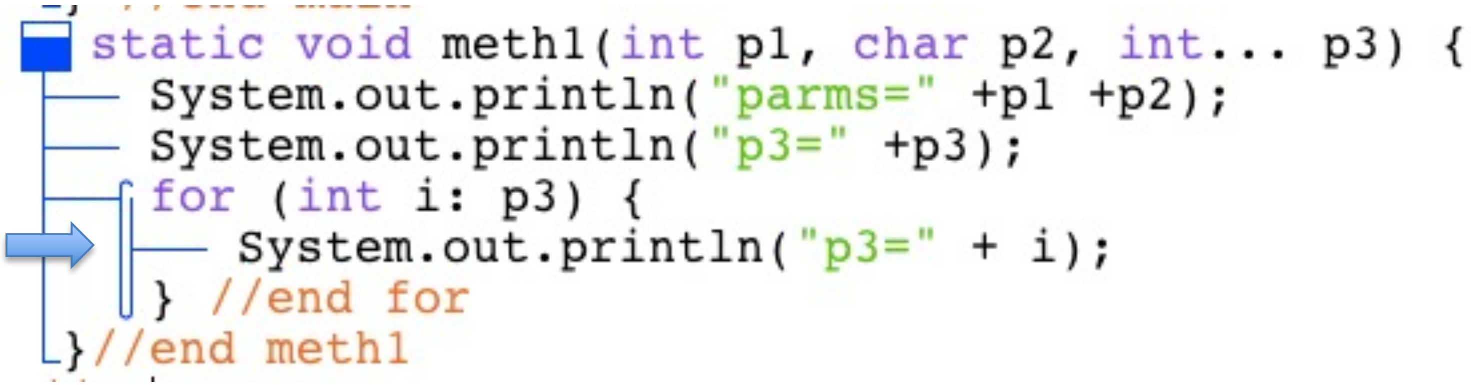


array ptr

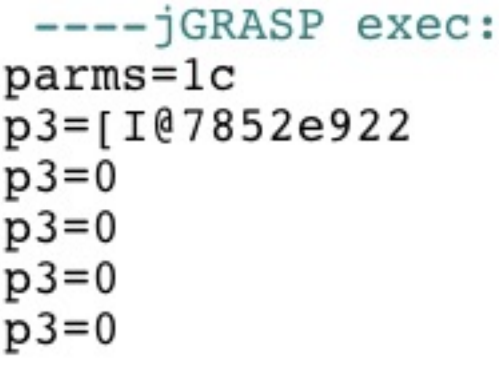
```
 ----jGRASP exec:
parms=1c
p3=[I@7852e922
```

Parameter Overloading

Parameter Overloading: try again



```
static void meth1(int p1, char p2, int... p3) {
   System.out.println("parms=" +p1 +p2);
   System.out.println("p3=" +p3);
   for (int i: p3) {
      System.out.println("p3=" + i);
   } //end for
}//end meth1
```



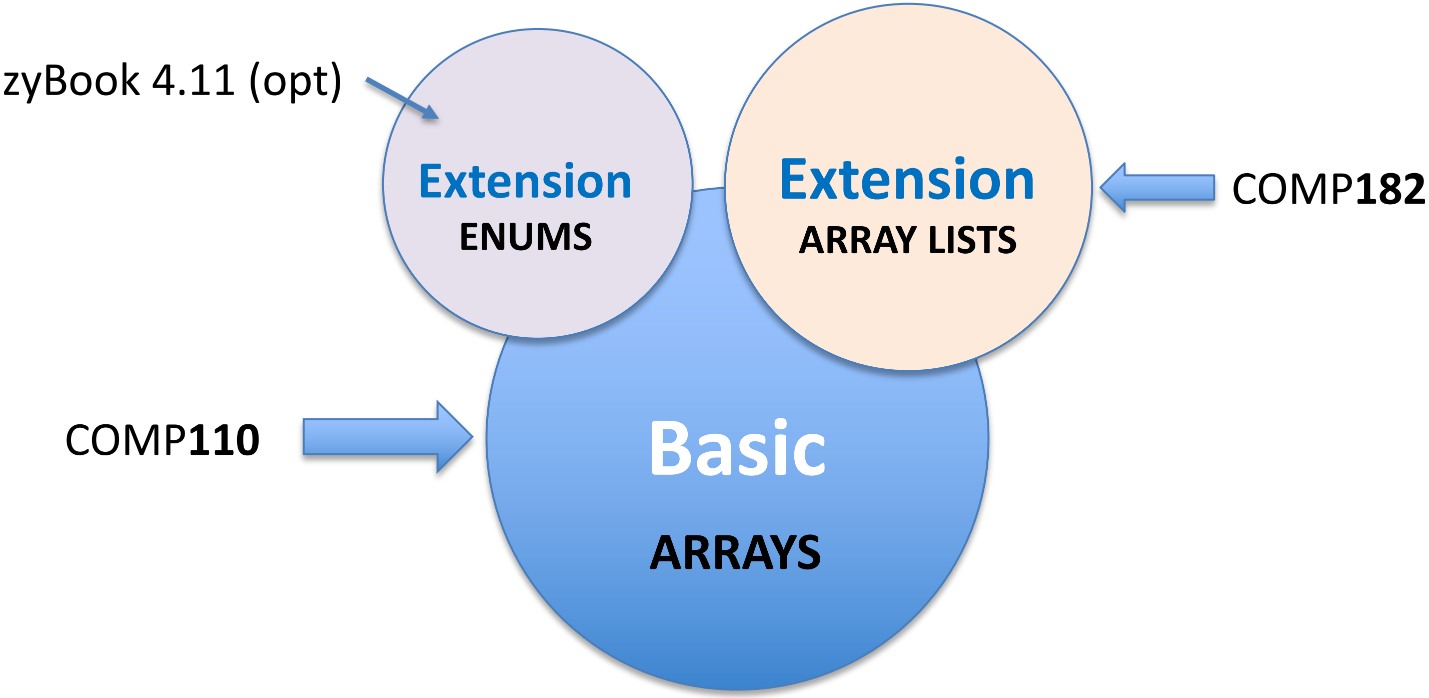
```
 ----jGRASP exec:
parms=1c
p3=[I@7852e922
p3=0
p3=0
p3=0
p3=0
```

array[4]

Arrays

Arrays

Basics vs. Extensions

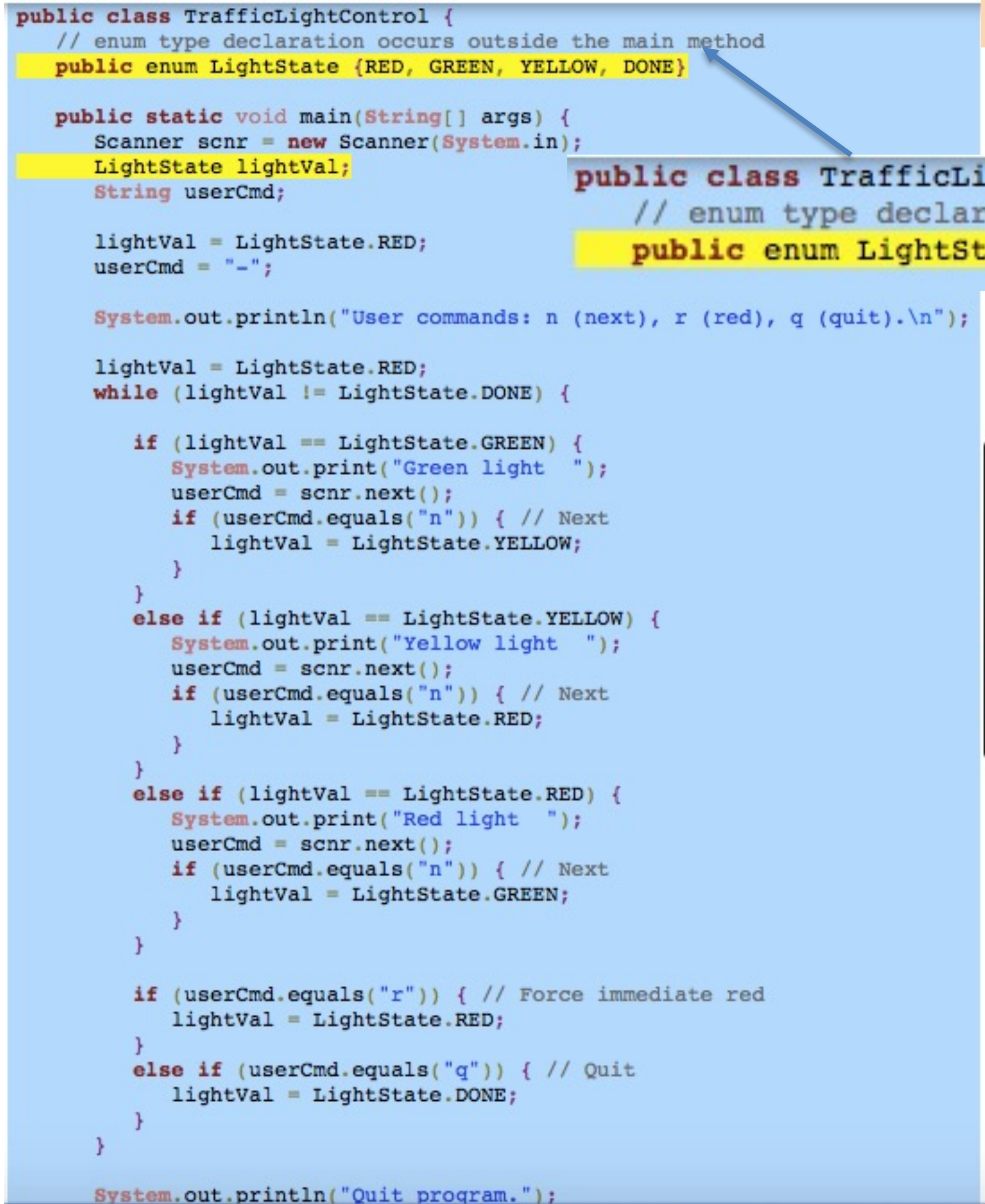


CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP110

Enums

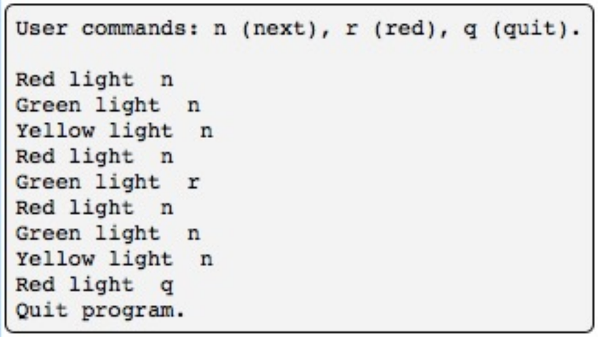
DR JEFF SOFTWARE
INDIE APP DEVELOPER
©2016-2022
Jeff Drobman

“Enumerations”



```
public class TrafficLightControl {
   // enum type declaration occurs outside the main method
   public enum LightState {RED, GREEN, YELLOW, DONE}

   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      LightState lightVal;
      String userCmd;

      lightVal = LightState.RED;
      userCmd = "-";

      System.out.println("User commands: n (next), r (red), q (quit).\n");

      lightVal = LightState.RED;
      while (lightVal != LightState.DONE) {

         if (lightVal == LightState.GREEN) {
            System.out.print("Green light  ");
            userCmd = scnr.next();
            if (userCmd.equals("n")) { // Next
               lightVal = LightState.YELLOW;
            }
         }
         else if (lightVal == LightState.YELLOW) {
            System.out.print("Yellow light  ");
            userCmd = scnr.next();
            if (userCmd.equals("n")) { // Next
               lightVal = LightState.RED;
            }
         }
         else if (lightVal == LightState.RED) {
            System.out.print("Red light  ");
            userCmd = scnr.next();
            if (userCmd.equals("n")) { // Next
               lightVal = LightState.GREEN;
            }
         }

         if (userCmd.equals("r")) { // Force immediate red
            lightVal = LightState.RED;
         }
         else if (userCmd.equals("q")) { // Quit
            lightVal = LightState.DONE;
         }
      }

      System.out.println("Quit program.");
```

```
public class TrafficLightControl {
   // enum type declaration occurs outside the main method
   public enum LightState {RED, GREEN, YELLOW, DONE}
```



```
User commands: n (next), r (red), q (quit).

Red light  n
Green light  n
Yellow light  n
Red light  n
Green light  r
Red light  n
Green light  n
Yellow light  n
Red light  q
Quit program.
```

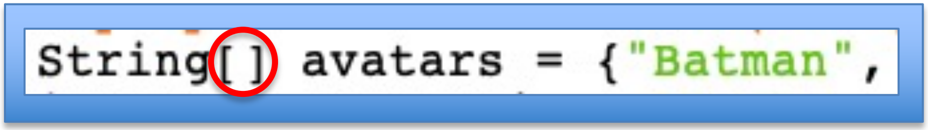
Arrays

- Organized collection ("List") of data (all *same* type)
- ***Indexed*** by Integers (Non-negative: 0..N)
- *Associative* arrays in other languages (e.g., PHP) but not Java
- *Single* Dimension ("Linear" array or "vector")
- *Multi* Dimension
  - ❑ 2-dimensional: "Matrix" or Table (databases)
  - ❑ N-dimensional: abstract

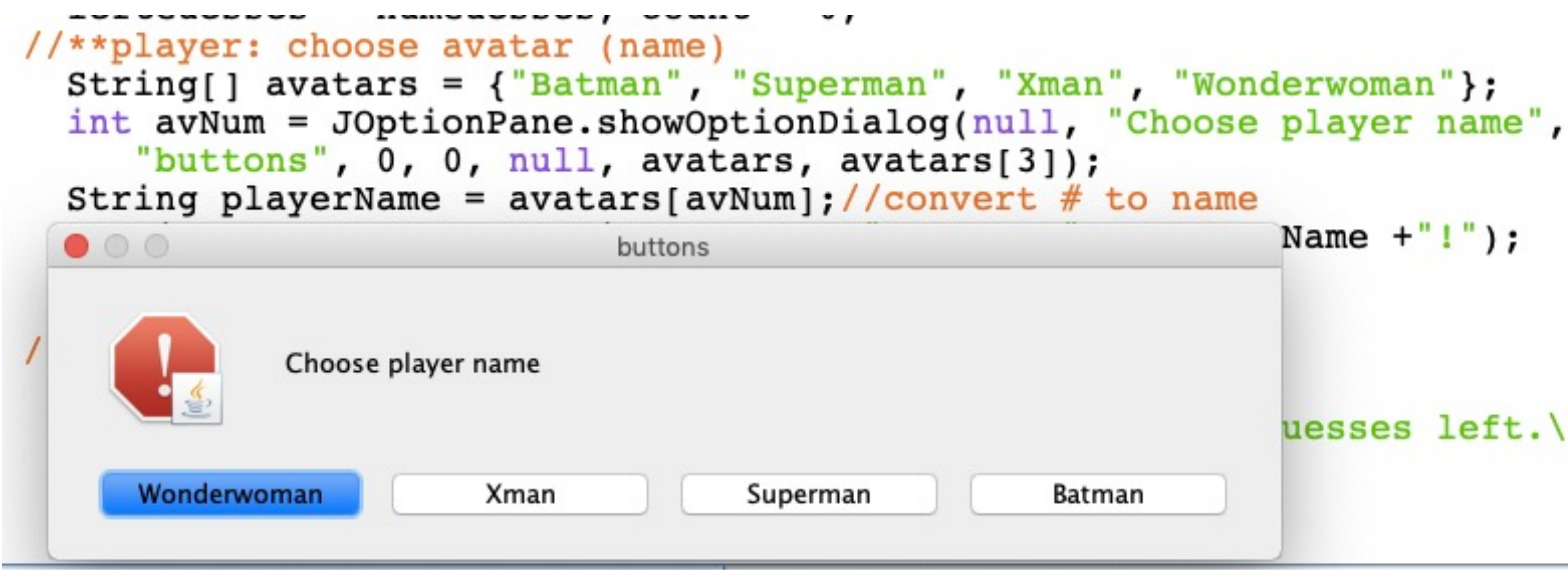
```
int[] myArr = new int[10];
```

```
int[] myArr = {3, 21, 0, 16, 0, 1};
```

Arrays



```
String[] avatars = {"Batman",
```



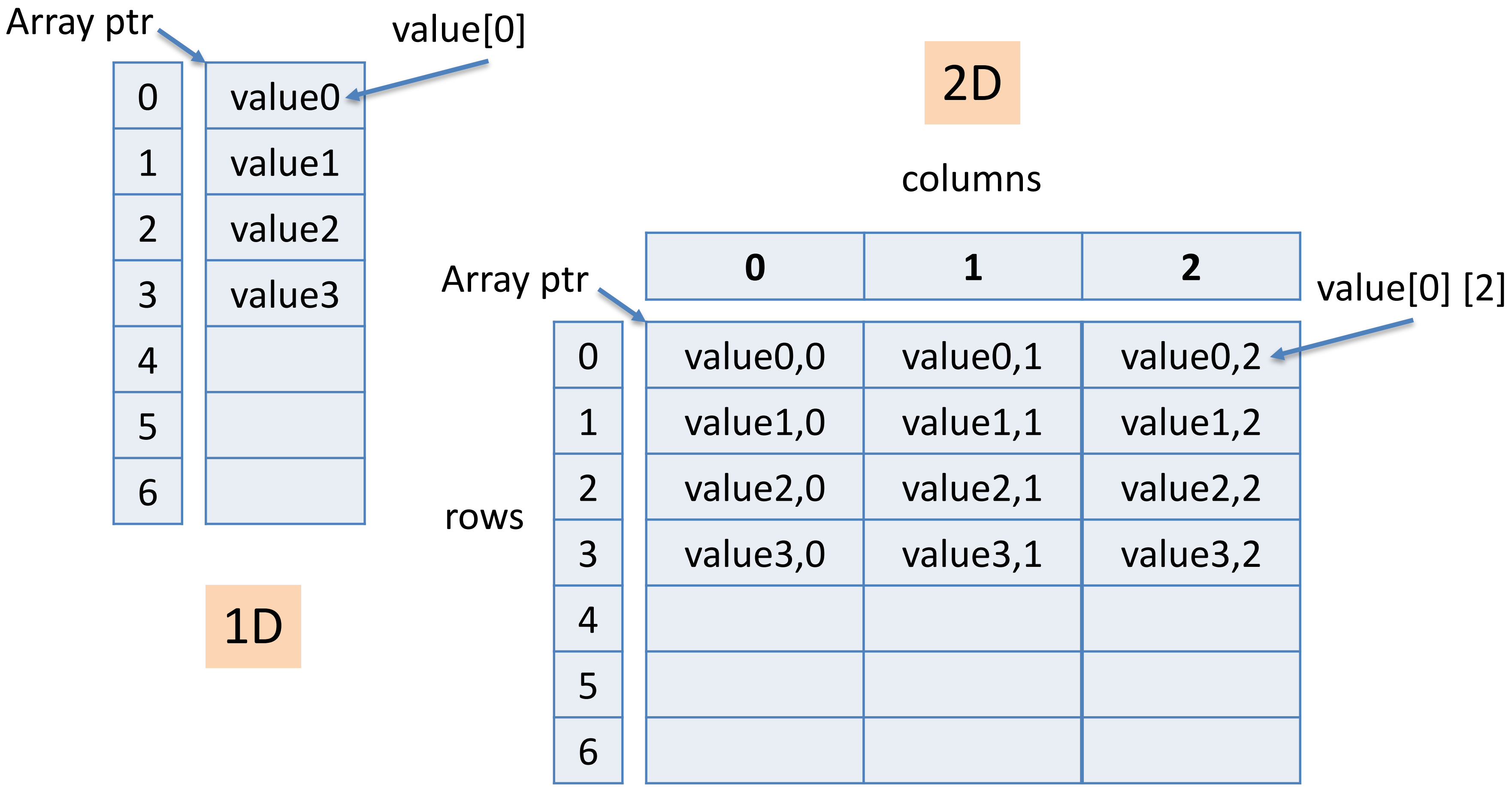
```
//**player: choose avatar (name)
  String[] avatars = {"Batman", "Superman", "Xman", "Wonderwoman"};
  int avNum = JOptionPane.showOptionDialog(null, "Choose player name",
     "buttons", 0, 0, null, avatars, avatars[3]);
  String playerName = avatars[avNum];//convert # to name
                                                       Name +"!");

/

                                                       uesses left.\
```

buttons

Choose player name

Arrays

❖ Indexed

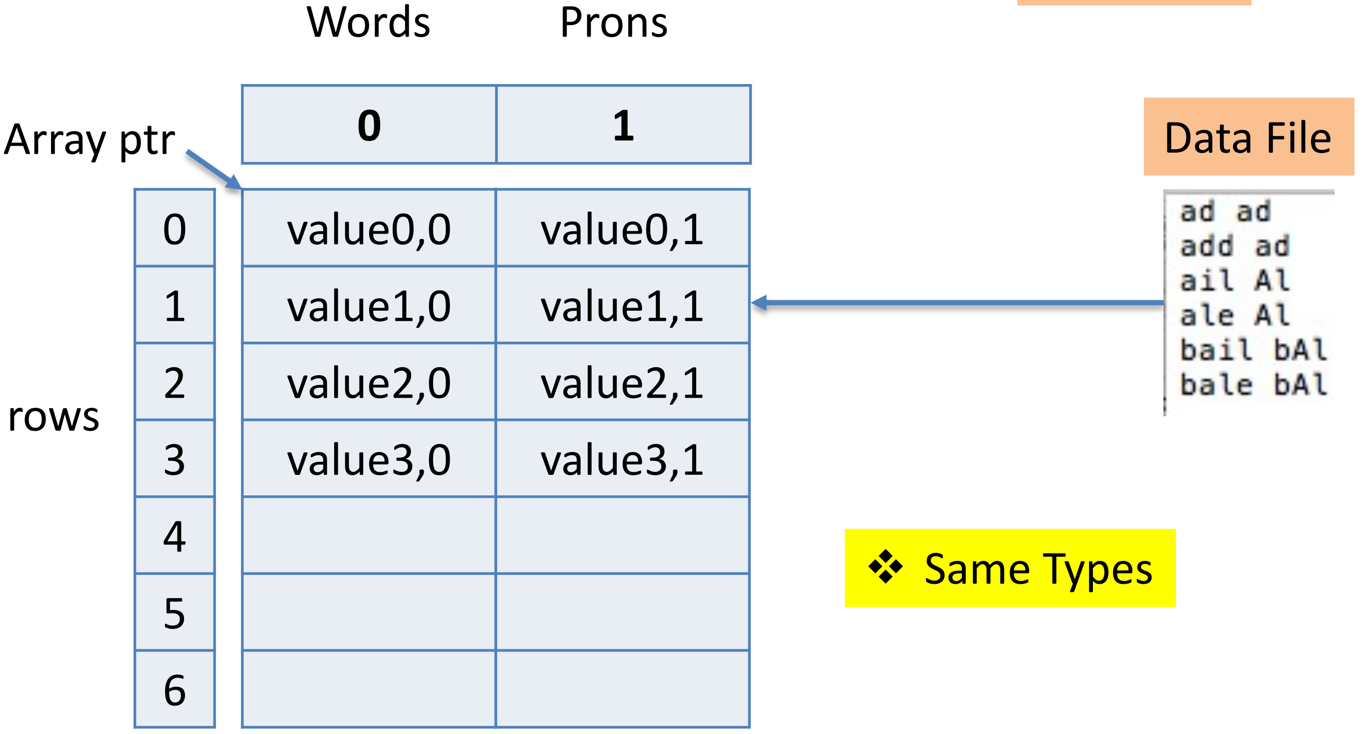


0	value0
1	value1
2	value2
3	value3
4	
5	
6	

	0	1	2
0	value0,0	value0,1	value0,2
1	value1,0	value1,1	value1,2
2	value2,0	value2,1	value2,2
3	value3,0	value3,1	value3,2
4			
5			
6			

Arrays

2D option



Array ptr

rows

	Words	Prons
	0	**1**
0	value0,0	value0,1
1	value1,0	value1,1
2	value2,0	value2,1
3	value3,0	value3,1
4		
5		
6		

Data File

```
ad  ad
add ad
ail Al
ale Al
bail bAl
bale bAl
```

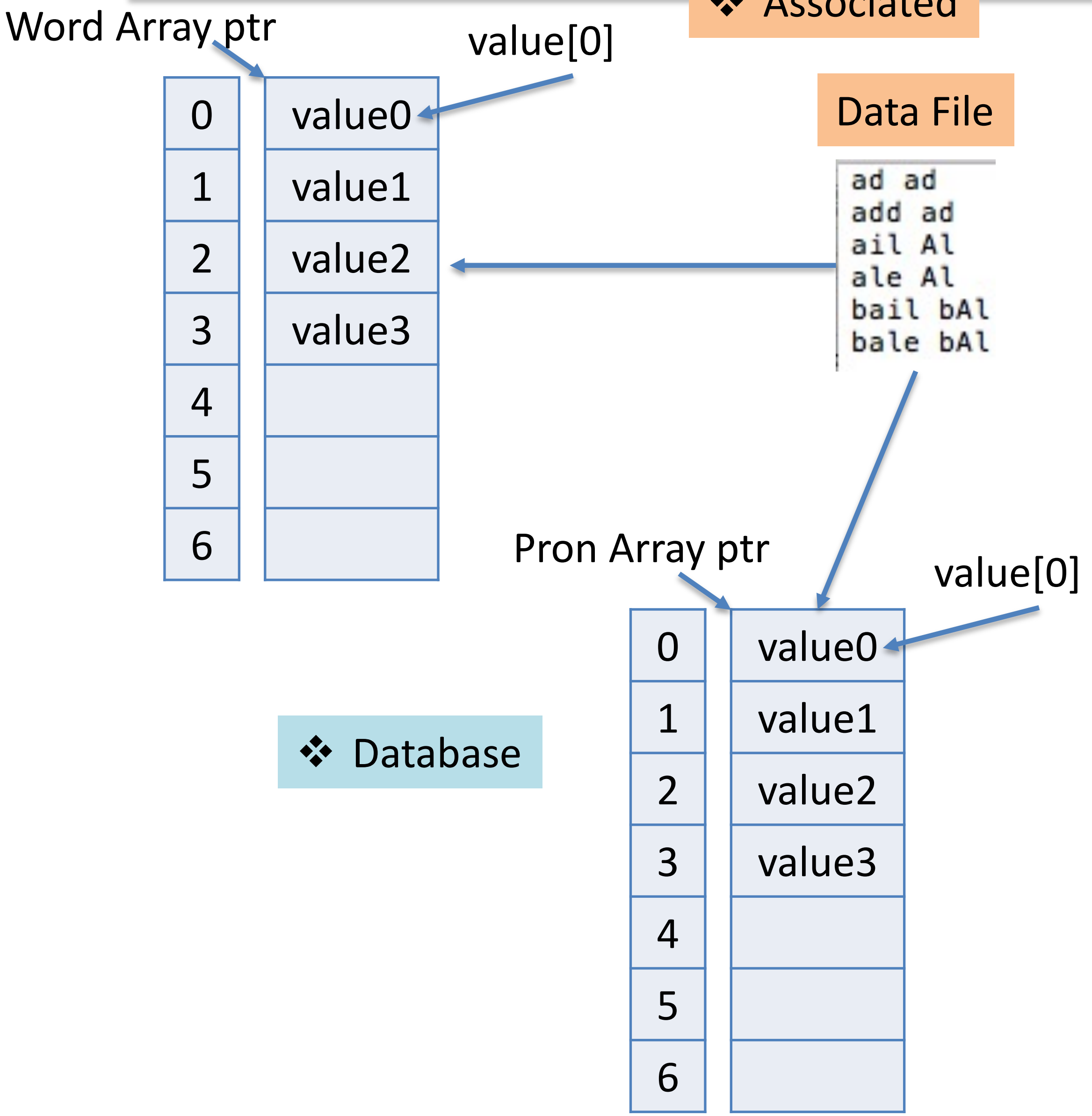
- ❖ Same Types

- ❖ Database

Arrays

❖ Associated

2x 1D option



Data File

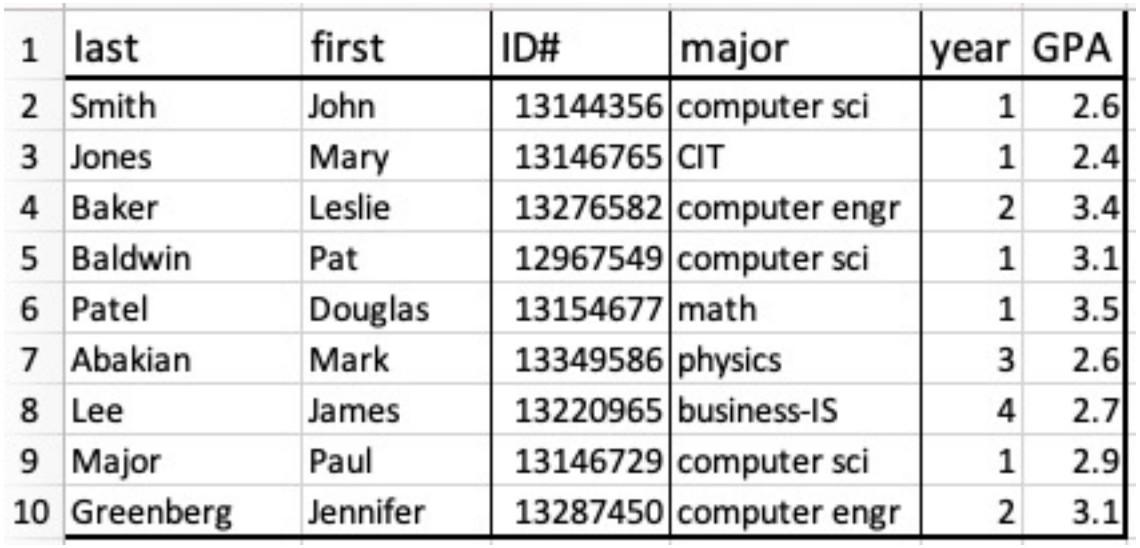
```
ad ad
add ad
ail Al
ale Al
bail bAl
bale bAl
```

❖ Dif Types (can be)

❖ Database

Arrays <-> Databases

```
String[][] database = new String [9][6];
```



1	last	first	ID#	major	year	GPA
2	Smith	John	13144356	computer sci	1	2.6
3	Jones	Mary	13146765	CIT	1	2.4
4	Baker	Leslie	13276582	computer engr	2	3.4
5	Baldwin	Pat	12967549	computer sci	1	3.1
6	Patel	Douglas	13154677	math	1	3.5
7	Abakian	Mark	13349586	physics	3	2.6
8	Lee	James	13220965	business-IS	4	2.7
9	Major	Paul	13146729	computer sci	1	2.9
10	Greenberg	Jennifer	13287450	computer engr	2	3.1

Data-Arrays/Blocks

❖ Arrays

- ❑ Assembly
  - DW A'0123456789'
- ❑ C
  - unsigned char strarr[10] = {'0','1','2','3','4','5','6','7','8','9'}
  - charx = strarr[n] //use array element
- ❑ C++
  - int arr[ ] = {0,1,2,3,4,5,6,7,8,9}
- ❑ Java
  - int [ ] arr = {0,1,2,3,4,5,6,7,8,9}
  - int **[ ] [ ]** arrRag = **{** {0,1,2,3,4},
    {5,6,7,8},
    {9,0} **}**;

❖ Java only: ***Ragged***

❖ Block Constructs (C/C++)

- ❑ Structures
- ❑ Unions

Arrays in Java

Declare a “pointer” (reference) to an array:

```
<type>[] <array name>;
```

or use this “C” type:

```
<type> <array name>[];
```

To allocate memory for, and thus be able to use, an array, one must *instantiate* the array:

```
<array name> = new <type>[<size>];
```

❖ Rarely used

❖ Use this form

one may ***combine*** the array *declaration* and *instantiation*:

```
<type>[] <array name> = new <type>[<size>];
```

Example:

```
int[] myArr = new int[10];
```

Initializing Arrays

Declare/Instantiate Example:

```
int[] myArr = new int[10];
```

(0..9)

Declare/Instantiate & *Initialize* Example:

```
int[] myArr = {3, 21, 0, 16, 0, 1};
```

(0..5) → 6

Fill Examples:

```
int[] myArr = new int[10];
for(i=0; i<10; i++) {
    myArr[i] = i;
}
```

{0..9}

❖ ***Foreach***

```
int[] myArr = new int[10];
for(int ax: myArr) {
    ax = n;
}
```

or use **i++** → {0..9}

Using Arrays

Declare/Instantiate Example:

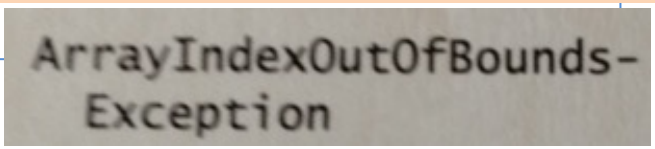
```
int[] myArr = new int[10];
```

(0..9)

Reference Example:

```
int x = myArr[10];
```

Error: array subscript out of bounds



ArrayIndexOutOfBounds-Exception

Oversized Example:

```
int[] myArr = {-1, 21,0,16,0,1, -1,-1,
-1,-1};
```

-OR after init-

❖ Marking *uninitialized* elements

```
for(i=6; i<10; i++) {myArr[i] = -1;}
```

Sizing Arrays

Perfect size

```
int[] myArr = new int[10];
```

OVER size

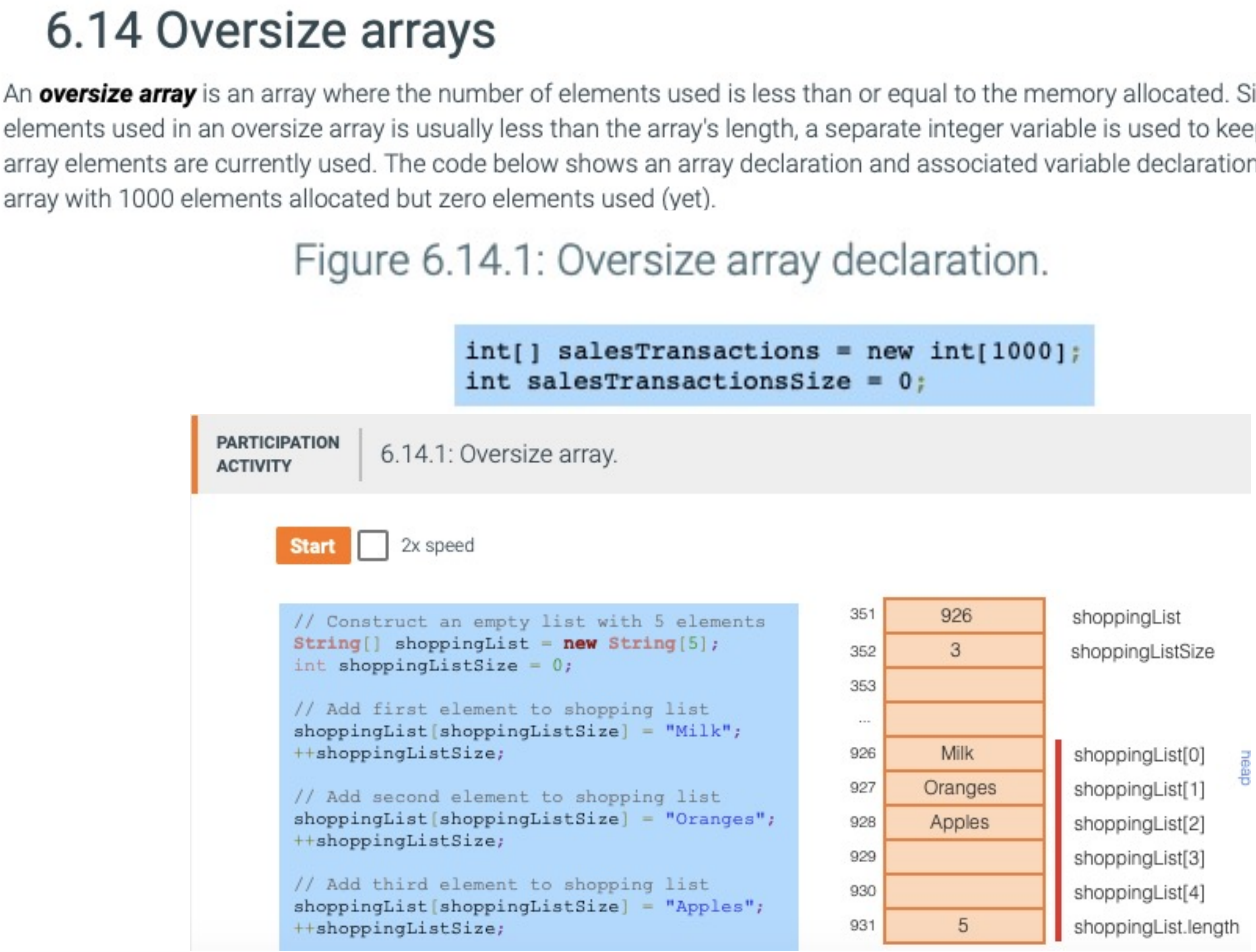
```
int size = 0;
int max = 1000;
int[] myArr = new int[max];
size++;
```

CSUN
CALIFORNIA
STATE UNIVERSITY
NORTHRIDGE

COMP110

Oversizing

DR JEFF
SOFTWARE
INDIE APP DEVELOPER
©2016-2022
Jeff Drobman



6.14 Oversize arrays

An ***oversize array*** is an array where the number of elements used is less than or equal to the memory allocated. Since the number of elements used in an oversize array is usually less than the array's length, a separate integer variable is used to keep track of how many array elements are currently used. The code below shows an array declaration and associated variable declaration that create an oversize array with 1000 elements allocated but zero elements used (yet).

Figure 6.14.1: Oversize array declaration.

```
int[] salesTransactions = new int[1000];
int salesTransactionsSize = 0;
```

PARTICIPATION ACTIVITY 6.14.1: Oversize array.

Start □ 2x speed

```
// Construct an empty list with 5 elements
String[] shoppingList = new String[5];
int shoppingListSize = 0;

// Add first element to shopping list
shoppingList[shoppingListSize] = "Milk";
++shoppingListSize;

// Add second element to shopping list
shoppingList[shoppingListSize] = "Oranges";
++shoppingListSize;

// Add third element to shopping list
shoppingList[shoppingListSize] = "Apples";
++shoppingListSize;
```

351	926	shoppingList
352	3	shoppingListSize
353		
...		
926	Milk	shoppingList[0]
927	Oranges	shoppingList[1]
928	Apples	shoppingList[2]
929		shoppingList[3]
930		shoppingList[4]
931	5	shoppingList.length

CSUN
CALIFORNIA
STATE UNIVERSITY
NORTHRIDGE

COMP110

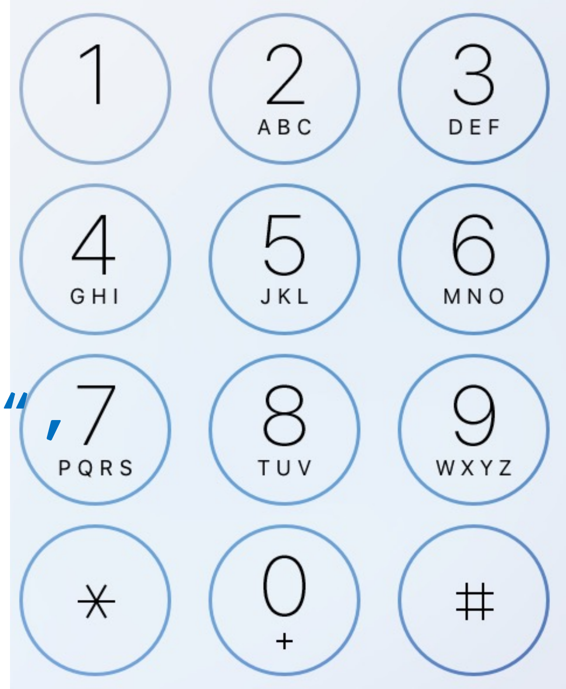
Array Example

DR JEFF SOFTWARE
INDIE APP DEVELOPER
©2016-2022
eff Drobman

find keys

```
char key, int[] numArr = new int[10]; //inits
for(i=0; i<10; i++) {numArr[i] = i;}
String[] stArr = {“+”,””,”abc”,”def”,”ghi”,
“jkl”,”mno”,”pqrs”,”tuv”,”wxyz”};
//find key
int ix = -1;
for(i=0; i<10; i++) {
   if stArr[i].indexOf(key) >=0 {
      ix = i; //found it
      break;} }
//print key
if (ix >0 && ix <10)
system.out.printf(“letter %1d=%4s“,
numArr[ix],stArr[ix]);
else
   system.out.println(“bad key”);
```

{0..9}



Strings vs. Arrays

- ❖ Strings
- ❖ Arrays

```
String alpha = “abcdefghijklmnopqrstuvwxyz”;
```

```
char[] alphaArr = {‘a’, ‘b’, ‘c’, ‘d’, ‘e’,
‘f’, …, ‘z’};
```

➢ Or `alpha.toCharArray()`

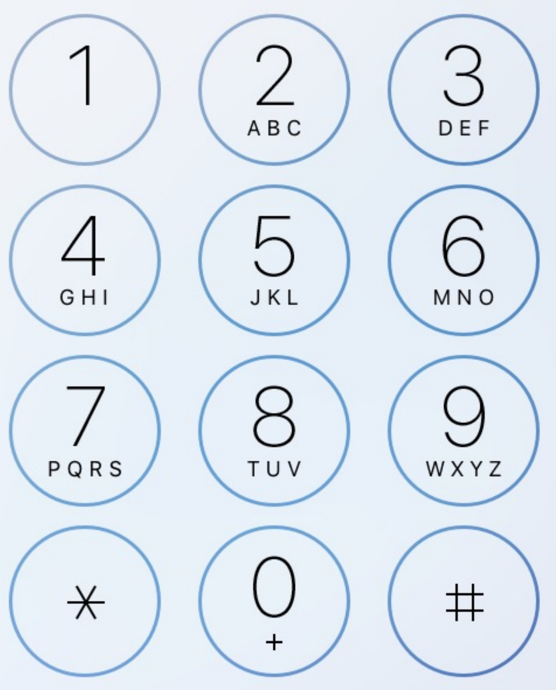
```
int ix = alpha.indexOf(chx); //letter number
if(ix< 0)
--or--
int ix = chx – ’a’; //’a’ = \u0061
if(ix < 0 || ix > 25)
 JOptionPane.showMessageDialog(null,“Error not a-z!”); //check
```

```
for (i=0; i<26; i++) {//searching array
  if (chx = alphaArr[i]) {
    ix = i;
    break;
}
```

Array Example Revisited

find keys

```
String alpha = “+\s\s \s\s\s abc def ghi jkl
mno pqrstuv wxyz”;

//find key
int ix = alpha.indexOf(key)/4;
   if ix >0 ix++;

//print key -- same
```

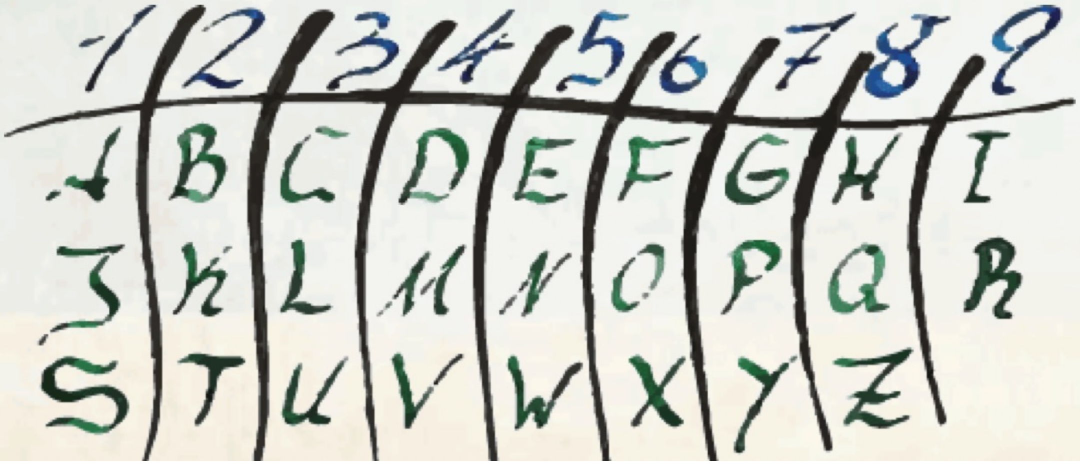


Pythagorean Alphabet

Arrays-Strings Example

Interesting Fact: The famous Renaissance Doctor, translator and Astrologer, Marsilio Ficino, devoted much of his life to studying the incredible ways one's life improves when he or she follows the promptings of the soul. The benefits he encountered were undeniable.

Here is an example of how to calculate your Soul Urge Number using the Pythagorean Alphabet:



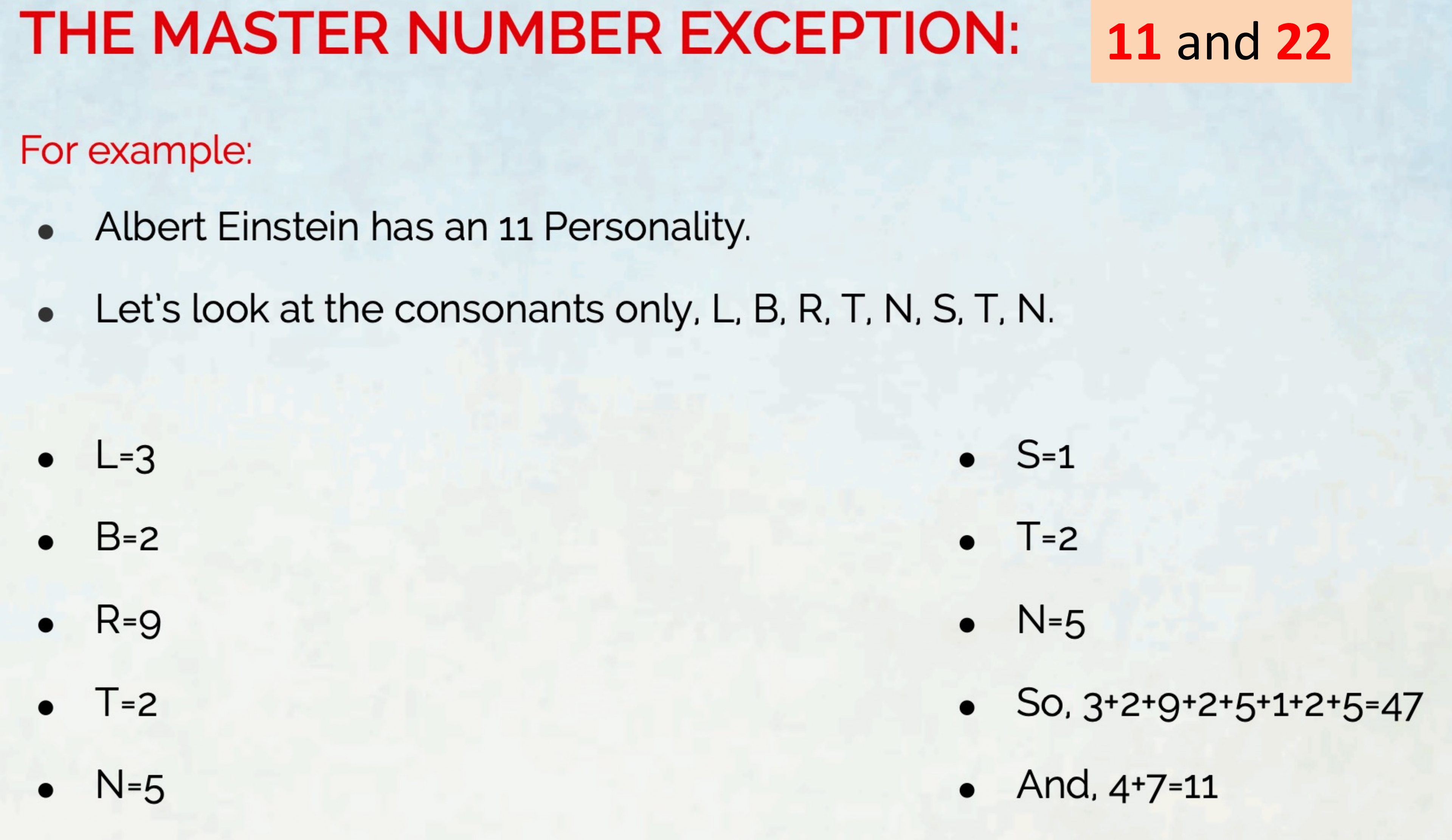
1	2	3	4	5	6	7	8	9
A	B	C	D	E	F	G	H	I
J	K	L	M	N	O	P	Q	R
S	T	U	V	W	X	Y	Z	

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP110

Pythagorean Alphabet

Arrays-Strings Example

DR JEFF SOFTWARE
INDIE APP DEVELOPER
©2016-2022
Jeff Drobman



THE MASTER NUMBER EXCEPTION:

11 and 22

For example:

- Albert Einstein has an 11 Personality.
- Let's look at the consonants only, L, B, R, T, N, S, T, N.

- L=3
- B=2
- R=9
- T=2
- N=5
- S=1
- T=2
- N=5
- So, 3+2+9+2+5+1+2+5=47
- And, 4+7=11

Arrays – Operations

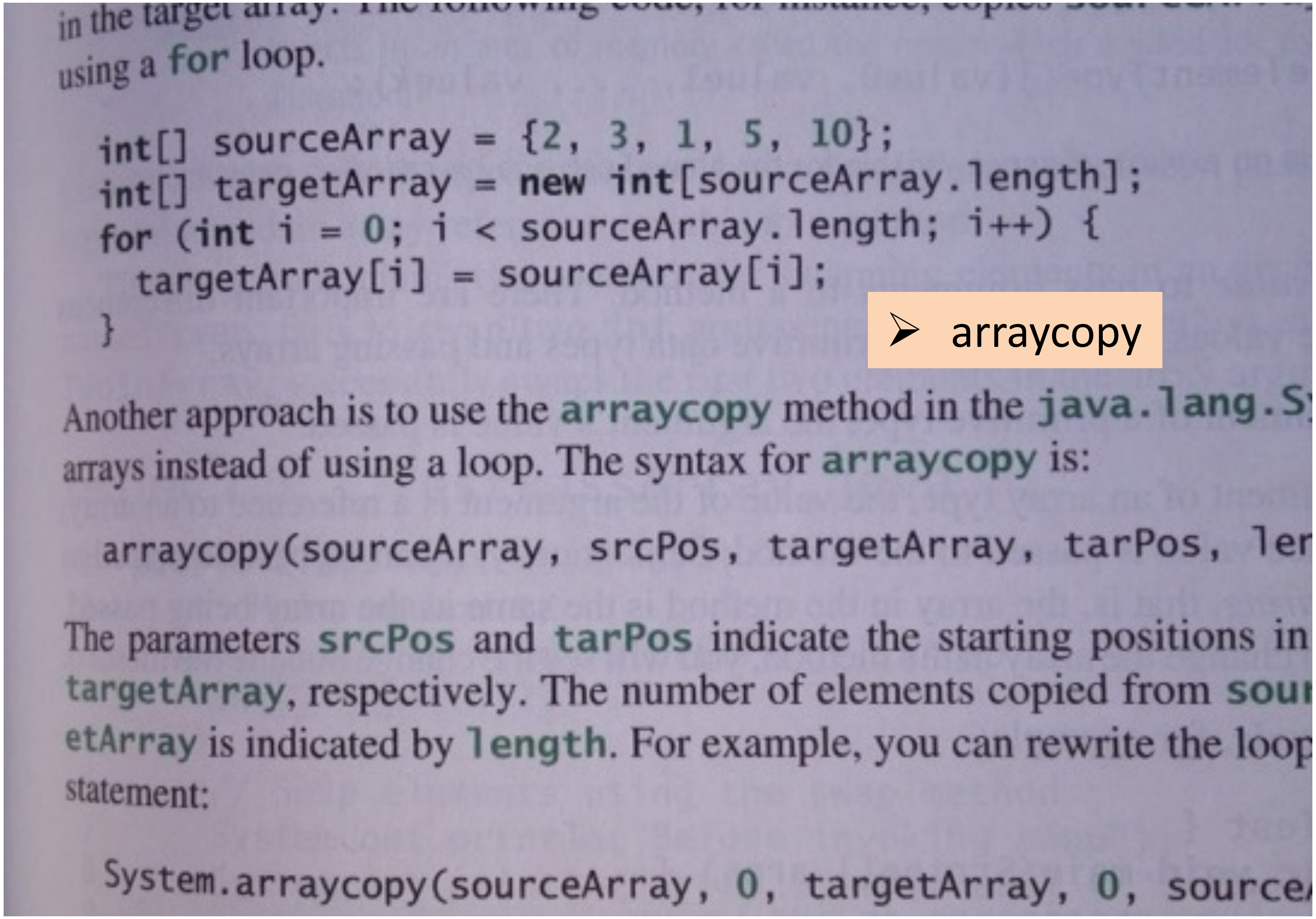
❖Common Array Operations

- ❑ Initialize
- ❑ Compare
- ❑ Copy
- ❑ Shift
- ❑ Reverse
- ❑ Shuffle
- ❑ Parameter Passing
  - ➢ Pass by *Sharing*

➢ using **for** loops

➢ by **Reference**

Arrays: Copy



in the target array. The following code, for instance, copies sourceArr
using a for loop.

```
int[] sourceArray = {2, 3, 1, 5, 10};
int[] targetArray = new int[sourceArray.length];
for (int i = 0; i < sourceArray.length; i++) {
  targetArray[i] = sourceArray[i];
}
```

➢ arraycopy

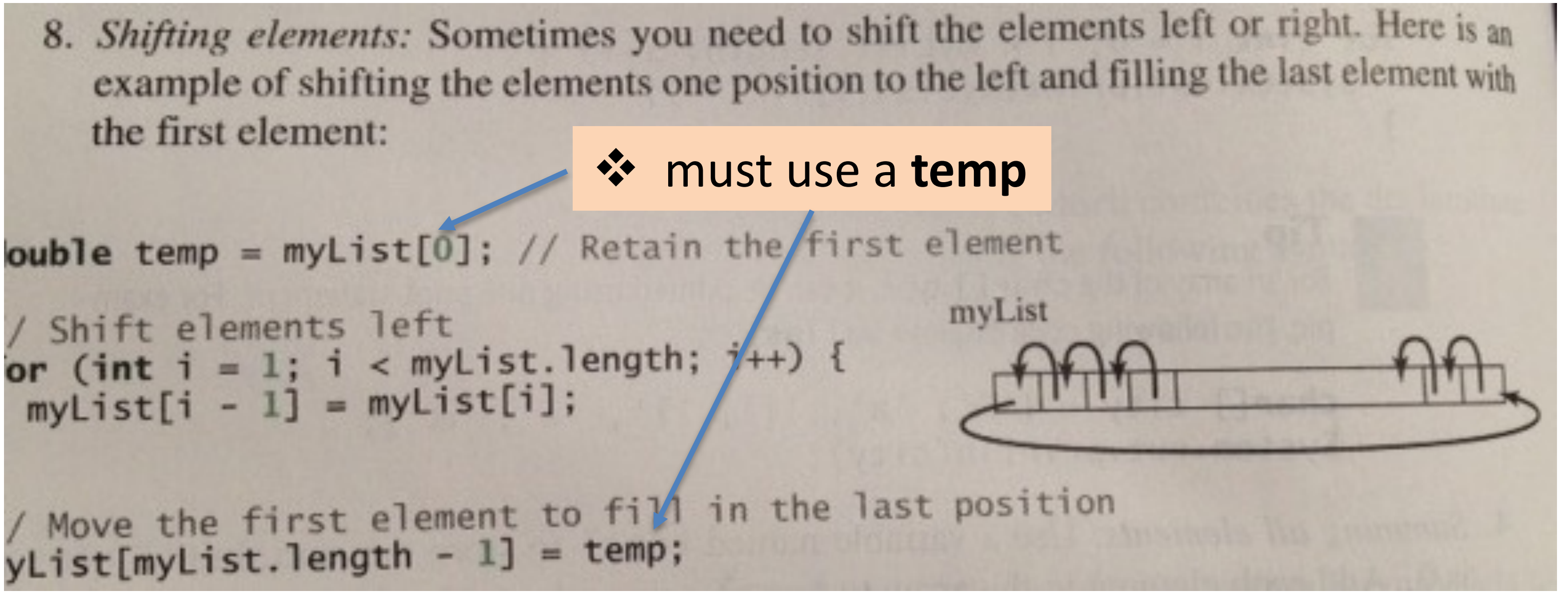
Another approach is to use the arraycopy method in the java.lang.Sy
arrays instead of using a loop. The syntax for arraycopy is:

```
arraycopy(sourceArray, srcPos, targetArray, tarPos, len
```

The parameters srcPos and tarPos indicate the starting positions in
targetArray, respectively. The number of elements copied from sour
etArray is indicated by length. For example, you can rewrite the loop
statement:

```
System.arraycopy(sourceArray, 0, targetArray, 0, source
```

Arrays: Shifting

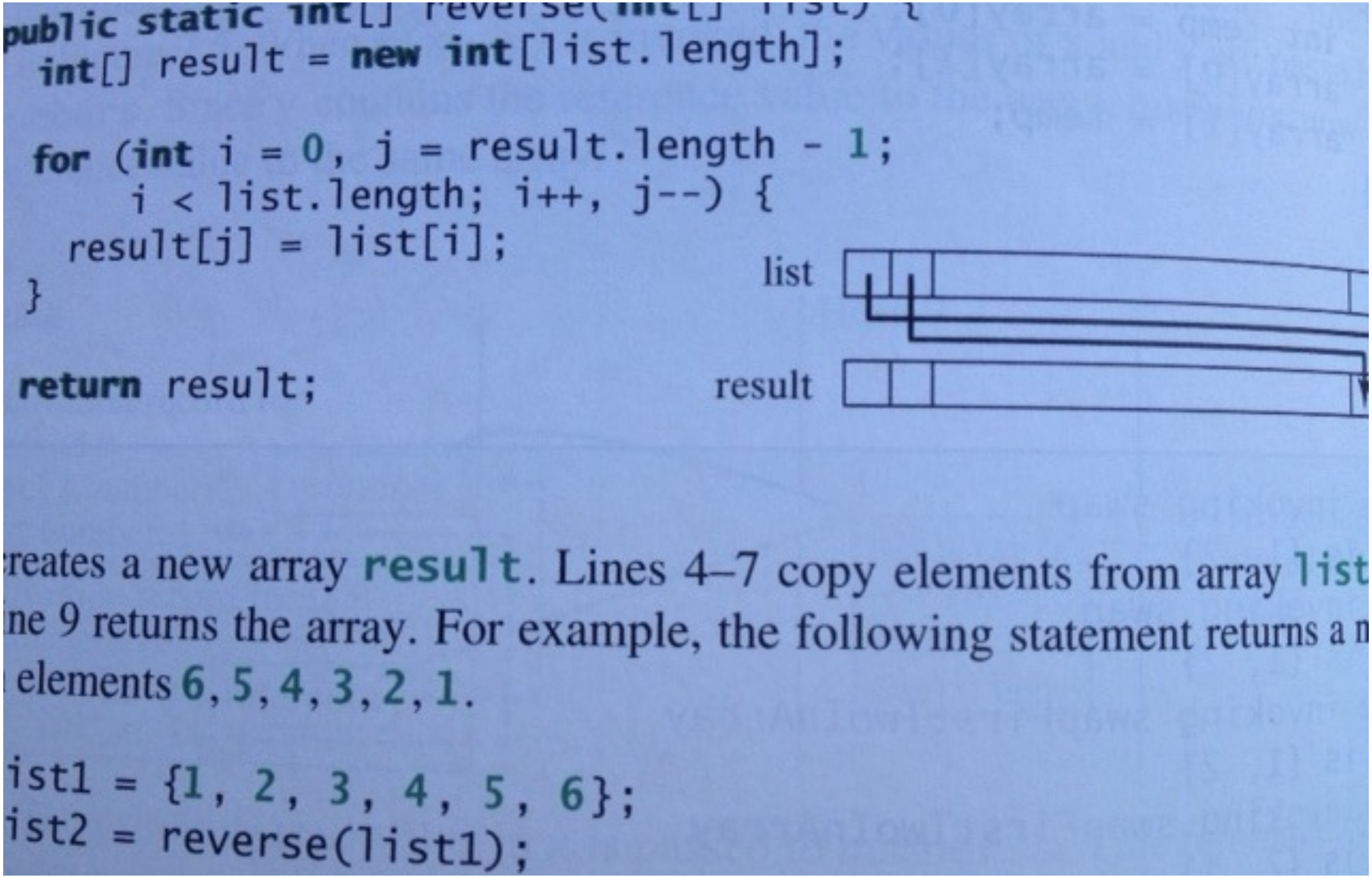


8. *Shifting elements:* Sometimes you need to shift the elements left or right. Here is an example of shifting the elements one position to the left and filling the last element with the first element:

```
ouble temp = myList[0]; // Retain the first element

/ Shift elements left
or (int i = 1; i < myList.length; i++) {
 myList[i - 1] = myList[i];


/ Move the first element to fill in the last position
yList[myList.length - 1] = temp;
```

❖ must use a **temp**

Arrays: Reverse



```
public static int[] reverse(int[] list) {
  int[] result = new int[list.length];

  for (int i = 0, j = result.length - 1;
       i < list.length; i++, j--) {
    result[j] = list[i];
  }

  return result;
```

list
result

reates a new array result. Lines 4–7 copy elements from array list
ne 9 returns the array. For example, the following statement returns a n
elements 6, 5, 4, 3, 2, 1.

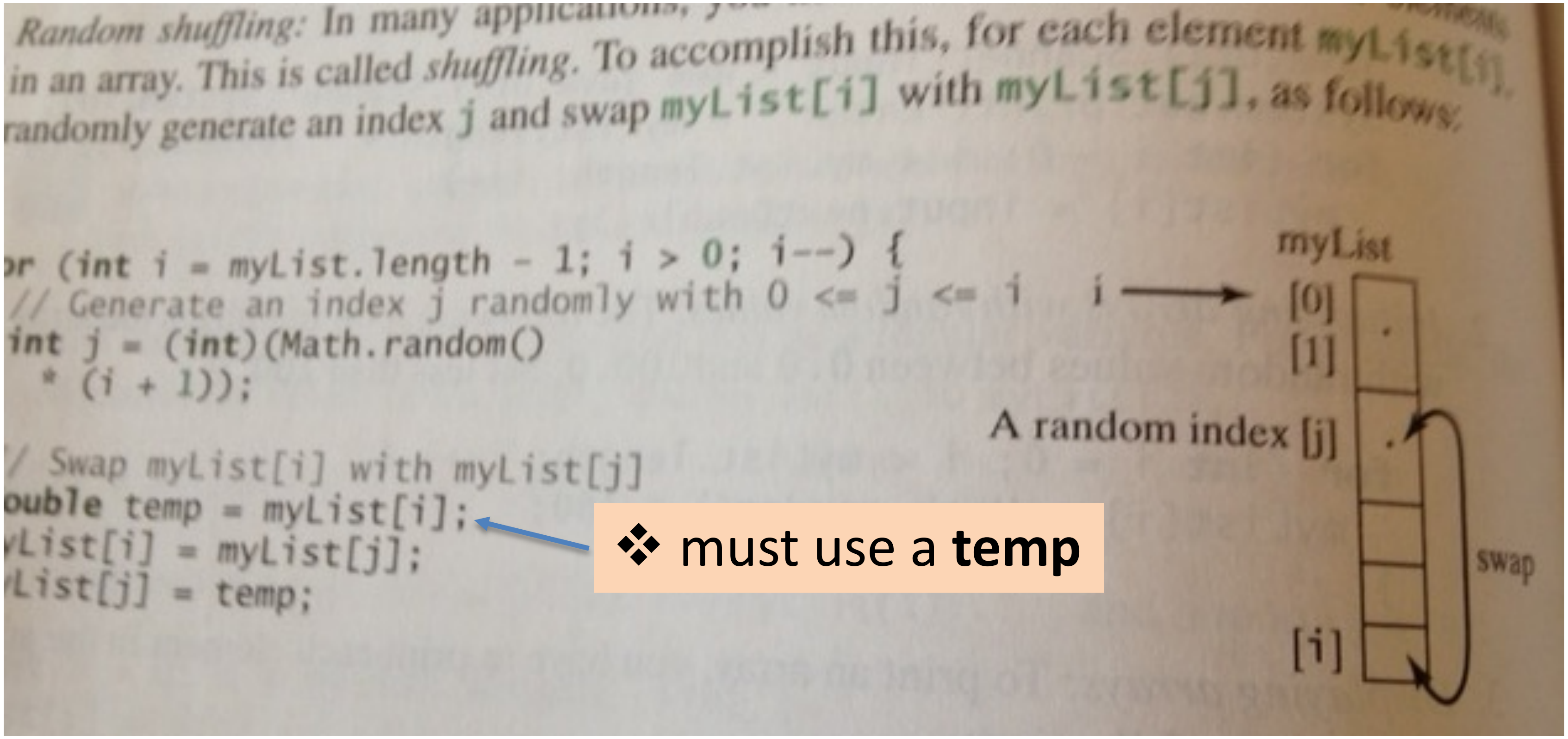
```
ist1 = {1, 2, 3, 4, 5, 6};
ist2 = reverse(list1);
```

CSUN CALIFORNIA STATE UNIVERSITY NORTHRIDGE

COMP110

Arrays: Random Shuffling

DR JEFF SOFTWARE INDIE APP DEVELOPER



Random shuffling: In many applications, you

in an array. This is called *shuffling*. To accomplish this, for each element myList[i], randomly generate an index j and swap myList[i] with myList[j], as follows:

```
or (int i = myList.length - 1; i > 0; i--) {
// Generate an index j randomly with 0 <= j <= i
int j = (int)(Math.random()
  * (i + 1));

/ Swap myList[i] with myList[j]
ouble temp = myList[i];
yList[i] = myList[j];
yList[j] = temp;
```

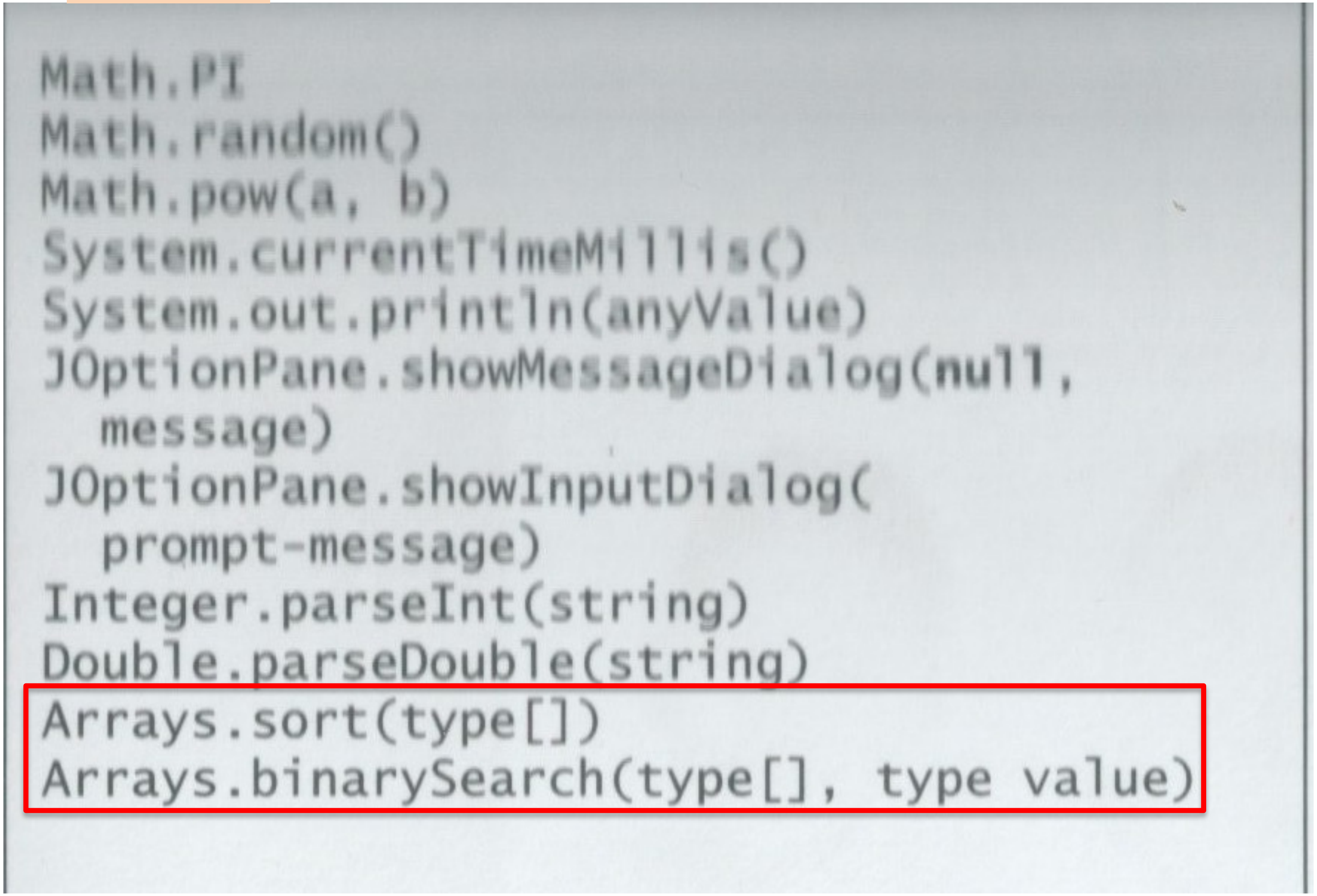
myList
i
[0]
[1]
A random index [j]
[i]
swap

❖ must use a **temp**

Arrays

Array Methods

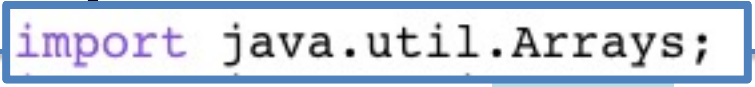
Library Functions



```
Math.PI
Math.random()
Math.pow(a, b)
System.currentTimeMillis()
System.out.println(anyValue)
JOptionPane.showMessageDialog(null,
   message)
JOptionPane.showInputDialog(
   prompt-message)
Integer.parseInt(string)
Double.parseDouble(string)
Arrays.sort(type[])
Arrays.binarySearch(type[], type value)
```

Arrays Class.methods

Ch 7



```
import java.util.Arrays;
```

sec 7.12

is?

equals

ArrName.isArray( )

Arrays.equals(arr1, arr2)

➢ Used in Lab 4

Fill

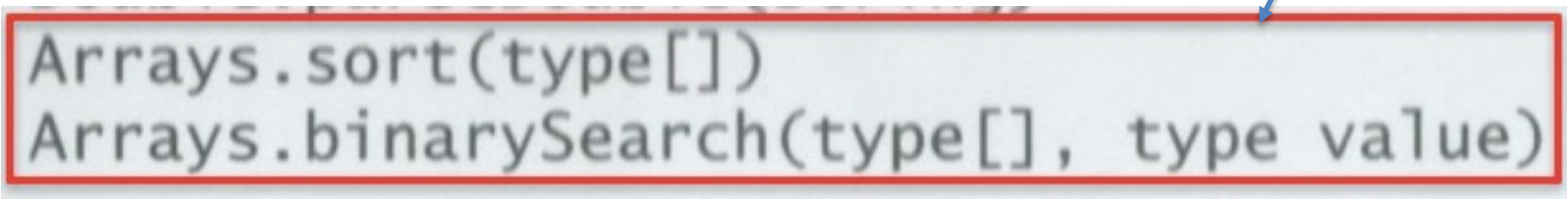
Arrays.fill(xArr, 8)

entire array

Arrays.fill(xArr, 2, 5, 8)

partial array [2..5]

sort & search



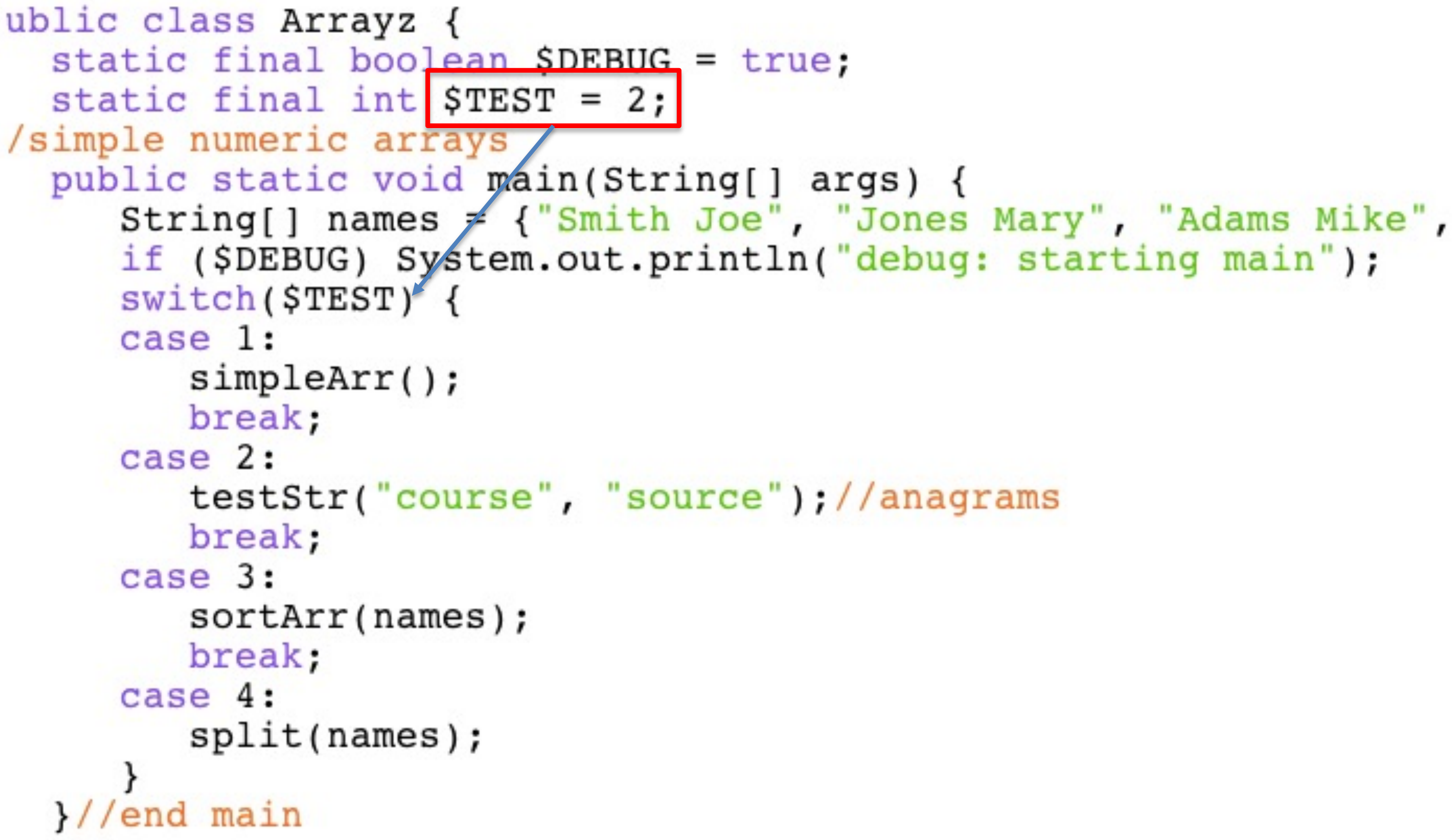
```
Arrays.sort(type[])
Arrays.binarySearch(type[], type value)
```

Partial Sort

Arrays.sort(xArr, 2, 5)

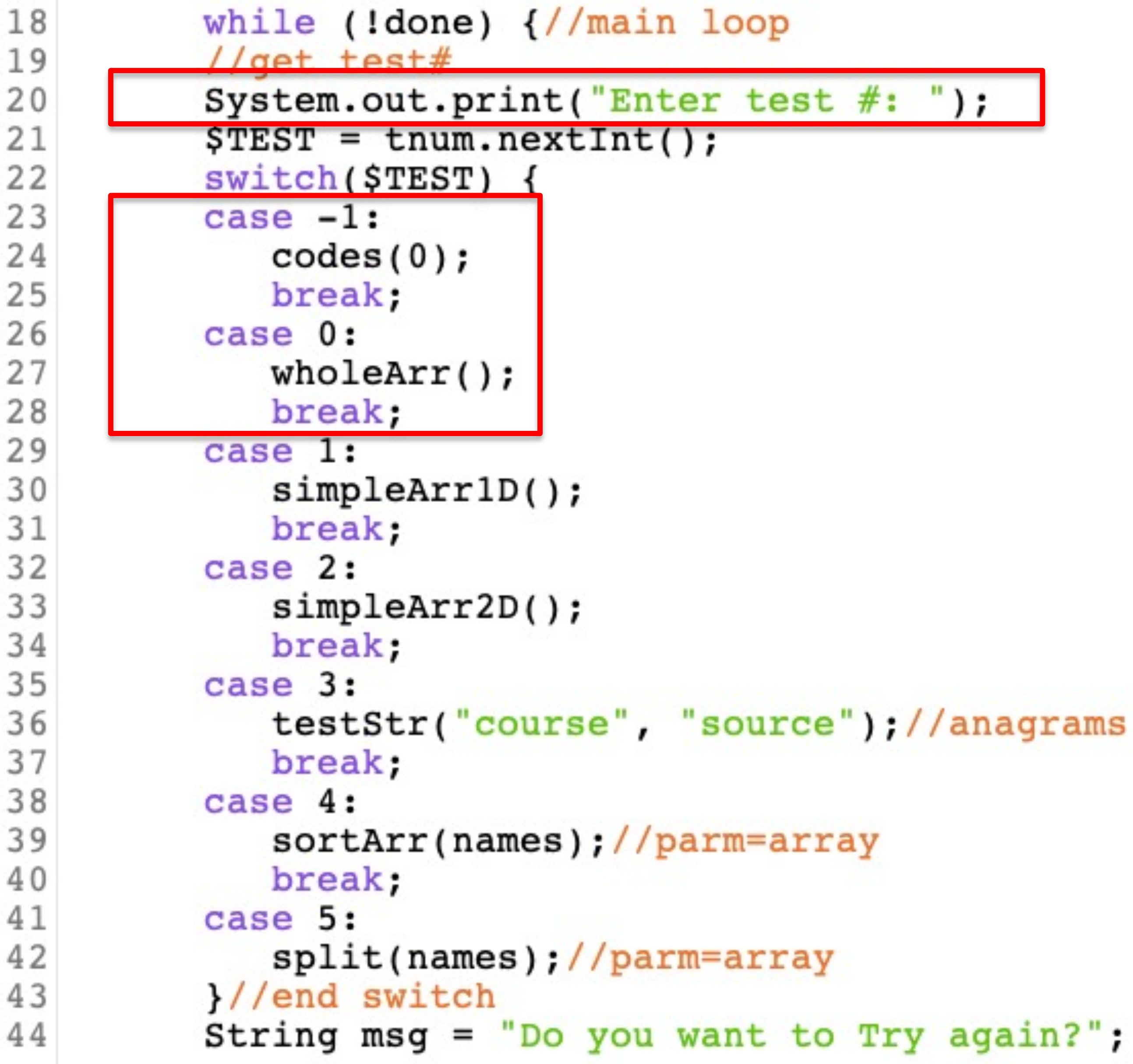
partial array [2..5]

Selecting Methods



```
ublic class Arrayz {
  static final boolean $DEBUG = true;
  static final int $TEST = 2;
/simple numeric arrays
  public static void main(String[] args) {
     String[] names = {"Smith Joe", "Jones Mary", "Adams Mike",
     if ($DEBUG) System.out.println("debug: starting main");
     switch($TEST) {
     case 1:
        simpleArr();
        break;
     case 2:
        testStr("course", "source");//anagrams
        break;
     case 3:
        sortArr(names);
        break;
     case 4:
        split(names);
     }
  }//end main
```

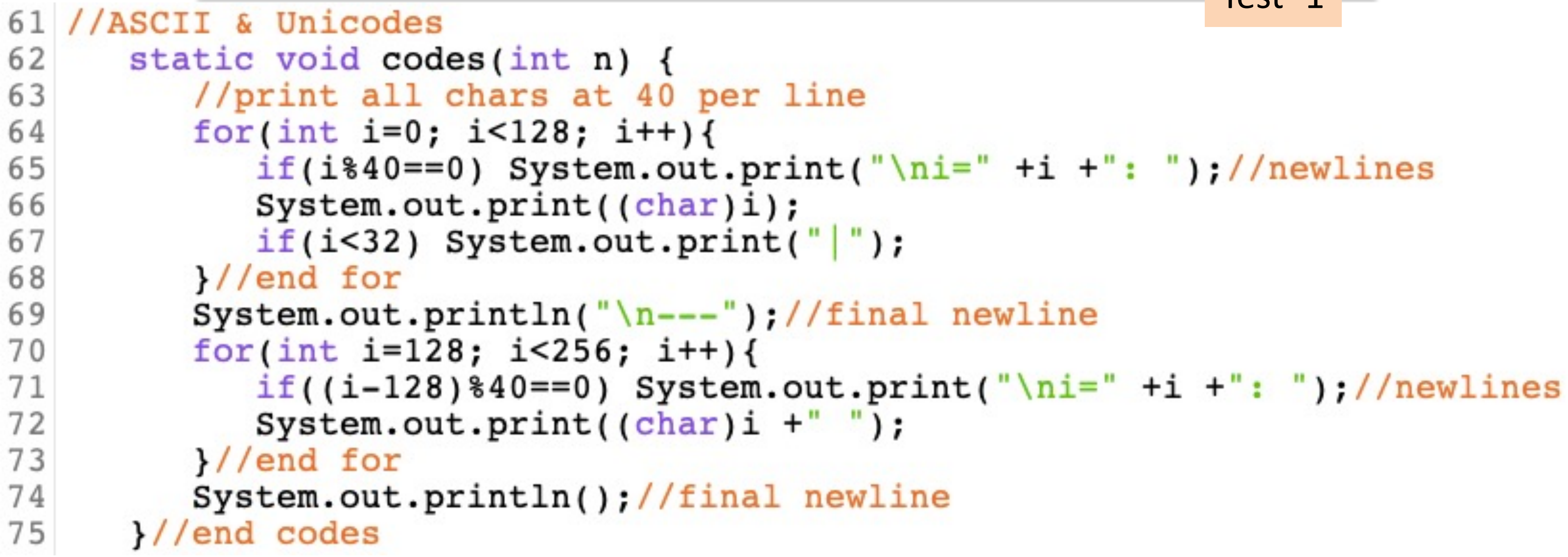
New Tests



```
        while (!done) {//main loop
        //get test#
        System.out.print("Enter test #: ");
        $TEST = tnum.nextInt();
        switch($TEST) {
        case -1:
            codes(0);
            break;
        case 0:
            wholeArr();
            break;
        case 1:
            simpleArr1D();
            break;
        case 2:
            simpleArr2D();
            break;
        case 3:
            testStr("course", "source");//anagrams
            break;
        case 4:
            sortArr(names);//parm=array
            break;
        case 5:
            split(names);//parm=array
        }//end switch
        String msg = "Do you want to Try again?";
```

ASCII + Unicodes

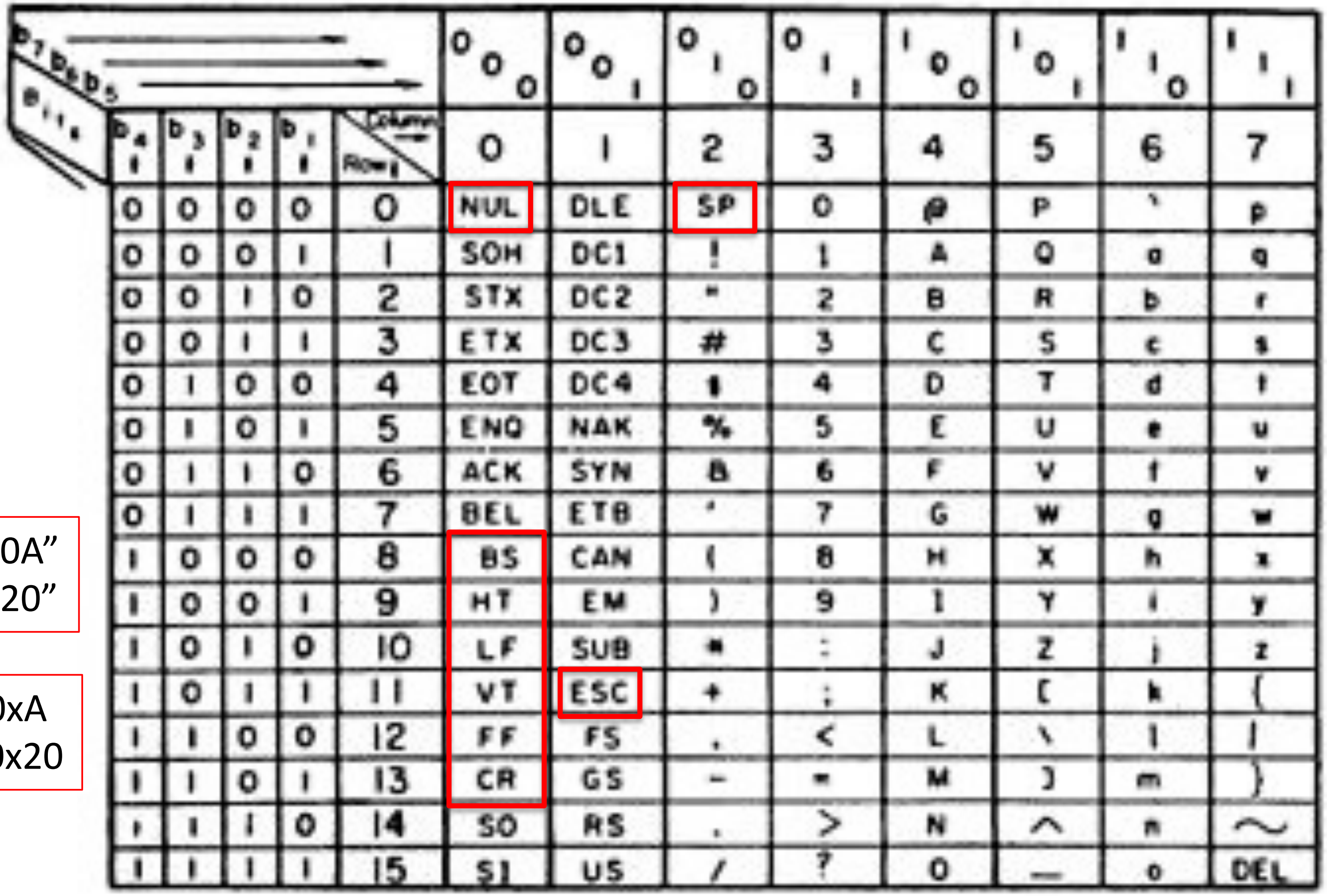
Test -1



```
//ASCII & Unicodes
    static void codes(int n) {
        //print all chars at 40 per line
        for(int i=0; i<128; i++){
            if(i%40==0) System.out.print("\ni=" +i +": ");//newlines
            System.out.print((char)i);
            if(i<32) System.out.print("|");
        }//end for
        System.out.println("\n---");//final newline
        for(int i=128; i<256; i++){
            if((i-128)%40==0) System.out.print("\ni=" +i +": ");//newlines
            System.out.print((char)i +"  ");
        }//end for
        System.out.println();//final newline
    }//end codes
```

ASCII Codes- 7-bit

USASCII code chart



b_7 b_6 b_5 →					0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b_4 ↓	b_3 ↓	b_2 ↓	b_1 ↓	Column → / Row ↓	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	|
1	1	0	1	13	CR	GS	-	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	_	o	DEL

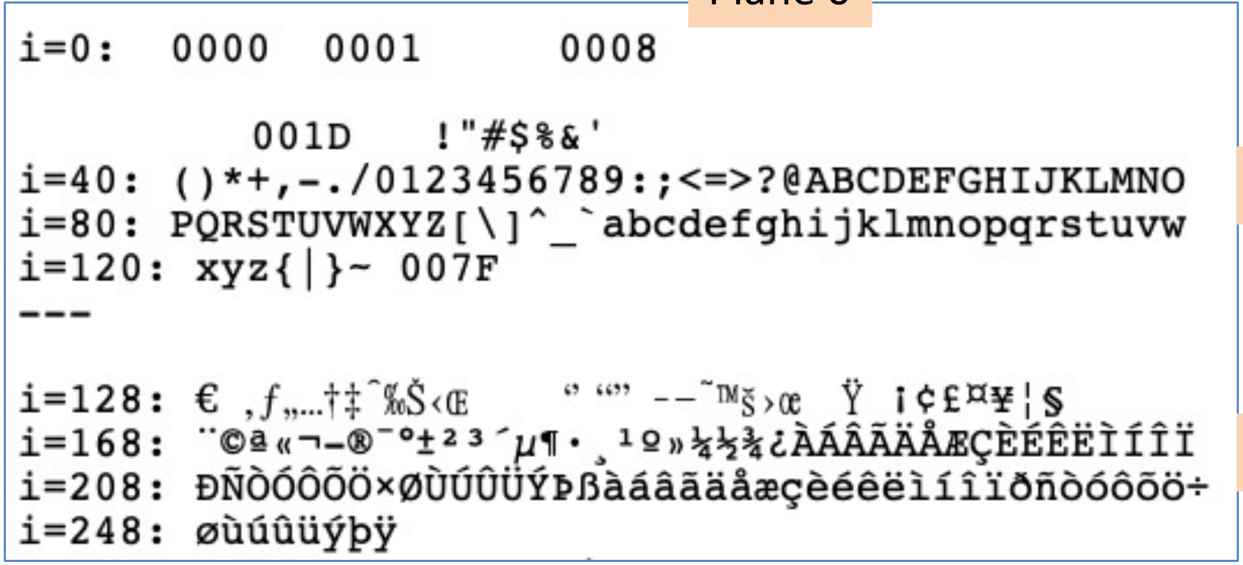
\n="\u000A"
sp="\u0020"

char ch=0xA
char sp=0x20

IANA encourages use of the name "US-ASCII" for Internet uses of ASCII

ASCII + Unicodes

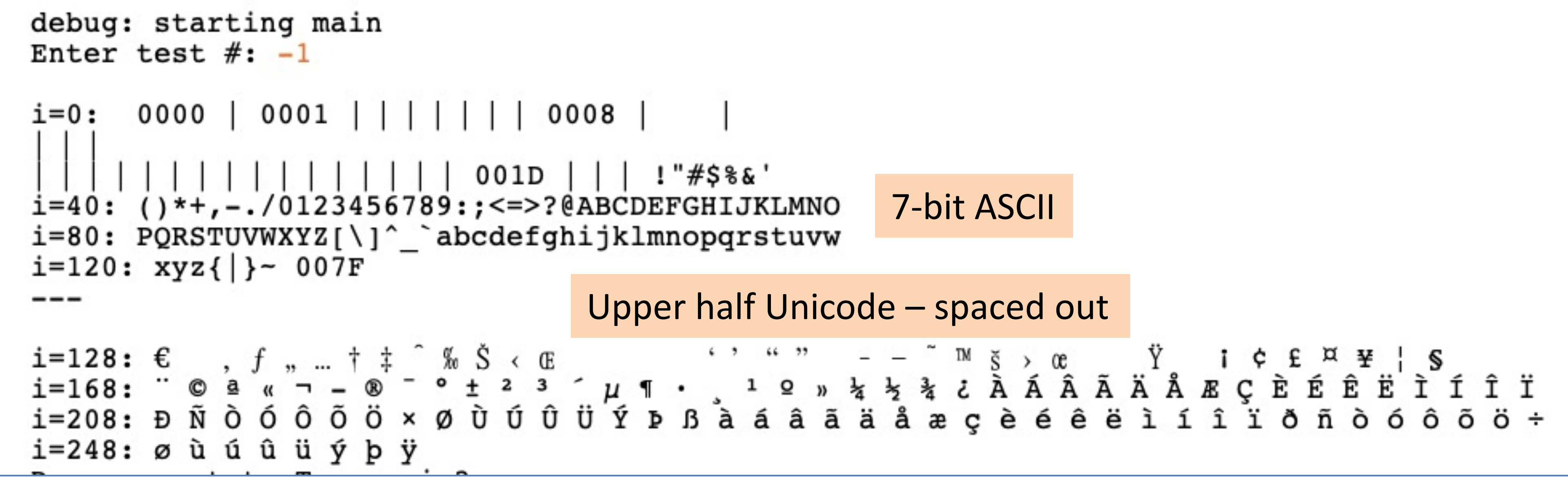
Plane 0



```
i=0:   0000  0001      0008


          001D    !"#$%&'
i=40: ()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNO
i=80: PQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvw
i=120: xyz{|}~ 007F
---

i=128: € ‚ƒ„…†‡ˆ‰Š‹Œ     ‘’“”–—˜™š›œ  Ÿ ¡¢£¤¥¦§
i=168: ¨©ª«¬-®¯°±²³´µ¶·¸¹º»¼½¾¿ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏ
i=208: ÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞßàáâãäåæçèéêëìíîïðñòóôõö÷
i=248: øùúûüýþÿ
```

7-bit ASCII

Upper half Unicode



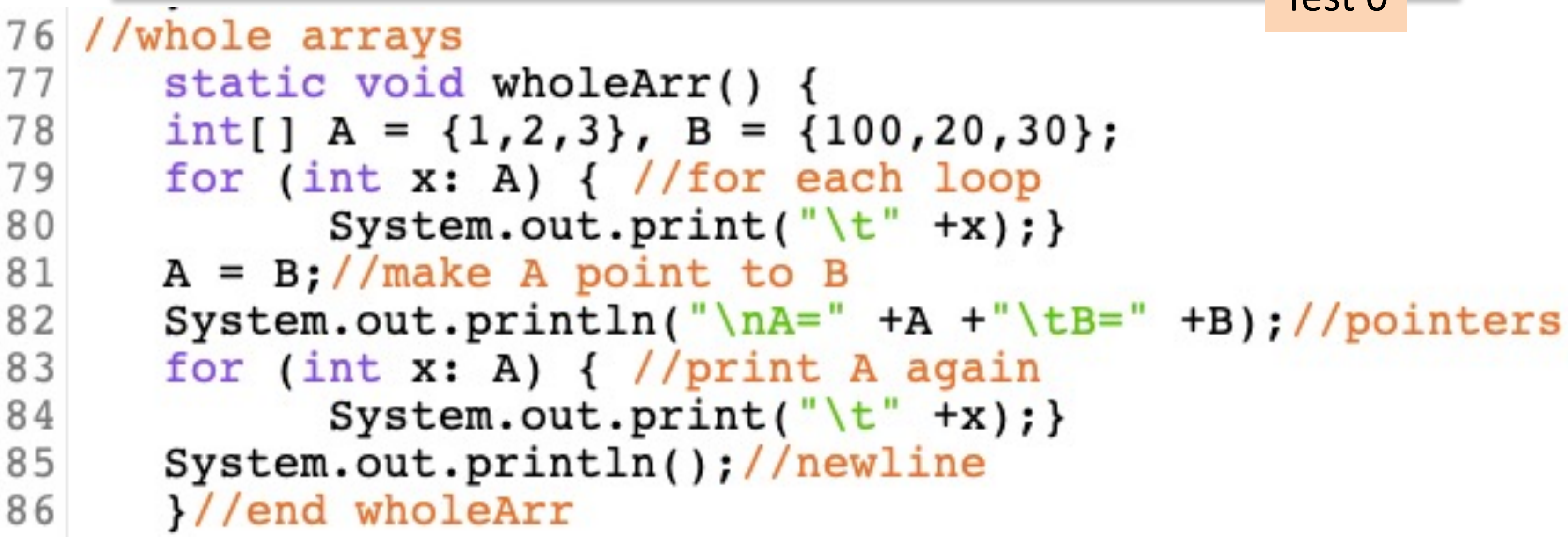
```
debug: starting main
Enter test #: -1

i=0:  0000 | 0001 | | | | | | | 0008 |    |
| | |
| | | | | | | | | | | | | | | | 001D | | | !"#$%&'
i=40: ()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNO
i=80: PQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvw
i=120: xyz{|}~ 007F
---

i=128: €   ‚ ƒ „ … † ‡ ˆ ‰ Š ‹ Œ           ‘ ’  “ ”   – — ˜ ™ š › œ    Ÿ   ¡ ¢ £ ¤ ¥ ¦ §
i=168: ¨ © ª « ¬ - ® ¯ ° ± ² ³ ´ µ ¶ · ¸ ¹ º » ¼ ½ ¾ ¿ À Á Â Ã Ä Å Æ Ç È É Ê Ë Ì Í Î Ï
i=208: Ð Ñ Ò Ó Ô Õ Ö × Ø Ù Ú Û Ü Ý Þ ß à á â ã ä å æ ç è é ê ë ì í î ï ð ñ ò ó ô õ ö ÷
i=248: ø ù ú û ü ý þ ÿ
```

7-bit ASCII

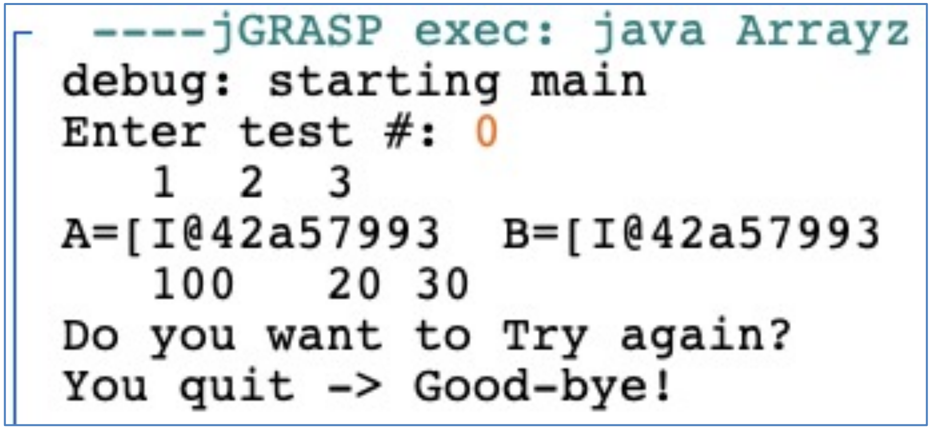
Upper half Unicode – spaced out

Whole Array

Test 0



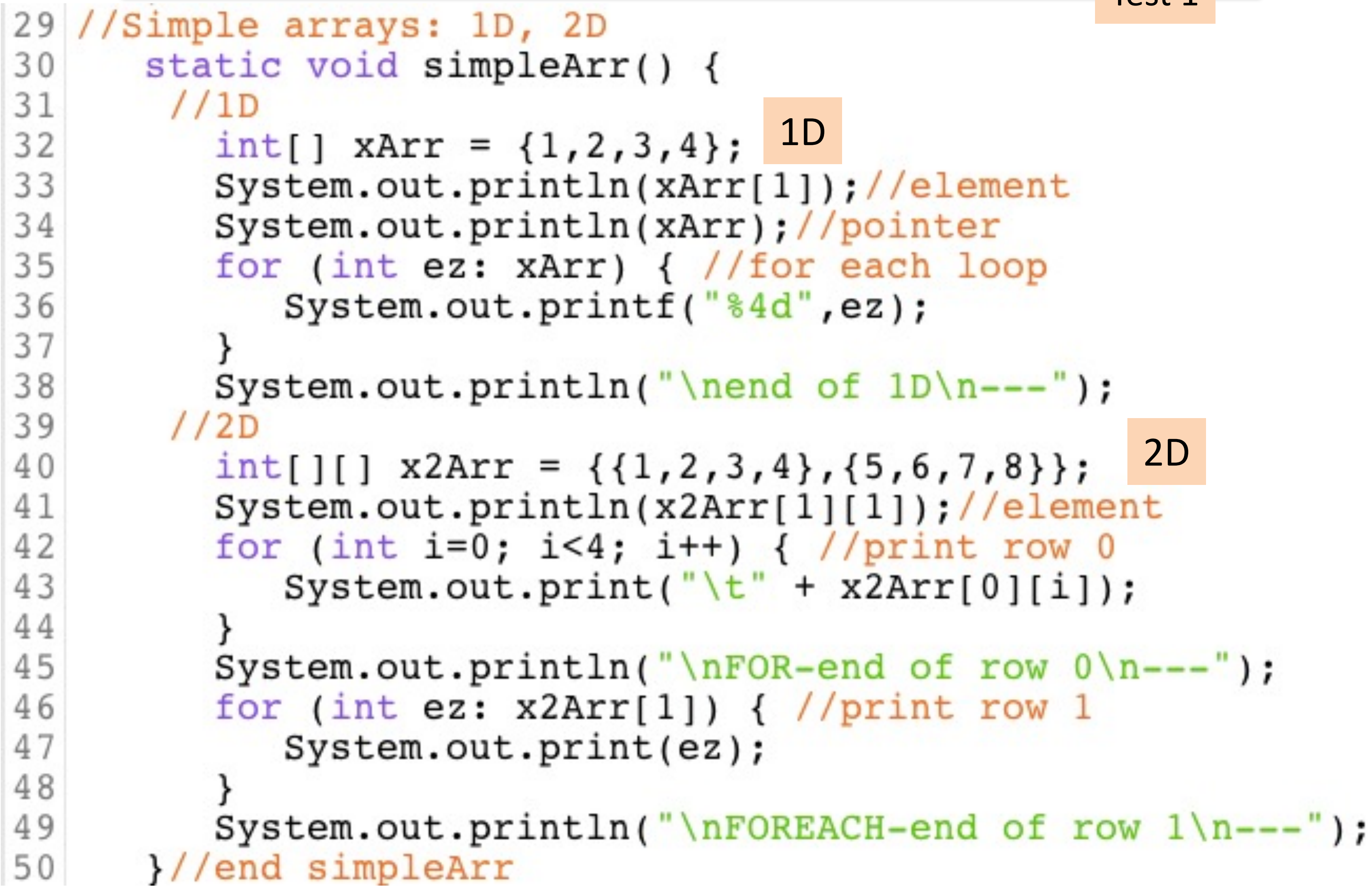
```
//whole arrays
   static void wholeArr() {
   int[] A = {1,2,3}, B = {100,20,30};
   for (int x: A) { //for each loop
         System.out.print("\t" +x);}
   A = B;//make A point to B
   System.out.println("\nA=" +A +"\tB=" +B);//pointers
   for (int x: A) { //print A again
         System.out.print("\t" +x);}
   System.out.println();//newline
   }//end wholeArr
```



```
 ----jGRASP exec: java Arrayz
debug: starting main
Enter test #: 0
   1  2  3
A=[I@42a57993  B=[I@42a57993
   100   20 30
Do you want to Try again?
You quit -> Good-bye!
```

Arrays Example #1

Test 1

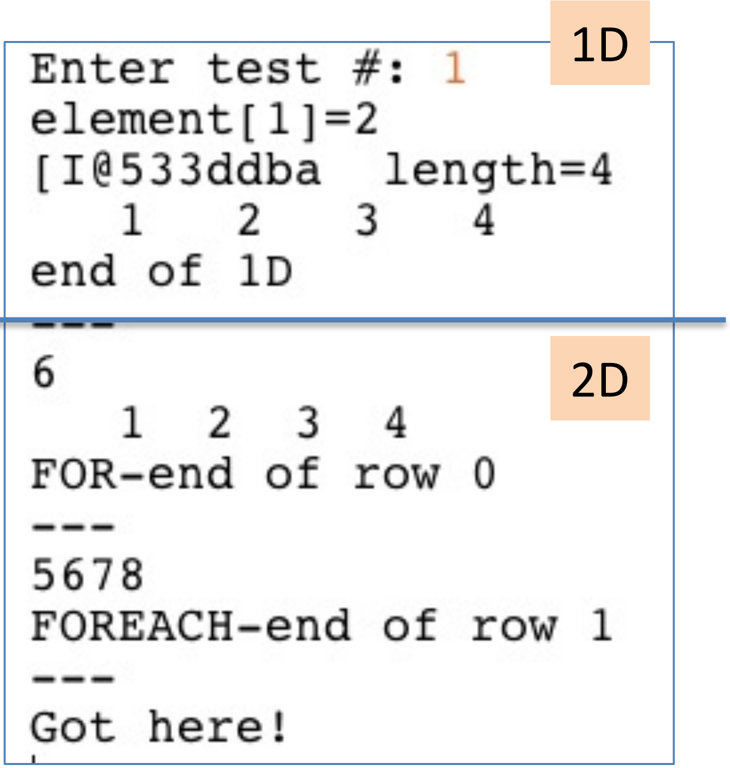


```
//Simple arrays: 1D, 2D
   static void simpleArr() {
    //1D
      int[] xArr = {1,2,3,4};
      System.out.println(xArr[1]);//element
      System.out.println(xArr);//pointer
      for (int ez: xArr) { //for each loop
         System.out.printf("%4d",ez);
      }
      System.out.println("\nend of 1D\n---");
    //2D
      int[][] x2Arr = {{1,2,3,4},{5,6,7,8}};
      System.out.println(x2Arr[1][1]);//element
      for (int i=0; i<4; i++) { //print row 0
         System.out.print("\t" + x2Arr[0][i]);
      }
      System.out.println("\nFOR-end of row 0\n---");
      for (int ez: x2Arr[1]) { //print row 1
         System.out.print(ez);
      }
      System.out.println("\nFOREACH-end of row 1\n---");
   }//end simpleArr
```

1D

2D

Arrays Example #1



1D

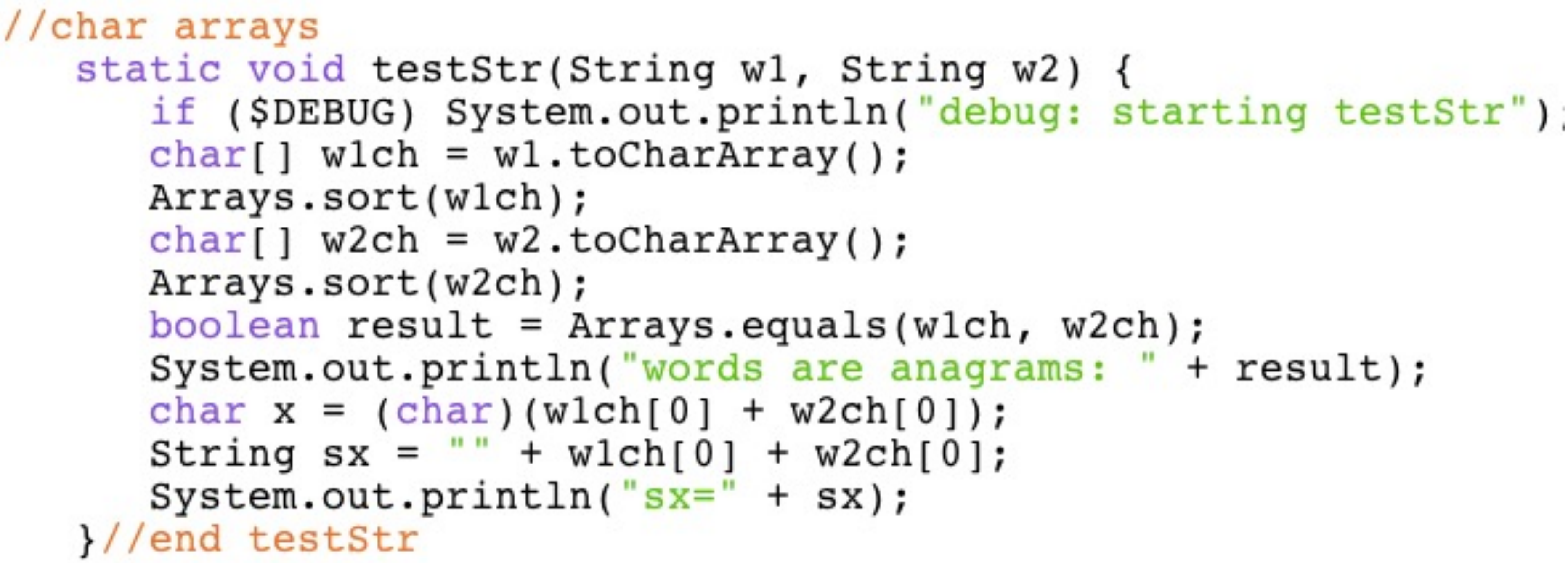
```
Enter test #: 1
element[1]=2
[I@533ddba  length=4
   1    2    3    4
end of 1D
---
6
   1  2  3  4
FOR-end of row 0
---
5678
FOREACH-end of row 1
---
Got here!
```

2D

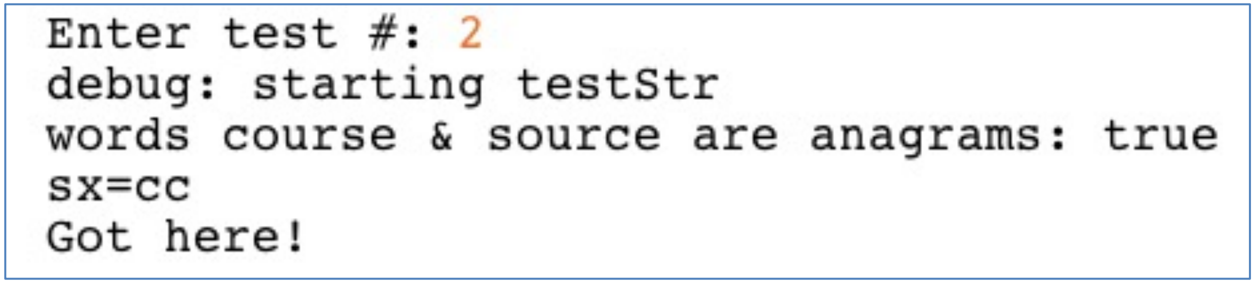
Arrays Example #2

Lab 4: anagrams

Test 2



```
//char arrays
   static void testStr(String w1, String w2) {
      if ($DEBUG) System.out.println("debug: starting testStr");
      char[] w1ch = w1.toCharArray();
      Arrays.sort(w1ch);
      char[] w2ch = w2.toCharArray();
      Arrays.sort(w2ch);
      boolean result = Arrays.equals(w1ch, w2ch);
      System.out.println("words are anagrams: " + result);
      char x = (char)(w1ch[0] + w2ch[0]);
      String sx = "" + w1ch[0] + w2ch[0];
      System.out.println("sx=" + sx);
   }//end testStr
```



```
Enter test #: 2
debug: starting testStr
words course & source are anagrams: true
sx=cc
Got here!
```

Sorting

Searching & Sorting Arrays

Arrays: Searching & Sorting

Ch 7

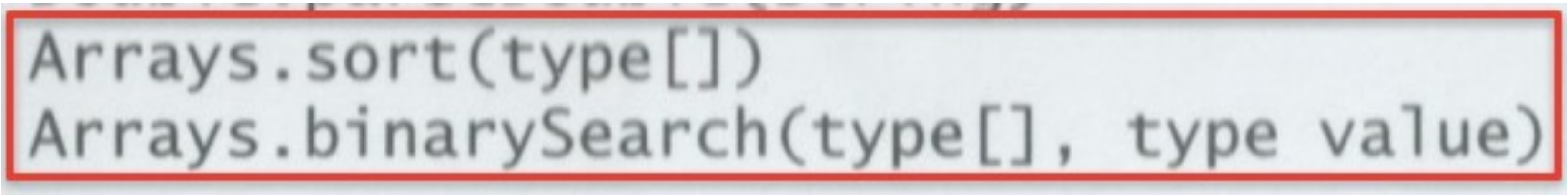
sec 7.12

❖Searching

- □ Linear (non-sorted)
- □ Binary (sorted)

❖Sorting

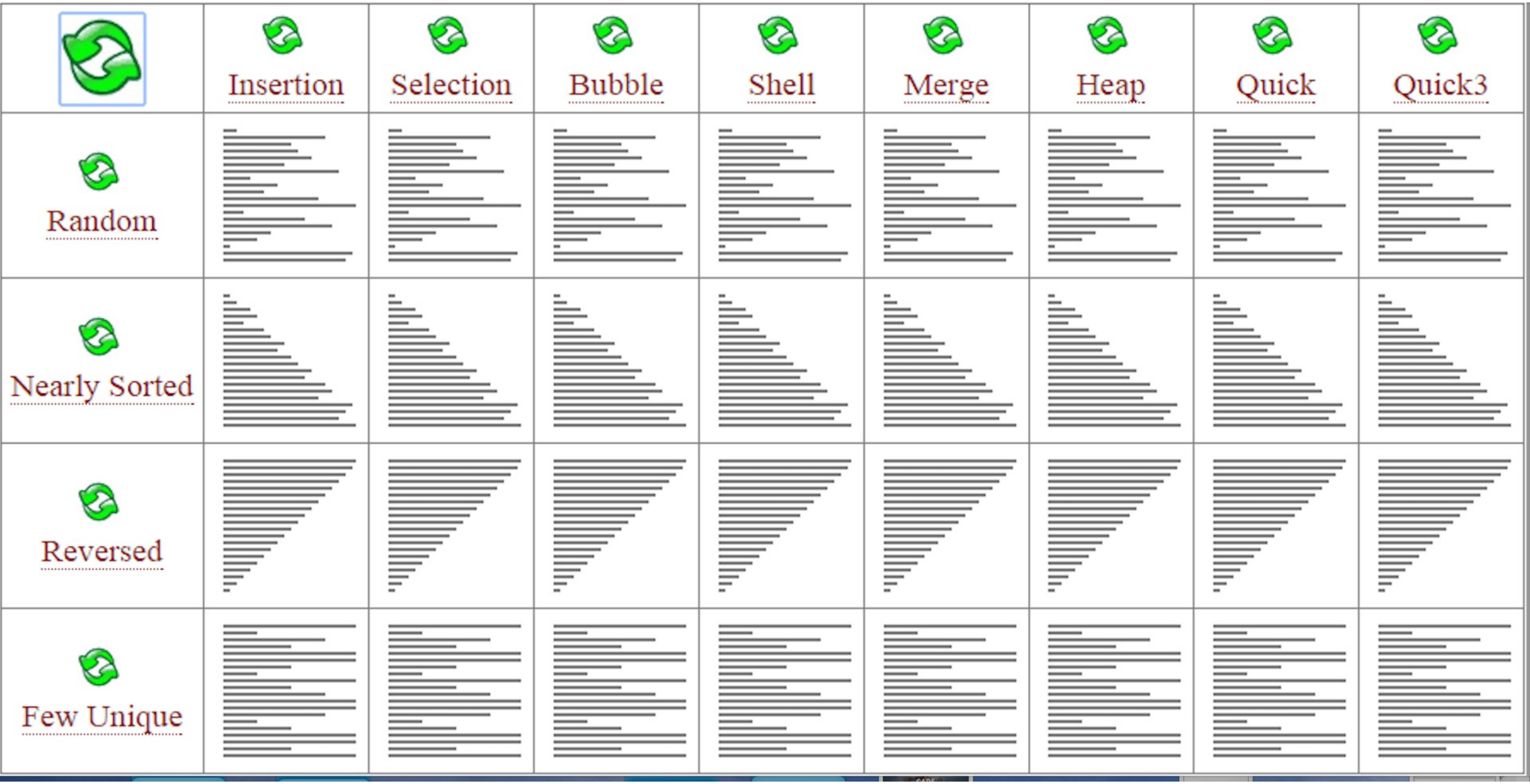
- □ Linear
- □ Selection (textbook Ch 7)
- □ Bubble
- □ Quicksort
- □ Qsort



```
Arrays.sort(type[])
Arrays.binarySearch(type[], type value)
```

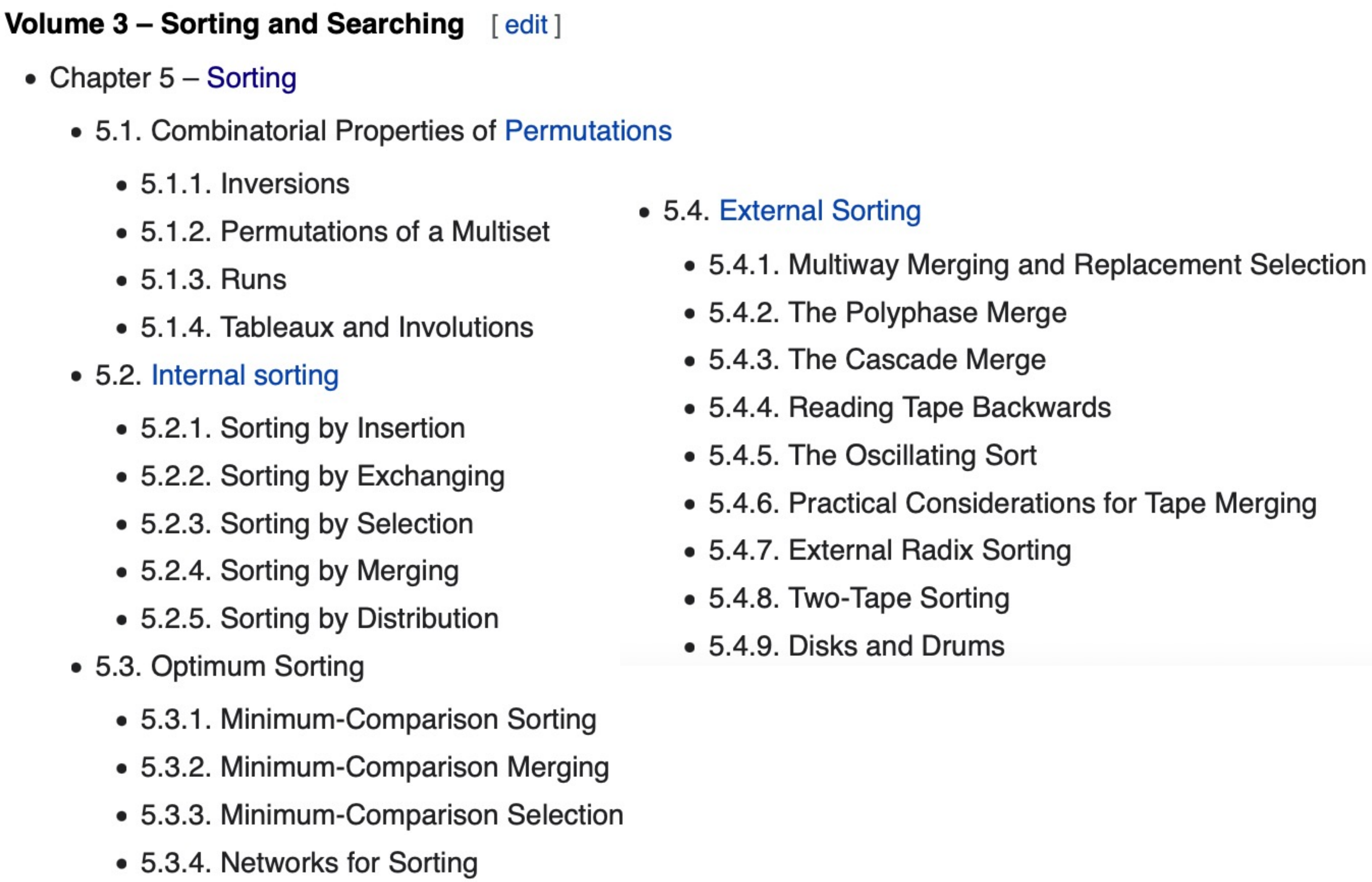
- ➢ binarySearch returns found array index or negative index
- ➢ = - (insertion index + 1)

Arrays: Sorting



	Insertion	Selection	Bubble	Shell	Merge	Heap	Quick	Quick3
Random								
Nearly Sorted								
Reversed								
Few Unique								

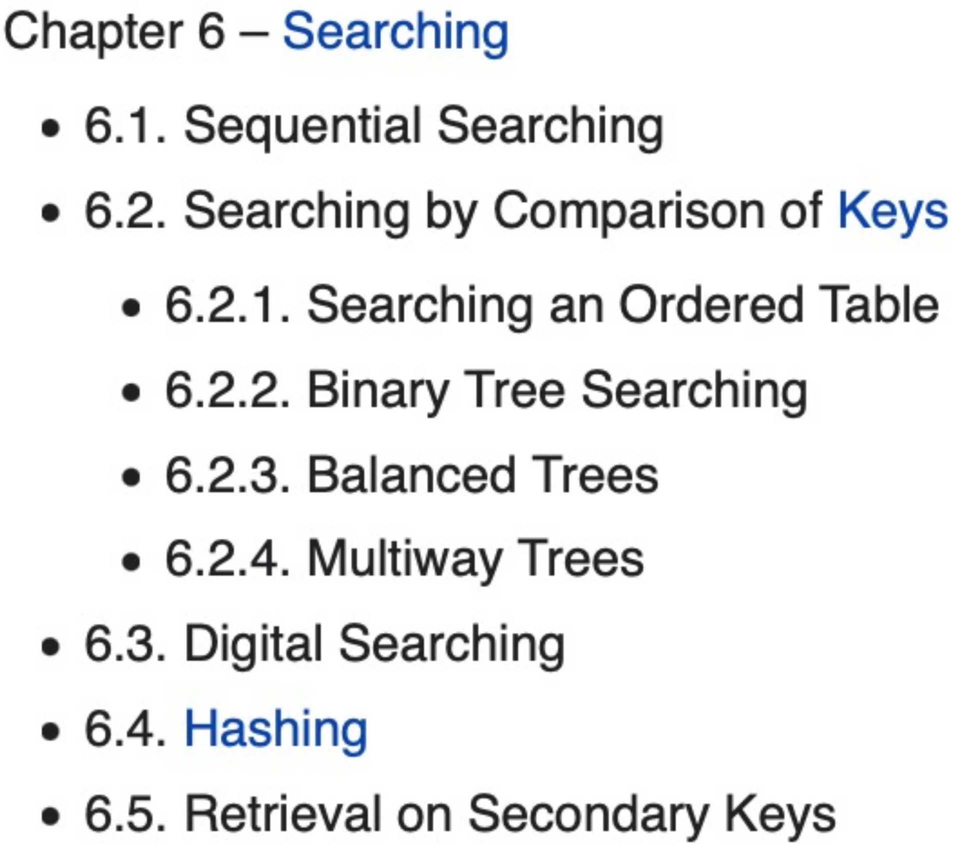
Sorting: Donald Knuth



Volume 3 – Sorting and Searching [edit]

- Chapter 5 – Sorting
  - 5.1. Combinatorial Properties of Permutations
    - 5.1.1. Inversions
    - 5.1.2. Permutations of a Multiset
    - 5.1.3. Runs
    - 5.1.4. Tableaux and Involutions
  - 5.2. Internal sorting
    - 5.2.1. Sorting by Insertion
    - 5.2.2. Sorting by Exchanging
    - 5.2.3. Sorting by Selection
    - 5.2.4. Sorting by Merging
    - 5.2.5. Sorting by Distribution
  - 5.3. Optimum Sorting
    - 5.3.1. Minimum-Comparison Sorting
    - 5.3.2. Minimum-Comparison Merging
    - 5.3.3. Minimum-Comparison Selection
    - 5.3.4. Networks for Sorting
  - 5.4. External Sorting
    - 5.4.1. Multiway Merging and Replacement Selection
    - 5.4.2. The Polyphase Merge
    - 5.4.3. The Cascade Merge
    - 5.4.4. Reading Tape Backwards
    - 5.4.5. The Oscillating Sort
    - 5.4.6. Practical Considerations for Tape Merging
    - 5.4.7. External Radix Sorting
    - 5.4.8. Two-Tape Sorting
    - 5.4.9. Disks and Drums

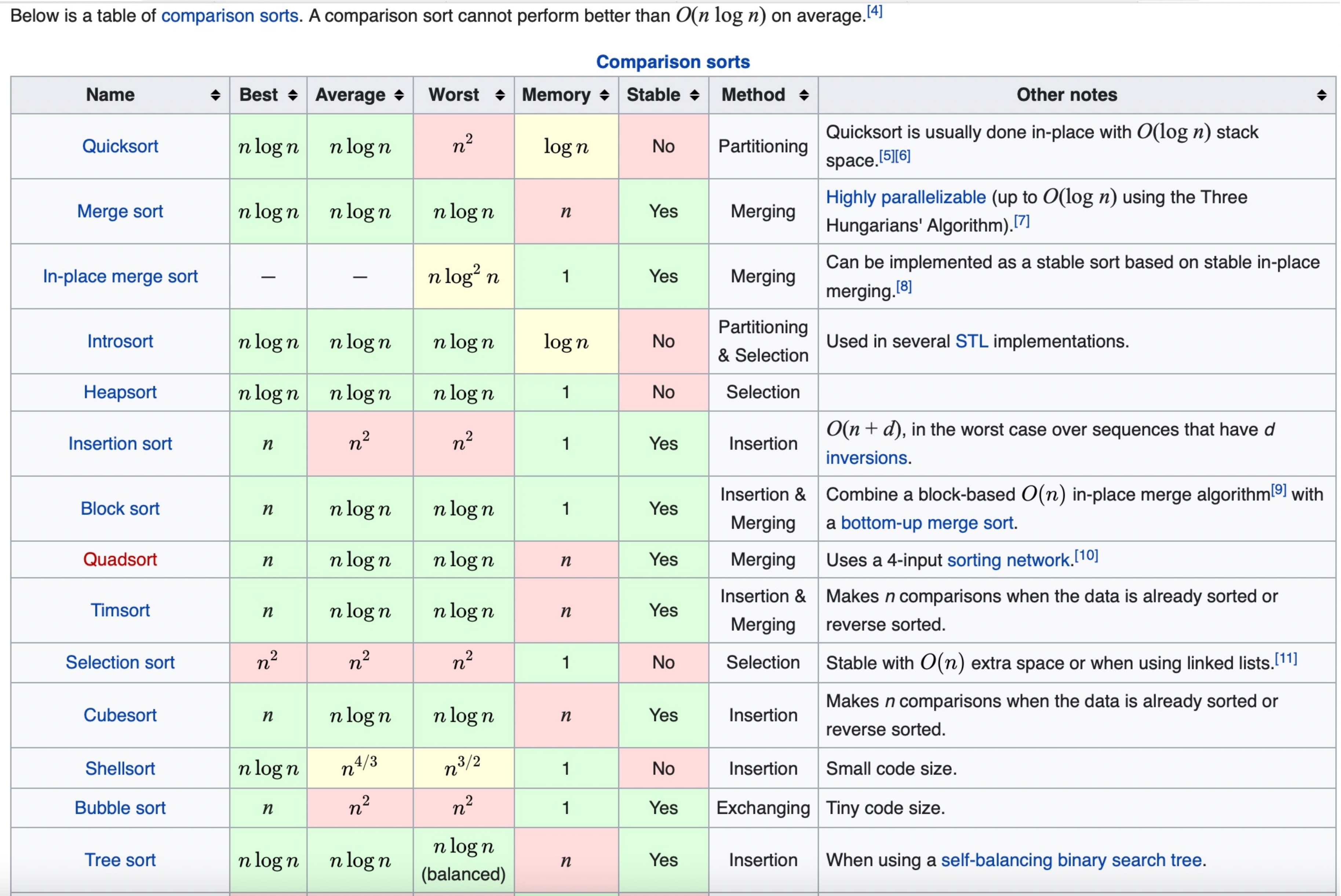
Searching: Donald Knuth



Chapter 6 – Searching

- 6.1. Sequential Searching
- 6.2. Searching by Comparison of Keys
  - 6.2.1. Searching an Ordered Table
  - 6.2.2. Binary Tree Searching
  - 6.2.3. Balanced Trees
  - 6.2.4. Multiway Trees
- 6.3. Digital Searching
- 6.4. Hashing
- 6.5. Retrieval on Secondary Keys

Arrays: Sorting



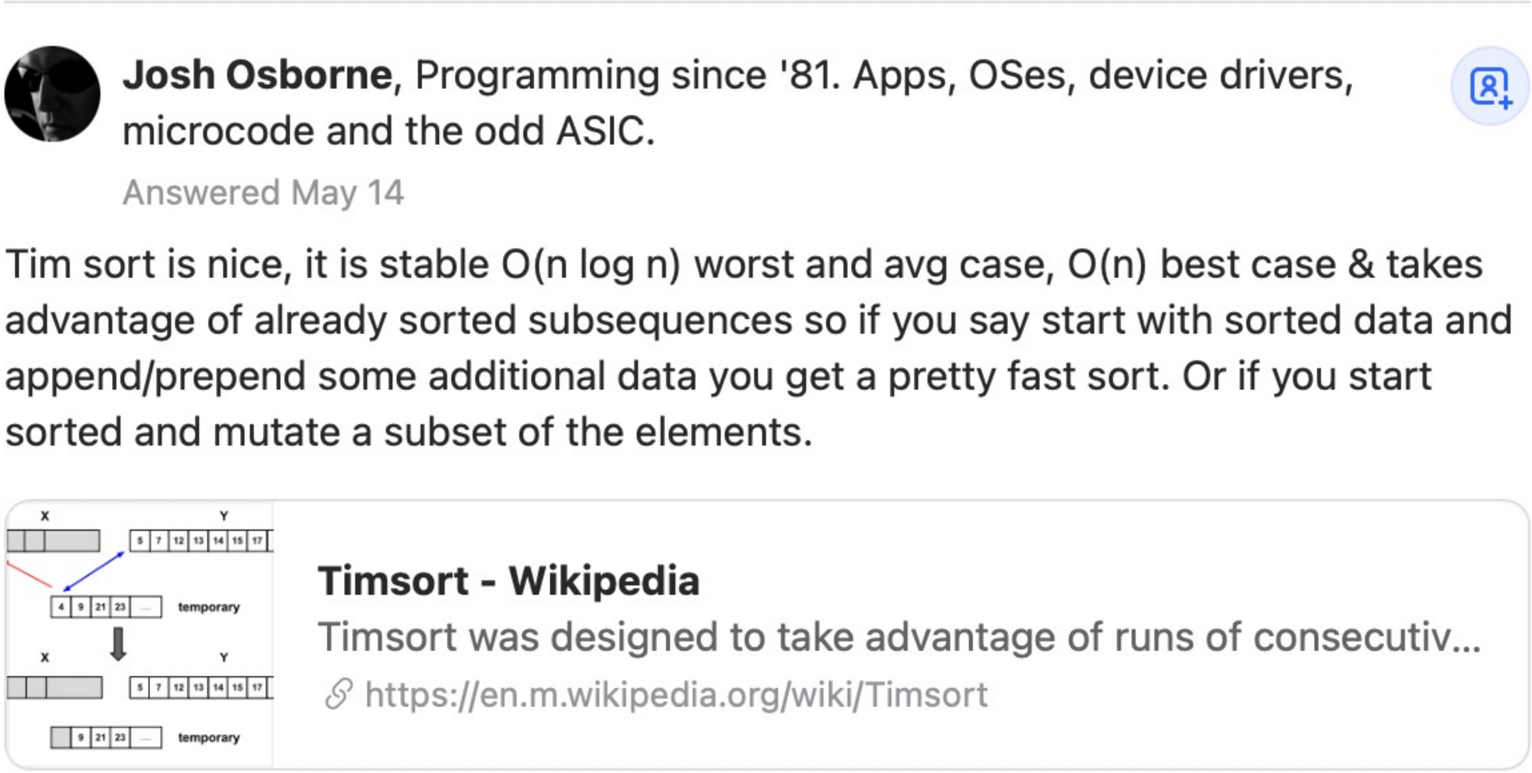
Below is a table of comparison sorts. A comparison sort cannot perform better than $O(n \log n)$ on average.[4]

Comparison sorts

Name	Best	Average	Worst	Memory	Stable	Method	Other notes
Quicksort	$n \log n$	$n \log n$	n^2	$\log n$	No	Partitioning	Quicksort is usually done in-place with $O(\log n)$ stack space.[5][6]
Merge sort	$n \log n$	$n \log n$	$n \log n$	n	Yes	Merging	Highly parallelizable (up to $O(\log n)$ using the Three Hungarians' Algorithm).[7]
In-place merge sort	—	—	$n \log^2 n$	1	Yes	Merging	Can be implemented as a stable sort based on stable in-place merging.[8]
Introsort	$n \log n$	$n \log n$	$n \log n$	$\log n$	No	Partitioning & Selection	Used in several STL implementations.
Heapsort	$n \log n$	$n \log n$	$n \log n$	1	No	Selection	
Insertion sort	n	n^2	n^2	1	Yes	Insertion	$O(n + d)$, in the worst case over sequences that have *d* inversions.
Block sort	n	$n \log n$	$n \log n$	1	Yes	Insertion & Merging	Combine a block-based $O(n)$ in-place merge algorithm[9] with a bottom-up merge sort.
Quadsort	n	$n \log n$	$n \log n$	n	Yes	Merging	Uses a 4-input sorting network.[10]
Timsort	n	$n \log n$	$n \log n$	n	Yes	Insertion & Merging	Makes *n* comparisons when the data is already sorted or reverse sorted.
Selection sort	n^2	n^2	n^2	1	No	Selection	Stable with $O(n)$ extra space or when using linked lists.[11]
Cubesort	n	$n \log n$	$n \log n$	n	Yes	Insertion	Makes *n* comparisons when the data is already sorted or reverse sorted.
Shellsort	$n \log n$	$n^{4/3}$	$n^{3/2}$	1	No	Insertion	Small code size.
Bubble sort	n	n^2	n^2	1	Yes	Exchanging	Tiny code size.
Tree sort	$n \log n$	$n \log n$	$n \log n$ (balanced)	n	Yes	Insertion	When using a self-balancing binary search tree.

Arrays: Sorting

- ❖ Sorting
  - ❑ Timsort



Josh Osborne, Programming since '81. Apps, OSes, device drivers, microcode and the odd ASIC.

Answered May 14

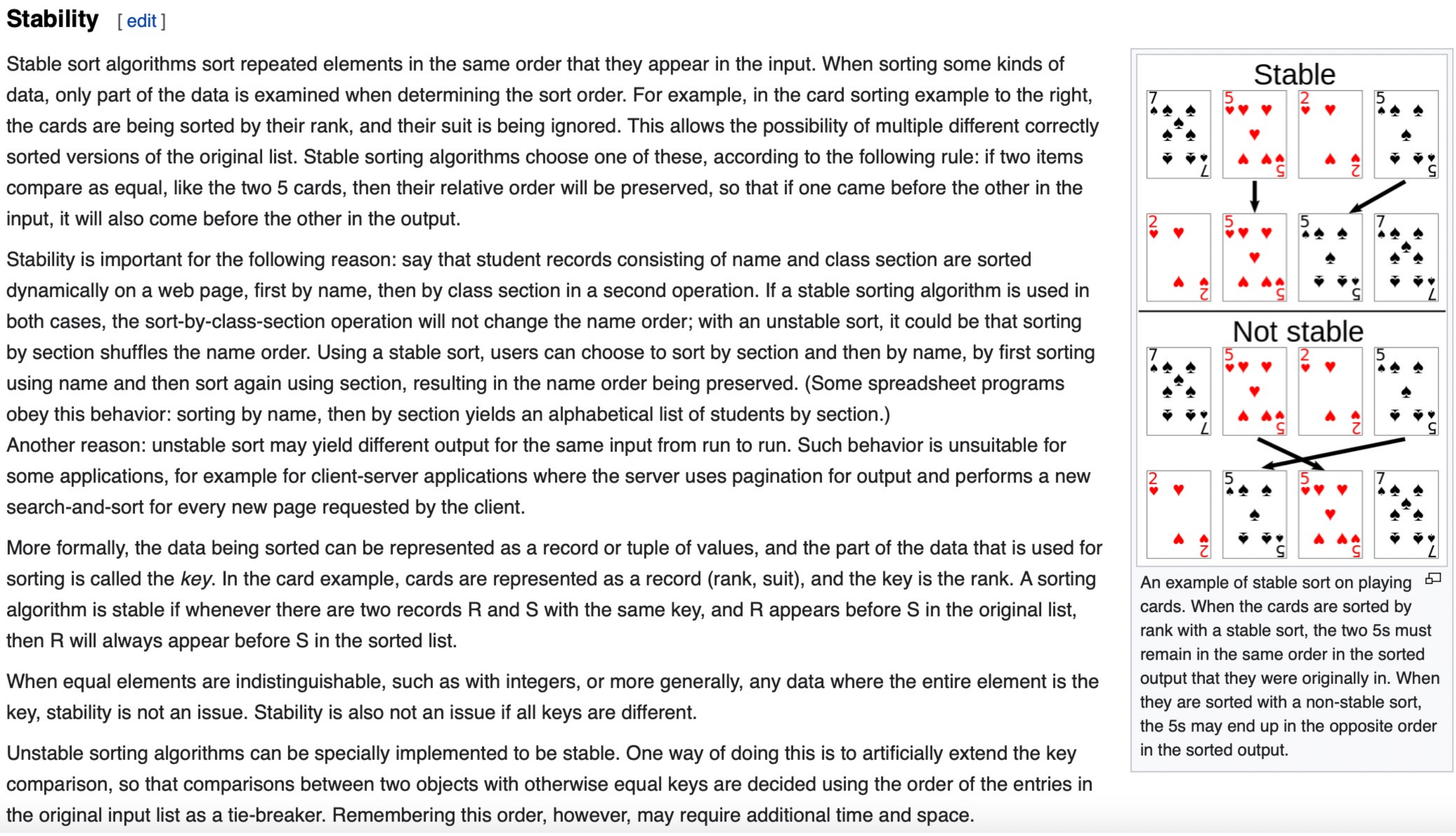
Tim sort is nice, it is stable O(n log n) worst and avg case, O(n) best case & takes advantage of already sorted subsequences so if you say start with sorted data and append/prepend some additional data you get a pretty fast sort. Or if you start sorted and mutate a subset of the elements.

Timsort - Wikipedia

Timsort was designed to take advantage of runs of consectiv...

https://en.m.wikipedia.org/wiki/Timsort

Sorting: Stability



Stability [edit]

Stable sort algorithms sort repeated elements in the same order that they appear in the input. When sorting some kinds of data, only part of the data is examined when determining the sort order. For example, in the card sorting example to the right, the cards are being sorted by their rank, and their suit is being ignored. This allows the possibility of multiple different correctly sorted versions of the original list. Stable sorting algorithms choose one of these, according to the following rule: if two items compare as equal, like the two 5 cards, then their relative order will be preserved, so that if one came before the other in the input, it will also come before the other in the output.

Stability is important for the following reason: say that student records consisting of name and class section are sorted dynamically on a web page, first by name, then by class section in a second operation. If a stable sorting algorithm is used in both cases, the sort-by-class-section operation will not change the name order; with an unstable sort, it could be that sorting by section shuffles the name order. Using a stable sort, users can choose to sort by section and then by name, by first sorting using name and then sort again using section, resulting in the name order being preserved. (Some spreadsheet programs obey this behavior: sorting by name, then by section yields an alphabetical list of students by section.)
Another reason: unstable sort may yield different output for the same input from run to run. Such behavior is unsuitable for some applications, for example for client-server applications where the server uses pagination for output and performs a new search-and-sort for every new page requested by the client.

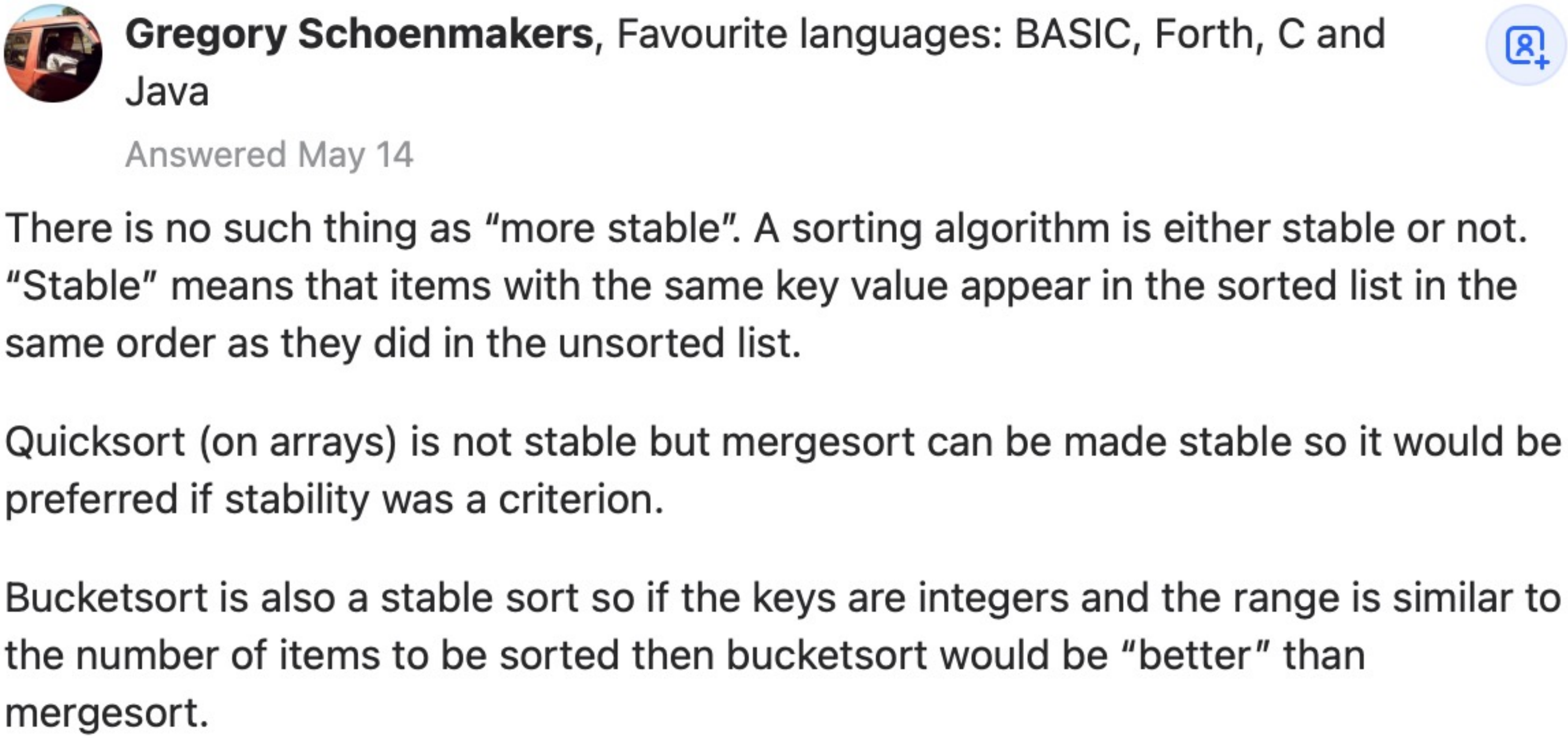
More formally, the data being sorted can be represented as a record or tuple of values, and the part of the data that is used for sorting is called the *key*. In the card example, cards are represented as a record (rank, suit), and the key is the rank. A sorting algorithm is stable if whenever there are two records R and S with the same key, and R appears before S in the original list, then R will always appear before S in the sorted list.

When equal elements are indistinguishable, such as with integers, or more generally, any data where the entire element is the key, stability is not an issue. Stability is also not an issue if all keys are different.

Unstable sorting algorithms can be specially implemented to be stable. One way of doing this is to artificially extend the key comparison, so that comparisons between two objects with otherwise equal keys are decided using the order of the entries in the original input list as a tie-breaker. Remembering this order, however, may require additional time and space.

An example of stable sort on playing cards. When the cards are sorted by rank with a stable sort, the two 5s must remain in the same order in the sorted output that they were originally in. When they are sorted with a non-stable sort, the 5s may end up in the opposite order in the sorted output.

Sorting: Stability



Gregory Schoenmakers, Favourite languages: BASIC, Forth, C and Java

Answered May 14

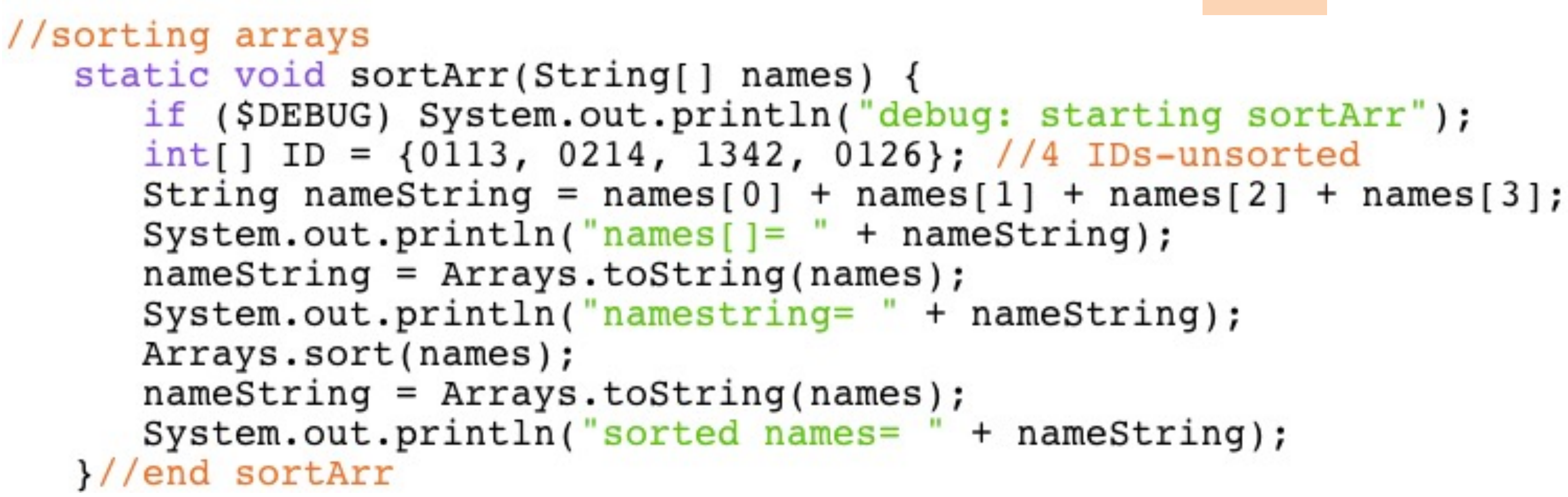
There is no such thing as “more stable”. A sorting algorithm is either stable or not. “Stable” means that items with the same key value appear in the sorted list in the same order as they did in the unsorted list.

Quicksort (on arrays) is not stable but mergesort can be made stable so it would be preferred if stability was a criterion.

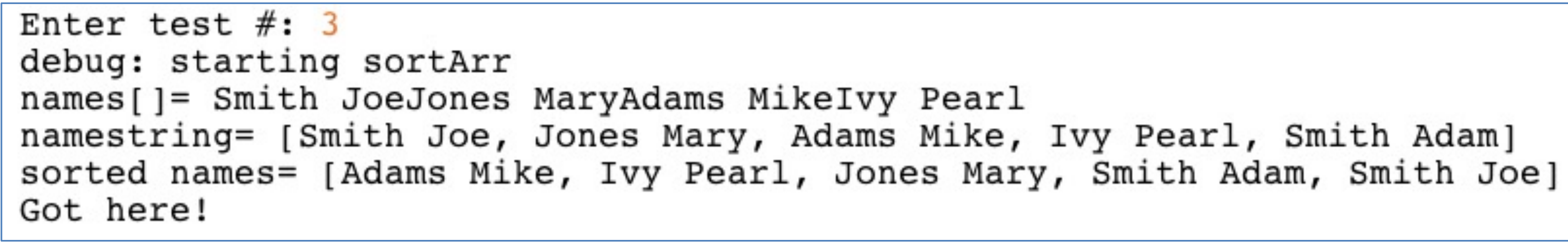
Bucketsort is also a stable sort so if the keys are integers and the range is similar to the number of items to be sorted then bucketsort would be “better” than mergesort.

Array Sort

Test 3



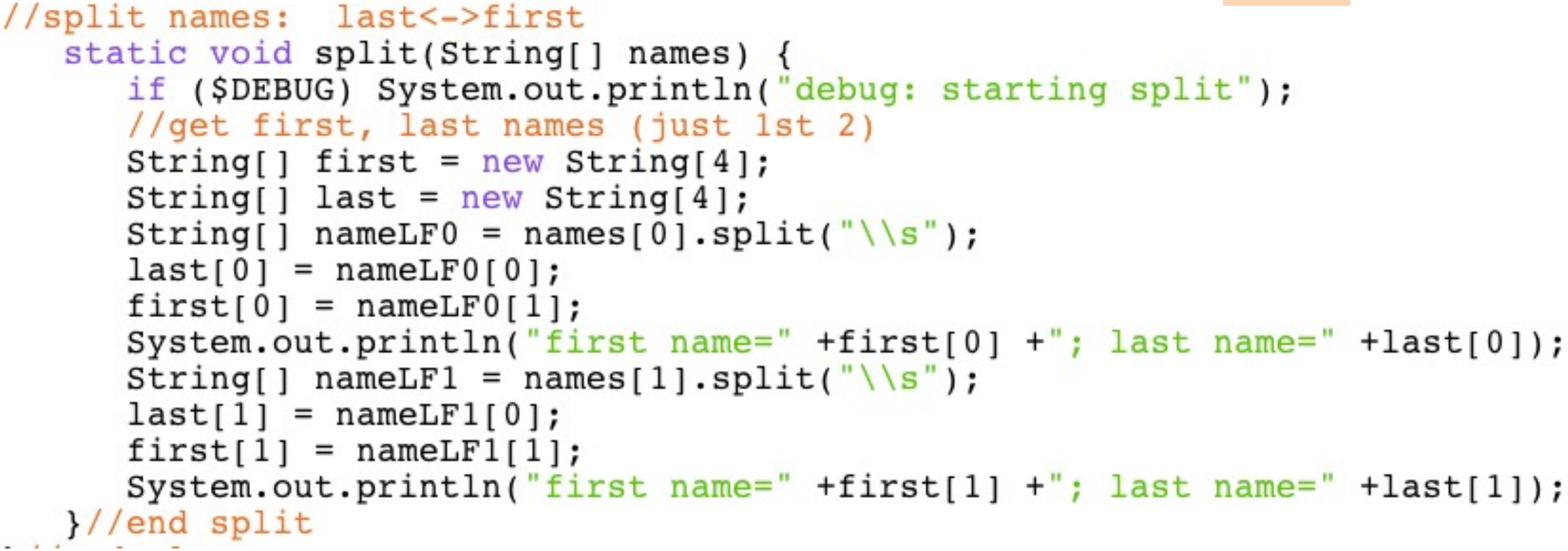
```
//sorting arrays
   static void sortArr(String[] names) {
      if ($DEBUG) System.out.println("debug: starting sortArr");
      int[] ID = {0113, 0214, 1342, 0126}; //4 IDs-unsorted
      String nameString = names[0] + names[1] + names[2] + names[3];
      System.out.println("names[]= " + nameString);
      nameString = Arrays.toString(names);
      System.out.println("namestring= " + nameString);
      Arrays.sort(names);
      nameString = Arrays.toString(names);
      System.out.println("sorted names= " + nameString);
   }//end sortArr
```



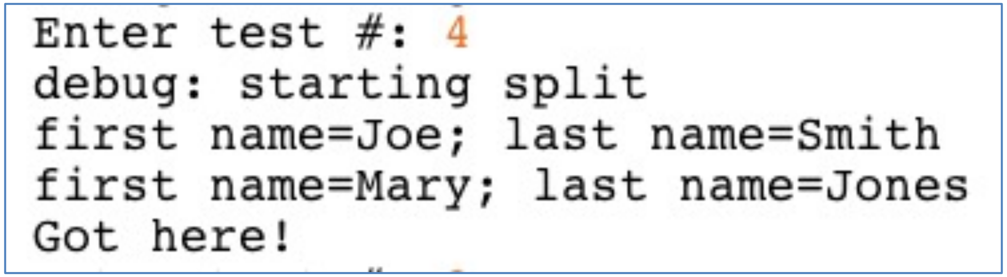
```
Enter test #: 3
debug: starting sortArr
names[]= Smith JoeJones MaryAdams MikeIvy Pearl
namestring= [Smith Joe, Jones Mary, Adams Mike, Ivy Pearl, Smith Adam]
sorted names= [Adams Mike, Ivy Pearl, Jones Mary, Smith Adam, Smith Joe]
Got here!
```

Array Split-Sort

Test 4

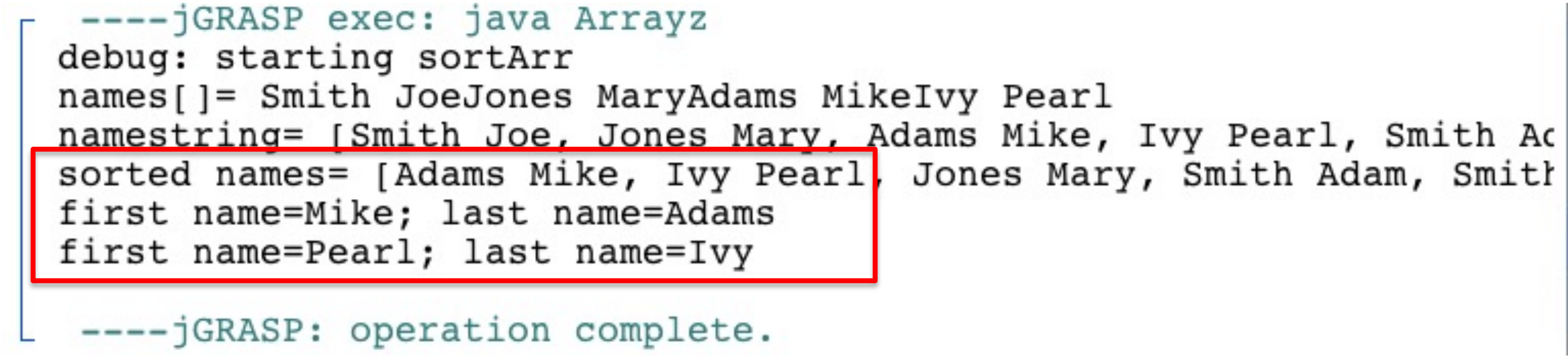


```
//split names:  last<->first
   static void split(String[] names) {
      if ($DEBUG) System.out.println("debug: starting split");
      //get first, last names (just 1st 2)
      String[] first = new String[4];
      String[] last = new String[4];
      String[] nameLF0 = names[0].split("\\s");
      last[0] = nameLF0[0];
      first[0] = nameLF0[1];
      System.out.println("first name=" +first[0] +"; last name=" +last[0]);
      String[] nameLF1 = names[1].split("\\s");
      last[1] = nameLF1[0];
      first[1] = nameLF1[1];
      System.out.println("first name=" +first[1] +"; last name=" +last[1]);
   }//end split
```



```
Enter test #: 4
debug: starting split
first name=Joe; last name=Smith
first name=Mary; last name=Jones
Got here!
```

Array Sort: Result



```
 ----jGRASP exec: java Arrayz
debug: starting sortArr
names[]= Smith JoeJones MaryAdams MikeIvy Pearl
namestring= [Smith Joe, Jones Mary, Adams Mike, Ivy Pearl, Smith Ac
sorted names= [Adams Mike, Ivy Pearl, Jones Mary, Smith Adam, Smith
first name=Mike; last name=Adams
first name=Pearl; last name=Ivy

 ----jGRASP: operation complete.
```

Chapter 8

Multi-Dimensional Arrays

Arrays

2D

❖ Database example

columns = ***fields***

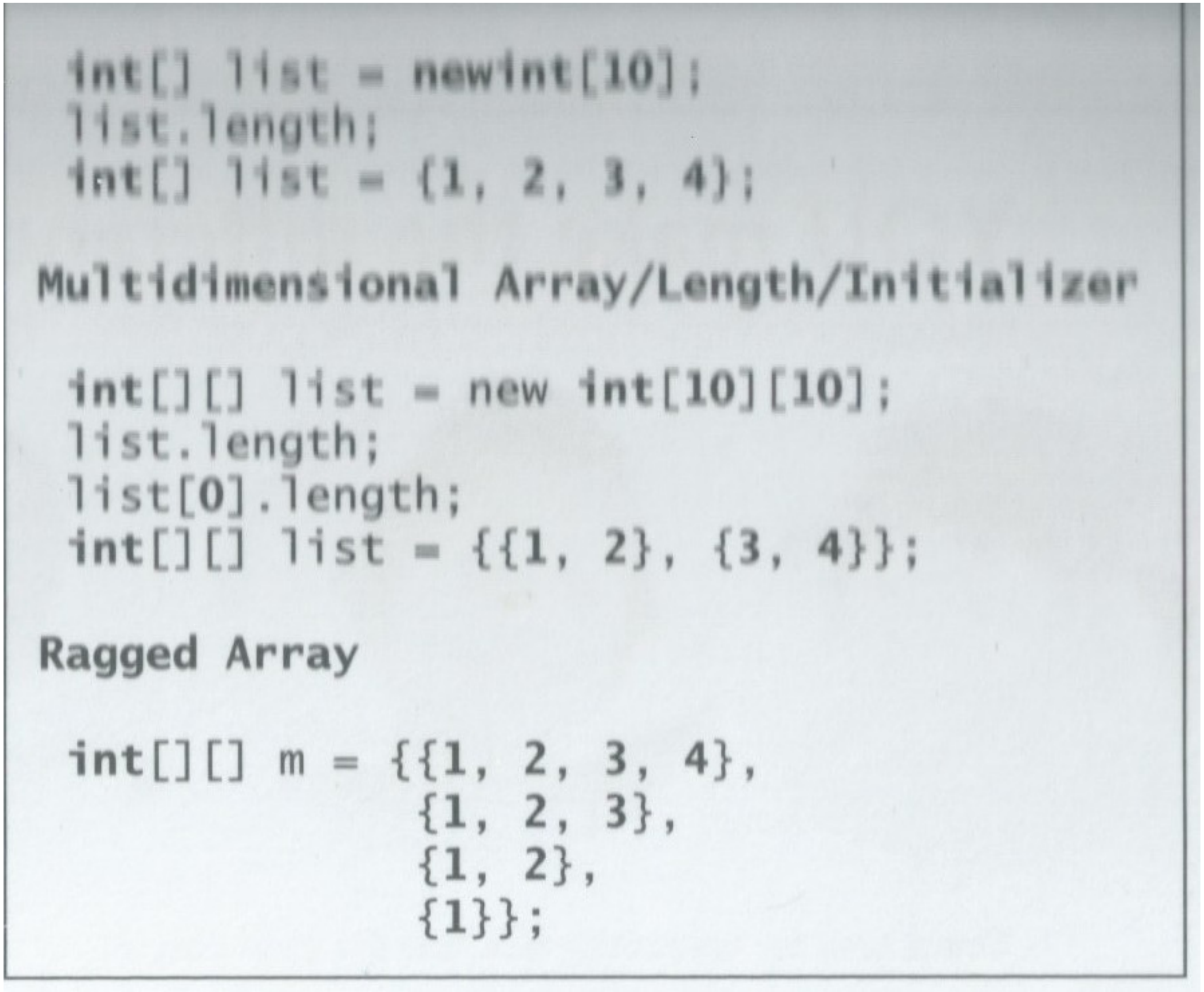
rows = ***records***

Array ptr

value[0] [4]

	0	1	2	3	4
	ID	**last**	**first**	**year**	**major**
0	12345	Smith	Joe	3	cs
1	12565	Jones	Mary	2	cit
2	12475	McDon	Chuck	1	ce
3	12787	Torres	Maria	4	math
4	12465	Stein	Mark	3	cs
5	12983	Chu	Song	1	cs
6	12167	Al-Dhu	Ahmad	1	ce

2D Arrays [ ] [ ]



```
int[] list = newint[10];
list.length;
int[] list = {1, 2, 3, 4};
```

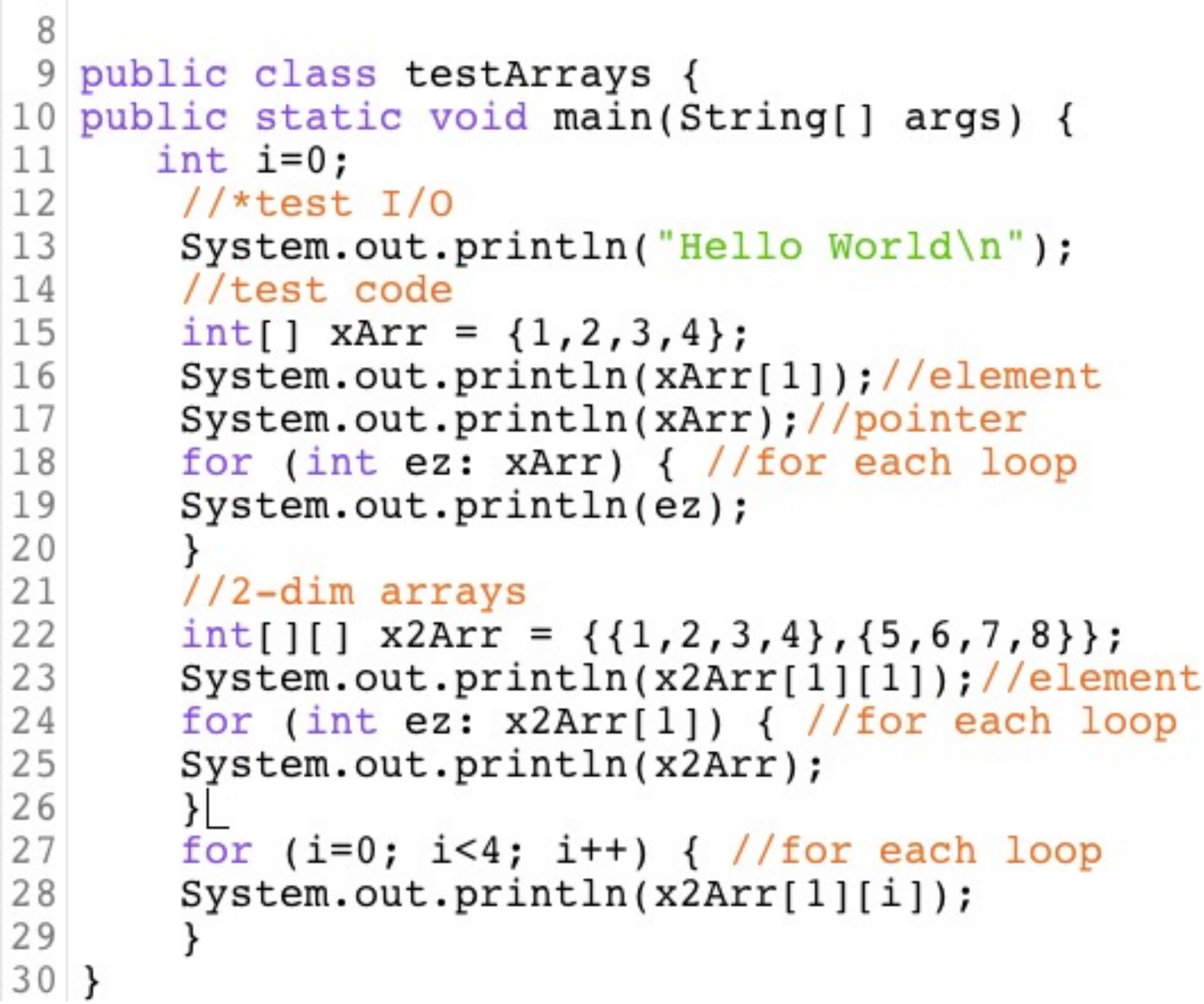
Multidimensional Array/Length/Initializer

```
int[][] list = new int[10][10];
list.length;
list[0].length;
int[][] list = {{1, 2}, {3, 4}};
```

Ragged Array

```
int[][] m = {{1, 2, 3, 4},
             {1, 2, 3},
             {1, 2},
             {1}};
```

2D Arrays



```

public class testArrays {
public static void main(String[] args) {
   int i=0;
    //*test I/O
    System.out.println("Hello World\n");
    //test code
    int[] xArr = {1,2,3,4};
    System.out.println(xArr[1]);//element
    System.out.println(xArr);//pointer
    for (int ez: xArr) { //for each loop
    System.out.println(ez);
    }
    //2-dim arrays
    int[][] x2Arr = {{1,2,3,4},{5,6,7,8}};
    System.out.println(x2Arr[1][1]);//element
    for (int ez: x2Arr[1]) { //for each loop
    System.out.println(x2Arr);
    }
    for (i=0; i<4; i++) { //for each loop
    System.out.println(x2Arr[1][i]);
    }
}
```

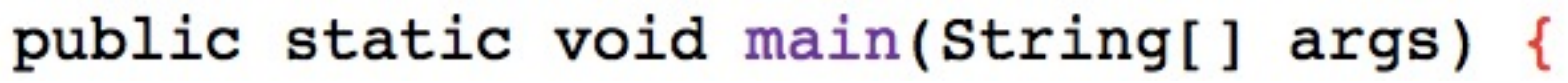
Section

Command Line Args

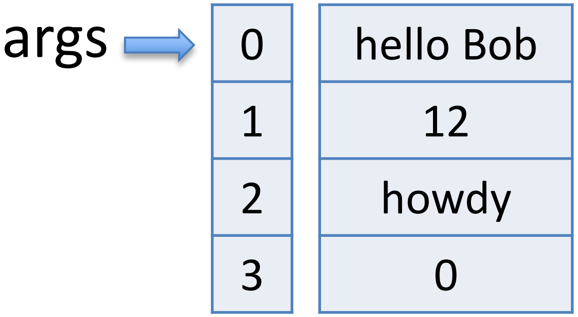
Command Line Args

sec 7.13

```
java program  “hello Bob”  12  howdy  0
```



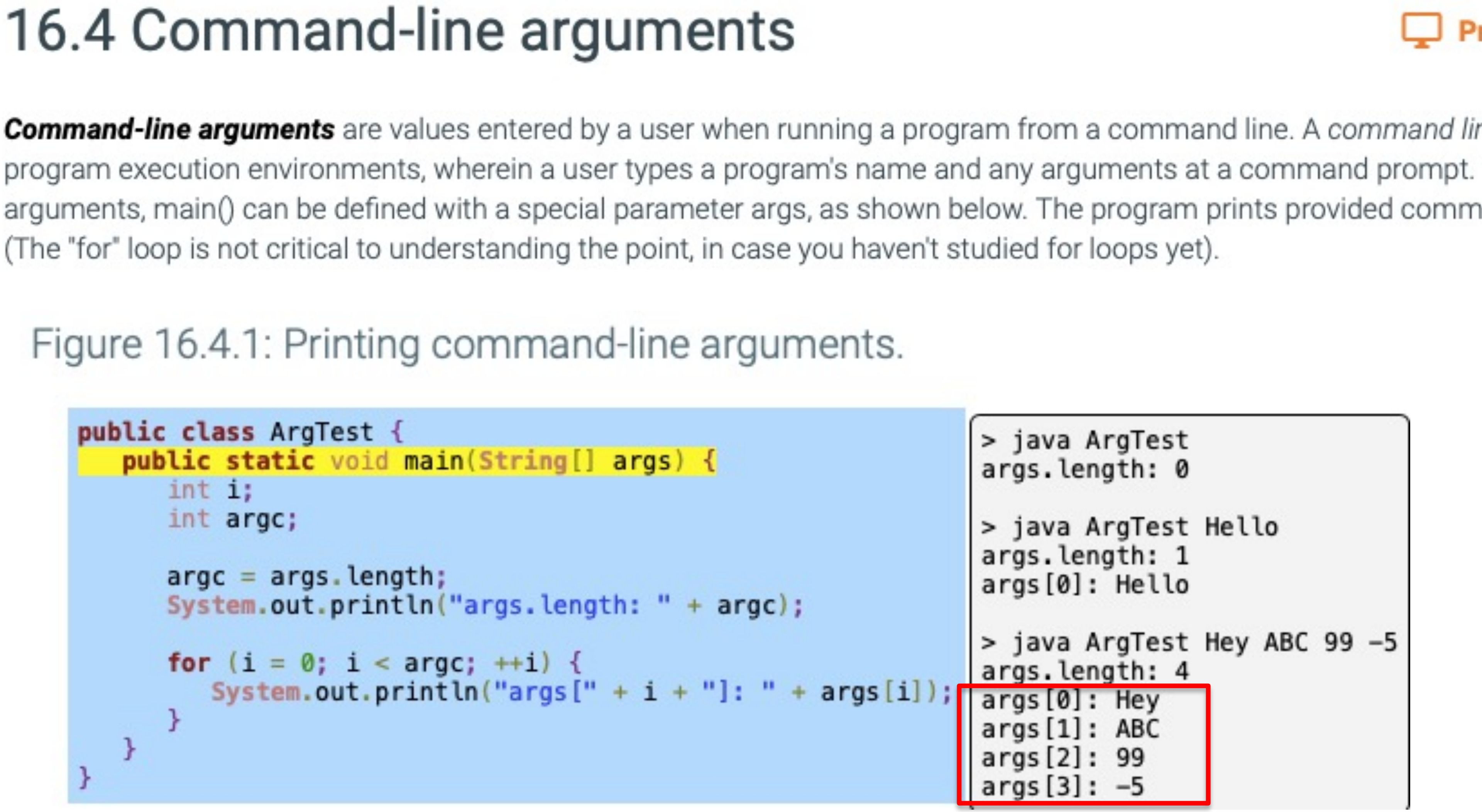
```
public static void main(String[] args) {
```



args

0	hello Bob
1	12
2	howdy
3	0

Command Line Args



16.4 Command-line arguments

Command-line arguments are values entered by a user when running a program from a command line. A *command lir*
program execution environments, wherein a user types a program's name and any arguments at a command prompt.
arguments, main() can be defined with a special parameter args, as shown below. The program prints provided comm
(The "for" loop is not critical to understanding the point, in case you haven't studied for loops yet).

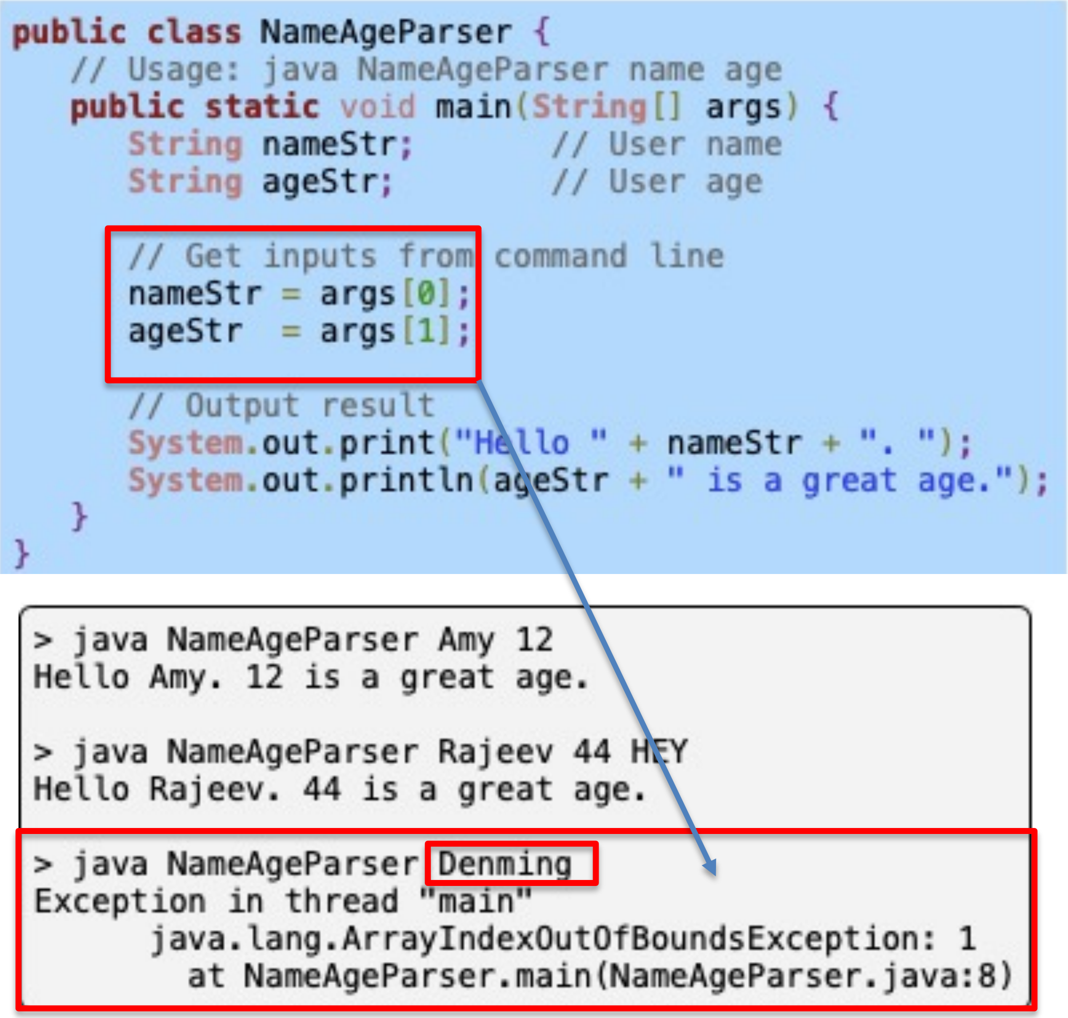
Figure 16.4.1: Printing command-line arguments.

```
public class ArgTest {
   public static void main(String[] args) {
      int i;
      int argc;

      argc = args.length;
      System.out.println("args.length: " + argc);

      for (i = 0; i < argc; ++i) {
         System.out.println("args[" + i + "]: " + args[i]);
      }
   }
}
```

```
> java ArgTest
args.length: 0

> java ArgTest Hello
args.length: 1
args[0]: Hello

> java ArgTest Hey ABC 99 -5
args.length: 4
args[0]: Hey
args[1]: ABC
args[2]: 99
args[3]: -5
```

Command Line Args

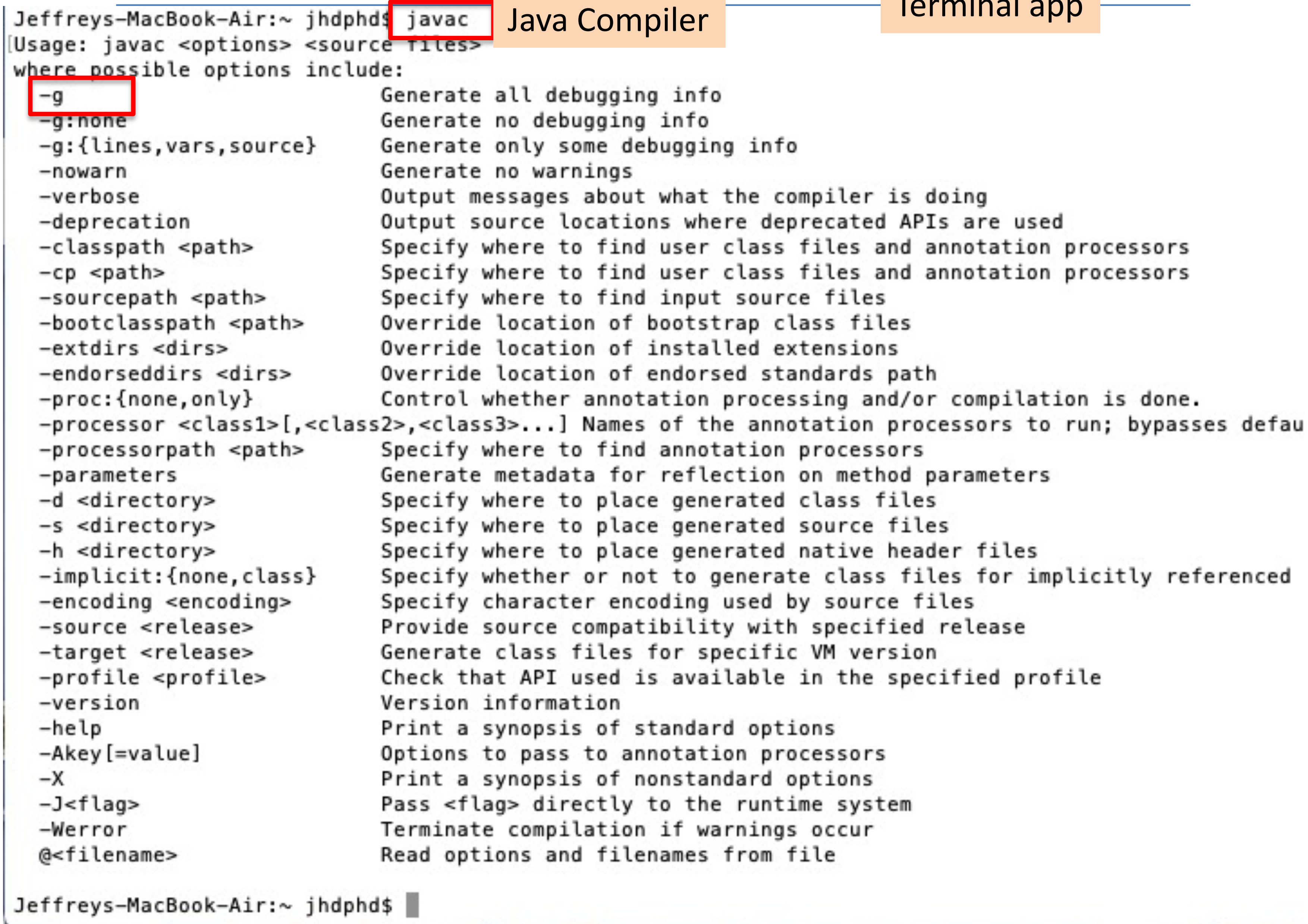
Figure 16.4.2: Simple use of command-line arguments.



```
public class NameAgeParser {
   // Usage: java NameAgeParser name age
   public static void main(String[] args) {
      String nameStr;       // User name
      String ageStr;        // User age

      // Get inputs from command line
      nameStr = args[0];
      ageStr  = args[1];

      // Output result
      System.out.print("Hello " + nameStr + ". ");
      System.out.println(ageStr + " is a great age.");
   }
}
```

```
> java NameAgeParser Amy 12
Hello Amy. 12 is a great age.

> java NameAgeParser Rajeev 44 HEY
Hello Rajeev. 44 is a great age.

> java NameAgeParser Denming
Exception in thread "main"
      java.lang.ArrayIndexOutOfBoundsException: 1
        at NameAgeParser.main(NameAgeParser.java:8)
```

Command Line Args



Java Compiler

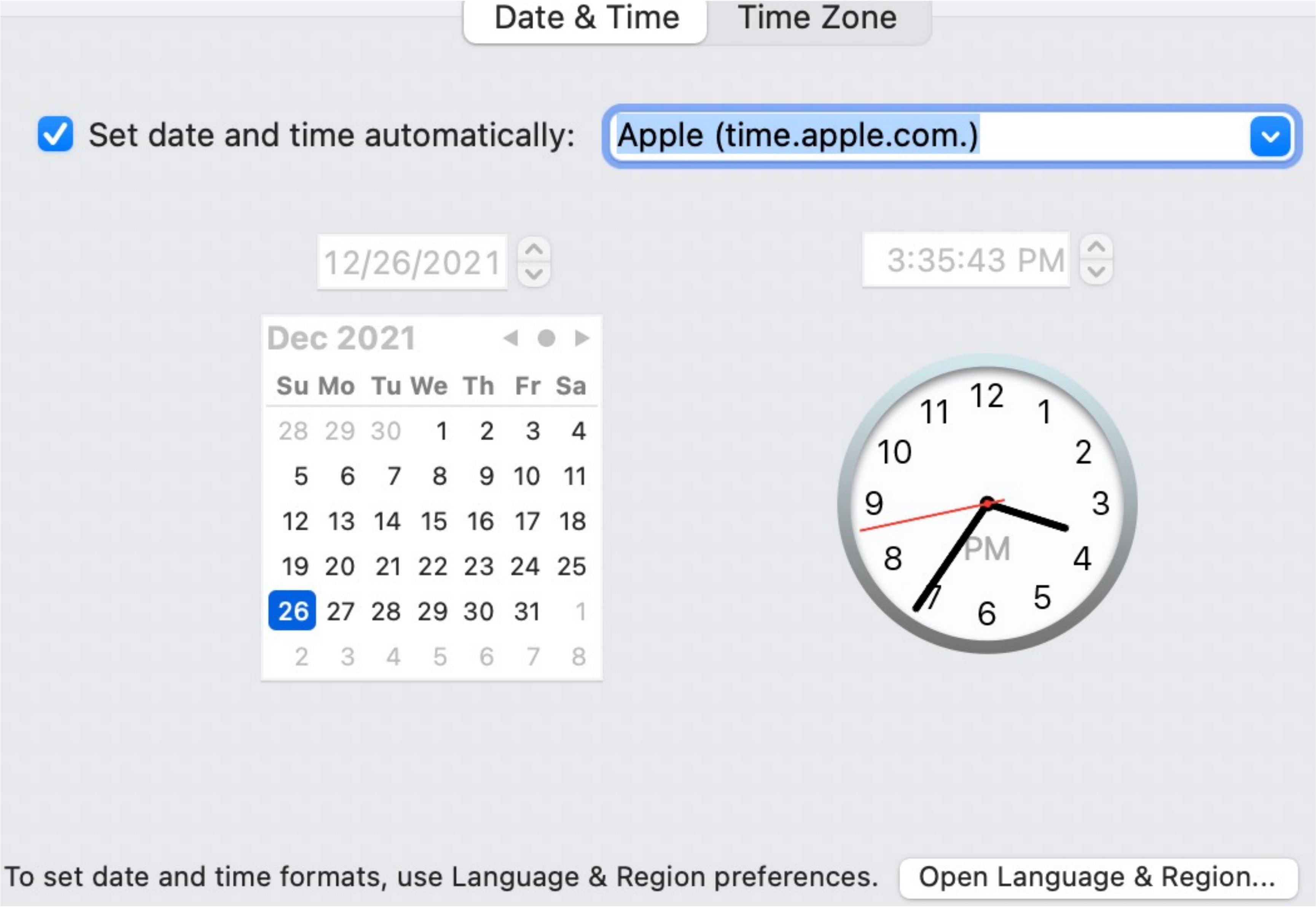
Terminal app

```
Jeffreys-MacBook-Air:~ jhdphd$ javac
Usage: javac <options> <source files>
where possible options include:
  -g                         Generate all debugging info
  -g:none                    Generate no debugging info
  -g:{lines,vars,source}     Generate only some debugging info
  -nowarn                    Generate no warnings
  -verbose                   Output messages about what the compiler is doing
  -deprecation               Output source locations where deprecated APIs are used
  -classpath <path>          Specify where to find user class files and annotation processors
  -cp <path>                 Specify where to find user class files and annotation processors
  -sourcepath <path>         Specify where to find input source files
  -bootclasspath <path>      Override location of bootstrap class files
  -extdirs <dirs>            Override location of installed extensions
  -endorseddirs <dirs>       Override location of endorsed standards path
  -proc:{none,only}          Control whether annotation processing and/or compilation is done.
  -processor <class1>[,<class2>,<class3>...] Names of the annotation processors to run; bypasses defau
  -processorpath <path>      Specify where to find annotation processors
  -parameters                Generate metadata for reflection on method parameters
  -d <directory>             Specify where to place generated class files
  -s <directory>             Specify where to place generated source files
  -h <directory>             Specify where to place generated native header files
  -implicit:{none,class}     Specify whether or not to generate class files for implicitly referenced
  -encoding <encoding>       Specify character encoding used by source files
  -source <release>          Provide source compatibility with specified release
  -target <release>          Generate class files for specific VM version
  -profile <profile>         Check that API used is available in the specified profile
  -version                   Version information
  -help                      Print a synopsis of standard options
  -Akey[=value]              Options to pass to annotation processors
  -X                         Print a synopsis of nonstandard options
  -J<flag>                   Pass <flag> directly to the runtime system
  -Werror                    Terminate compilation if warnings occur
  @<filename>                Read options and filenames from file

Jeffreys-MacBook-Air:~ jhdphd$ 
```

Section

Date & Time

Data & Time (Mac)



Date & Time | Time Zone

☑ Set date and time automatically: Apple (time.apple.com.)

12/26/2021

3:35:43 PM

Dec 2021

Su	Mo	Tu	We	Th	Fr	Sa
28	29	30	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	1
2	3	4	5	6	7	8

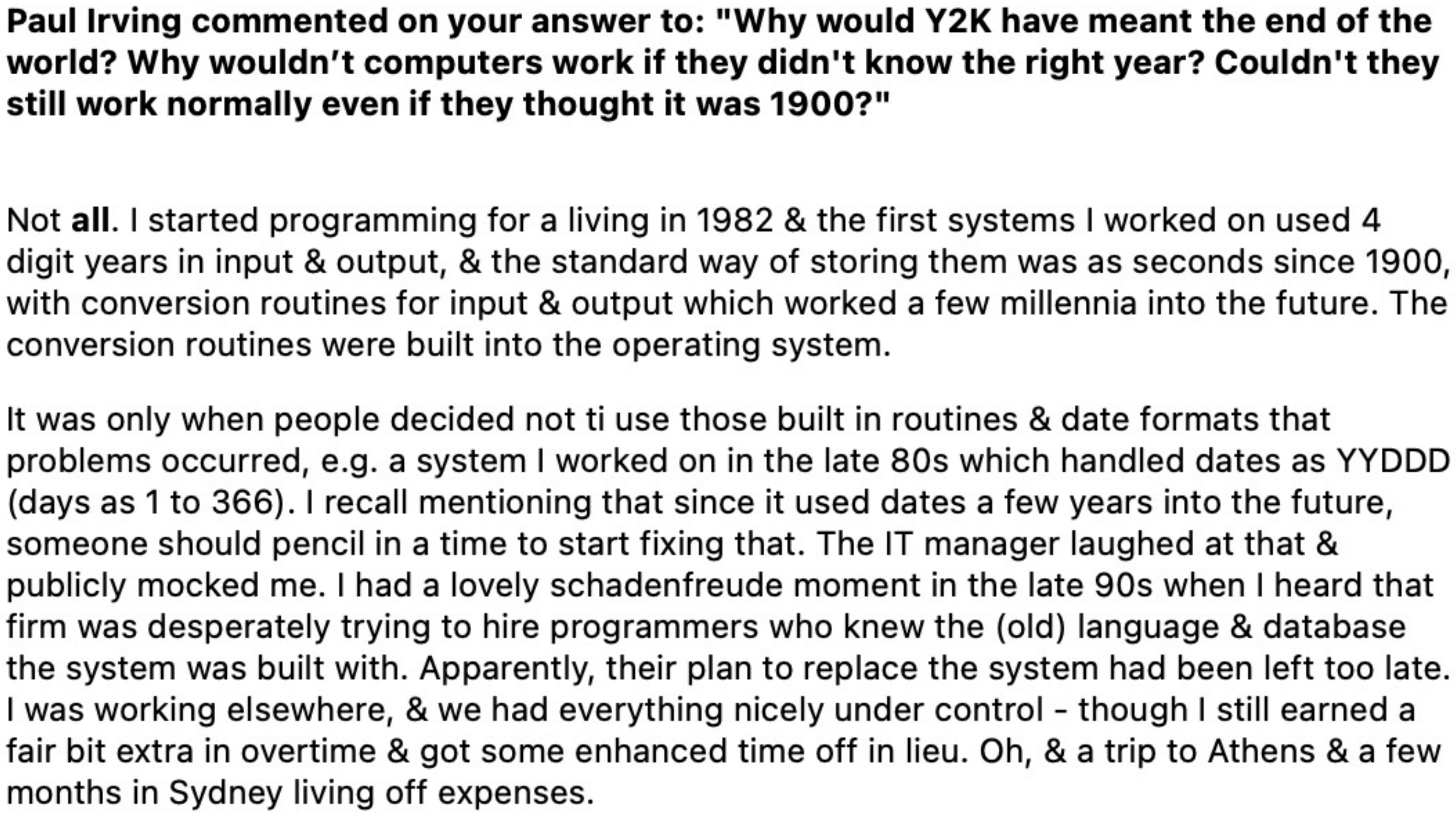
To set date and time formats, use Language & Region preferences. Open Language & Region...

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP110

Y2K Dates

DR JEFF SOFTWARE
INDIE APP DEVELOPER

Quora

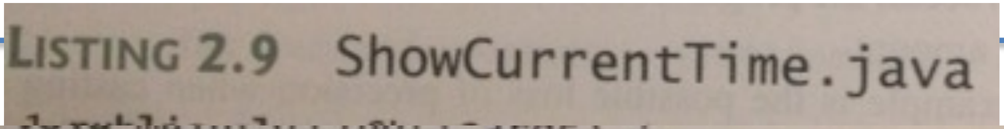


Paul Irving commented on your answer to: "Why would Y2K have meant the end of the world? Why wouldn’t computers work if they didn't know the right year? Couldn't they still work normally even if they thought it was 1900?"

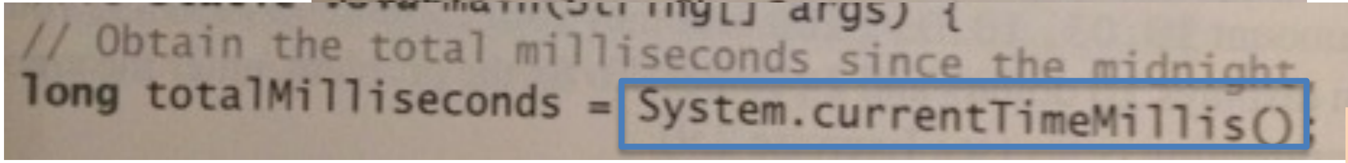
Not **all**. I started programming for a living in 1982 & the first systems I worked on used 4 digit years in input & output, & the standard way of storing them was as seconds since 1900, with conversion routines for input & output which worked a few millennia into the future. The conversion routines were built into the operating system.

It was only when people decided not ti use those built in routines & date formats that problems occurred, e.g. a system I worked on in the late 80s which handled dates as YYDDD (days as 1 to 366). I recall mentioning that since it used dates a few years into the future, someone should pencil in a time to start fixing that. The IT manager laughed at that & publicly mocked me. I had a lovely schadenfreude moment in the late 90s when I heard that firm was desperately trying to hire programmers who knew the (old) language & database the system was built with. Apparently, their plan to replace the system had been left too late. I was working elsewhere, & we had everything nicely under control - though I still earned a fair bit extra in overtime & got some enhanced time off in lieu. Oh, & a trip to Athens & a few months in Sydney living off expenses.

Time (*System* Class)

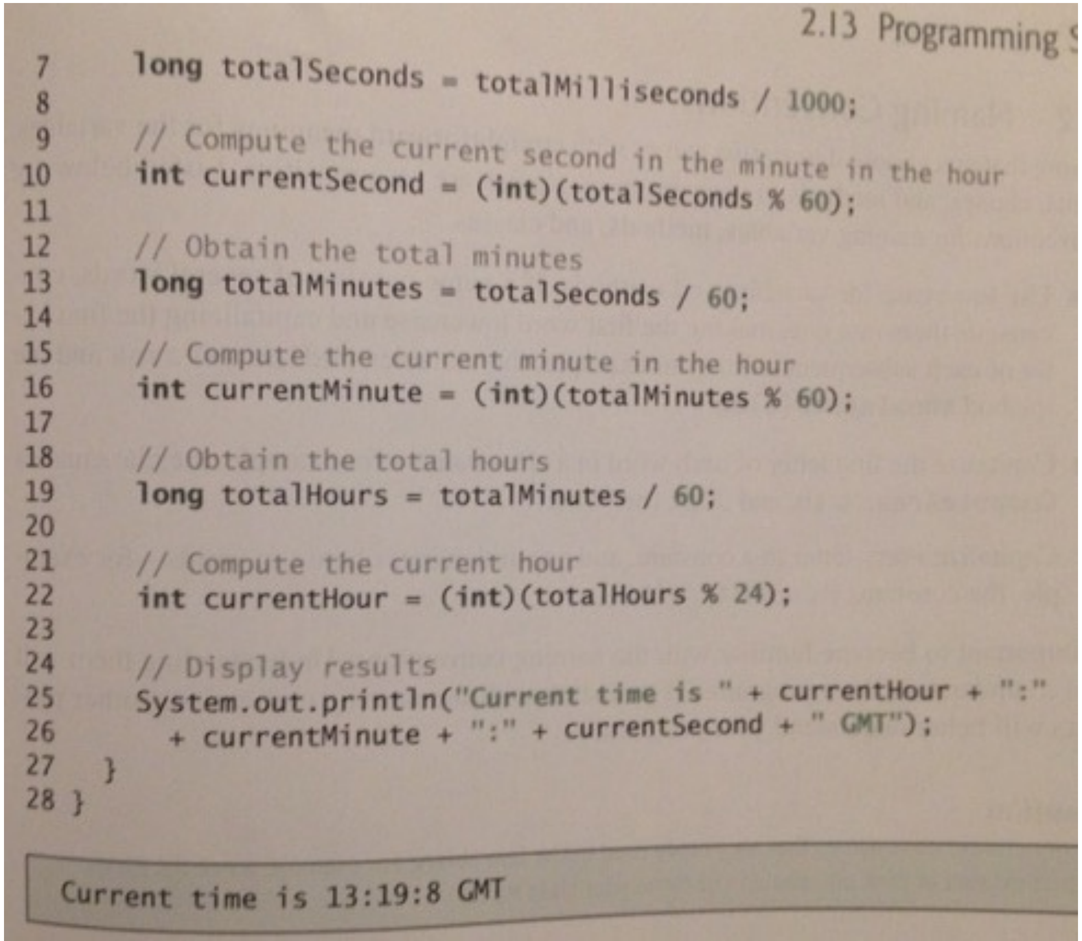


LISTING 2.9 ShowCurrentTime.java



```
    main(String[] args) {
// Obtain the total milliseconds since the midnight, Jan 1, 1970
long totalMilliseconds = System.currentTimeMillis();
```

- ❖ org 1-1-1970



2.13 Programming S

```
      long totalSeconds = totalMilliseconds / 1000;

      // Compute the current second in the minute in the hour
      int currentSecond = (int)(totalSeconds % 60);

      // Obtain the total minutes
      long totalMinutes = totalSeconds / 60;

      // Compute the current minute in the hour
      int currentMinute = (int)(totalMinutes % 60);

      // Obtain the total hours
      long totalHours = totalMinutes / 60;

      // Compute the current hour
      int currentHour = (int)(totalHours % 24);

      // Display results
      System.out.println("Current time is " + currentHour + ":"
        + currentMinute + ":" + currentSecond + " GMT");
    }
}
```

```
Current time is 13:19:8 GMT
```

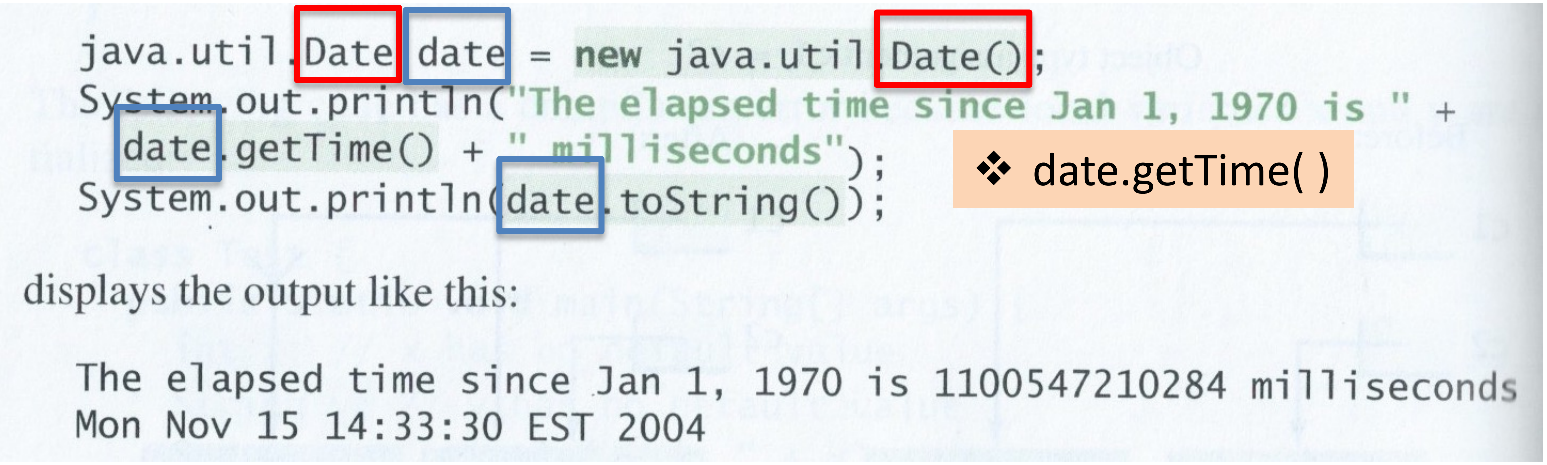
Date & Time (*Date* Class)

- Date date = new Date( )
- date.toString( )

```
System.out.println(date.toString( ) );
```

```
System.out.println(date);
```

prints out → Wed Sep 21 14 : 33 : 30 EST 2016



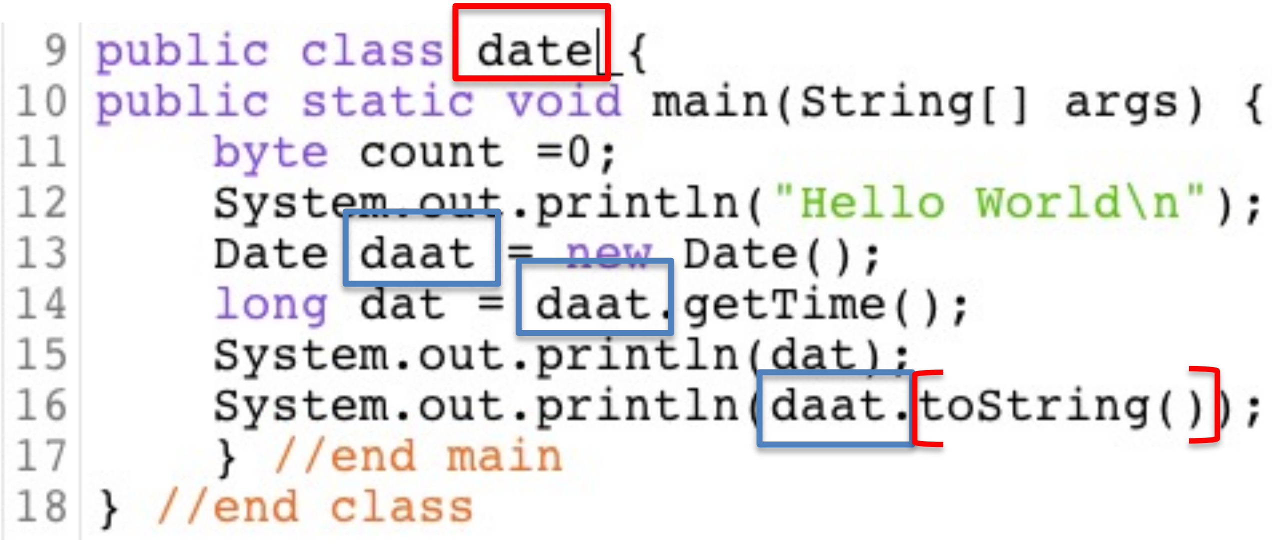
```
java.util.Date date = new java.util.Date();
System.out.println("The elapsed time since Jan 1, 1970 is " +
  date.getTime() + " milliseconds");
System.out.println(date.toString());
```

displays the output like this:

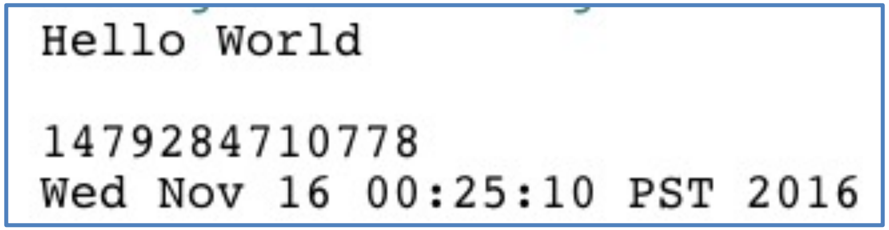
```
The elapsed time since Jan 1, 1970 is 1100547210284 milliseconds
Mon Nov 15 14:33:30 EST 2004
```

- date.getTime( )

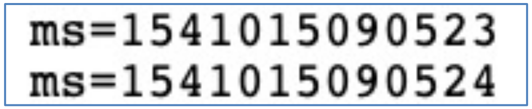
Date & Time Examples



```
public class date {
public static void main(String[] args) {
    byte count =0;
    System.out.println("Hello World\n");
    Date daat = new Date();
    long dat = daat.getTime();
    System.out.println(dat);
    System.out.println(daat.toString());
    } //end main
} //end class
```



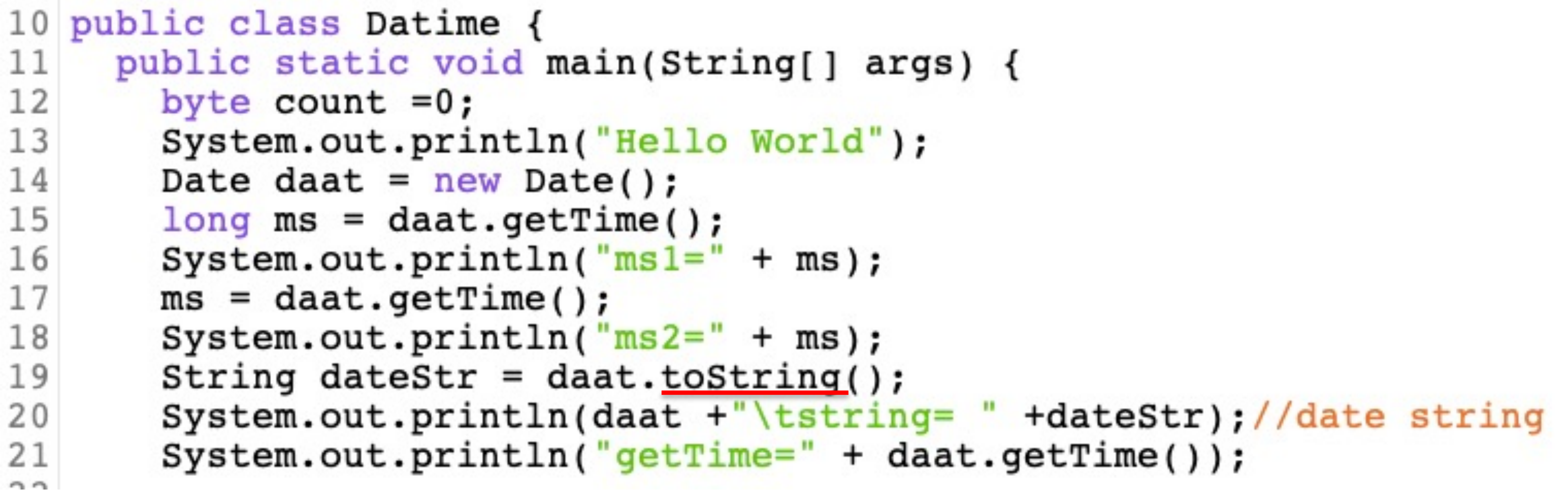
```
Hello World

1479284710778
Wed Nov 16 00:25:10 PST 2016
```



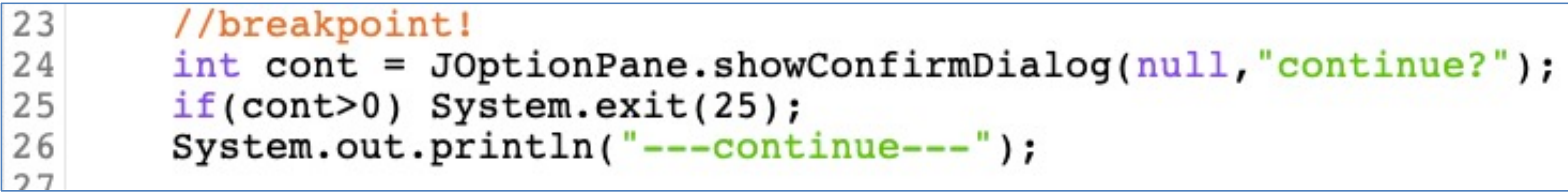
```
ms=1541015090523
ms=1541015090524
```

Miniscule (or 0) difference

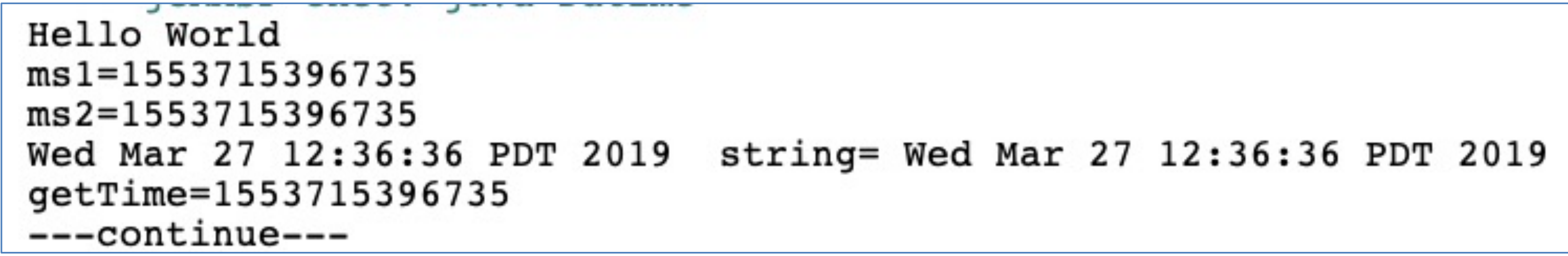
Date & Time Examples



```
public class Datime {
  public static void main(String[] args) {
    byte count =0;
    System.out.println("Hello World");
    Date daat = new Date();
    long ms = daat.getTime();
    System.out.println("ms1=" + ms);
    ms = daat.getTime();
    System.out.println("ms2=" + ms);
    String dateStr = daat.toString();
    System.out.println(daat +"\tstring= " +dateStr);//date string
    System.out.println("getTime=" + daat.getTime());
```

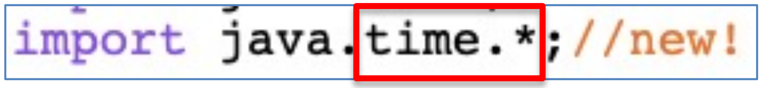


```
    //breakpoint!
    int cont = JOptionPane.showConfirmDialog(null,"continue?");
    if(cont>0) System.exit(25);
    System.out.println("---continue---");
```

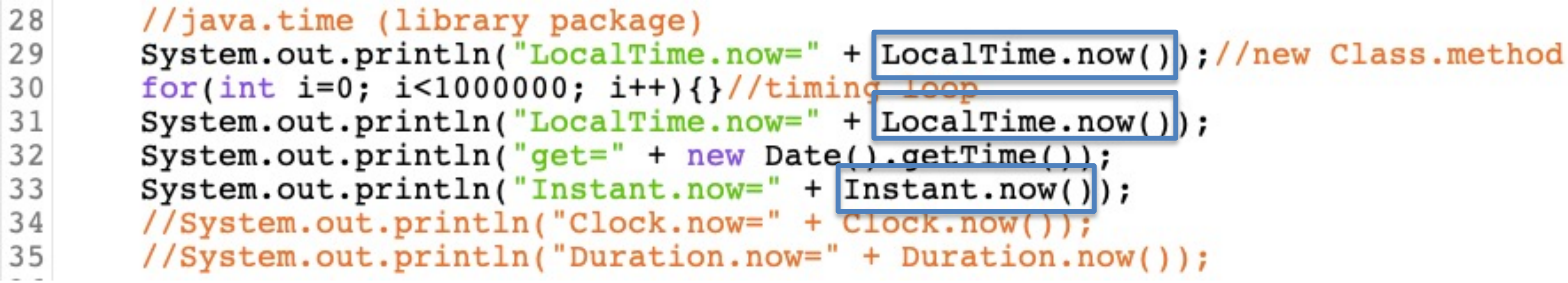


```
Hello World
ms1=1553715396735
ms2=1553715396735
Wed Mar 27 12:36:36 PDT 2019  string= Wed Mar 27 12:36:36 PDT 2019
getTime=1553715396735
---continue---
```

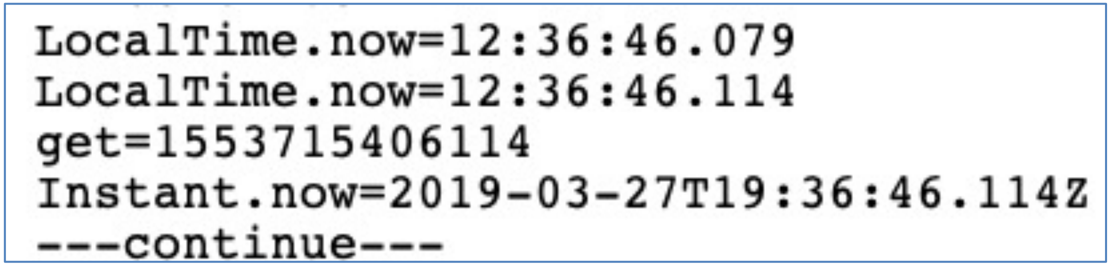
Date & Time Examples



```
import java.time.*;//new!
```



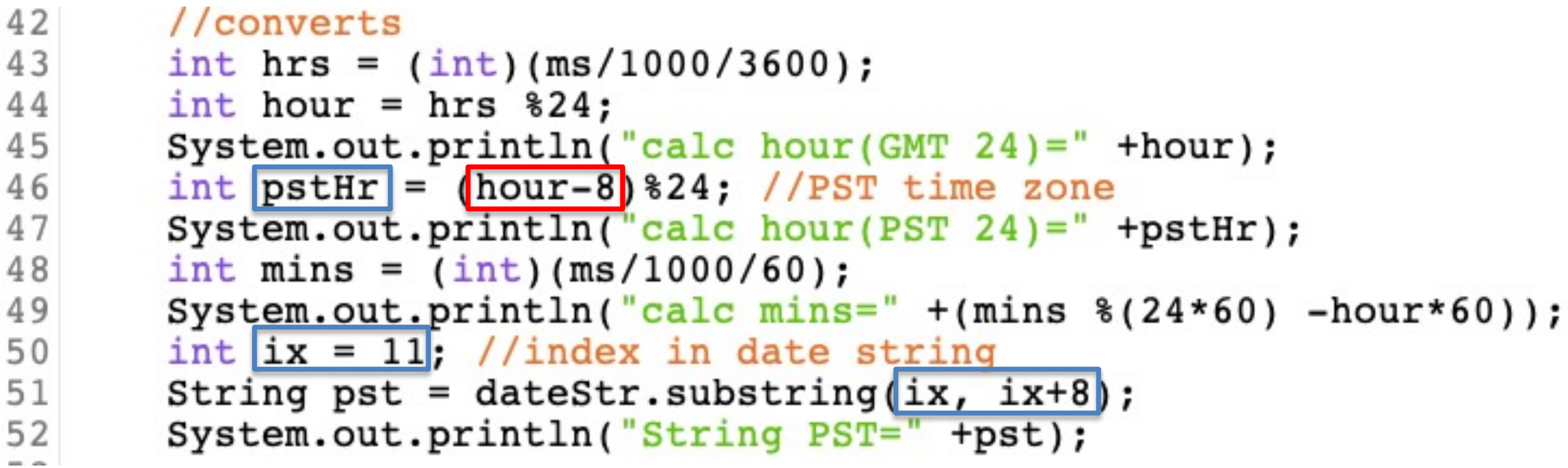
```
        //java.time (library package)
        System.out.println("LocalTime.now=" + LocalTime.now());//new Class.method
        for(int i=0; i<1000000; i++){}//timing loop
        System.out.println("LocalTime.now=" + LocalTime.now());
        System.out.println("get=" + new Date().getTime());
        System.out.println("Instant.now=" + Instant.now());
        //System.out.println("Clock.now=" + Clock.now());
        //System.out.println("Duration.now=" + Duration.now());
```



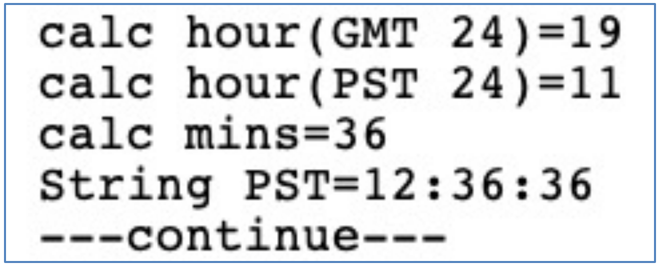
```
LocalTime.now=12:36:46.079
LocalTime.now=12:36:46.114
get=1553715406114
Instant.now=2019-03-27T19:36:46.114Z
---continue---
```

Date & Time Examples

PST<>PDT



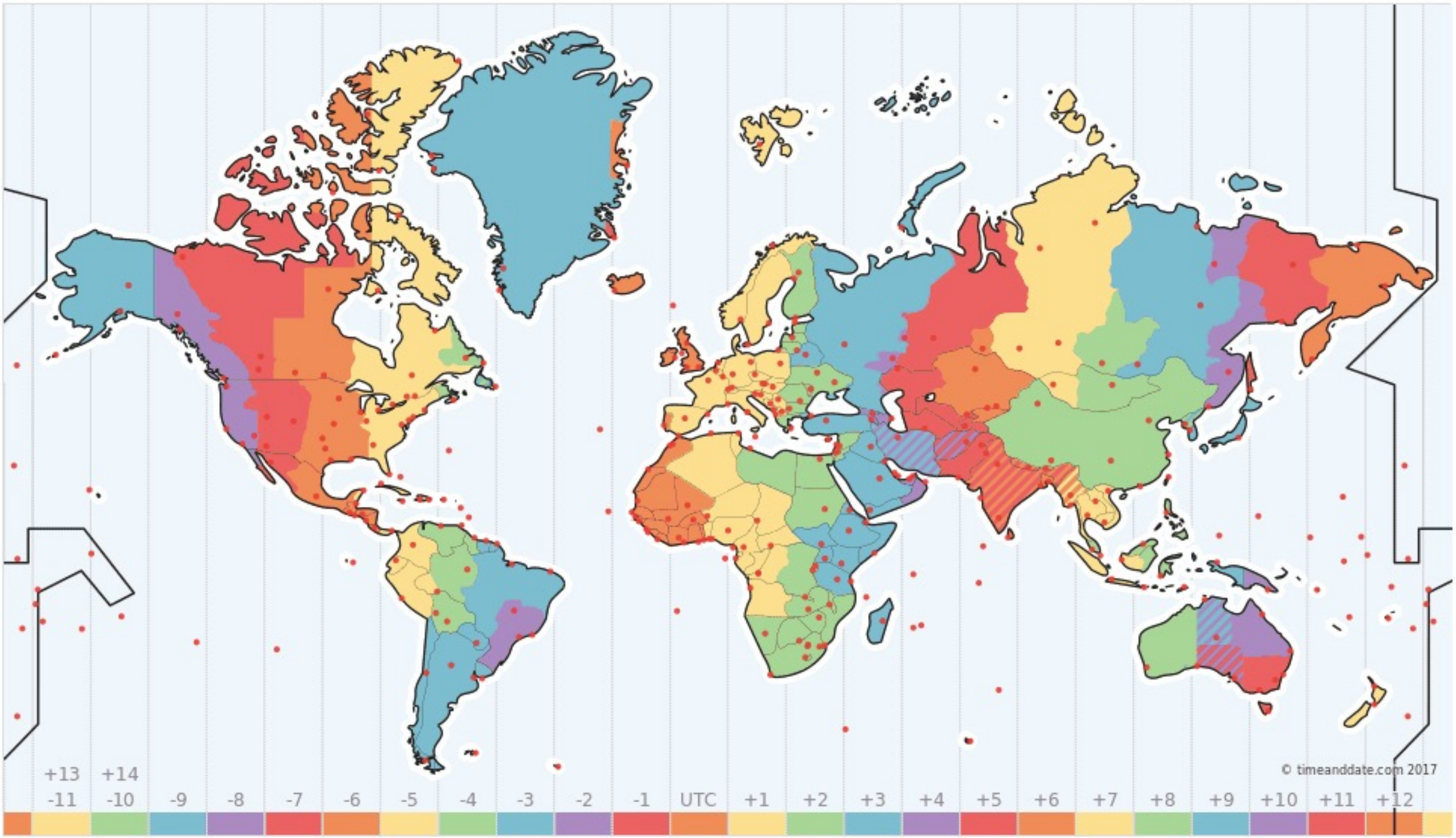
```
    //converts
    int hrs = (int)(ms/1000/3600);
    int hour = hrs %24;
    System.out.println("calc hour(GMT 24)=" +hour);
    int pstHr = (hour-8)%24; //PST time zone
    System.out.println("calc hour(PST 24)=" +pstHr);
    int mins = (int)(ms/1000/60);
    System.out.println("calc mins=" +(mins %(24*60) -hour*60));
    int ix = 11; //index in date string
    String pst = dateStr.substring(ix, ix+8);
    System.out.println("String PST=" +pst);
```



```
calc hour(GMT 24)=19
calc hour(PST 24)=11
calc mins=36
String PST=12:36:36
---continue---
```

Time Zones

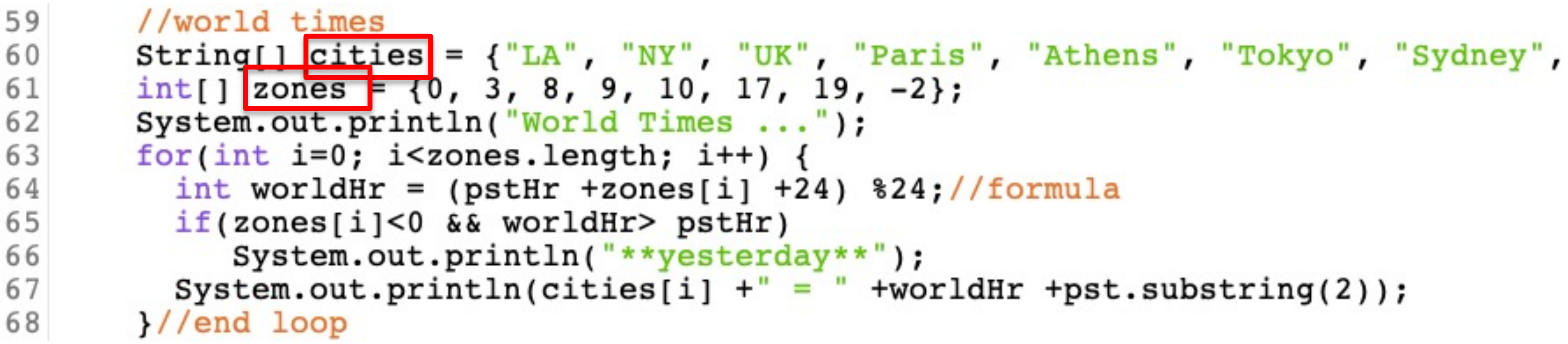
Time Zone Map



© timeanddate.com 2017

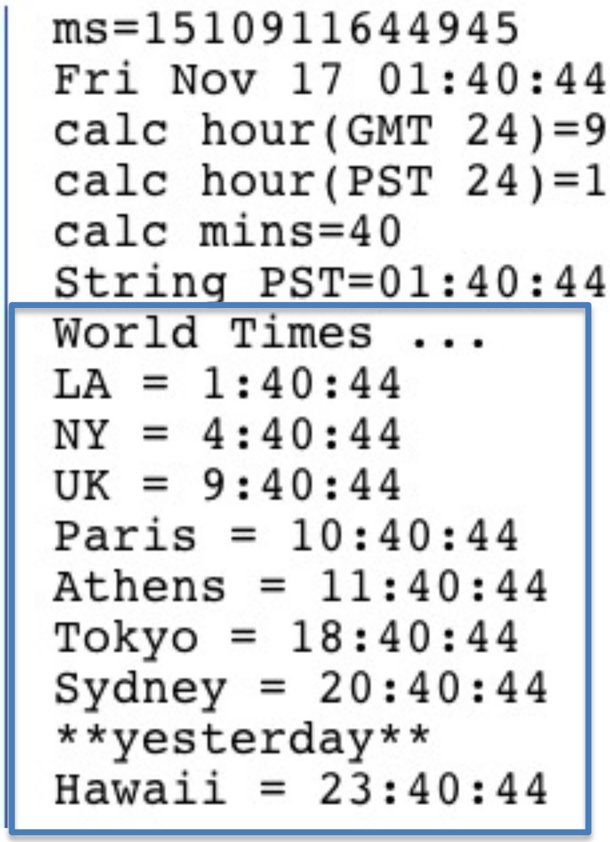
Date & Time Examples

World Times

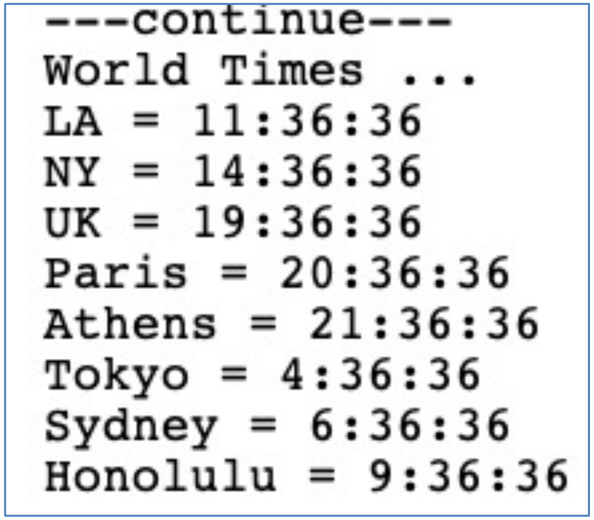


```
//world times
String[] cities = {"LA", "NY", "UK", "Paris", "Athens", "Tokyo", "Sydney",
int[] zones = {0, 3, 8, 9, 10, 17, 19, -2};
System.out.println("World Times ...");
for(int i=0; i<zones.length; i++) {
  int worldHr = (pstHr +zones[i] +24) %24;//formula
  if(zones[i]<0 && worldHr> pstHr)
     System.out.println("**yesterday**");
  System.out.println(cities[i] +" = " +worldHr +pst.substring(2));
}//end loop
```

Date & Time Example

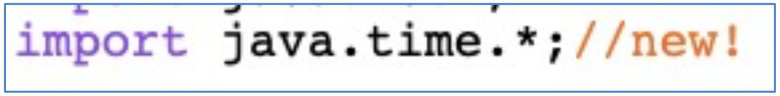


```
ms=1510911644945
Fri Nov 17 01:40:44 PST 2017
calc hour(GMT 24)=9
calc hour(PST 24)=1
calc mins=40
String PST=01:40:44
World Times ...
LA = 1:40:44
NY = 4:40:44
UK = 9:40:44
Paris = 10:40:44
Athens = 11:40:44
Tokyo = 18:40:44
Sydney = 20:40:44
**yesterday**
Hawaii = 23:40:44
```

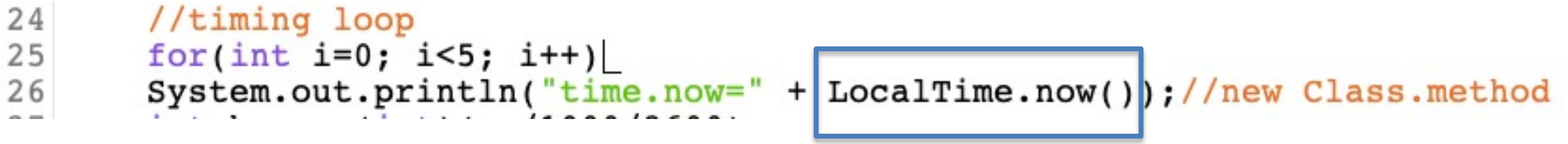


```
---continue---
World Times ...
LA = 11:36:36
NY = 14:36:36
UK = 19:36:36
Paris = 20:36:36
Athens = 21:36:36
Tokyo = 4:36:36
Sydney = 6:36:36
Honolulu = 9:36:36
```

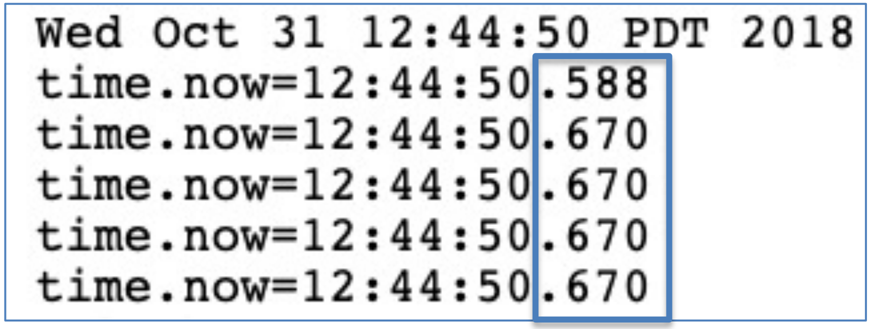
New Class.method



```
import java.time.*;//new!
```

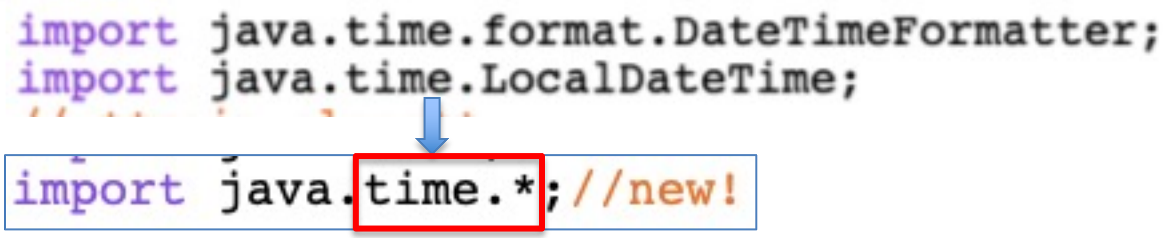


```
    //timing loop
    for(int i=0; i<5; i++)
    System.out.println("time.now=" + LocalTime.now());//new Class.method
```



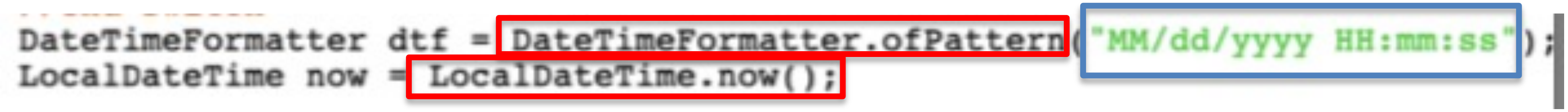
```
Wed Oct 31 12:44:50 PDT 2018
time.now=12:44:50.588
time.now=12:44:50.670
time.now=12:44:50.670
time.now=12:44:50.670
time.now=12:44:50.670
```

Date & Time More

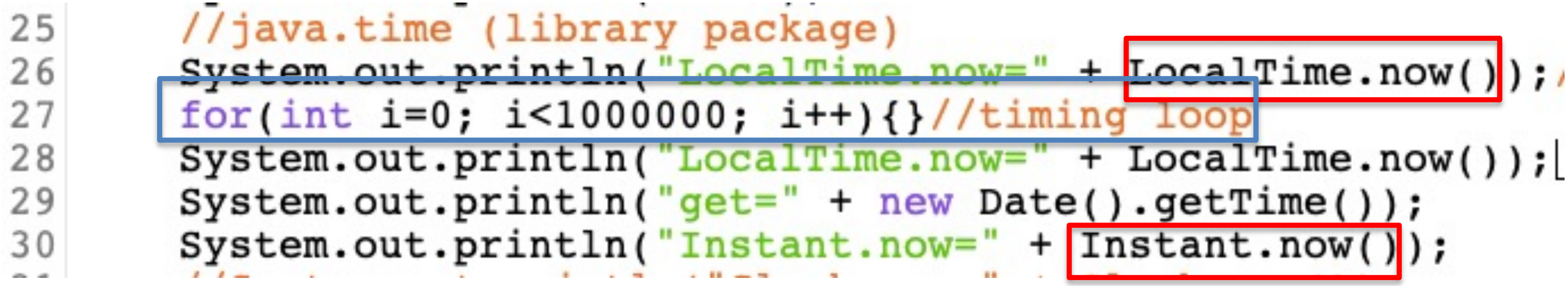


```
import java.time.format.DateTimeFormatter;
import java.time.LocalDateTime;
```

```
import java.time.*;//new!
```

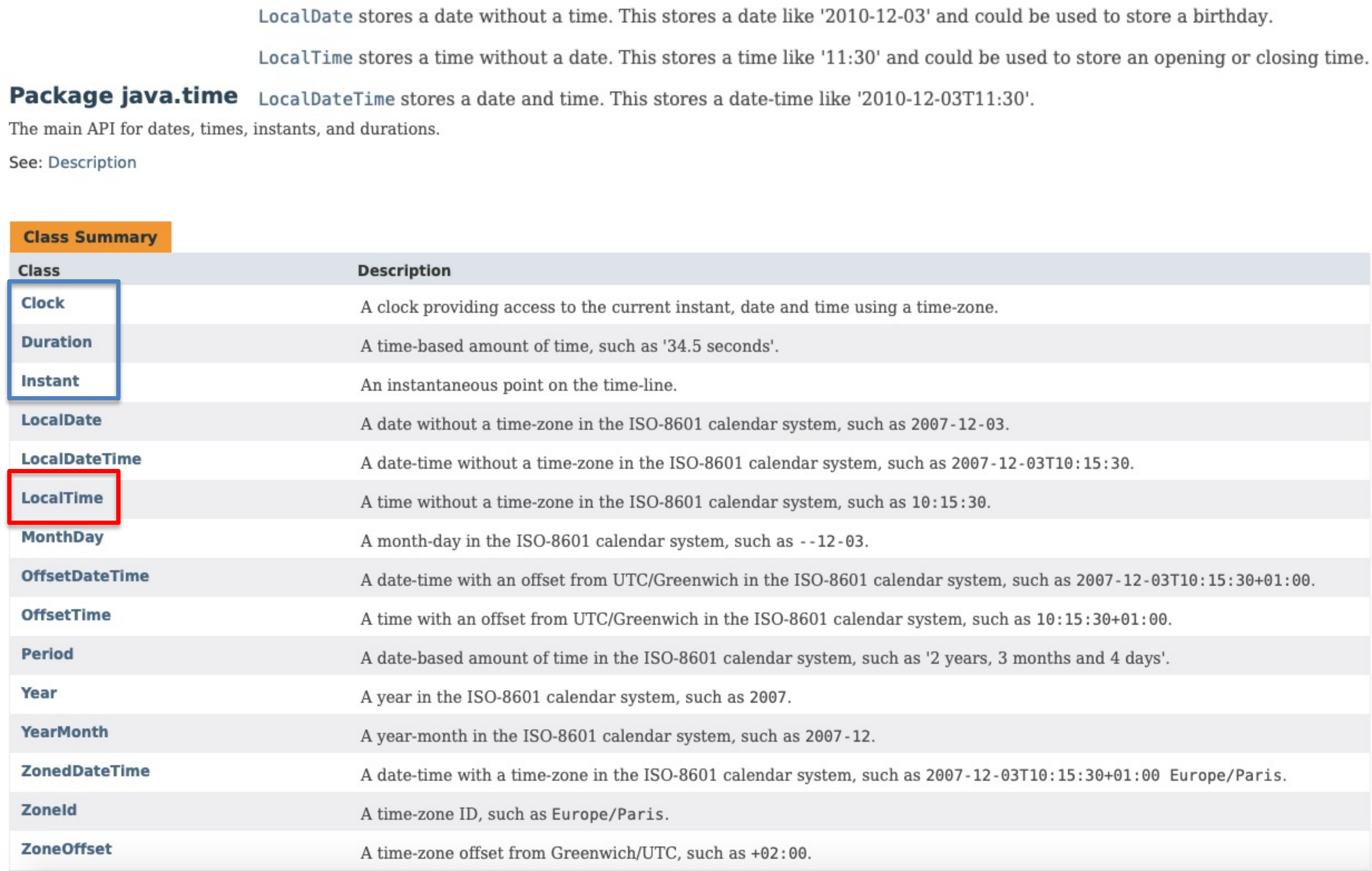


```
DateTimeFormatter dtf = DateTimeFormatter.ofPattern("MM/dd/yyyy HH:mm:ss");
LocalDateTime now = LocalDateTime.now();
```



```
    //java.time (library package)
    System.out.println("LocalTime.now=" + LocalTime.now());
    for(int i=0; i<1000000; i++){}//timing loop
    System.out.println("LocalTime.now=" + LocalTime.now());
    System.out.println("get=" + new Date().getTime());
    System.out.println("Instant.now=" + Instant.now());
```

java.time



`LocalDate` stores a date without a time. This stores a date like '2010-12-03' and could be used to store a birthday.

`LocalTime` stores a time without a date. This stores a time like '11:30' and could be used to store an opening or closing time.

`LocalDateTime` stores a date and time. This stores a date-time like '2010-12-03T11:30'.

Package java.time

The main API for dates, times, instants, and durations.

See: Description

Class Summary

Class	Description
Clock	A clock providing access to the current instant, date and time using a time-zone.
Duration	A time-based amount of time, such as '34.5 seconds'.
Instant	An instantaneous point on the time-line.
LocalDate	A date without a time-zone in the ISO-8601 calendar system, such as `2007-12-03`.
LocalDateTime	A date-time without a time-zone in the ISO-8601 calendar system, such as `2007-12-03T10:15:30`.
LocalTime	A time without a time-zone in the ISO-8601 calendar system, such as `10:15:30`.
MonthDay	A month-day in the ISO-8601 calendar system, such as `--12-03`.
OffsetDateTime	A date-time with an offset from UTC/Greenwich in the ISO-8601 calendar system, such as `2007-12-03T10:15:30+01:00`.
OffsetTime	A time with an offset from UTC/Greenwich in the ISO-8601 calendar system, such as `10:15:30+01:00`.
Period	A date-based amount of time in the ISO-8601 calendar system, such as '2 years, 3 months and 4 days'.
Year	A year in the ISO-8601 calendar system, such as `2007`.
YearMonth	A year-month in the ISO-8601 calendar system, such as `2007-12`.
ZonedDateTime	A date-time with a time-zone in the ISO-8601 calendar system, such as `2007-12-03T10:15:30+01:00 Europe/Paris`.
ZoneId	A time-zone ID, such as `Europe/Paris`.
ZoneOffset	A time-zone offset from Greenwich/UTC, such as `+02:00`.

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP110

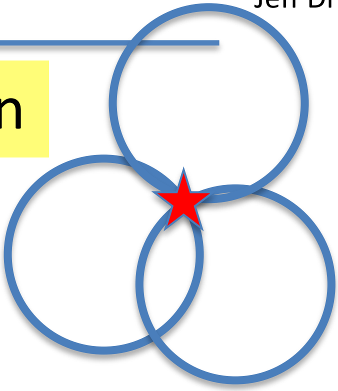
Time & Location: GPS

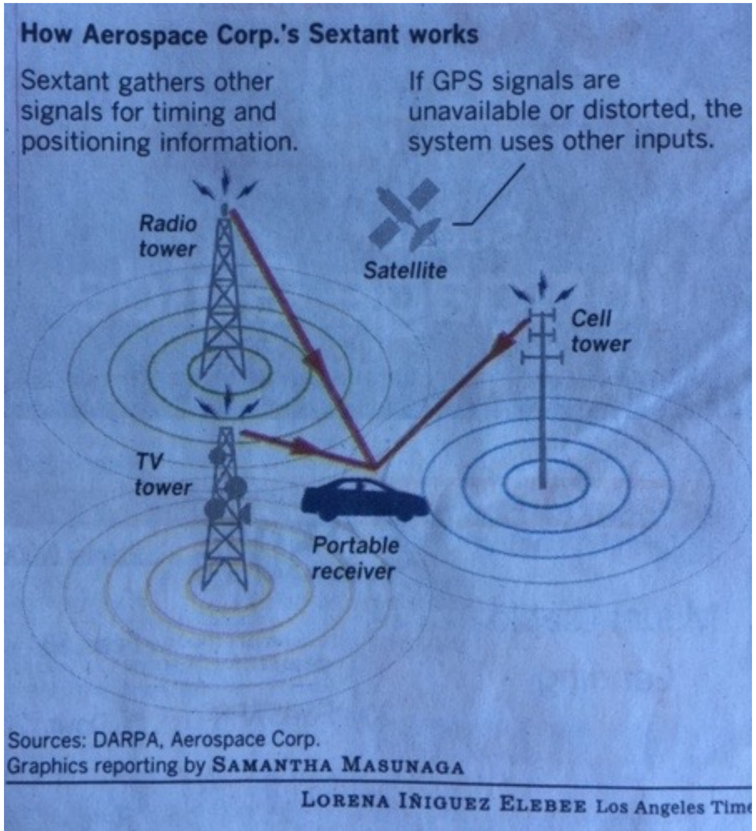
DR JEFF SOFTWARE
INDIE APP DEVELOPER

- ❖ US GPS: 30+ satellites (since 1978)
- ❖ Alternatives as backups
  - ❑ Iridium – 67+ satellites
  - ❑ DARPA *ANS*
  - ❑ USAF/Aerospace *Sextant*

➢ Triangulation

4th satellite
for elevation





How Aerospace Corp.'s Sextant works

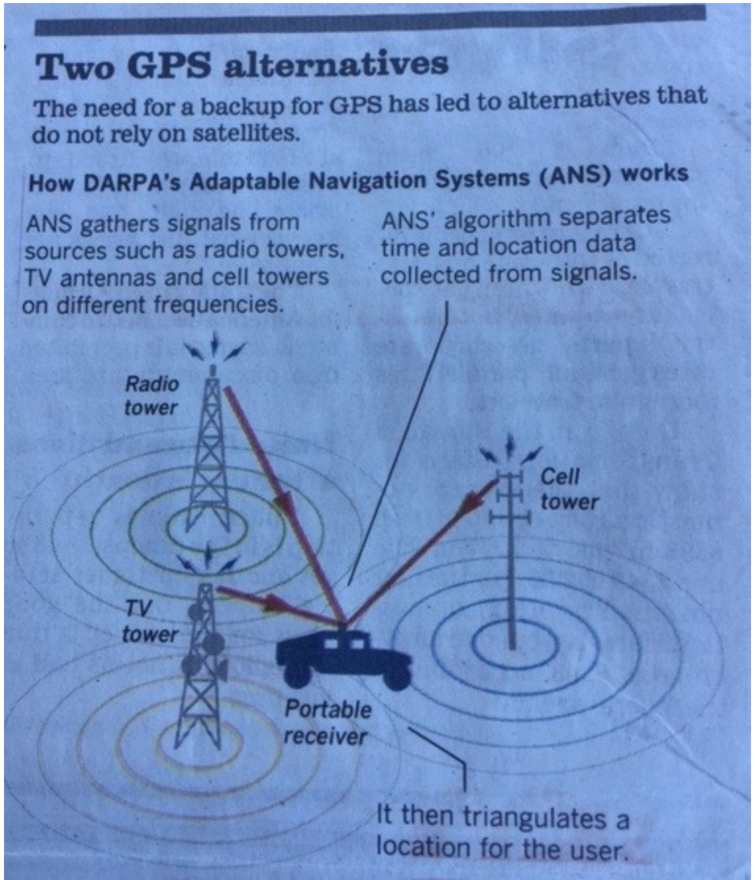
Sextant gathers other signals for timing and positioning information.

If GPS signals are unavailable or distorted, the system uses other inputs.

Radio tower
Satellite
Cell tower
TV tower
Portable receiver

Sources: DARPA, Aerospace Corp.
Graphics reporting by SAMANTHA MASUNAGA

LORENA IÑIGUEZ ELEBEE Los Angeles Time



Two GPS alternatives

The need for a backup for GPS has led to alternatives that do not rely on satellites.

How DARPA's Adaptable Navigation Systems (ANS) works

ANS gathers signals from sources such as radio towers, TV antennas and cell towers on different frequencies.

ANS' algorithm separates time and location data collected from signals.

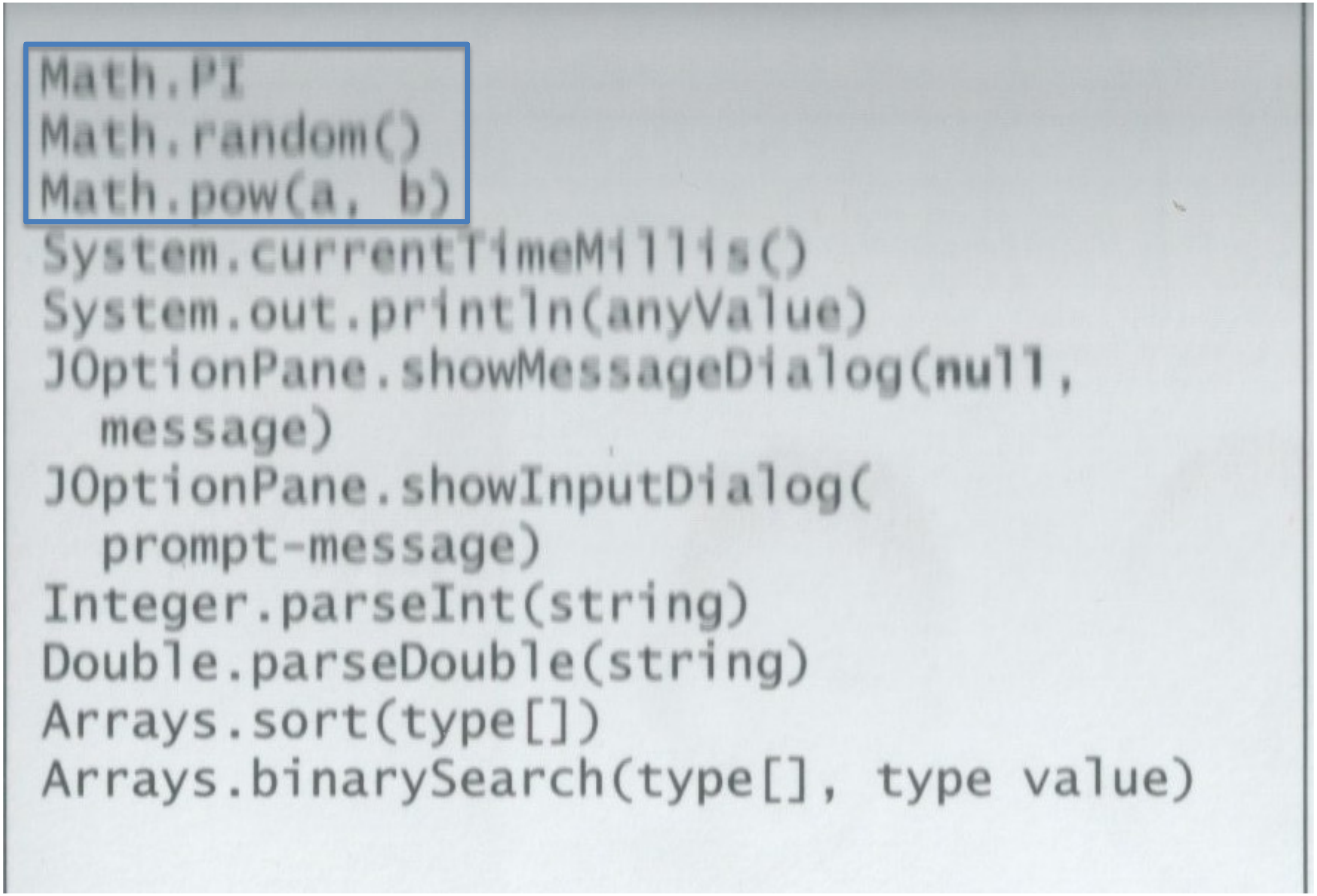
Radio tower
Cell tower
TV tower
Portable receiver

It then triangulates a location for the user.

Math Lib

Random Numbers

Library Functions

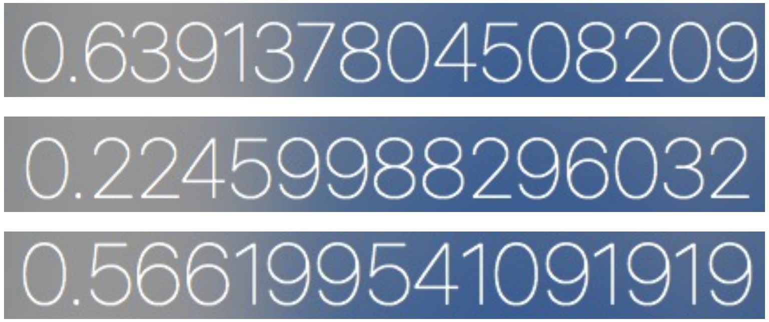


```
Math.PI
Math.random()
Math.pow(a, b)
System.currentTimeMillis()
System.out.println(anyValue)
JOptionPane.showMessageDialog(null,
  message)
JOptionPane.showInputDialog(
  prompt-message)
Integer.parseInt(string)
Double.parseDouble(string)
Arrays.sort(type[])
Arrays.binarySearch(type[], type value)
```

Random Numbers

Math.random( )

returns *double* 0 < N < 1 ⬄ [0-1]
0.000000001 to 0.999999999



0.639137804508209

0.22459988296032

0.566199541091919

*10 → 6.391378045…

for 0 <= N <= 9
use
(int) (Math.random( ) * 10)

*10 → 6.391378045…

int → 6

for 1 <= N <= 10
use
(int) (Math.random( ) * 10) +1

Random Numbers

Generating Random numbers

- Pseudo-random
- Seeded
  - Default = system time (ms)
  - Specified
- Random seeded (2-level, …, N-level)
- Java: **method** + **Class**(seed)

Using Random numbers

- 0 to N-1: (int) (N * Math.random( ));
- 1 to N: (int) (N * Math.random( )) + 1;
- selecting which digit to use (more randomizing)

*10 → $\boxed{6}$. 3 $\boxed{9}$ 1 3 $\boxed{7}$..

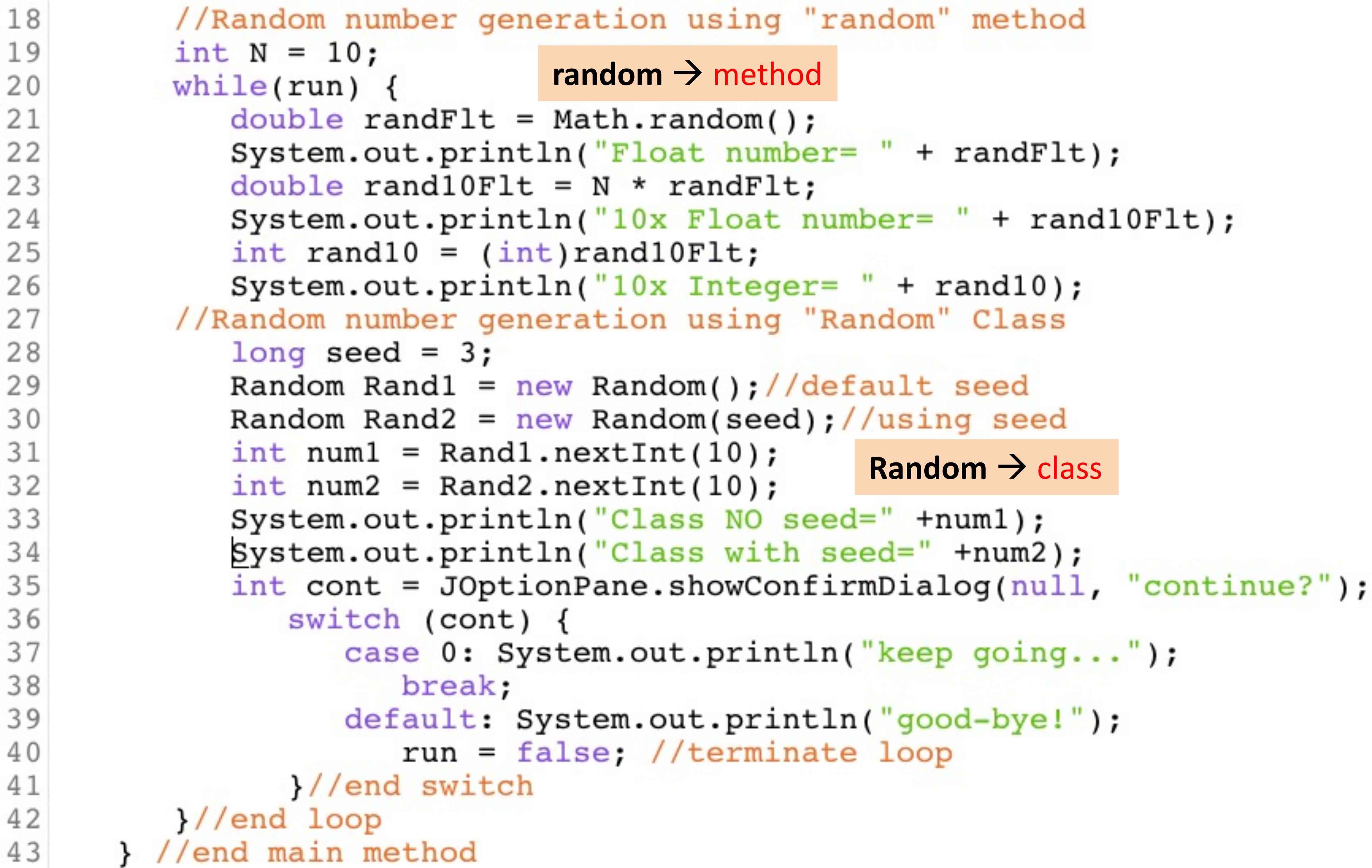
Random Numbers

Alternative

- Class: Random
  - Random Ranx = new Random( );
  - int rand = Ranx.nextInt(10);

> Returns 0..9 {10 values}

Generating Randoms

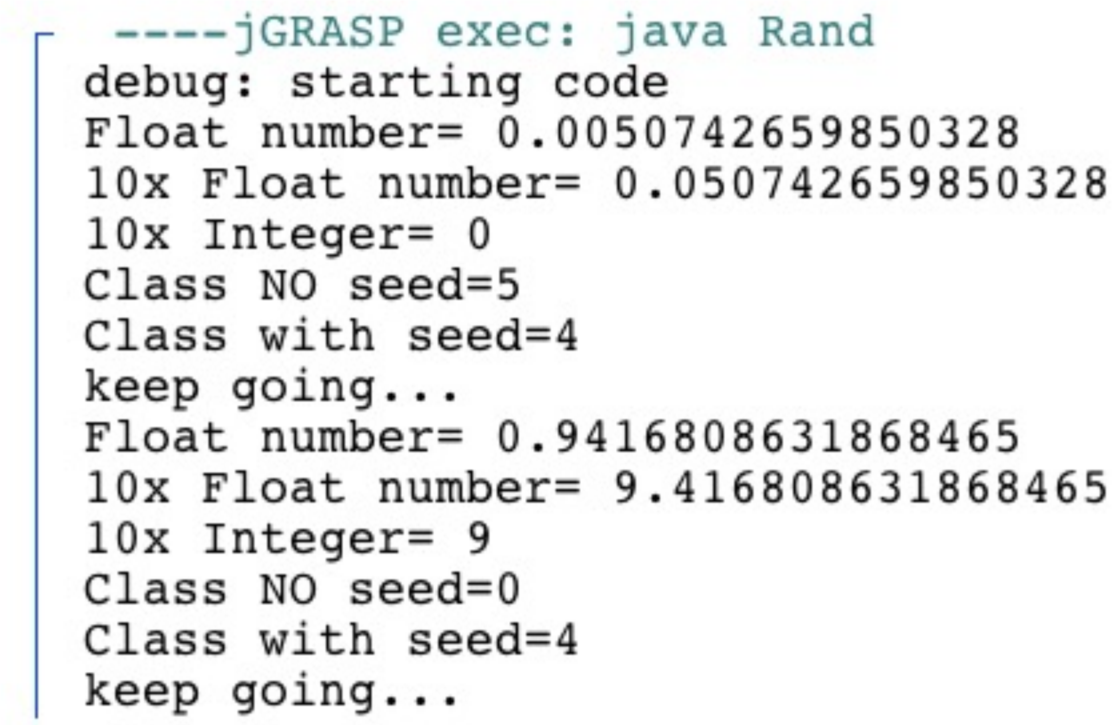


```
   //Random number generation using "random" method
   int N = 10;
   while(run) {
      double randFlt = Math.random();
      System.out.println("Float number= " + randFlt);
      double rand10Flt = N * randFlt;
      System.out.println("10x Float number= " + rand10Flt);
      int rand10 = (int)rand10Flt;
      System.out.println("10x Integer= " + rand10);
   //Random number generation using "Random" Class
      long seed = 3;
      Random Rand1 = new Random();//default seed
      Random Rand2 = new Random(seed);//using seed
      int num1 = Rand1.nextInt(10);
      int num2 = Rand2.nextInt(10);
      System.out.println("Class NO seed=" +num1);
      System.out.println("Class with seed=" +num2);
      int cont = JOptionPane.showConfirmDialog(null, "continue?");
         switch (cont) {
            case 0: System.out.println("keep going...");
               break;
            default: System.out.println("good-bye!");
               run = false; //terminate loop
         }//end switch
   }//end loop
} //end main method
```

random → method

Random → class

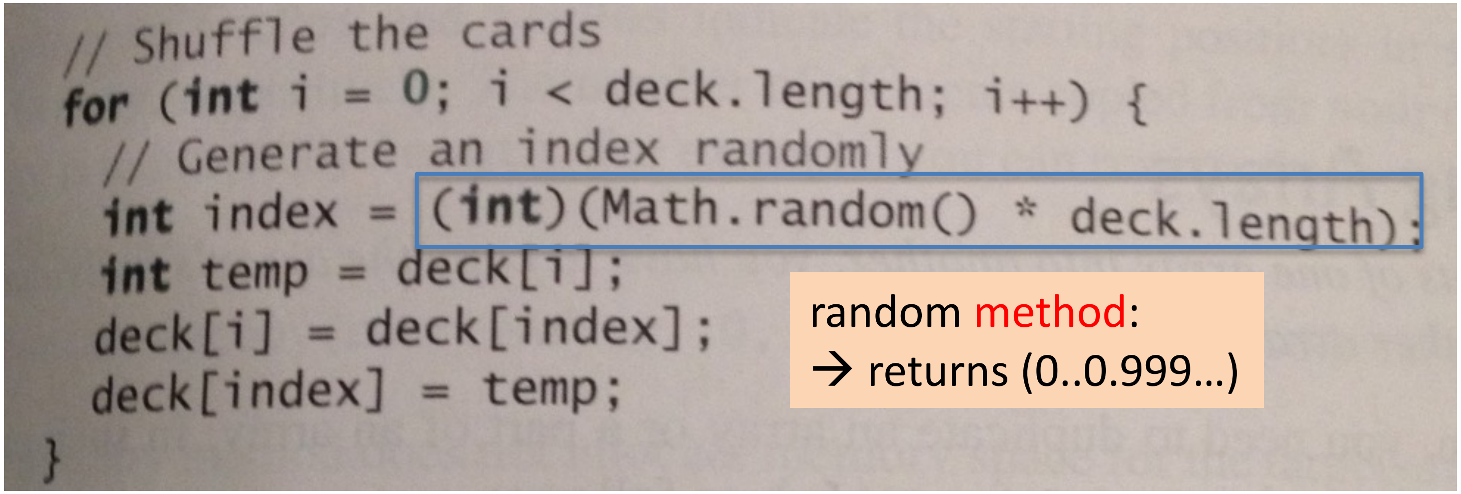
Generating Randoms



```
 ----jGRASP exec: java Rand
debug: starting code
Float number= 0.0050742659850328
10x Float number= 0.050742659850328
10x Integer= 0
Class NO seed=5
Class with seed=4
keep going...
Float number= 0.9416808631868465
10x Float number= 9.416808631868465
10x Integer= 9
Class NO seed=0
Class with seed=4
keep going...
```

Math.random( )

shuffle deck of 52 cards:
→ returns integer = (0..51)

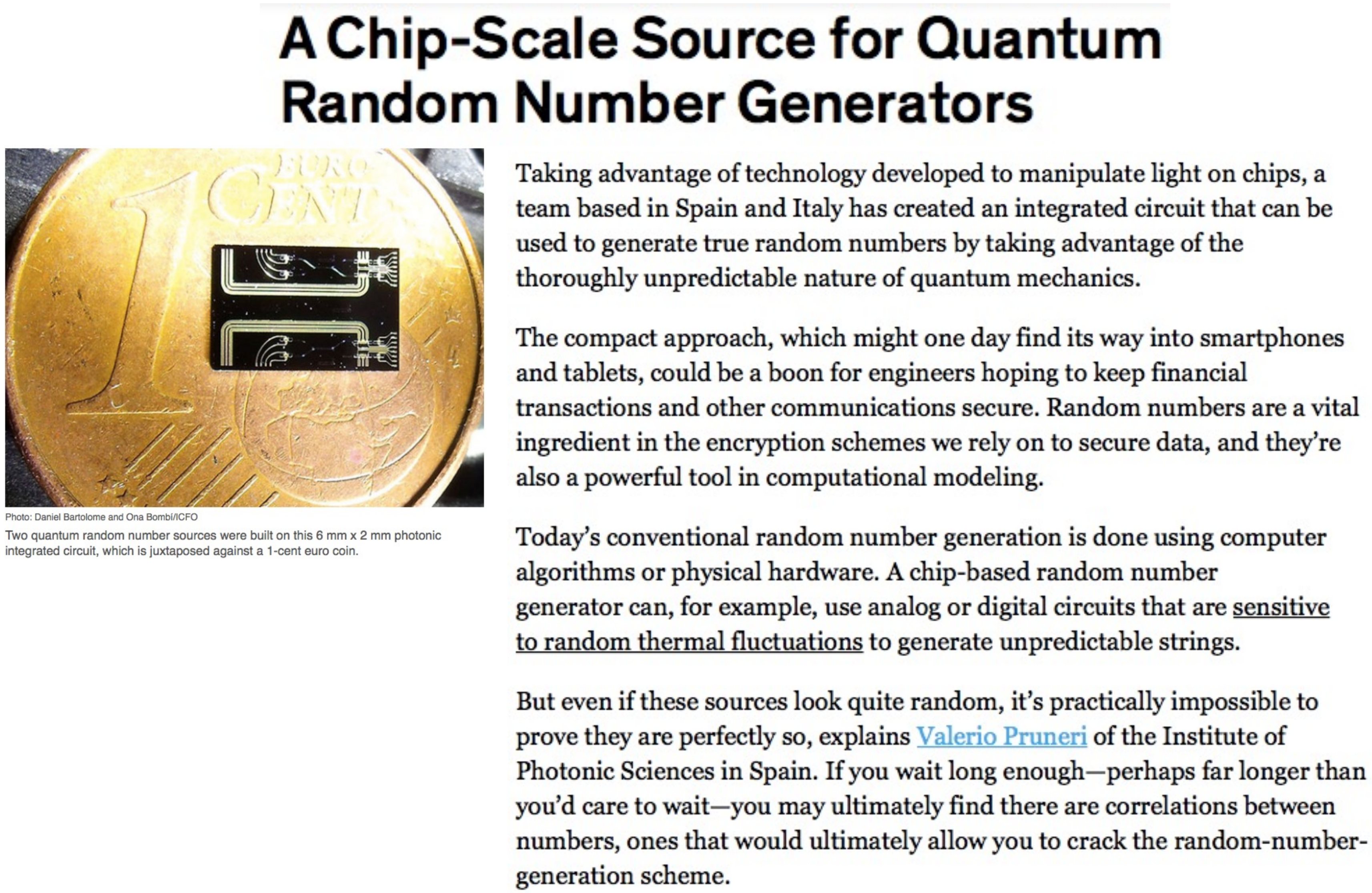


```
// Shuffle the cards
for (int i = 0; i < deck.length; i++) {
  // Generate an index randomly
  int index = (int)(Math.random() * deck.length);
  int temp = deck[i];
  deck[i] = deck[index];
  deck[index] = temp;
}
```

random method:
→ returns (0..0.999…)

Swap “deck[i]” ←→ “deck[random]”

Random Numbers



A Chip-Scale Source for Quantum Random Number Generators

Photo: Daniel Bartolome and Ona Bombi/ICFO

Two quantum random number sources were built on this 6 mm x 2 mm photonic integrated circuit, which is juxtaposed against a 1-cent euro coin.

Taking advantage of technology developed to manipulate light on chips, a team based in Spain and Italy has created an integrated circuit that can be used to generate true random numbers by taking advantage of the thoroughly unpredictable nature of quantum mechanics.

The compact approach, which might one day find its way into smartphones and tablets, could be a boon for engineers hoping to keep financial transactions and other communications secure. Random numbers are a vital ingredient in the encryption schemes we rely on to secure data, and they're also a powerful tool in computational modeling.

Today's conventional random number generation is done using computer algorithms or physical hardware. A chip-based random number generator can, for example, use analog or digital circuits that are sensitive to random thermal fluctuations to generate unpredictable strings.

But even if these sources look quite random, it's practically impossible to prove they are perfectly so, explains Valerio Pruneri of the Institute of Photonic Sciences in Spain. If you wait long enough—perhaps far longer than you'd care to wait—you may ultimately find there are correlations between numbers, ones that would ultimately allow you to crack the random-number-generation scheme.

Math Lib

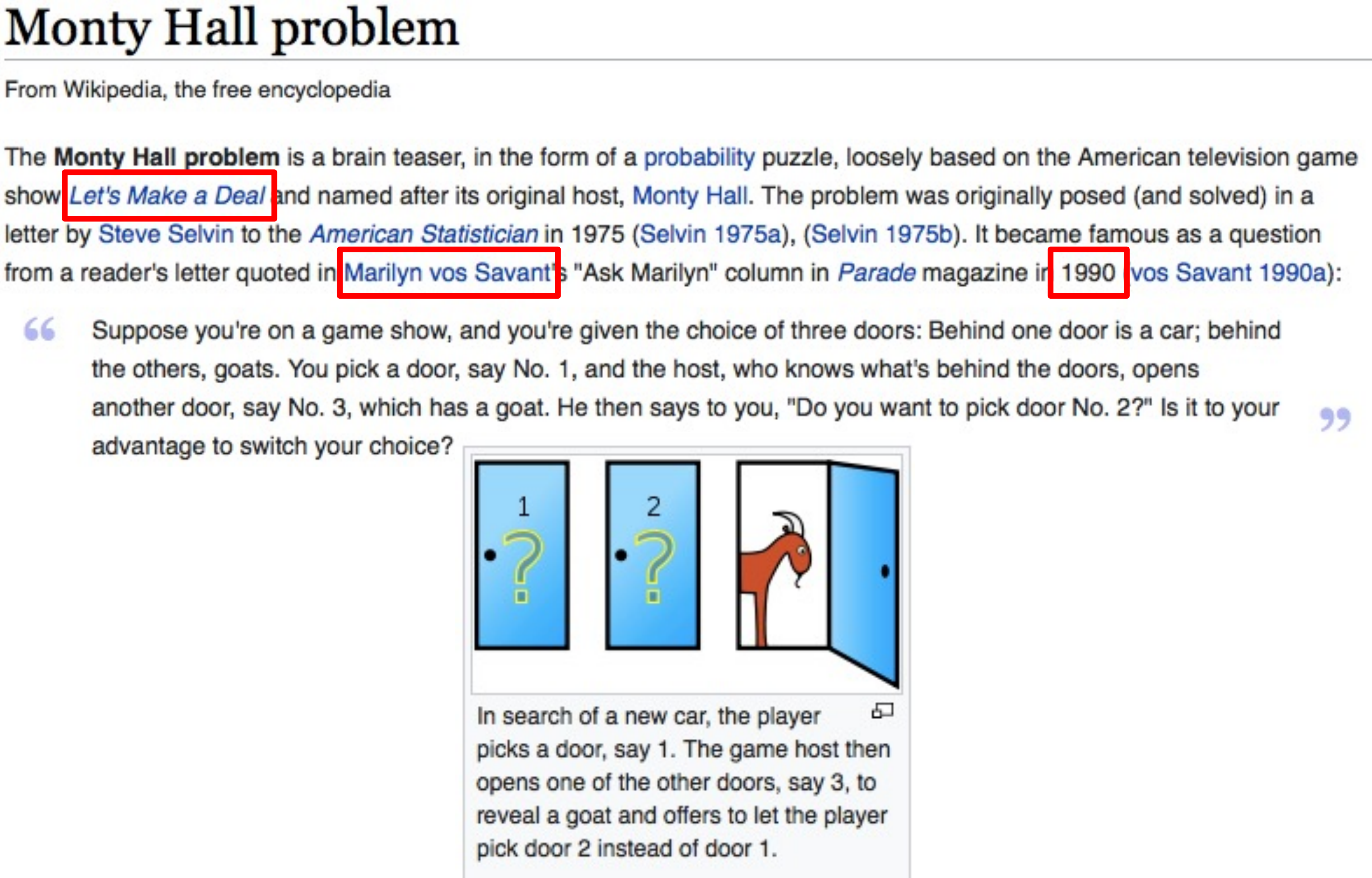
Monty Hall Problem

- ➢ 3 Prisoners Problem

Monty Hall Problem

- Pick a Door

Monty Hall Problem



Monty Hall problem

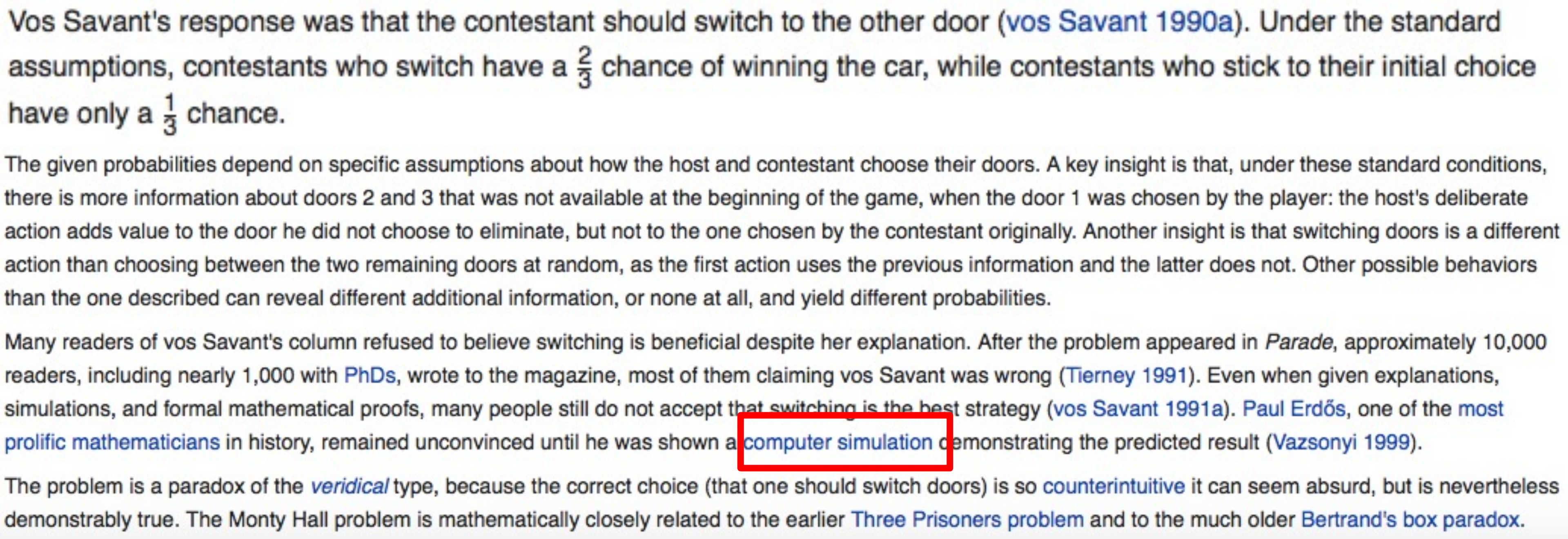
From Wikipedia, the free encyclopedia

The **Monty Hall problem** is a brain teaser, in the form of a probability puzzle, loosely based on the American television game show *Let's Make a Deal* and named after its original host, Monty Hall. The problem was originally posed (and solved) in a letter by Steve Selvin to the *American Statistician* in 1975 (Selvin 1975a), (Selvin 1975b). It became famous as a question from a reader's letter quoted in Marilyn vos Savant's "Ask Marilyn" column in *Parade* magazine in 1990 (vos Savant 1990a):

> Suppose you're on a game show, and you're given the choice of three doors: Behind one door is a car; behind the others, goats. You pick a door, say No. 1, and the host, who knows what's behind the doors, opens another door, say No. 3, which has a goat. He then says to you, "Do you want to pick door No. 2?" Is it to your advantage to switch your choice?

In search of a new car, the player picks a door, say 1. The game host then opens one of the other doors, say 3, to reveal a goat and offers to let the player pick door 2 instead of door 1.

Monty Hall Problem



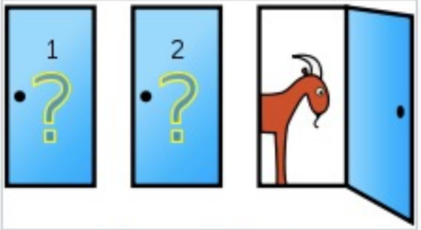
Vos Savant's response was that the contestant should switch to the other door (vos Savant 1990a). Under the standard assumptions, contestants who switch have a $\frac{2}{3}$ chance of winning the car, while contestants who stick to their initial choice have only a $\frac{1}{3}$ chance.

The given probabilities depend on specific assumptions about how the host and contestant choose their doors. A key insight is that, under these standard conditions, there is more information about doors 2 and 3 that was not available at the beginning of the game, when the door 1 was chosen by the player: the host's deliberate action adds value to the door he did not choose to eliminate, but not to the one chosen by the contestant originally. Another insight is that switching doors is a different action than choosing between the two remaining doors at random, as the first action uses the previous information and the latter does not. Other possible behaviors than the one described can reveal different additional information, or none at all, and yield different probabilities.

Many readers of vos Savant's column refused to believe switching is beneficial despite her explanation. After the problem appeared in *Parade*, approximately 10,000 readers, including nearly 1,000 with PhDs, wrote to the magazine, most of them claiming vos Savant was wrong (Tierney 1991). Even when given explanations, simulations, and formal mathematical proofs, many people still do not accept that switching is the best strategy (vos Savant 1991a). Paul Erdős, one of the most prolific mathematicians in history, remained unconvinced until he was shown a computer simulation demonstrating the predicted result (Vazsonyi 1999).

The problem is a paradox of the *veridical* type, because the correct choice (that one should switch doors) is so counterintuitive it can seem absurd, but is nevertheless demonstrably true. The Monty Hall problem is mathematically closely related to the earlier Three Prisoners problem and to the much older Bertrand's box paradox.

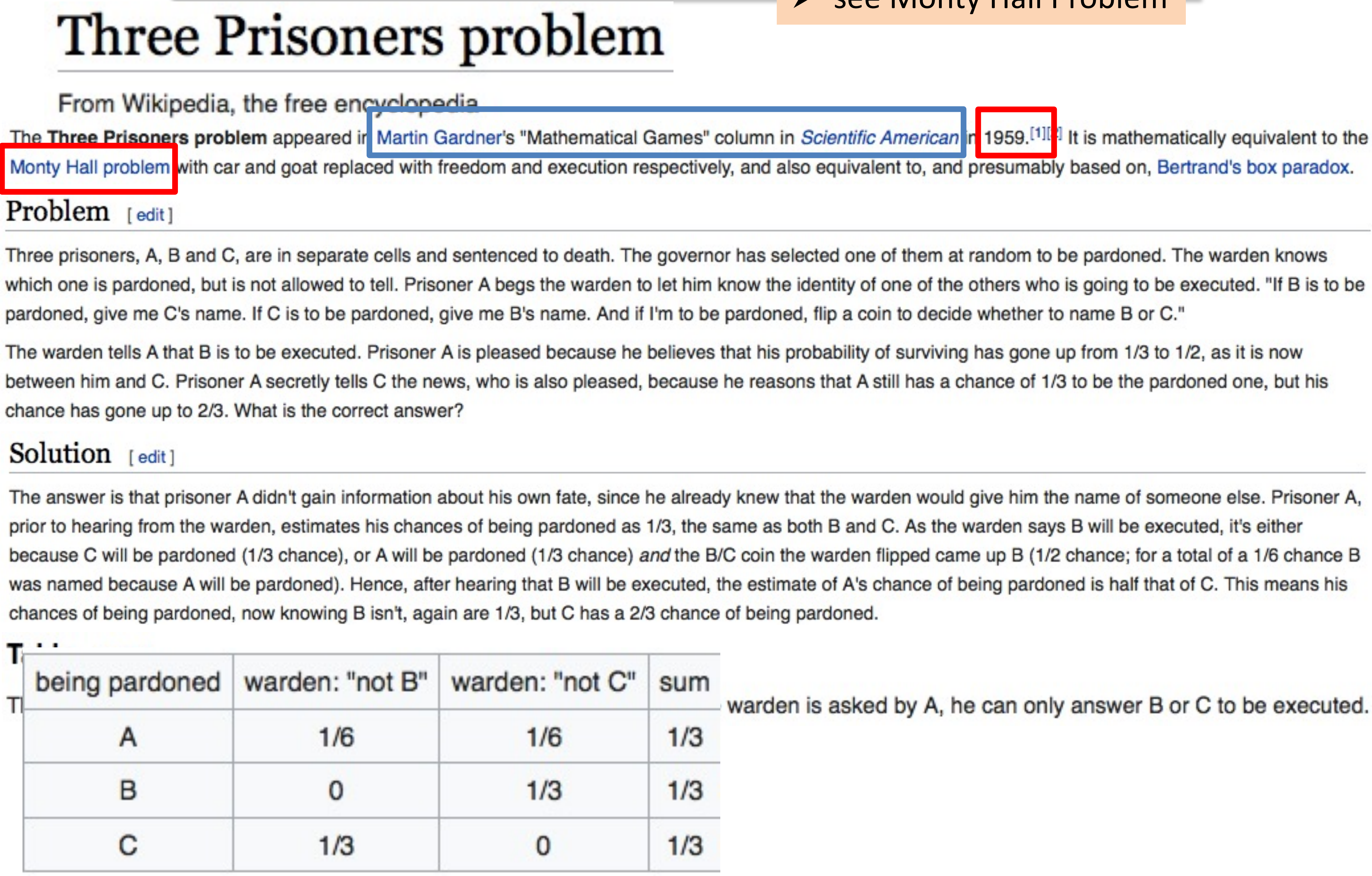
- we will do a computer simulation in *Java*



In search of a new car, the player picks a door, say 1. The game host then opens one of the other doors, say 3, to reveal a goat and offers to let the player pick door 2 instead of door 1.

3 Prisoners Problem

- ➢ see Monty Hall Problem



Three Prisoners problem

From Wikipedia, the free encyclopedia

The **Three Prisoners problem** appeared in Martin Gardner's "Mathematical Games" column in *Scientific American* in 1959.[1][2] It is mathematically equivalent to the Monty Hall problem with car and goat replaced with freedom and execution respectively, and also equivalent to, and presumably based on, Bertrand's box paradox.

Problem [edit]

Three prisoners, A, B and C, are in separate cells and sentenced to death. The governor has selected one of them at random to be pardoned. The warden knows which one is pardoned, but is not allowed to tell. Prisoner A begs the warden to let him know the identity of one of the others who is going to be executed. "If B is to be pardoned, give me C's name. If C is to be pardoned, give me B's name. And if I'm to be pardoned, flip a coin to decide whether to name B or C."

The warden tells A that B is to be executed. Prisoner A is pleased because he believes that his probability of surviving has gone up from 1/3 to 1/2, as it is now between him and C. Prisoner A secretly tells C the news, who is also pleased, because he reasons that A still has a chance of 1/3 to be the pardoned one, but his chance has gone up to 2/3. What is the correct answer?

Solution [edit]

The answer is that prisoner A didn't gain information about his own fate, since he already knew that the warden would give him the name of someone else. Prisoner A, prior to hearing from the warden, estimates his chances of being pardoned as 1/3, the same as both B and C. As the warden says B will be executed, it's either because C will be pardoned (1/3 chance), or A will be pardoned (1/3 chance) *and* the B/C coin the warden flipped came up B (1/2 chance; for a total of a 1/6 chance B was named because A will be pardoned). Hence, after hearing that B will be executed, the estimate of A's chance of being pardoned is half that of C. This means his chances of being pardoned, now knowing B isn't, again are 1/3, but C has a 2/3 chance of being pardoned.

T

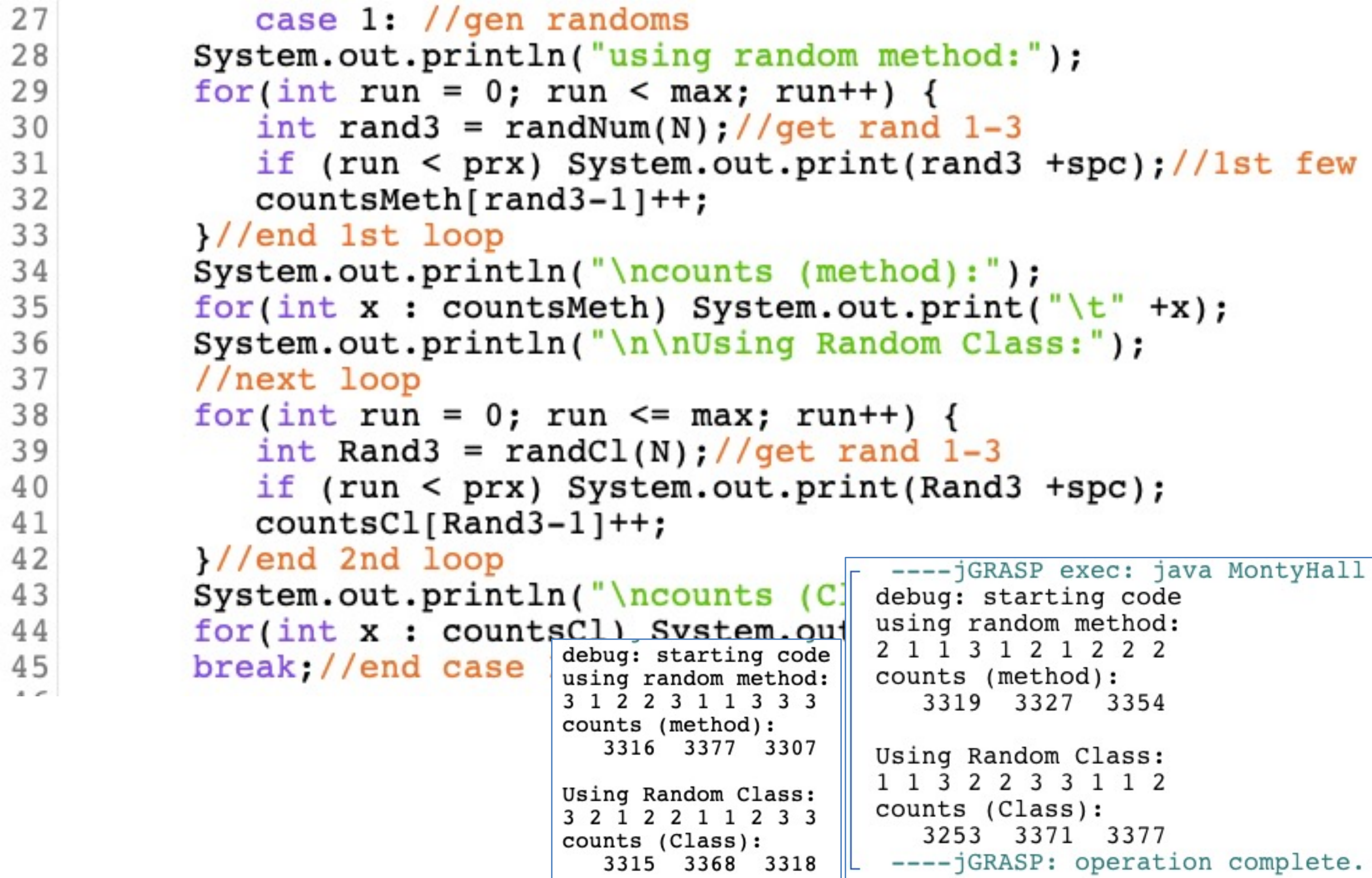
Tl warden is asked by A, he can only answer B or C to be executed.

being pardoned	warden: "not B"	warden: "not C"	sum
A	1/6	1/6	1/3
B	0	1/3	1/3
C	1/3	0	1/3

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP110

Randoms for Monty Hall

DR JEFF SOFTWARE
INDIE APP DEVELOPER

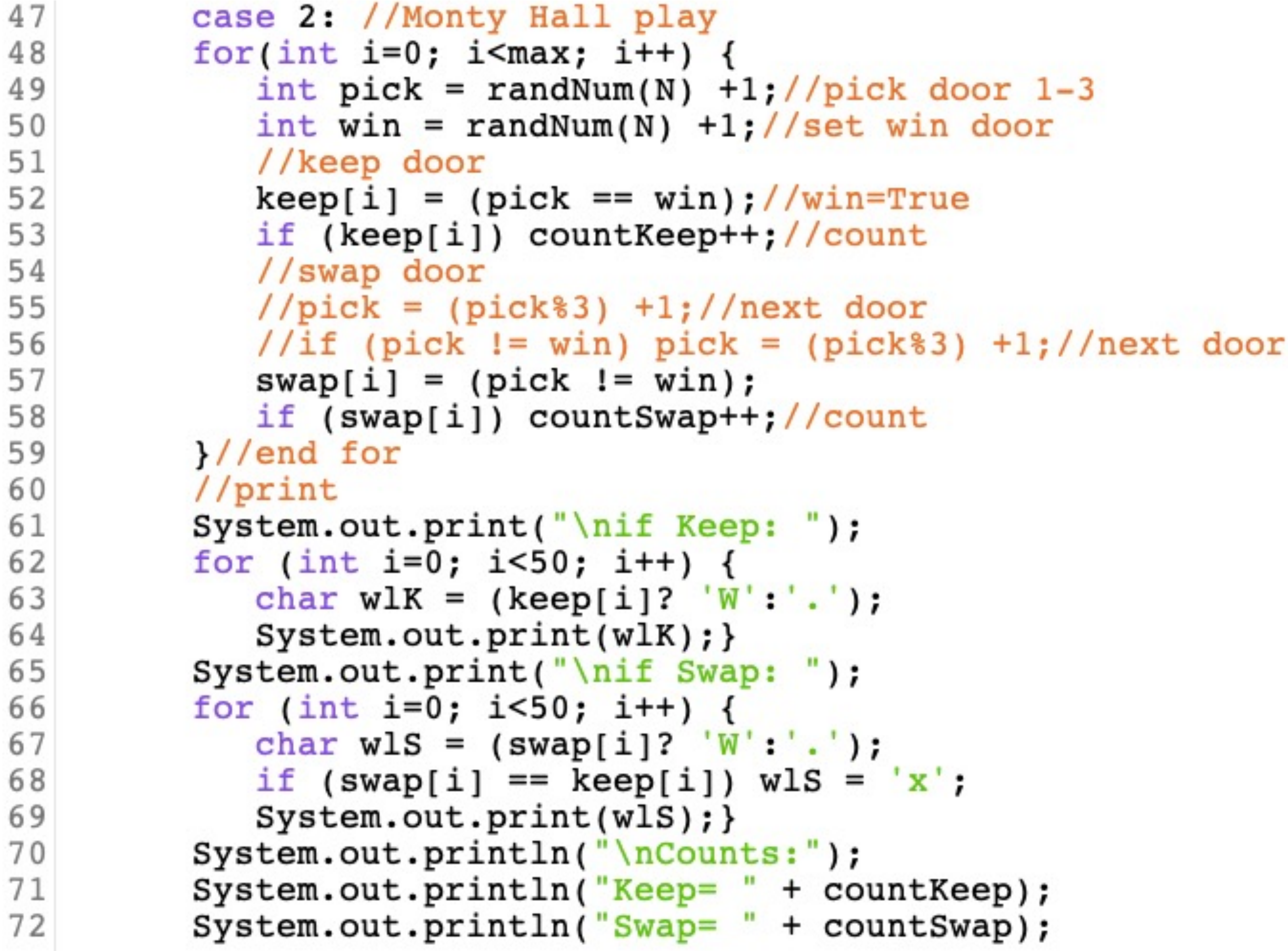


```
    case 1: //gen randoms
System.out.println("using random method:");
for(int run = 0; run < max; run++) {
    int rand3 = randNum(N);//get rand 1-3
    if (run < prx) System.out.print(rand3 +spc);//1st few
    countsMeth[rand3-1]++;
}//end 1st loop
System.out.println("\ncounts (method):");
for(int x : countsMeth) System.out.print("\t" +x);
System.out.println("\n\nUsing Random Class:");
//next loop
for(int run = 0; run <= max; run++) {
    int Rand3 = randCl(N);//get rand 1-3
    if (run < prx) System.out.print(Rand3 +spc);
    countsCl[Rand3-1]++;
}//end 2nd loop
System.out.println("\ncounts (Cl
for(int x : countsCl) System.out
break;//end case
```

```
debug: starting code
using random method:
3 1 2 2 3 1 1 3 3 3
counts (method):
   3316   3377   3307

Using Random Class:
3 2 1 2 2 1 1 2 3 3
counts (Class):
   3315   3368   3318
```

```
 ----jGRASP exec: java MontyHall
debug: starting code
using random method:
2 1 1 3 1 2 1 2 2 2
counts (method):
   3319   3327   3354

Using Random Class:
1 1 3 2 2 3 3 1 1 2
counts (Class):
   3253   3371   3377
 ----jGRASP: operation complete.
```

Monty Hall: Play

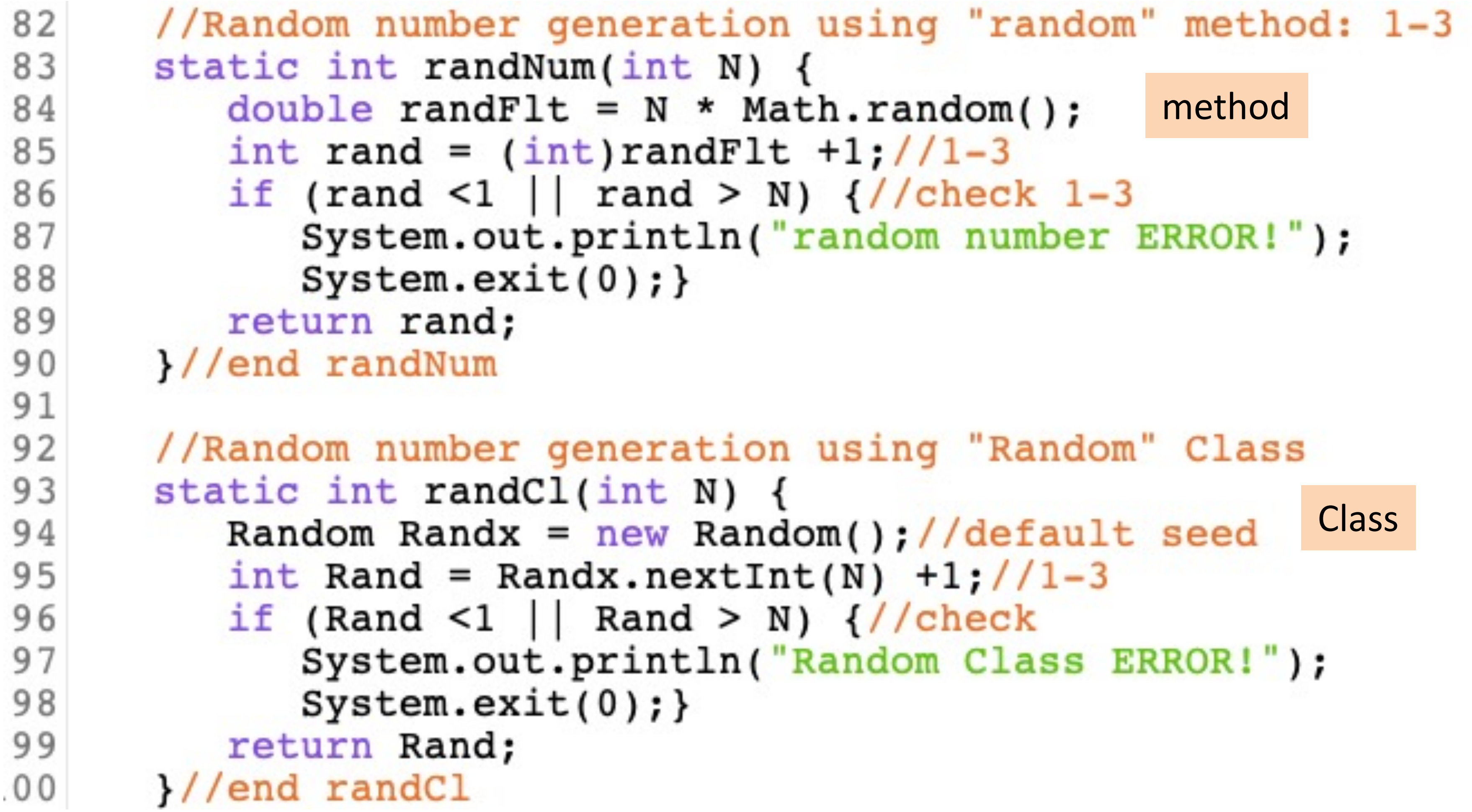


```
case 2: //Monty Hall play
for(int i=0; i<max; i++) {
   int pick = randNum(N) +1;//pick door 1-3
   int win = randNum(N) +1;//set win door
   //keep door
   keep[i] = (pick == win);//win=True
   if (keep[i]) countKeep++;//count
   //swap door
   //pick = (pick%3) +1;//next door
   //if (pick != win) pick = (pick%3) +1;//next door
   swap[i] = (pick != win);
   if (swap[i]) countSwap++;//count
}//end for
//print
System.out.print("\nif Keep: ");
for (int i=0; i<50; i++) {
   char wlK = (keep[i]? 'W':'.');
   System.out.print(wlK);}
System.out.print("\nif Swap: ");
for (int i=0; i<50; i++) {
   char wlS = (swap[i]? 'W':'.');
   if (swap[i] == keep[i]) wlS = 'x';
   System.out.print(wlS);}
System.out.println("\nCounts:");
System.out.println("Keep= " + countKeep);
System.out.println("Swap= " + countSwap);
```

COMP110

Random Methods

©2016-2022
Jeff Drobman

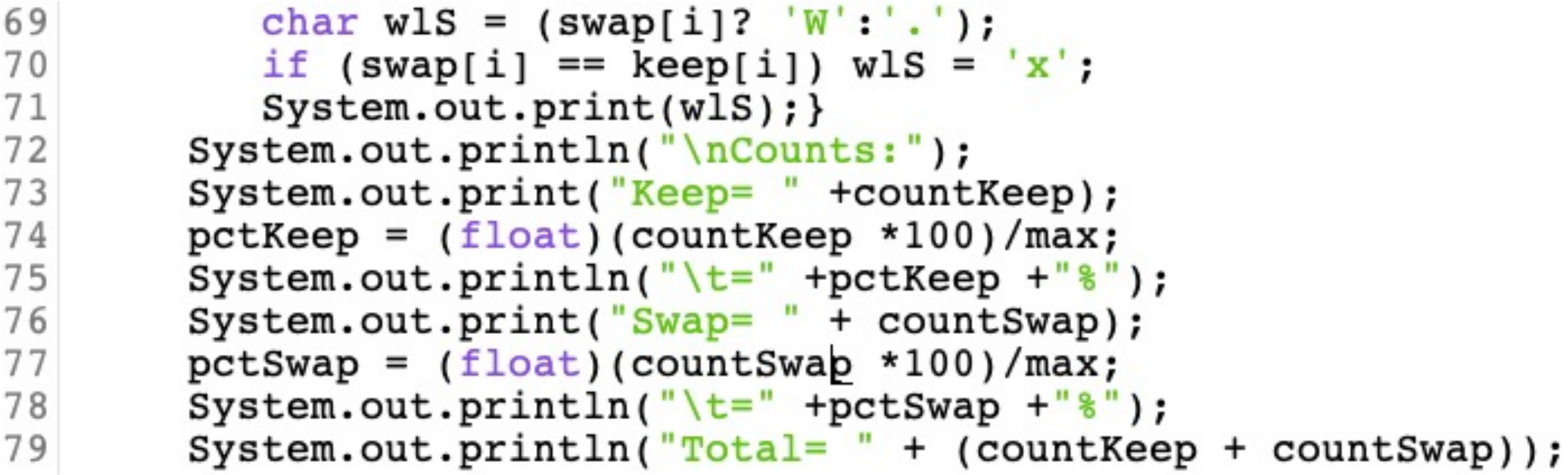


```java
//Random number generation using "random" method: 1-3
static int randNum(int N) {
    double randFlt = N * Math.random();
    int rand = (int)randFlt +1;//1-3
    if (rand <1 || rand > N) {//check 1-3
        System.out.println("random number ERROR!");
        System.exit(0);}
    return rand;
}//end randNum

//Random number generation using "Random" Class
static int randCl(int N) {
    Random Randx = new Random();//default seed
    int Rand = Randx.nextInt(N) +1;//1-3
    if (Rand <1 || Rand > N) {//check
        System.out.println("Random Class ERROR!");
        System.exit(0);}
    return Rand;
}//end randCl
```

method

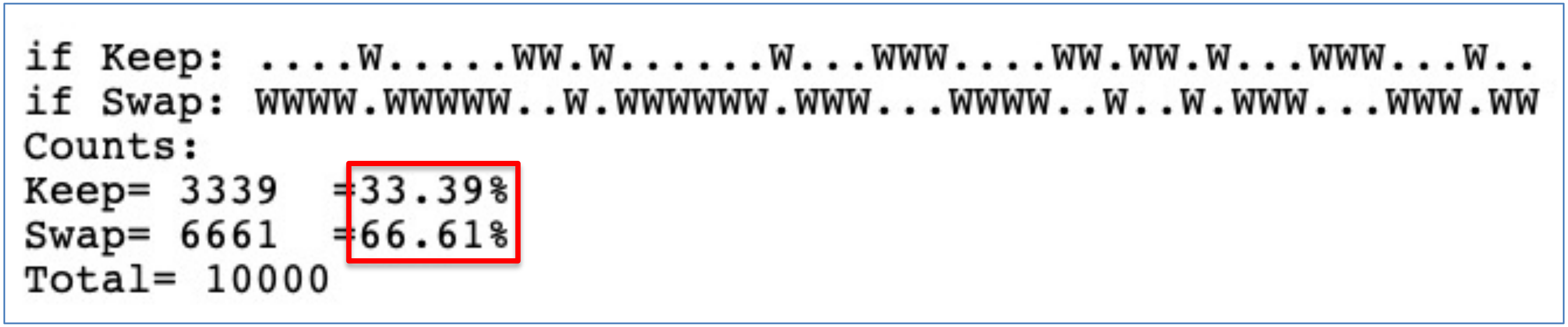
Class

Monty Hall Results



```
            char wlS = (swap[i]? 'W':'.');
            if (swap[i] == keep[i]) wlS = 'x';
            System.out.print(wlS);}
        System.out.println("\nCounts:");
        System.out.print("Keep= " +countKeep);
        pctKeep = (float)(countKeep *100)/max;
        System.out.println("\t=" +pctKeep +"%");
        System.out.print("Swap= " + countSwap);
        pctSwap = (float)(countSwap *100)/max;
        System.out.println("\t=" +pctSwap +"%");
        System.out.println("Total= " + (countKeep + countSwap));
```

Counts + percents (%)



```
if Keep: ....W.....WW.W......W...WWW....WW.WW.W...WWW...W..
if Swap: WWWW.WWWWW..W.WWWWWW.WWW...WWWW..W..W.WWW...WWW.WW
Counts:
Keep= 3339   =33.39%
Swap= 6661   =66.61%
Total= 10000
```