

Java

Intro to Algorithms & Programming

Part 1

Dr Jeff Drobman

Index

- ❖ Models → slide 3
- ❖ **Structure** → slide 10
- ❖ Reference → slide 17
- ❖ Data Codes → slide 21
- ❖ Data Types (Ch 2) → slide 32
- ❖ Type conversion → slide 49
- ❖ Formatted output → slide 54
- ❖ Logic (Ch 3) → slide 57
- ❖ **Control structures** → slide 62
- ❖ Strings → slide 79
- ❖ **Methods** → slide 95
- ❖ Arrays → slide 111
- ❖ Array Methods → slide 133
- ❖ Multi-Dim (2D) Arrays → slide 149
- ❖ Command Line Args → slide 153
- ❖ Date & Time → slide 155
- ❖ Random Numbers → slide 169
- ❖ Monty Hall Problem → slide 177
- ❖ Theory → slide 186

KEY

DATA
STRUCTURE
LOGIC
OTHER

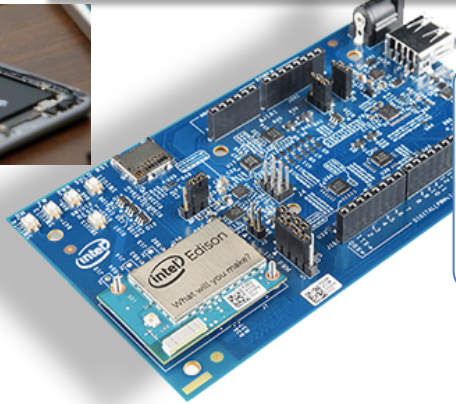
Software

Models

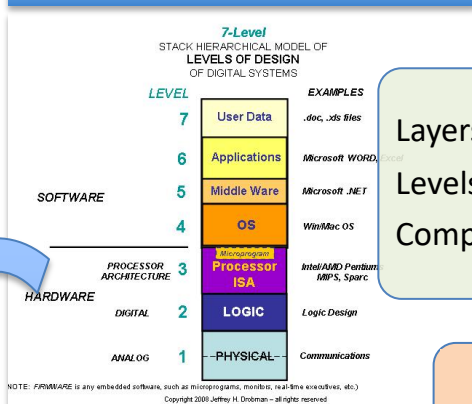
Digital Systems

Hardware

Software



Computer
 Phone
 Controller



Layers
 Levels
 Composition

❖ CODE
 ❖ DATA

Computer
 Engineering

- ❖ Architecture
- ❖ Physical design

External Storage

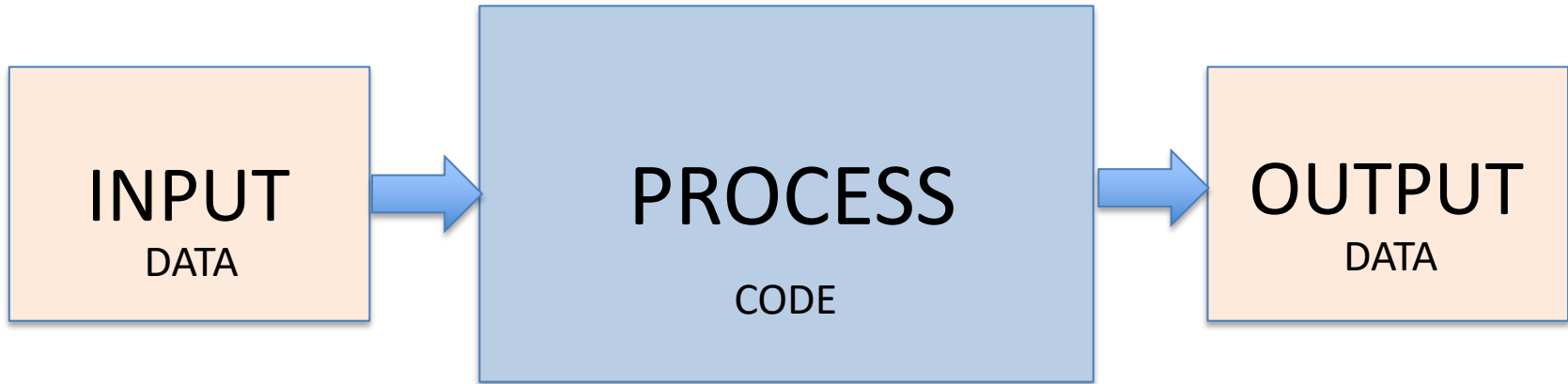


Hard disk
 Flash disk
 Flash
 EEPROM

Computer
 Science

- ❖ Algorithms
- ❖ Theory

IPO Model



Software Development

Development Procedure
(similar to **SDLC**)

ANALYZE PROBLEM

WRITE REQUIREMENTS

CREATE DESIGN

WRITE PROGRAM

ANALYZE RESULTS

Systems Analysis

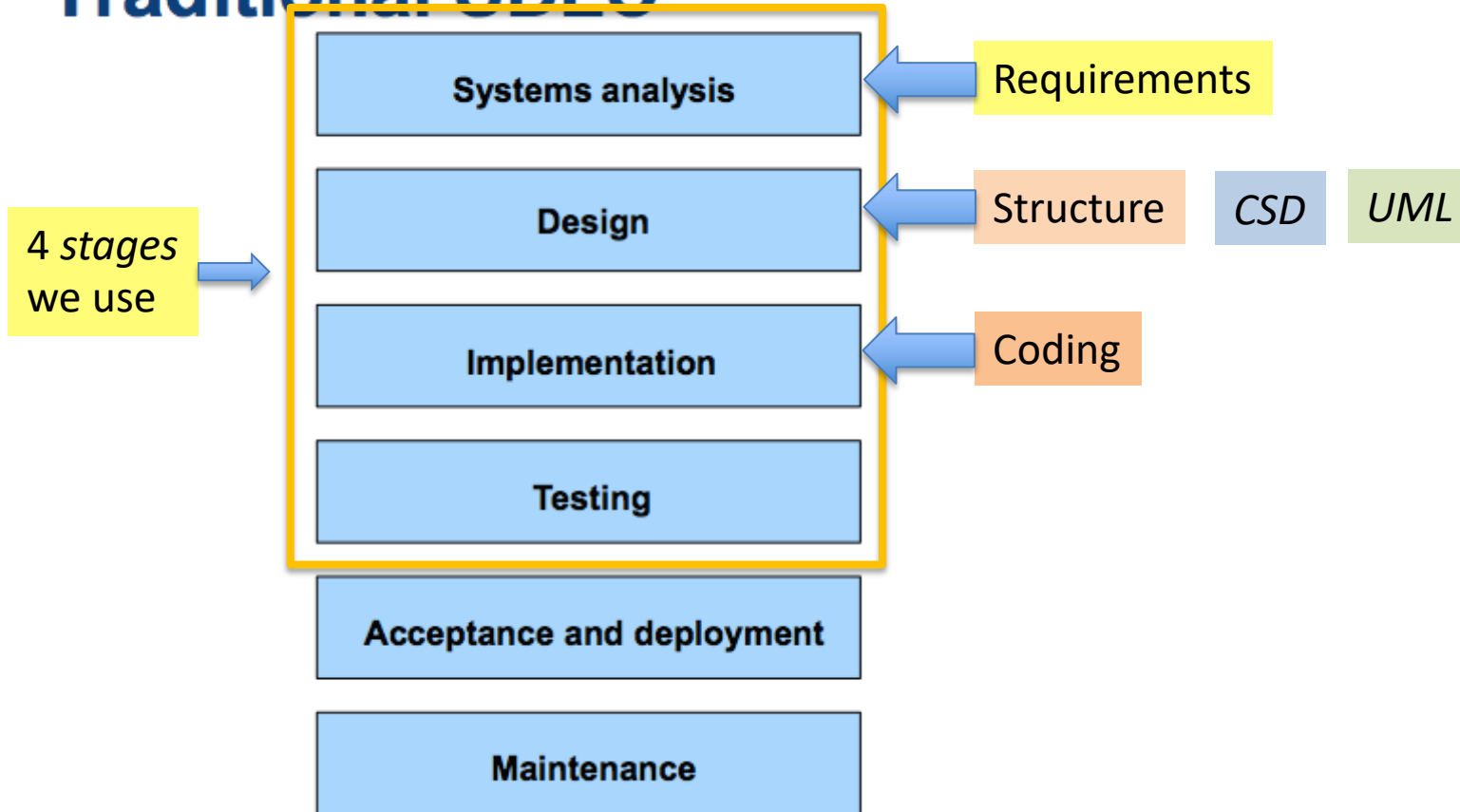
- STRUCTURE
 - CODE
 - DATA
- ALGORITHMS

Language Independent

Select Language

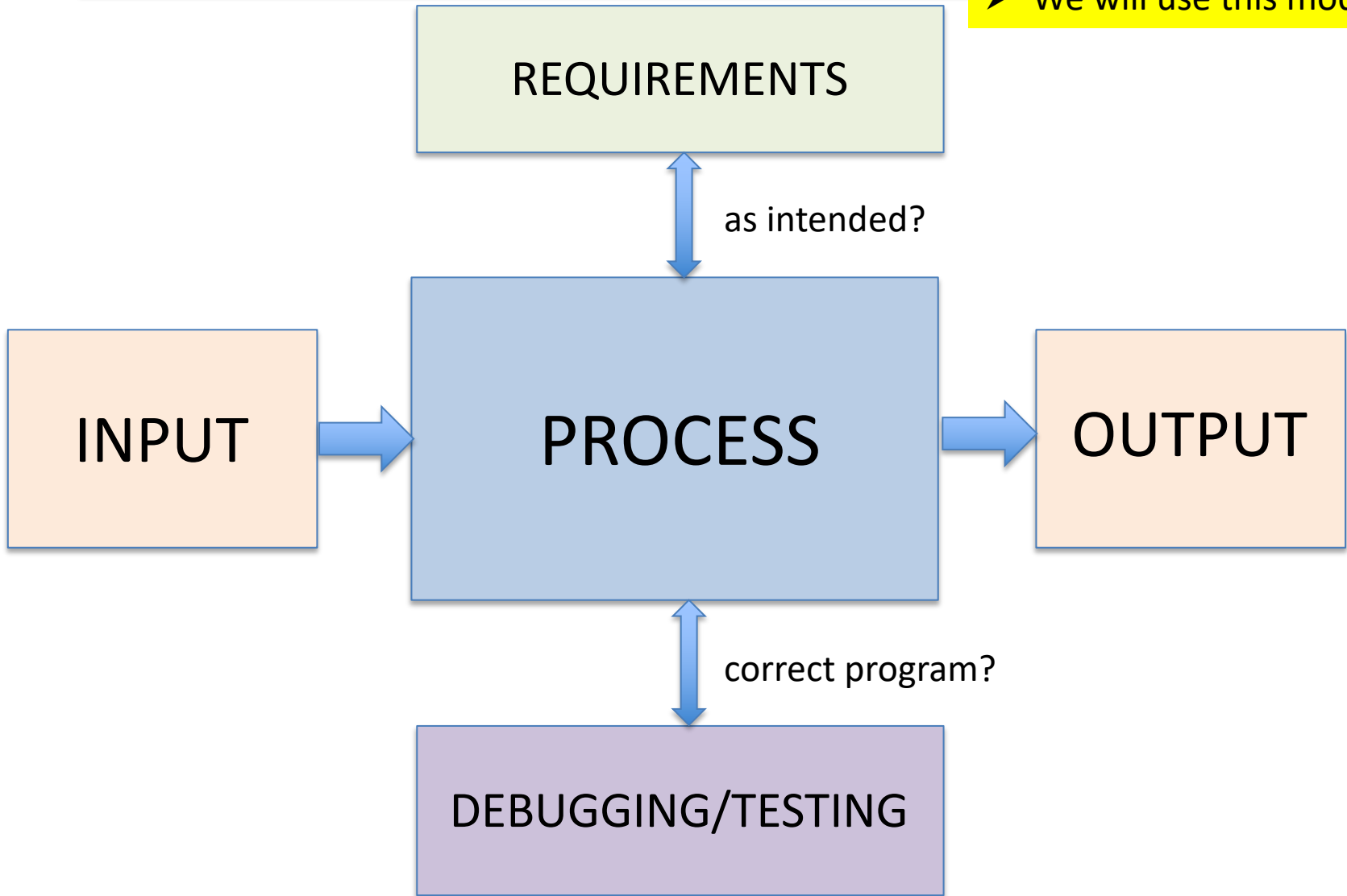
SOFTWARE DEVELOPMENT LIFE CYCLE

Traditional SDLC

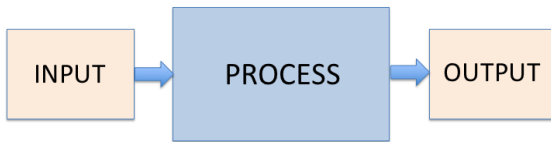


SDLC + IPO

➤ We will use this model



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

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
- 

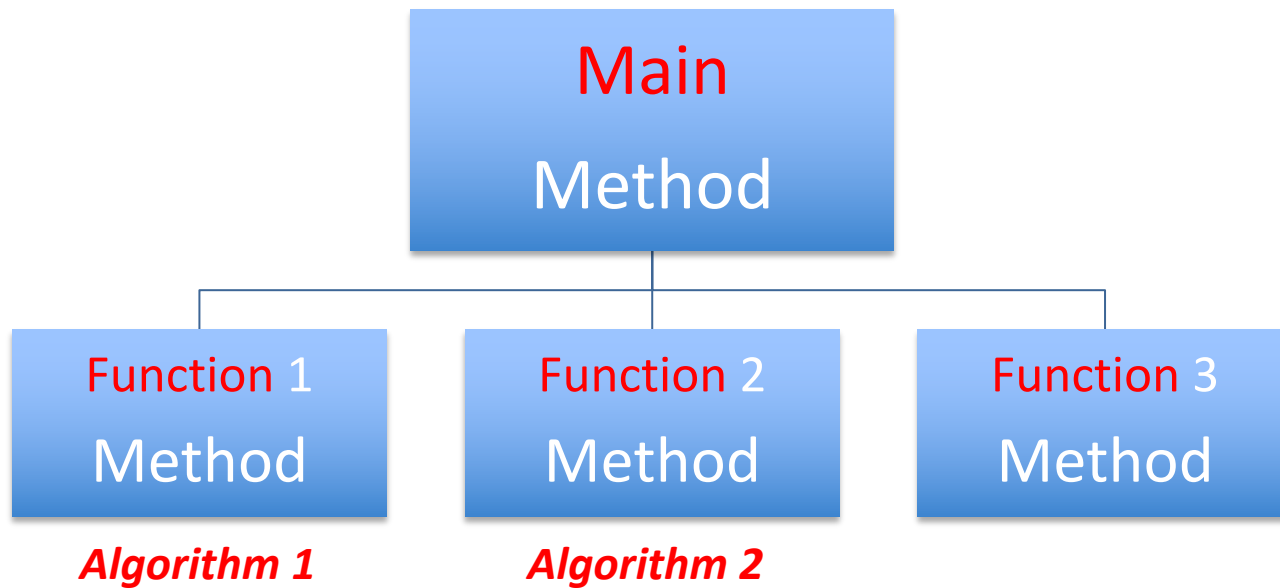
❖ Data types

- ❑ 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)

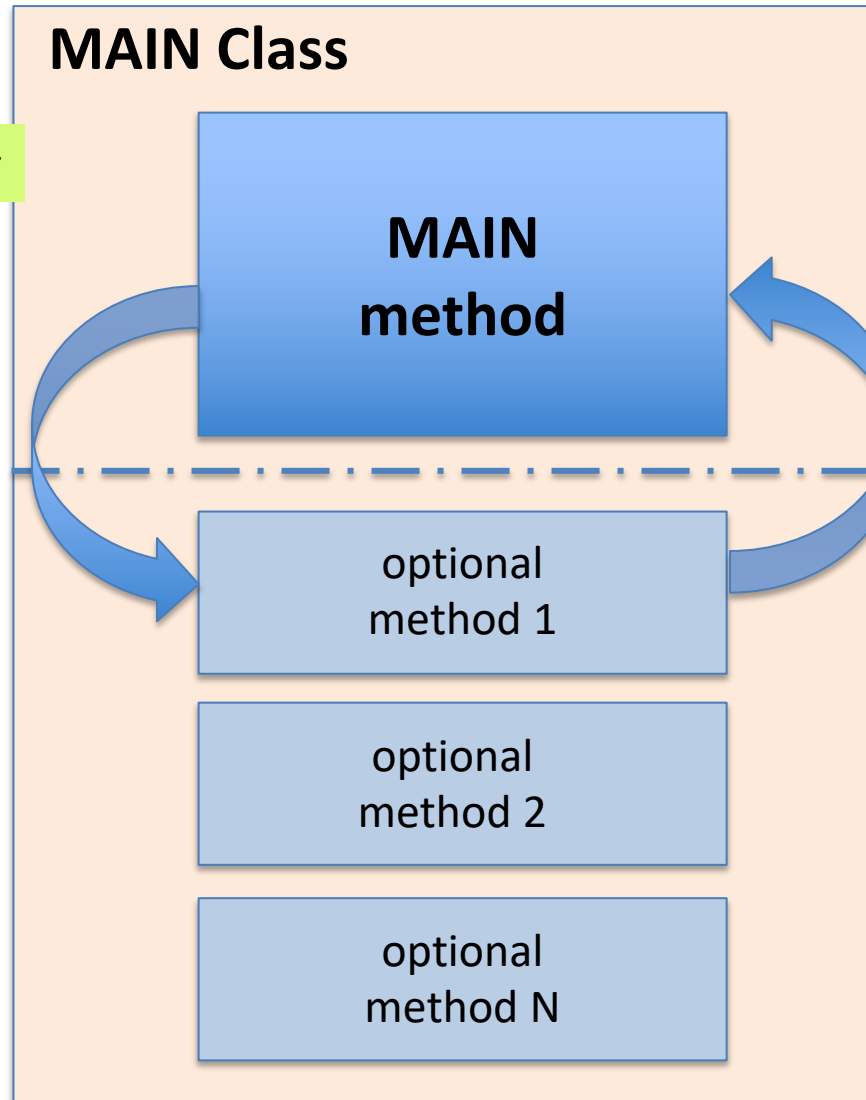
Java ↔ OOP

OOP
Structures

Classes/methods

Execution is
by *call* sequence

Minimum
required



MAIN *Class* = any name

MAIN *Method* = "main"

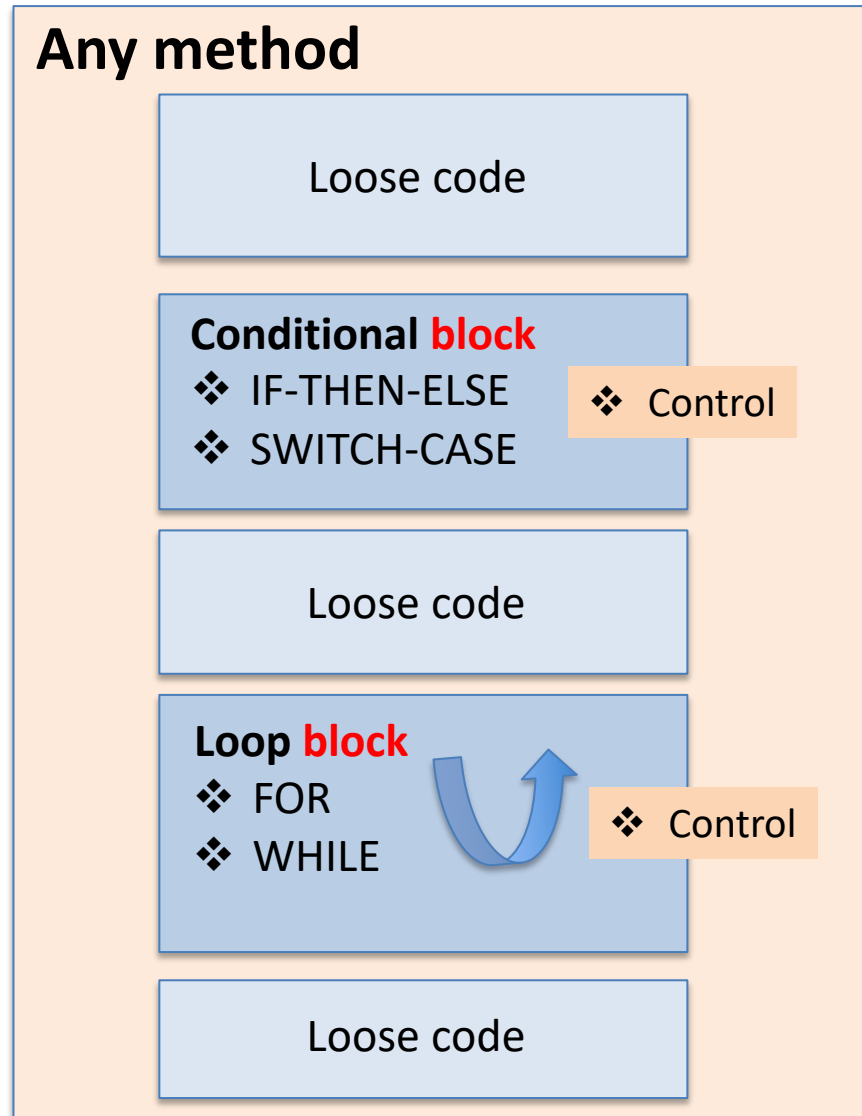
Code Structure (Micro)

Java ↔ OOP

Block Structures

Constructs

Execution is sequential



Programs & Algorithms

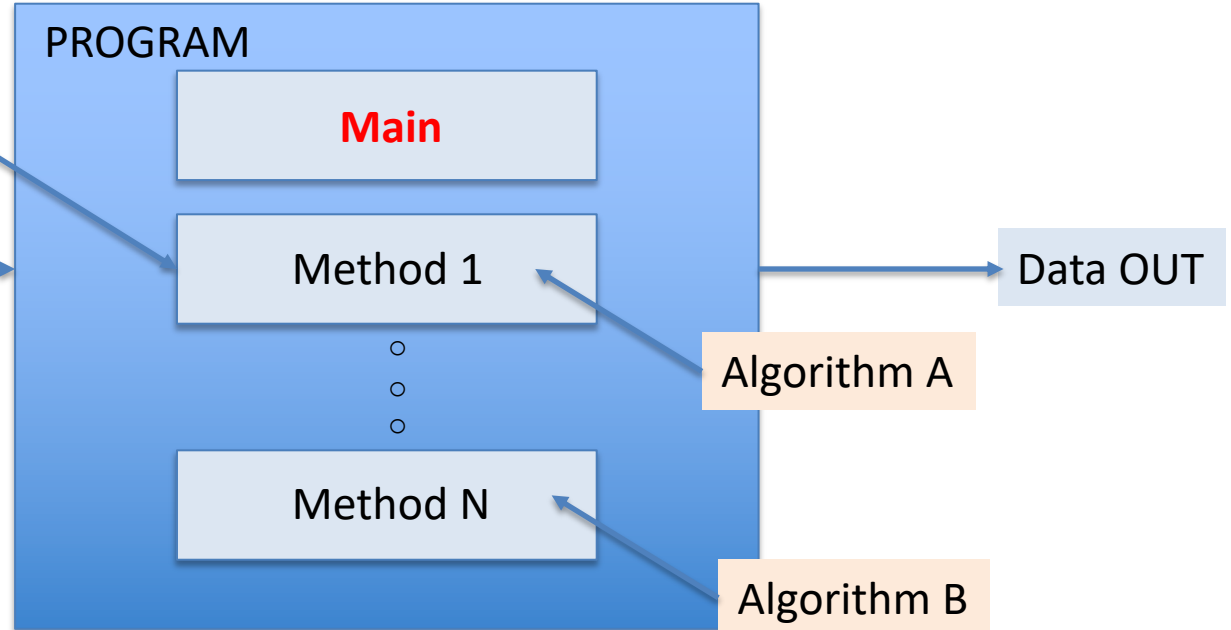
Nested Structure

Requirements

Procedure → Process

➤ Map Fns → Methods

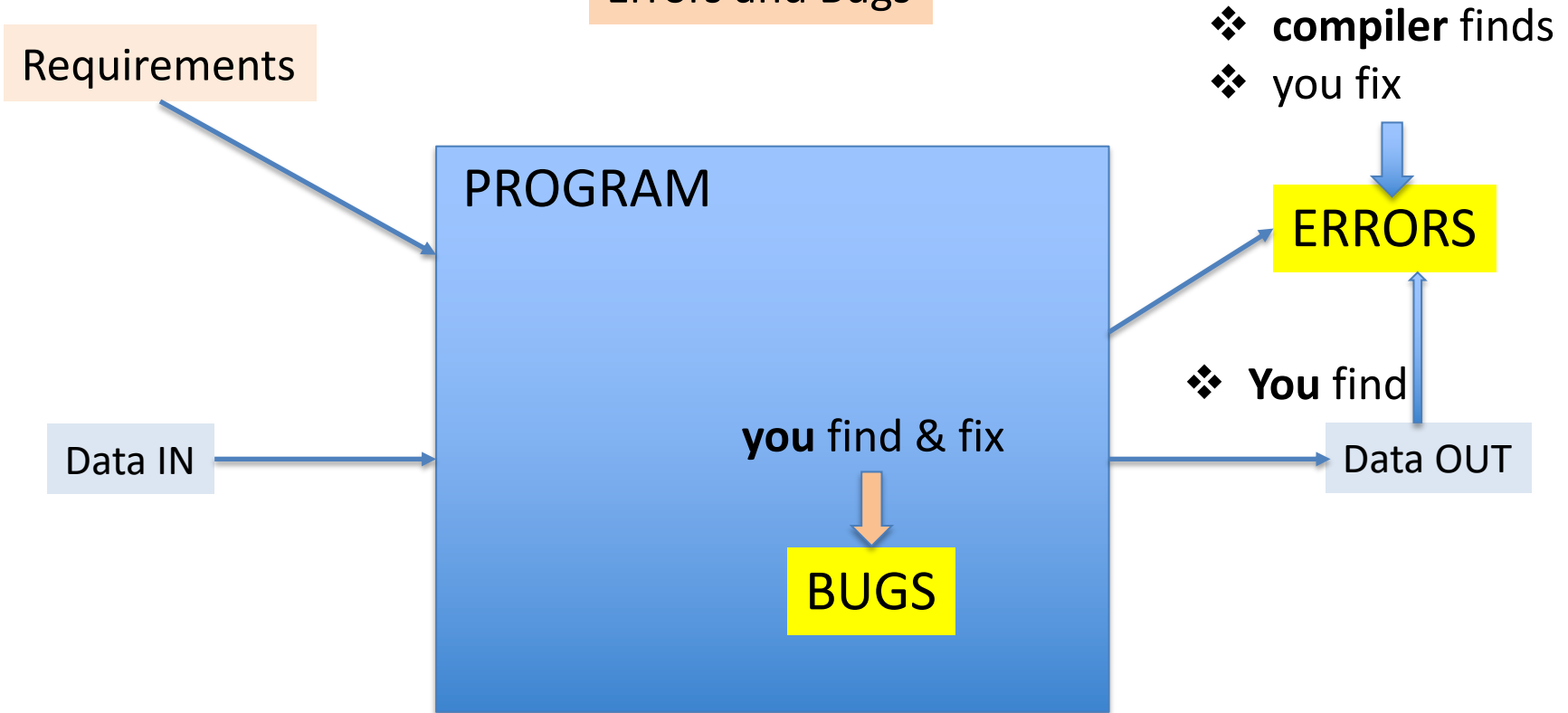
Data IN



➤ 1 Algorithm per *function* or *method*

Program Debugging

Errors and Bugs



❖ ERRORS

- ❑ syntax → compiler
- ❑ semantic/logic → debug
- ❑ exceptions → handlers

❖ BUGS

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

Java

Reference

Meta-language: BNF

❖ BNF = Backus-Naur Form

Backus–Naur Form

From Wikipedia, the free encyclopedia

<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

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;
}
```

Java Ref Pg 2

```
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 = new int[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();
String s1 = 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.

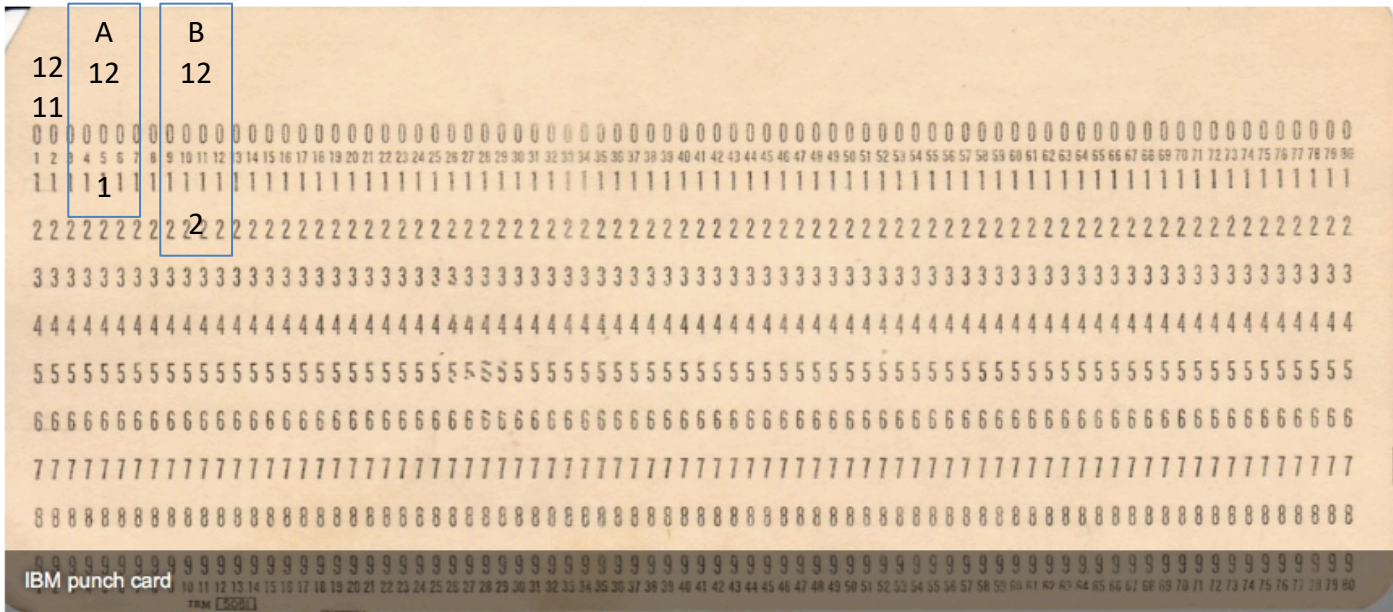
A	• —	U	• • —
B	— • • •	V	• • • —
C	— • — •	W	• — • —
D	— • •	X	— • • —
E	•	Y	— • — • —
F	• • • —	Z	— — • •
G	— — •		
H	• • • •		
I	• •		
J	• — — —		
K	— • — —		
L	• — • •		
M	— —		
N	— •		
O	— — —		
P	• — — •		
Q	— — • —		
R	• — •		
S	• • •		
T	—		
		1	• — — —
		2	• • — — —
		3	• • • — —
		4	• • • •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

					0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
					0	1	2	3	4	5	6	7
b ₄	b ₃	b ₂	b ₁	Row								
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	ENO	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`

IBM EBCDIC

EBCDIC

Extended Binary Coded Decimal

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

ASCII Code			EBCDIC Code		
Character	Decimal	Hex	Character	Decimal	Hex
space	32	20	space	64	40
!	33	21	.	75	4B
"	34	22	<	76	4C
#	35	23	(77	4D
\$	36	24	+	78	4E
%	37	25	!	79	4F
&	38	26	&	80	50
single quote	39	27	\$	91	5B
(40	28	*	92	5C
)	41	29)	93	5D
*	42	2A	;	94	5E
+	43	2B	minus -	96	60
comma	44	2C	/	97	61
-	45	2D	comma	107	6B
.	46	2E	%	108	6C
/	47	2F	>	110	6E
0	48	30	?	111	6F
:	:	:	:	122	7A
9	57	39	#	123	7B
:	58	3A	@	124	7C
:	59	3B	single quote	125	7D
<	60	3C	=	126	7E
=	61	3D	"	127	7F
>	62	3E	a	129	81
?	63	3F	b	130	82
@	64	40	:	:	:
A	65	41	z	169	A9
:	:	:	A	193	C1
Z	90	5A	:	:	:
a	97	61	Z	233	E9
:	:	:	0	240	F0
Z	122	7A	9	249	F9

Old Mac Char Codes

16-bit

Second digit	First 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	•	±	ı	-		
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	¶	∂	Δ	÷		
7	BEL	ETB	'	7	G	W	g	w	á	ó	β	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	'	æ	À			
C	FF	FS	,	<	L	\	l		å	ú	"	ø	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			

unique special chars

┌ 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

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?) ; (カタカナ or 片仮名?)
- ❖ Other foreign languages
 - ❑ alphabets
 - ❑ special characters (vis-à-vis, *oomlaut*)



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:

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	<i>unassigned</i>	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

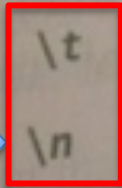
1. If the most significant bit of a byte is zero, then it is a single-byte character, and is completely ASCII-compatible.
2. If the two most significant bits in a byte are set to one, then the byte is the beginning of a multi-byte character.
3. If the most significant bit is set to one, and the second 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

TABLE 2.5 Java Escape Sequences

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

➤ BACKslash



Chapter 2

LIANG

- 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

$$N = \sum_i (n_i * R^i)$$

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

Notes:

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

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

0)	0	8)	1000
1)	1	9)	1001
2)	10	10)	1010
3)	11	11)	1011
4)	100	12)	1100
5)	101	13)	1101
6)	110	14)	1110
7)	111	15)	1111

hexadecimal

A
B
C
D
E
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

1.234512345e+4

Pi = 3.14159e0

- ❖ Java **rounds** floats

Accuracy vs. Precision

no. digits correct

no. digits stored/displayed

Ordinals

Powers of 2 <> 10: 10:3

Technical ordinals

$10^{(-24)}$	yocto
$10^{(-21)}$	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*

***standard variables

Dim i, j, k, l, m, n **As Byte** In VB

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

❖ Integers (16-bit)

C

- 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: $\pm N \times 10^{64}$)
- Double (64-bit double precision: $\pm N \times 10^{256}$)

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

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

FF	-128 to +127
FFFF	-32,768 to +32,767
FFFF FFFF	-2,147,483,648 to +2.15B
FFFF FFFF FFFF FFFF	-1.845x10 ¹⁹ to +X(-1)
FFFF FFFF	0 to +3.4x10 ³⁸
FFFF FFFF FFFF FFFF	0 to +1.8x10 ³⁰⁸
FFFF	0 to 65,535
FF	0 or -1 {00, FF}

❖ all signed

❖ unsigned

Data Types– Assembly Level

- ❖ Hex: **0x**5EC46A or 5EC46A**H** 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)

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

- ✧ 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 [2¹⁶-] MMMM
- ✧ Sign extension SSSSSSSS [2^N-] MMMM

Floating Point

ALL REAL NUMBERS

Java

Integers:

Div by 0 → ArithmeticException

1.123456 x 10ⁿⁿⁿ

↑
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

IEEE 754

Positive & Negative

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

Floating Point Math Functions

Transcendentals as Intrinsic

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 → 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

Literals – Numeric, String

Java


□ Decimal

- `int i = 0;`
- `long k = 123456789`
- `double xx = 3.141592653`
- `float x = 1.23456e-21`

□ Hexadecimal

- `int hx = 0x1F8C`
- `long hxyz = 0x1F8C5D1B`

□ 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

$1 / 2 \rightarrow 0$

Operations – Shorthand

Java

Symbol	Operation	Example	Result
+=	addition	<code>x += 8</code>	$x \leftarrow x + 8$
-=	subtraction	<code>x -= 8</code>	$x \leftarrow x - 8$
*=	multiplication	<code>x *= 8</code>	$x \leftarrow x * 8$
/=	division	<code>x /= 8</code>	$x \leftarrow x / 8$
%=	remainder (mod residue)	<code>x %= 8</code>	$x \leftarrow x \% 8$

every programming language uses these

Operations – Incr/Decr

Java

Symbol	Operation	Example	Result
++var	PRE incr	$y = (++x) + 2$	$x += 1$ $y \leftarrow x + 2$
var++	POST incr	$y = (x++) + 2$	$y \leftarrow x + 2$ $x += 1$
--var	PRE decr	$y = (--x) + 2$	$x -= 1$ $y \leftarrow x + 2$
var--	POST decr	$y = (x--) + 2$	$y \leftarrow x + 2$ $x -= 1$

most programming languages uses these

Type Conversion

Type Conversions

Numeric

❖ Java *truncates* Integers

- to *Round* **add 0.5**

$$5/9 \rightarrow 0$$

$$5/9 + 0.5 \rightarrow 1.0555 \rightarrow 1$$

❖ Expressions

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

$$5.0/9 \rightarrow 0.5555$$

$$5/9 \rightarrow 0$$

❖ 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)

Operation Precedence

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)	arithmetic
	+ , - (Binary addition and subtraction)	
	< , <= , > , >= (Comparison)	comparison
	== , != (Equality)	
	^ (Exclusive OR)	logic
	&& (AND)	
	 (OR)	
	= , += , -= , *= , /= , %= (Assignment operator)	

The 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

$$a - b + c - d \quad \underline{\text{equivalent}} \quad ((a - b) + c) - d$$

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

$$a = b += c = 5 \quad \underline{\text{equivalent}} \quad a = (b += (c = 5))$$

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

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

```
1 boolean trueOrFalse(boolean b) {  
2     if (b == true) {  
3         return true;  
4     } else if (b == false) {  
5         return false;  
6     }  
7     return false ;  
8 }
```

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

Typical usage:

```
1 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

AND

X	Y	$X \ \&\& \ Y$
0	0	0
0	1	0
1	0	0
1	1	1

XOR

	X	Y	$X \ \wedge \ Y$
pass	0	0	0
	0	1	1
flip	1	0	1
	1	1	0

↑ control
↑ data

INclusive

- ❖ 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

EXclusive

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

*2ⁿ

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*

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>  
}
```

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

Java

if  then 

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

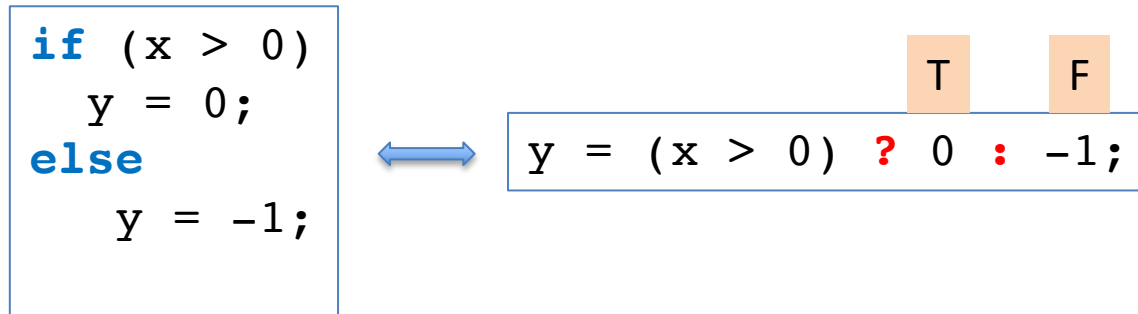
Example: *IF-THEN-ELSE*

```
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*



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*

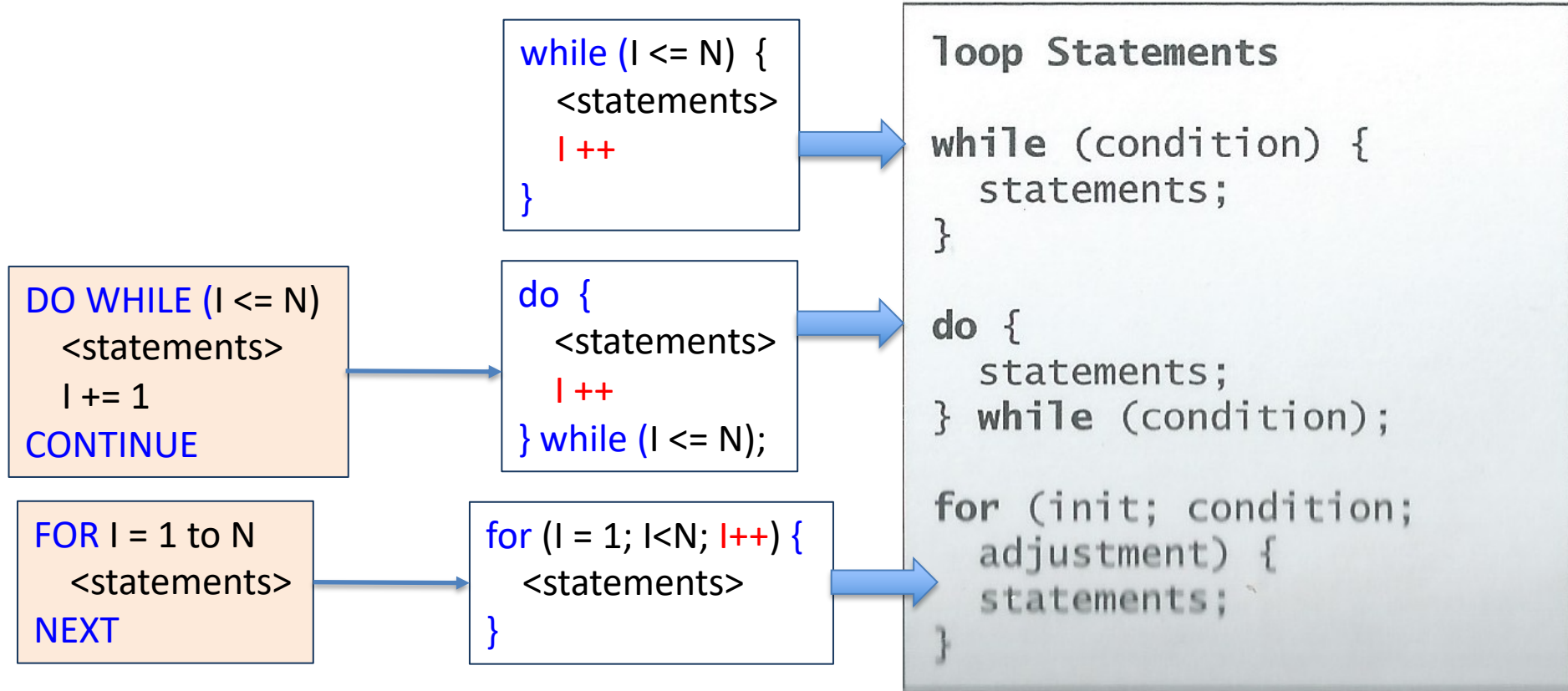
```
String name = "Joe";  
switch (name) {  
    case "Joe": //Joe  
        str = "do this";  
        break;  
    case "Mary": //Mary  
        str = "do something else";  
        break;  
    default: //Bob or other  
        str = "Bob, do Joe's job";  
}  
System.out.println (nam + ", " + str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

❖ Switch can also be a String or Object

Chapter 4 (5)

- Loops
 - For
 - While
 - Do-While

Code Blocks – *Loops*



VB

C

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;  
}
```

❖ test **FIRST**

❖ break → EXIT early

```
do {  
    statements;  
} while (condition);
```

❖ test **LAST**

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

❖ test **FIRST**

Loops: *Continue*

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

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

not break

- ❖ break
- ❖ continue

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

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

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

not continue

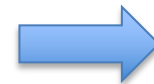
Loops: *While, Do*

```
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);
```

Type-Classes

Strings

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

<i>Method</i>	<i>Description</i>
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	<code>isLetter('x') // true</code> <code>isLetter('6') // false</code> <code>isLetter('!') // false</code>	toUpperCase(c)	Uppercase version	<code>toUpperCase('a') // A</code> <code>toUpperCase('A') // A</code> <code>toUpperCase('3') // 3</code>
isDigit(c)	true if digit: 0-9.	<code>isDigit('x') // false</code> <code>isDigit('6') // true</code>	toLowerCase(c)	Lowercase version	<code>toLowerCase('A') // a</code> <code>toLowerCase('a') // a</code> <code>toLowerCase('3') // 3</code>
isWhitespace(c)	true if whitespace.	<code>isWhitespace(' ') // true</code> <code>isWhitespace('\n') // true</code> <code>isWhitespace('x') // false</code>			

Char Methods Example

zyBook

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

```
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>)
```

convert to int

```
int intValue = Integer.parseInt(string);
```

```
double doubleValue =  
Double.parseDouble(string);
```

convert to double

name of String



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);
```

equalsIgnoreCase

```
String s1 = s.trim();  
String s1 = s.toUpperCase();  
String s1 = s.toLowerCase();
```

```
int index = s.indexOf(ch);  
int index = s.lastIndexOf(ch);
```

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

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

ch ::= <beginIndex>

i,j ::= <beginIndex, endIndex>

```
char[] chs = s.toCharArray();  
String s1 = s.replaceAll(regex, repl);  
String[] tokens = s.split(regex);
```

regex

String Compare

Liang

TABLE 4.8 Comparison Methods for String Objects

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

<div style="border: 1px solid black; padding: 2px; display: inline-block;">concat</div>	<p>concat(moreString) creates a new String that appends the String moreString at the end.</p>	<pre>// userText is "Hi" userText = userText.concat(" friend"); // Now "Hi friend" newText = userText.concat(" there"); // newText is "Hi friend there"</pre>
<p>replace()</p>	<p>replace(findStr, replaceStr) returns a new String in which all occurrences of findStr have been replaced with replaceStr.</p> <p>replace(findChar, replaceChar) returns a new String in which all occurrences of findChar have been replaced with replaceChar.</p>	<pre>// 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"</pre>
<div style="border: 1px solid black; padding: 2px; display: inline-block;">str1 + str2</div>	<p>Returns a new String that is a copy of str1 with str2 appended.</p> <p>str1 may be a String variable or string literal. Likewise for str2. One of str1 or str2 (not both) may be a character.</p>	<pre>// 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"</pre> <div style="background-color: yellow; padding: 5px; display: inline-block; margin-top: 10px;">Concatenation ("Cat")</div>
<p>str1 += str2</p>	<p>Shorthand for <code>str1 = str1 + str2</code>.</p> <p>str1 must be a String variable, and str2 may be a String variable, a string literal, or a character.</p>	<pre>// userText is "My name is " userText += "Tom"; // Now "My name is Tom"</pre>

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 ^{0|1}**zero or one** occurrences of the preceding element. For example, `colou?r` matches both "color" and "colour".
- `*` The asterisk indicates ^{0..N}**zero or more** occurrences of the preceding element. For example, `ab*c` matches "ac", "abc", "abbc", "abbbc", and so on.
- `+` The plus sign indicates ^{1..N}**one or more** occurrences of the preceding element. For example, `ab+c` matches "abc", "abbc", "abbbc", and so on, but not "ac".
- `{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 **+**, **-**, **x**, 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 \)

\d digit **\D** NON digit

\w *word* char **\W** NON \Rightarrow ([a-z] | [A-Z] | [0-9] | _)

\s space **\S** NON space (actually "whitespace" to include others)

[a-z] \Leftrightarrow \w && \D && [^_]

ix = str.indexOf(sp);  find spaces

where

char sp = ' ';

char sp = \u0020;

Search Alpha Via Regex

```
String alpha = "abcdefghijklmnopqrstuvwxyz";
```

```
if (alpha.indexOf(chx.toLowerCase))
```



```
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 = "YNNYYNYNN"

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

Chapter 6

Ch 6

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

Ch 6

```
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

Ch 6

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

Ch 6

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) {
```

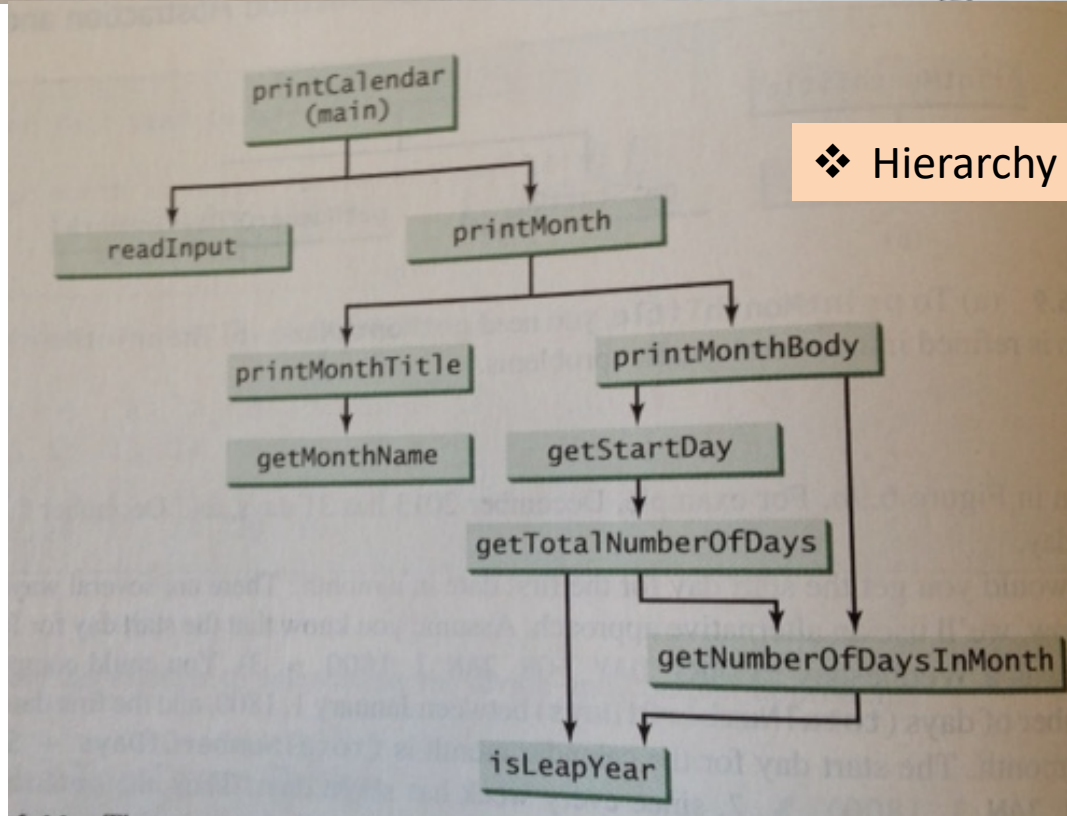
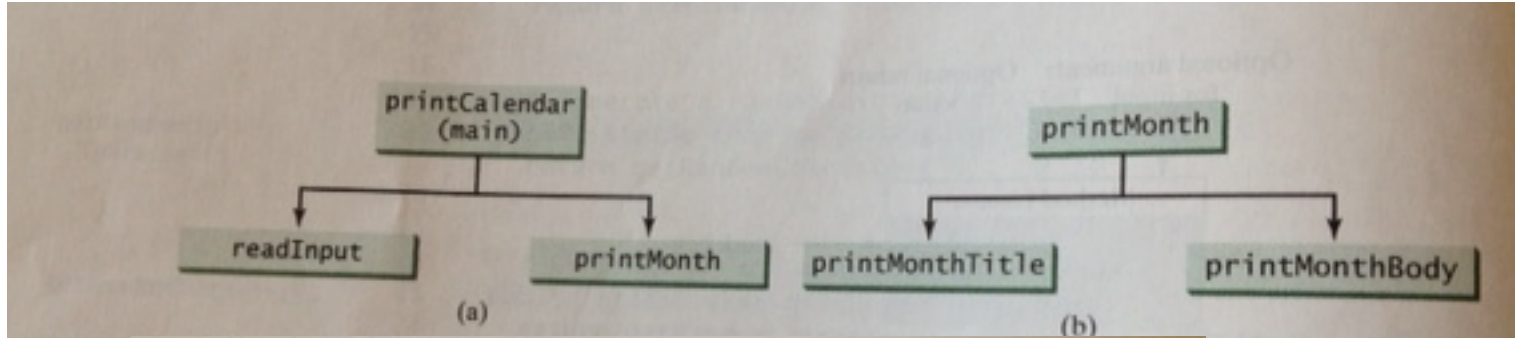
```
//convert miles to meters  
static double miles2m(double m) {
```

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

Methods: Structuring

Ch 6

Calendar App Example



❖ Hierarchy of Methods

Methods: Structuring

Ch 6

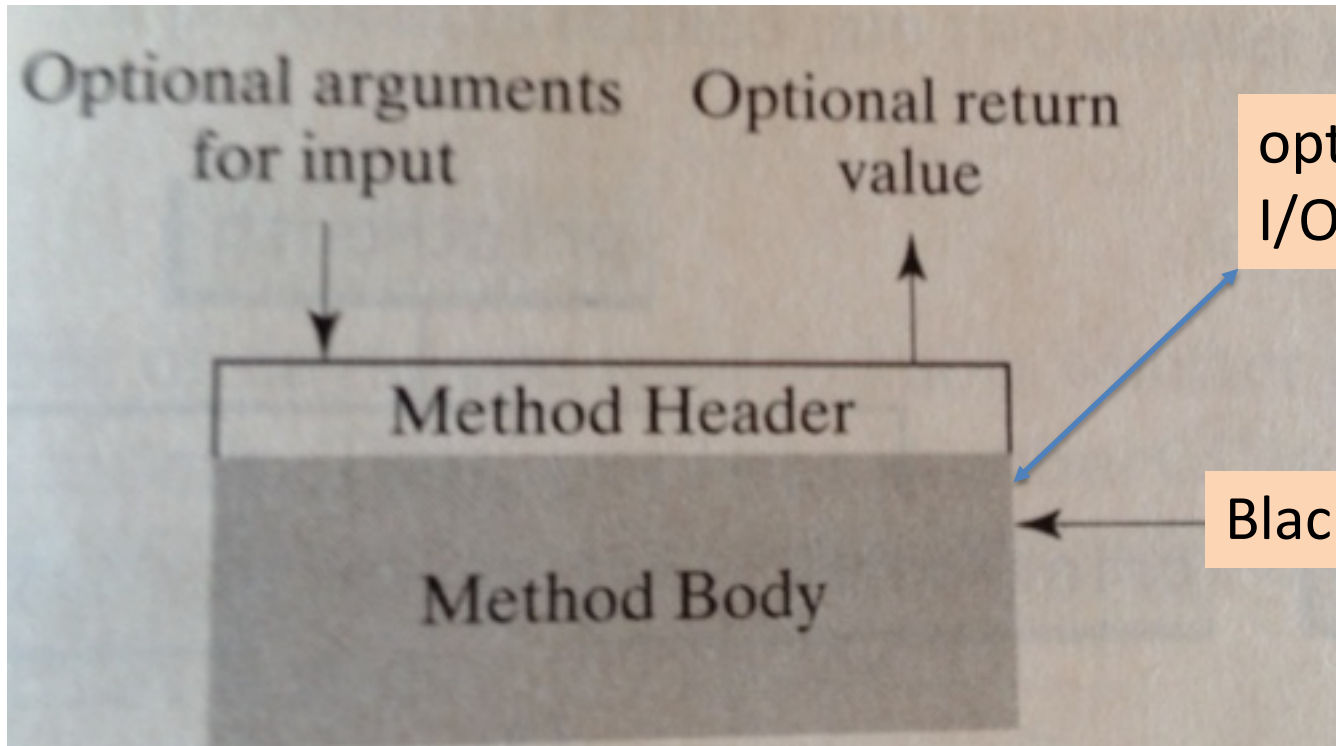
Calendar App Example

```
LISTING 6.12 PrintCalendar.java
1 import java.util.Scanner;
2
3 public class PrintCalendar {
4     /** Main method */
5     public static void main(String[] args) {
6         Scanner input = new Scanner(System.in);
7
8         // Prompt the user to enter year
9         System.out.print("Enter full year (e.g., 2012): ");
10        int year = input.nextInt();
11
12        // Prompt the user to enter month
13        System.out.print("Enter month as a number between 1 and
14        int month = input.nextInt();
15
16        // Print calendar for the month of the year
17        printMonth(year, month);
18    }
19
20    /** Print the calendar for a month in a year */
21    public static void printMonth(int year, int month) {
22        // Print the headings of the calendar
23        printMonthTitle(year, month);
24
25        // Print the body of the calendar
26        printMonthBody(year, month);
27    }
28
29    /** Print the month title, e.g., March 2012 */
30    public static void printMonthTitle(int year, int month) {
31        System.out.println("
32        + " " + year);
33        System.out.println("-----");
34        System.out.println(" Sun Mon Tue Wed Thu Fri Sat");
35    }
36 }
```

Abstraction/Encapsulation

Ch 6

Hiding



Methods: *Overloading*

Ch 6

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 int values */
public static int max(int num1, int num2) {
    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);
}
```


Methods: *Overloading*

Ch 6

Example code

```
17 //calling Meth1
18 Meth1();
19 System.out.println("called Meth1()");
20 Meth1(123);
21 System.out.println("called Meth1(pInt)");
22 Meth1("hello 1");
23 System.out.println("called Meth1(pStr)");
24 Meth1(234, "hello 2");
25 System.out.println("called Meth1(pInt,pStr)");
26 Meth1("hello 3", 345);
27 System.out.println("called Meth1(pStr,pInt)");
28
29 } //end main method
30 //testing methods below main
31 public static void Meth1() {
32 System.out.println("got Meth1()");
33 //return?
34 }
35 public static void Meth1(int pInt) {
36 System.out.println("got Meth1(pInt)..." + pInt);
37 }
38 public static void Meth1(String pStr) {
39 System.out.println("got Meth1(pStr)..." + pStr);
40 }
41 public static void Meth1(int pInt, String pStr) {
42 System.out.println("got Meth1(pInt,pStr)..." + pInt + ".." + pStr);
43 }
44 public static void Meth1(String pStr, int pInt) {
45 System.out.println("got Meth1(pStr,pInt)..." + pStr + ".." + pInt);
46 }
47 } //end class
```

Methods: *Overloading*

Ch 6

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

Ch 6

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) in  
1 error
```

Parameter Overloading

Ch 6

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

```
----jGRASP exec:
```

```
parms=1c
```

```
array ptr → p3=[ I@7852e922
```

Parameter Overloading

Ch 6

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
```

array[4] → {
 p3=0
 p3=0
 p3=0
 p3=0

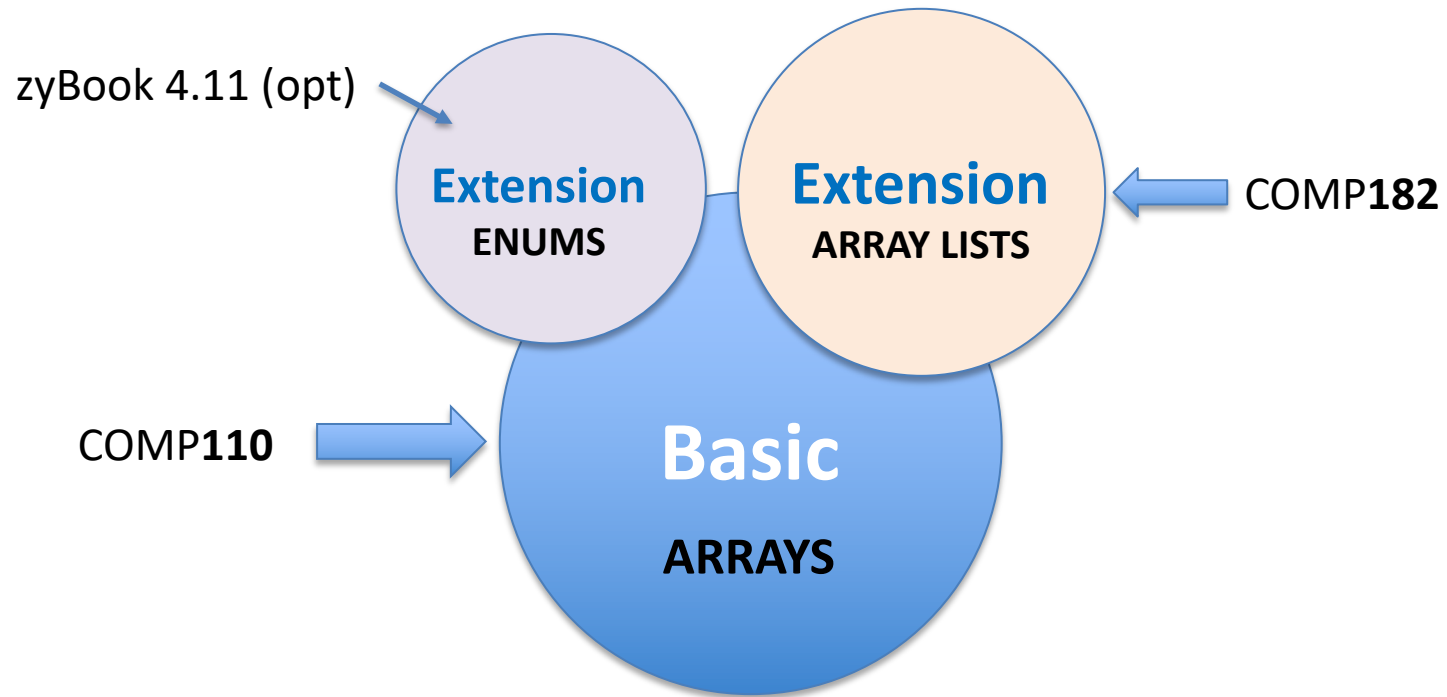
Chapter 7



Ch 7

Arrays

Basics vs. Extensions



Enums

“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

Ch 7/5

- ❖ 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

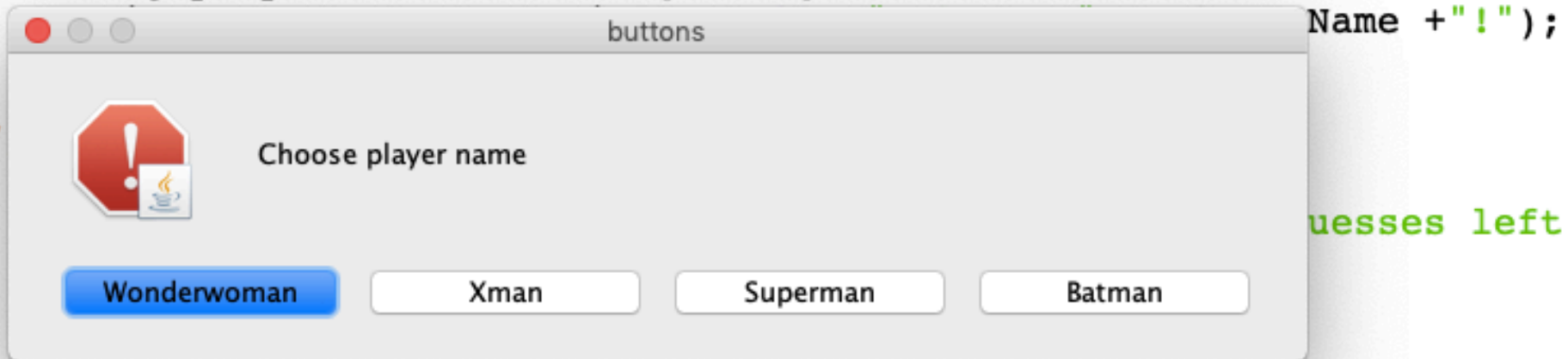
```
int[] myArr = new int[10];
```

```
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
```

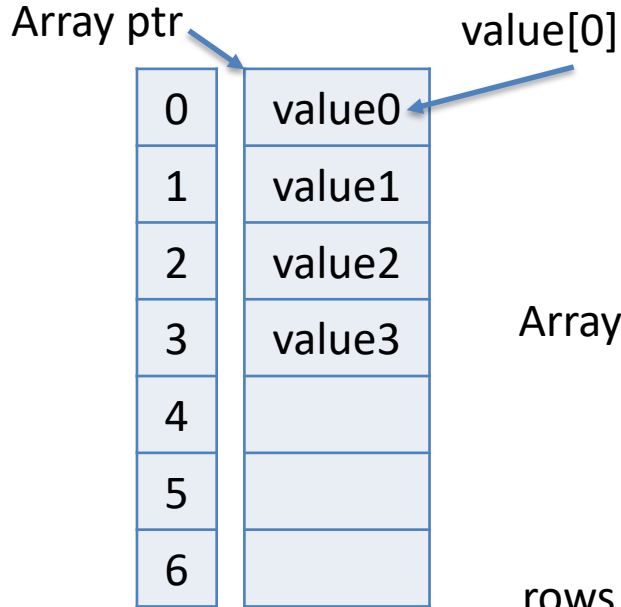


```
Name +"!");
```

```
guesses left.\
```

Arrays

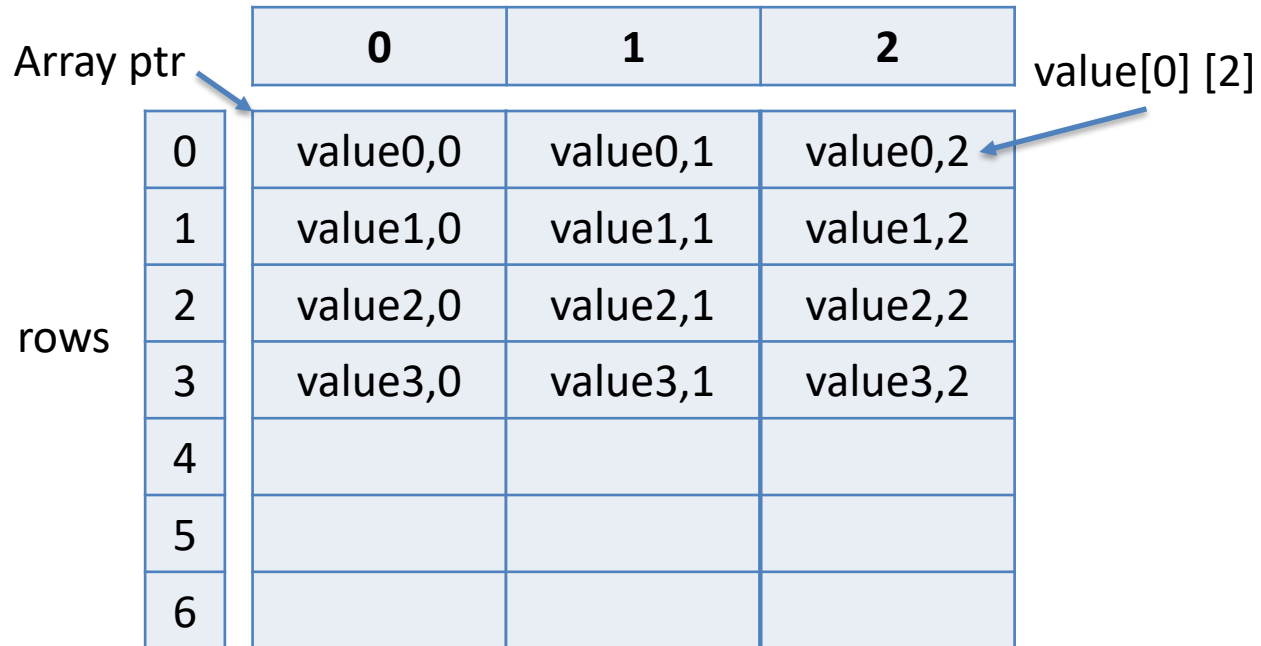
❖ Indexed



1D

2D

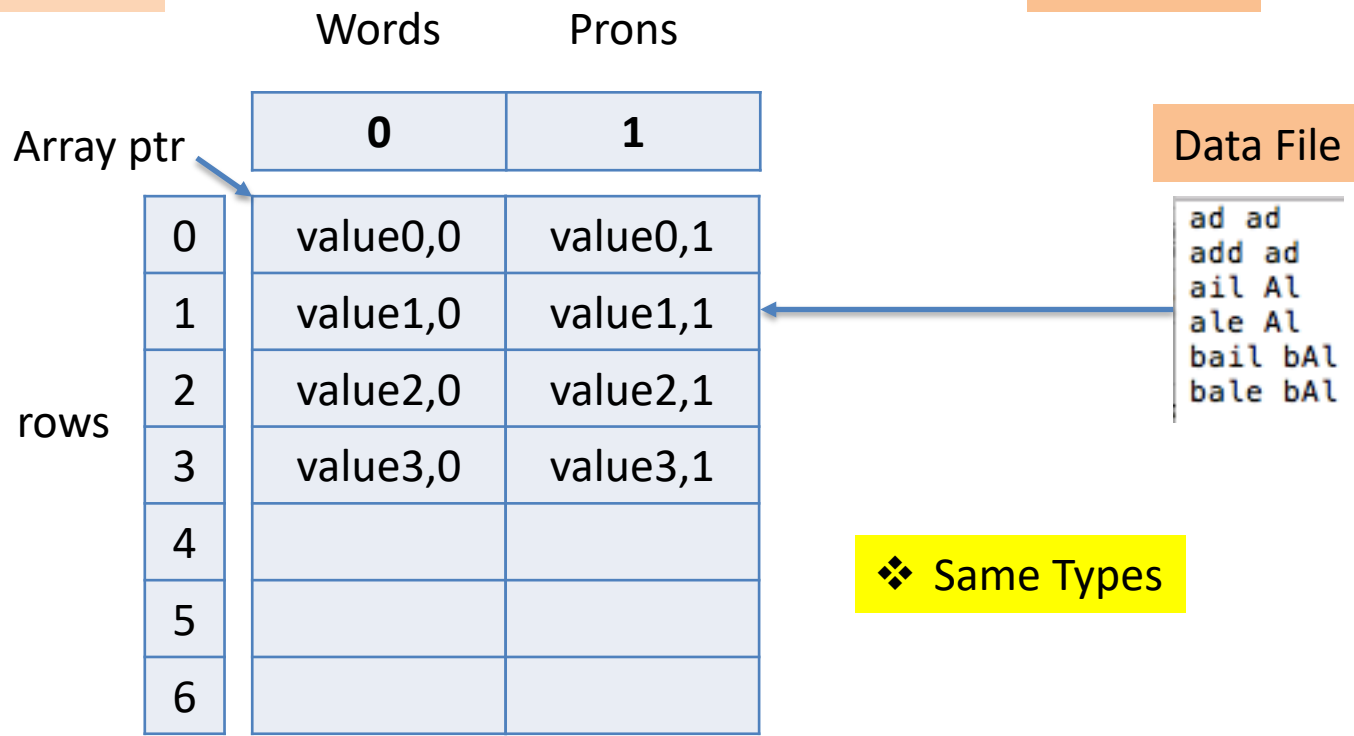
columns



Arrays

Ch 7/5

2D option



Arrays

❖ Associated

2x 1D option

Word Array ptr

0	value0
1	value1
2	value2
3	value3
4	
5	
6	

value[0]

Data File

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

❖ Dif Types (can be)

Pron Array ptr

0	value0
1	value1
2	value2
3	value3
4	
5	
6	

value[0]

❖ 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

Ch 7/5

❖ 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

Ch 7/5

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

Ch 7/5

Declare/Instantiate Example:

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

Declare/Instantiate & *Initialize* Example:

(0..9)

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

Fill Examples:

(0..5) → 6

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

{0..9}

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

or use **i++** → {0..9}

❖ *Foreach*

Using Arrays

Ch 7/5

Declare/Instantiate Example:

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

(0..9)

Reference Example:

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

Error: array subscript out of bounds

ArrayIndexOutOfBoundsException

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;}
```

Oversizing

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;
```



Array Example

find keys

```
char key, int[] numArr = new int[10]; //inits
for(i=0; i<10; i++) {numArr[i] = i;} {0..9}
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");
```



Strings vs. Arrays

❖ Strings

```
String alpha = "abcdefghijklmnopqrstuvwxy";
```

❖ Arrays

```
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 abc def ghi jkl mno  
pqrstuv wxyz";
```

```
//find key
```

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

```
//print key -- same
```



Arrays – Operations

Ch 7/5

❖ Common Array Operations

Initialize

Compare

➤ using **for** loops

Copy

Shift

Reverse

Shuffle

Parameter Passing

➤ Pass by *Sharing*

➤ by **Reference**

Arrays: Copy

in the target array. The following code, for instance, copies `sourceArray` into the target array 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.System` class instead of using a loop. The syntax for `arraycopy` is:

```
arraycopy(sourceArray, srcPos, targetArray, tarPos, length)
```

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

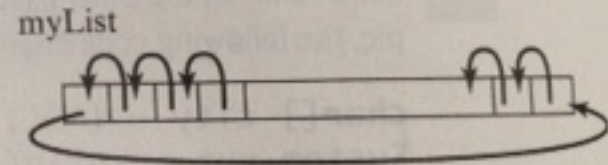
```
System.arraycopy(sourceArray, 0, targetArray, 0, sourceArray.length);
```

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:

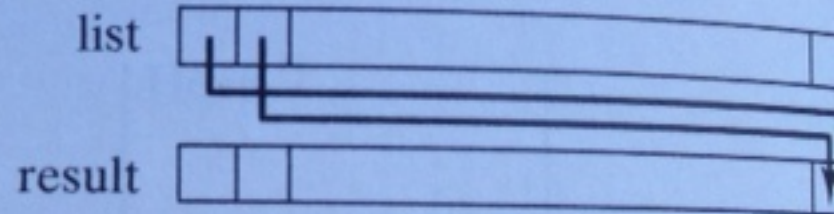
❖ must use a **temp**

```
double temp = myList[0]; // Retain the first element  
  
/ Shift elements left  
for (int i = 1; i < myList.length; i++) {  
    myList[i - 1] = myList[i];  
}  
  
/ Move the first element to fill in the last position  
myList[myList.length - 1] = 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;  
}
```



creates a new array **result**. Lines 4–7 copy elements from array **list**. Line 9 returns the array. For example, the following statement returns an array with the elements **6, 5, 4, 3, 2, 1**.

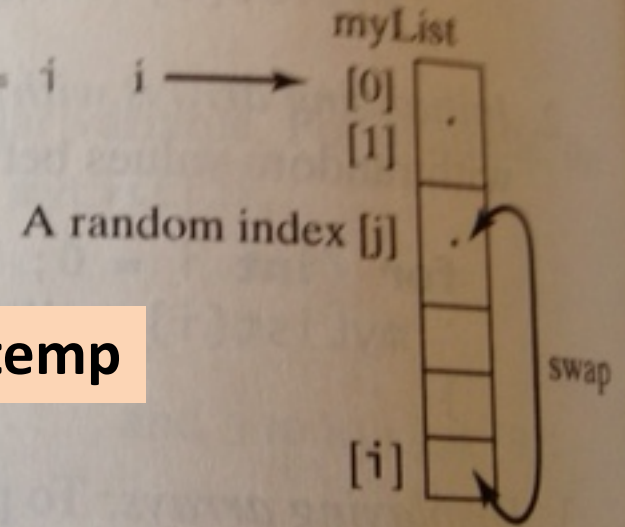
```
list1 = {1, 2, 3, 4, 5, 6};  
list2 = reverse(list1);
```

Arrays: Random Shuffling

Random shuffling: In many applications, you need to shuffle the elements 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:

```
for (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]  
    double temp = myList[i];  
    myList[i] = myList[j];  
    myList[j] = temp;  
}
```

❖ must use a **temp**



Arrays

Liang Ch 7

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

```
public 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

```

18     while (!done) { //main loop
19         //get test#
20         System.out.print("Enter test #: ");
21         $TEST = tnum.nextInt();
22         switch($TEST) {
23             case -1:
24                 codes(0);
25                 break;
26             case 0:
27                 wholeArr();
28                 break;
29             case 1:
30                 simpleArr1D();
31                 break;
32             case 2:
33                 simpleArr2D();
34                 break;
35             case 3:
36                 testStr("course", "source");//anagrams
37                 break;
38             case 4:
39                 sortArr(names);//parm=array
40                 break;
41             case 5:
42                 split(names);//parm=array
43         } //end switch
44         String msg = "Do you want to Try again?";

```

ASCII + Unicodes

Test -1

```
61 //ASCII & Unicodes
62     static void codes(int n) {
63         //print all chars at 40 per line
64         for(int i=0; i<128; i++){
65             if(i%40==0) System.out.print("\ni=" +i +": "); //newlines
66             System.out.print((char)i);
67             if(i<32) System.out.print("|");
68         } //end for
69         System.out.println("\n---"); //final newline
70         for(int i=128; i<256; i++){
71             if((i-128)%40==0) System.out.print("\ni=" +i +": "); //newlines
72             System.out.print((char)i + " ");
73         } //end for
74         System.out.println(); //final newline
75     } //end codes
```

ASCII Codes- 7-bit

USASCII code chart

Bits					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	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	ENO	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 + Unicode

CS110

Plane 0

```
i=0: 0000 0001 0008

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

i=128: € , f „ … † ‡ ^ % Š < Œ ‘ ’ “ ” - - ~ ™ š > œ ÿ İ ç £ ¤ ¥ ¦ §
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

debug: starting main
Enter test #: -1

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

7-bit ASCII

Upper half Unicode – spaced out

```
i=128: € , f „ … † ‡ ^ % Š < Œ ‘ ’ “ ” - - ~ ™ š > œ ÿ İ ç £ ¤ ¥ ¦ §
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
```

Whole Array

Test 0

```

76 //whole arrays
77     static void wholeArr() {
78         int[] A = {1,2,3}, B = {100,20,30};
79         for (int x: A) { //for each loop
80             System.out.print("\t" +x);}
81     A = B; //make A point to B
82     System.out.println("\nA=" +A +"\tB=" +B); //pointers
83     for (int x: A) { //print A again
84         System.out.print("\t" +x);}
85     System.out.println(); //newline
86 } //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

CS110

Test 1

```

29 //Simple arrays: 1D, 2D
30 static void simpleArr() {
31     //1D
32     int[] xArr = {1,2,3,4}; 1D
33     System.out.println(xArr[1]); //element
34     System.out.println(xArr); //pointer
35     for (int ez: xArr) { //for each loop
36         System.out.printf("%4d", ez);
37     }
38     System.out.println("\nend of 1D\n---");
39     //2D
40     int[][] x2Arr = {{1,2,3,4},{5,6,7,8}}; 2D
41     System.out.println(x2Arr[1][1]); //element
42     for (int i=0; i<4; i++) { //print row 0
43         System.out.print("\t" + x2Arr[0][i]);
44     }
45     System.out.println("\nFOR-end of row 0\n---");
46     for (int ez: x2Arr[1]) { //print row 1
47         System.out.print(ez);
48     }
49     System.out.println("\nFOREACH-end of row 1\n---");
50 } //end simpleArr

```

Arrays Example #1

```
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!
```

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!
```


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)`

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

Ch 8

Multi-Dimensional Arrays

Arrays

Ch 8

2D

❖ Database example

columns = *fields*

Array ptr

value[0] [4]

rows =
records

	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 [] []

Ch 8

```
int[] list = new int[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

Ch 8

```
8
9 public class testArrays {
10 public static void main(String[] args) {
11     int i=0;
12     /*test I/O
13     System.out.println("Hello World\n");
14     //test code
15     int[] xArr = {1,2,3,4};
16     System.out.println(xArr[1]); //element
17     System.out.println(xArr); //pointer
18     for (int ez: xArr) { //for each loop
19     System.out.println(ez);
20     }
21     //2-dim arrays
22     int[][] x2Arr = {{1,2,3,4},{5,6,7,8}};
23     System.out.println(x2Arr[1][1]); //element
24     for (int ez: x2Arr[1]) { //for each loop
25     System.out.println(x2Arr);
26     }
27     for (i=0; i<4; i++) { //for each loop
28     System.out.println(x2Arr[1][i]);
29     }
30 }
```


Section

Ch 7

Command Line Args

Command Line Args

Ch 7

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

Section



Date & Time

Time (System Class)

LISTING 2.9 ShowCurrentTime.java

```
// Obtain the total milliseconds since the midnight, Jan 1, 1970  
long totalMilliseconds = System.currentTimeMillis();
```

❖ org 1-1-1970

```
2.13 Programming S  
7   long totalSeconds = totalMilliseconds / 1000;  
8  
9   // Compute the current second in the minute in the hour  
10  int currentSecond = (int)(totalSeconds % 60);  
11  
12  // Obtain the total minutes  
13  long totalMinutes = totalSeconds / 60;  
14  
15  // Compute the current minute in the hour  
16  int currentMinute = (int)(totalMinutes % 60);  
17  
18  // Obtain the total hours  
19  long totalHours = totalMinutes / 60;  
20  
21  // Compute the current hour  
22  int currentHour = (int)(totalHours % 24);  
23  
24  // Display results  
25  System.out.println("Current time is " + currentHour + ":"  
26    + currentMinute + ":" + currentSecond + " GMT");  
27  }  
28 }
```

```
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());
```

❖ `date.getTime()`

displays the output like this:

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

Date & Time Examples

```
9 public class date {  
10 public static void main(String[] args) {  
11     byte count = 0;  
12     System.out.println("Hello World\n");  
13     Date daat = new Date();  
14     long dat = daat.getTime();  
15     System.out.println(dat);  
16     System.out.println(daat.toString());  
17 } //end main  
18 } //end class
```

Hello World

1479284710778

Wed Nov 16 00:25:10 PST 2016

ms=1541015090523
ms=1541015090524

Miniscule (or 0) difference

Date & Time Examples

```
10 public class Datime {
11     public static void main(String[] args) {
12         byte count =0;
13         System.out.println("Hello World");
14         Date daat = new Date();
15         long ms = daat.getTime();
16         System.out.println("ms1=" + ms);
17         ms = daat.getTime();
18         System.out.println("ms2=" + ms);
19         String dateStr = daat.toString();
20         System.out.println(daat + "\tstring= " +dateStr); //date string
21         System.out.println("getTime=" + daat.getTime());
22
23         //breakpoint!
24         int cont = JOptionPane.showConfirmDialog(null, "continue?");
25         if(cont>0) System.exit(25);
26         System.out.println("---continue---");
27     }
```

```
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!
```

```
28 //java.time (library package)
29 System.out.println("LocalTime.now=" + LocalDateTime.now()); //new Class.method
30 for(int i=0; i<1000000; i++){ //timing loop
31 System.out.println("LocalTime.now=" + LocalDateTime.now());
32 System.out.println("get=" + new Date().getTime());
33 System.out.println("Instant.now=" + Instant.now());
34 //System.out.println("Clock.now=" + Clock.now());
35 //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

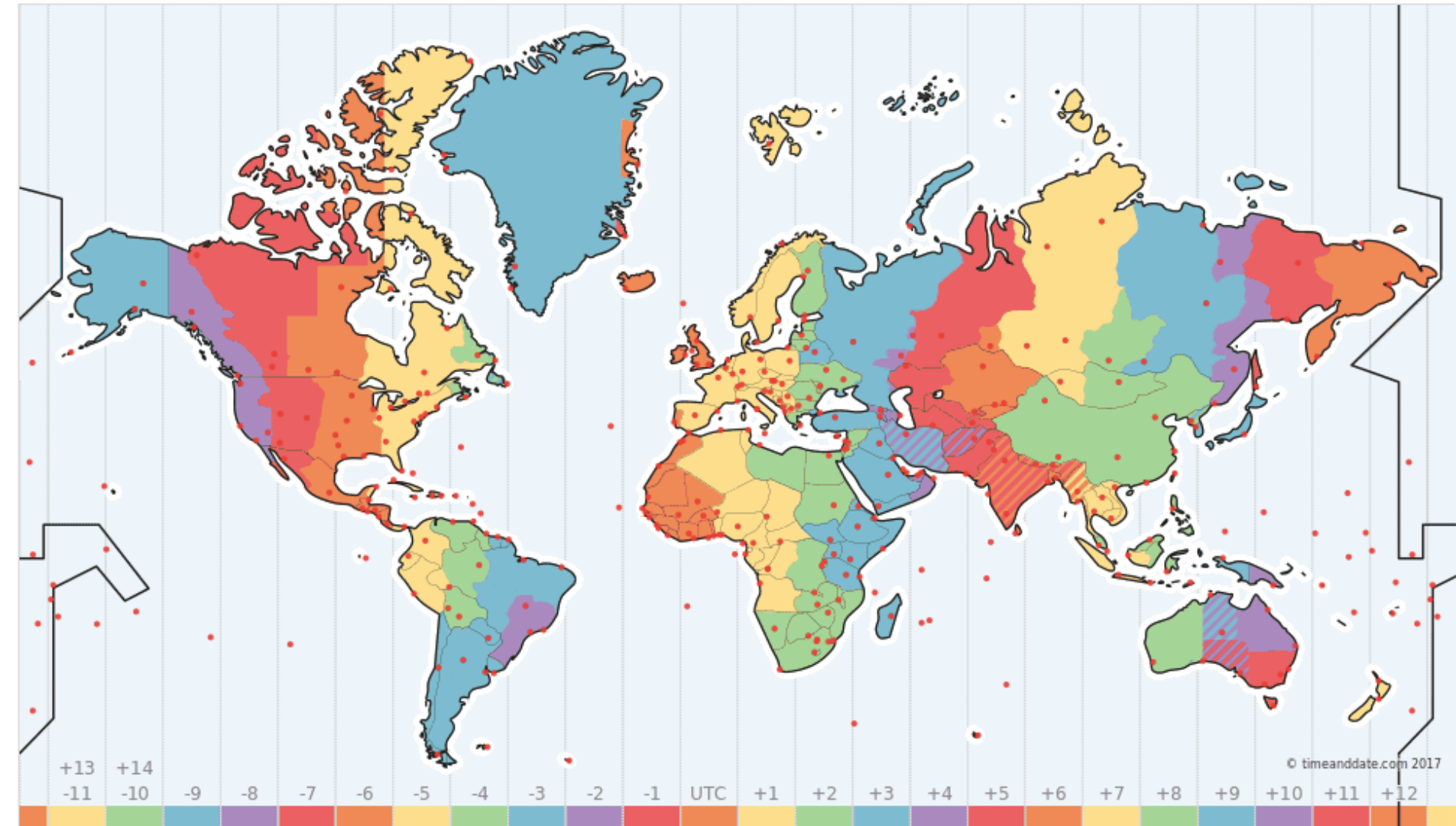
PST<>PDT

```
42 //converts
43 int hrs = (int)(ms/1000/3600);
44 int hour = hrs %24;
45 System.out.println("calc hour(GMT 24)=" +hour);
46 int pstHr = (hour-8)%24; //PST time zone
47 System.out.println("calc hour(PST 24)=" +pstHr);
48 int mins = (int)(ms/1000/60);
49 System.out.println("calc mins=" +(mins %(24*60) -hour*60));
50 int ix = 11; //index in date string
51 String pst = dateStr.substring(ix, ix+8);
52 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



Date & Time Examples

World Times

```
59 //world times
60 String[] cities = {"LA", "NY", "UK", "Paris", "Athens", "Tokyo", "Sydney",
61 int[] zones = {0, 3, 8, 9, 10, 17, 19, -2};
62 System.out.println("World Times ...");
63 for(int i=0; i<zones.length; i++) {
64     int worldHr = (pstHr +zones[i] +24) %24;//formula
65     if(zones[i]<0 && worldHr> pstHr)
66         System.out.println("**yesterday**");
67     System.out.println(cities[i] + " = " +worldHr +pst.substring(2));
68 }//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!
```

```
24 //timing loop  
25 for(int i=0; i<5; i++)  
26 System.out.println("time.now=" + LocalDateTime.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();
```

```
25 //java.time (library package)  
26 System.out.println("LocalTime.now=" + LocalTime.now());  
27 for(int i=0; i<1000000; i++){ } //timing loop  
28 System.out.println("LocalTime.now=" + LocalTime.now());  
29 System.out.println("get=" + new Date().getTime());  
30 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.

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

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

See: Description

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.

❖ US **GPS**: 30+ satellites (since 1978)

❖ Alternatives as backups

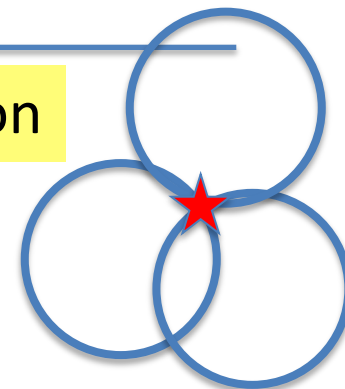
❑ **Iridium** – 67+ satellites

❑ DARPA **ANS**

❑ USAF/Aerospace **Sextant**

➤ **Triangulation**

4th satellite
for elevation



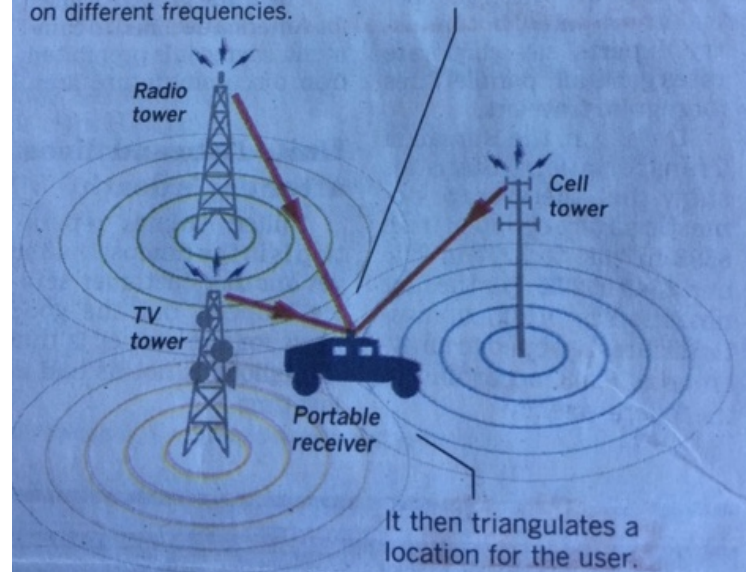
Two GPS alternatives

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.

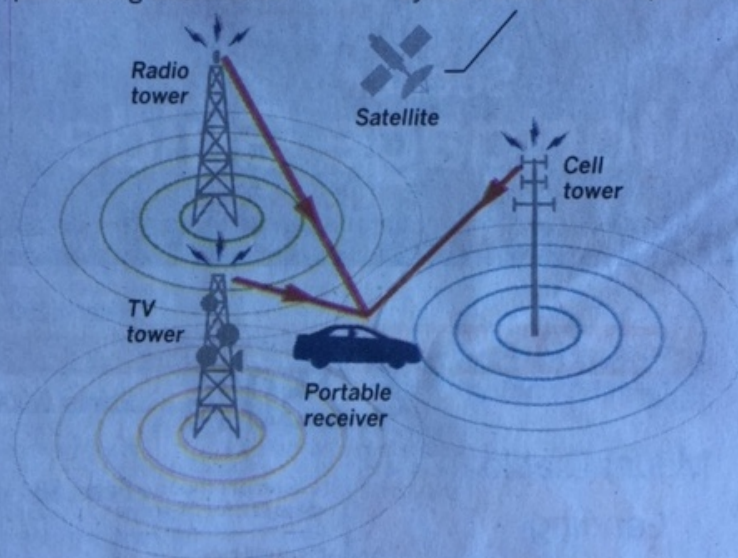
ANS' algorithm separates time and location data collected from signals.



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.



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

LORENA INIGUEZ ELEBEE Los Angeles Times

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* $0 < N < 1 \Leftrightarrow [0-1]$
0.000000001 to 0.999999999

0.639137804508209

*10 → 6.391378045...

0.22459988296032

0.566199541091919

for $0 \leq N \leq 9$

use

`(int) (Math.random() * 10)`

*10 → 6.391378045...

int → 6

for $1 \leq N \leq 10$

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 → 6.39137...

Random Numbers

Alternative

❖ Class: Random

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

➤ Returns 0..9 {10 values}

Generating Randoms

```
18 //Random number generation using "random" method
19 int N = 10;
20 while(run) {
21     double randFlt = Math.random();
22     System.out.println("Float number= " + randFlt);
23     double rand10Flt = N * randFlt;
24     System.out.println("10x Float number= " + rand10Flt);
25     int rand10 = (int)rand10Flt;
26     System.out.println("10x Integer= " + rand10);
27 //Random number generation using "Random" Class
28 long seed = 3;
29 Random Rand1 = new Random();//default seed
30 Random Rand2 = new Random(seed);//using seed
31 int num1 = Rand1.nextInt(10);
32 int num2 = Rand2.nextInt(10);
33 System.out.println("Class NO seed=" + num1);
34 System.out.println("Class with seed=" + num2);
35 int cont = JOptionPane.showConfirmDialog(null, "continue?");
36     switch (cont) {
37         case 0: System.out.println("keep going...");
38             break;
39         default: System.out.println("good-bye!");
40             run = false; //terminate loop
41     } //end switch
42 } //end loop
43 } //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

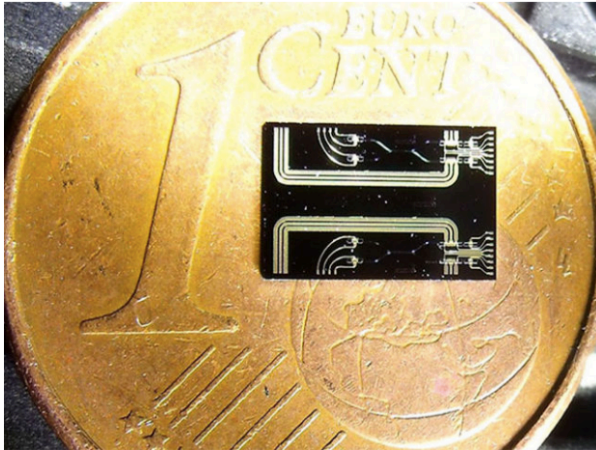


Photo: Daniel Bartolome and Ona Bombi/ICFO

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.

Monty Hall Problem

➤ 3 Prisoners Problem

Monty Hall Problem

➤ Pick a Door



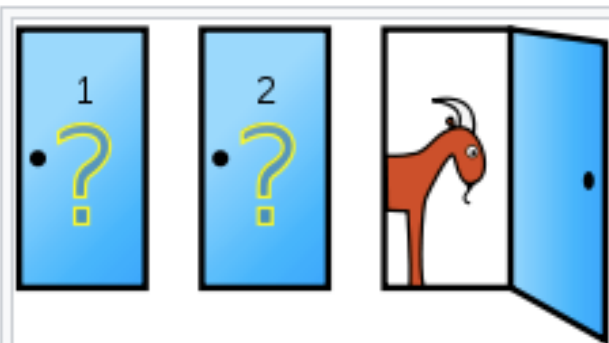
Monty Hall Problem

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

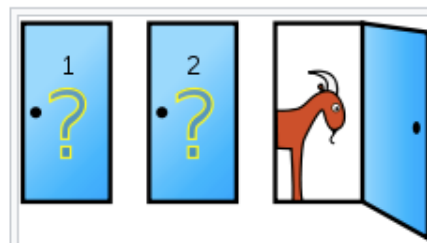
Vos Savant's response was that the contestant should switch to the other door (vos Savant 1990a). Under the standard assumptions, contestants who switch have a $\frac{2}{3}$ chance of winning the car, while contestants who stick to their initial choice have only a $\frac{1}{3}$ chance.

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] 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.

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

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

Randoms for Monty Hall

```

27     case 1: //gen randoms
28     System.out.println("using random method:");
29     for(int run = 0; run < max; run++) {
30         int rand3 = randNum(N); //get rand 1-3
31         if (run < prx) System.out.print(rand3 + spc); //1st few
32         countsMeth[rand3-1]++;
33     } //end 1st loop
34     System.out.println("\ncounts (method):");
35     for(int x : countsMeth) System.out.print("\t" + x);
36     System.out.println("\n\nUsing Random Class:");
37     //next loop
38     for(int run = 0; run <= max; run++) {
39         int Rand3 = randCl(N); //get rand 1-3
40         if (run < prx) System.out.print(Rand3 + spc);
41         countsCl[Rand3-1]++;
42     } //end 2nd loop
43     System.out.println("\ncounts (Class):");
44     for(int x : countsCl) System.out.print("\t" + x);
45     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

```

47 case 2: //Monty Hall play
48 for(int i=0; i<max; i++) {
49     int pick = randNum(N) +1;//pick door 1-3
50     int win = randNum(N) +1;//set win door
51     //keep door
52     keep[i] = (pick == win);//win=True
53     if (keep[i]) countKeep++;//count
54     //swap door
55     //pick = (pick%3) +1;//next door
56     //if (pick != win) pick = (pick%3) +1;//next door
57     swap[i] = (pick != win);
58     if (swap[i]) countSwap++;//count
59 }//end for
60 //print
61 System.out.print("\nif Keep: ");
62 for (int i=0; i<50; i++) {
63     char wLK = (keep[i]? 'W':'.');
64     System.out.print(wLK);}
65 System.out.print("\nif Swap: ");
66 for (int i=0; i<50; i++) {
67     char wLS = (swap[i]? 'W':'.');
68     if (swap[i] == keep[i]) wLS = 'x';
69     System.out.print(wLS);}
70 System.out.println("\nCounts:");
71 System.out.println("Keep= " + countKeep);
72 System.out.println("Swap= " + countSwap);

```


Random Methods

```
82 //Random number generation using "random" method: 1-3
83 static int randNum(int N) {
84     double randFlt = N * Math.random();      method
85     int rand = (int)randFlt +1; //1-3
86     if (rand <1 || rand > N) { //check 1-3
87         System.out.println("random number ERROR!");
88         System.exit(0);}
89     return rand;
90 } //end randNum
91
92 //Random number generation using "Random" Class
93 static int randCl(int N) {
94     Random Randx = new Random(); //default seed      Class
95     int Rand = Randx.nextInt(N) +1; //1-3
96     if (Rand <1 || Rand > N) { //check
97         System.out.println("Random Class ERROR!");
98         System.exit(0);}
99     return Rand;
.00 } //end randCl
```

Monty Hall Results

```
69     char wls = (swap[i]? 'W': '.');
70     if (swap[i] == keep[i]) wls = 'x';
71     System.out.print(wls);}
72 System.out.println("\nCounts:");
73 System.out.print("Keep= " + countKeep);
74 pctKeep = (float)(countKeep *100)/max;
75 System.out.println("\t=" +pctKeep + "%");
76 System.out.print("Swap= " + countSwap);
77 pctSwap = (float)(countSwap *100)/max;
78 System.out.println("\t=" +pctSwap + "%");
79 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...WWW..W..W.WWW...WWW.WW
Counts:
Keep= 3339   =33.39%
Swap= 6661   =66.61%
Total= 10000
```

Theory: Computability

Uncomputable?

Are there any problems in computer programming, that are seen as impossible?



Thomas Cormen, I've been teaching Computer Science at Dartmouth College since 1992.



Written Nov 5 · Upvoted by Timothy Johnson, PhD student in CS, UC Irvine and Jeff Nelson, [Invented Chromebook](#), [#Xoogler](#)

Yes, indeed. The best-known one is the [Halting Problem](#), shown to be uncomputable by Alan Turing. Suppose you want to know whether a program, call it P , given an input, call it x , would run to completion. Not whether P produces a correct answer, and not whether P produces an answer at all. Just: does program P , running on input x , run to completion? That is, does P running on x halt? Turing proved that it is impossible to write a computer program that takes two inputs, P and x , and correctly tells you *every time* whether P running on x halts.

❖ Halting

Once you have one uncomputable problem, you can find others. For example, [Post's Correspondence Problem](#). Here, we are given two lists of n strings, say A_1, A_2, \dots, A_n and B_1, B_2, \dots, B_n . The problem is to determine whether there exists a sequence of indices i_1, i_2, \dots, i_m such that $A_{i_1}, A_{i_2}, A_{i_3}, \dots, A_{i_m}$ (the strings $A_{i_1}, A_{i_2}, A_{i_3}, \dots, A_{i_m}$ concatenated together) gives the same string as $B_{i_1}, B_{i_2}, B_{i_3}, \dots, B_{i_m}$. Using an example I wrote in *Algorithms Unlocked*, suppose that $n = 5$, and $A_1 = \text{ey}, A_2 = \text{er}, A_3 = \text{mo}, A_4 = \text{on}, A_5 = \text{h}$ and $B_1 = \text{ym}, B_2 = \text{r}, B_3 = \text{oon}, B_4 = \text{e}, B_5 = \text{hon}$. Then the sequence $\langle 5, 4, 1, 3, 4, 2 \rangle$ works, since both $A_5 A_4 A_1 A_3 A_4 A_2$ and $B_5 B_4 B_1 B_3 B_4 B_2$ form honeymooner.

❖ Strings