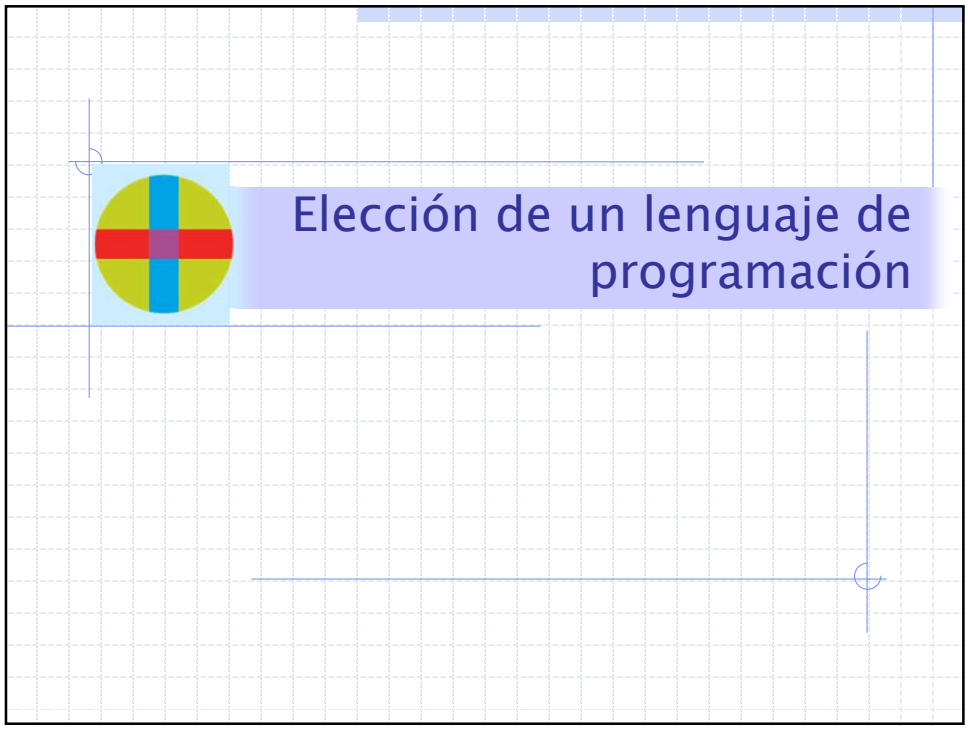
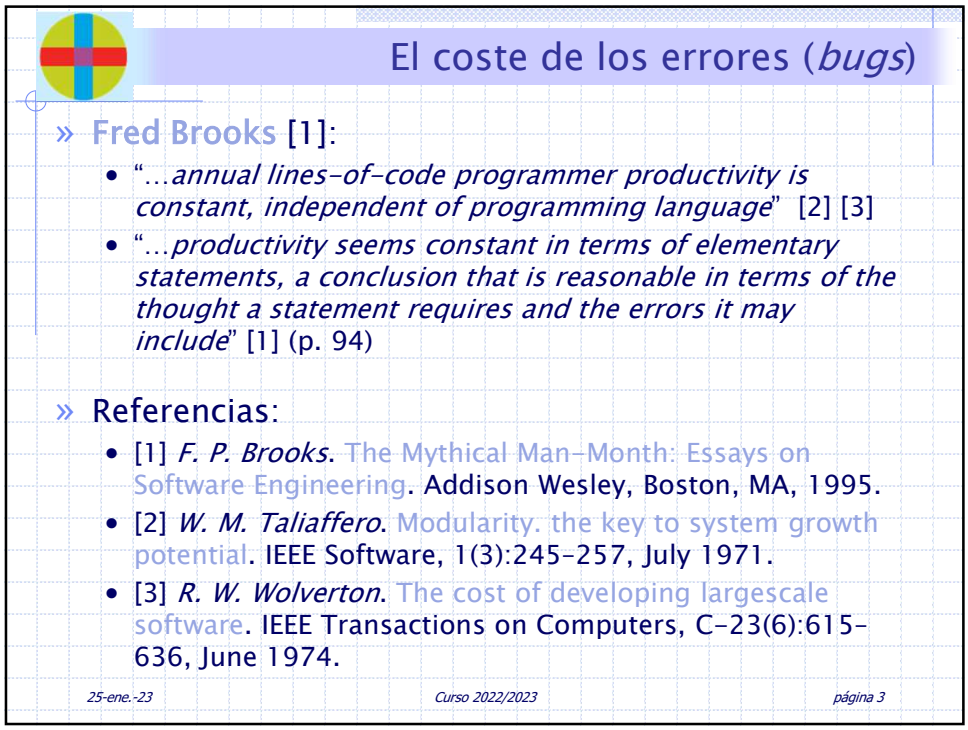

Tecnología de la programación
Raúl García García
Curso 2022/2023
Universidad San Pablo-CEU
Escuela Politécnica Superior
Campus de Montepríncipe

1


Elección de un lenguaje de programación

2

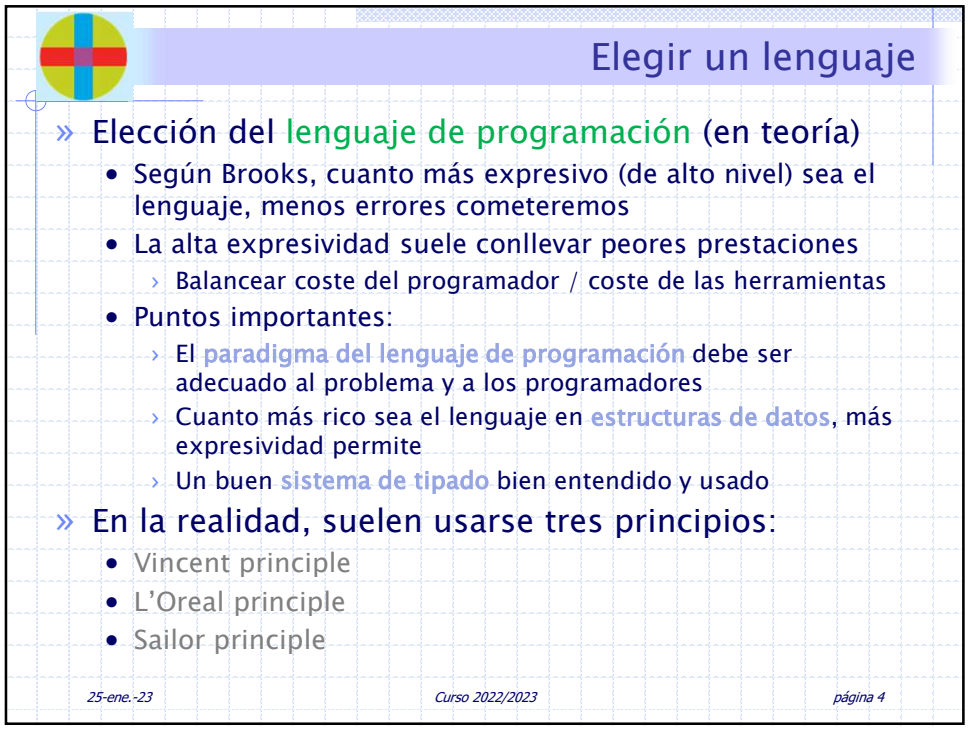


El coste de los errores (*bugs*)

- Fred Brooks [1]:
  - "…*annual lines-of-code programmer productivity is constant, independent of programming language*" [2] [3]
  - "…*productivity seems constant in terms of elementary statements, a conclusion that is reasonable in terms of the thought a statement requires and the errors it may include*" [1] (p. 94)
- Referencias:
  - [1] *F. P. Brooks*. The Mythical Man-Month: Essays on Software Engineering. Addison Wesley, Boston, MA, 1995.
  - [2] *W. M. Taliaffero*. Modularity. the key to system growth potential. IEEE Software, 1(3):245-257, July 1971.
  - [3] *R. W. Wolverton*. The cost of developing largescale software. IEEE Transactions on Computers, C-23(6):615-636, June 1974.

25-ene.-23 Curso 2022/2023 página 3

3

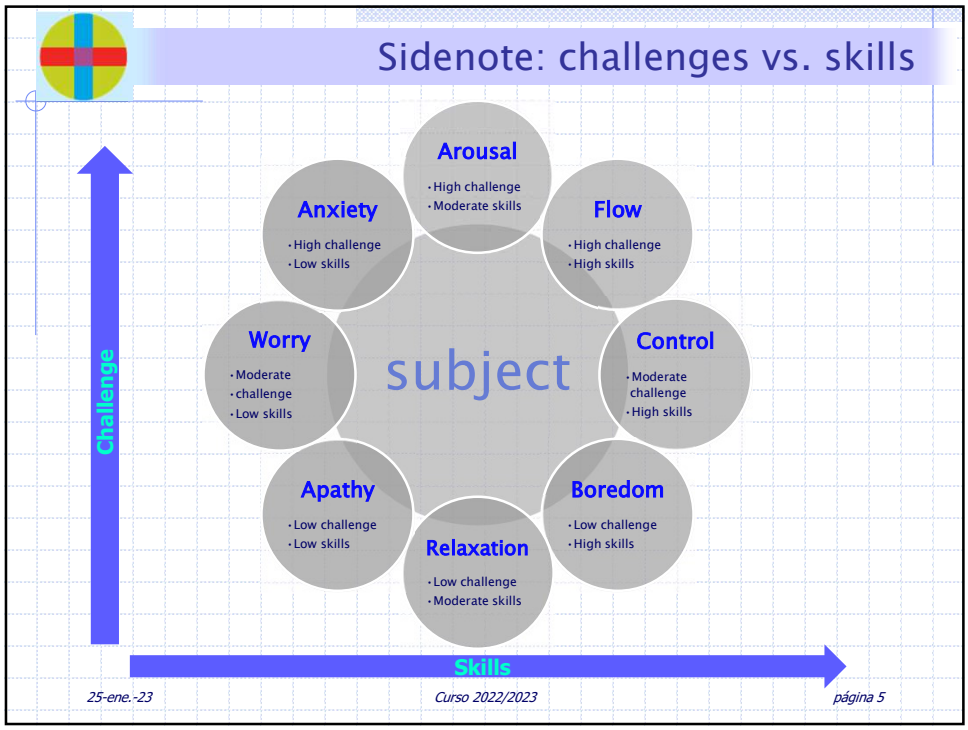


Elegir un lenguaje

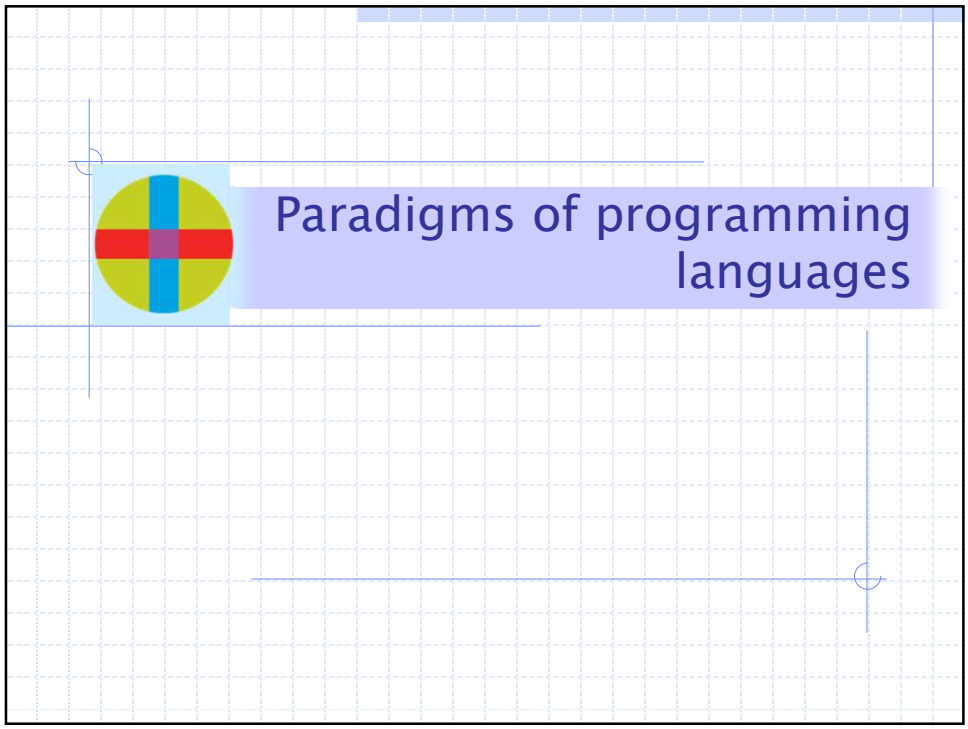
- Elección del lenguaje de programación (en teoría)
  - Según Brooks, cuanto más expresivo (de alto nivel) sea el lenguaje, menos errores cometeremos
  - La alta expresividad suele conllevar peores prestaciones
    - Balancear coste del programador / coste de las herramientas
  - Puntos importantes:
    - El paradigma del lenguaje de programación debe ser adecuado al problema y a los programadores
    - Cuanto más rico sea el lenguaje en estructuras de datos, más expresividad permite
    - Un buen sistema de tipado bien entendido y usado
- En la realidad, suelen usarse tres principios:
  - Vincent principle
  - L'Oreal principle
  - Sailor principle

25-ene.-23 Curso 2022/2023 página 4

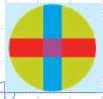
4


Sidenote: challenges vs. skills
Arousal
· High challenge
· Moderate skills
Anxiety
· High challenge
· Low skills
Flow
· High challenge
· High skills
Worry
· Moderate
· challenge
· Low skills
subject
Control
· Moderate challenge
· High skills
Apathy
· Low challenge
· Low skills
Boredom
· Low challenge
· High skills
Relaxation
· Low challenge
· Moderate skills
Challenge
Skills
25-ene.-23
Curso 2022/2023
página 5

5


Paradigms of programming languages

6

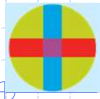


Principal Paradigms

- Imperative / Procedural
- Functional / Applicative
- Logic
- Object-Oriented
- Concatenative / Stack-based
- Concurrent
- Scripting

- In reality, very few languages are "pure"
  - Most combine features of different paradigms

7

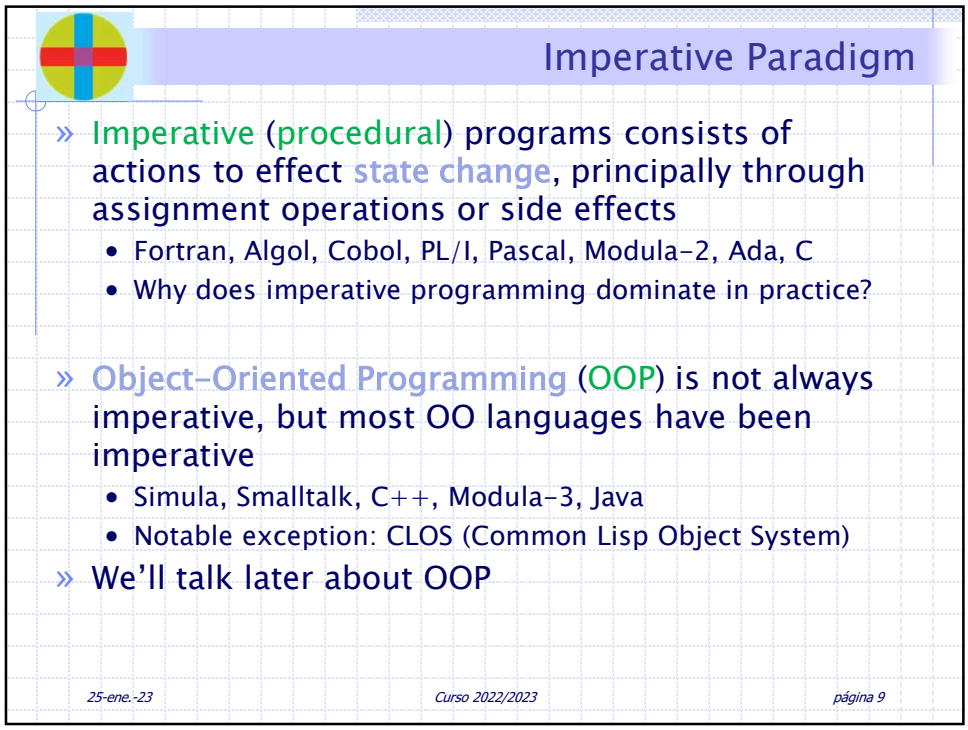


Where Do Paradigms Come From?

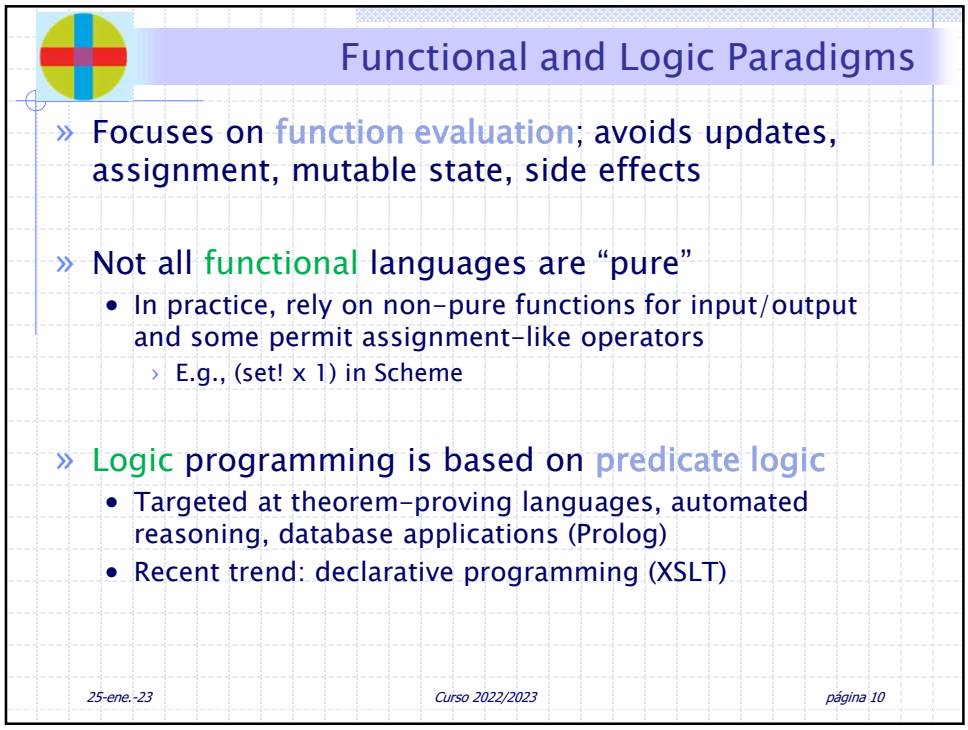
- **Paradigms** emerge as the result of social processes in which people develop ideas and create principles and practices that embody those ideas
  - Thomas Kuhn. "*The Structure of Scientific Revolutions*."

- Programming paradigms are the result of people's ideas about how programs should be constructed
  - … and formal linguistic mechanisms for expressing them
  - … and software engineering principles and practices for using the resulting programming language to solve problems

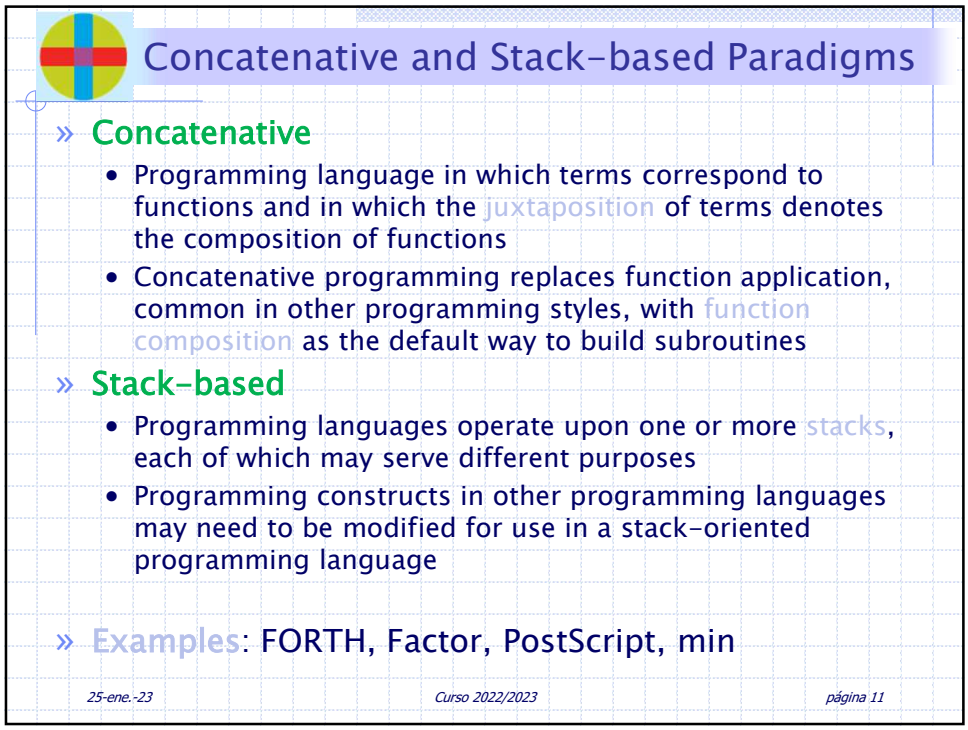
8


Imperative Paradigm
» Imperative (procedural) programs consists of actions to effect state change, principally through assignment operations or side effects
• Fortran, Algol, Cobol, PL/I, Pascal, Modula-2, Ada, C
• Why does imperative programming dominate in practice?
» Object-Oriented Programming (OOP) is not always imperative, but most OO languages have been imperative
• Simula, Smalltalk, C++, Modula-3, Java
• Notable exception: CLOS (Common Lisp Object System)
» We'll talk later about OOP
25-ene.-23 Curso 2022/2023 página 9

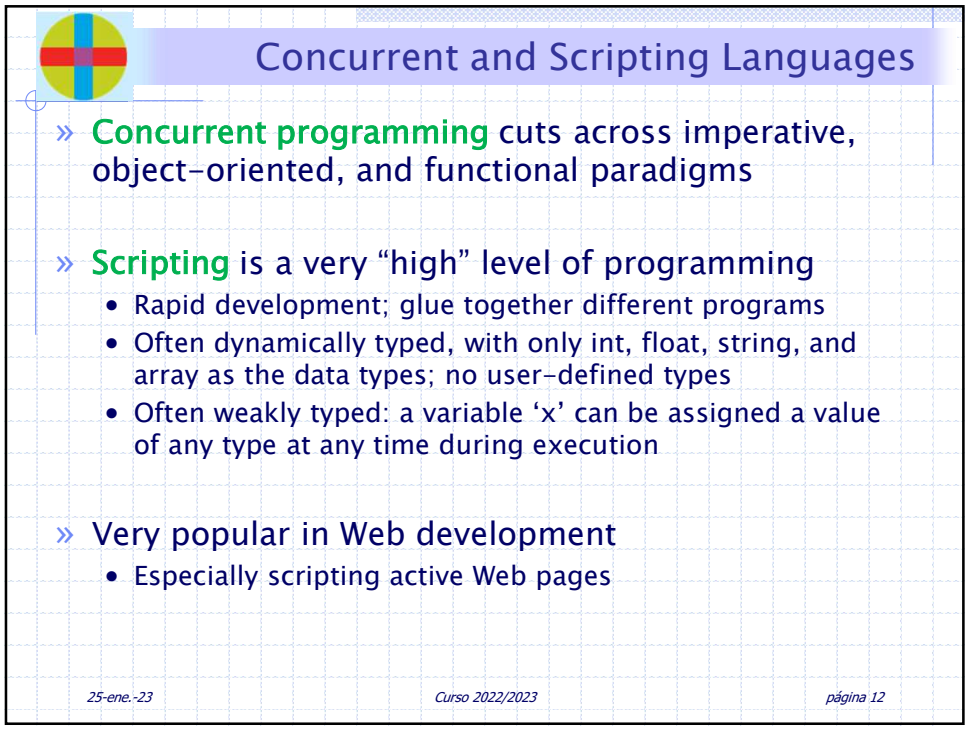
9


Functional and Logic Paradigms
» Focuses on function evaluation; avoids updates, assignment, mutable state, side effects
» Not all functional languages are "pure"
• In practice, rely on non-pure functions for input/output and some permit assignment-like operators
› E.g., (set! x 1) in Scheme
» Logic programming is based on predicate logic
• Targeted at theorem-proving languages, automated reasoning, database applications (Prolog)
• Recent trend: declarative programming (XSLT)
25-ene.-23 Curso 2022/2023 página 10

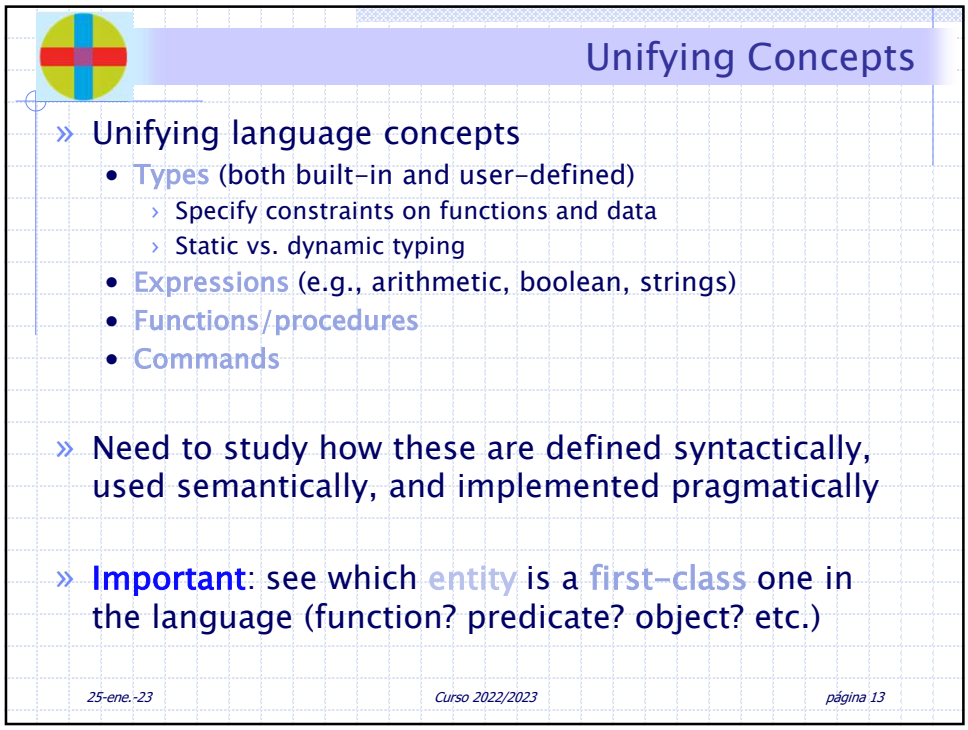
10


Concatenative and Stack-based Paradigms
» Concatenative
• Programming language in which terms correspond to functions and in which the juxtaposition of terms denotes the composition of functions
• Concatenative programming replaces function application, common in other programming styles, with function composition as the default way to build subroutines
» Stack-based
• Programming languages operate upon one or more stacks, each of which may serve different purposes
• Programming constructs in other programming languages may need to be modified for use in a stack-oriented programming language
» Examples: FORTH, Factor, PostScript, min
25-ene.-23
Curso 2022/2023
página 11

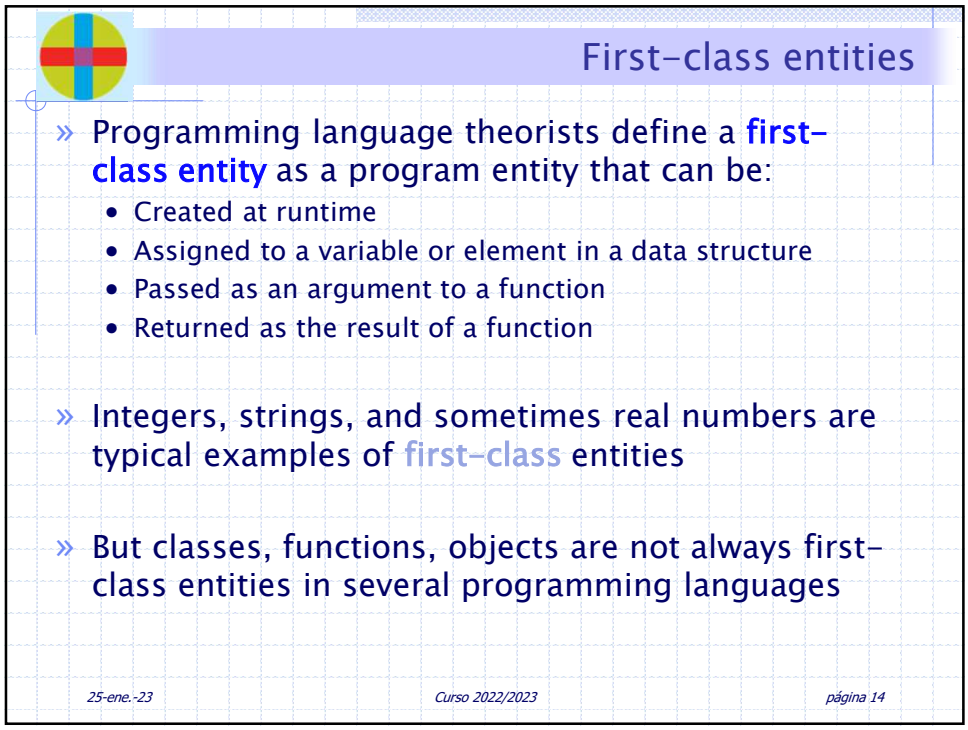
11


Concurrent and Scripting Languages
» Concurrent programming cuts across imperative, object-oriented, and functional paradigms
» Scripting is a very "high" level of programming
• Rapid development; glue together different programs
• Often dynamically typed, with only int, float, string, and array as the data types; no user-defined types
• Often weakly typed: a variable 'x' can be assigned a value of any type at any time during execution
» Very popular in Web development
• Especially scripting active Web pages
25-ene.-23
Curso 2022/2023
página 12

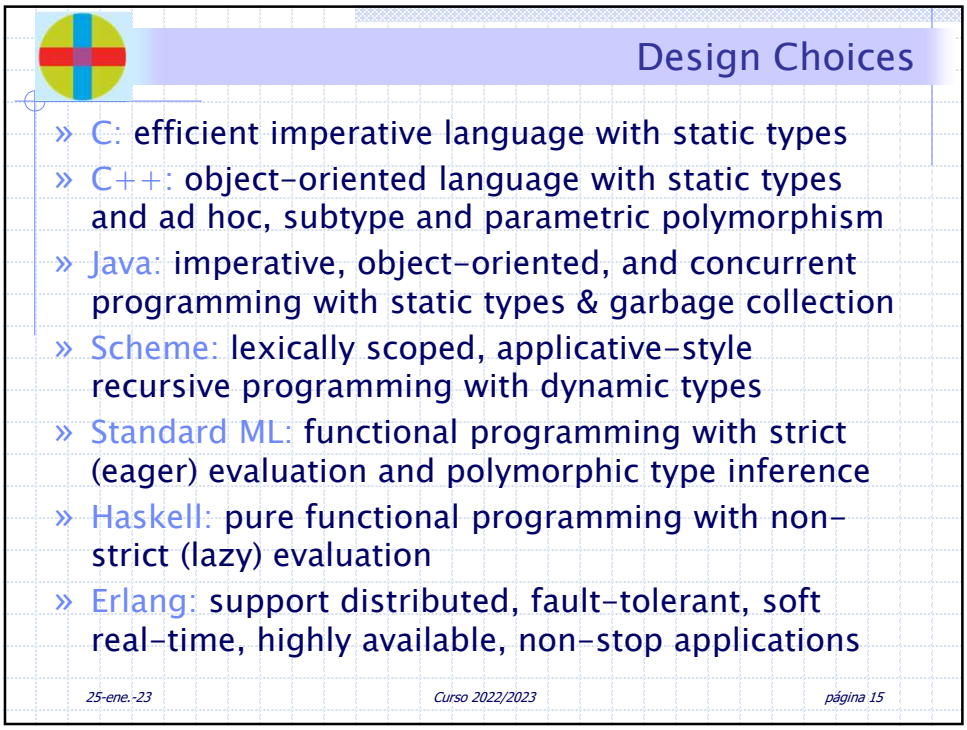
12


Unifying Concepts
» Unifying language concepts
• Types (both built-in and user-defined)
› Specify constraints on functions and data
› Static vs. dynamic typing
• Expressions (e.g., arithmetic, boolean, strings)
• Functions/procedures
• Commands
» Need to study how these are defined syntactically, used semantically, and implemented pragmatically
» Important: see which entity is a first-class one in the language (function? predicate? object? etc.)
25-ene.-23 Curso 2022/2023 página 13

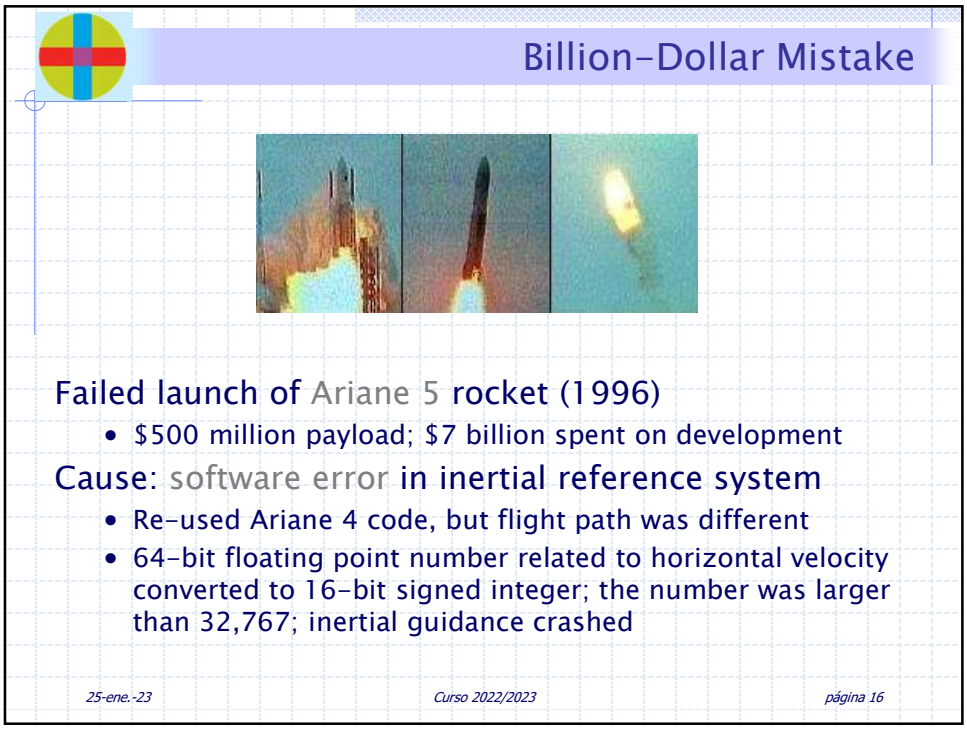
13


First-class entities
» Programming language theorists define a first-class entity as a program entity that can be:
• Created at runtime
• Assigned to a variable or element in a data structure
• Passed as an argument to a function
• Returned as the result of a function
» Integers, strings, and sometimes real numbers are typical examples of first-class entities
» But classes, functions, objects are not always first-class entities in several programming languages
25-ene.-23 Curso 2022/2023 página 14

14


Design Choices
» C: efficient imperative language with static types
» C++: object-oriented language with static types and ad hoc, subtype and parametric polymorphism
» Java: imperative, object-oriented, and concurrent programming with static types & garbage collection
» Scheme: lexically scoped, applicative-style recursive programming with dynamic types
» Standard ML: functional programming with strict (eager) evaluation and polymorphic type inference
» Haskell: pure functional programming with non-strict (lazy) evaluation
» Erlang: support distributed, fault-tolerant, soft real-time, highly available, non-stop applications
25-ene.-23 Curso 2022/2023 página 15

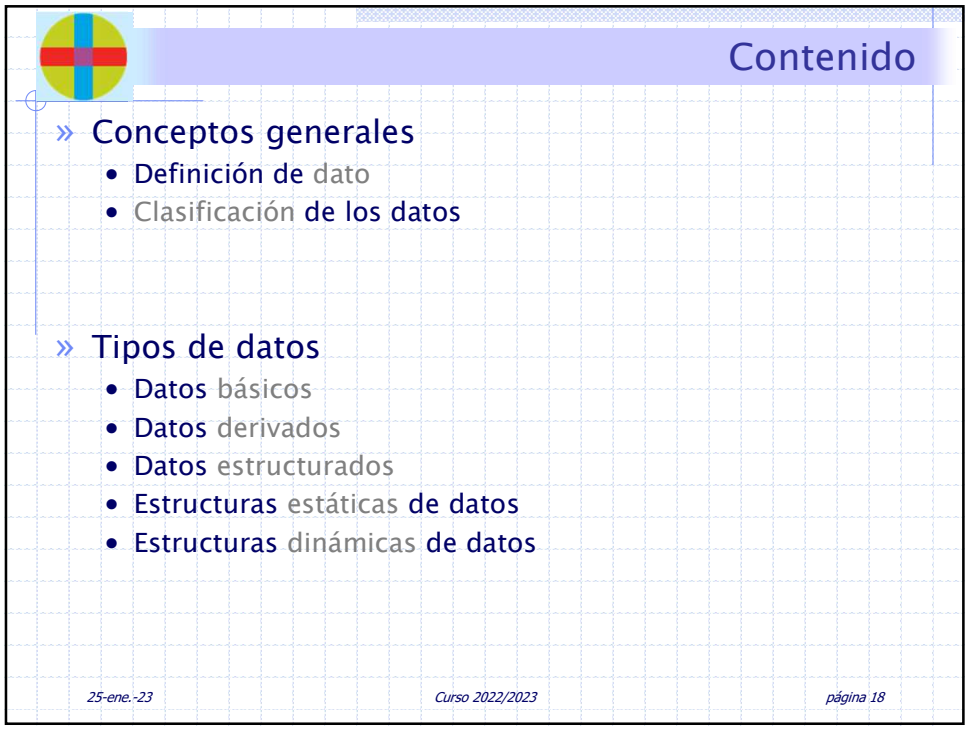
15


Billion-Dollar Mistake
Failed launch of Ariane 5 rocket (1996)
• $500 million payload; $7 billion spent on development
Cause: software error in inertial reference system
• Re-used Ariane 4 code, but flight path was different
• 64-bit floating point number related to horizontal velocity converted to 16-bit signed integer; the number was larger than 32,767; inertial guidance crashed
25-ene.-23 Curso 2022/2023 página 16

16


Estructuras de datos
Un lenguaje es más rico cuanto mejores son sus tipos de datos

17


Contenido
» Conceptos generales
• Definición de dato
• Clasificación de los datos
» Tipos de datos
• Datos básicos
• Datos derivados
• Datos estructurados
• Estructuras estáticas de datos
• Estructuras dinámicas de datos
25-ene.-23
Curso 2022/2023
página 18

18

Definición de dato

» **DATO**: Unidades de tratamiento dentro de un sistema informático. Los datos de entrada junto con los datos de salida constituyen lo que se denomina **INFORMACIÓN**.

DISEÑO DE UN PROGRAMA

1. Definir *estructuras de datos*
2. Definir *operaciones* sobre los datos.

» Componentes de un dato:

- **IDENTIFICADOR**: nombre utilizado para referenciar al dato.
- **TIPO**: rango de valores que puede tomar el dato.
- **VALOR**: elemento que pertenece al rango de valores según el tipo.

25-ene.-23 *Curso 2022/2023* *página 19*

19

Clasificación de los datos

DATOS BÁSICOS	*NUMÉRICOS*			**ENTERO REAL COMPLEJO INTERVALO DECIMAL RACIONAL**
				CARÁCTER CADENA LÓGICO ENUMERADO
DATOS DERIVADOS				**PUNTERO REFERENCIA**
DATOS ESTRUCTURADOS	*INTERNOS*	*ESTÁTICOS*	*LINEALES*	**TABLA VECTOR MATRIZ**
		DINÁMICOS	*LINEALES*	**LISTA/PILA/COLA CONJUNTO DICCIONARIO**
			NO LINEALES	**ÁRBOL GRAFO**
	EXTERNOS			**FICHERO FLUJO BASE DE DATOS**
	COMPUESTO			**REGISTRO**

25-ene.-23 *Curso 2022/2023* *página 20*

20

Datos lógicos

- Un **dato lógico** o **booleano** describe a una magnitud que tiene sólo dos valores o estados
  - verdadero–falso, si–no, etc.
- Su representación depende del lenguaje de programación:
  - `true/false, 0/1, 0/-1, V/F, ...`
- Se basan en el **álgebra de Boole**

  Operaciones:
  - Suma lógica (+): OR
  - Producto lógico (·): AND
  - Negación lógica (–): NOT

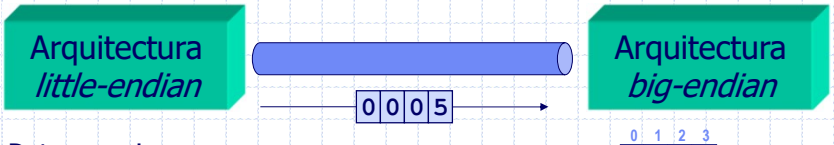
AND	0	1
0	0	0
1	0	1

OR	0	1
0	0	1
1	1	1

NOT	0	1
	1	0

Almacenamiento de datos multibyte

- ***Endianness***
  - El término inglés *endianness* ("extremidad") designa el formato en el que se almacenan los datos de más de un byte en un ordenador
  - Tipos de *endianness:*
    - *Little-endian:* Intel
    - *Big-endian:* Motorola, PowerPC, SPARC, etc.
    - *Bi-endian:* ARM, MIPS, DEC Alpha



Dato a enviar: 5

3 2 1 0
0 0 0 5

Valor: $0\times2^{24}+0\times2^{16}+0\times2^{8}+5$

0 1 2 3
0 0 0 5

Valor: $5\times2^{24}+0\times2^{16}+0\times2^{8}+0$

Dato recibido: 83.886.080

Datos numéricos (I)

» Numéricos
- Numérico entero (con y sin signo)
- Numérico real
- Otros: racional, monetario, complejo, etc.

» Numérico entero (sin signo)

- **Binario puro**: (longitud de la palabra)

  25 → 00011001

- **Decimal codificado en binario (*BCD*)**: agrupaciones de 4 bits por cada dígito decimal.

  25 → 0010 0101

Datos numéricos (II)

» Numérico entero (con signo)

- **Binario (Módulo y signo MS)**: signo en el bit más a la izquierda.

  25 → 0 0011001
  -25 → 1 0011001

  Rango de representación: $-2^{n-1}+1 \leq x \leq 2^{n-1}-1$

- **Complemento a 1 (C-1)**: igual a MS para positivos . Negativos: cambio 0s por 1s.

  25 → 0 0011001
  -25 → 1 1100110

  Rango de representación: $-2^{n-1}+1 \leq x \leq 2^{n-1}-1$.

Datos numéricos (III)

» **Numérico entero** (con signo)

- **Complemento a (C-2):** igual a MS para positivos.
  Negativos: C-1 y suma 1.

  25 → 0 0011001

  -25 → 1 1100110 + 1= 1 1100111

  Rango de representación: $-2^{n-1} \leq x \leq 2^{n-1}-1$

- **Exceso a 2^{n-1}:** No hay bit para el signo.
  Valor = número + exceso (2^{n-1}).

  25 → 128 + 25 = 153 → 10011001

  -25 → 128 - 25 = 103 → 01100111

  Rango de representación: $-2^{n-1} \leq x \leq 2^{n-1}-1$

- **Decimal desempaquetado, Decimal empaquetado, ...**

25

Datos numéricos (IV)

» **Numérico real**

- Números muy grandes o muy pequeños

» Representación:

- **Punto decimal:** dígitos del 0 al 9 más el punto decimal:

  25.32 -48759.4596 -0.41526 ...

- **Notación científica o exponencial:** "mantisa**E**característica"
  Número = mantisa * Base de exponenciación $^{\text{característica}}$
  - **Mantisa:** número real
  - **Base de exponenciación (E):** base decimal ('e').
  - **Característica:** exponente correspondiente a un número entero con su signo

26

Datos numéricos (V)

» **Numérico real** (notación científica o exponencial)

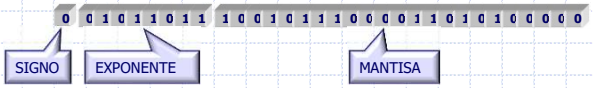
$2.5E3 \rightarrow 2.5\times10^{3}$

$0.25E-2 \rightarrow 0.25\times10^{-2}$

- Representación interna (norma IEEE 754)
  - Base: 2 ó 10.
  - Mantisa: binario puro (MS, C-1, C-2, 0.M / 1.M)
  - Característica: exceso a $2^{n-1}-1$ (ó MS)

Simple precisión (32 bits):
- **Signo**: 1 bit (pos. 31)
- **Exponente**: 8 bits (pos. 23 a 30)
- **Mantisa**: 23 bits (pos. 0 a 22)

Doble precisión (64 bits):
- **Signo**: 1 bit (pos. 63)
- **Exponente**: 11 bits (pos. 52 a 62)
- **Mantisa**: 52 bits (pos. 0 a 51)



27

La norma IEEE 754: valores normalizados

» **Valor V** de un **número binario IEEE 754** de **exponente E** y **mantisa M**:

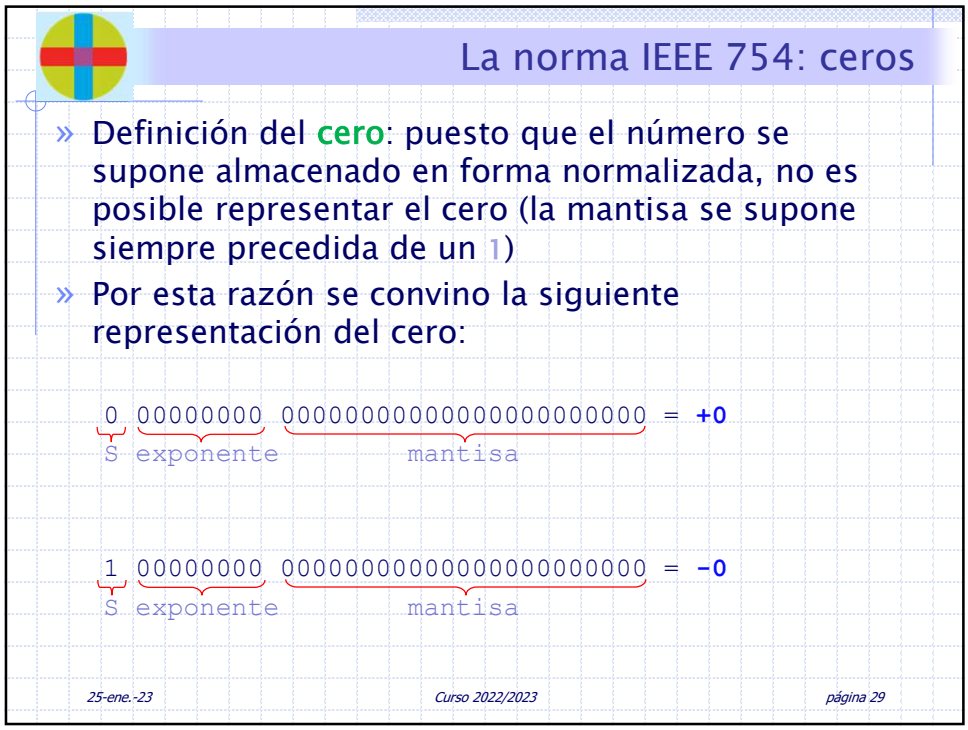
- El **exponente E** es un número binario representado en exceso a n (n depende de la precisión del número)
- La **mantisa M** representa una fracción binaria en notación normalizada (hay que sumar una unidad ya que se asume que el primer bit de la mantisa, potencia 0, vale 1)
  - Por ejemplo, si el contenido de la mantisa es M=0.254 el valor de la mantisa es 1+0.254
- Ejemplo (precisión simple):

0 01111111 11010000000000000000000 = **+1.8125**
S exponente mantisa

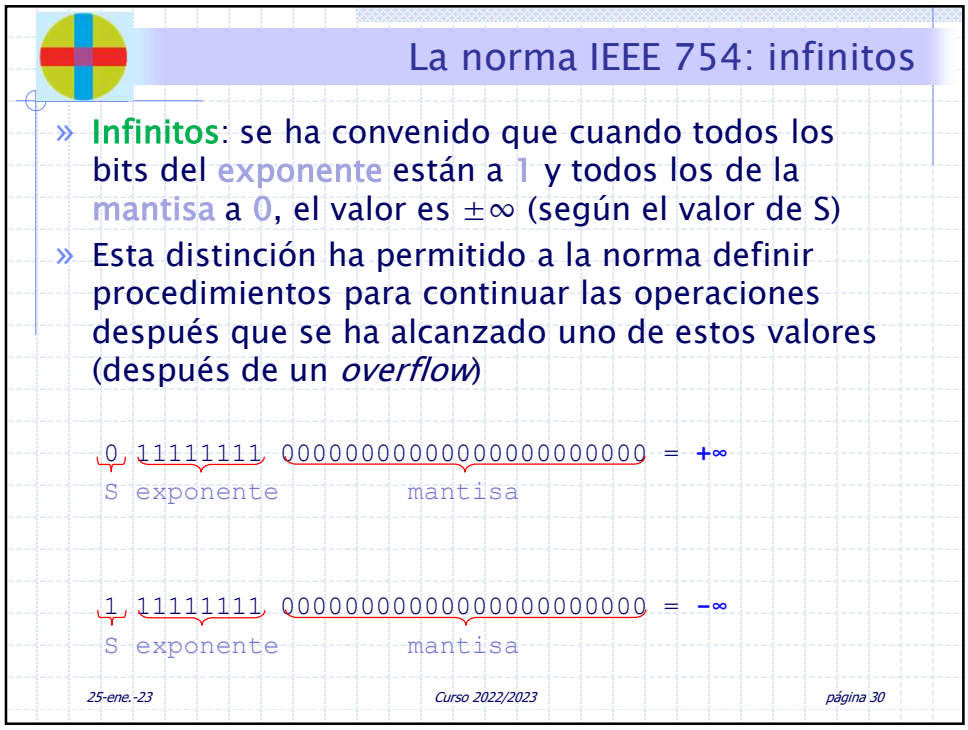
S = 0 (**+**) E = 127 → E − 127 = **0**

M = $1\times2^{-1} + 1\times2^{-2} + 0\times2^{-3} + 1\times2^{-4} + \ldots$ = **0.8125**

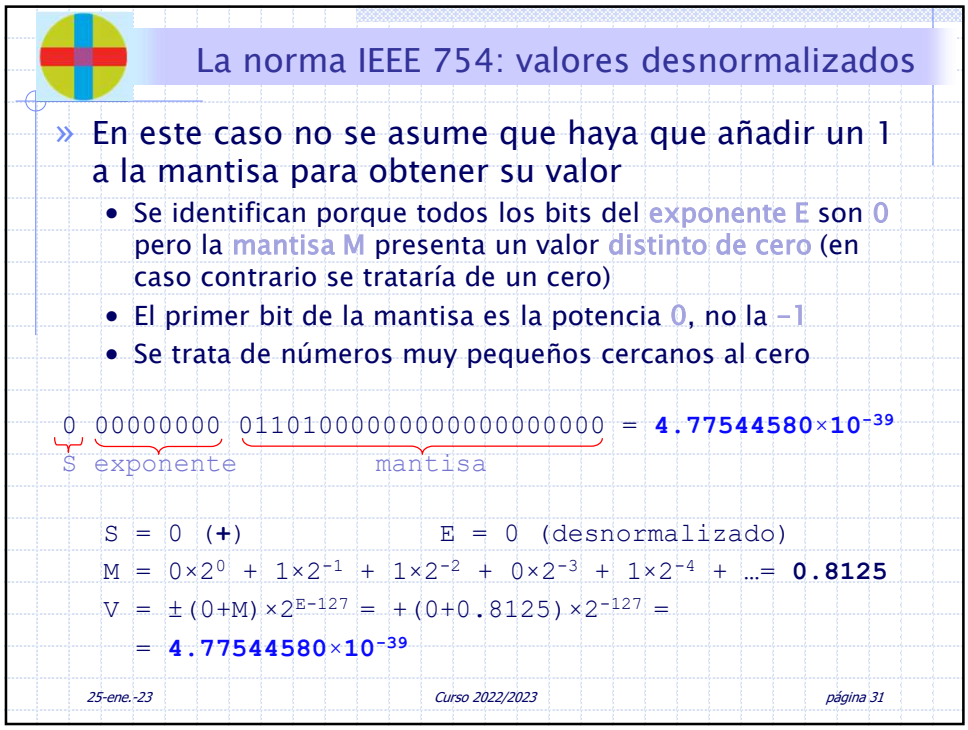
28


La norma IEEE 754: ceros
» Definición del cero: puesto que el número se supone almacenado en forma normalizada, no es posible representar el cero (la mantisa se supone siempre precedida de un 1)
» Por esta razón se convino la siguiente representación del cero:
0 00000000 00000000000000000000000 = +0
S exponente mantisa
1 00000000 00000000000000000000000 = -0
S exponente mantisa
25-ene.-23 Curso 2022/2023 página 29

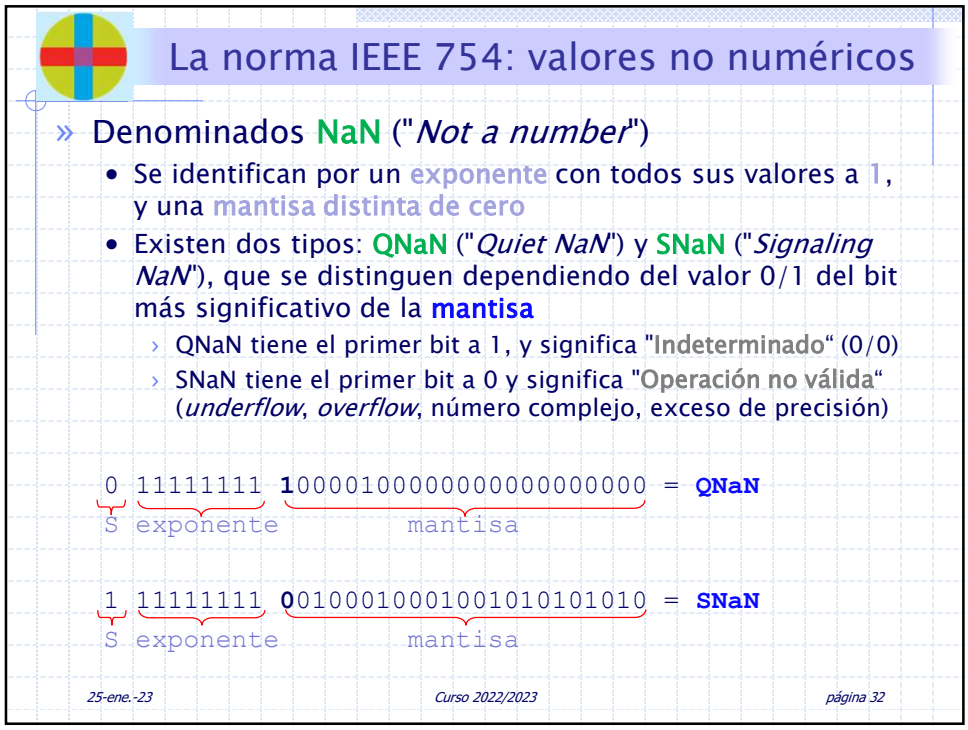
29


La norma IEEE 754: infinitos
» Infinitos: se ha convenido que cuando todos los bits del exponente están a 1 y todos los de la mantisa a 0, el valor es ±∞ (según el valor de S)
» Esta distinción ha permitido a la norma definir procedimientos para continuar las operaciones después que se ha alcanzado uno de estos valores (después de un overflow)
0 11111111 00000000000000000000000 = +∞
S exponente mantisa
1 11111111 00000000000000000000000 = -∞
S exponente mantisa
25-ene.-23 Curso 2022/2023 página 30

30


La norma IEEE 754: valores desnormalizados
» En este caso no se asume que haya que añadir un 1 a la mantisa para obtener su valor
• Se identifican porque todos los bits del exponente E son 0 pero la mantisa M presenta un valor distinto de cero (en caso contrario se trataría de un cero)
• El primer bit de la mantisa es la potencia 0, no la -1
• Se trata de números muy pequeños cercanos al cero
0 00000000 01101000000000000000000 = 4.77544580×10-39
S exponente mantisa
S = 0 (+) E = 0 (desnormalizado)
M = 0×20 + 1×2-1 + 1×2-2 + 0×2-3 + 1×2-4 + …= 0.8125
V = ±(0+M)×2E-127 = +(0+0.8125)×2-127 =
= 4.77544580×10-39
25-ene.-23 Curso 2022/2023 página 31

31


La norma IEEE 754: valores no numéricos
» Denominados NaN ("Not a number")
• Se identifican por un exponente con todos sus valores a 1, y una mantisa distinta de cero
• Existen dos tipos: QNaN ("Quiet NaN") y SNaN ("Signaling NaN"), que se distinguen dependiendo del valor 0/1 del bit más significativo de la mantisa
› QNaN tiene el primer bit a 1, y significa "Indeterminado" (0/0)
› SNaN tiene el primer bit a 0 y significa "Operación no válida" (underflow, overflow, número complejo, exceso de precisión)
0 11111111 10000100000000000000000 = QNaN
S exponente mantisa
1 11111111 00100010001001010101010 = SNaN
S exponente mantisa
25-ene.-23 Curso 2022/2023 página 32

32

La norma IEEE 754: cálculo de valores

» El **valor decimal** V de una representación binaria estándar, puede calcularse mediante las siguientes fórmulas:

$V = \pm(1+M)\times 2^{E-127}$ *simple precisión normalizado*

$V = \pm(0+M)\times 2^{E-127}$ *simple precisión no normalizado*

$V = \pm(1+M)\times 2^{E-1023}$ *doble precisión normalizado*

$V = \pm(0+M)\times 2^{E-1023}$ *doble precisión no normalizado*

1 10001101 01101000000000000000000 = **-23040**
S exponente mantisa

S = 1 (**-**) E = 141 → E − 127 = **14**

$M = 0\times 2^{-1} + 1\times 2^{-2} + 1\times 2^{-3} + 0\times 2^{-4} + 1\times 2^{-4} \ldots =$ **0.40625**

$V = \pm(1+M)\times 2^{E-127} = -(1+0.40625)\times 2^{14} =$ **-23040**

33

La norma IEEE 754: rangos

» Rangos de representación de la norma IEEE 754:

IEEE 754	Precisión simple	Precisión doble	Precisión cuádruple
nº de bits	32	64	128
signo	1	1	1
exponente	8	11	15
mantisa	23	52	112
exceso del exponente	127	1023	262143
valor más pequeño desnorm.	$2^{-126-23} = 2^{-149} \approx 1.5\times 10^{-45}$	$2^{-1022-52} = 2^{-1074} \approx 5\times 10^{-324}$	$2^{-16493} \approx 10^{-4965}$
valor más pequeño	$2^{-126} \approx 1\times 10^{-38}$	$2^{-1022} \approx 2\times 10^{-308}$	$2^{-16382} \approx 3\times 10^{-4932}$
valor más grande	$(1+(1-2^{-23}))\times 2^{127} \approx 3.4\times 10^{38}$	$(1+(1-2^{-52}))\times 2^{1023} \approx 2\times 10^{308}$	$2^{16384} - 2^{16272} \approx 1\times 10^{4932}$
precisión	23 bits, $\pm 2^{-150}$	52 bits, $\pm 2^{-1075}$	112 bits, $\pm 2^{-16494}$
en dígitos$_{10}$	7	15	33

34

Datos de carácter

- Utilizados para representar un carácter dentro del juego de caracteres definido por el fabricante del ordenador
- Juegos de caracteres clásicos:
  - ASCII (*American Standard Code for Information Interchange*)
    - 7 bits (128 caracteres)
  - EBCDIC (*Extended Binary Coded Decimal Interchange Code*)
    - 8 bits (256 caracteres)
- UNICODE (www.unicode.org):
  - Representa caracteres (*characters*) mediante números (*code points*) que se dibujan mediante ideogramas o glifos (*glyphs*)
  - Puede representar hasta 2097152 (2^{21}) elementos
  - Todos los lenguajes del mundo están incluidos
  - Requiere codificación para los *code points*
    - La más usada y conocida es UTF-8

Tabla de códigos ASCII

- El código ASCII es un código de 7 bits que se suele almacenar en 8 bits.
  - Obsérvese que una letra minúscula es el código ASCII de la mayúscula más 20 hexadecimal (32 decimal)

00	NUL	10	DLE	20	SP	30	0	40	@	50	P	60	`	70	p
01	SOH	11	DC1	21	!	31	1	41	A	51	Q	61	a	71	q
02	STX	12	DC2	22	"	32	2	42	B	52	R	62	b	72	r
03	ETX	13	DC3	23	#	33	3	43	C	53	S	63	c	73	s
04	EOT	14	DC4	24	$	34	4	44	D	54	T	64	d	74	t
05	ENQ	15	NAK	25	%	35	5	45	E	55	U	65	e	75	u
06	ACK	16	SYN	26	&	36	6	46	F	56	V	66	f	76	v
07	BEL	17	ETB	27	'	37	7	47	G	57	W	67	g	77	w
08	BS	18	CAN	28	(	38	8	48	H	58	X	68	h	78	x
09	HT	19	EM	29	)	39	9	49	I	59	Y	69	i	79	y
0A	LF	1A	SUB	2A	*	3A	:	4A	J	5A	Z	6A	j	7A	z
0B	VT	1B	ESC	2B	+	3B	;	4B	K	5B	[	6B	k	7B	{
0C	FF	1C	FS	2C	´	3C	<	4C	L	5C	\	6C	l	7C	|
0D	CR	1D	GS	2D	-	3D	=	4D	M	5D	]	6D	m	7D	}
0E	SO	1E	RS	2E	.	3E	>	4E	N	5E	^	6E	n	7E	~
0F	SI	1F	US	2F	/	3F	?	4F	O	5F	_	6F	o	7F	DEL

NUL	Null	FF	Form feed	CAN	Cancel
SOH	Start of heading	CR	Carriage return	EM	End of medium
STX	Start of text	SO	Shift out	SUB	Substitute
ETX	End of text	SI	Shift in	ESC	Escape
EOT	End of transmission	DLE	Data link escape	FS	File separator
ENQ	Enquiry	DC1	Device control 1	GS	Group separator
ACK	Acknowledge	DC2	Device control 2	RS	Record separator
BEL	Bell	DC3	Device control 3	US	Unit separator
BS	Backspace	DC4	Device control 4	SP	Space
HT	Horizontal tab	NAK	Negative acknowledge	DEL	Delete
LF	Line feed	SYN	Synchronous idle		
VT	Vertical tab	ETB	End of transmission block		

Tabla de códigos EBCDIC

- El código EBCDIC es un código de 8 bits

00	NUL	20	DS	40	SP	60	–	80		A0		C0	{	E0	\
01	SOH	21	SOS	41		61	/	81	a	A1	~	C1	A	E1	
02	STX	22	FS	42		62		82	b	A2	s	C2	B	E2	S
03	ETX	23		43		63		83	c	A3	t	C3	C	E3	T
04	PF	24	BYP	44		64		84	d	A4	u	C4	D	E4	U
05	HT	25	LF	45		65		85	e	A5	v	C5	E	E5	V
06	LC	26	ETB	46		66		86	f	A6	w	C6	F	E6	W
07	DEL	27	ESC	47		67		87	g	A7	x	C7	G	E7	X
08		28		48		68		88	h	A8	y	C8	H	E8	Y
09		29		49		69		89	i	A9	z	C9	I	E9	Z
0A	SMM	2A	SM	4A	¢	6A	‘	8A		AA		CA		EA	
0B	VT	2B	CU2	4B		6B	,	8B		AB		CB		EB	
0C	FF	2C		4C	<	6C	%	8C		AC		CC		EC	
0D	CR	2D	ENQ	4D	(	6D	_	8D		AD		CD		ED	
0E	SO	2E	ACK	4E	+	6E	>	8E		AE		CE		EE	
0F	SI	2F	BEL	4F	|	6F	?	8F		AF		CF		EF	
10	DLE	30		50	&	70		90		B0		D0	}	F0	0
11	DC1	31		51		71		91	j	B1		D1	J	F1	1
12	DC2	32	SYN	52		72		92	k	B2		D2	K	F2	2
13	TM	33		53		73		93	l	B3		D3	L	F3	3
14	RES	34	PN	54		74		94	m	B4		D4	M	F4	4
15	NL	35	RS	55		75		95	n	B5		D5	N	F5	5
16	BS	36	UC	56		76		96	o	B6		D6	O	F6	6
17	IL	37	EOT	57		77		97	p	B7		D7	P	F7	7
18	CAN	38		58		78		98	q	B8		D8	Q	F8	8
19	EM	39		59		79		99	r	B9		D9	R	F9	9
1A	CC	3A		5A	!	7A	:	9A		BA		DA		FA	|
1B	CU1	3B	CU3	5B	$	7B	#	9B		BB		DB		FB	
1C	IFS	3C	DC4	5C	·	7C	@	9C		BC		DC		FC	
1D	IGS	3D	NAK	5D	)	7D	'	9D		BD		DD		FD	
1E	IRS	3E		5E	;	7E	=	9E		BE		DE		FE	
1F	IUS	3F	SUB	5F	¬	7F	"	9F		BF		DF		FF	

STX	Start of text	RS	Reader Stop	DC
DLE	Data Link Escape	PF	Punch Off	DC
BS	Backspace	DS	Digit Select	DC
ACK	Acknowledge	PN	Punch On	CU
SOH	Start of Heading	SM	Set Mode	CU
ENQ	Enquiry	LC	Lower Case	CU
ESC	Escape	CC	Cursor Control	SY
BYP	Bypass	CR	Carriage Return	IF
CAN	Cancel	EM	End of Medium	EC
RES	Restore	FF	Form Feed	ET
SI	Shift In	TM	Tape Mark	NA
SO	Shift Out	UC	Upper Case	SM
DEL	Delete	FS	Field Separator	SC
SUB	Substitute	HT	Horizontal Tab	IG
NL	New Line	VT	Vertical Tab	IR
LF	Line Feed	UC	Upper Case	IU

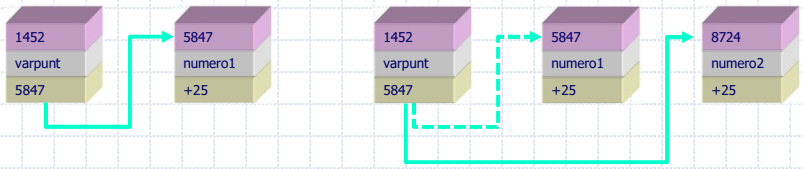
37

Unicode

38

Tipos de datos: datos derivados

- Se emplean para referenciar otro dato (de otro tipo) mediante algún medio, por ejemplo usando la dirección de memoria del otro dato (puntero)
- Características de los datos de tipo puntero:
  - **Tipo**: Mismo tipo que el del dato al que apunta.
  - **Contenido**: la dirección del dato al que apunta.
  - **Indirección**: referencias indirectas al dato al que apunta.
  - **Reutilización**: se puede utilizar para apuntar a otro dato del mismo tipo que el puntero.



39

Tipos de datos: datos estructurados

- **Internos**: residen en la memoria principal del ordenador
- **Externos**: residen en la memoria auxiliar del ordenador (dispositivos de almacenamiento masivo)

- **Estáticos**: su tamaño no puede cambiar durante la ejecución del programa
- **Dinámicos**: su tamaño puede cambiar durante la ejecución del programa

- **Lineales**: sólo pueden estar enlazados a un elemento anterior y uno posterior
- **No lineales**: pueden enlazarse a más de un elemento, de diferentes formas

- **Compuestos**: formados por el programador mediante agregaciones de datos básicos y derivados

40

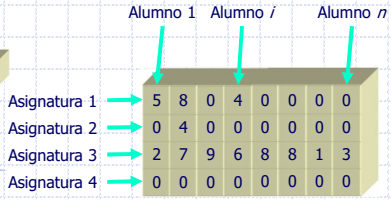
Tipos de datos: estructuras estáticas

TABLAS

» Estructura estática de datos constituida por un número fijo de **elementos** del mismo tipo en direcciones de memoria contiguas

» Clasificación:

- Unidimensionales (**vectores**)
- Bidimensionales (**matrices**)
- Multidimensionales (**poliedros**)



Tipos de datos: estructuras dinámicas (I)

DATOS ESTRUCTURADOS	INTERNOS	DINÁMICOS	LINEALES	LISTA/PILA/COLA CONJUNTO DICCIONARIO
			NO LINEALES	ÁRBOL GRAFO

» **LISTA**: estructura dinámica de datos constituida por elementos (habitualmente, pero no necesariamente, del mismo tipo) denominados **nodos**. El orden de los elementos es significativo.

- **PILA**: lista basada en el criterio **LIFO** (*last in first out*, el último que entra es el primero que sale). Los elementos se añaden apilándose unos sobre otros y sólo se tiene acceso al que está en la parte superior.
- **COLA**: lista basada en el criterio **FIFO** (*first in first out*, el primero que entra es el primero que sale). Los nuevos elementos se añaden en la parte de atrás y la extracción de elementos se realiza por la de delante.

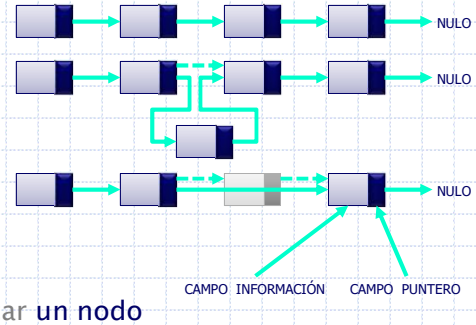
» **CONJUNTO**: estructura dinámica constituida por **elementos** (habitualmente, pero no necesariamente, del mismo tipo). Los elementos no están ordenados y sólo es significativa su pertenencia o no al conjunto y la posibilidad de que estén repetidos o no.

» **DICCIONARIO** o **MAPA**: estructura dinámica de datos formada por una serie de parejas (**clave** → **valor**). El orden de las parejas puede ser significativo o no, así como la repetición de claves.

Tipos de datos: estructuras dinámicas (II)

LISTAS

» Tipos de listas:
- Contiguas
- Simplemente enlazadas
- Doblemente enlazadas
- Circulares



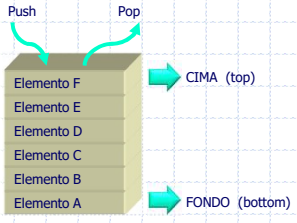
» Operaciones:
- Inicializar lista
- Insertar, eliminar o buscar un nodo
- Copiar, borrar la lista completa
- Subdividir o concatenar listas

Tipos de datos: estructuras dinámicas (III)

PILAS

» Utilizado en compiladores e intérpretes

» Operaciones:
- Inicializar pila
- Apilar (*push*) o desapilar (*pop*) un elemento
- Pila llena
- Pila vacía
- Top

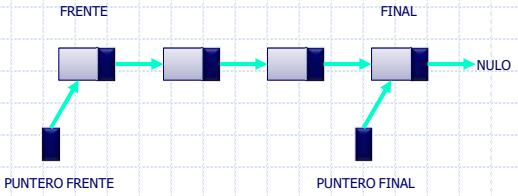


Tipos de datos: estructuras dinámicas (IV)

COLAS

» Elementos interiores inaccesibles

» Operaciones:
- Inicializar cola
- Encolar (*enqueue*)
- Desencolar (*dequeue*)
- Cola llena
- Cola vacía



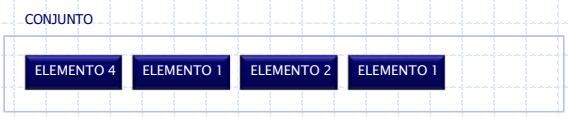
45

Tipos de datos: estructuras dinámicas (V)

CONJUNTOS

» Elementos no ordenados dentro del conjunto

» Operaciones:
- Inicializar conjunto
- Insertar (*insert*) o eliminar (*delete*) un elemento
- Pertenencia de un elemento al conjunto
- Número de repeticiones de un elemento
- Conjunto vacío
- Cardinalidad



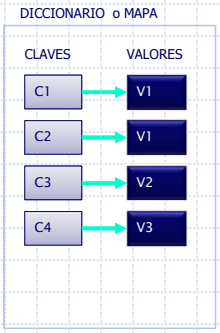
46

Tipos de datos: estructuras dinámicas (VI)

DICCIONARIOS o MAPAS

» Asocian valores a claves

» Operaciones:
- Inicializar mapa
- Insertar pareja (clave, valor)
- Eliminar pareja (clave, valor)
- Lista de claves
- Lista de valores
- Valor de una clave
- Pertenencia de una clave

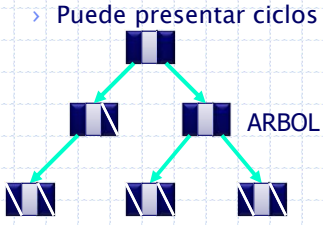


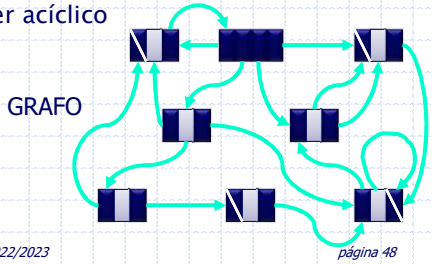
47

Tipos de datos: estructuras dinámicas (VII)

ÁRBOL – representación de estructuras jerarquizadas
- Características:
  - Un antecesor
  - Varios sucesores
  - Un único elemento llamado raíz (sin antecesor)

GRAFO – conjunto de nodos conectados por arcos
- Características:
  - Los arcos pueden tener sentido (dirigido) o no
  - Puede presentar ciclos o ser acíclico



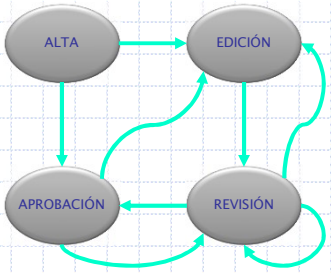


48

Ejemplo de grafos

- Supongamos que estamos diseñando un programa de control de versión de documentos. Durante el análisis se recoge la siguiente información:
- Es posible dar de alta un documento.
- Una vez dado de alta se puede pasar a editar el documento. Si ya existe el documento se puede pasar directamente a aprobación.
- Una vez editado, se puede pasar a revisión.
- De revisión puede ser enviado de nuevo a edición, o dejar pendiente de revisión de nuevo. Otra posibilidad es que se proceda a la aprobación del documento.
- Una vez que está aprobado, el paso del tiempo puede dar lugar a inconsistencias en el contenido del mismo por lo que es posible que se envíe a revisar o directamente a edición.

	ALTA	EDICIÓN	REVISIÓN	APROBACIÓN
ALTA		X		X
EDICIÓN			X	
REVISIÓN		X	X	X
APROBACIÓN		X	X	



25-ene.-23 *Curso 2022/2023* *página 49*

49

Propiedades de los tipos de datos

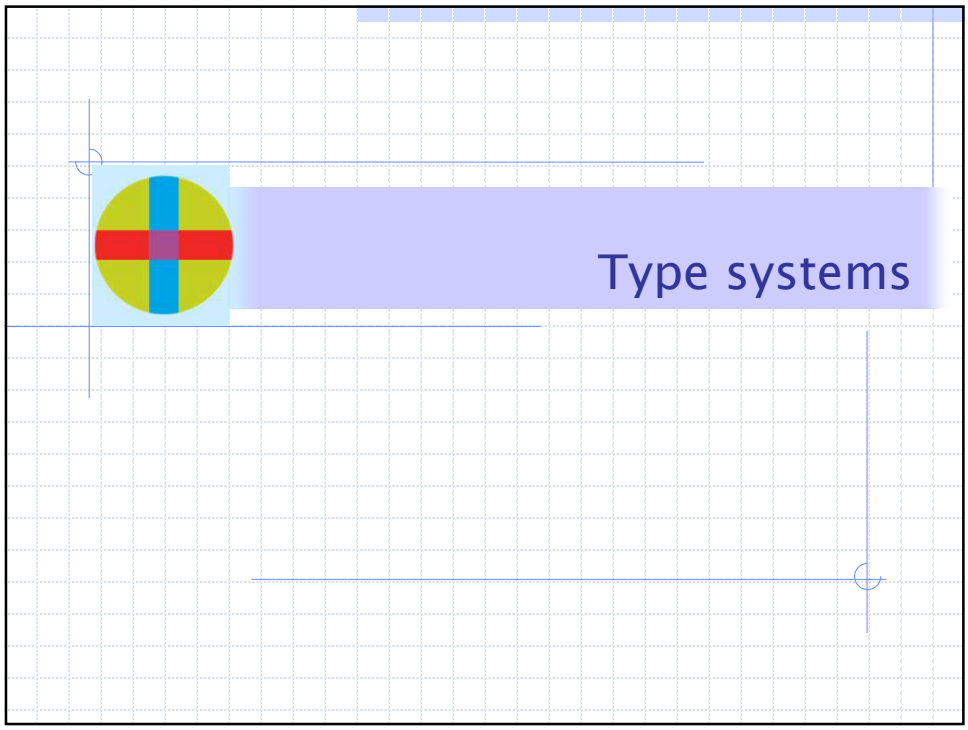
- Propiedades de los tipos de datos compuestos:

Propiedad	Tabla	Vector	Lista (pila / cola)	Conjunto	Diccionario	Árbol	Grafo
tamaño	fijo	estático	dinámico	dinámico	dinámico	dinámico	dinámico
orden	sí	sí	sí	no	no	complejo	complejo
homogéneo	sí	sí	no (sí)	no (sí)	¿claves?	no	no
acceso secuencial (recorrido)	no	sí	variado (no)	sí	claves, valores y parejas	múltiple	complejo
acceso aleatorio	sí	sí	complejo	sí	sí	no	no
inserción	no	complejo	sí (fin/fin)	sí	sí	caro	sí
borrado	no	complejo	sí (ini/fin)	sí	sí	caro	sí
sustitución	sí	sí	sí	no	sí	sí	sí
búsqueda	no	sí	lineal	perte-nencia	clave → valor	rápida	compleja
concatenar	no	compleja	sí	sí	sí	complejo	complejo
operación básica	acceso aleatorio	acceso aleatorio	recorrido	perte-nencia	búsqueda	recorrido	camino

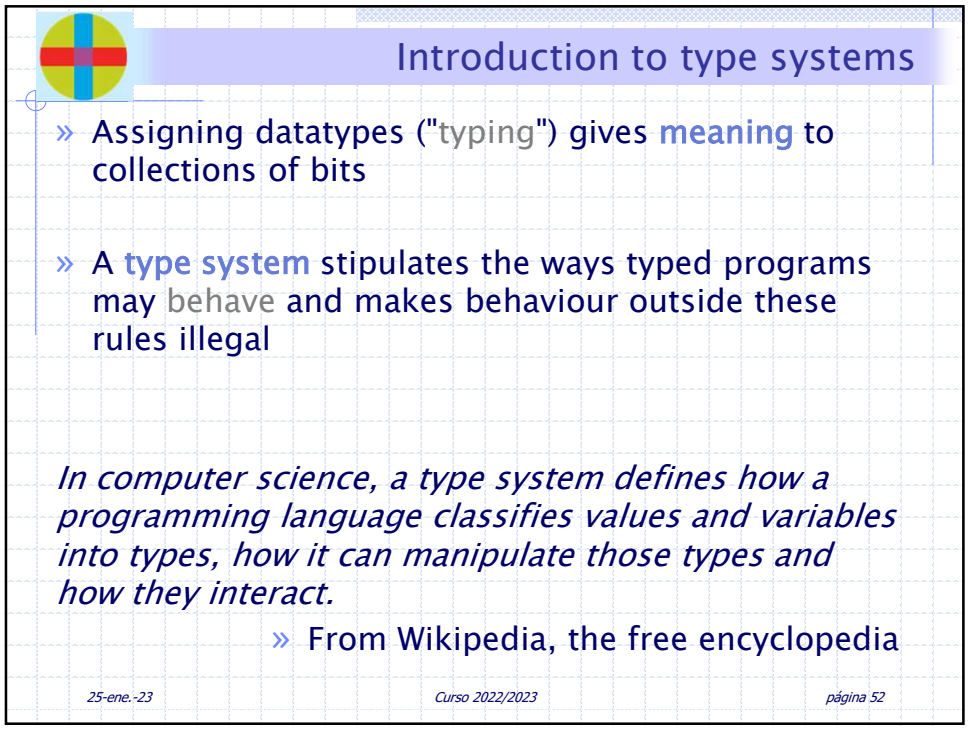
- **Cadena**: normalmente tabla o vector de caracteres

25-ene.-23 *Curso 2022/2023* *página 50*

50

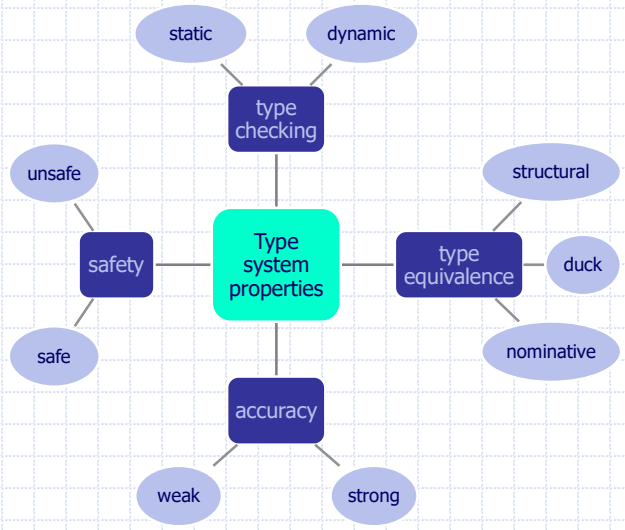

Type systems

51


Introduction to type systems
» Assigning datatypes ("typing") gives meaning to collections of bits
» A type system stipulates the ways typed programs may behave and makes behaviour outside these rules illegal
In computer science, a type system defines how a programming language classifies values and variables into types, how it can manipulate those types and how they interact.
» From Wikipedia, the free encyclopedia
25-ene.-23
Curso 2022/2023
página 52

52

Properties of type systems

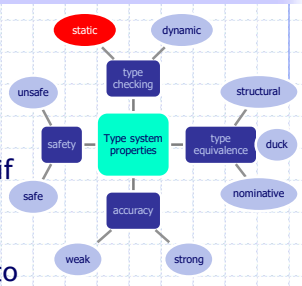


53

Static type checking

» Every name is bound both to a type (at compile time, via a data declaration) to an object

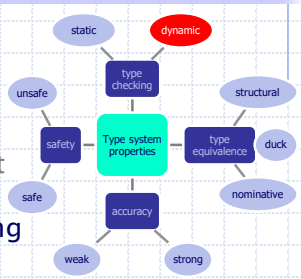
- The binding to an object is optional — if a name is not bound to an object, the name is said to be *null*
- Once a variable name has been bound to a type (declared) it can be bound (via an assignment statement) only to objects of that type; it cannot ever be bound to an object of a different type. An attempt to bind the name to an object of the wrong type will raise a type exception
- All checks are done in compiling time
- Can only check if the data processing is right but not the data itself



54

Dynamic type checking

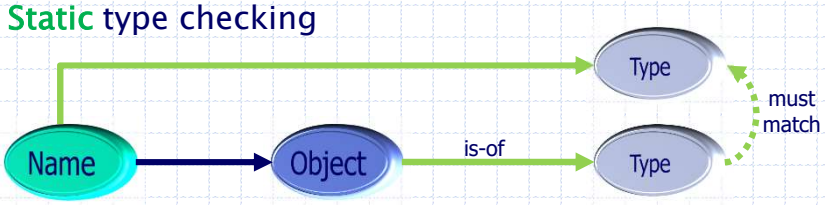
- Every name is (unless it is *null*) bound only to an object
  - Names are bound to the objects at execution time by means of assignment statements, and it is possible to bind a name to objects of different types during the execution of the program
  - Type errors not detected until a piece of code is executed
  - Allows more kinds of code constructs (expression evaluation, introspection)
  - Shortens the edit-compile-test-debug cycle (REPL: Read-Eval-Print Loop)
  - Makes *metaprogramming* more powerful and easier to use



55

Static *versus* Dynamic type checking

- Static type checking



- Dynamic type checking

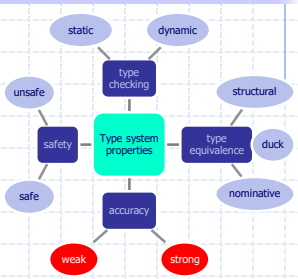
56

Static *versus* Dynamic type checking

- Benefits of **static** type checking
  - Errors / Warnings in your editor (now also on dynamic)
  - Generics (but this can be also a negative)
  - Ability to follow a chain and figure out what type of object is required at each step
  - Refactoring
  - Common in compiled languages, considered "safer"
- Benefits of **dynamic** type checking
  - More concise and precise, less typing
  - No badly implemented generics
  - Closures / Functions
  - REPL: Read-Eval-Print Loop
  - Common in interpreted languages

Strong / Weak typing

- A **strongly typed** language doesn't allow an operation to succeed on arguments which have the wrong type
- A **weakly typed** language does




```
int x = 5, y;
float c = 24.6;

y = x + c;
```

Example

```
y = 29
```

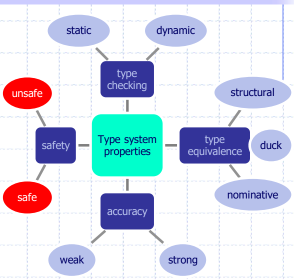
Result in C

```
java.lang.Error:
Unresolved compilation problems:
Type mismatch:
cannot convert from float to int
```

Result in Java

Safe / Unsafe typing

- A language is type-safe if it does not allow operations or conversions which lead to erroneous conditions
- A language is type-unsafe if allows these conversions (type coercion)




```
var x = 5;
var y = "hi";
var z = x + y;
```

Example in Visual Basic

```
int x = 5;
char y[] = "hi";
char* z = x + y;
```

Example in C

```
z = "5hi";
```

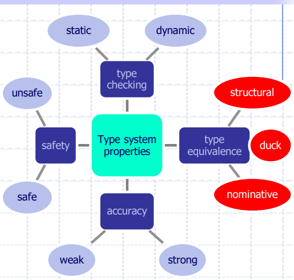
Result in Visual Basic

```
z = "   l @☻\ @☻E¯┴w "
```

Result in C

Type equivalence

- A type system has always an own definition of equivalence
- Two extreme cases are
  - Nominative type systems
    - (e.g. Java, C, Fortran)
    - Types must have the same name in order to be equal
  - Structural type systems
    - (e.g. Haskell, Lisp)
    - Types must have the same structure in order to be equal
- A special form of dynamic typing is "duck typing"
  - The language "guesses" which type is meant (e.g. Perl, Python, Ruby, Smalltalk)

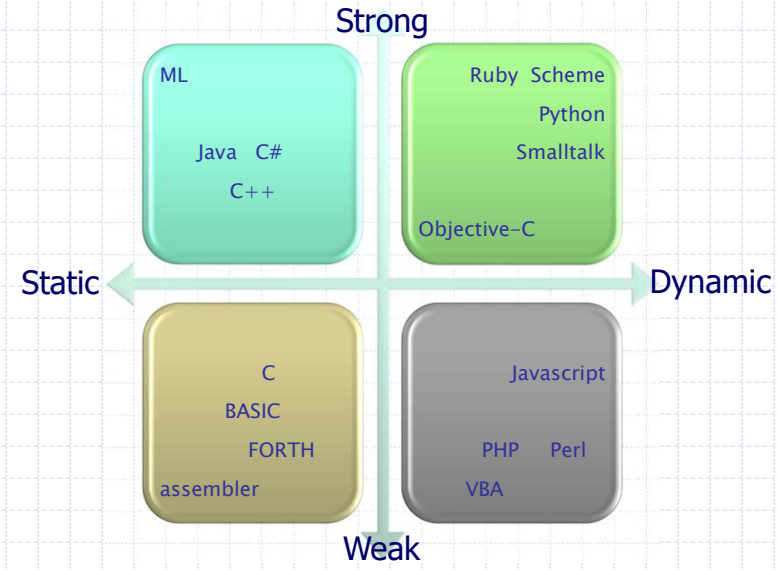


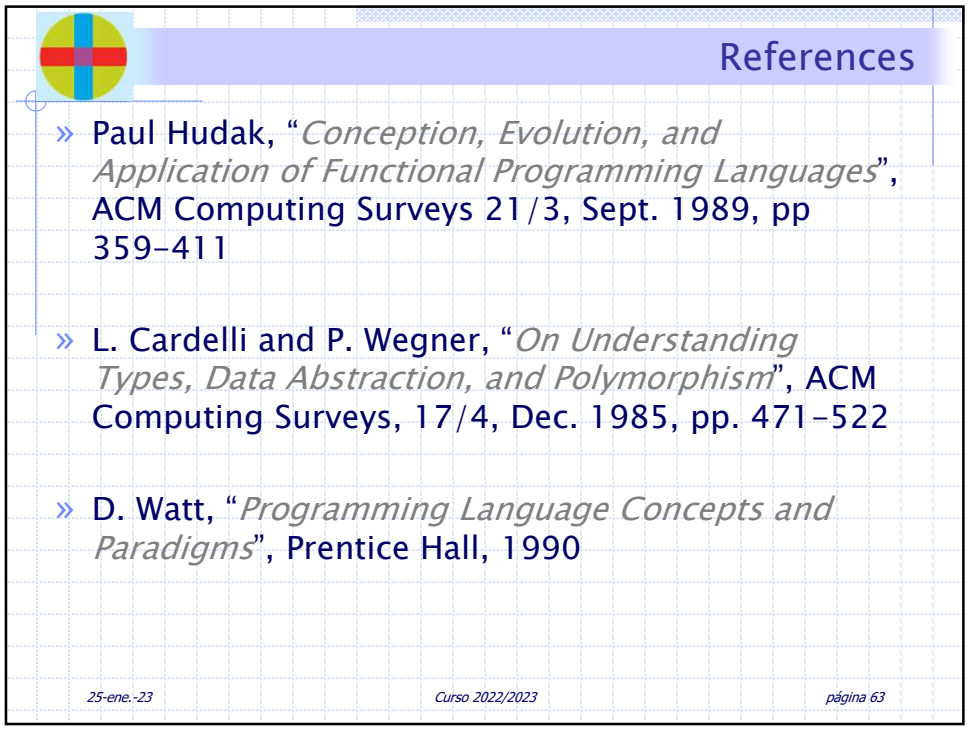
Programming languages & type systems

Language	type checking	accuracy	safety	equivalence
Ada	static	strong	safe	nominative
assembler	none	weak	unsafe	structural
BASIC	static	weak	safe	nominative
C	static	weak	unsafe	nominative
C++	static	strong	unsafe	nominative
C#	static	strong	both	nominative
Eiffel	static	strong	safe	nominative
Erlang	dynamic	strong	safe	nominative
FORTH	none	weak	unsafe	structural
Haskell	static	strong	safe	nominative
Java	static	strong	safe	nominative
Javascript	dynamic	weak	safe	duck
Lisp	dynamic	strong	safe	structural
ML	static	strong	safe	structural
Objective-C	dynamic	strong	unsafe	duck
Pascal	static	strong	safe	nominative
Perl	dynamic	weak	safe	nominative
PHP	dynamic	weak	safe	nominative
Python	dynamic	strong	safe	duck
Ruby	dynamic	strong	safe	duck
Scheme	dynamic	strong	safe	nominative
Smalltalk	dynamic	strong	safe	duck

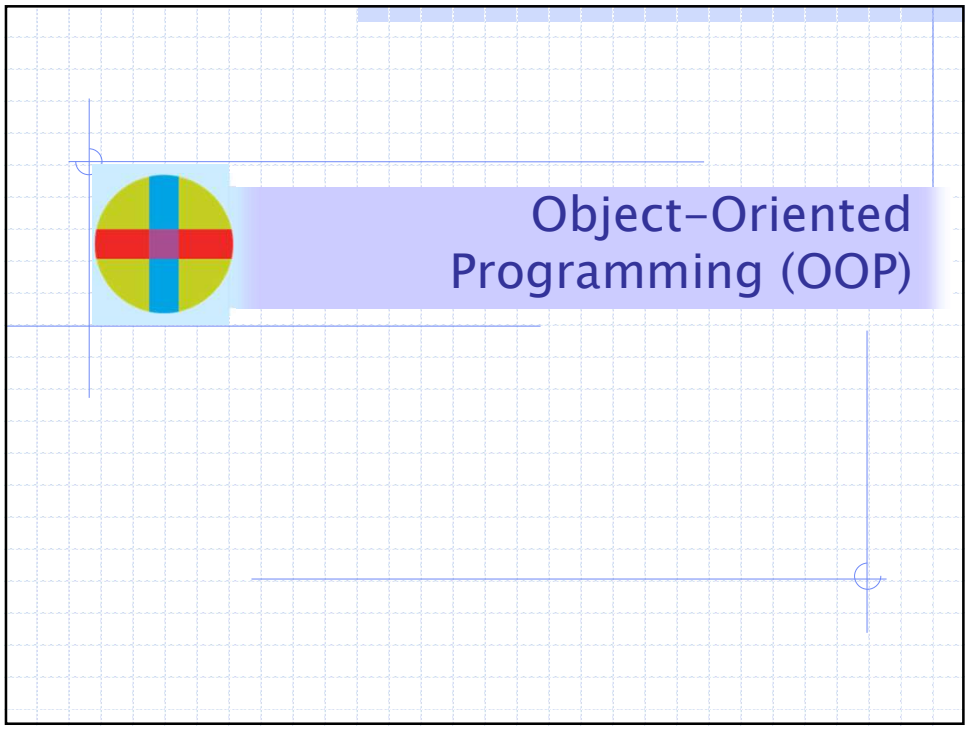
Source: http://wikipedia.org/wiki/Template:Type_system_cross_reference_list

The type matrix (strength and dynamicity)

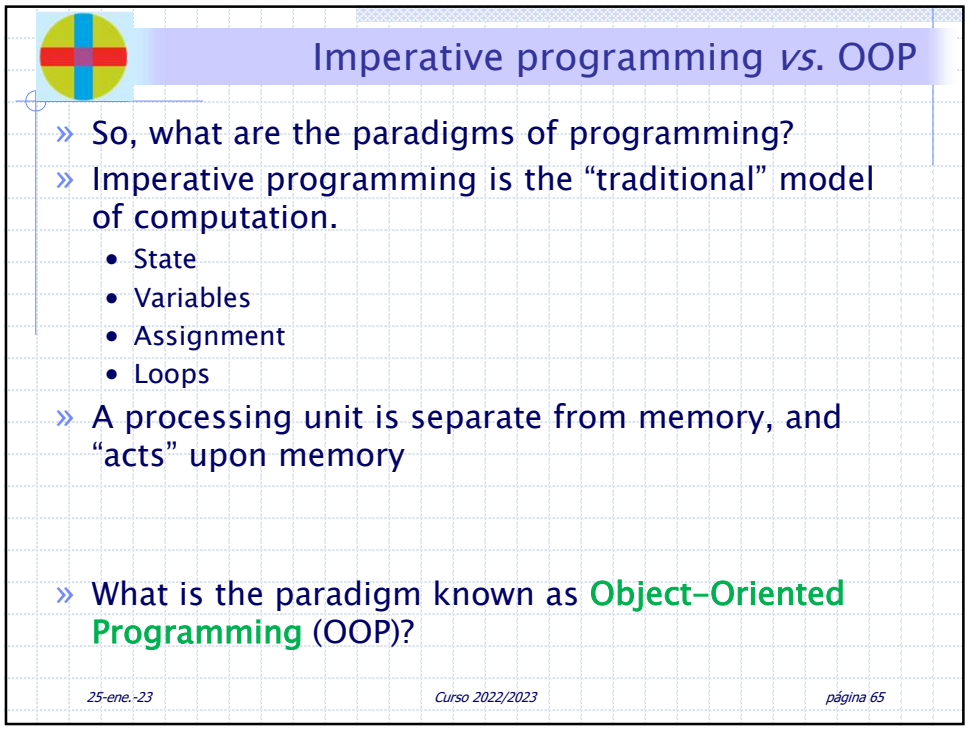



References
» Paul Hudak, "Conception, Evolution, and Application of Functional Programming Languages", ACM Computing Surveys 21/3, Sept. 1989, pp 359-411
» L. Cardelli and P. Wegner, "On Understanding Types, Data Abstraction, and Polymorphism", ACM Computing Surveys, 17/4, Dec. 1985, pp. 471-522
» D. Watt, "Programming Language Concepts and Paradigms", Prentice Hall, 1990
25-ene.-23 Curso 2022/2023 página 63

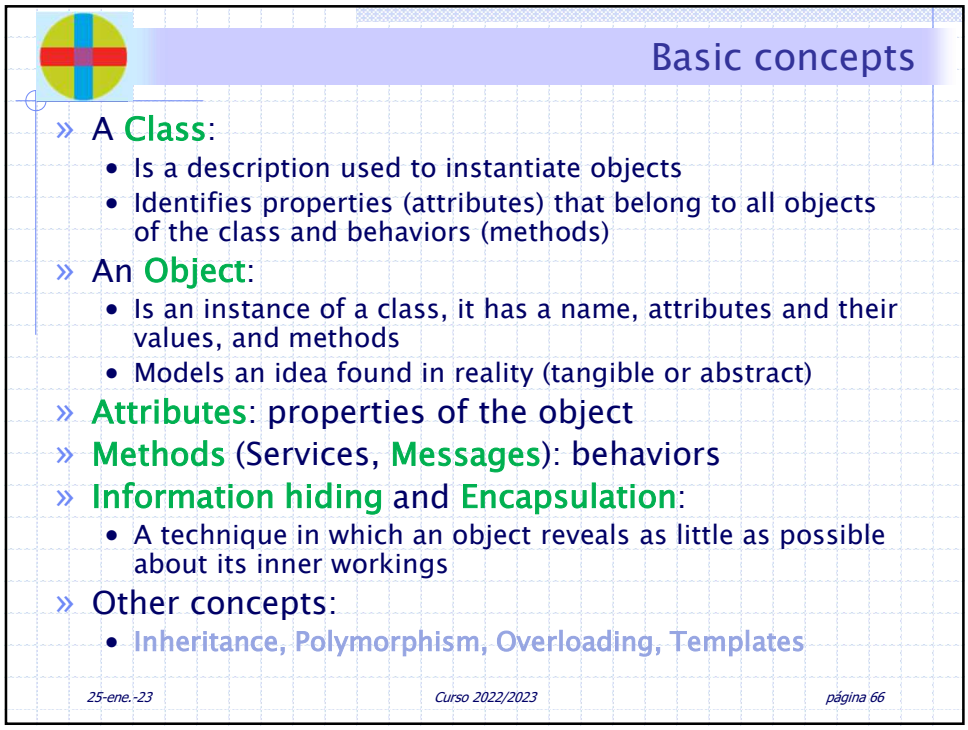
63


Object-Oriented Programming (OOP)

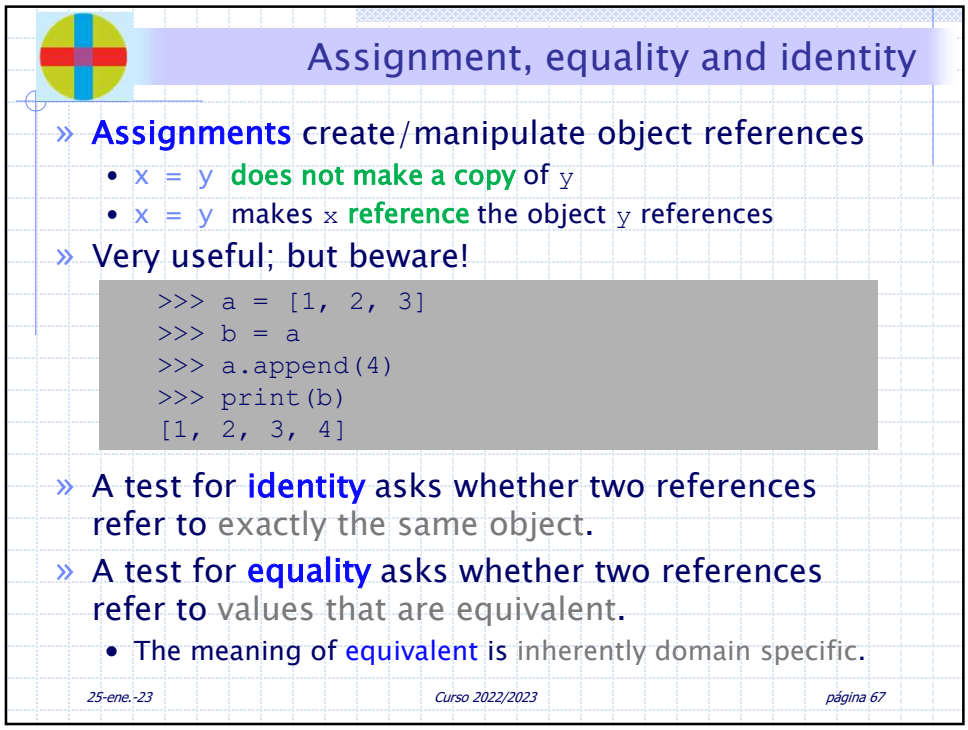
64


Imperative programming *vs.* OOP
» So, what are the paradigms of programming?
» Imperative programming is the "traditional" model of computation.
• State
• Variables
• Assignment
• Loops
» A processing unit is separate from memory, and "acts" upon memory
» What is the paradigm known as Object-Oriented Programming (OOP)?
25-ene.-23 Curso 2022/2023 página 65

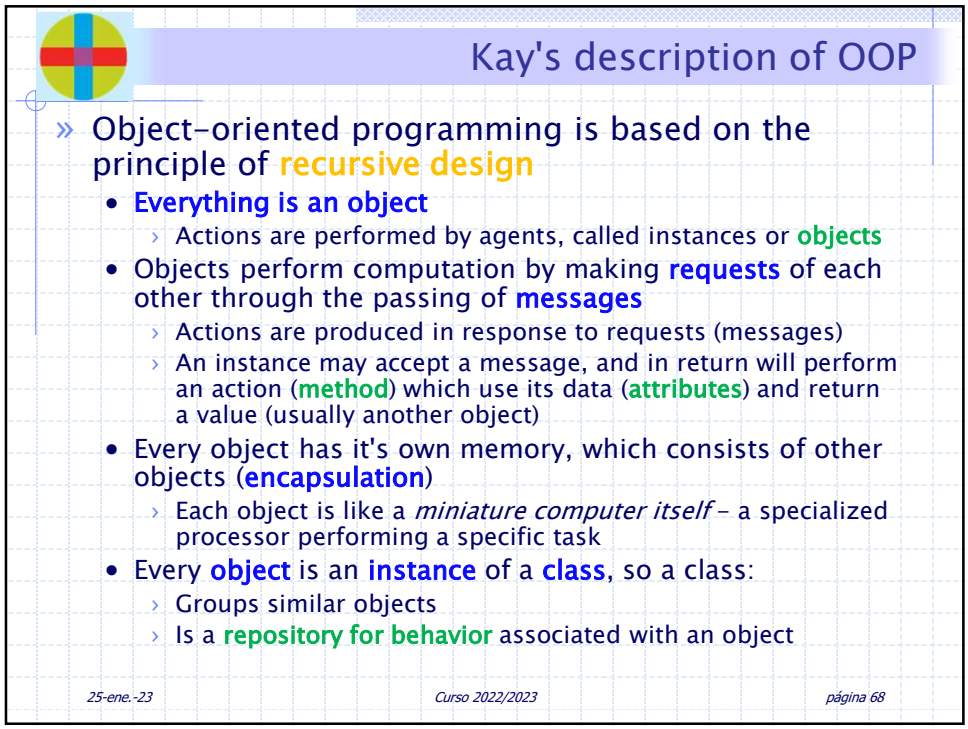
65


Basic concepts
» A Class:
• Is a description used to instantiate objects
• Identifies properties (attributes) that belong to all objects of the class and behaviors (methods)
» An Object:
• Is an instance of a class, it has a name, attributes and their values, and methods
• Models an idea found in reality (tangible or abstract)
» Attributes: properties of the object
» Methods (Services, Messages): behaviors
» Information hiding and Encapsulation:
• A technique in which an object reveals as little as possible about its inner workings
» Other concepts:
• Inheritance, Polymorphism, Overloading, Templates
25-ene.-23 Curso 2022/2023 página 66

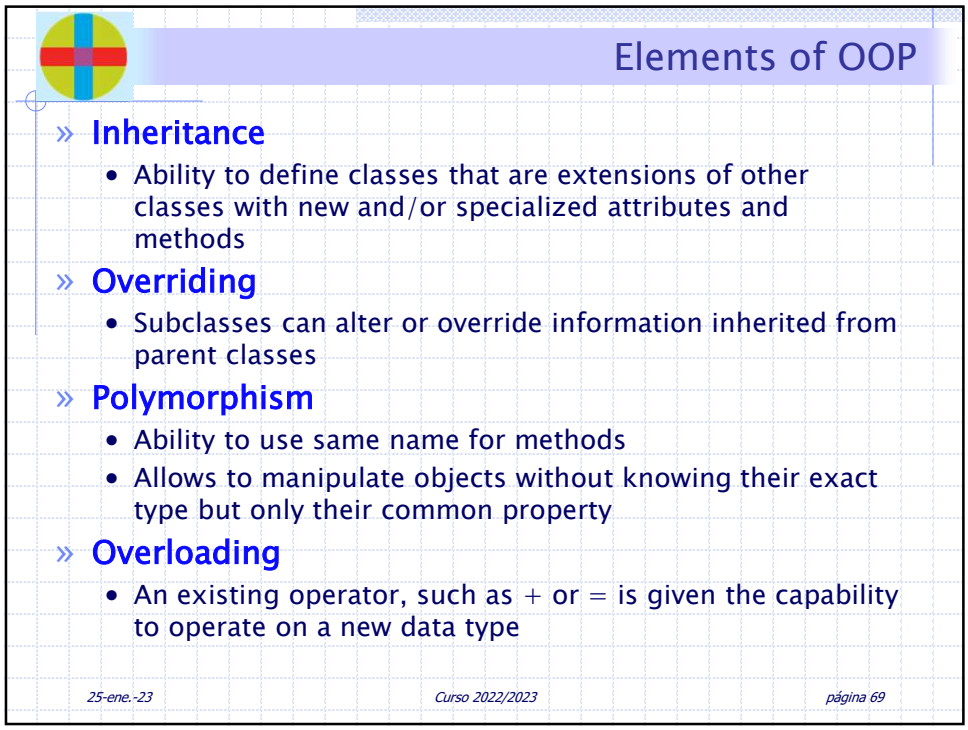
66


Assignment, equality and identity
» Assignments create/manipulate object references
• x = y does not make a copy of y
• x = y makes x reference the object y references
» Very useful; but beware!
>>> a = [1, 2, 3]
>>> b = a
>>> a.append(4)
>>> print(b)
[1, 2, 3, 4]
» A test for identity asks whether two references refer to exactly the same object.
» A test for equality asks whether two references refer to values that are equivalent.
• The meaning of equivalent is inherently domain specific.
25-ene.-23 Curso 2022/2023 página 67

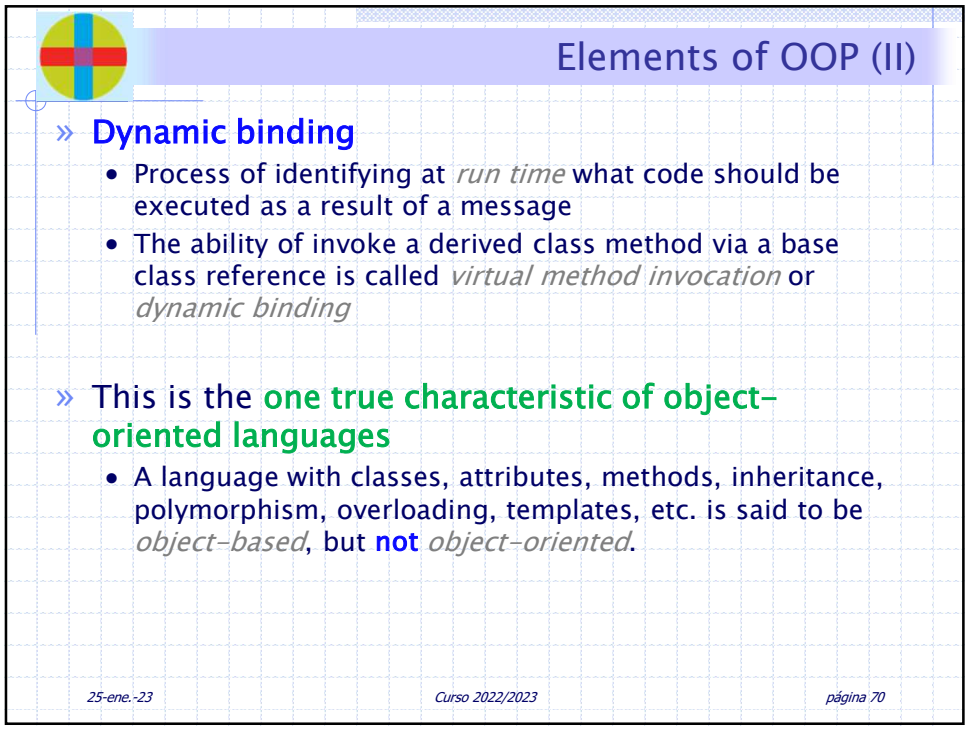
67


Kay's description of OOP
» Object-oriented programming is based on the principle of recursive design
• Everything is an object
› Actions are performed by agents, called instances or objects
• Objects perform computation by making requests of each other through the passing of messages
› Actions are produced in response to requests (messages)
› An instance may accept a message, and in return will perform an action (method) which use its data (attributes) and return a value (usually another object)
• Every object has it's own memory, which consists of other objects (encapsulation)
› Each object is like a miniature computer itself – a specialized processor performing a specific task
• Every object is an instance of a class, so a class:
› Groups similar objects
› Is a repository for behavior associated with an object
25-ene.-23 Curso 2022/2023 página 68

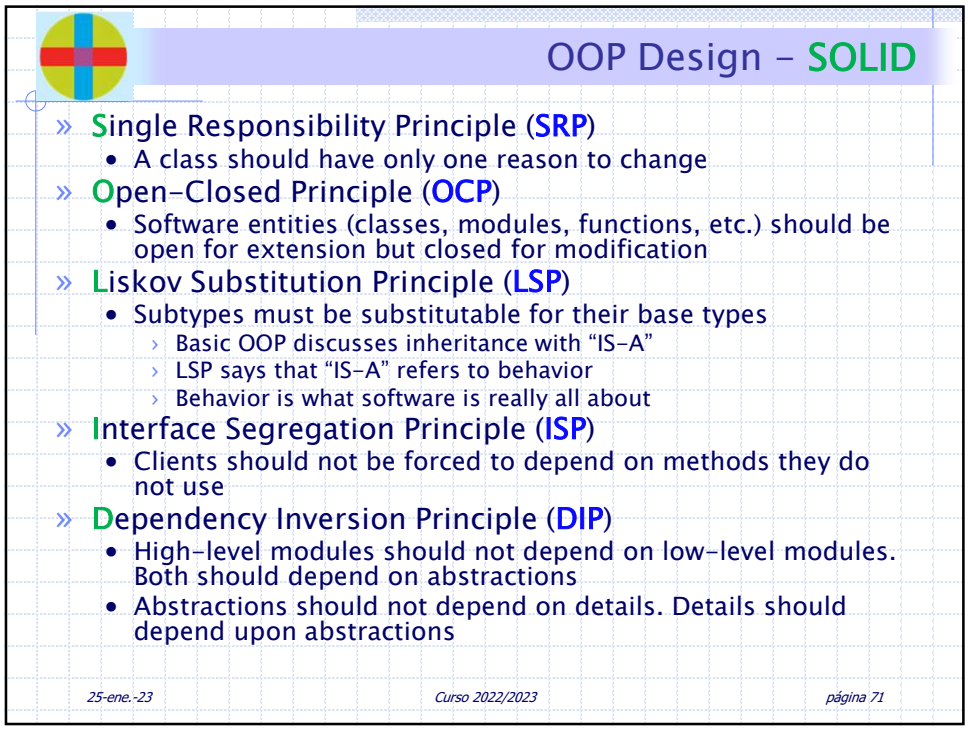
68


Elements of OOP
» Inheritance
• Ability to define classes that are extensions of other classes with new and/or specialized attributes and methods
» Overriding
• Subclasses can alter or override information inherited from parent classes
» Polymorphism
• Ability to use same name for methods
• Allows to manipulate objects without knowing their exact type but only their common property
» Overloading
• An existing operator, such as + or = is given the capability to operate on a new data type
25-ene.-23 Curso 2022/2023 página 69

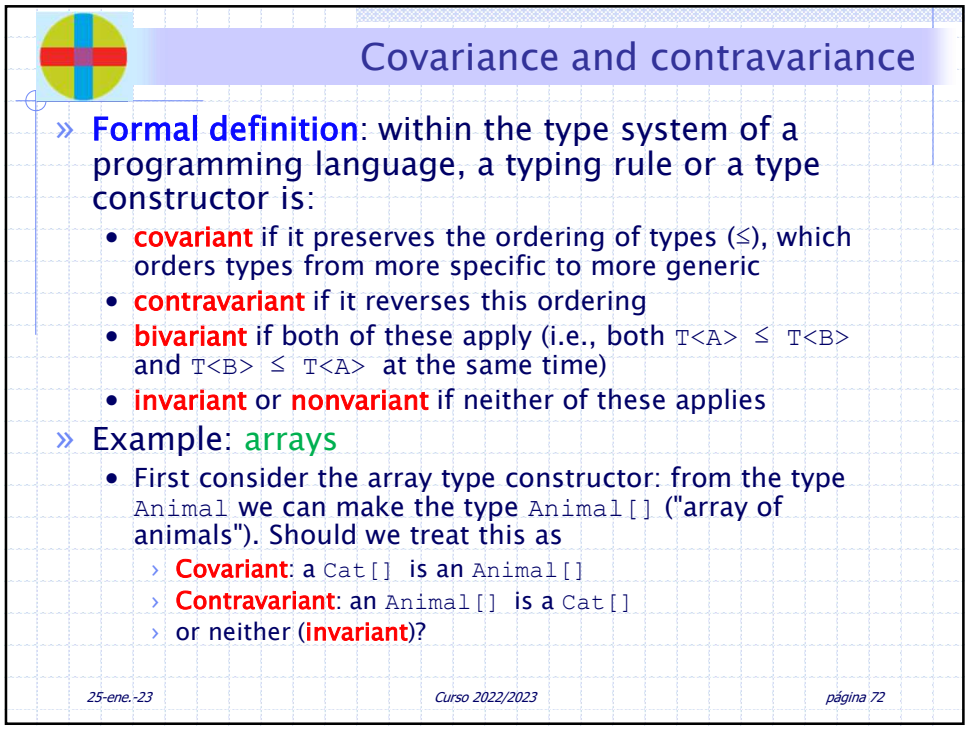
69


Elements of OOP (II)
» Dynamic binding
• Process of identifying at run time what code should be executed as a result of a message
• The ability of invoke a derived class method via a base class reference is called virtual method invocation or dynamic binding
» This is the one true characteristic of object-oriented languages
• A language with classes, attributes, methods, inheritance, polymorphism, overloading, templates, etc. is said to be object-based, but not object-oriented.
25-ene.-23 Curso 2022/2023 página 70

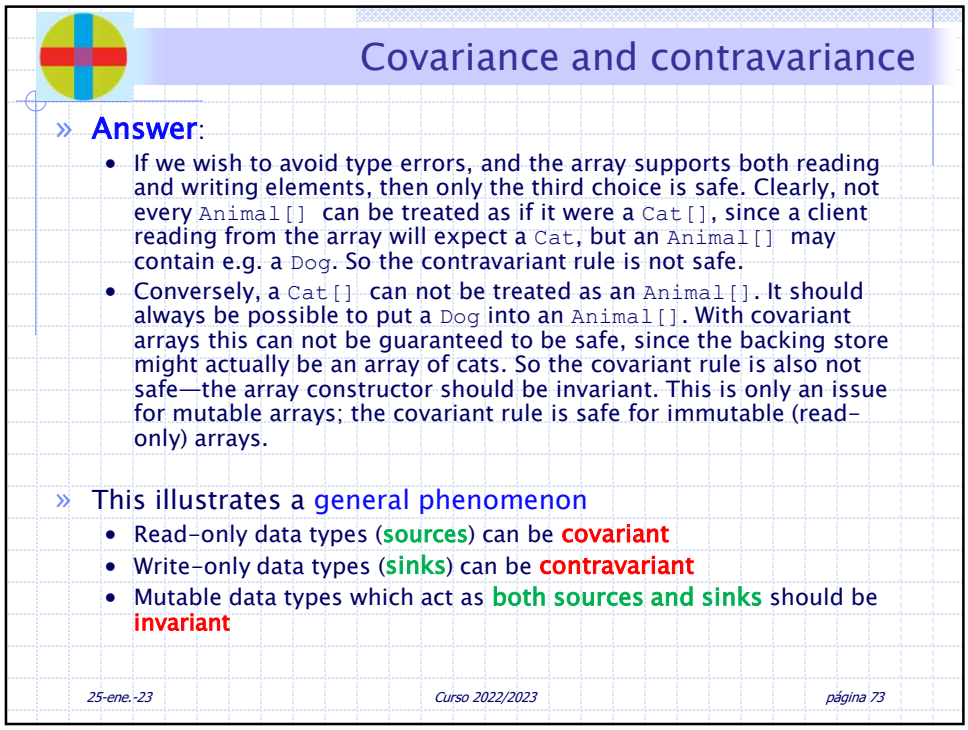
70


OOP Design – SOLID
» Single Responsibility Principle (SRP)
• A class should have only one reason to change
» Open-Closed Principle (OCP)
• Software entities (classes, modules, functions, etc.) should be open for extension but closed for modification
» Liskov Substitution Principle (LSP)
• Subtypes must be substitutable for their base types
› Basic OOP discusses inheritance with "IS-A"
› LSP says that "IS-A" refers to behavior
› Behavior is what software is really all about
» Interface Segregation Principle (ISP)
• Clients should not be forced to depend on methods they do not use
» Dependency Inversion Principle (DIP)
• High-level modules should not depend on low-level modules. Both should depend on abstractions
• Abstractions should not depend on details. Details should depend upon abstractions
25-ene.-23
Curso 2022/2023
página 71

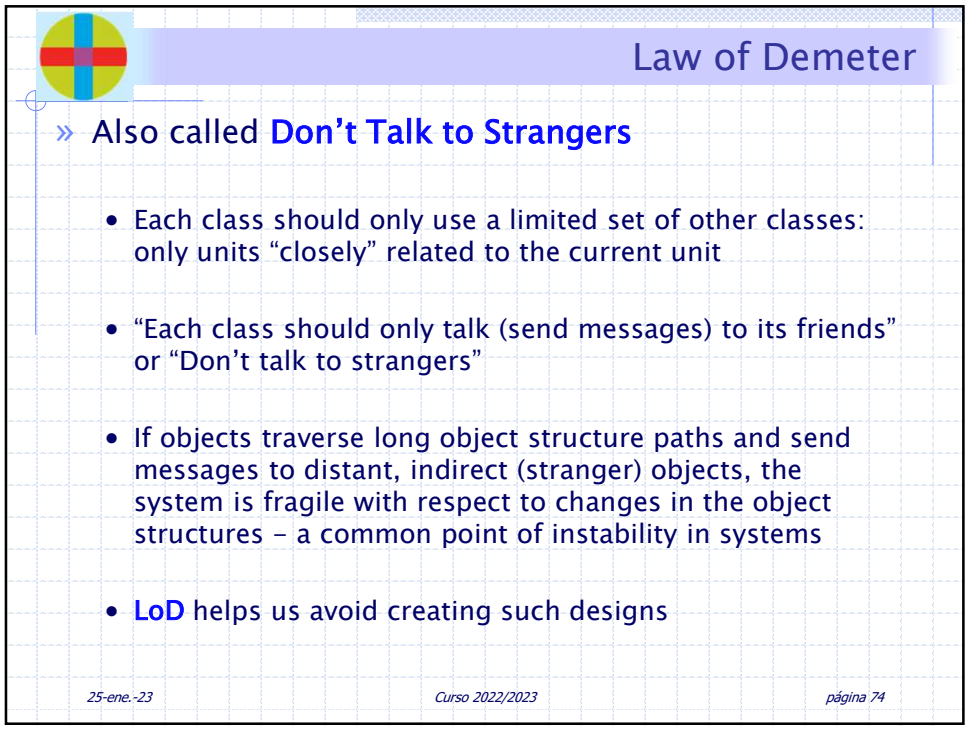
71


Covariance and contravariance
» Formal definition: within the type system of a programming language, a typing rule or a type constructor is:
• covariant if it preserves the ordering of types (≤), which orders types from more specific to more generic
• contravariant if it reverses this ordering
• bivariant if both of these apply (i.e., both T<A> ≤ T<B> and T<B> ≤ T<A> at the same time)
• invariant or nonvariant if neither of these applies
» Example: arrays
• First consider the array type constructor: from the type Animal we can make the type Animal[] ("array of animals"). Should we treat this as
› Covariant: a Cat[] is an Animal[]
› Contravariant: an Animal[] is a Cat[]
› or neither (invariant)?
25-ene.-23
Curso 2022/2023
página 72

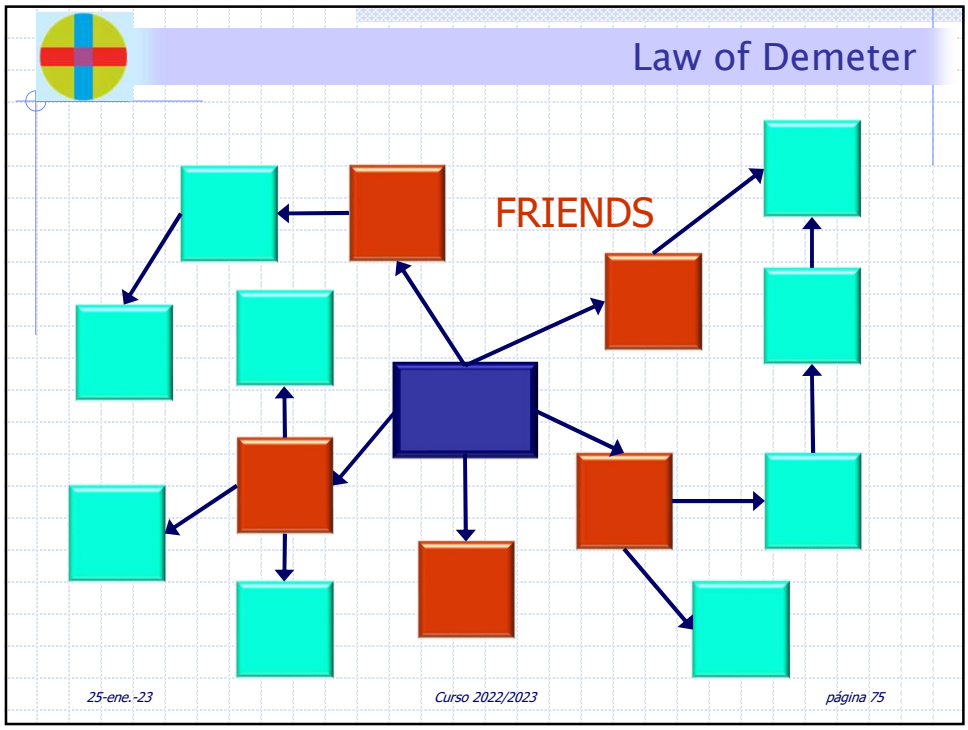
72


Covariance and contravariance
» Answer:
• If we wish to avoid type errors, and the array supports both reading and writing elements, then only the third choice is safe. Clearly, not every Animal[] can be treated as if it were a Cat[], since a client reading from the array will expect a Cat, but an Animal[] may contain e.g. a Dog. So the contravariant rule is not safe.
• Conversely, a Cat[] can not be treated as an Animal[]. It should always be possible to put a Dog into an Animal[]. With covariant arrays this can not be guaranteed to be safe, since the backing store might actually be an array of cats. So the covariant rule is also not safe—the array constructor should be invariant. This is only an issue for mutable arrays; the covariant rule is safe for immutable (read-only) arrays.
» This illustrates a general phenomenon
• Read-only data types (sources) can be covariant
• Write-only data types (sinks) can be contravariant
• Mutable data types which act as both sources and sinks should be invariant
25-ene.-23
Curso 2022/2023
página 73

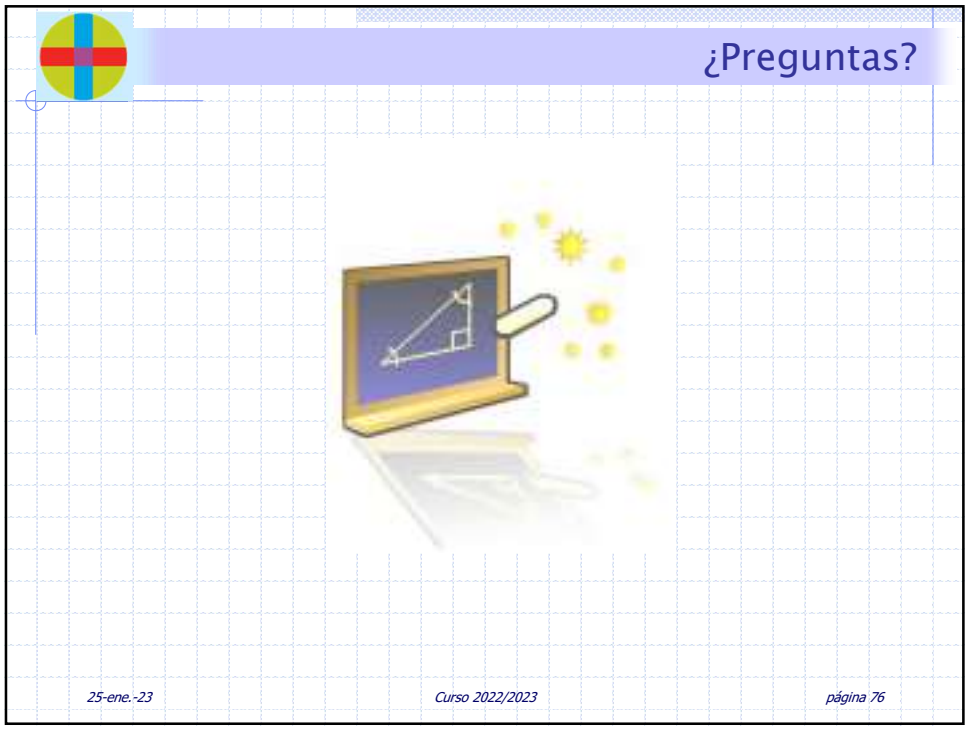
73


Law of Demeter
» Also called Don't Talk to Strangers
• Each class should only use a limited set of other classes: only units "closely" related to the current unit
• "Each class should only talk (send messages) to its friends" or "Don't talk to strangers"
• If objects traverse long object structure paths and send messages to distant, indirect (stranger) objects, the system is fragile with respect to changes in the object structures - a common point of instability in systems
• LoD helps us avoid creating such designs
25-ene.-23
Curso 2022/2023
página 74

74


Law of Demeter
FRIENDS
25-ene.-23
Curso 2022/2023
página 75

75


¿Preguntas?
25-ene.-23
Curso 2022/2023
página 76

76