



# Tema 3. Variables: definició, tipus i ús en Java

Introducció a la Informàtica i la Programació (IIP)

Curs 2019/20

Departament de Sistemes Informàtics i Computació



- **Crea** una carpeta **Tema 3** dins de la teua carpeta **W:\IIP\**
- **Descarrega** (del Tema 3 de PoliformaT) els fitxers **exemplesT3.jar** i **exercicisT3.jar** en **Tema 3**
- Des de l'opció **Projecte** de **BlueJ**, usa l'opció **Open ZIP/JAR...** per tal d'obrir-los com projectes **BlueJ** i prepara't per usar-los

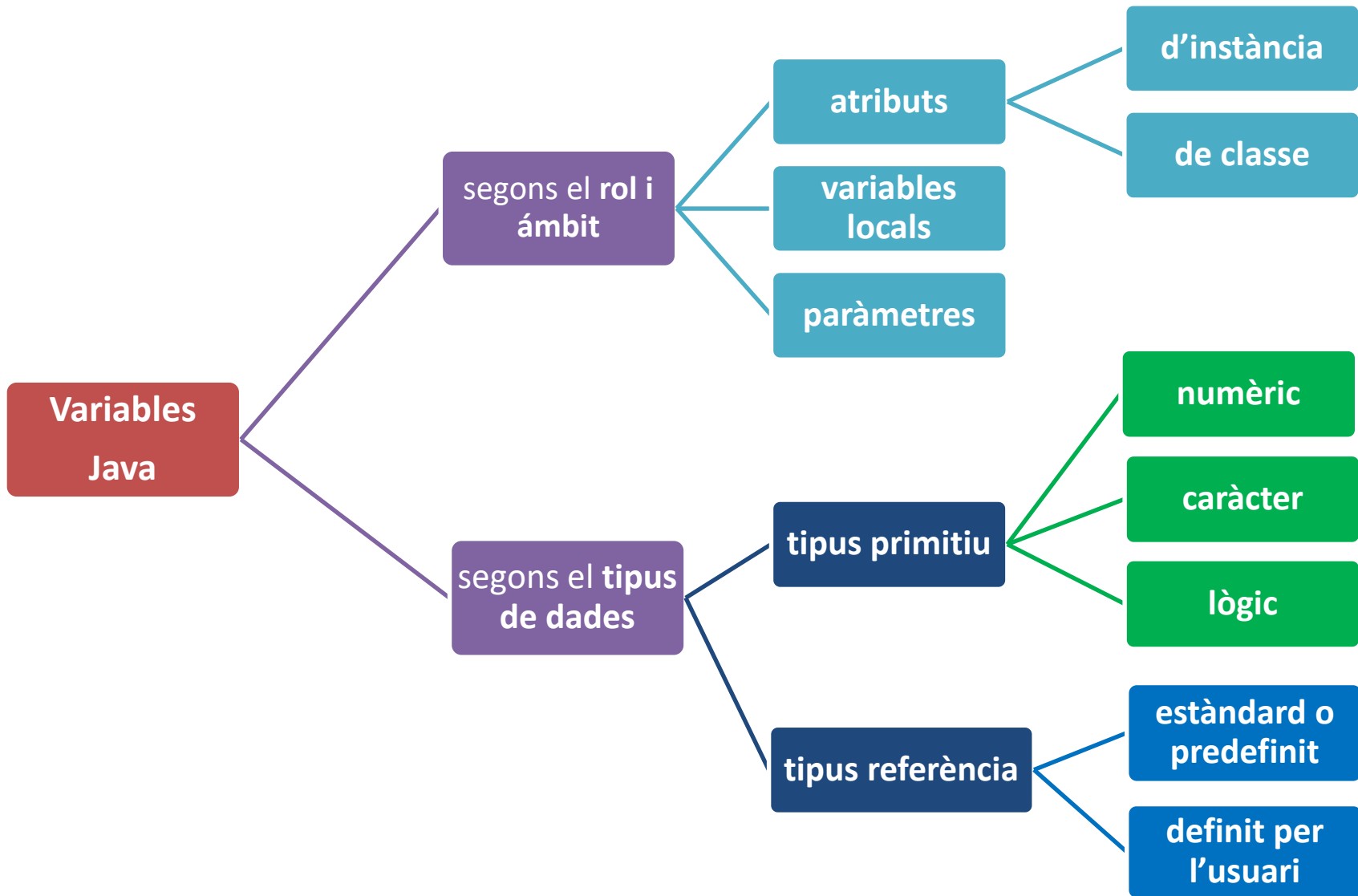


1. Variables i tipus de dades en Java
  - Variables segons el seu rol i àmbit de definició: d'instància, de classe, locals i paràmetres
  - Variables segons el seu tipus de dades: primitius i referència
2. Declaració de variables en Java: sintaxi i inicialització
3. Ús de variables de tipus primitiu
  - Modificació de l'estat d'una variable. Traça d'execució
  - L'operador d'assignació per a canviar l'estat de les variables
  - Expressions. Compatibilitat i conversió de tipus. Precedència d'operadors
4. Ús de variables de tipus referència
  - Assignació i operador `new`
  - L'ús de mètodes modificadors per a canviar l'estat dels objectes. L'operador `.`
5. Altres operacions sobre variables i objectes
  - Intercanvi de variables
  - Objectes desreferenciats i Garbage Collector
  - Còpia d'objectes
  - Igualtat de variables i objectes
6. Modicadors `final` i `static`: constants i variables de classe
7. La classe `Math`
8. Les classes `String` i `Scanner`

# Variables i tipus de dades

- La **informació** (dades, resultats, etc.) a gestionar durant la resolució d'un problema se representa mitjançant **variables**
- Cada **variable** ha de definir-se d'un **tipus de dades** que determina...
  - el **conjunt de valors** que pot emmagatzemar  
⇒ la **zona** de memòria que poden ocupar i el seu **format** (la **grandària**)
  - el **conjunt d'operacions** que se li poden aplicar
- Les variables s'han de definir abans de ser utilitzades; segons el **bloc on se defineix**, una variable té un **paper** (rol) o un altre. El bloc de codi on es defineixen (on són conegudes i es poden utilitzar) s'anomena **àmbit de definició**.

# Classificació de les variables en Java



# Variables segons el rol i àmbit de definició

- **Variables o Atributs d'instància:** representen la informació associada a cada objecte; són accessibles a través de l'identificador des de qualsevol punt de la Classe Tipus de Dades on s'han definit. Se pot usar modificadors de visibilitat per fer-les accessibles des de fora. L'accés es fa utilitzant l'operador "." precedit de l'identificador de l'objecte.
- **Variables o Atributs de classe:** representen la informació comuna a tots els objectes de la classe; també son accessibles a través de l'identificador des de qualsevol punt de la Classe Tipus de Dades on s'han definit i se pot usar modificadors de visibilitat per fer-les accessibles des de fora. L'accés es fa utilitzant l'operador "." precedit ara de l'identificador de la classe.
- **Variables Locals:** se defineixen en un mètode i són accessibles, a través del seu identificador, únicament en aquest mètode.
- **Paràmetres:** representen les dades o informació requerida per un mètode. Són accesibles, a través del seu identificador, únicament en el mètode on estan definides. Es veuran al Tema 4.

# Variables segons el rol i àmbit de definició

## Elecció d'identificadors

```
public class Cercle {  
    private double radi;  
    private String color;  
    private int centreX, centreY;  
    ...  
    public void setRadi(double nouRadi) { radi = nouRadi; }  
}
```

### Atributs o variables d'instància

- **SÍ** admeten modificadors, **ABANS** del tipus
- **Àmbit**: dins de la classe, al menys

### Paràmetre

```
public class PrimerPrograma {  
    public static void main(String[] args) {  
        Pissarra meaPissarra = new Pissarra("ESPAI DIBUIX", 300, 300);  
        Cercle c1 = new Cercle(50, "groc", 100, 100);  
        ...  
    }  
}
```

### Paràmetre

### Variables locals -i Paràmetres-

- **NO** admeten modificadors -excepte **final**-
- **Àmbit**: cos del mètode on se declaren

# Variables segons el seu tipus de dades

- **Tipus elementals, bàsics o primitius:** no se defineixen a partir d'altres i la seua representació i operacions venen donades pel propi llenguatge de programació.

Tipus de dades primitius o elementals	Nom
Tipus numèrics - nombres enters - nombres reals (en coma flotant)	byte, short, <b>int</b> , long float, <b>double</b>
Tipus caràcter	char
Tipus lògic	boolean

- **Tipus referència, complexes o estructurats:** es construeixen per l'agregació de dades del mateix tipus o de tipus diferents. Poden ser predefinits o definits per l'usuari. Se manipulen utilitzant el concepte de referència.

Tipus de dades referència (complexes o estructurats)	Nom
Classes Tipus de Dades estàndard, predefinides en llibreries com: - java.lang - java.util	System, String, Math,... Scanner...
Classes Tipus de Dades d'Usuari, definides pel programador	Cercle, Pissarra,...

# Variables segons el seu tipus de dades

- El **tipus d'una variable** determina el conjunt de **valors** que pot emmagatzemar i el conjunt d'**operacions** que se li poden aplicar

Tipus **primitius**: `int`, `double`,...

Tipus **referència**: `String`, `Cercle`, `Pissarra`,...

```
public class Cercle {  
    private double radi;  
    private String color;  
    private int centreX, centreY;  
    ...  
    public void setRadi(double nouRadi) { radi = nouRadi; }  
    public double àrea() { return 3.14 * radi * radi; }  
}
```

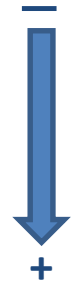
emmagatzema un valor

emmagatzema una referència a un objecte

```
public class Primer {  
    public static void main(String[] args) {  
        Pissarra meuaPissarra = new Pissarra("ESPAI DIBUIX", 300, 300);  
        Cercle c1 = new Cercle(50, "groc", 100, 100);  
        meuaPissarra.add(c1);  
        ...  
    }  
}
```



# Tipus de dades: reserva de memòria



Tipus enter	Grandària	Valor mínim $-(2^{N-1})$	Valor màxim $+(2^{N-1} - 1)$
byte	8 bits	-128	127
short	16 bits	-32768	32767
int	32 bits	-2147483648	2147483647
long	64 bits	$-2^{63}$	$2^{63}-1$

Complement a 2

IEEE 754



Tipus real	Grandària	Valor mínim	Valor màxim	Precisió.
float	32 bits	$1.4 \times 10^{-45}$	$3.4 \times 10^{38}$	7 dígits
double	64 bits	$4.9 \times 10^{-324}$	$1.8 \times 10^{308}$	15 dígits

Tipus	Grandària	Codificació
char	16 bits	Unicode

UTF-16

Tipus	Grandària	Valors	Significat
boolean	1 bit	true false	Verdader Fals

Tipus	Grandària	Valors	Significat
Classe Java	32/64 bits	Direccions de memòria	Referència a un objecte de la classe

# Tipus numèrics: valors, literals i memòria

Tipus enter	Grandària	Valor mínim $-(2^{N-1})$	Valor màxim $+(2^{N-1} - 1)$	Complement a 2
byte	8 bits	-128	127	
short	16 bits	-32768	32767	
int	32 bits	-2147483648	2147483647	
long	64 bits	$-2^{63}$	$2^{63}-1$	
Tipus real	Grandària	Valor mínim	Valor màxim	Precisió. • IEEE 754
float	32 bits	$1.4 \times 10^{-45}$	$3.4 \times 10^{38}$	7 dígits
double	64 bits	$4.9 \times 10^{-324}$	$1.8 \times 10^{308}$	15 dígits

- Literals enters:** per defecte són de tipus `int`. Per tal de **forçar** que un enter s'interprete com un `long` s'afegeix al final `L` o `l`. Poden expressar-se en:
  - decimal: 193, -50, +1200
  - octal: 0301 ( $3 \cdot 8^2 + 0 \cdot 8^1 + 1 \cdot 8^0 = 3 \cdot 64 + 1 = 193$ )
  - hexadecimal: 0xC1 ( $C \cdot 16^1 + 1 \cdot 16^0 = 12 \cdot 16 + 1 = 193$ )
- Literals reals o en coma flotant:** per defecte són de tipus `double`. Per tal de **forçar** que un real s'interprete com un `float` s'afegeix al final `F` o `f`. Poden representar-se en dos tipus de notació:
  - decimal: -123.05    0.2243    0.000000000001
  - científica: 23.4e2    -1.9E-18    1e-11

# Tipus numèrics: operadors aritmètics

- Realitzen operacions sobre dades de tipus numèric i tenen com resultat un valor de tipus numèric.

Operador	Operació
+	Suma o signe
-	Resta o signe
*	Multiplicació
/	Divisió
%	Mòdul

- Si es divideixen enters, el resultat és el quocient enter.

```
> 10 / 3
3 (int)
> 10 % 3
1 (int)
> 10 / 3.0
3.3333333333333335 (double)
> 10.0 / 3.0
3.3333333333333335 (double)
> |
```

- Si es divideix un nombre enter per zero es produeix una **excepció** (un error en temps d'execució), acabant de forma brusca l'execució.

```
> 100 / 0
Exception: java.lang.ArithmeticException (/ by zero)
```

- Si es divideix un nombre en coma flotant per zero no es genera cap excepció, el resultat és **Infinity**, **-Infinity** o **NaN**.

```
> 5.0 / 0.0
Infinity (double)
> -5.0 / 0.0
-Infinity (double)
> 0.0 / 0.0
NaN (double)
> |
```

# Tipus numèrics: desbordament

- Realitzar operacions amb nombres pot produir que el resultat excedisca la capacitat de representació del tipus. Es parla de **desbordament**.
- En l'aritmètica dels **enters** quan es sobrepassa el valor màxim representable s'obté un resultat incorrecte.

byte	$127 + 1 = -128$
short	$32767 + 1 = -32768$
int	<b><math>2147483647 + 1 = -2147483648</math></b>
long	$9223372036854775807 + 1 = -9223372036854775808$

- Per tal d'obtindre el resultat correcte, hem de tenir en compte el rang de valors de cada tipus de dades.

int	<b><math>1000000 * 1000000 = -727379968</math></b>
long	$1000000L * 1000000L = 1000000000000L$

- En l'aritmètica **real** es produeixen **desbordaments** cap a infinit (overflow) o cap a zero (underflow).
- Quan el resultat d'una operació està fora de rang, s'obté **Infinity** o **-Infinity**.

float	$1e38f * 10$	Infinity
double	$1e308 * 10$	Infinity

- Els infinits es propaguen a l'avaluació d'expressions.

$(5.0 / 0.0) + 166.386$	Infinity
-------------------------	----------

# Tipus **caràcter**: valors, literals i memòria

Tipus	Grandària	Codificació
char	16 bits	<u>Unicode</u>

UTF-16

• **Literals char**: 'a', '8', '\u006E'

Codificació ASCII (7 bits)

'A' fila 4 columna 1 -> 41 hexadecimal -> 65 decimal

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENO	ACK	BEL	BS	TAB	LF	VT	FF	CR	SO	SI
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2		!	"	#	\$	%	&	'	(	)	*	+	,	-	.	/
	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL
	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127

# Tipus caràcter: seqüències d'escapament

- Per a representar caràcters de control que no són visibles però tenen un efecte especial, s'usen **seqüències d'escapament**:

Seqüència d'escapament	Descripció
<code>\t</code>	Tabulador
<code>\n</code>	Avanç de línia (new line)
<code>\'</code>	Cometes simples
<code>\"</code>	Cometes dobles
<code>\\</code>	Barra invertida

- Una seqüència de caràcters s'escriu entre dobles cometes:

`"hola"`

`"hola\n"`

`"\"hola\""`

`"hola\tmón"`

- Les variables de tipus seqüència de caràcters se representen en Java mitjançant el tipus predefinit `String`, que s'estudiarà més endavant.

# Tipus lògic: valors, literals, memoria i operadors relacionals

Tipus	Grandària	Valors	Significat
boolean	1 bit	true false	Verdader Fals

- **Literals boolean:** true i false
- **Operadors relacionals o de comparació:** El resultat sempre és un valor de tipus `boolean`.
- Amb operands de tipus `boolean` només poden emprar-se `==` i `!=`.

Operador	Operació
<code>==</code>	Igual
<code>!=</code>	Distint
<code>&gt;</code>	Major que
<code>&gt;=</code>	Major o igual que
<code>&lt;</code>	Menor que
<code>&lt;=</code>	Menor o igual que

# Tipus lògic: operadors lògics

Operador	Operació	Significat
!	NOT	negació lògica
&	AND	conjunció o 'y' lògic
	OR	disjunció u 'o' lògic
^	XOR	'o' exclusiu
&&	AND curtcircuitat	Si el primer operand és false, el segon ja no s'avalua i el resultat és false
	OR curtcircuitat	Si el primer operand és true, el segon ja no s'avalua i el resultat és true

- Realitzen operacions sobre dades de tipus **boolean** i tenen com resultat un valor de tipus **boolean**.
- Els operadors lògics i els curtcircuitats es diferencien en que els primers avaluen els seus dos arguments, mentre que els segons no continuen amb l'avaluació si s'obté el resultat abans d'avaluar tota l'expressió.

5 < 3 && 5 < x s'avalua a false

false

- Taules de veritat

x	y	x && y x & y	x    y x   y	x ^ y	!x
false	false	false	false	false	true
false	true	false	true	true	true
true	false	false	true	true	false
true	true	true	true	false	false



# Tipus referència

Tipus	Grandària	Valors	Significat
Classe Tipus de Dades Java	32/64 bits	Direccions de memòria	Referència a un objecte de la classe

- Tots els objectes en Java es manipulen mitjançant variables de tipus referència.
- Una variable referència de tipus T conté la direcció de memòria de l'objecte de tipus T que li assigna automàticament el gestor de memòria de la JVM.
- No hi ha operacions explícites de manipulació de referències.
- Les variables referència només es poden comparar mitjançant els operadors `==` (són iguals, açò és, fan referència al mateix objecte) o `!=` (són diferents, fan referència a objectes distints).
- La constant `null` indica que no existeix cap objecte referenciat per la variable que té aquest valor especial.

# Variables i tipus de dades: **exercici**

- Es vol representar als alumnes matriculats en un grup d'IIP de l'ETSInf (A, B, C, D,...). Completa la taula indicant per a cada alumne de quin tipus seria la variable que representaria la informació que s'indica:

Dada a representar	Identificador de la variable	Tipus (Primitiu o Referència)	Identificador del tipus	Grandària en memòria
Nom				
DNI o NIE				
any de naixement				
nota del primer parcial				
grup de teoria				
amb dispensa?				

# Declaració de variables: sintaxi

- **Declaració de variable:** instrucció de definició d'una variable on se descriuen les seues característiques, definint el seu **identificador** i **tipus de dades** que restringeix els seus valors i les operacions que es poden realitzar amb ella.
- Java és un **llenguatge fortament tipat**, exigeix la **declaració** de totes les variables abans del seu ús. **OBLIGATÒRIA!**

- **Atributs o variables d'instància:**

```
[modVisibilitat] tipus nomVar1, nomVar2, ... , nomVarN;
```

- **Atributs o variables de classe:**

```
[modVisibilitat] static tipus nomVar1, nomVar2, ... , nomVarN;
```

- **Variables locals:**

```
tipus nomVar1, nomVar2, ... , nomVarN;
```

- Comporta una reserva de memòria, segons tipus: veure taula transpa 9.

- Quines són les **diferències** entre la **declaració** d'una **variable local** i un **atribut**?

# Declaració de variables: sintaxi

- La declaració d'una variable es fa en el lloc d'un **bloc** que correspon al seu **paper** (al seu **rol**)

```
public class Cercle {  
    private double radi;  
    private String color;  
    private int centreX, centreY;  
    ...  
    public void setRadi(double nouRadi) { radi = nouRadi; }  
}
```

**Atributs o variables d'instància**

- **SÍ** admeten modificadors, **ABANS** del tipus
- **Àmbit**: dins de la classe, al menys

**Paràmetre**


```
public class PrimerPrograma {  
    public static void main(String[] args) {  
        Pissarra meaPissarra = new Pissarra("ESPAI DIBUIX", 300, 300);  
        Cercle c1 = new Cercle(50, "groc", 100, 100);  
        ...  
    }  
}
```

**Paràmetre**

**Variables locals -i Paràmetres-**

- **NO** admeten modificadors -excepte **final**-
- **Àmbit**: cos del mètode on se declaren

# Declaració de variables: exercici

 BlueJ:exemplesT3

**Obre** les classes **Punt** i **ProvaPunt** del projecte i completa la següent taula.

Comença per les variables de **Punt**, afegint-les a la taula en l'ordre en el que apareixen -hem completat la primera fila a mode d'exemple-

Identificador de variable	Tipus (Primitiu o Referència)	Rol	Àmbit (Classe o Mètode)
abs	double - P	atribut	Classe Punt

# Ús d'una variable: principis bàsics

- Les variables poden canviar el seu valor al llarg de l'execució d'un programa.
- L'**estat** d'una variable és el valor que té en un moment determinat.
- Una **variable** pot canviar el seu **valor** (**estat**) mitjançant l'operador **d'assignació**.
- L'execució d'un programa pot veure's com una successió de canvis d'estat que transformen un cert estat inicial (dades) en un determinat estat final (solució).
- **L'estat d'un programa** és el contingut de les seues variables en un moment de l'execució.
- La **traça de l'execució** d'un programa és el seguiment de l'evolució dels valors de les variables en una execució.

# Canvi d'estat de les variables: l'operador d'assignació

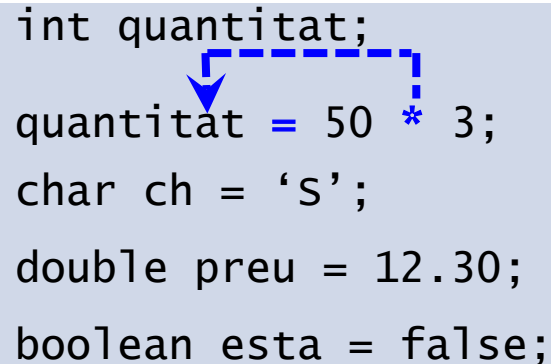
- L'**assignació** s'utilitza per donar valors inicials a les variables o per reemplaçar els valors que ja tenen per altres nous.

`identificador = expressió;`

on identificador i expressió han de ser de **tipus compatibles**.

- Una **expressió** és una successió (sintàcticament correcta) de valors, variables, operadors i crides a mètodes que s'avalua a un únic valor, sent el tipus de l'expressió el tipus d'aquest valor.
- L'operació d'assignació primer avalua l'expressió i després guarda el valor resultant en la variable identificador.

```
int quantitat;  
quantitat = 50 * 3;  
char ch = 'S';  
double preu = 12.30;  
boolean esta = false;
```



# Canvi d'estat de les variables: inicialització

- Les **variables locals** s'han d'inicialitzar **explícitament**.
- Els **atributs** s'han d'inicialitzar **en els constructors** (Tema 4).

Evita inicialitzar-los **explícitament** al declarar-los!!
- La **JVM inicialitza** cada atribut al valor **per defecte** del seu tipus Java, automàticament, quan l'objecte es crea a l'executar el mètode constructor via **new**:
  - Tipus primitius:
    - **int** i la resta de tipus enters: literal **0**
    - **double** i **float**: literal **0.0**
    - **char**: literal **'\u0000'**
    - **boolean**: literal **false**
  - Tipus referència: literal **null**, que indica que **NO EXISTEIX OBJECTE** referenciat
- En tot cas, la **seqüència d'execució de la inicialització** d'una variable atribut és:

**Per defecte → explícitament → en els constructors**



# Canvi d'estat de les variables: inicialització d'una **variable local**

Inicialització **obligatòria**, en el mètode on s'utilitza

BlueJ:exemplesT3

En la classe **ProvaCercle** del projecte BlueJ *exemplesT3*...

- **Comprova** que no hi ha errors de compilació
- **Elimina** la inicialització de la variable local **radi** del seu mètode **main** i **compila**.... **Algun error de compilació? Per què?**

```
6 public class ProvaCercle {
7     // No s'uses objectes d'aquesta classe
8     private ProvaCercle() { }
9
10    public static void main(String[] args) {
11        // crear un cercle de radi 30.0, color groc
12        // i centre en (50, 50)
13        double radi; // = 30.0;
14        double diametre = 2 * radi;
15    }
```

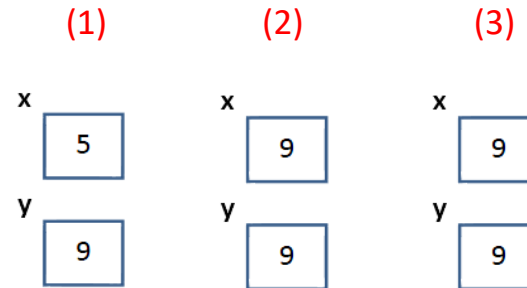
variable radi might not have been initialized

# Ús de variables de tipus primitiu:

## Intercanvi del valor de dues variables

- Fes la **traça** del codi proposat a continuació. Què ocorre? S'**intercanvien** els valors de x i y?

```
int x = 5, y = 9; (1)
x = y; (2)
y = x; (3)
```

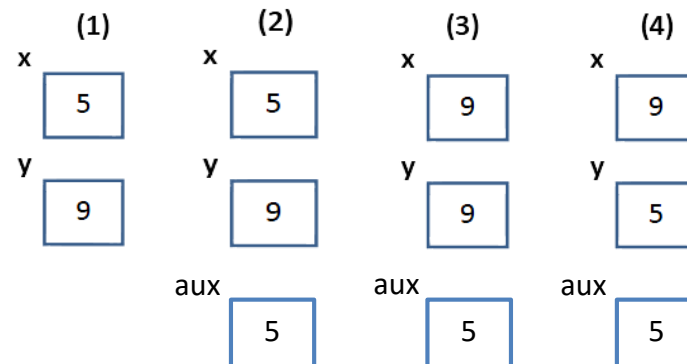


El valor que emmagatzema una variable es perd quan se li assigna un de nou

↪ **Les variables de programació NO són variables matemàtiques**

- Fes la **traça** del codi següent. Què ocorre? S'**intercanvien** ara els valors de x i y?

```
int x = 5, y = 9; (1)
int aux = x; (2)
x = y; (3)
y = aux; (4)
```



# Ús de variables de tipus primitiu:

## Expressions. Compatibilitat

- En l'assignació, variable i expressió han de ser del mateix tipus de dades o de **tipus compatibles**.
- Són **conversions de tipus implícites** o automàtiques les següents:

char  
↓  
byte → short → int → long → float → double

- El tipus d'una expressió amb operadors aritmètics i operands numèrics depèn dels tipus dels operands involucrats.
- El resultat és de tipus **double** si, almenys, un dels operands és **double**.
- El resultat és de tipus **float** si, almenys, un dels operands és **float** i cap és **double**.
- El resultat és de tipus **long** si, almenys, un dels operands és **long** i cap és real (**float** o **double**).
- El resultat és de tipus **int** si cap dels operands és **long** i tampoc és real (**float** o **double**); fins i tot si cap d'ells és de tipus **int** (com per exemple, quan en l'expressió només apareixen operands de tipus **byte**, **short** o **char**).




És a dir, el tipus de l'expressió serà el **tipus de l'operand de tipus superior**.

# Ús de variables de tipus primitiu:

## Expressions. Compatibilitat

```
// Declaració i inicialització
// Inicialització a un valor del tipus o compatible, per assignació
char c = 'A'; // el valor assignat a c és un literal de tipus char
double d1 = 2; // el valor assignat a d1 és un literal int,
               // de tipus COMPATIBLE amb double (valor 2.0 en memòria)
double d2 = 3.0 + d1; // el valor assignat a d2 és el resultat
                     // d'avaluar l'expressió 3.0 + d1, i.e. 5.0
```

 BlueJ:exemplesT3

Còpia les instruccions anteriors –no els comentaris– en el **CodePad** de BlueJ i després “**tradueix**” a Java els següents enunciats:

- **Mostrar** el **valor** de les variables c, d1 i d2 (en el *CodePad*)
- En la mateixa línia, **declarar** de tipus `int` i **inicialitzar** la variable d1Truncada a 2. **Mostrar** el seu **valor**. En què se diferencia del mostrat per a la variable d1? Per què?
- **Avaluar** (l'expressió) `3.5 + d1`, per a mostrar el seu valor i tipus
- En la mateixa línia, **declarar** de tipus `int` i **inicialitzar** la variable d2Truncada al valor `3.5 + d1`. Què ha passat? Per què?

# Ús de variables de tipus primitiu: compatibilitat i conversió de tipus

- En l'assignació, variable i expressió han de ser del mateix tipus de dades o de tipus compatibles.
- Són **conversions de tipus implícites** o automàtiques les següents:

char ↘  
byte → short → int → long → float → double

- La **conversió de tipus explícita** força la conversió entre tipus: **casting**. (tipus) expressió

BlueJ:exemplesT3


- Executa** en el **CodePad de BlueJ** del projecte les instruccions de les columnes **Exemples vàlids** i **No vàlids** de la taula, **mostrant** el seu **valor** i **comprovant** que, per als **Exemples vàlids**, coincideix amb l'indicat a la columna **Resultat**.

Exemples vàlids	Resultat	No vàlids
<pre>double dou = 123.67; long x = 98; int j = 55; long y = j; long z = 9 * j; int k = (int) 55L; j = (int) y; int enter = (int) dou; float f = (float) dou; dou = 3.403e50; f = (float) dou; int num = 6, den = 10;  double quotient = num / den; quotient = (double) num / den;</pre>	<pre>dou és 123.67 x és 98L j és 55 y és 55L z és 495L k és 55 j és 55 enter és 123 f és 123.67F dou és 3.403E50 f és Infinity; num és 6 den és 10 quotient és 0.0 quotient és 0.6</pre>	<pre>z = dou; k = 55L; j = y; enter = dou; f = dou;</pre>

# Ús de variables de tipus primitiu:

## Expressions. Precedència d'operadors

- Les expressions s'avaluen d'esquerra a dreta, respectant les regles de precedència.
- Si en una expressió apareixen operacions del mateix grup, s'avaluen amb **associativitat per l'esquerra** (d'esquerra a dreta).



Grup	Classificació	Operadors
0	Parèntesi	( )
1	Operadors unaris postfixes	(paràmetres) expr++ expr--
2	Operadors unaris prefixes	++expr --expr +expr -expr !
3	Creació o conversió	new (tipus) expr
4	Multiplicació	* / %
5	Suma	+ -
6	Relacionals	> >= < <=
7	Igualtat	== !=
8	Conjunció lògica	&
9	Disjunció exclusiva	^
10	Disjunció lògica	
11	Conjunció curtcircuitada	&&
12	Disjunció curtcircuitada	
13	Operador ternari	? :
14	Assignació	= += -= *= /= %=

# Ús de variables de tipus primitiu:

## Operadors aritmètics simples – Exercici 1 – Capítol 3

- Fes el següent exercici per tal de comprovar que has entès com s'avalua una expressió aritmètica amb operadors simples: propietats dels operadors, regles de precedència i associativitat per l'esquerra, com avaluar l'assignació, etc.

BlueJ:exercicisT3

- Executa** la Classe Programa **Exercici1C3** del **projecte BlueJ exercicisT3**. Obtindràs el resultat que es mostra a la figura de la Finestra de terminal de BlueJ.
- Ara **fes la traça** del programa, i.e. executa tu, en lloc de la JVM, en un paper, aquest programa. **Quins valors de x i y obtens?**
- Si són diferents dels que es mostren per pantalla... Em sap greu, però alguna cosa estàs fent malament. Si són els mateixos, **explica per quin motiu y val 0.0**

```
7 public class Exercici1C3 {  
8     // No s'usen objectes d'aquesta classe  
9     private Exercici1C3() { }  
10  
11     public static void main(String[] args)  
12     {  
13         double x = 5.0;  
14         double y = 7 / 9 * (x + 1);  
15         System.out.println("x = " + x + " y = " + y);  
16     }  
}
```


BlueJ: BlueJ: Finestra de terminal - exercicisT3

Opcions

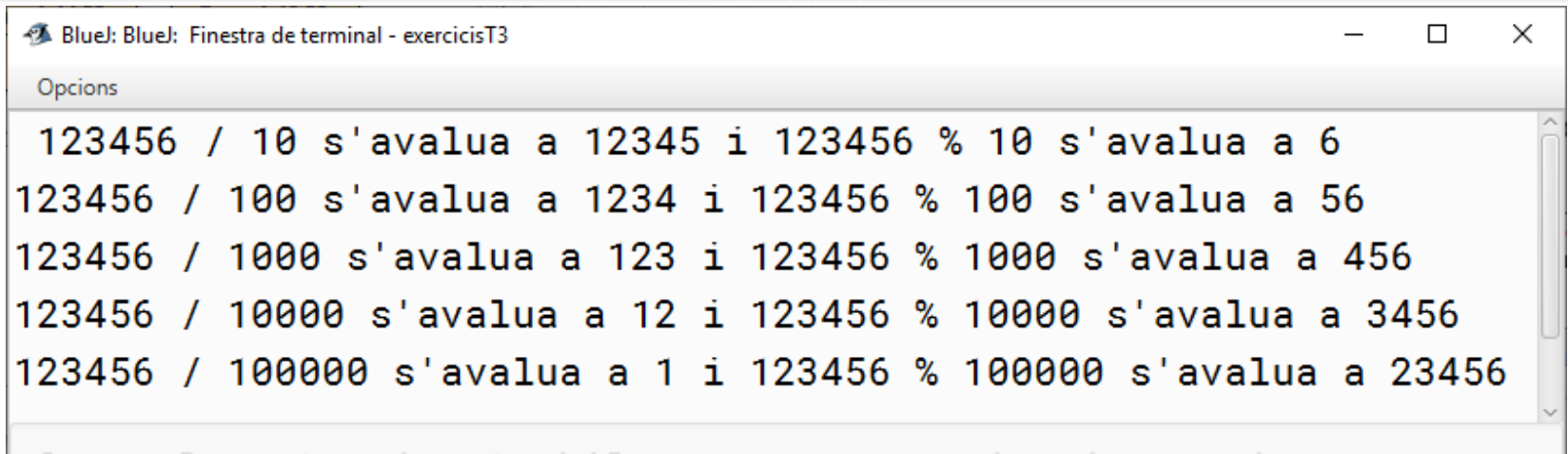
x = 5.0 y = 0.0

# Ús de variables de tipus primitiu:

## Operadors aritmètics simples – Exercici 3 – Capítol 3

 BlueJ:exercicisT3

- Aquest exercici és molt similar a l'anterior. En el projecte **exercicisT3** tens la Classe Programa **Exercici3C3** per si vols executar-lo i comprovar que si tu avalues a mà, en paper, les expressions de l'exercici obtens els mateixos resultats que la JVM, que es mostren a continuació.
- Com abans, si els resultats són diferents... Em sap greu, però alguna cosa estàs fent malament. Si són els mateixos, **explica dues coses**:
  - per a què et serveix obtenir el quocient (operador `/`) i el residu (operador `%`) de la divisió entera d'un nombre enter entre **10**, entre **100**,...?
  - Si tens un n<sup>o</sup> enter, p.e. **123456**, com obtindries el dígit de les unitats (**6** per a l'exemple)? I el de les desenes (**5** per a l'exemple)? I el de les centenes (**4** per a l'exemple)?




```
BlueJ: BlueJ: Finestra de terminal - exercicisT3
Opcions
123456 / 10 s'avalua a 12345 i 123456 % 10 s'avalua a 6
123456 / 100 s'avalua a 1234 i 123456 % 100 s'avalua a 56
123456 / 1000 s'avalua a 123 i 123456 % 1000 s'avalua a 456
123456 / 10000 s'avalua a 12 i 123456 % 10000 s'avalua a 3456
123456 / 100000 s'avalua a 1 i 123456 % 100000 s'avalua a 23456
```




# Ús de variables de tipus primitiu:

## Operadors aritmètics simples – Exercici 4 – Capítol 3

 BlueJ:exercicisT3

- Aquest exercici segueix la tònica dels anteriors. En el projecte **exercicisT3** tens la Classe Programa **Exercici4C3** per si vols executar-lo i comprovar que els resultats als que s'avaluen actualment les expressions que has de corregir són els que es mostren a continuació.
- Si tu avalues a mà, en paper, les expressions de l'exercici, **obtens els mateixos resultats que la JVM?**
- **Què modificaries en cada expressió per tal que s'avalue al resultat desitjat que proposa l'exercici?**

 BlueJ: BlueJ: Finestra de terminal - exercicisT3

Opcions

```
3 / 4 * (a * a - b)    s'avalua a 0,      però el resultat desitjat és 16.5
a / b * 1000 + 304     s'avalua a 1304,   però el resultat desitjat és 1970.6666666666667
(100 / a + b / 2) * 5  s'avalua a 105,    però el resultat desitjat és 107.5
```

Can only enter input while your programming is running

# Ús de variables de tipus primitiu:

## operadors aritmètics compostos i d'increment i decrement en 1

Operador	Operació
<code>+=</code>	Suma i assignació
<code>-=</code>	Resta i assignació
<code>*=</code>	Multiplicació i assignació
<code>/=</code>	Divisió i assignació
<code>%=</code>	Mòdul i assignació
<code>++</code>	Increment en 1
<code>--</code>	Decrement en 1

- Realitzen operacions sobre dades de tipus numèric i tenen com resultat un valor de tipus numèric.

`a++`  $\equiv$  `a = a + 1`      `a += b`  $\equiv$  `a = a + b`

`++` i `--` poden aparèixer en notació **prefixa** o **sufixa**:

`++a`    primer s'incrementa el valor de la variable i després s'utilitza


`a++`    primer s'utilitza la variable i després s'incrementa el seu valor

# Ús de variables de tipus primitiu:

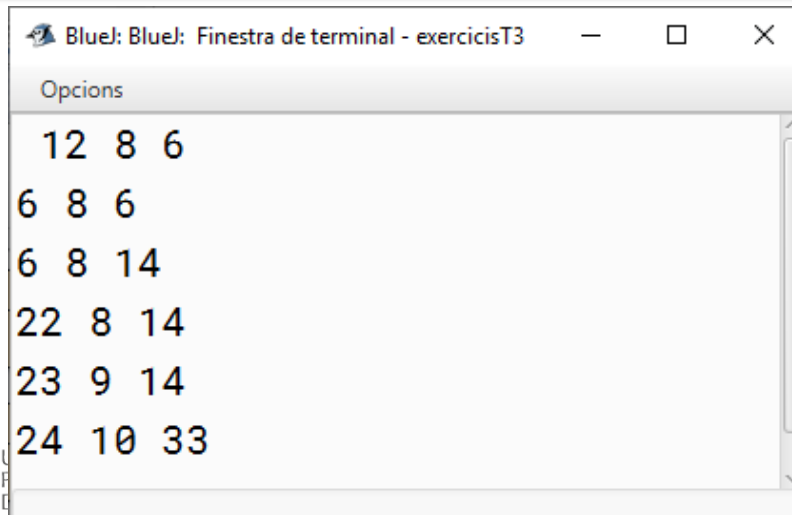
## Operadors aritmètics compostos i d'increment i decrement en 1

### Exercici 9 – Capítol 3

- Fes és el següent exercici per tal de comprovar que has entès com funcionen els operadors aritmètics compostos i d'increment i decrement en 1.

 BlueJ:exercicisT3

- Executa** la Classe Programa **Exercici9C3** del projecte **exercicisT3**. Obtindràs el resultat que es mostra a la figura de la Finestra de terminal de BlueJ.
- Ara **fes la traça** del programa, i.e. executa tu, en lloc de la JVM, en un paper, aquest programa. **Quins valors obtens?**
- Si són diferents dels que es mostren per pantalla ... Em sap greu, però alguna cosa estàs fent malament. Si són els mateixos, **explica per quin motiu**



```
BlueJ: BlueJ: Finestra de terminal - exercicisT3
Options
12 8 6
6 8 6
6 8 14
22 8 14
23 9 14
24 10 33
```

# Ús de variables de tipus primitiu:

## Operadors aritmètics compostos

BlueJ:exemplesT3

```
7 public class Exemple35C3 {
8     // No s'uses objectes d'aquesta classe
9     private Exemple35C3() { }
10
11     public static void main(String[] args) {
12         long segons = 765432;    // quantitat de segons
13         long dies = segons / (24 * 60 * 60);    // seg / (24h/dia * 60min/h * 60seg/min)
14         segons %= 24 * 60 * 60;    // equival a segons = segons % (24 * 60 * 60);
15         System.out.println("Dies: " + dies);
16         System.out.println(" (Segons restants: " + segons + ")");
17         long hores = segons / (60 * 60);
18         segons %= 60 * 60;
19         System.out.println("Hores: " + hores);
20         System.out.println(" (Segons restants: " + segons + ")");
21         long minuts = segons / 60;
22         segons %= 60;
23         System.out.println("Minuts: " + minuts);
24         System.out.println("Segons restants: " + segons);
25     }
26 }
```

BlueJ: BlueJ: Finestra de terminal - exemplesT3

Opcions

```
Dies: 8
(Segons restants: 74232)
Hores: 20
(Segons restants: 2232)
Minuts: 37
Segons restants: 12
```

# Ús de variables de tipus primitiu:

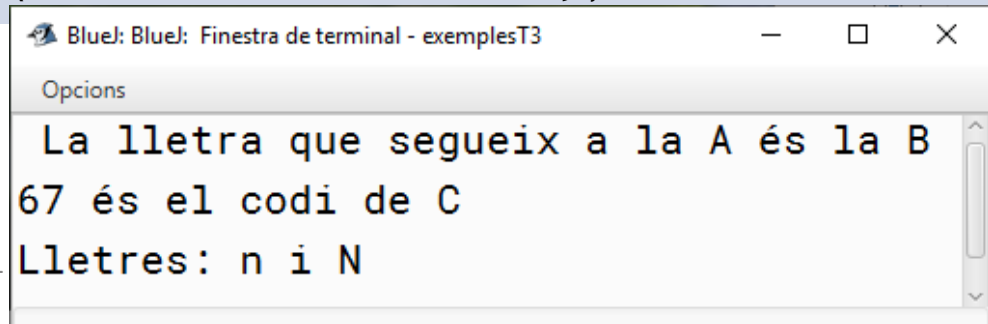
## Operacions amb char. Detalls i exemples

- Un literal de tipus char es representa internament com un valor enter positiu (però sense la representació en complement a 2). Per tant, pot usar-se l'aritmètica amb enters i la conversió forçada de tipus per operar amb ells.

BlueJ:exemplesT3


- Executa** les següents instruccions en el **CodePad de BlueJ** del projecte.
- Comprova** que el resultat és el que es mostra a la figura de la Finestra de terminal de BlueJ

```
char ch = 'A';
char lletraB = (char) ((int) ch + 1); // (char)(ch + 1)
System.out.println("La lletra que segueix a la " + ch + " és la " + lletraB);
char lletraC = 'B' + 1;
System.out.println((int) lletraC + " és el codi de " + lletraC);
char lletraN = '\u006E'; // codi unicode hexadecimal de la n
int dif = 'a' - 'A'; // int dif = 32;
lletraN = (char) (lletraN - dif);
System.out.println("Lletres: " + '\u006E' + " i " + lletraN);
```



# Ús de variables de tipus primitiu:

## Operadors relacionals. Detalls i exemples

 BlueJ:exemplesT3


- **Executa** les següents instruccions en el **CodePad de BlueJ** del projecte.
- **Quin és el valor de** esMenor, esMajor, esDoble, esPositiu i esParell?

```
boolean esMenor = 'a' < 'b';  
int x = 5, y = 2 * x;  
boolean esMajor = x > 10;  
boolean esDoble = y == 2 * x;  
boolean esPositiu = y > 0;  
boolean esParell = x % 2 == 0;
```

# Ús de variables de tipus primitiu:

## Operadors lògics curtcircuitats. Detalls i exemples

- Els operadors lògics simples i els curtcircuitats es diferencien en què els primers avaluen els seus dos arguments, mentre que els segons NO continuen amb l'avaluació si s'obté el resultat abans d'avaluar tota l'expressió.

 BlueJ:exemplesT3

- Què han de complir les variables valor i x en cada una de les expressions següents per tal que s'avaluen a true?**

```
(valor >= 15) && (valor <= 20)
(valor < 15) || (valor > 20)
!(valor >= 15 && valor <= 20)
(valor > 15) || (valor == 15)
(x % 2 != 1) && (x >= 0 && x < 5 || x >= 10 && x <= 20)
```

Equivalent a  
 $x \% 2 == 0$

# Ús de variables de tipus referència: assignació i operador new

- En el moment de la declaració d'una variable de tipus referència encara no existeix l'objecte referenciat.
- Quan es vol utilitzar un objecte d'un tipus determinat és necessari crear-lo explícitament, utilitzant l'operador **new**.
- Fins el moment de la seua creació, l'objecte no existeix i qualsevol intent de referenciar-lo provocarà en error en temps d'execució.
- A una variable de tipus **referència** se li assigna **un objecte** del seu tipus o compatible, després d'haver-lo creat via l'operador **new**.
  - La constant **null** és un *valor especial* que se li pot assignar a una variable de tipus referència per indicar que, inicialment, no se li assigna cap objecte.

```
Cercle c;  
c = new Cercle();  
// o també...  
Cercle c = new Cercle();
```



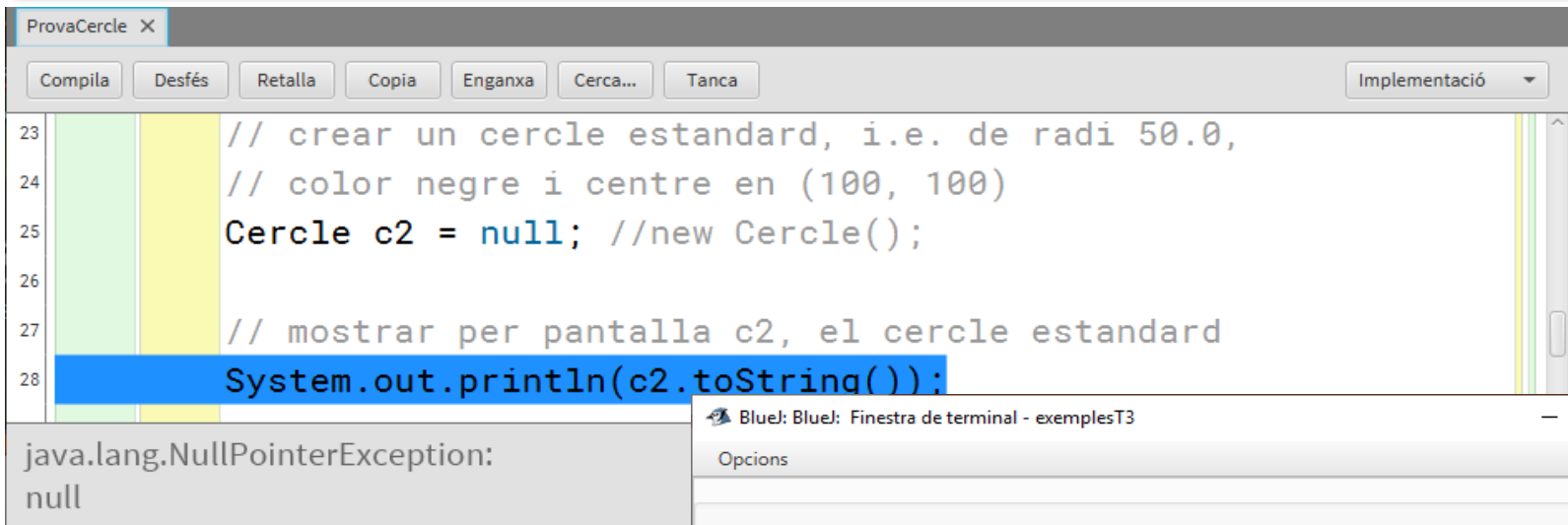
# Ús de variables de tipus referència: inicialització d'una variable local

Inicialització **obligatòria**, en el mètode on s'utilitza

BlueJ:exemplesT3

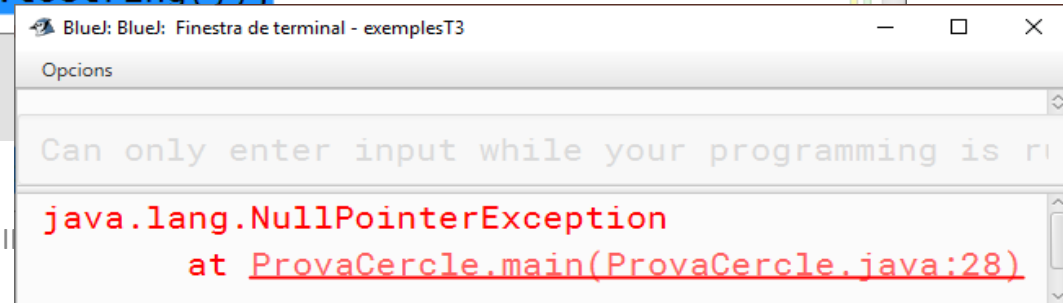
En la classe **ProvaCercle** del projecte BlueJ **exemplesT3**...

- **Comprova** que no hi ha errors de compilació
- **Canvia** a **null** la inicialització de la variable local **c2** del seu mètode **main** i **compila**... **Error de compilació?**
- Si compila sense error, **executa**... **Error d'execució? Per què?**



```
23 // crear un cercle estandard, i.e. de radi 50.0,
24 // color negre i centre en (100, 100)
25 Cercle c2 = null; //new Cercle();
26
27 // mostrar per pantalla c2, el cercle estandard
28 System.out.println(c2.toString());
```

java.lang.NullPointerException:  
null




BlueJ: BlueJ: Finestra de terminal - exemplesT3

Opcions

Can only enter input while your programming is running

java.lang.NullPointerException  
at ProvaCercle.main(ProvaCercle.java:28)

# Exercici: inicialització d'atributs

 BlueJ:exercicisT3

**Completa** la classe **AlumneIIP** del projecte **exercicisT3**, per a representar a qualsevol alumne matriculat en IIP en un grup de l'ETSinf, de manera que...

- Els seus **atributs** siguen tants com els de la taula que has completat en l'**exercici de la transparència nº 18** d'aquest document, amb els mateixos identificadors i tipus que vas definir.
- Tinga un únic **mètode** que siga un **constructor per defecte**, és a dir, que cree un alumne de nom **Pau Gasol**; amb DNI **123456Y**; nascut l'any **1980**; matriculat en el grup **A**; amb una nota de **9.8** en el primer parcial; i, finalment, per motius obvis, amb **dispensa** d'assistència a classe.

# Ús de variables de tipus referència: representació en memòria

- Quan es crea un objecte amb l'operador new i s'assigna a una variable de tipus referència, el sistema li associa memòria com segueix:
  - Tota la informació pròpia de l'objecte s'emmagatzema en una zona de memòria coneguda com **monticle o heap**. És una zona de memòria dinàmica ja que, en crear-se un objecte se li assigna memòria en aquesta zona però, quan l'objecte es destrueix, se li desassigna la memòria, i pot ser reutilitzada posteriorment pel sistema.
  - La referència (o adreça de memòria de l'objecte) s'emmagatzema amb la resta de variables (referències o elementals) en un altra zona de memòria. Així, el sistema és capaç de determinar durant l'execució del programa on es troba l'objecte per poder utilitzar-lo.

# Ús de variables de tipus referència: representació en memòria

- El **tipus** d'una variable (**primitiu** o **referència**) determina
    - el conjunt de **valors** que pot emmagatzemar
    - el conjunt d'**operacions** que se li poden aplicar
- **zona de memòria** que ocupa (*Heap VS No Heap*) i el seu **format**

The diagram illustrates the memory representation of a variable of type `Cercle`. On the left, a red window titled "c1 : Cercle" shows the object's fields: `private double radi` (50.0), `private String color` ("groc"), `private int centreX` (100), and `private int centreY` (100). Below this window, the text "MONTICLE o HEAP" is visible. On the right, a larger window shows the memory layout. It lists the fields: `radiC1` (50.0 (double)), `centreXC1` (100 (int)), `centreYC1` (100 (int)), and `c1` (@5e21151e (Cercle)). The variable `c1` is highlighted with a green box, and its value is shown as @5e21151e. A small figure with a magnifying glass is positioned over the `c1` variable. The text "No HEAP" is visible in the bottom right corner of the right window.

- A la variable `c1` se li assigna la posició de memòria (un `int`), on comença la descripció de l'objecte

→ La variable `c1` **referència** a l'objecte, **NO ÉS** l'objecte

→ `c1` **ÉS** una **variable referència**

# Canvi d'estat dels objectes:

## variables referència, l'operador . i mètodes modificadors

- El canvi d'estat d'algun dels atributs d'un objecte implica el canvi d'estat del propi objecte.
- Si els atributs s'han definit privats, la **modificació directa** dels atributs via assignació només es pot realitzar **dins de la classe Tipus de Dades on s'han definit**.

```
public class Cercle {  
    private double radi;  
    ...  
    public void setRadi(double nou) { radi = nou; }  
    ...  
}
```

- **Fora de la classe**, els atributs només poden ser **inicialitzats** a través dels mètodes **constructors** (via l'operador **new**) o **modificats** a través dels mètodes **modificadors** (via l'identificador referència i l'operador **.**).

```
Cercle c = new Cercle();  
c.setRadi(5.0);
```

- Així doncs, un **objecte** pot **canviar el seu estat** (els valors dels seus atributs) a través dels mètodes **modificadors** utilitzats amb l'**identificador de la variable** que el **referència** i l'**operador .**

# Canvi d'estat dels objectes:

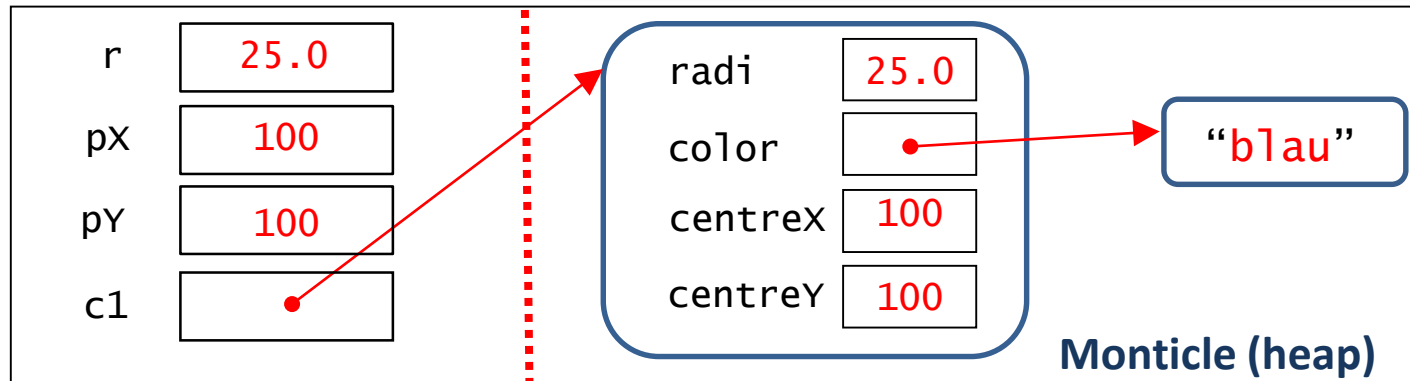
## Traça – estat de la memòria

```
double r = 25.0;  
int px = 100, py = 100;  
Cercle c1 = new Cercle(r, "blau", px, py);
```

BlueJ:exemplesT3

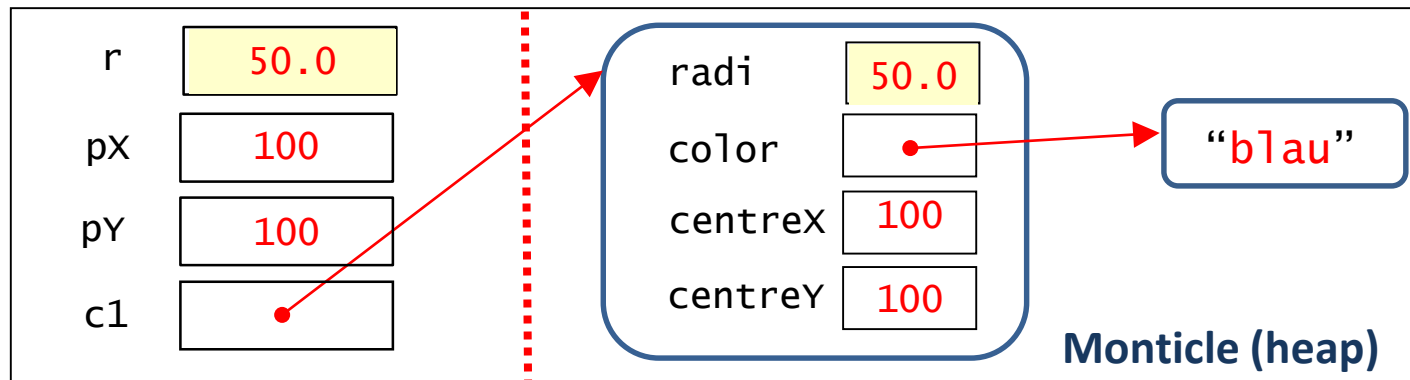
- Executa les següents instruccions (sense els comentaris) en el **CodePad de BlueJ** del projecte, comprovant que l'**estat de la memòria** coincideix amb el que es mostra a les figures.

Memòria del sistema



```
r = 2 * c1.getRadi(); // avaluació de l'expressió a un valor double o compatible  
c1.setRadi(r);
```

Memòria del sistema



El valor que emmagatzema una variable es perd quan se li assigna un de nou

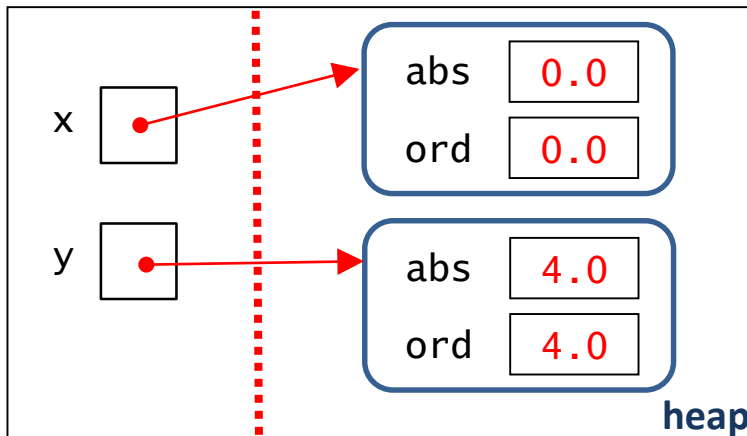
↳ Les variables de programació **NO** són variables matemàtiques

# Canvi d'estat de variables referència: Intercanvi del valor de dues variables

- Fes la **traça** del codi proposat a continuació. S'**intercanvien** els **valors** de x i y o els **objectes** que referencien?

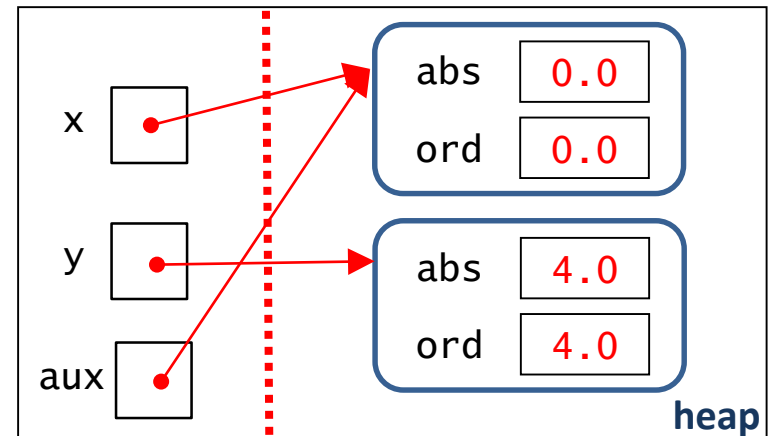
```
Punt x = new Punt(), y = new Punt(4, 4); (1)  
Punt aux = x; (2)  
x = y; (3)  
y = aux; (4)
```

Memòria del sistema (1)



 2 objectes i 2 referències

Memòria del sistema (2)



 2 objectes i 3 referències

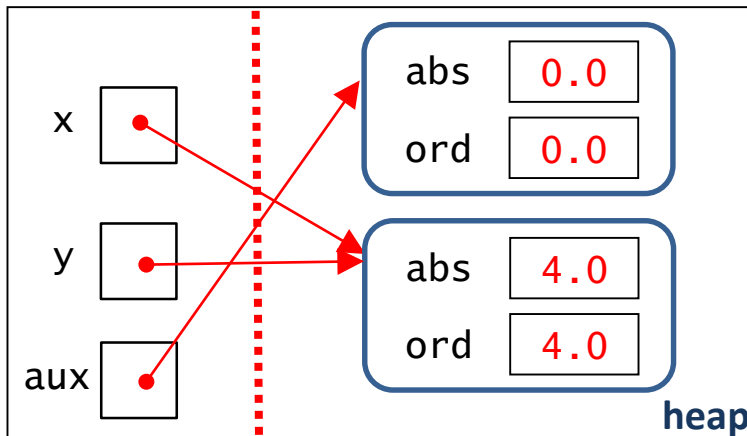
# Canvi d'estat de variables referència: Intercanvi del valor de dues variables

- Fes la **traça** del codi proposat a continuació. S'**intercanvien** els **valors** de x i y o els **objectes** que referencien?

```
Punt x = new Punt(), y = new Punt(4, 4);  
Punt aux = x;  
x = y;  
y = aux;
```

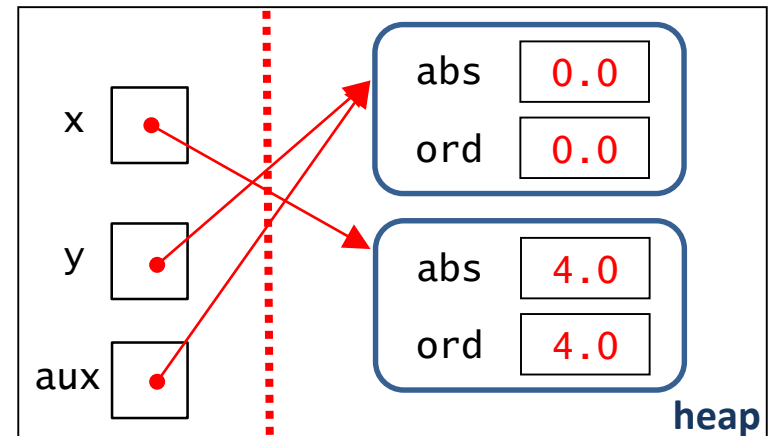
(1)  
(2)  
(3)  
(4)

Memòria del sistema (3)



 canvi de referència

Memòria del sistema (4)



 intercanvi de referencies



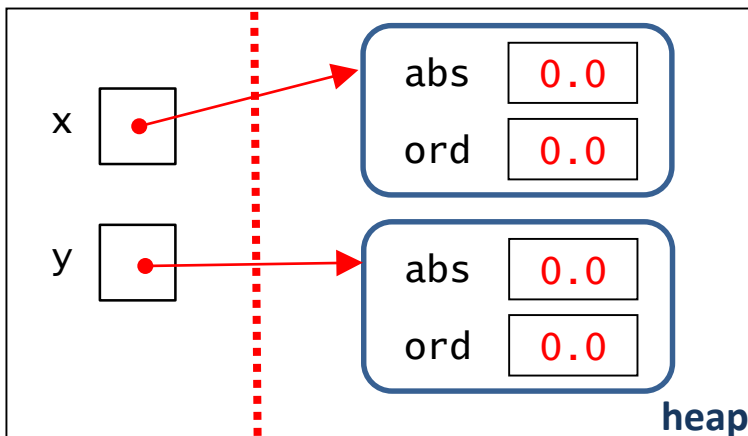
# Canvi d'estat de variables referència:

## Objecte “desreferenciat” – *Garbage collector*

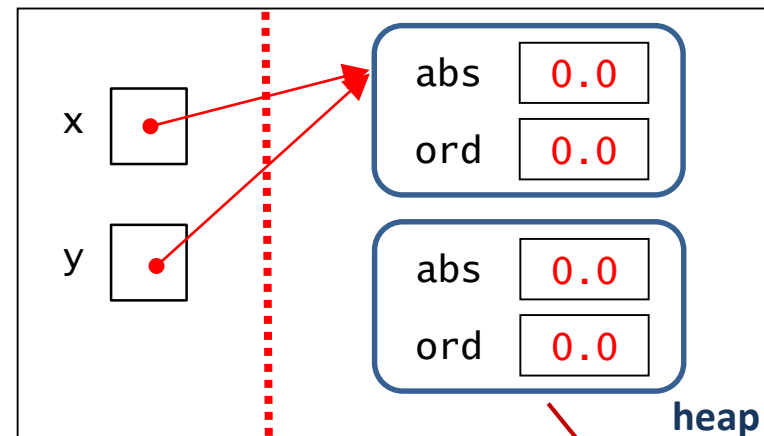
- Fes la **traça** del codi proposat a continuació. Què ocorre amb l'**objecte desreferenciat**? Es queda per sempre en el heap, ocupant espai inútilment?

```
(1) Punt x = new Punt(), y = new Punt();  
(2) y = x;           // x i y referencien al primer objecte,  
                    // no hi ha res que referencie al segon
```

Memòria del sistema (1)



Memòria del sistema (2)



objecte  
**desreferenciat**

# Canvi d'estat de variables referència:

## Objecte “desreferenciat” – *Garbage collector*

- És un **subsistema de la JVM** que recupera la memòria dels objectes no referenciats perquè puga tornar a ser utilitzada.
- És freqüent en els llenguatges actuals basats en l'ús d'una màquina virtual (per exemple Java, C#, Python). En altres llenguatges (per exemple C++ o Ada) cal que el programador allibere explícitament la memòria que s'ha deixat d'utilitzar.
- En Java **entra en funcionament automàticament**, però és possible desactivar el seu funcionament, si es vol, o suggerir al sistema que l'execute mitjançant una utilitat de la classe System (System.gc()).

# Altres operacions sobre variables:

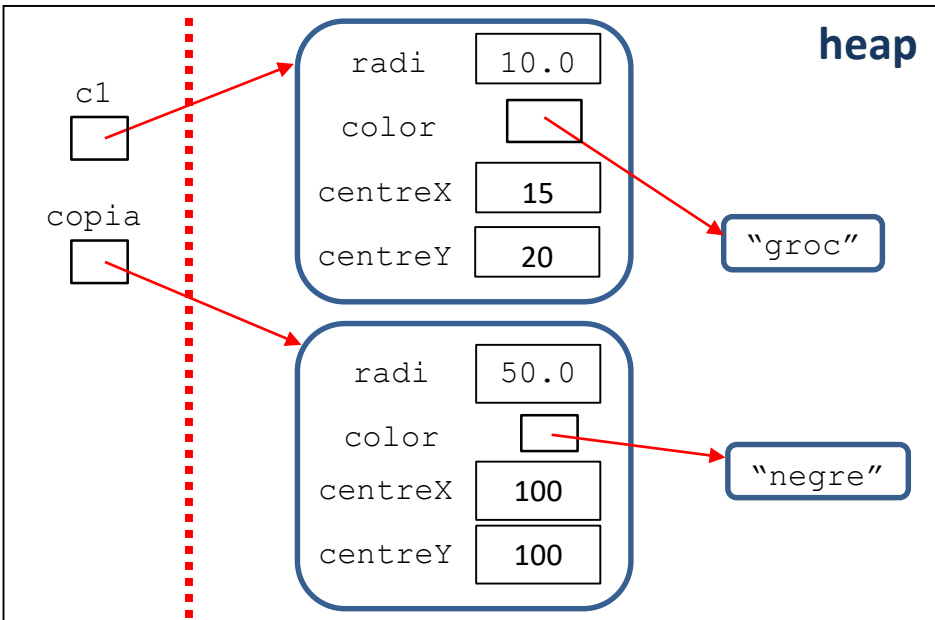
## Còpia d'objectes

```
// Crea un Cercle de radi 10, groc i centre en (15,20)
Cercle c1 = new Cercle(10, "groc", 15, 20);
// Crea un Cercle per defecte: de radi 50, negre i centre en (100,100)
Cercle copia = new Cercle();
// Còpia atribut a atribut
copia.setRadi(c1.getRadi());
copia.setColor(c1.getColor());
copia.setCentre(c1.getCentreX(), c1.getCentreY());
```

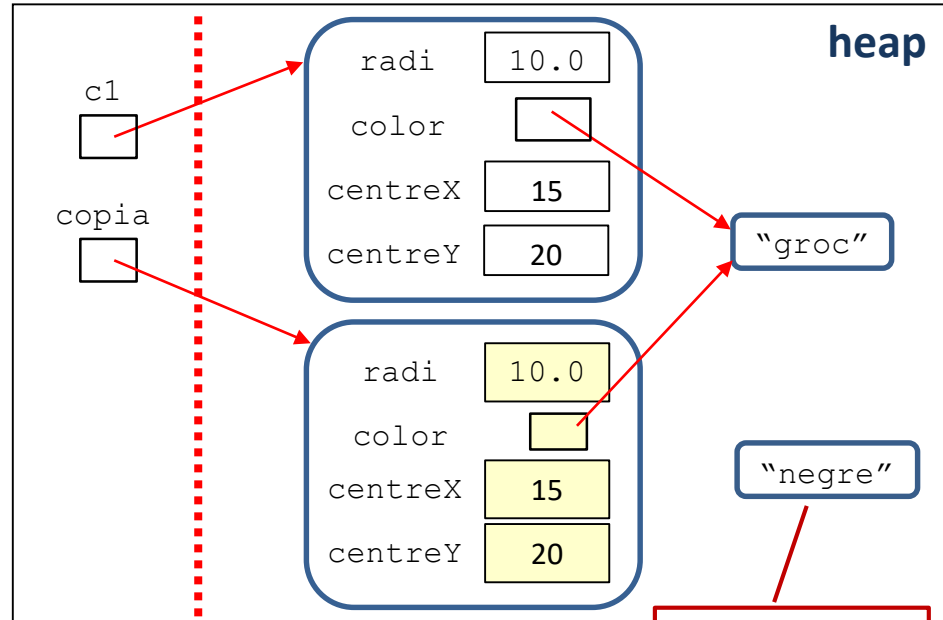
(1)

(2)

Memòria del sistema (1)



Memòria del sistema (2)



objecte  
desreferenciat

# Altres operacions sobre variables:

## Igualtat del valor de dues variables

- L'operador de **comparació d'igualtat** `==` és cert o fals segons siguin o no iguals els valors de les variables que es comparen.

BlueJ:exemplesT3

- **Executa** les següents instruccions en el **CodePad de BlueJ** del projecte. **Quin és el resultat?**

```
int x = 5, y = 9;
x = y;
y = x;

boolean esIgual = (x == y);
System.out.println("x es igual a y? " + esIgual);
boolean esDistint = (x != y);
System.out.println("x es distint de y? " + esDistint);
```

transpa 26

BlueJ: BlueJ: Finestra de terminal - exemplesT3

Opcions

```
x es igual a y? true
x es distint de y? false
```

# Altres operacions sobre variables:

## Igualtat del valor de dues variables

BlueJ:exemplesT3

- **Executa** les següents instruccions en el **CodePad de BlueJ** del projecte. **Quin és el resultat?**

```
int x = 5, y = 9;  
int aux = x;  
x = y;  
y = aux;  
boolean esIgual = (x == y);  
System.out.println("x es igual a y? " + esIgual);  
boolean esDistint = (x != y);  
System.out.println("x es distint de y? " + esDistint);
```

transpa 26

BlueJ: BlueJ: Finestra de terminal - exemplesT3

Opcions

```
x es igual a y? false  
x es distint de y? true
```

Can only enter input while your programming is

# Altres operacions sobre variables:

## Igualtat del valor de dues variables

- Quan l'operador `==` s'aplica a variables referència, tornarà `true` o `false` segons siguin o no iguals les referències contingudes.
- Si les referències són distintes, encara que els atributs de l'objecte que referencien tinguen el mateix valor, el resultat serà `false`.

BlueJ:exemplesT3

- **Executa** les següents instruccions en el **CodePad de BlueJ** del projecte.  
**Quin és el resultat?**

```
Punt x = new Punt(), y = new Punt(4, 4);  
y = x;  
boolean esIgual = (x == y);  
System.out.println("x es igual a y? " + esIgual);  
boolean esDistint = (x != y);  
System.out.println("x es distint de y? " + esDistint);
```

transpa 49

BlueJ: BlueJ: Finestra de terminal - exemplesT3

Opcions

```
x es igual a y? true  
x es distint de y? false
```

# Altres operacions sobre variables:

## Igualtat del valor de dues variables

BlueJ:exemplesT3

- **Executa** les següents instruccions en el **CodePad de BlueJ** del projecte. **Quin és el resultat?**

```
Punt x = new Punt(), y = new Punt(4, 4);
Punt aux = x;
x = y;
y = aux;

boolean esIgual = (x == y);
System.out.println("x es igual a y? " + esIgual);
boolean esDistint = (x != y);
System.out.println("x es distint de y? " + esDistint);
esIgual = (aux == x);
System.out.println("aux es igual a x? " + esIgual);
esIgual = (aux == y);
System.out.println("aux es igual a y? " + esIgual);
```

transpes 47  
i 48

BlueJ: BlueJ: Finestra de terminal - exemplesT3


Opcions

```
x es igual a y? false
x es distint de y? true
aux es igual a x? false
aux es igual a y? true
```

# Altres operacions sobre variables:


## Igualtat d'objectes

- Si el que es desitja és determinar la igualtat interna dels objectes, és necessari fer-ho mitjançant una **comparació** atribut a atribut

 BlueJ:exemplesT3

- **Executa** les següents instruccions en el **CodePad de BlueJ** del projecte. **Quin és el resultat?**

```
➤ Punt x = new Punt(), y = new Punt();
➤ boolean esIgual = (x == y);
➤ System.out.println("x es igual a y? " + esIgual);
➤ boolean esIgualAbs = (x.getAbs() == y.getAbs());
➤ boolean esIgualOrd = (x.getOrd() == y.getOrd());
➤ esIgual = (esIgualAbs && esIgualOrd);
➤ System.out.println("El punt x es igual a y? " + esIgual);
➤ y = new Punt(4, 4);
➤ esIgual = (x == y);
➤ System.out.println("x es igual a y? " + esIgual);
➤ esIgualAbs = (x.getAbs() == y.getAbs());
➤ esIgualOrd = (x.getOrd() == y.getOrd());
➤ esIgual = (esIgualAbs && esIgualOrd);
➤ System.out.println("El punt x es igual a y? " + esIgual);
➤
```

 BlueJ: BlueJ: Finestra de terminal - exemplesT3

Opcions

```
x es igual a y? false
El punt x es igual a y? true
x es igual a y? false
El punt x es igual a y? false
```



# Altres operacions sobre variables:

## Igualtat d'objectes

- Per tal que dos punts siguen iguals, les seues abscisses i ordenades han de ser-ho
- ➡ **Tema 4:** Millor definir el mètode especialitzat `equals` en la classe `Punt`
- Com `toString`, és un mètode predefinit en la classe `Object`

BlueJ:exemplesT3

- Executa** les següents instruccions en el **CodePad de BlueJ** del projecte. **Quin és el resultat?**

```
> Punt x = new Punt(), y = new Punt();  
> boolean esIgual = x.equals(y);  
> System.out.println("El punt x es igual a y? " + esIgual);  
> y = new Punt(4, 4);  
> esIgual = x.equals(y);  
> System.out.println("El punt x es igual a y? " + esIgual);  
>
```

BlueJ: BlueJ: Finestra de terminal - exemplesT3

Opcions

```
El punt x es igual a y? true  
El punt x es igual a y? false
```

# Modificadors final i static: constants i **variables de classe**

```
public class Cercle {  
    private double radi ;  
    private String color ;  
    private int centreX , centreY ;  
    ...  
    // mètodes de la classe  
}
```

## Atributs

- ***variables d'instància***

Emmagatzemen informació ***pròpia de cada objecte***

I si volem definir **atributs** que emmagatzemen informació ***COMUNA a tots els objectes d'una classe?***

↪ ***variables de classe***

Per exemple, per a la classe Cercle, el número de cercles estàndard (o per defecte) i no estàndard creats fins un moment donat.

# Modificadors final i static: constants i variables de classe

```
public class Cercle {  
    private static int numeroDeStd,  
                    numeroDeNoStd;
```

```
    private double radi;  
    private String color;  
    private int centreX, centreY;
```

```
    public Cercle() {  
        radi = 50.0; color = "negre"; centreX = 100; centreY = 100;  
        numeroDeStd++;  
    }
```

```
    public Cercle(double r, String c, int px, int py) {  
        radi = r; color = c; centreX = px; centreY = py;  
        numeroDeNoStd++;  
    }
```

```
    ...
```

```
}
```

## - *variables de classe*

Emmagatzemen informació *comuna*  
*a tots els objectes*

A nivell de memòria  
**TAMBÉ!!**

## Atributs

## - *variables d'instància*

Emmagatzemen informació *pròpia*  
*de cada objecte*

Blue:exemplesT3

- **Modifica** el codi de la classe **Cercle**
- **Crea** diversos objectes de tipus **Cercle** i **inspecciona**'ls per visualitzar què és una **variable de classe**.

# Modificadors final i static: constants i variables de classe

- Com que no estan associats a cap objecte (pertanyen a la classe):
  - El sistema els assigna memòria la primera vegada que s'executa codi de la classe (la primera vegada que es crea un objecte de la mateixa).
  - La memòria associada a les variables de classe continua en ús fins que la classe deixa d'estar **carregada** en memòria, normalment en finalitzar el programa que la utilitza.
- Si la variable de classe és accessible (no és privada), per referenciar-la des d'una altra classe s'utilitza la forma:

```
NomDeClasse.nomDAtributDeClasse
```

- Des de la classe ProvaCercle, si numeroDeStd fora public (en lloc de private) se podria accedir així:

```
Cercle.numeroDeStd
```

# Modificadors `final` i `static`: `constants` i variables de classe

- I si volem definir **atributs** *auxiliars* per a NO usar “números màgics”?

Per exemple, per a la classe `Cercle`, `3.14` és un número màgic –literal de tipus `double`– que s'utilitza en alguns mètodes com valor aproximat del número  $\pi$ .

[Què és un "número màgic" \(\*magic number\*\) en Programació?](#)

```
public class Cercle {  
    public final double PI_APROX = 3.14; immutable  
  
    private double radi;  
    private String color;  
    private int centreX, centreY;  
  
    // Mètodes de la classe  
    // reemplaçar 3.14 per PI_APROX en area() i perimetre()  
    public double area() { return PI_APROX * radi * radi; }  
    public double perimetre() { return 2 * PI_APROX * radi; }  
    ...  
}
```

**Atributs** - *variables d'instància*

# Modificadors `final` i `static`: `constants` i variables de classe

- El modificador `final` permet definir variables **immutable**, i.e. el seu valor **NO pot canviar**.

 **constants**

```
double area, radi; double PI = 3.14159;  
radi = 12.0;  
area = PI * radi * radi;  
PI = 3.1416;
```

variable

```
double area, radi; final double PI = 3.14159;  
radi = 12.0;  
area = PI * radi * radi;  
PI = 3.1416;
```

constant

cannot assign a value to final variable  
PI

# Modificadors final i static: constants i variables de classe

```
public class Cercle {  
    public static final double PI_APROX = 3.14;  
  
    private double radi ;  
    private String color ;  
    private int centreX , centreY ;  
  
    // Mètodes de la classe  
    // reemplaçar 3.14 per PI_APROX en area() i perimetre()  
    public double area() { return PI_APROX * radi * radi; }  
    public double perimetre() { return 2 * PI_APROX * radi; }  
    ...  
}
```


**Atributs**

- *variable d'instància*

- *variable de classe*

*immutable*

A nivell de memòria  
Té sentit?

 BlueJ:exemplesT3

- **Modifica** el codi de la classe **Cercle**
- **Crea** diversos objectes de tipus **Cercle** i **inspecciona**'ls per visualitzar la constant **PI\_APROX**.
- **Afegeix** a la classe **ProvaCercle** una instrucció que mostre per pantalla el valor de **PI\_APROX**. **Executa**-la.
- **Afegeix** a la classe **ProvaCercle** una instrucció que li assigne a **PI\_APROX** el valor 3.1416. **Què ocorre?**

# La classe Math

*Referència:*

<https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/Math.html>

- **Capítol 5 – Secció 5.2** del llibre de l'assignatura.
- En PoliformaT, [IIP: recursos](#) / [Profesores](#) / [Llorens Agost, Marisa. Grup 1B](#) / [Material propi](#) / Tema 3  
transpes: [La classe Math.pdf](#)  
vídeo: [La-classe-Math.mp4](#)



# La classe String

*Referència:*

<https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/String.html>

- **Capítol 5 – Secció 5.1** del llibre de l'assignatura.
- En PoliformaT, [IIP: recursos](#) / [Profesores](#) / [Llorens Agost, Marisa. Grup 1B](#) / [Material propi](#) / Tema 3  
transpes: [La classe String.pdf](#)  
vídeos: [La-classe-String-1.mp4](#) i [La-classe-String-2.mp4](#)

# La classe Scanner

*Referència:*

<https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Scanner.html>

- **Capítol 6 – Secció 6.2** del llibre de l'assignatura.
- En PoliformaT, [IIP: recursos](#) / [Profesores](#) / [Llorens Agost, Marisa. Grup 1B](#) / [Material propi](#) / Tema 3  
transpes: [Entrada i eixida elemental.pdf](#)  
codi: [exemples – Scanner.jar](#)  
vídeos: [Entrada-i-eixida-elemental-1.mp4](#) i [Entrada-i-eixida-elemental-2.mp4](#)