



UNIVERSITAT  
POLITÈCNICA  
DE VALÈNCIA

Escola Tècnica Superior d'Enginyeria Informàtica



# Tema 4. Mètodes: definició, tipus i ús en Java

Introducció a la Informàtica i a la Programació (IIP)

Curs 2019/20

Departament de Sistemes Informàtics i Computació



# Continguts

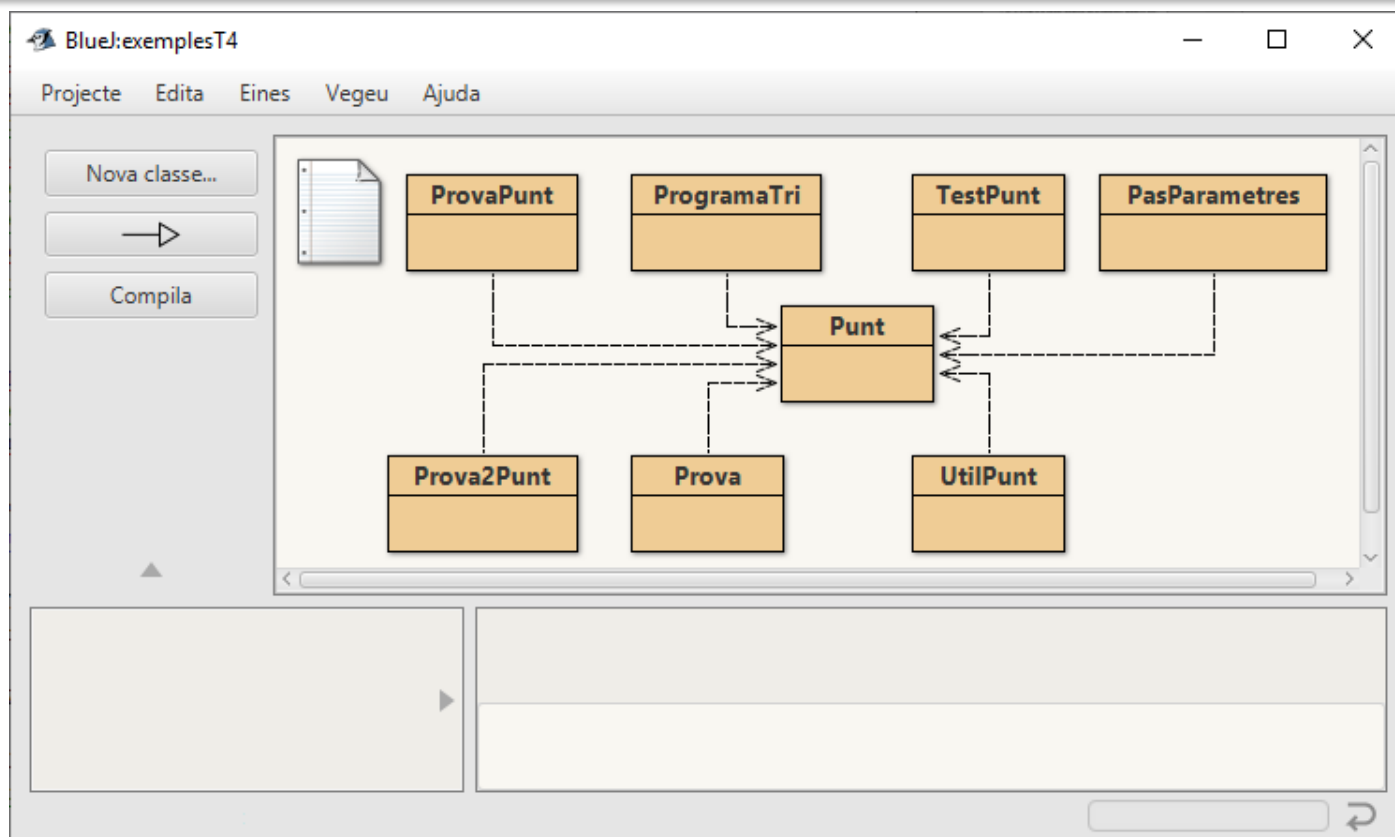
Duració: 4 sessions

1. Mètodes en Java
  - Mètodes dinàmics vs estàtics (constructors, d'instància, main i altres)
  - Tipus de classes segons el tipus dels mètodes que contenen (Programa, Tipus de Dades i d'Utilitats)
2. Declaració i ús d'un mètode segons el seu tipus i especificació
  - Perfil d'un mètode: identificador, modificadors, tipus del resultat i paràmetres formals
  - Cos d'un mètode: instrucció return
  - Ús d' (o crida a) un mètode. Arguments. Pas de paràmetres. Objecte en curs i referència this
  - Documentació
3. Sobrecàrrega i sobreescritura d'un mètode
  - Sobrecàrrega de mètodes i variables en una classe (Principi de Màxima Proximitat)
  - Sobreescritura dels mètodes toString i equals d'Object
4. Execució d' (una crida a) un mètode
  - Registre d'activació. Pas de paràmetres per valor. Traces
  - Pila de crides

- Per tal que pugues practicar amb els conceptes que s'introdueixen en aquesta sessió i respondre les qüestions que se't formulen des d'ara endavant ...



- **Crea** una carpeta **Tema 4** dins de la teua carpeta **W:\IIP\**
- **Descarrega** (del Tema 4 de PoliformaT) el fitxer **exemplesT4.jar** en **Tema 4**
- Des de l'opció **Projecte** de **BlueJ**, usa l'opció **Open ZIP/JAR...** per tal d'obrir-lo com un projecte **BlueJ** i prepara't per usar-lo
- **Atenció:** la classe **Punt** que apareix en el projecte és una versió de la classe que vam usar en el tema anterior

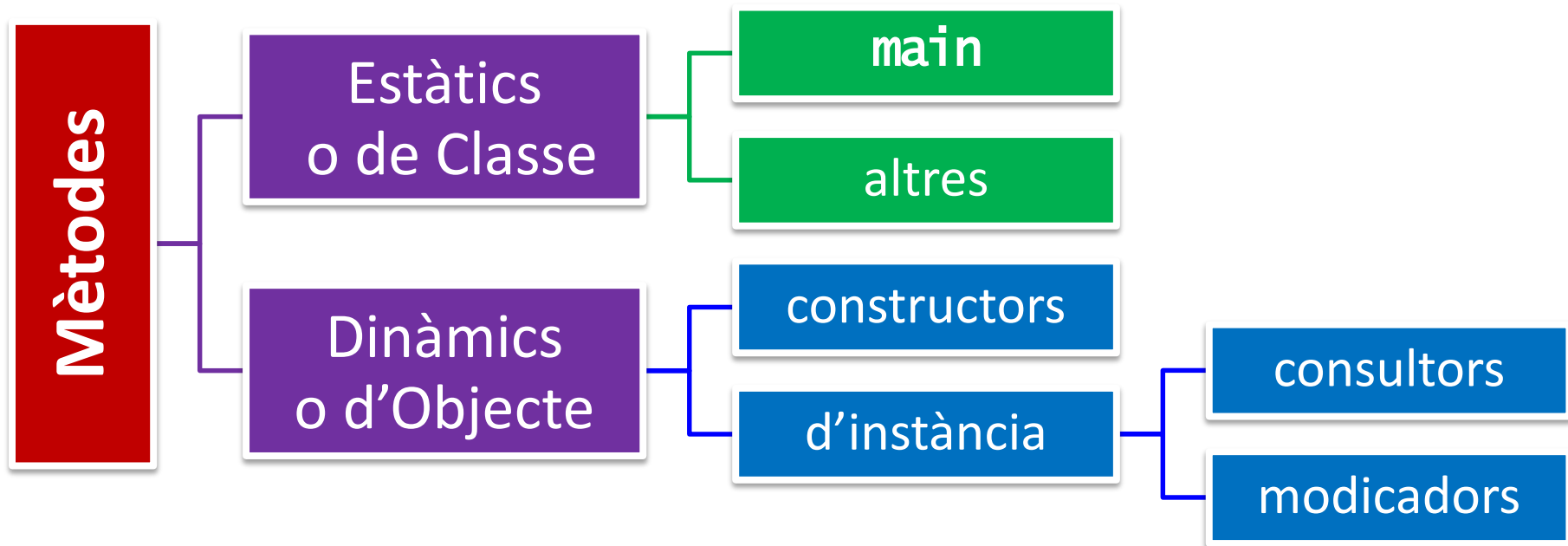


# Mètodes en Java: definició i tipus

- Un **mètode Java** és un bloc d'instruccions que defineix una funcionalitat:
  - Sobre un objecte d'una classe:
    - Crear un objecte: constructors
    - Consultar la seua informació: consultors (*gets*)
    - Modificar el seu estat: modificadors (*sets*)
    - Altres operacions
  - Sobre tota la classe (sense associació a un objecte concret):
    - Llançar una aplicació: *main*
    - Càlculs matemàtics: *sqrt(x)*, *pow(x, y)*, etc.
    - Qualsevol operació que requerisca, en general, unes dades i torne un resultat. Per exemple, donats dos objectes de tipus *Cercle*, calcular l'àrea de la seua intersecció.

# Mètodes en Java

## mètodes dinàmics vs estàtics



# Mètodes en Java

## tipus de classes segons els mètodes que contenen

Classe	Mètodes dinàmics o d'objecte			Mètodes estàtics o de classe	
	Constructors	Consultors	Modificadors	main	altres
Tipus de Dades	sí (public)	sí	sí	no	sí
Programa	sí (private)	no	no	sí	sí
d'Utilitats	sí (private)	no	no	no	sí

➡ **Agrupen mètodes estàtics d'utilitat general** sobre tipus prèviament definits, per la qual cosa **no** tenen atributs d'instància, només estàtics, com PI en Math

- El constructor per defecte crea objectes buits (per això se sol fer-lo **private**)
- És habitual **agrupar en paquets** (packages) classes d'utilitats que prestin funcionalitats relacionades d'alguna manera. Per exemple, **Math**, **String** i altres classes d'ús molt freqüent s'agrupen en el paquet `java.lang`
- Exemples de Classes d'Utilitats **d'usuari**: la classe **IIPMath** de la pràctica 6; la classe **MiLibreria** dels exercicis CAP Redondeo i Aleatorio en un interval; la classe **UtilPunt** (del llibre de l'assignatura) en el projecte BlueJ **exemplesT4**

# Declaració i ús d'un mètode estructura bàsica

```
[mod. visibilitat] [altres] tipusRetorn nomMetode([llistaParams]) {
```

Capçalera (o perfil)

```
[[final] tipus nomVarLocal1;
 [final] tipus nomVarLocal2;
 ... ..
 [final] tipus nomVarLocalN;]

[instrucció1;
 ... ;
 instruccióM;]
[return [expressióTipus];]
```

Cos

```
}
```

Accés a un element amb <b>modificador de visibilitat</b> des de	la classe on està definit	una classe del mateix paquet	altres classes
<b>private</b>	sí	<b>no</b>	<b>no</b>
<b>public</b>	sí	sí	sí
<b>sense especificar (friendly)</b>	sí	sí	<b>no</b>

# Declaració i ús d'un mètode capçalera o perfil segons el tipus de mètode

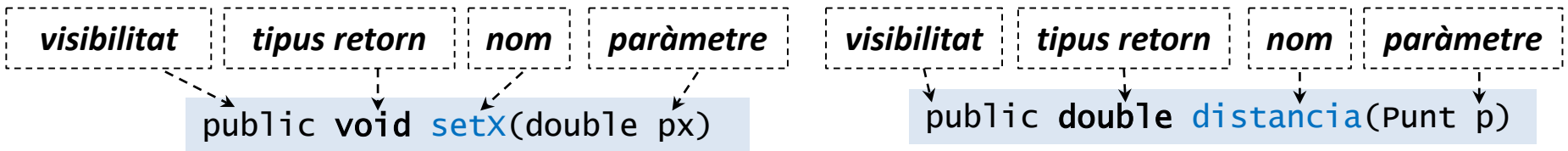
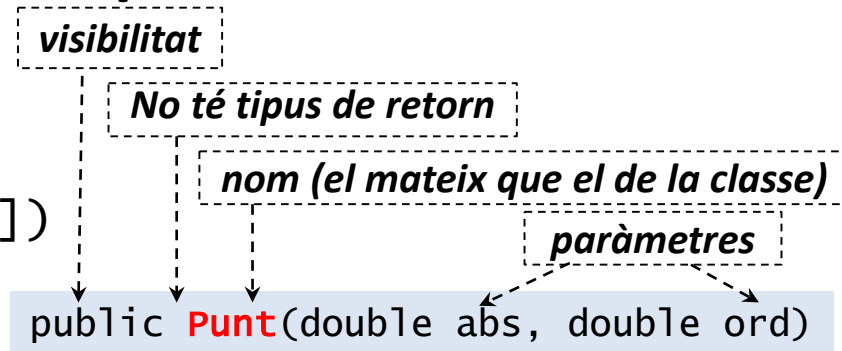
- Mètodes dinàmics o d'objecte

## Constructors:

[mod.visib] NomClasse([LlistaParams])

## D'instància:

[mod.visibilitat] TipusRetorn nomMètode([LlistaParams])



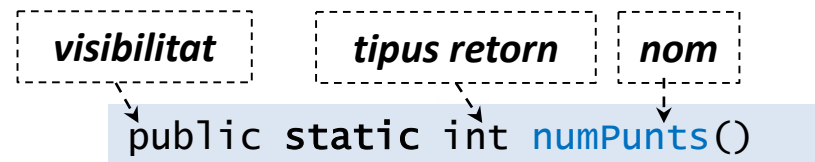
- Mètodes estàtics o de classe

**main:** `public static void main(String[] args)`



## Altres:

[mod.visibilitat] **static** TipusRetorn nomMètode([LlistaParams])





# Declaració i ús d'un mètode

## mètodes estàtics i dinàmics en classes predefinides

- Classe [String](#)

**String()**

Initializes a newly created String object so that it represents an empty character sequence.

⋮

**String(String original)**

Initializes a newly created String object so that it represents the same sequence of characters as the argument; in other words, the newly created string is a copy of the argument string.

int

**indexOf(int ch)**

Returns the index within this string of the first occurrence of the specified character.

int

**indexOf(int ch, int fromIndex)**

Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index.

⋮

⋮

**static String**

**valueOf(double d)**

Returns the string representation of the double argument.

- Classe [Math](#)

**static double**

**pow(double a, double b)**

Returns the value of the first argument raised to the power of the second argument.

**static double**

**random()**

Returns a double value with a positive sign, greater than or equal to 0.0 and less than 1.0.

# Declaració i ús d'un mètode

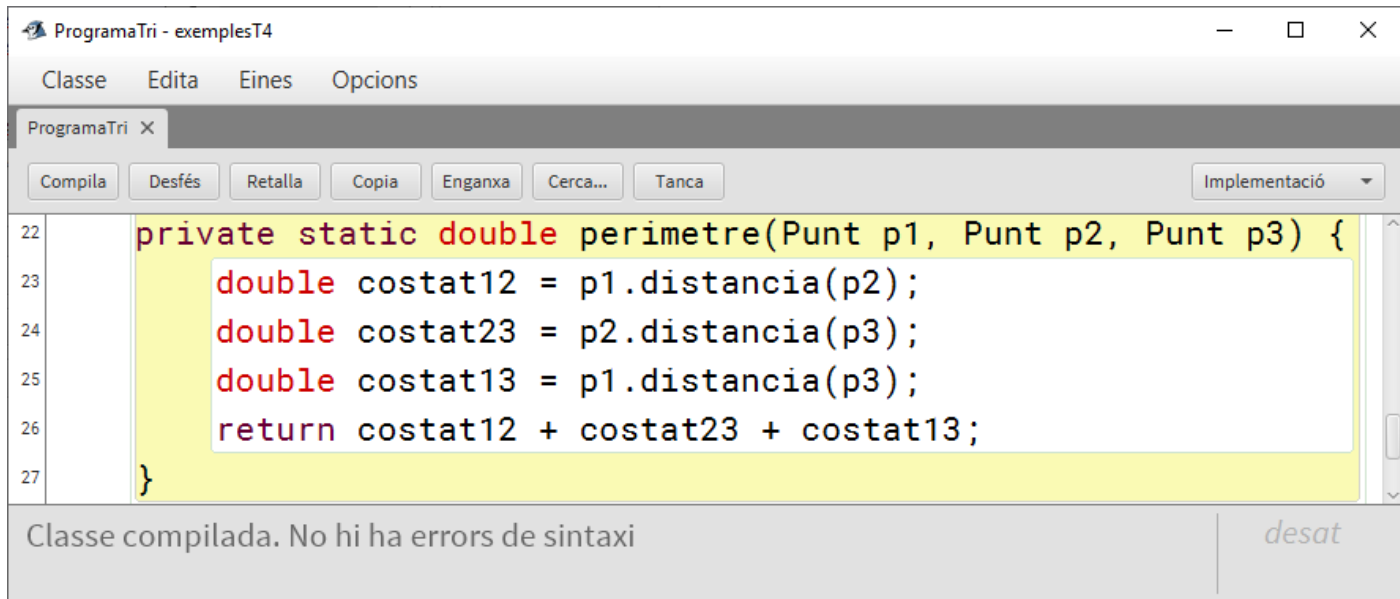
## cos del mètode i instrucció return

- En qualsevol mètode, el cos inclou:
  - La **declaració de variables locals** que siguin necessàries
  - La **seqüència d'instruccions** que implementen la seua funcionalitat
- Excepte els mètodes constructors, el main i qualsevol altre mètode que indique que el resultat que torna és void, el cos del mètode ha d'incloure una **instrucció return**:

**return expressió;**

on expressió ha de ser compatible amb el tipus de retorn que explicita la capçalera del mètode.

- Quan s'executa aquesta instrucció, l'execució del mètode es dona per finalitzada.



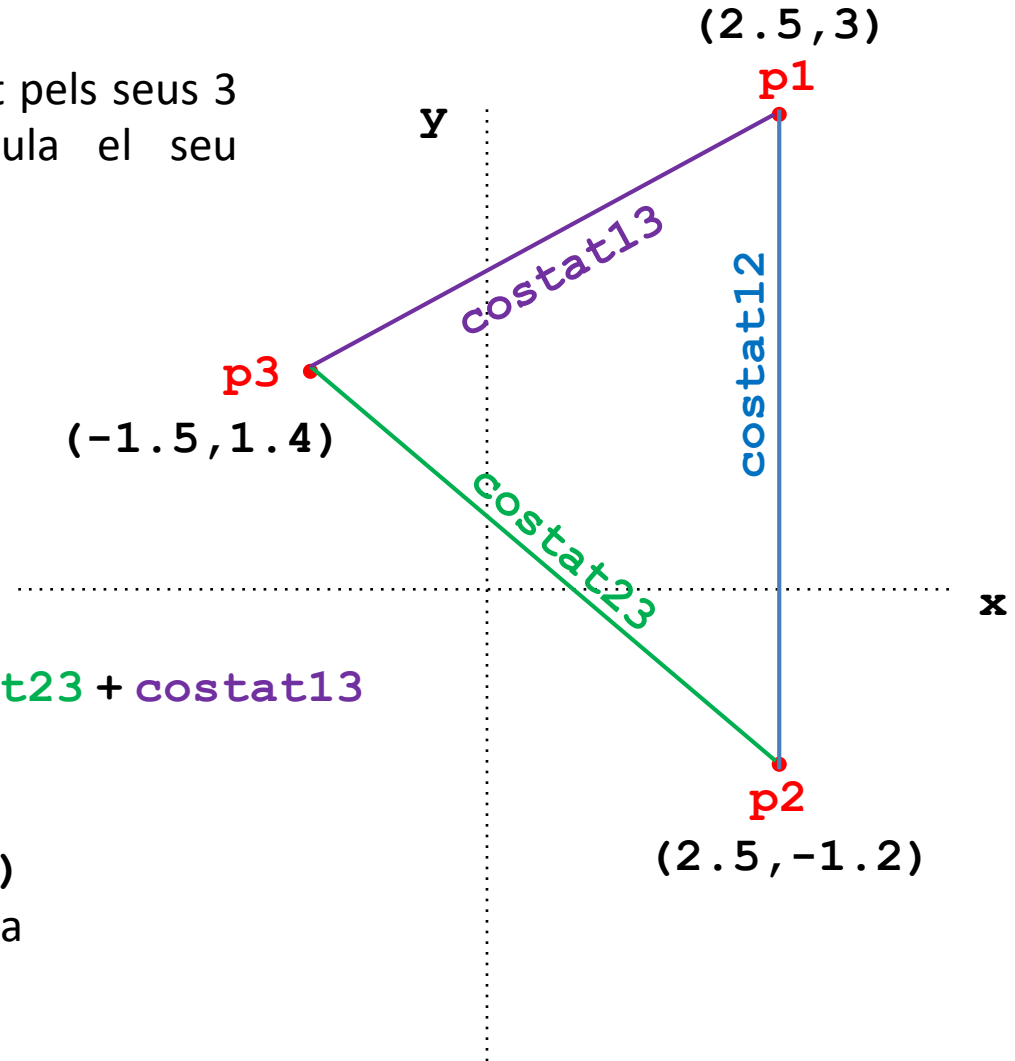
The screenshot shows a Java IDE window titled "ProgramaTri - exemplesT4". The menu bar includes "Classe", "Edita", "Eines", and "Opcions". The toolbar contains buttons for "Compila", "Desfés", "Retalla", "Copia", "Enganxa", "Cerca...", "Tanca", and a dropdown for "Implementació". The code editor displays the following Java code:

```
22 private static double perimetre(Punt p1, Punt p2, Punt p3) {  
23     double costat12 = p1.distancia(p2);  
24     double costat23 = p2.distancia(p3);  
25     double costat13 = p1.distancia(p3);  
26     return costat12 + costat23 + costat13;  
27 }
```

The status bar at the bottom indicates "Classe compilada. No hi ha errors de sintaxi" and "desat".

# Quin problema resol ProgramaTri?

Donat un triangle al pla cartesià, definit pels seus 3 vèrtexs (punts **p1**, **p2** i **p3**), calcula el seu **perímetre** (suma dels seus 3 costats).



$$\text{perímetre} = \text{costat12} + \text{costat23} + \text{costat13}$$

La distància entre dos punts  $(a_1, b_1)$  i  $(a_2, b_2)$  es pot calcular aplicant la fórmula:  $\sqrt{(a_1 - a_2)^2 + (b_1 - b_2)^2}$

# Declaració i ús d'un mètode

## cos del mètode: informació accessible segons el tipus del mètode

Objecte en curs  
(this)

Variable segons el seu rol	Mètodes dinàmics o d'objecte			Mètodes estàtics o de classe	
	Constructors	Consultors	Modificadors	main	altres
Atributs o variables d'instància	sí	sí	sí	no	no
Atributs o variables de classe	sí	sí	sí	sí	sí
Variables locals	sí	sí	sí	sí	sí
Paràmetres	sí	sí	sí	sí	sí

# Declaració i ús d'un mètode

## crida a un mètode segons el seu tipus

- La crida o invocació a un mètode es fa utilitzant els seu **identificador** o nom del mètode, seguit de la **llista d'arguments** entre parèntesi.

`nomMetode(arg1, arg2, ...)`

- Els paràmetres del mètode s'inicialitzen amb els valors dels arguments (**pas de paràmetres**).
  - Ha d'haver **concordança** en el número, tipus i ordre d'aparició de paràmetres i arguments.
- Si el mètode s'invoca **des de fora de la classe**:
    - Si és un **mètode estàtic**, ha d'anar precedit pel nom de la classe i l'operador punt: `Math.sqrt(x)`
    - Si és un **constructor**, ha d'usar-se amb l'operador `new`: `new Cercle()`
    - Si és un **mètode d'instància** ha d'anar precedit de la referència a l'objecte sobre el qual s'aplica i de l'operador punt: `c1.getRadi()`, `c1.setRadi(10)`

# Declaració i ús d'un mètode

## crida a un mètode: paràmetres formals i arguments

### Definició

```
public class Punt {  
    ...  
    public Punt(double abs, double ord) {  
        x = abs; y = ord;  
        comptador++;  
    }  
    ...  
}
```

**Paràmetres formals**

### Ús

```
public static void main(String[] args) {  
    ...  
    Punt p = new Punt(2.5, 3);  
    ...  
}
```


**Arguments**

**PAS DE PARÀMETRES**

En la invocació a un mètode, inicialització dels **paràmetres formals** amb els **valors** dels **arguments** o **paràmetres reals**

⇒ Pas de paràmetres **PER VALOR**

**Atenció:** Ha d'haver **concordança** en quant a nombre, tipus i ordre d'aparició entre els **paràmetres formals** i els **arguments**

 BlueJ:exemplesT4

- **Estableix** un **punt de ruptura** en la primera línia del cos del constructor general i del mètode moure de la classe **Punt** del projecte **BlueJ exemplesT4**.
- **Executa**, des del *Code Pad* de **BlueJ**, la instrucció `Punt p = new Punt(2.5, 3);` i en detindre's l'execució en el punt de ruptura, **observa**, en la finestra del depurador, que els **paràmetres formals** **abs** i **ord** s'han inicialitzat amb els valors dels **arguments** **2.5** i **3**, respectivament.
- **Fes clic** sobre el botó **Pas** de la finestra del depurador i **observa** el contingut de les zones **Variables** estàtiques, **Variables d'instància** i **Variables locals**.
- **Executa**, des del *Code Pad* de **BlueJ**, les instruccions `int x = 3; p.moure(x, x + 1);`. A quins valors s'han inicialitzat els **paràmetres formals** **px** i **py**?
- **Fes clic** sobre el botó **Pas** i **observa** com canvien les **variables d'instància** del **Punt p**.

# Declaració i ús d'un mètode

## crida a un mètode segons el tipus del resultat

- En funció del **tipus del resultat** que torna un mètode:
  - Si el mètode torna un resultat explícitament, la **invocació** al mètode pot apareixer en qualsevol context on s'espere un resultat del mateix tipus (o compatible). Per exemple:

```
Punt p = new Punt();  
System.out.println("L'abscissa del punt és " + p.getX());  
double d = p.distOrigen();  
System.out.println("La distància del punt a l'origen és " + d);  
Punt q = Punt.polarsAPunt(p.distOrigen(), 0.5);
```

- Si el mètode NO torna un resultat explícitament, i.e., el tipus del resultat és void: la **invocació** al mètode és una instrucció més. Per exemple:

```
Punt p = new Punt(2.5, 3);  
int x = 3;  
p.moure(x, x + 1);
```


# Exercicis de declaració i ús de mètodes estàtics

## 1. Se DECLAREN com **static** ...

- **main**, el representant d'aquest tipus de mètodes, **NOMÉS** en una classe Programa: el seu nom i la resta de la seua capçalera estan **predefinitos**, i.e., son immutables
- **Qualsevol altre**, en **QUALSEVOL** tipus de classe

## 2. **NO** actuen sobre UN OBJECTE de la seua classe sino, **podem dir, sobre TOTA** la seua classe

- **bé actuen sobre els seus atributs estàtics**, amb informació comuna a tots els seus objectes
- **bé NO existeix objecte al qual aplicar-los**, com en el cas de **main**, que invoca PER DEFECTE la JVM en executar el comandament `java NomClasse`

 BlueJ:exemplesT4

- **Edita** la classe **Punt** del projecte i **respon...**

1. Quants mètodes **estàtics** té? Quins són els seus **identificadors**?
2. Quines **variables** de la classe són **accessibles** des de cadascun d'ells?
3. Què **fa** cadascun dels mètodes estàtics de la classe?
4. Quina (única) **expressió** permet saber el **nº d'objectes** de tipus **Punt** creats en el projecte fins el moment? **Escriu-la** en el *Code Pad*
5. Quina (única) **expressió** crea un objecte de tipus **Punt** a partir de les coordenades polars (0.79, 5)? **Escriu-la** en el *Code Pad* del projecte, situa l'objecte resultant de la seua avaluació l'*Object Bench* i **inspecciona'l**

- **Edita** la classe **ProgramaTri** del projecte i **respon** les qüestions anteriors de la 1 a la 3. Després, **invoca al seu mètode main**




# Exercicis de declaració i ús de mètodes dinàmics

## 1. NO se DECLAREN MAI com `static` ...

- **Constructors**, els representants d'aquest tipus de mètodes: excepte el seu modificador de visibilitat, la seua capçalera està **predeterminada**, proporcionant Java un constructor per defecte –sense paràmetres– per a qualsevol classe
- **D'instància** (consultors i modificadors), **NOMÉS** en una **classe Tipus de Dades**

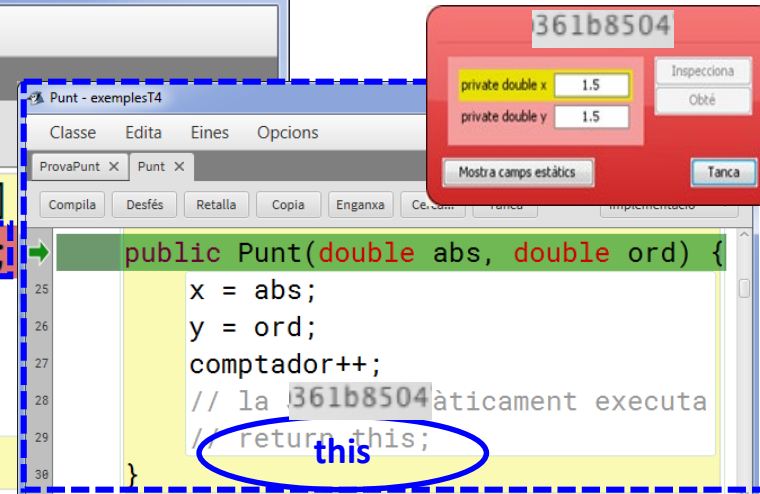
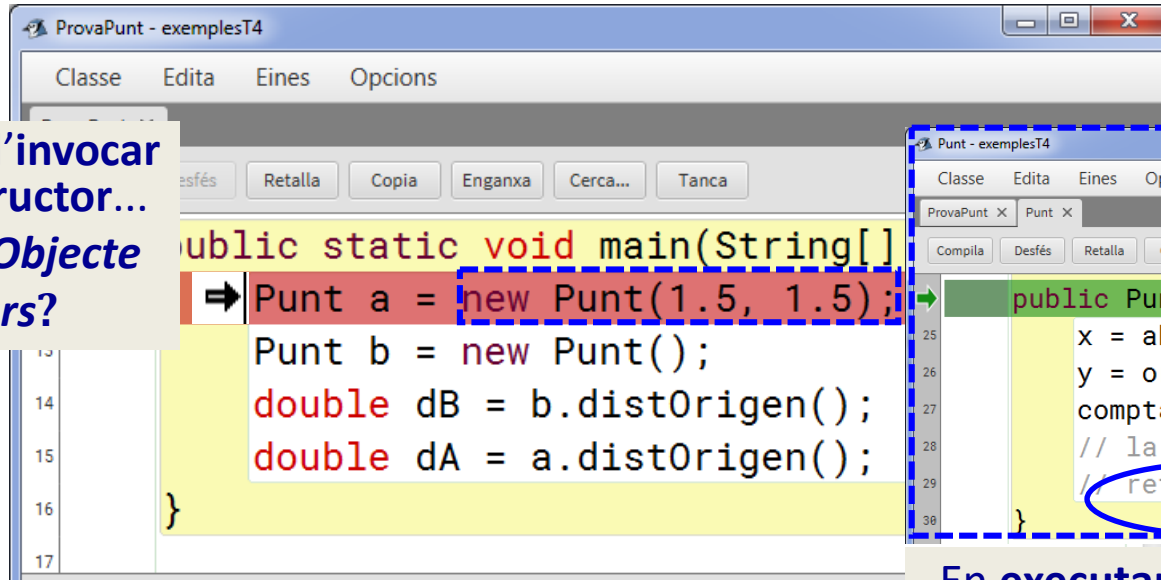
## 2. SÍ actuen sobre UN OBJECTE de la seua classe, l'anomenat **objecte en curs o actual**

- **bé creant un objecte de la classe existent via operador `new`** (constructors), inicialitzant les seues variables d'instància
- **bé s'apliquen sobre un objecte existent via operador `.`** (mètodes d'instància)  
**Atenció:** per evitar errors l'objecte en curs ha de crear-se prèviament

 BlueJ:exemplesT4

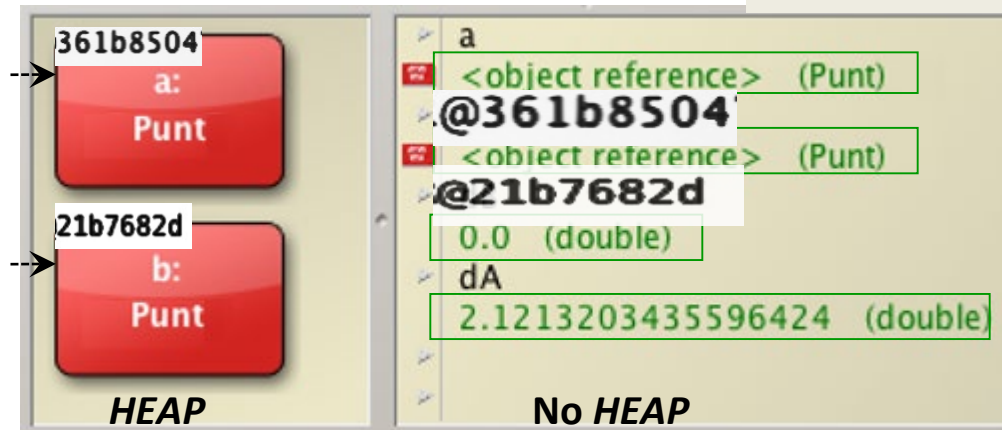
- **Edita** la classe **Punt** del projecte, busca el mètode `moureAleat` i **respon...**
  1. **Quines dues raons** fan d'ell un mètode **dinàmic**?
  2. **Quines variables** de la classe són **accessibles** des d'ell?
  3. **Què fa?** Aleshores, **crea, modifica o consulta l'estat** d'un punt?
  4. Quina seqüència d'**instruccions** permet **crear** el punt (2.0, 3.0) i desplaçar-lo de forma aleatòria? **Escriu-les** en el *Code Pad* del projecte, **situa** el punt nou en l'*Object Bench* i **inspecciona'l** abans i després d'haver-lo desplaçat aleatòriament

# Declaració i ús d'un mètode objecte en curs - referència this per a mètodes dinàmics



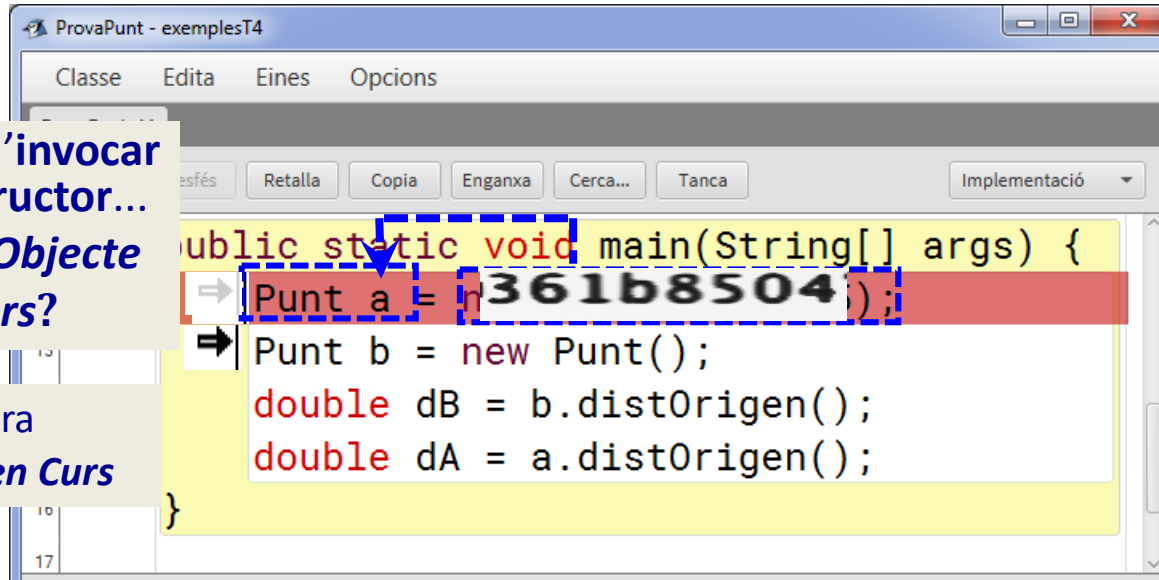
En executar un mètode dinàmic ...  
**this** és (referència a)  
**l'Objecte en Curs**

- Després d'executar la darrera instrucció del main de ProvaPunt de *exemplesT4*, l'estat de la memòria és el següent:



**this** és una referència del tipus de la classe, per anomenar a l'objecte actual u objete en curs. És una variable **final**

# Declaració i ús d'un mètode objecte en curs - referència this per a mètodes dinàmics

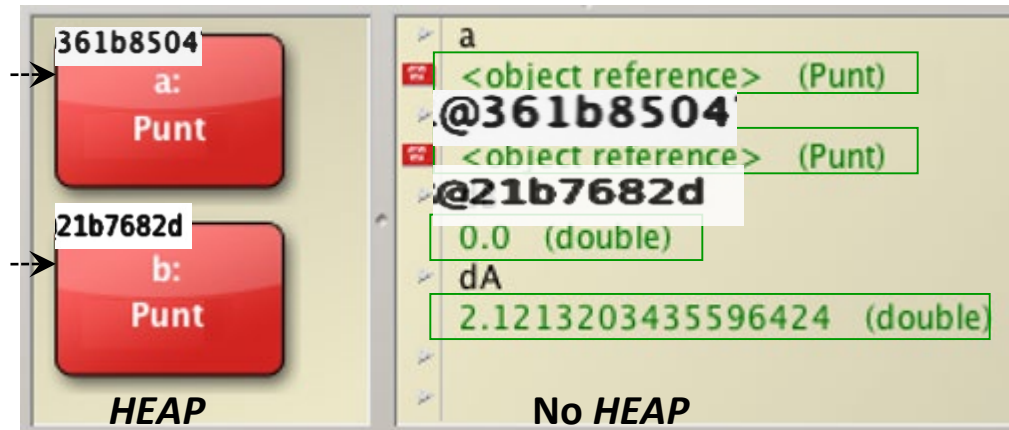


```
public static void main(String[] args) {  
    Punt a = new Punt(361b8504);  
    Punt b = new Punt();  
    double dB = b.distOrigen();  
    double dA = a.distOrigen();  
}
```

Després d'invocar  
a un constructor...  
Quin és l'**Objecte**  
en Curs?

a és ara  
l'**Objecte en Curs**

- Després d'executar la darrera instrucció del main de ProvaPunt de **exemplesT4**, l'estat de la memòria és el següent:



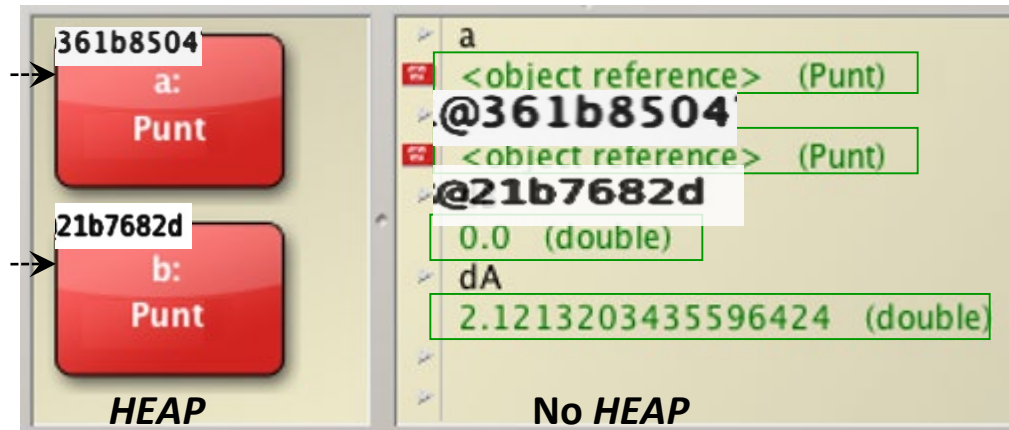
# Declaració i ús d'un mètode objecte en curs - referència this per a mètodes dinàmics

```
ProvaPunt - exemplesT4
Classe  Edita  Eines  Opcions
ProvaPunt x Punt x
sfés  Retalla  Copia  Enganxa  Cerca...  Tanca  Implementació
public static void main(String[] args)
{
    Punt a = new Punt(1.5, 1.5);
    Punt b = new Punt(2.1, 1.7);
    double dB = b.distOrigen();
    double dA = a.distOrigen();
}
```

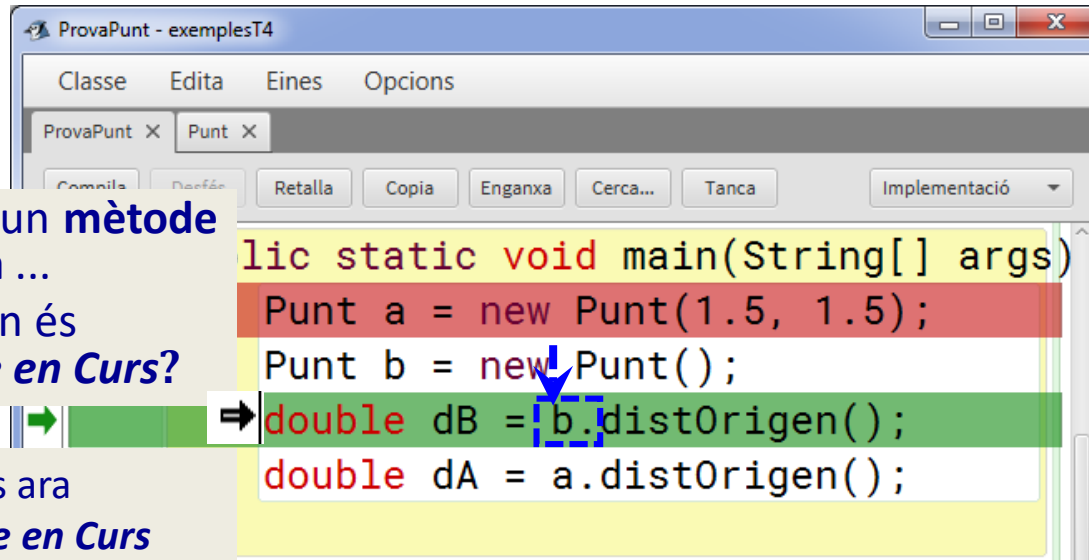
Després d'invocar  
a un constructor...  
Quin és l'**Objecte**  
en Curs?

**b** és ara  
l'**Objecte en Curs**

- Després d'executar la darrera instrucció del main de ProvaPunt de **exemplesT4**, l'estat de la memòria és el següent:



# Declaració i ús d'un mètode objecte en curs - referència this per a mètodes dinàmics



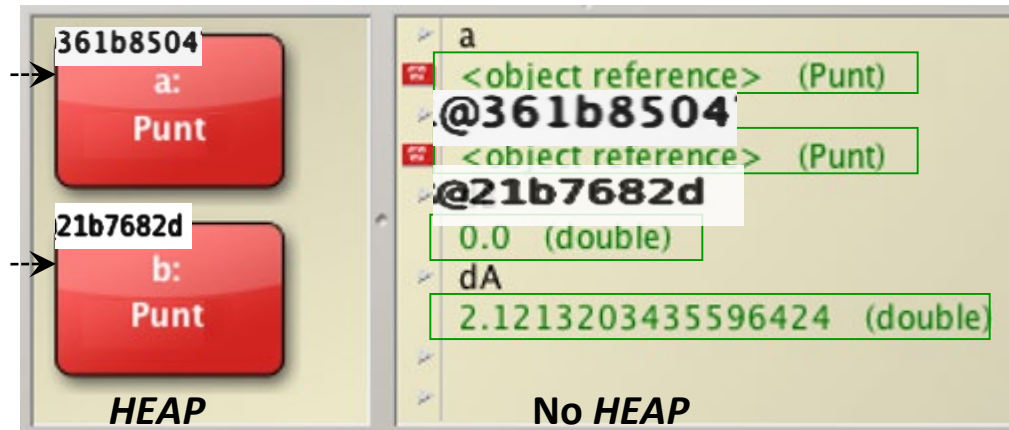
```
public static void main(String[] args)
{
    Punt a = new Punt(1.5, 1.5);
    Punt b = new Punt();
    double dB = b.distOrigen();
    double dA = a.distOrigen();
}
```

A l'aplicar un mètode  
d'instància ...

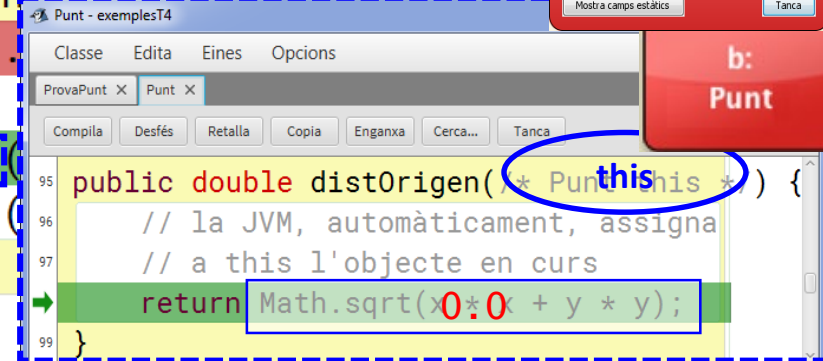
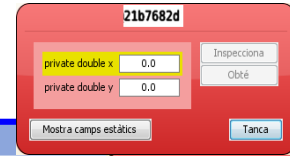
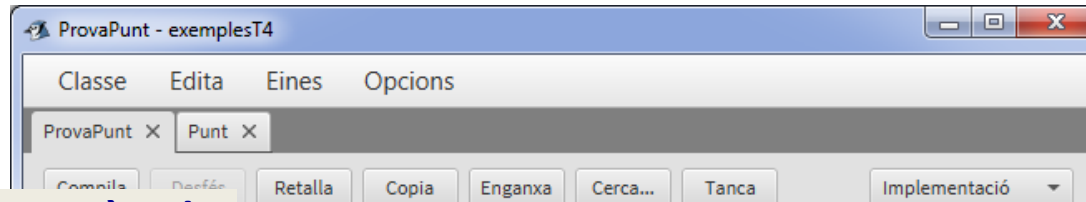
Quin és  
l'Objecte en Curs?

**b** és ara  
l'Objecte en Curs

- Després d'executar la darrera instrucció del main de ProvaPunt de *exemplesT4*, l'estat de la memòria és el següent:



# Declaració i ús d'un mètode objecte en curs - referència this per a mètodes dinàmics

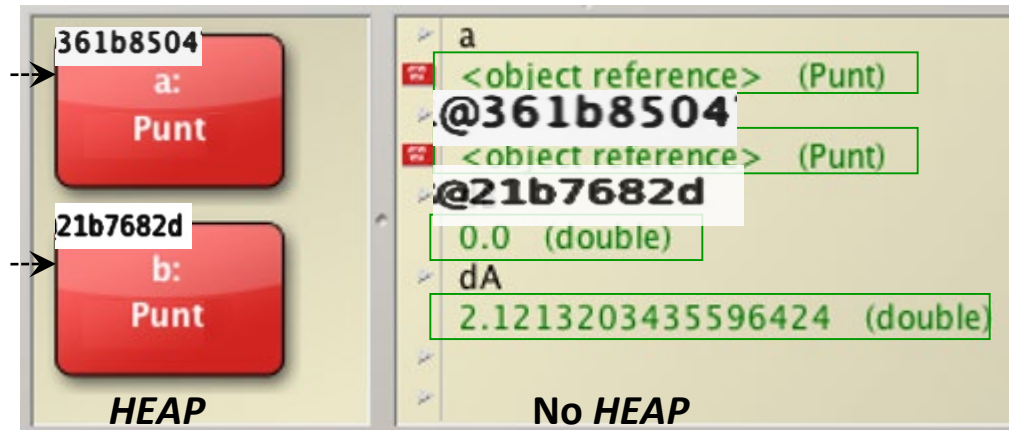


A l'aplicar un mètode d'instància ...

Quin és l'Objecte en Curs?

b és ara l'Objecte en Curs

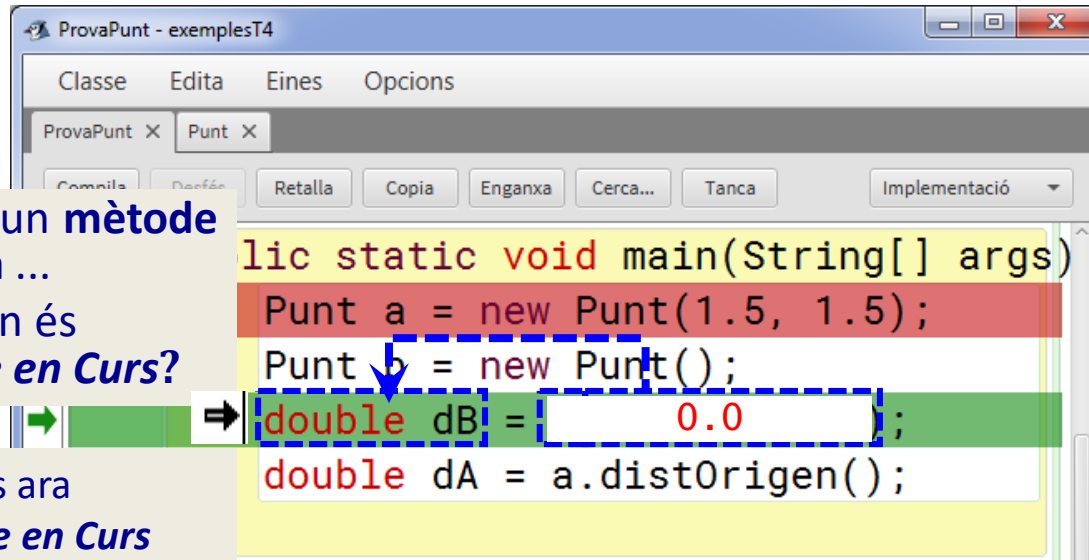
- Després d'executar la darrera instrucció del main de ProvaPunt de *exemplesT4*, l'estat de la memòria és el següent:



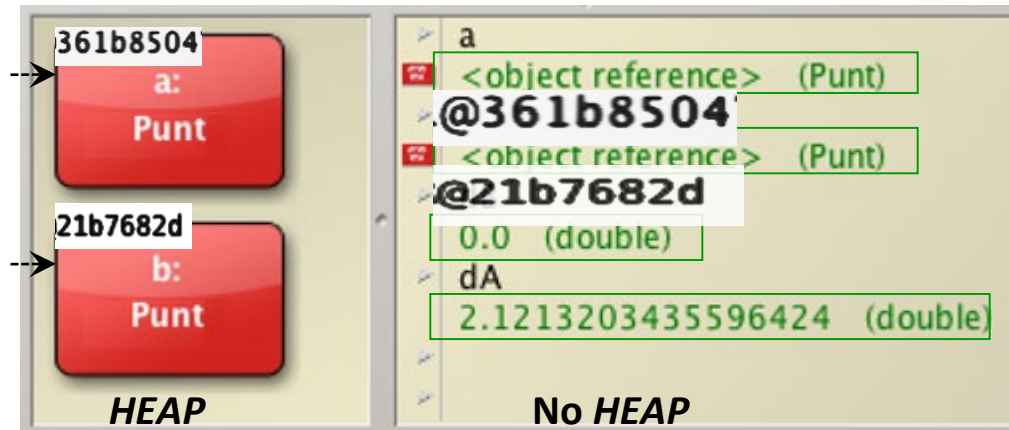
En executar un mètode dinàmic ...  
**this** és (referència a) l'Objecte en Curs



# Declaració i ús d'un mètode objecte en curs - referència this per a mètodes dinàmics




- Després d'executar la darrera instrucció del main de ProvaPunt de *exemplesT4*, l'estat de la memòria és el següent:



# Declaració i ús d'un mètode

## invocació des d'un mètode **de la classe** a un altre de la mateixa **classe**

```
public class Punt {  
    private double x, y;  
    ...  
    public double distOrigen() { return Math.sqrt(x * x + y * y); }  
    private double aleatori() {  
        return Math.random() * (distOrigen() + 1);  
    }  
    ...  
}
```

 BlueJ:exemplesT4

**Observa** el cos del mètode **aleatori** de la classe **Punt** i **respon...**

- 1. Quins dos mètodes usa?**
- 2. Quin d'ells és dinàmic? De quina classe és? Sobre quin objecte s'aplica (Objecte en Curs), via operador .? Raona la teua resposta**

**Recorda:** dins d'un mètode **dinàmic**, **this** és l'**Objecte en Curs**

Encara que en Java **no** és obligatori, pots explicitar-lo usant l'expressió **this.distOrigen()**



# Declaració i ús d'un mètode

## invocació des d'un mètode **de la classe** a un altre d'una **altra classe**

```
public class Punt {  
    private double x, y;  
    ...  
    public double distOrigen() { return Math.sqrt(x * x + y * y); }  
    private double aleatori() {  
        return Math.random() * (distOrigen() + 1);  
    }  
    ...  
}
```

**valor double**

**valor double**

BlueJ:exemplesT4

- Sobre els mètodes **distOrigen** i **aleatori** de la classe **Punt**, respon...


**Quins mètodes usa cadascun d'ells? Quins són estàtics i de quina classe? Sobre quin objecte s'apliquen (Objecte en Curs), via operador .? Raona les teues respostes**

- Accedeix** des de BlueJ a la documentació de **Math** i **comprova** si els paràmetres formals dels seus mètodes **sqrt** i **random** concorden (nº, **tipus** i ordre d'aparició) amb els arguments amb els que s'invoquen en **distOrigen** i **aleatori**
- Comprova** si les expressions que retornen **distOrigen** i **aleatori** són del mateix **tipus** (o compatible) que el del resultat que figura en la seua capçalera

# Declaració i ús d'un mètode

## invocació des d'un mètode d'un altra classe a un altre de la classe

```
public class Prova2Punt {  
    public static void main(String[] args) {  
        Punt p = new Punt(); // p és el punt (0.0, 0.0)  
        int x = 3;  
        p.moure(x, x + 1); 3.0, 4.0 // p és ara (3.0, 4.0)  
        double d = 3 * p.distOrigen(); 15.0  
    }  
}
```

 BlueJ:examplesT4

- **Observa** el cos del mètode `main` de la classe `Prova2Punt` i **respon**...
  1. Quins mètodes usa?
  2. Quins d'ells són dinàmics? De quina classe són? Sobre quin objecte s'apliquen (Objecte en Curs), via operador `.`? Raona les teues respostes
- **Avalua**, en el *Code Pad* del projecte si vols, les següents **expressions**:
  1. **Arguments** de la invocació a `moure` i punt resultant de `p.moure(x, x + 1)`
  2. Valor resultant de `p.distOrigen()` i, després, el valor de la variable `d`

# Declaració i ús d'un mètode segons el seu tipus i especificació: **documentació d'un mètode**

- **Què és?**

- **Especificació de totes les característiques d'un mètode**, tant les de les seues dades (paràmetres i precondicions) com les del resultat exacte que obté per a cada entrada especificada
- **Evita referències als detalls d'implementació**, per tal de no confondre què fa el mètode amb com ho fa

- **Per a què serveix?**

- **Per a reutilitzar el *software***, i.e. per a saber com usar-lo independentment de la seua implementació: quin és el seu perfil, quines condicions especials han de complir els seus paràmetres (precondicions) i quin és exactament el seu resultat per a cada entrada especificada
- **Per a produir un *software* de qualitat**, on la implementació sempre satisfaga l'especificació, a mode de contracte

- **Exemples:**

- Documentació de l'API de Java
- Documentació de qualsevol classe de qualsevol projecte BlueJ usat fins ara

# Declaració i ús d'un mètode segons el seu tipus i especificació: **documentació. Per a què?**

## Class Data

java.lang.Object  
Data

```
public class Data  
extends java.lang.Object
```

La classe Data permet representar dates en format dia, mes i any.

Version:

Curs 2018-19

Author:

IIP

### Constructor Summary

#### Constructors

##### Constructor and Description

Data ()

Crea una Data per defecte 01/01/2012.

Data(Data d)

Crea una Data a partir dels valors de dia, mes i any de una Data donada.

Data(int d, int m, int a)

Crea una Data amb els valors donats de dia, mes i any.

### Method Summary

#### All Methods Instance Methods Concrete Methods

##### Modifier and Type

##### Method and Description

Data

clone ()

Torna una nova Data còpia exacta de l'actual.

Data

diaSeguent ()

Torna la Data del dia següent a la Data actual, suposant que és correcta.

boolean

Objecta

boolean

Objecta

boolean

Objecta

boolean

Objecta

boolean

Objecta

boolean

Objecta

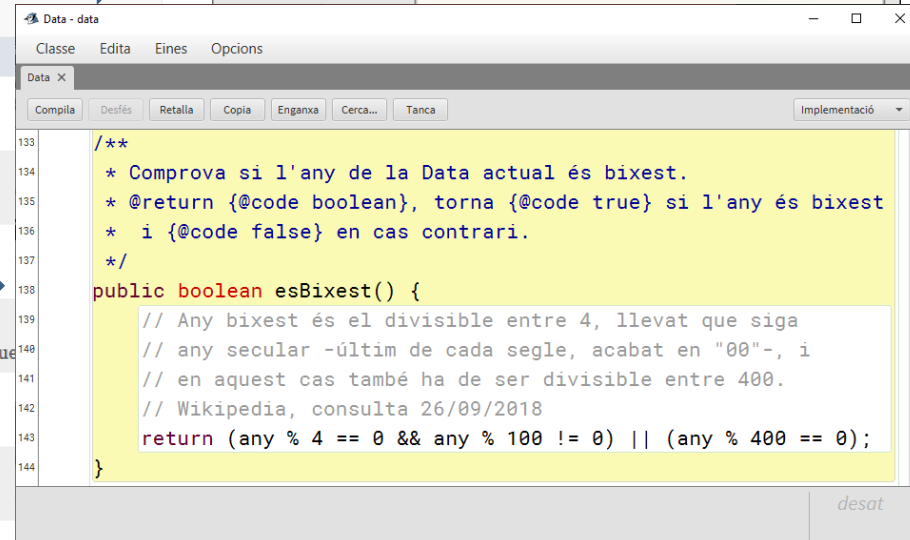
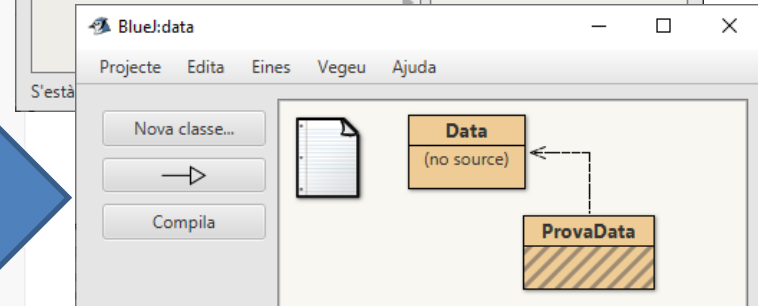
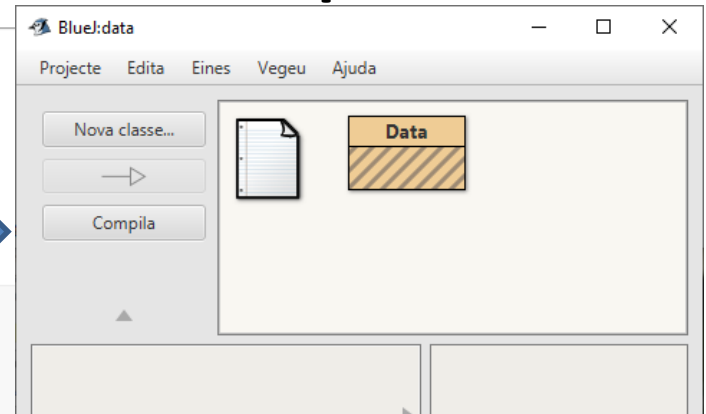
**Documentació interna**

**Facilita el manteniment...**

**Guia per a implementar...**

**Documentació externa**

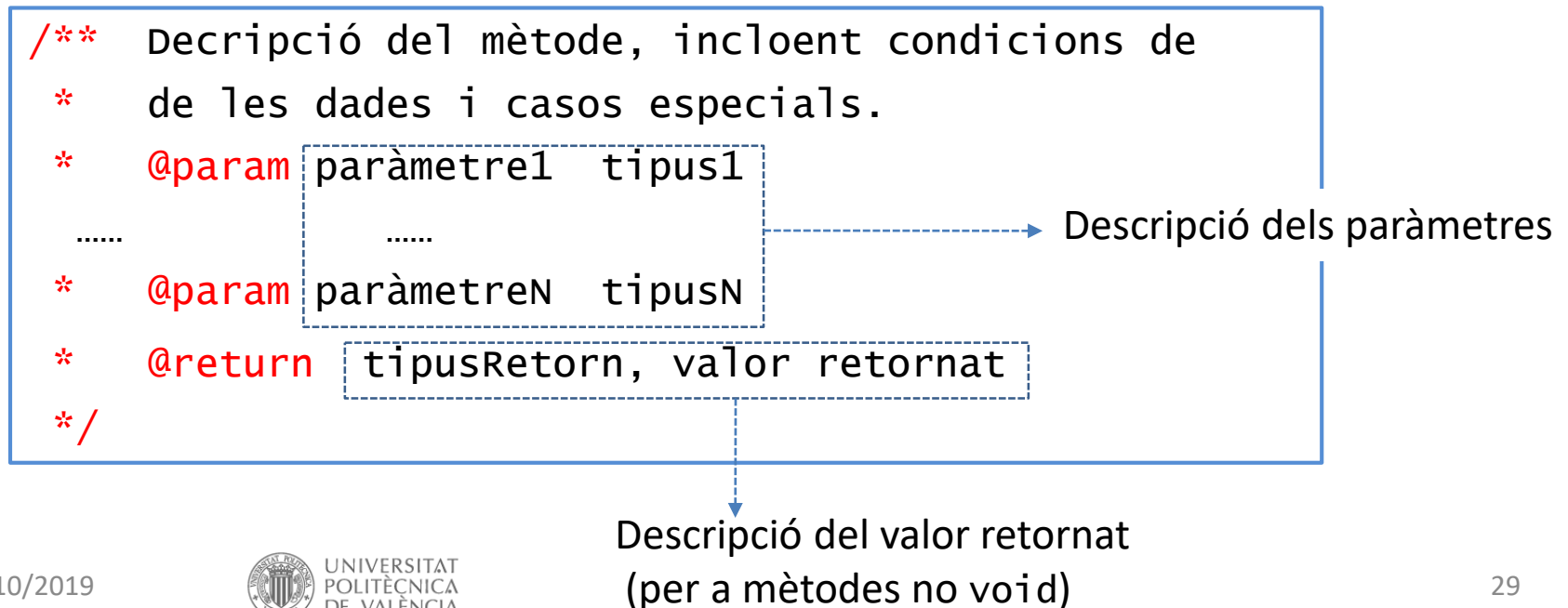
**Guia per a utilitzar...**




# Declaració i ús d'un mètode segons el seu tipus i especificació: **documentació. Com?**

Java té definit un estàndard de documentació en forma de comentaris a incloure en el codi font.

- Si es segueix l'estàndard, l'eina de Java **javadoc** genera automàticament el document *html* amb un estil definit.
- Recorda que en **BlueJ** javadoc s'executa en passar de la vista de codi (**Implementació**) a la vista de documentació (**Interfície**).
- Bàsicament:



# Declaració i ús d'un mètode segons el seu tipus i especificació: **documentació**

 BlueJ:exemplesT4

- **Accedeix** des de BlueJ a la documentació de la classe String i **busca** un mètode que te permetia resoldre el següent problema:

Convertir en un `String` un valor de tipus `double`, per exemple, el resultat de  $23.5 + 5.2$

- Quan el trobes, **escriu en el Code Pad** del projecte l'expressió corresponent, usant el mètode de `String` d'acord al seu tipus i la seua especificació.
- **Edita** la classe `Punt` del projecte i **respon...**
  1. Quins **símbols** s'utilitzen per a escriure la documentació externa d'un mètode?
  2. Quines **etiquetes** s'utilitzen per a descriure els paràmetres? i els resultats?
  3. Quins **símbols** s'utilitzen per a escriure la documentació interna d'un mètode?
- **Genera** la **documentació** de la classe `Punt`
- **Edita** la classe `Punt` del projecte i **afegeix, degudament documentat**, el mètode `divideix(double)` que modifica el punt actual dividint les seues coordenades per la quantitat que se passa com paràmetre.

# Exercici: la classe Cuadrado que usa Punto

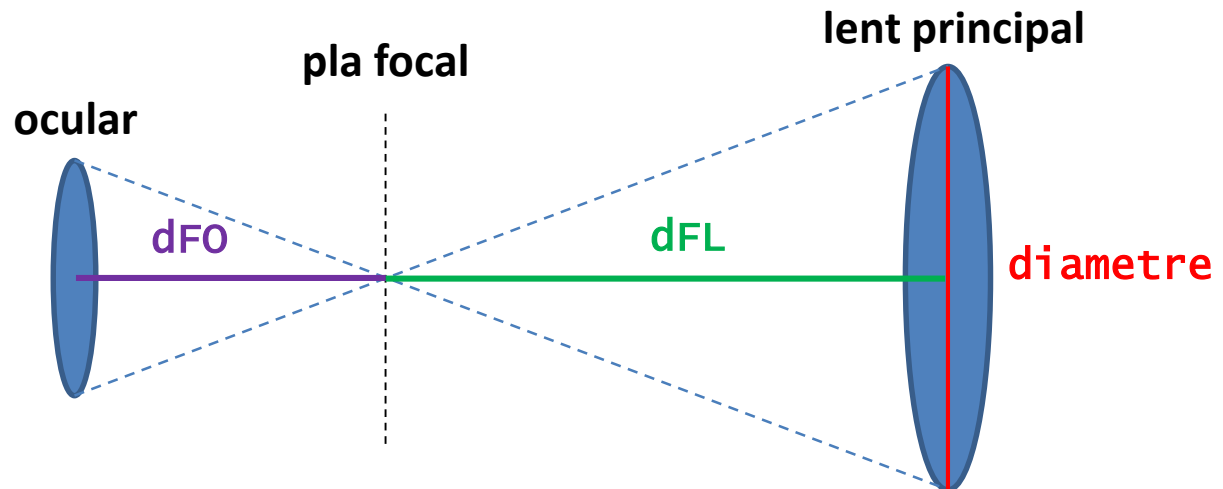
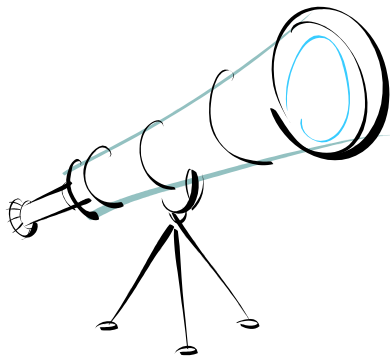


## CAP: La clase Cuadrado que usa Punto (clave CCDHK4ai)

- Fer aquest exercici t'ajudarà a entendre el que és ...
  - Usar una classe, en base a la seua especificació, per tal de dissenyar-ne un altra
  - Invocar l'execució d'un mètode des del cos d'un altre

# Exercici: la classe Telescopi

- Un telescopi es pot caracteritzar mitjançant (TÉ UN):
  - *diàmetre* de l'objectiu o lent principal (**diàmetre** en mm)
  - *distància focal* de la *lent* principal (**dFL** en mm)
  - *distància focal* de l'*ocular* (**dFO** en mm)
- A partir d'aquests valors es poden calcular, entre altres:
  - els *augment*s: relació entre **dFL** i **dFO**
  - la *relació focal*: relació entre **dFL** i **diàmetre**





## Exercici – P1: usar una classe seguint la seua especificació



BlueJ:exercicisT4

- **Accedeix** a la **especificació** de la classe **Telescopi** del projecte **BlueJ exercicisT4**, (editant el .class de **Telescopi** tens la vista de documentació o **Interfície**), i **úsala** per a dissenyar (en el mateix projecte) un programa **TestTelescopiP1** tal que...
  1. **crea** un objecte de la classe **Telescopi** amb una lent principal de **76.2** mm de diàmetre, una distància focal de **165.1** mm i una distància focal ocular de **20.32** mm
  2. **mostra per pantalla** els augments i la relació focal del telescopi **t** (**amb 2 xifres decimals**)
  3. **actualitza** a un nou valor llegit des de teclat el diàmetre de **t**
  4. **mostra per pantalla** la nova relació focal de **t** (**amb 2 xifres decimals**)
- **RECORDA** comprovar amb **Checkstyle** si estàs utilitzant un bon estil de programació


## Exercici – P2: dissenyar una classe seguint la seua especificació

BlueJ:exercicisT4

- **Esborra** del directori **exercicisT4** el .class de la classe **Telescopi** (fes-ho des de la carpeta **exercicisT4**, **NO** des de l'opció **Elimina** de BlueJ)
- **Dissenya** la teua classe **Telescopi**, en el projecte **BlueJ exercicisT4**, seguint la seua **especificació** disponible en **exercicisT4/doc** (fitxer **index.html**)
- **Comprova** després que el teu programa **TestTelescopiP1** segueix funcionant correctament

# Sobrecàrrega i sobreescriptura d'un mètode Java


- En una mateixa classe pot haver més d'un mètode amb el mateix nom, inclús amb el mateix tipus de retorn, sempre que la seua llista de paràmetres siga diferent: nombre, tipus u ordre dels paràmetres en la llista. En aquest cas es diu que estan **sobrecarregats**.

 BlueJ:exemplesT4

- Accedeix** des de BlueJ a la documentació de la classe [String](#) i **comprova** que, per exemple, els mètodes **indexOf** i **substring** estan sobrecarregats.
- Escriu en el Code Pad** del projecte les instruccions següents (sense els comentaris), per provar la funcionalitat d'aquests mètodes de la classe **String**.

```
String s = "Sobrecarrega";  
int p = s.indexOf('r');      // p val 3  
int q = s.indexOf('r', 4);   // q val 7  
  
String ss = s.substring(0, 5); // ss és "Sobre"  
ss = s.substring(5);         // ss és "carrega"
```

# Sobrecàrrega i sobreescriptura d'un mètode Java: ús de this en els mètodes constructors

 BlueJ:exemplesT4

- **Edita** la classe **Punt** i **respon** ...
  1. Per què estan **sobrecarregats** els **constructors**?
  2. Fixa't en les instruccions del cos de cada constructor, **són similars**?
- **Modifica** els constructors per tal que usen **this**(...) com segueix:

Mètodes constructors **sobrecarregats**:  
declarats amb el mateix nom però amb  
diferents llistes de paràmetres.

```
public class Punt {  
    private static int comptador = 0;  
    private double x;  
    private double y;  
  
    public Punt(double abs, double ord) {  
        this.x = abs; this.y = ord;  
        comptador++;  
    }  
  
    public Punt() { this(0, 0); }  
  
    public Punt(Punt p) { this(p.x, p.y); }  
    ...  
}
```

# Sobrecàrrega de variables:

## Principi de Màxima Proximitat

- Són **variables locals** als mètodes les que es declaren en el seu cos, els seus paràmetres i l'objecte en curs (**this**).
- Des de qualsevol mètode es pot accedir, a més, als atributs de classe i d'instància definits en la seua classe (**variables globals**), i en altres classes (si són públics per a la classe del mètode).
- Si una variable local i una global tenen el mateix nom sempre s'associa el nom a la local (**principi de màxima proximitat**). Exemple:

```
public class Punt {  
    private static int comptador = 0;  
    private double x, y;  
    ...  
    public void moure(double x, double y) {  
        x += x; y += y;  
    }
```

**ERROR LÒGIC !**

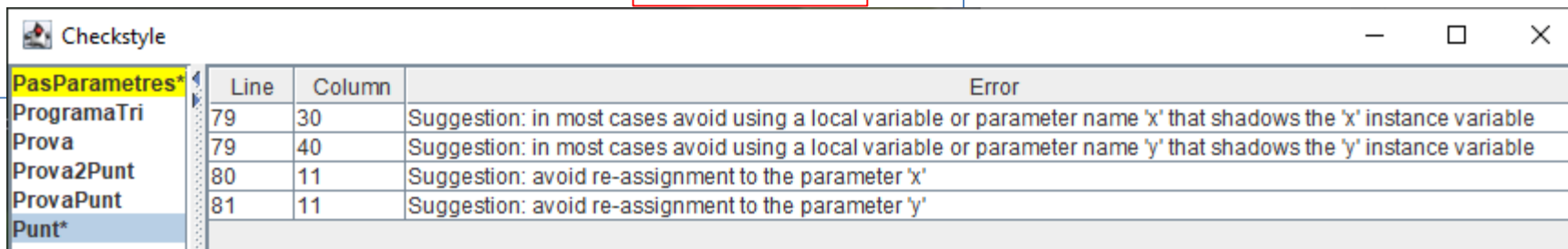
```
public void moure(double x, double y) {  
    this.x += x; this.y += y;  
}
```

**CORRECTE !**

**this** és una referència del tipus de la classe, per anomenar a l'objecte actual u objete en curs. És una variable **final**.

```
public void canvi(Punt p) {  
    this = p;  
}
```

**ERROR DE COMPILACIÓ !**



	Line	Column	Error
PasParametres*			
ProgramaTri	79	30	Suggestion: in most cases avoid using a local variable or parameter name 'x' that shadows the 'x' instance variable
Prova	79	40	Suggestion: in most cases avoid using a local variable or parameter name 'y' that shadows the 'y' instance variable
Prova2Punt	80	11	Suggestion: avoid re-assignment to the parameter 'x'
ProvaPunt	81	11	Suggestion: avoid re-assignment to the parameter 'y'
Punt*			

# Sobrecàrrega de variables:

## Principi de Màxima Proximitat

```
public class Punt {  
    private static int comptador = 0;  
    private double x;  
    private double y;
```

Per agilitat en l'escriptura, Java permet ometre la paraula `this` mentre no hi haja confusió possible

```
...  
    public void moure(double px, double py) { x += px; y += py; }
```

```
    public double distancia(Punt p) {  
        double x = p.x - x;  
        double y = p.y - y;  
        return Math.sqrt(x * x + y * y);  
    }
```

**ERROR  
DE  
COMPILACIÓ !**

```
    public double distancia(Punt p) {  
        double x = p.x - this.x;  
        double y = p.y - this.y;  
        return Math.sqrt(x * x + y * y);  
    }  
}
```

**CORRECTE !**

Checkstyle

PasParametres\*

ProgramaTri

Prova

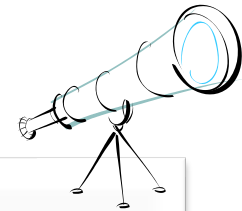
Prova2Punt

ProvaPunt

Punt\*

Line	Column	Error
107	16	Suggestion: in most cases avoid using a local variable or parameter name 'x' that shadows the 'x' instance variable
108	16	Suggestion: in most cases avoid using a local variable or parameter name 'y' that shadows the 'y' instance variable

## Exercici – P3: ampliació de la classe Telescopi



BlueJ:exercicisT4

- **Afegeix** a la classe **Telescopi** els següents constructors:
  1. Un **constructor** d'un telescopi amb una lent principal de diàmetre **d**, una distància focal de la lent **dF** i una distància focal ocular **dFOcular**
  2. Un **constructor** d'un telescopi amb una lent principal de diàmetre **d** i distància focal i ocular estàndards. Usa les **constants** de la classe i **this(...)**
- Fet açò, **modifica** el constructor sense paràmetres ja existent per tal que use **this(...)**

## Exercici – P4: creació de la classe TestTelescopi

BlueJ:exercicisT4

- **Implementa** una classe Programa **TestTelescopi** de forma que (en el seu **main**):
  1. **Cree** tres telescopis:
    - **t1**, amb una lent principal de **76.2** mm de diàmetre, una distància focal de **165.1** mm i una distància focal ocular de **20.32** mm
    - **t2**, amb una lent principal de **76.2** mm de diàmetre, però amb distàncies focal i ocular estàndards
    - **t3**, un telescopi per defecte, de mesures estàndards
  2. **Mostre per pantalla** els augments i la relació focal de cada telescopi
- **RECORDA** comprovar amb **Checkstyle** si estàs utilitzant un bon estil de programació

# Sobrecàrrega i **sobreescritura** d'un mètode Java: **mètodes toString i equals d'Object**

- [Object](#) és una classe predefinida de Java que defineix el comportament comú de tots els objectes del llenguatge: en ser creats resideixen en el *heap* i són accessibles mitjançant referències.
- Un Object no té estructura interna, és un objecte buit.
- Qualsevol objecte de Java és un cas particular d'Object, de qui **hereten** els seus mètodes, entre els que destaquen:

```
public boolean equals(Object o)
```

Comprova si l'objecte actual i o són el mateix objecte del heap.

```
public String toString()
```

Retorna un String que indica la classe de l'objecte actual i un codi numèric (la referència).

# Sobreescritura dels mètodes toString i equals d'Object

p1 : Punt

private double x: 0.0

private double y: 0.0

Inspecciona

Obté

Mostra camps estàtics

Tanca

p2 : Punt

private double x: 0.0

private double y: 0.0

Inspecciona

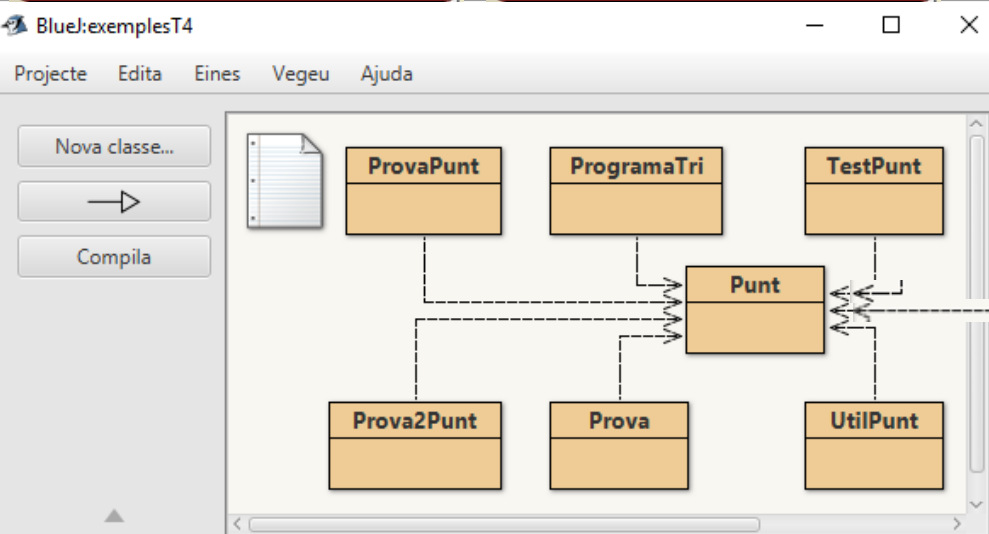
Obté

Mostra camps estàtics

Tanca

BlueJ:exemplesT4

- Crea dos punts **p1** i **p2** amb el constructor per defecte de la classe **Punt**
- Consulta els mètodes heretats d'**Object** que es poden aplicar a **p1**
- Executa el mètode **toString**. Quin és el resultat?
- Executa el mètode **equals**, passant-li com argument **p2**. Quin és el resultat?



p1: Punt

p2: Punt

p1 : Punt

heretat de Object

double distOrigen()

double distancia(Punt p)

double getX()

double getY()

void intercanviX(Punt b)

void moure(double px, double py)

void moureAleat()

void setX(double px)

void setY(double py)

Inspecciona

Treu

boolean equals(Object)

Class<?> getClass()

int hashCode()

void notify()

void notifyAll()

String toString()

void wait(long)

void wait(long, int)

void wait()

BlueJ: Resultat del mètode

boolean equals(Object)

p1.equals(p2) retornat:

boolean false

Inspecciona

Obté

Tanca

BlueJ: Resultat del mètode

String toString()

p1.toString() retornat:

String "Punt@48b2144b"

Inspecciona


Obté

Tanca



# Sobreescritura dels mètodes toString i equals d'Object

- Una característica primordial dels mètodes heretats és que en cada classe es poden **sobreescriure** a conveniència.

 Blue:exemplesT4

- Desfés el comentari dels mètodes equals i toString en la classe **Punt**. Aquests mètodes **sobreescriuen** als de la classe **Object**.

```
/** Comprova si o és un Punt i les seues coordenades
 * coincideixen amb les del Punt actual.
 * @param o Object a comparar.
 * @return boolean, true si són iguals o false en cas contrari.
 */
```

```
public boolean equals(Object o) {
    return o instanceof Punt
        && this.x == ((Punt) o).x
        && this.y == ((Punt) o).y;
}
```

L'estàndard recomana que **equals** comprove si o és de la classe (operador **instanceof**), i també la coincidència atribut a atribut.

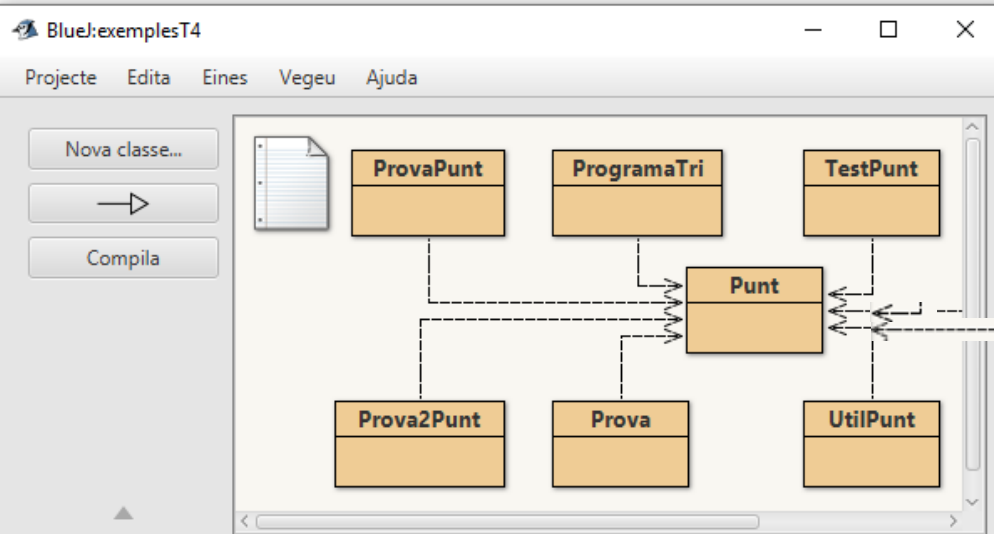
```
/** Torna un String que representa el Punt actual en el
 * format típic matemàtic, i.e., (abscissa,ordenada)
 * @return String, el resultat.
 */
public String toString() {
    return "(" + x + "," + y + ")";
}
```

# Sobreescritura dels mètodes toString i equals d'Object

- Si Java troba el mètode **sobreescrit** en la classe, usa el codi intern de la classe, sino, usa el codi del mètode en Object.

BlueJ:examplesT4

- Crea** dos punts **p1** i **p2** amb el constructor per defecte de **Punt**
- Consulta** els mètodes que es poden aplicar a **p1**
- Executa** el mètode **toString**. Quin és ara el resultat?
- Executa** el mètode **equals**, passant-li com argument **p2**. Quin és ara el resultat?
- La classe **TetPunt** prova aquestos mètodes. Revisa-la i **executa-la**



BlueJ:examplesT4

Projecte Edita Eines Vegeu Ajuda

Nova classe...

→

Compila

p1: Punt

p2: Punt

p1 : Punt

heretat de Object

- double distOrigen()
- double distancia(Punt p)
- boolean equals(Object o)
- double getX()
- double getY()
- void intercanviX(Punt b)
- void moure(double px, double py)
- void moureAleat()
- void setX(double px)
- void setY(double py)
- String toString()

Inspecciona

Treu

boolean equals(Object o) [ redefinit en Punt ]

Class<?> getClass()

int hashCode()

void notify()

void notifyAll()

String toString() [ redefinit en Punt ]

void wait(long)

void wait(long, int)

void wait()

BlueJ: Resultat del mètode

```
// Comprova si o és un Punt i les seues coordenades
// coincideixen amb les del Punt actual.
// @param o Object a comparar.
// @return boolean, true si són iguals o false en cas contrari.
boolean equals(Object o)
```

p1.equals(p2) retornat:

boolean true

Inspecciona

Obté

BlueJ: Resultat del mètode

```
// Torna un String que representa el Punt actual en el
// format típic matemàtic, i.e., (abscissa,ordenada)
// @return String, el resultat.
String toString()
```

p1.toString() retornat:

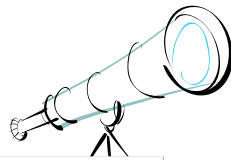
String "(0,0,0,0)"

Inspecciona

Obté

Tanca

## Exercici – P5: ampliació de la classe Telescopi



BlueJ:exercisT4

- En el *Code Pad* de BlueJ, **crea** dos telescopis **t1** i **t2** amb el constructor per defecte de la classe **Telescopi**
- **Consulta** els mètodes heretats d'**Object** que es poden aplicar a **t1**
- **Executa** el mètode **toString**. Quin és el resultat?
- **Executa** el mètode **equals**, passant-li com argument **t2**. Quin és el resultat?
- **Sobreescriu** a la classe **Telescopi** els mètodes heretats d'**Object**:
  - **equals**, de manera que permeta comprovar si dos telescopis són iguals, i.e., si coincideixen en diàmetre, distància focal de la lent i distància focal ocular
  - **toString**, de manera que torne un **String** amb la informació del telescopi en el següent format:  
Telescopi de diàmetre = **d**, distància focal = **dFL** i distància focal ocular = **dFO**  
(usant **String.format** per mostrar els valors reals amb 2 decimals)
- **Comprova** en el *Code Pad* de BlueJ que ara el resultat és l'esperat

## Exercici – P6: modificació de la classe TestTelescopi

BlueJ:exercisT4

- **Modifica** el disseny de la classe **TestTelescopi** de forma que:
  1. **mostre per pantalla** les característiques dels tres telescopis creats en el **main**
  2. **mostre per pantalla** el resultat de comprovar si **t1** és igual a algun altre dels telescopis creats
- **RECORDA** comprovar amb **Checkstyle** si estàs utilitzant un bon estil de programació

# Execució d' (una crida a) un mètode: registre d'activació i pila de crides



[Video1: TestRAMActivo](#)

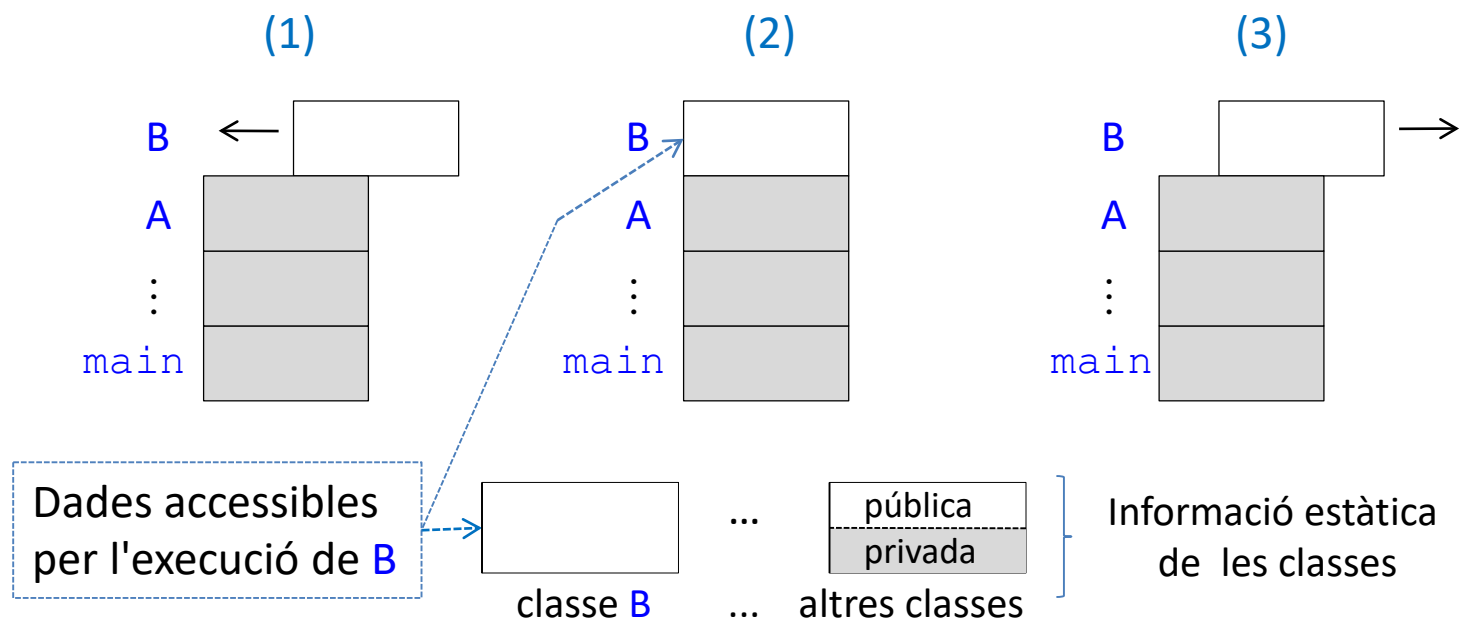
- La JVM sols executa un mètode a la vegada: **mètode actiu**.
- La JVM associa una zona de memòria exclusiva per a les dades i càlculs del mètode: **registre d'activació**, que conté:
  - Una variable per cada **variable local** i per cada **paràmetre formal** del mètode, incloent una variable **final this** si el mètode és dinàmic. Aquestes variables seran del tipus que corresponga segons la definició del mètode.
  - Si el mètode torna un resultat, una variable del mateix tipus que el tipus del resultat del mètode, on s'emmagatzema el **valor a tornar** com resultat de la crida: **VR (valor de retorn)**.
  - Una variable que emmagatzema el **punt al que s'ha de tornar el control** de l'execució quan acabe d'executar-se la crida, o el que es el mateix, el punt on es va suspendre la crida anterior: **AR (adreça de retorn)**.
  - Las dues últimes variables són manipulades automàticament per la JVM.
- Quan un mètode **A** invoca a un mètode **B**:
  - l'execució d'**A** queda en suspens,
  - l'estat d'**A** es preserva en el seu registre d'activació, que no es destrueix fins que es reanude i acabe la seua execució.
- En memòria poden coexistir diversos registres que se gestionen com una **Pila**:
  - el registre del mètode actiu o registre actiu,
  - un registre per cada mètode que permaneixca en suspens.

# Execució d' (una crida a) un mètode: registre d'activació i pila de crides



[Video2: TestPilaRA](#)

- **Pila de crides** (*stack*): Java gestiona els registres ordenant-los per antiguitat. Quan **A** invoca a **B**, el seu registre es disposa en el cim de la pila (1); en acabar, el seu registre es desempila (3).
- El mètode actiu sols pot accedir a les variables del cim (2).
- Les variables de classe estan a banda i són accessibles per qualsevol mètode actiu (sempre que siguin públiques per al mètode).



# Execució d' (una crida a) un mètode: registre d'activació i pila de crides

- Si s'està executant el mètode **A** i s'arriba a una instrucció en que es crida a un mètode **B**:
  - S'avaluen en **A** les expressions que apareixen com arguments en la crida.
  - Es reserva espai en memòria per al registre d'activació de **B**.
  - Els paràmetres de **B** s'inicien (en el registre) als valors dels arguments (**pas de paràmetres per valor**). La variable `this` del nou registre s'inicia a l'objecte en curs, si **B** és dinàmic.
  - S'executen les instruccions de **B**; el mètode acaba en executar-se un `return`.
  - Acabada la crida a **B**, el seu registre d'activació s'allibera.
  - L'execució d'**A** continua des del punt en que es va cridar a **B**. La crida feta en **A** pren el valor de retorn de **B**.
- En els mètodes constructors:
  - una vegada avaluats els paràmetres reals o arguments es crea un registre d'activació i es passa el seu valor als paràmetres formals.
  - S'obté un bloc de memòria lliure en el monticle, lo suficientment gran per a albergar un objecte de la classe, que passarà a ser l'objecte en curs amb el que s'inicia `this`.
  - S'inicialitzen les variables d'instància de l'objecte creat amb el valor per defecte del tipus.
  - Si hi ha instruccions en el cos del constructor, se executen (com per exemple, per a canviar els valors per defecte dels atributs de `this`).

# Execució d' (una crida a) un mètode: traça

- Exemple (1). El mètode `main` d'una classe `Prova` comença la seua execució:

NO hi ha objecte en curs

NO hi ha valor de retorn

`main` és el mètode actiu

Classe `Punt`

```
public double distancia(Punt p) {  
    double abs = p.x - this.x;  
    double ord = p.y - this.y;  
    return Math.sqrt(abs * abs + ord * ord);  
}
```

Classe `Prova`

```
public static void main(String[] args) {  
    ➔ double x = 0.0;  
    Punt q1 = new Punt(3.0, 4.0);  
    Punt q2 = new Punt();  
    x = q2.distancia(q1);  
    System.out.println(x);  
}
```

*Prova.main*

args {}

AR

pila

monticle

```
java Prova
```

Execució del `main` sense arguments!

# Execució d' (una crida a) un mètode: traça

- Exemple (2). El mètode `main` va a invocar al mètode `distancia` de `Punt`:

q2 és l'objecte en curs

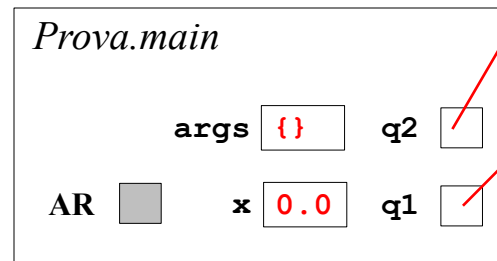
Classe `Punt`

```
public double distancia(Punt p) {  
    double abs = p.x - this.x;  
    double ord = p.y - this.y;  
    return Math.sqrt(abs * abs + ord * ord);  
}
```

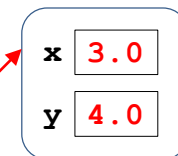
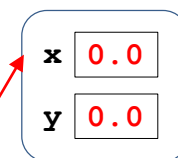
Classe `Prova`

```
public static void main(String[] args) {  
    double x = 0.0;  
    Punt q1 = new Punt(3.0, 4.0);  
    Punt q2 = new Punt();  
    → x = q2.distancia(q1);  
    System.out.println(x);  
}
```

```
java Prova
```



pila



monticle



# Execució d' (una crida a) un mètode: traça

- Exemple (2): El mètode **distancia** comença la seua execució i l'execució del **main** queda en suspens:

**distancia és el mètode actiu**

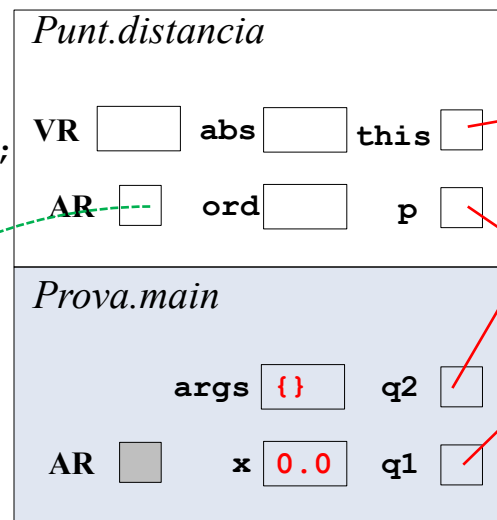
**this és l'objecte en curs**

Classe Punt

```
public double distancia(Punt p) {  
    ➔ double abs = p.x - this.x;  
    double ord = p.y - this.y;  
    return Math.sqrt(abs * abs + ord * ord);  
}
```

Classe Prova

```
public static void main(String[] args) {  
    double x = 0.0;  
    Punt q1 = new Punt(3.0, 4.0);  
    Punt q2 = new Punt();  
    x = q2.distancia(q1);  
    System.out.println(x);  
}
```



pila

monticle

java Prova

# Execució d' (una crida a) un mètode: traça

- Exemple (3): El mètode **distancia** ha calculat el resultat en la variable **VR** del seu registre, i la JVM està a punt de reanudar **main** per on indica **AR**:

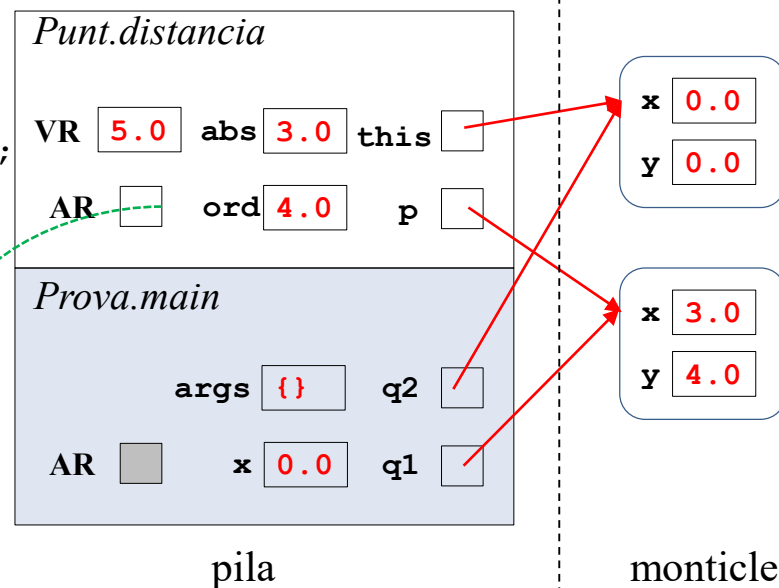
**distancia és el mètode actiu**

Classe **Punt**

```
public double distancia(Punt p) {  
    double abs = p.x - this.x;  
    double ord = p.y - this.y;  
    → return Math.sqrt(abs * abs + ord * ord);  
}
```

Classe **Prova**

```
public static void main(String[] args) {  
    double x = 0.0;  
    Punt q1 = new Punt(3.0, 4.0);  
    Punt q2 = new Punt();  
    x = q2.distancia(q1);  
    System.out.println(x);  
}
```



java Prova

# Execució d' (una crida a) un mètode: traça

- Exemple (4). El mètode **main** ha rebut en x el resultat de la crida a **distancia** i està a punt d'acabar:

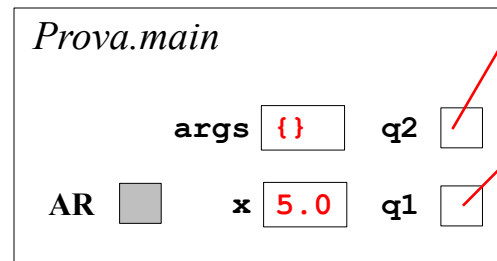
**main és el mètode actiu**

Classe **Punt**

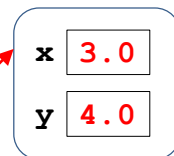
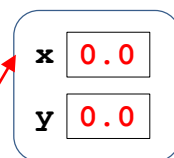
```
public double distancia(Punt p) {  
    double abs = p.x - this.x;  
    double ord = p.y - this.y;  
    return Math.sqrt(abs * abs + ord * ord);  
}
```

Classe **Prova**

```
public static void main(String[] args) {  
    double x = 0.0;  
    Punt q1 = new Punt(3.0, 4.0);  
    Punt q2 = new Punt();  
    x = q2.distancia(q1);  
    → System.out.println(x);  
}
```



pila



monticle

java Prova

# Execució d' (una crida a) un mètode: traça

- Exemple (5). Ha acabat l'execució del mètode **main**:

Classe **Punt**

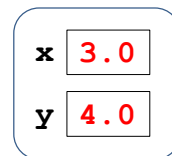
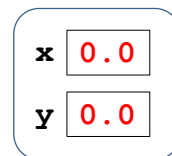
```
public double distancia(Punt p) {  
    double abs = p.x - this.x;  
    double ord = p.y - this.y;  
    return Math.sqrt(abs * abs + ord * ord);  
}
```

Classe **Prova**

```
public static void main(String[] args) {  
    double x = 0.0;  
    Punt q1 = new Punt(3.0, 4.0);  
    Punt q2 = new Punt();  
    x = q2.distancia(q1);  
    System.out.println(x);  
}
```

```
java Prova  
5.0
```

pila



monticle

# Execució d' (una crida a) un mètode: traça – depurador de BlueJ

BlueJ:exemplesT4

- En el mètode **main** de la classe **Prova** del projecte **BlueJ exemplesT4**, estableix un **punt de ruptura** en la línia on s'invoca al mètode **distancia**.
- Executa** el mètode **main** i en detindre's l'execució, una vegada alcançat el punt de ruptura, **inspecciona** les seues variables fent doble clic sobre elles en la finestra del depurador.

Prova - exemplesT4

Classe Edita Eines Opcions

Prova X

Compila Desfés Retalla Copia Enganxa Cerca... Tanca Implementació

```
12 public static void main(String[] args) {
13     double x = 0.0;
14     Punt q1 = new Punt(3.0, 4.0);
15     Punt q2 = new Punt();
16     x = q2.distancia(q1);
17     System.out.println(x);
18 }
19 }
```

BlueJ: BlueJ: Depurador

Opcions

Fils  
main (en el punt d'atu...)

Seqüència de crida  
Prova.main

Variables estàtiques

Variables d'instància

Variables locals

- String[] args = <object reference>
- double x = 0.0
- Punt q1 = <object reference>
- Punt q2 = <object reference>

Interrupció Pas Pas endins Continua Finalitza

: Punt

private double x 3.0

private double y 4.0

Inspecciona Obté

Mostra camps estàtics Tanca

: Punt

private double x 0.0

private double y 0.0

Inspecciona Obté

Mostra camps estàtics Tanca

# Execució d' (una crida a) un mètode: traça – depurador de BlueJ

BlueJ:exemplesT4

- **Fes clic** sobre el botó **Pas endins** de la finestra del depurador i **observa** el contingut de les zones Seqüència de crida, Variables estàtiques, Variables d'instància i Variables locals. Pots veure que:
  - En el cim de Seqüència de crida, el **mètode actiu** és **distancia**.
  - En Variables estàtiques, com s'han creat 2 objectes **Punt**, **comptador** val **2**.
  - En Variables d'instància, els atributs d'instància de l'objecte en curs (**q2**) a qui se li aplica el mètode **distancia**, i.e. **this**.
  - En Variables locals, el paràmetre formal **p** s'ha inicialitzat al valor de l'argument (**q1**) de la crida al mètode **distancia** des de **main**.

```
186 public double distancia(Punt p) {  
187     double abs = p.x - this.x;  
188     double ord = p.y - this.y;  
189     return Math.sqrt(abs * abs + ord * ord);  
190 }
```

Classe compilada. No hi ha errors de sintaxi

Opcions

Fils  
main (en el punt d'atu...)

Seqüència de crida  
Punt.distancia  
Prova.main

Variables estàtiques  
private int **comptador** = 2

Variables d'instància  
private double **x** = 0.0  
private double **y** = 0.0

Variables locals  
Punt **p** = <object reference>

Interrupció Pas Pas endins Continua Finalitza

# Execució d' (una crida a) un mètode: traça – depurador de BlueJ

BlueJ:exemplesT4

- Continua** l'execució pas a pas **fent clic** sobre el botó **Pas** fins que acabe l'execució del codi del mètode **distancia**. **Observa** el canvi d'estat de les variables involucrades.

```
106 public double distancia(Punt p) {  
107     double abs = p.x - this.x;  
108     double ord = p.y - this.y;  
109     return Math.sqrt(abs * abs + ord * ord);  
110 }
```

Classe compilada. No hi ha errors de sintaxi

Opcions

Fils  
main (en el punt d'atu...)

Seqüència de crida  
Punt.distancia  
Prova.main

Variables estàtiques  
private int **comptador** = 2

Variables d'instància  
private double **x** = 0.0  
private double **y** = 0.0

Variables locals  
Punt **p** = <object reference>  
double **abs** = 3.0  
double **ord** = 4.0

Interrupció Pas Pas endins Continua Finalitza

# Execució d' (una crida a) un mètode: traça – depurador de BlueJ

BlueJ:exemplesT4

- Continua** l'execució pas a pas **fent clic** sobre el botó **Pas** fins que acabe l'execució del codi del mètode **main**. **Observa** el canvi d'estat de les variables involucrades i que el **mètode actiu** en Seqüència de crida torna a ser el **main**.

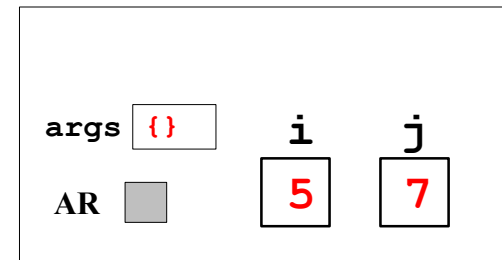


# Execució d' (una crida a) un mètode: pas de paràmetres per valor – tipus primitius

- a) S'avaluen en el **main**, el mètode actiu, les expressions que apareixen com arguments en la crida a **intercanvi** en curs.

```
...
public static void intercanvi(int a, int b) {
    int aux = a;
    a = b;
    b = aux;
}
...

public static void main(String[] args) {
    ...
    int i = 5, j = 7;
    ➔ intercanvi(i , j);
    ...
}
...
```



pila

monticle

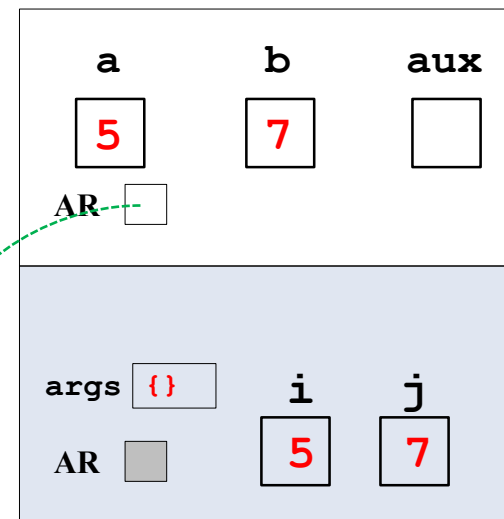
# Execució d' (una crida a) un mètode: pas de paràmetres per valor – tipus primitius

- b) S'activa en memòria el registre d'activació associat al mètode invocat **intercanvi**, el mètode actiu.
- c) Els paràmetres del registre s'inicien als valors dels arguments que s'han calculat en el **main**.

En RA se copien els valors de les variables,  
NO les variables

→

```
...  
public static void intercanvi(int a, int b) {  
    int aux = a;  
    a = b;  
    b = aux;  
}  
...  
  
public static void main(String[] args) {  
    ...  
    int i = 5, j = 7;  
    intercanvi(i, j);  
    ...  
}  
...
```

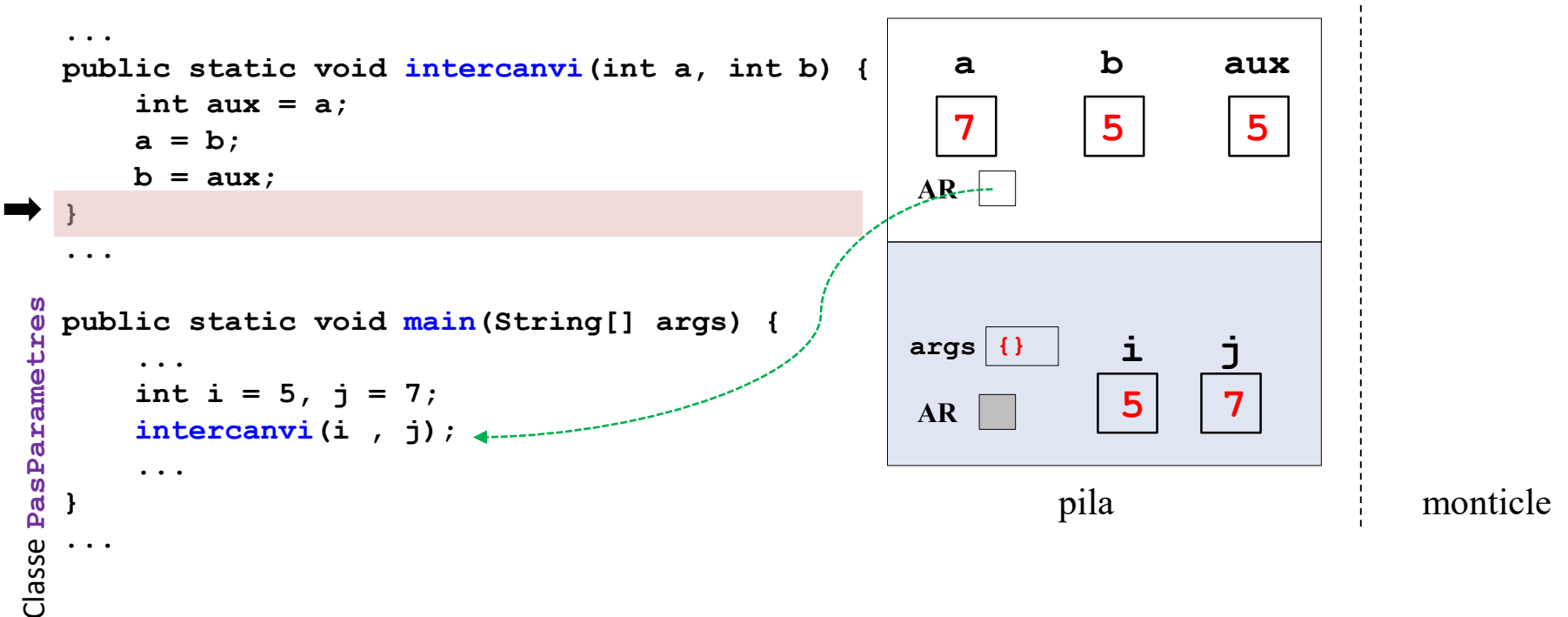


pila

monticle

# Execució d' (una crida a) un mètode: pas de paràmetres per valor – tipus primitius

d) Se realitza l'execució del codi del cos d'**intercanvi**, el mètode actiu.



# Execució d' (una crida a) un mètode: pas de paràmetres per valor – tipus primitius

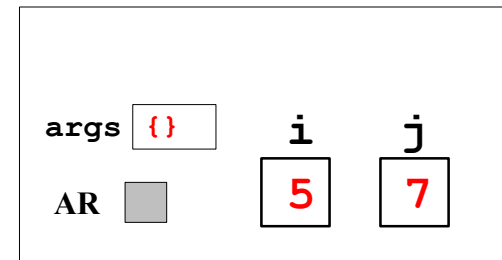
- e) Acabada la crida a **intercanvi**, el seu registre d'activació s'allibera, tornant a ser el **main** el mètode actiu.

Els canvis són locals, NO permanents.  
Cap modificació d'un tipus **primitiu** dins  
d'un mètode és visible des de fora

```
...  
public static void intercanvi(int a, int b) {  
    int aux = a;  
    a = b;  
    b = aux;  
}
```

```
...  
public static void main(String[] args) {  
    ...  
    int i = 5, j = 7;  
    intercanvi(i , j);  
    ...  
}
```

Classe PasParameters



pila

monticle

# Exercici 1: sobre pas de paràmetres per valor



- **CAP: Paso de parámetros: traza del programa Ejemplo1 (clave CCDHG4ai)**
  - Fer aquest exercici t'ajudarà a entendre el que és ...
    - Invocar l'execució d'un mètode des del cos d'un altre
    - Un argument i un paràmetre formal d'un mètode
    - El pas de paràmetres per valor (en Java)

# Execució d' (una crida a) un mètode:

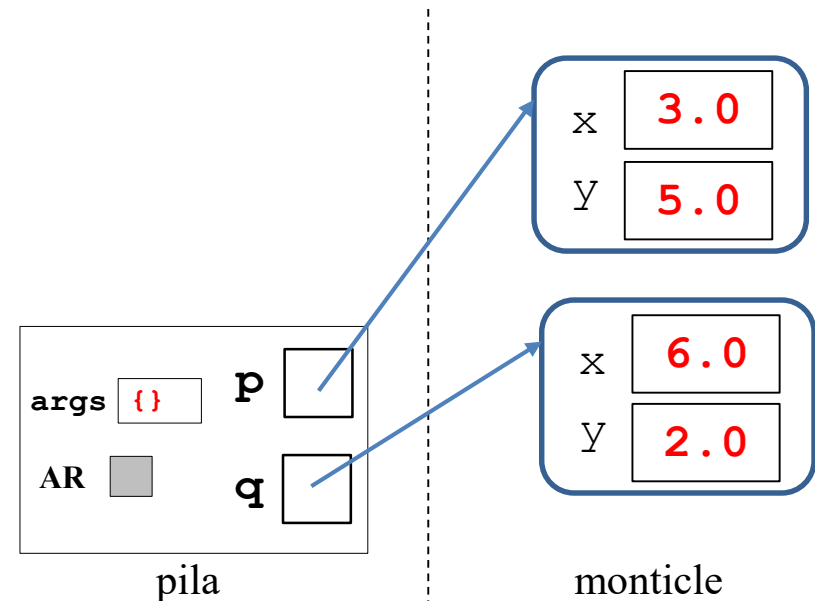
## pas de paràmetres per valor – tipus referència

### Exemple 1

- a) S'avaluen en el **main**, el mètode actiu, les expressions que apareixen com arguments en la crida a **intercanvi** en curs.

```
...
public static void intercanvi(Punt a, Punt b) {
    Punt aux = a;
    a = b;
    b = aux;
}
...

public static void main(String[] args) {
    ...
    Punt p = new Punt(3.0, 5.0);
    Punt q = new Punt(6.0, 2.0);
    → intercanvi(p, q);
    ...
}
```



# Execució d' (una crida a) un mètode: pas de paràmetres per valor – tipus referència

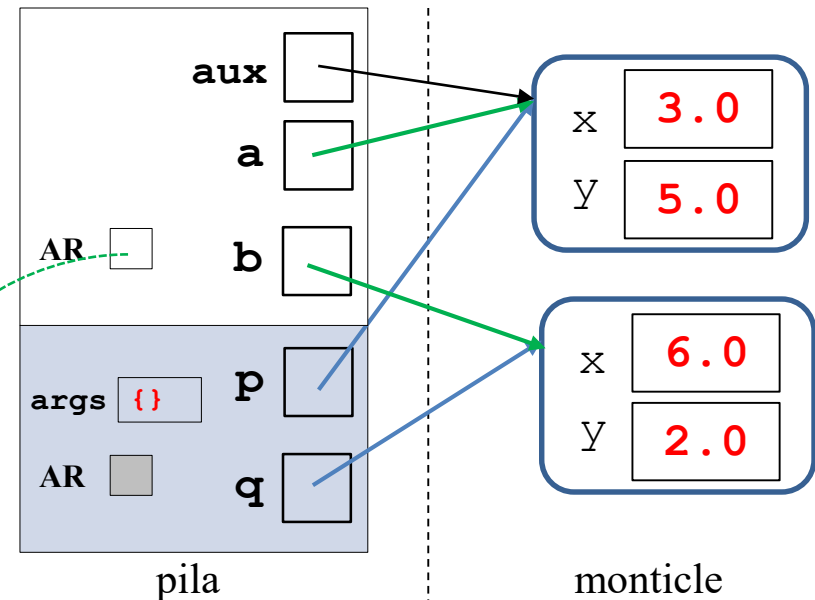
Exemple 1

- b) S'activa en memòria el registre d'activació associat al mètode invocat **intercanvi**, el mètode actiu.
- c) Els paràmetres del registre s'inicien als valors dels arguments que s'han calculat en el **main**.

L'objecte no es còpia en el RA, només la seua referència
- d) Se realitza l'execució del codi del cos d'**intercanvi**, el mètode actiu.

```
...  
public static void intercanvi(Punt a, Punt b) {  
    Punt aux = a;  
    a = b;  
    b = aux;  
}  
...
```

```
public static void main(String[] args) {  
    ...  
    Punt p = new Punt(3.0, 5.0);  
    Punt q = new Punt(6.0, 2.0);  
    intercanvi(p, q);  
    ...  
}
```



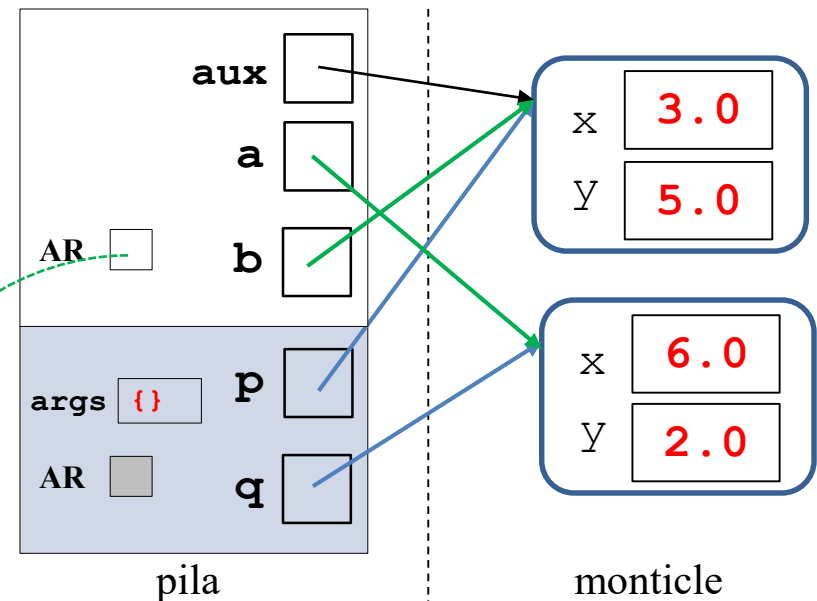
# Execució d' (una crida a) un mètode: pas de paràmetres per valor – tipus referència

## Exemple 1

d) Se realitza l'execució del codi del cos d'**intercanvi**, el mètode actiu.

```
...  
public static void intercanvi(Punt a, Punt b) {  
    Punt aux = a;  
    a = b;  
    b = aux;  
}
```

```
...  
public static void main(String[] args) {  
    ...  
    Punt p = new Punt(3.0, 5.0);  
    Punt q = new Punt(6.0, 2.0);  
    intercanvi(p, q);  
    ...  
}
```





# Execució d' (una crida a) un mètode:

## pas de paràmetres per valor – tipus referència

Exemple 1

- e) Acabada la crida a **intercanvi**, el seu registre d'activació s'allibera, tornant a ser el **main** el mètode actiu.

Els canvis són locals, NO permanents.  
Cap modificació d'una **referència** dins  
d'un mètode és visible des de fora

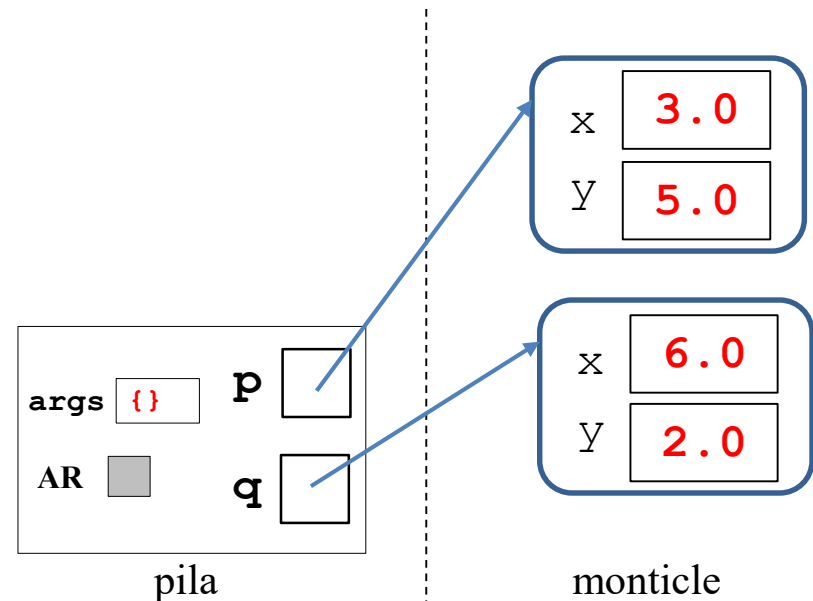
```
...  
public static void intercanvi(Punt a, Punt b) {  
    Punt aux = a;  
    a = b;  
    b = aux;  
}
```

...

```
public static void main(String[] args) {  
    ...  
    Punt p = new Punt(3.0, 5.0);  
    Punt q = new Punt(6.0, 2.0);  
    intercanvi(p, q);  
    ...  
}
```

→ ...

...



# Execució d' (una crida a) un mètode: pas de paràmetres per valor – tipus referència

Exemple 2

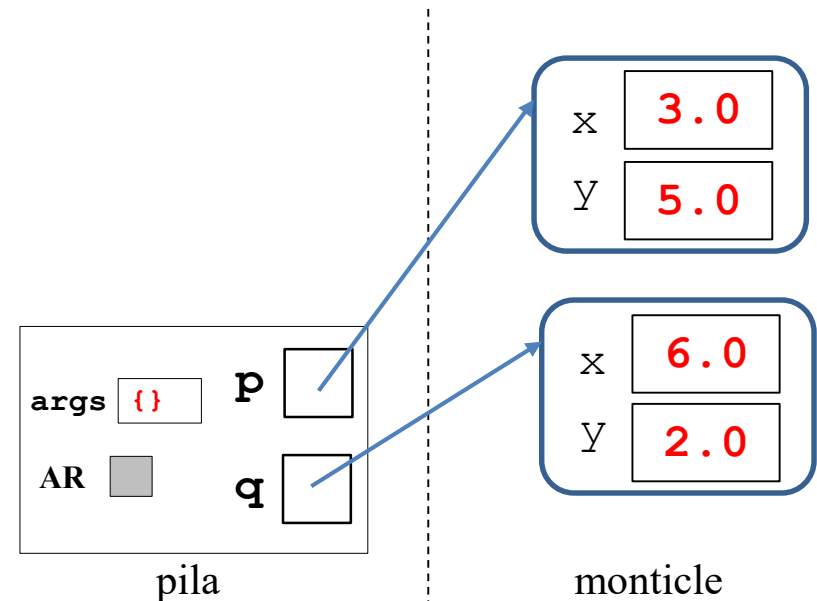
- a) S'avaluen en el **main**, el mètode actiu, les expressions que apareixen com arguments en la crida a **intercanviX** en curs.

Classe Punt

```
public void intercanviX(Punt b) {  
    double aux = this.x;  
    this.x = b.x;  
    b.x = aux;  
}
```

Classe PasParametres

```
public static void main(String[] args) {  
    ...  
    Punt p = new Punt(3.0, 5.0);  
    Punt q = new Punt(6.0, 2.0);  
    → p.intercanviX(q);  
    ...  
}
```



# Execució d' (una crida a) un mètode: pas de paràmetres per valor – tipus referència

Exemple 2

- b) S'activa en memòria el registre d'activació associat al mètode invocat **intercanviX**, el mètode actiu.
- c) Els paràmetres del registre s'inicien als valors dels arguments que s'han calculat en el **main**.

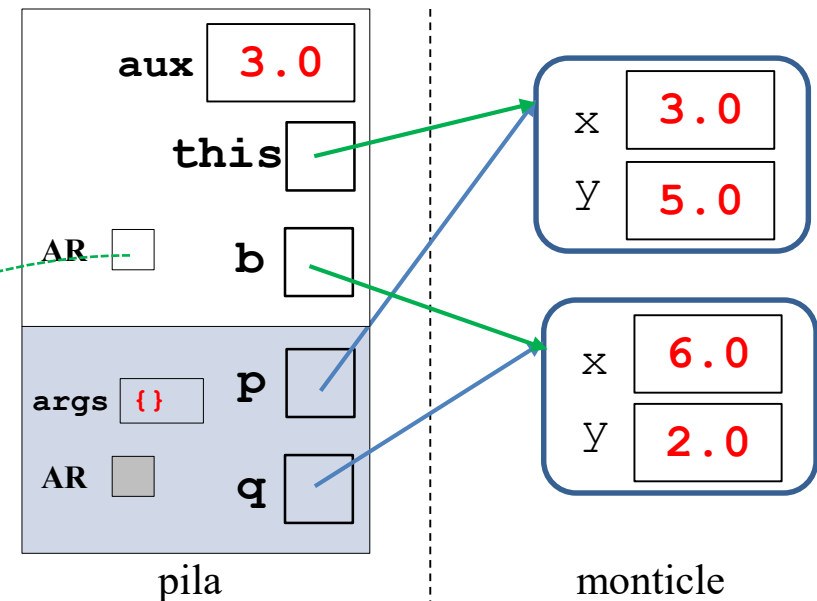
L'objecte no es còpia en el RA, només la seua referència
- d) Se realitza l'execució del codi del cos d'**intercanviX**, el mètode actiu.

Classe Punt

```
public void intercanviX(Punt b) {  
    double aux = this.x;  
    ➔ this.x = b.x;  
    b.x = aux;  
}
```

Classe PasParametres

```
public static void main(String[] args) {  
    ...  
    Punt p = new Punt(3.0, 5.0);  
    Punt q = new Punt(6.0, 2.0);  
    ➔ p.intercanviX(q);  
    ...  
}
```



# Execució d' (una crida a) un mètode: pas de paràmetres per valor – tipus referència

Exemple 2

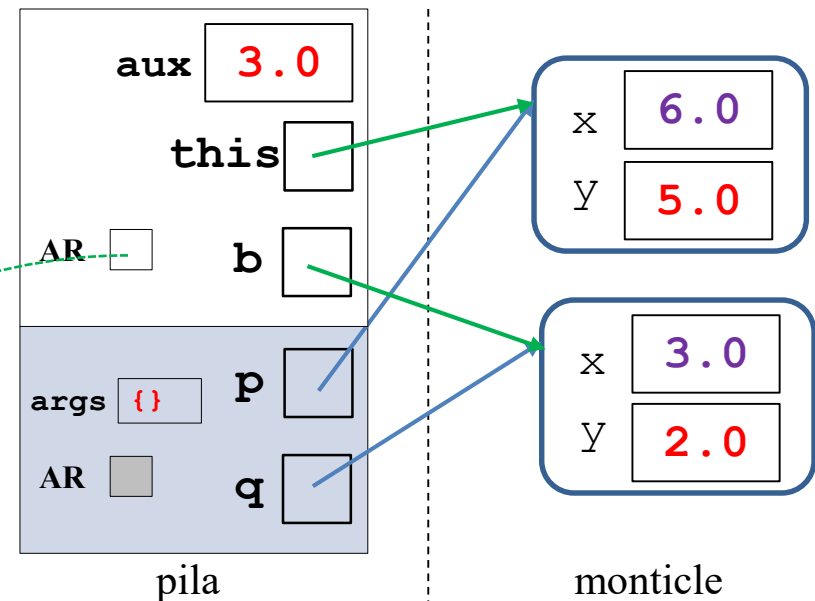
d) Se realitza l'execució del codi del cos d'**intercanviX**, el mètode actiu.

Classe Punt

```
public void intercanviX(Punt b) {  
    double aux = this.x;  
    this.x = b.x;  
    b.x = aux;  
}
```

Classe PasParametres

```
public static void main(String[] args) {  
    ...  
    Punt p = new Punt(3.0, 5.0);  
    Punt q = new Punt(6.0, 2.0);  
    → p.intercanviX(q);  
    ...  
}
```



# Execució d' (una crida a) un mètode: pas de paràmetres per valor – tipus referència

Exemple 2

- e) Acabada la crida a **intercanviX**, el seu registre d'activació s'allibera, tornant a ser el **main** el mètode actiu.

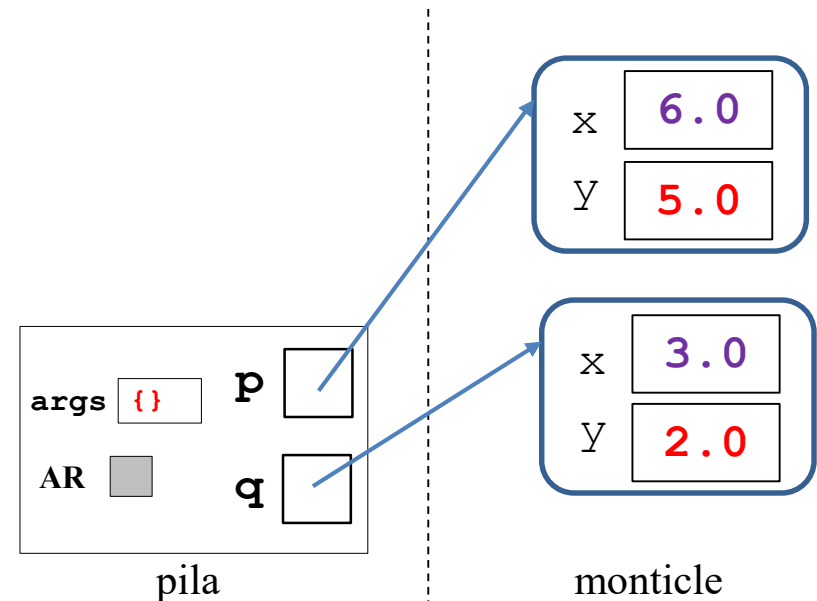
Quan es passa una **referència** a un objecte, SÍ que es possible accedir als seus atributs i modificar-los.  
L'objecte SÍ se modifica ja que no està en el RA

Classe Punt

```
public void intercanviX(Punt b) {  
    double aux = this.x;  
    this.x = b.x;  
    b.x = aux;  
}
```

Classe PasParametres

```
public static void main(String[] args) {  
    ...  
    Punt p = new Punt(3.0, 5.0);  
    Punt q = new Punt(6.0, 2.0);  
    p.intercanviX(q);  
    ...  
}
```



## Exercici 2: sobre pas de paràmetres per valor



- **CAP: Paso de parámetros: traza del programa Ejemplo2 (clave CCDHH4ai)**
  - Fer aquest exercici t'ajudarà a entendre el que és ...
    - Invocar l'execució d'un mètode des del cos d'un altre
    - Un argument i un paràmetre formal d'un mètode
    - El pas de paràmetres per valor (en Java)