



UNIVERSITAT  
POLITÈCNICA  
DE VALÈNCIA

Escola Tècnica Superior d'Enginyeria Informàtica



# Tema 2. Objectes, Classes i Programes

Introducció a la Informàtica i a la Programació (IIP)

Curs 2019/20

Departament de Sistemes Informàtics Computació



# Continguts

1. Introducció: estratègia POO de resolució d'un problema
2. Definició d'una classe Java: tipus i components
3. Creació i manipulació d'un objecte Java: operadors new i punt ( . )
4. Documentació de classes Java
5. Errors de compilació i execució de classes Java
6. Organització de classes Java en llibreries (*packages*). La llibreria estandard java.lang i les seues classes Object, String i System

Duració: 2 sessions



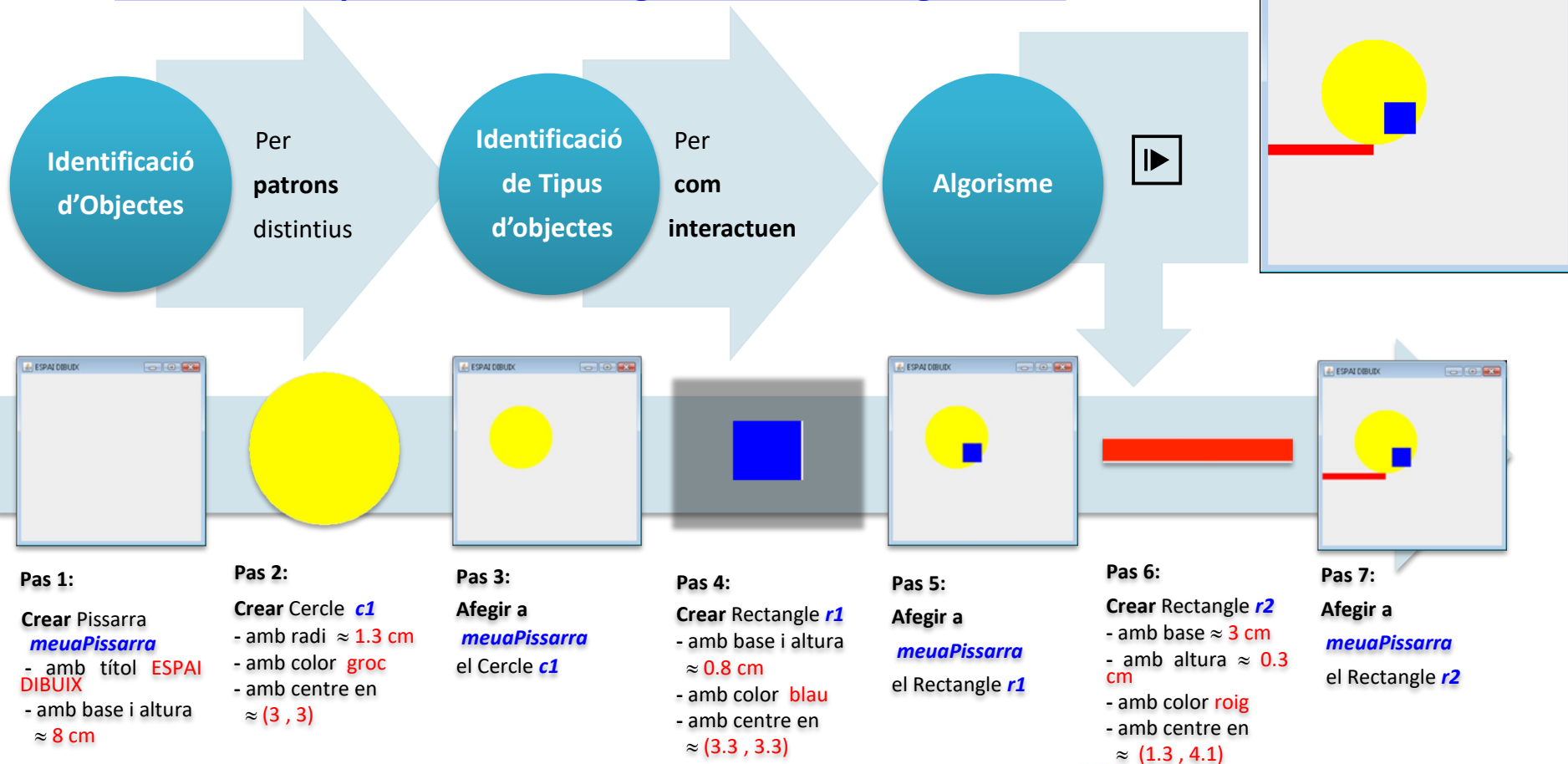
- **Crea** una carpeta **Tema 2** dins de la teua carpeta **W:\IIP\**
- **Descarrega** (del Tema 2 de PoliformaT) el fitxer **exemplesT2.jar** en **Tema 2**
- Des de l'opció **Projecte** de **BlueJ**, usa l'opció **Open ZIP/JAR...** per tal d'obrir aquest com un projecte **BlueJ** i prepara't per usar-lo



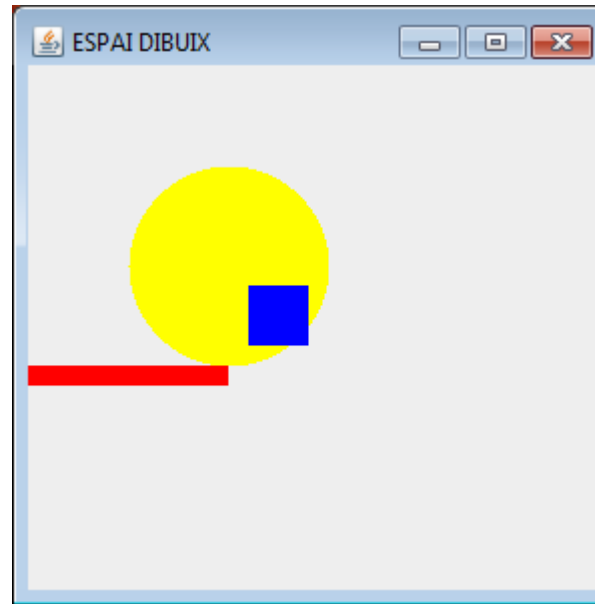
# Introducció: estratègia POO de resolució d'un problema (I). La fase MENTAL

**PROBLEMA:** es vol disposar d'un espai de dibuix, a mode de pissarra, que puga tindre un tamany variable, un títol i on es puguin dibuixar cercles i rectangles de diferents tamanyes, colors i en diferents posicions. En la pissarra es podria dibuixar, **PER EXEMPLE,**

Com s'obté aquest RESULTAT seguint una estratègia POO?



# Introducció: estratègia POO de resolució d'un problema (I). Un exemple



## Objectes identificats

Pissarra **meuaPissarra**, de títol **ESPAI DIBUIX** i tamany  $\approx 8 \times 8$  cm

Cercle **c1** de radi  $\approx 1.3$  cm, color **groc** i centre en  $\approx (3, 3)$

Rectangle **r1** de base=altura  $\approx 0.8$  cm, color **blau** i centre en  $(3.3, 3.3)$

Rectangle **r2** de base  $\approx 3$  cm, altura  $\approx 0.3$  cm, de color **roig** i centre en  $\approx (1.3, 4.1)$

## Patrons distintius

**Atributs:** títol, base, altura

**Accions:** Crear, Afegir una figura donada, ...

**Atributs:** radi, color, centre

**Accions:** Crear, Àrea, Obtenir o canviar radi, ...

**Atributs:** base, altura, color, centre

**Accions:** Crear, Àrea, Obtenir o canviar base, ...

## Tipus identificats

Pissarra

Pissarra **USA**  
Cercles

Cercle

Pissarra **USA**  
Rectangles

Rectangle

**Executable**  
(Crear objectes i Manipular-los)

¿ Algorisme ?

USA A

!!TOTS!!

# Introducció: estratègia POO de resolució d'un problema (I). Algunes definicions

- Un **objecte** es pot definir com una agrupació o col·lecció de **dades** i **operacions** que tenen una determinada estructura i mitjançant els quals es modelen aspectes rellevants d'un problema.
- Els objectes que comparteixen certes característiques i comportaments (**patrons distintius**) es diu que són del mateix **tipus** o de la mateixa **classe**.
- Una **classe** descriu el comportament de cadascun dels objectes. Es diu llavors que **l'objecte és una instància de la classe**.
- Java és un LOO i programar en Java consisteix a **escriure les definicions de les classes i utilitzar eixes classes per a crear objectes** de forma que representen adequadament el problema a resoldre.



# Introducció: estratègia POO de resolució d'un problema (II). La fase MENTALL

- Què és programar en Java? -



- Escriure en Java **la definició de cada Tipus** d'objectes que se crea i manipula en l'Algorisme  
**Classe Tipus de Dades**
- Escriure en Java **l'Algorisme** que descriu la resolució del problema, una instrucció per pas  
**Classe Programa**  
**¡¡Executable!!**

**Algorisme**

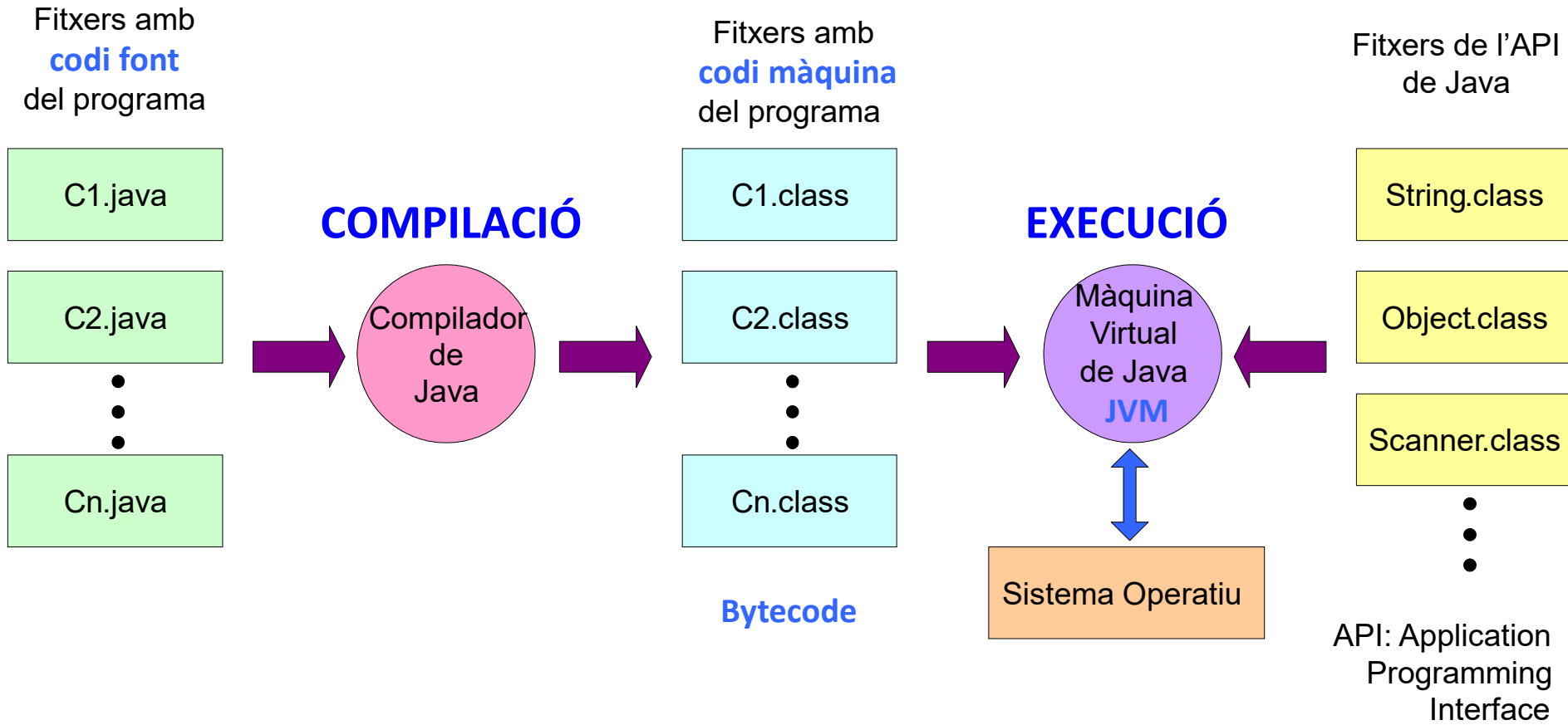
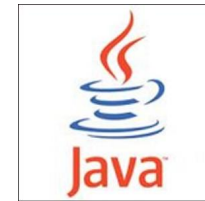
The BlueJ IDE interface, showing the BlueJ logo, a penguin icon, and logos for King's College London, University of Kent, and Oracle.

**Entorn de  
Desenvolupament Java  
- IDE -**

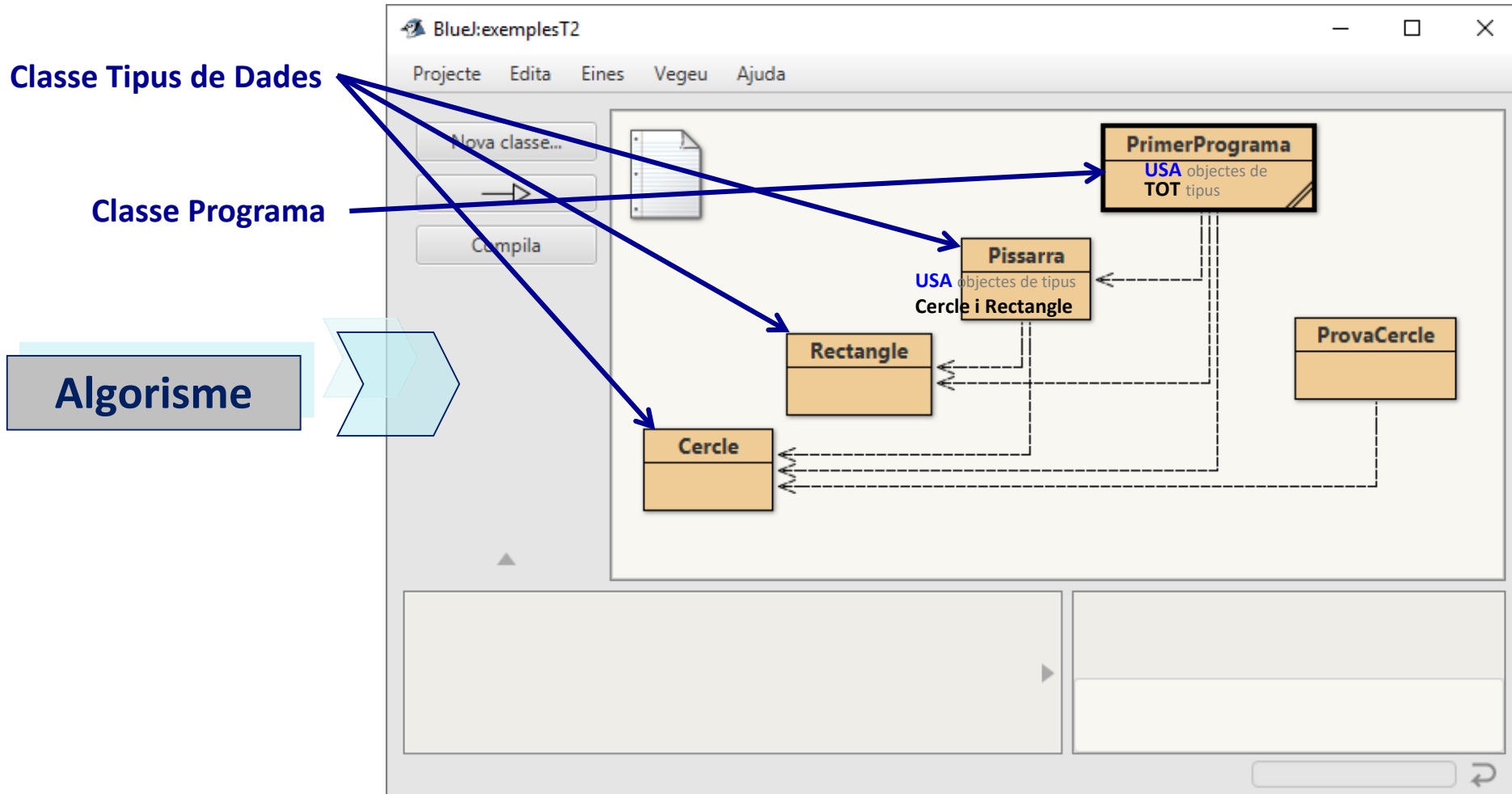
- Editar (**.java**)
- Compilar (**.class**)
- Executar (**bytecode**)

# Introducció: estratègia POO de resolució d'un problema (II). La fase MENTALL

- Què és programar en Java? -



# Introducció: estratègia POO de resolució d'un problema (II). Un exemple





# Definició d'una classe Java:

## tipus de classes

- Segons l'estructura de la classe i l'ús que es farà d'ella:
  - **Classe Tipus de dades**: definició d'un tipus d'objecte, que se fa codificant en Java els seus atributs i les operacions que es poden realitzar sobre aquests.
  - **Classe Programa**: les úniques executables, són les que inicien l'execució del codi.
  - **Classe d'Utilitats**: són magatzems d'operacions que poden utilitzar-se des d'altres classes.
- Segons qui siga l'**autor**: les teues classes i les predefinides –del estándar de Java- i les estandaritzades per altres autors.

# Definició d'una classe Java

## Exemple de Classe Programa (I)

```
public class PrimerPrograma {  
    public static void main (String[] args) {  
        // Crear meuaPissarra per a dibuixar, donant-li nom i dimensió  
        Pissarra meuaPissarra = new Pissarra("ESPAI DIBUIX", 300, 300);  
        // Crear un Cercle c1 de radi 50, groc, amb centre en (100,100)  
        Cercle c1 = new Cercle(50, "groc", 100, 100);  
        // Afegir-lo a la pissarra i dibuixar-lo  
        meuaPissarra.add(c1);  
        // Crear un Rectangle r1 de 30 per 30, blau, amb centre en (125,125)  
        Rectangle r1 = new Rectangle(30, 30, "blau", 125, 125);  
        // Afegir-lo a la pissarra i dibuixar-lo  
        meuaPissarra.add(r1);  
        // Crear un Rectangle r2 de 100 per 10, roig, amb centre en (50,155)  
        Rectangle r2 = new Rectangle(100, 10, "roig", 50, 155);  
        // Afegir-lo a la pissarra i dibuixar-lo  
        meuaPissarra.add(r2);  
    }  
}
```



BlueJ:exemplesT2

- **Observa** què ocorre quan s'executa el mètode main (en la transpa següent) i, després, **repeteix** el procés descrit.
- **Obre** la resta de classes del projecte *exemplesT2*,
  - Alguna més té un mètode main? Si es que sí, executa'l i observa què ocorre.

# Definició d'una classe Java

## Exemple de Classe Programa (II): execució

3

The screenshot displays a Java IDE environment. On the left, a window titled 'ESPAI DIBUIX' (Drawing Space) shows a yellow circle with a blue square inside it, and a red horizontal line below the circle. The main IDE window, titled 'niplesT2', shows a class hierarchy diagram with four classes: 'PrimerPrograma' at the top, followed by 'Pissarra', 'Rectangle', and 'Cercle' at the bottom. Dashed arrows indicate inheritance relationships from 'PrimerPrograma' to 'Pissarra', 'Rectangle', and 'Cercle', and from 'Pissarra' to 'Rectangle' and 'Cercle'. A context menu is open over the 'PrimerPrograma' class, listing actions: 'void main(String[] args)', 'Obre editor', 'Compila', 'Inspecciona', 'Elimina', 'Duplicate...', 'Convert to Stride', and 'Crea la classe de test'. A blue star with the number '1' is next to the 'void main(String[] args)' option. In the bottom right, a dialog box titled 'BlueJ: BlueJ: Crida a mètode' (Call Method) is open, showing the method signature 'void main(String[] args)' and the code 'PrimerPrograma.main( { } )'. A blue star with the number '2' is next to the dialog box. At the bottom of the dialog box are 'Aceptar' and 'Cancelar' buttons.

ESPAI DIBUIX

niplesT2

PrimerPrograma

Pissarra

Rectangle

Cercle

void main(String[] args)

Obre editor

Compila

Inspecciona

Elimina

Duplicate...

Convert to Stride

Crea la classe de test

BlueJ: BlueJ: Crida a mètode

void main(String[] args)

PrimerPrograma.main( { } )

Aceptar Cancelar

# Definició d'una classe Java

## Exemple de Classe Tipus de dades

```
/** Classe Cercle: defineix un cercle d'un determinat radi, color i ... */
```

```
public class Cercle {
```

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

VALORS

ESTAT  
de l'objecte



ATRIBUTS

```
/** crea un cercle de radi 50, negre i centre en (100,100)*/
```

```
public Cercle() { radi = 50; color = "negre";  
                centreX = 100; centreY = 100; }
```

```
/** torna el radi del cercle. */
```

```
public double getRadi() { return radi; }
```

```
/** actualitza el radi del cercle a nouRadi. */
```

```
public void setRadi(double nouRadi) { radi = nouRadi; }
```

```
/** redueix el radi del cercle en un factor de 1.3. */
```

```
public void decreix() { radi = radi / 1.3; }
```

```
/** torna l'àrea del cercle. */
```

```
public double area() { return Math.PI * radi * radi; }
```

```
/** torna un String amb les components del cercle. */
```

```
public String toString() {  
    return "Cercle de radi " + radi + ", color " + color  
        + " i centre (" + centreX + "," + centreY + ")";  
}
```

OPERACIONS

```
// més mètodes
```

COMPORTAMENT  
de l'objecte



MÈTODES

# Definició d'una classe Java

## Exemple de Classe Tipus de dades



BlueJ:exemplesT2

- **Obre** les classes **PrimerPrograma** i **Rectangle** del projecte:
  - Quina té atributs i quina no? Per què?
  - Quina té un mètode equivalent a **public cercle()** i quina no? Per què?

# Estructura bàsica d'una classe: capçalera i cos (atributs i mètodes)

```
[modificadors] class NomDeLaClasse [ extends AltraClasse ] {  
                                capçalera (de declaració)
```

```
// Definició d'atributs
```

```
[ [modificadors] tipus nomVar1;  
  [modificadors] tipus nomVar2;  
  ...  
  [modificadors] tipus nomVarN; ]
```

```
// Definició de mètodes
```

```
[ [modificadors] tipus nomMetode1 ([llistaParams]) { cos }  
  [modificadors] tipus nomMetode2 ([llistaParams]) { cos }  
  ...  
  [modificadors] tipus nomMetodeM ([llistaParams]) { cos } ]
```

```
}
```

**COS**



Blue!examplesT2

- A la vista de l'estructura general d'una classe Java ...
  - Què tenen en comú totes les classes del projecte?

# Estructura bàsica d'una classe: **modificadors** que defineixen l'àmbit de la declaració (*private* i *public*)

- Tota la informació declarada **private** és exclusiva de l'objecte i inaccessible des de fora de la classe.
  - Qualsevol intent d'accés als atributs **radi** o **color** que es realitze fora de la classe **cercle** (per exemple, en la classe **PrimerPrograma**) donarà lloc a un error de compilació.

```
private double radi; private String color;
```

- Tota la informació declarada **public** és accessible des de fora de la classe.
  - És el cas dels mètodes **getRadi()** o **area()** de la classe **cercle**.

```
/** torna el radi del cercle. */  
public double getRadi() { return radi; }  
/** torna l'àrea del cercle. */  
public double area() { return Math.PI * radi * radi; }
```

# Estructura bàsica d'una classe:

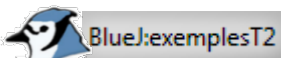
## atributs

- Els **atributs** o **variables d'instància** (nomVar1, nomVar2, ..., nomVarN) representen informació pròpia de cada objecte de la classe i es declaren d'un tipus de dades determinat, éssent definits habitualment d'accés privat.
- El tipus de dades defineix els valors que l'atribut pot prendre i les operacions que sobre ell es poden realitzar.

### // Definició d'atributs

```
[ [modificadors] tipus nomVar1;  
  [modificadors] tipus nomVar2;  
    ...      ...      ...  
  [modificadors] tipus nomVarN;  
]
```

```
public class Cercle {  
    private double radi;  
    private String color;  
    private int centreX,  
                centreY;  
    ...  
}
```



- **Obre** la classe **Rectangle** del projecte:
  - Quins atributs té? De quins tipus? Per quin modificador de visibilitat van precedits?



# Estructura bàsica d'una classe:

## mètodes

- Els **mètodes** defineixen les operacions que es poden aplicar sobre els objectes de la classe i es descriuen indicant:
  - La **capçalera o perfil**: nom, tipus del resultat i llista de paràmetres que es requereixen per al càlcul.
    - És possible que un mètode no retorne cap valor (tipus **void**)
  - El **cos**: conté la seqüència d'instruccions que s'efectuen en executar-lo.
    - la instrucció **return** és d'aparició obligada (excepte quan el resultat és **void**) i el seu efecte és retornar el resultat calculat.

Capçalera o

Perfil del mètode

Cos del mètode

Tipus del resultat    Identificador    Llista de paràmetres

```
public class Cercle {
```

...

```
public void setRadi ( double nouRadi ) { radi = nouRadi; }
```

```
public double area() { return Math.PI * radi * radi; }
```

```
}
```

# Estructura bàsica d'una classe:

## tipus de mètodes

- Segons la seua funció respecte a l'objecte, es classifiquen en:
  - **Constructors**: creen l'objecte i s'utilitzen per inicialitzar els seus atributs.
  - **Modificadors**: alteren l'estat de l'objecte, canviant els valors dels seus atributs (com, per exemple, `setRadi`).
  - **Consultors**: permeten conèixer, sense alterar, l'estat de l'objecte, tornant el valor d'un o més dels seus atributs (com, per exemple, `getRadi`).

```
public class Cercle {  
    ...  
    public Cercle() {  
        radi = 50; color = "negre"; centreX = 100; centreY = 100;  
    }  
    public Cercle(double r, String c, int px, int py) {  
        radi = r; color = c; centreX = px; centreY = py;  
    }  
    public double getRadi() { return radi; }  
    public void setRadi(double nouRadi) { radi = nouRadi; }  
}
```

*Constructor*

*Constructor*

*Consultor*

*Modificador*

- **Obre** la classe **Rectangle** del projecte i **classifica** els seus mètodes segons la seua funció respecte a l'objecte, com hem fet per a la classe **Cercle**.

# Estructura bàsica d'una classe:

## El mètode **main**

- Marca el punt d'inici de l'execució d'una aplicació (classe programa).
- El programa més senzill en Java s'implementa com una **classe** amb un únic mètode **main**.

```
public class NomDeLaClasse {  
    public static void main(String[] args) {  
        ...  
    }  
}
```

```
public class PrimerPrograma {  
    public static void main(String[] args) {  
        // Crear meaPissarra per a dibuixar, donant-li nom i dimensió  
        Pissarra meaPissarra = new Pissarra("ESPAI DIBUIX", 300, 300);
```

Creació i identificació d'un objecte

```
        // Crear un Cercle c1 de radi 50, groc, amb centre en (100,100)  
        Cercle c1 = new Cercle(50, "groc", 100, 100);
```

Creació i identificació d'un objecte

```
        // Afegir-lo a la Pissarra i dibuixar-lo  
        meaPissarra.add(c1);  
        ...
```

Invocació al mètode add

```
    }  
}
```

# Creació i manipulació d'un objecte Java:

## **operadors new i .**

- L'**operador new** s'utilitza per a crear un objecte d'una classe.
- L'**operador punt (.)** s'usa per a seleccionar l'atribut desitjat o el mètode específic que es vol aplicar (executar) sobre l'objecte.

```
public class PrimerPrograma {  
    // No s'usen objectes d'aquesta classe  
    private PrimerPrograma() { }  
  
    public static void main(String[] args) {  
        // Iniciar l'espai per a dibuixar donant-li nom i dimensió  
        Pissarra meaPissarra = new Pissarra("ESPAI DIBUIX", 300, 300);  
        // Crear un Cercle de radi 50, groc, amb centre en (100,100)  
        Cercle c1 = new Cercle(50, "groc", 100, 100);  
        // Afegir-lo a la Pissarra i dibuixar-lo  
        meaPissarra.add(c1);  
        ...  
    }  
}
```

# Estructura bàsica d'una classe:

## Blocs

- Java és un llenguatge orientat a blocs, o “unitats” de codi
- Delimitadors de bloc: claus d'inici ( { ) i final de bloc ( }
- **Blocs d'instruccions**
  1. Exemple: cos d'un mètode
  2. Seqüència de zero o més instruccions compreses entre les claus { i }
  3. Les instruccions estan separades per ; i seran executades, també, seqüencialment.
  4. Propòsit: agrupar EN UNA SOLA les instruccions que conté; d'aquesta forma, un bloc es pot utilitzar en els mateixos llocs que una instrucció simple.

# Documentació de classes Java

## Per què és important?

```
public class PrimerPrograma {  
    // No s'usen objectes d'aquesta classe  
    private PrimerPrograma() { }  
  
    public static void main(String[] args) {  
        // Iniciar l'espai per a dibuixar donant-li un nom  
        Pissarra meaPissarra = new Pissarra("ESPAI DIBUIX", 300, 300);  
        // Crear un Cercle de radi 50, groc, amb centre en (100,100)  
        Cercle c1 = new Cercle(50, "groc", 100, 100);  
        // Afegir-lo a la Pissarra i dibuixar-lo  
        meaPissarra.add(c1);  
        ...  
    }  
}
```

Si no sé res de la classe **Pissarra**, com puc usar-la?

Si no recorde algun detall de la classe **Cercle**, què faig?

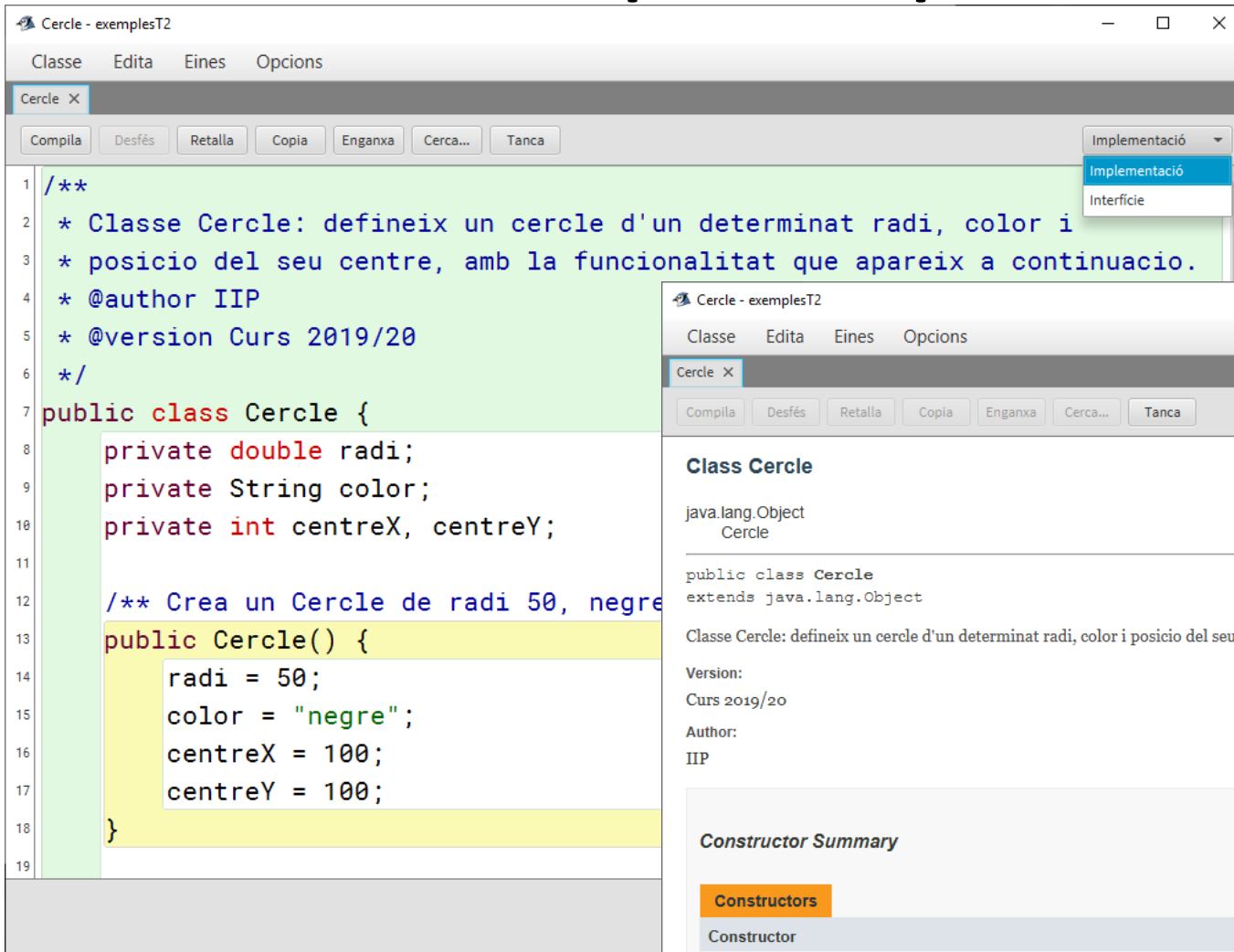


BlueJ:exemplesT2

- **Obre**, per exemple, la classe **Cercle** del projecte i canvia **Implementació** per **Interfície**, com en la següent transpa.
  - Què apareix en lloc del codi Java?

# Documentació de classes Java

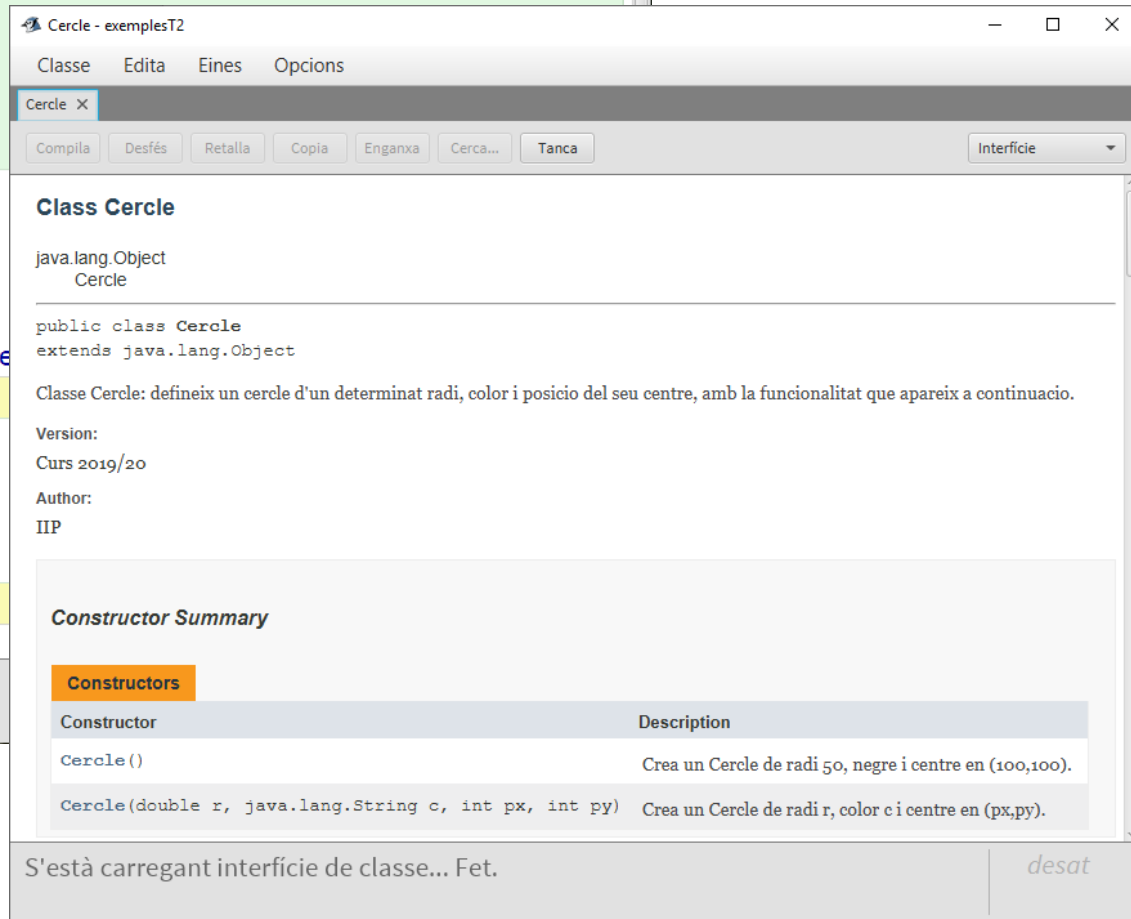
## Per què és important?



The image shows a screenshot of an IDE window titled 'Cercle - exemplesT2'. The main editor displays the following Java code:

```
1 /**
2  * Classe Cercle: defineix un cercle d'un determinat radi, color i
3  * posicio del seu centre, amb la funcionalitat que apareix a continuacio.
4  * @author IIP
5  * @version Curs 2019/20
6  */
7 public class Cercle {
8     private double radi;
9     private String color;
10    private int centreX, centreY;
11
12    /** Crea un Cercle de radi 50, negre
13     */
14    public Cercle() {
15        radi = 50;
16        color = "negre";
17        centreX = 100;
18        centreY = 100;
19    }
```

A dropdown menu is open, showing options: Implementació, Implementació, and Interficie. The 'Interficie' option is selected.



The screenshot shows the documentation window for the 'Cercle' class. The window title is 'Cercle - exemplesT2'. The 'Interficie' dropdown is selected. The documentation content is as follows:

**Class Cercle**

java.lang.Object  
Cercle

---

public class Cercle  
extends java.lang.Object

Classe Cercle: defineix un cercle d'un determinat radi, color i posicio del seu centre, amb la funcionalitat que apareix a continuacio.

Version:  
Curs 2019/20

Author:  
IIP

---

**Constructor Summary**

Constructors	Description
Cercle()	Crea un Cercle de radi 50, negre i centre en (100,100).
Cercle(double r, java.lang.String c, int px, int py)	Crea un Cercle de radi r, color c i centre en (px,py).

S'està carregant interfície de classe... Fet.

desat

# Documentació de classes Java

## Per què és important?

### Class Pissarra

```
java.lang.Object
  java.awt.Component
    java.awt.Container
      java.awt.Window
        java.awt.Frame
          javax.swing.JFrame
            Pissarra
```

#### All Implemented Interfaces:

```
java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable,
javax.accessibility.Accessible, javax.swing.RootPaneContainer, javax.swing.WindowConstants
```

```
public class Pissarra
  extends javax.swing.JFrame
```

Classe Pissarra: defineix una Pissarra sobre la que es poden dibuixar elements de tipus Cercle i Rectangle.

#### Version:

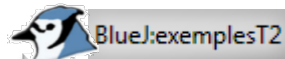
Curs 2019/20

#### Author:

IIP

#### See Also:

Serialized Form



- **Genera** la documentació de la classe **Pissarra** que es mostra (incompleta) en la següent imatge.



- **Consulta** la documentació de les classes predefinides de Java en:

<http://docs.oracle.com/en/java/javase/11/docs/api/>

### Constructor Summary

#### Constructors

##### Constructor and Description

**Pissarra()**

Crea una Pissarra per defecte en la que és possible situar elements gràfics.

**Pissarra(java.lang.String titol, int dimX, int dimY)**

Crea una Pissarra amb cert títol i tamany en la que es possible situar elements gràfics.

### Method Summary

#### All Methods Instance Methods Concrete Methods

##### Modifier and Type

##### Method and Description

void

**add(java.lang.Object o)**

Afegeix un objecte gràfic a la Pissarra i el dibuixa.

void

**dibuixaTot()**

Redibuixa tots els elements gràfics que s'han afegit a la Pissarra.



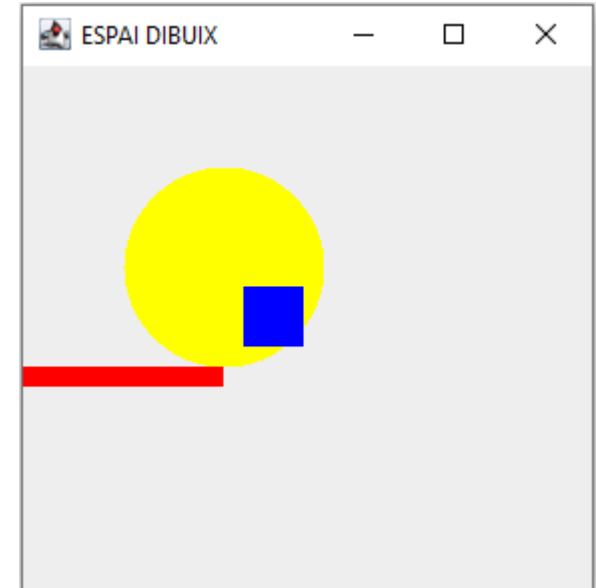
# Errors de compilació i execució en Java

## PrimerPrograma.java

```
/**Programa de prova de les classes Cercle, Rectangle i Pissarra
 * @author IIP
 * @version Curs 2019/20
 */
public class PrimerPrograma {
    public static void main (String[] args) {
        // Crear meuPissarra per a dibuixar, donant-li nom i dimensió
        Pissarra meuPissarra = new Pissarra("ESPAI DIBUIX",300,300);
        // Crear un Cercle c1 de radi 50, groc, amb centre en (100,100)
        Cercle c1 = new Cercle(50,"groc",100,100);
        // Afegir-lo a la pissarra i dibuixar-lo
        meuPissarra.add(c1);
        // Crear un Rectangle r1 de 30 per 30, blau, amb centre en (125,125)
        Rectangle r1 = new Rectangle(30,30,"blau",125,125);
        // Afegir-lo a la pissarra i dibuixar-lo
        meuPissarra.add(r1);
        // Crear un Rectangle r2 de 100 per 10, roig, amb centre en (50,155)
        Rectangle r2 = new Rectangle(100,10,"roig",50,155);
        // Afegir-lo a la pissarra i dibuixar-lo
        meuPissarra.add(r2);
    }
}
```

### Estil de codificació

- [Convencions Java](#)
- [Checkstyle](#)



javac PrimerPrograma.java

PrimerPrograma.class

... bytecodes ...

SII no hi ha errors d'execució

java PrimerPrograma

SII no hi ha errors de compilació

# Errors de compilació i execució en Java

- És possible que apareguen errors que impossibiliten l'execució d'un programa o que alteren el seu comportament respecte a allò que es pretén:
    - **Errors de compilació:** apareixen quan el programa incompleix alguna de les característiques de la definició del llenguatge.
      - El compilador del Java és de múltiple passada, està organitzat en fases que s'executen sols si s'han passat correctament les fases prèvies.
      - Generalment aquests errors són senzills de corregir gràcies a l'ajuda proporcionada pel compilador i a l'ús de la documentació del llenguatge.
    - **Errors d'execució:** provoquen un funcionament incorrecte del programa:
      - **Errors en temps d'execució:** provoquen la detenció de l'execució.
      - **Errors lògics:** provoquen que els resultats obtinguts no siguin correctes encara que el programa acabe amb aparent normalitat.
- En general, els errors d'execució poden ser difícils de detectar i resoldre.

# Errors de **compilació** i execució en Java

```
public class Cercle {  
    private double radi;  private String color;  
    private int centreX, centreY;  
  
    /** crea un cercle de radi 50, negre i centre en (100,100) */  
    public Cercle() { radi = 50; color = "negre";  
                    centreX = 100; centreY = 100; }  
  
    /** torna el radi del cercle. */  
    public double getRadi() { return radi; }  
  
    /** actualitza el radi del cercle a nouRadi. */  
    public void setRadi(double nouRadi) { radi = nouRadi; }  
  
    /** redueix el radi del cercle en un factor de 1.3. */  
    public        decreix() { radi = radi / 1.3   }  
  
    /** torna l'àrea del cercle. */  
    public double area() { return Math.PI * radi * radi; }  
  
    /** torna un String amb les components del cercle. */  
    public String toString       {  
        return "Cercle de radi " + radi + ", color " + color  
            + " i centre (" + centreX + "," + centreY + ")"; }  
  
    // més mètodes  
}
```



BlueJ:exemplesT2

- **Obre** la classe **Cercle** del projecte i **modifica** el seu codi com s'indica en aquesta transpa subratllat en roig. Després compila... què passa?

# Errors de **compilació** i execució en Java

```
/** Programa de prova de les classes Cercle, Rectangle i Pissarra
 * @author IIP
 * @version Curs 2019/20
 */
public class PrimerPrograma {
    // No s'usen objectes d'aquesta classe
    private PrimerPrograma() { }

    public static void main(String[] args) {
        // Crear meaPissarra per a dibuixar, donant-li nom i dimensió
        Pissarra meaPissarra;

        // Crear un Cercle c1 de radi 50, groc, amb centre en (100,100)
        Cercle c1 = new Cercle(50, "groc", 100, 100);
        // Afegir-lo a la pissarra i dibuixar-lo
        meaPissarra.add(c1);

        // Crear un Rectangle r1 de 100 per 10, roig, amb centre en (25,125)
        Rectangle r1 = new Rectangle(100, 10, "roig", 25, 125);
        // Afegir-lo a la Pissarra i dibuixar-lo
        meaPissarra.add(r1);

        // Crear un Rectangle r2 de 100 per 10, roig, amb centre en (50,155)
        Rectangle r2 = new Rectangle(100, 10, "roig", 50, 155);
        // Afegir-lo a la pissarra i dibuixar-lo
        meaPissarra.add(r2);
    }
}
```

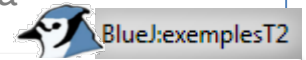
variable meaPissarra might not have been initialized

- **Obre** la classe **PrimerPrograma** del projecte i **modifica** el seu codi com s'indica en aquesta transpa. Després compila... què passa?

# Errors de compilació i **execució** en Java

## (**en temps d'execució** –excepcions–)

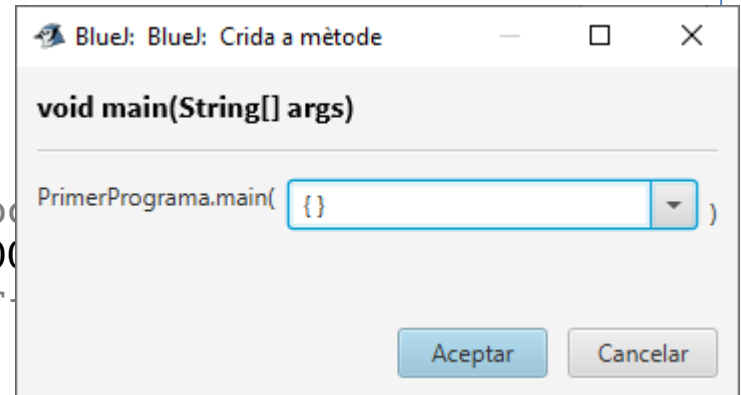
```
/** Programa de prova de les classes Cercle, Rectangle i Pissarra
 * @author IIP
 * @version Curs 2019/20
 */
public class PrimerPrograma {
    // No s'usen objectes
    private PrimerPrograma
```



- **Obre** la classe **PrimerPrograma** del projecte i **modifica** el seu codi com s'indica en aquesta transpa. Després compila i executa... què passa?

```
public static void main(String[] args) {
    // Crear meaPissarra per a dibuixar,
    Pissarra meaPissarra = null;

    // Crear un Cercle c1 de radi 50, groc
    Cercle c1 = new Cercle(50, "groc", 100);
    // Afegir-lo a la pissarra i dibuixar
    meaPissarra.add(c1);
```



```
// C
Rect
// A
meua
```

```
// C
Rect
// A
meua

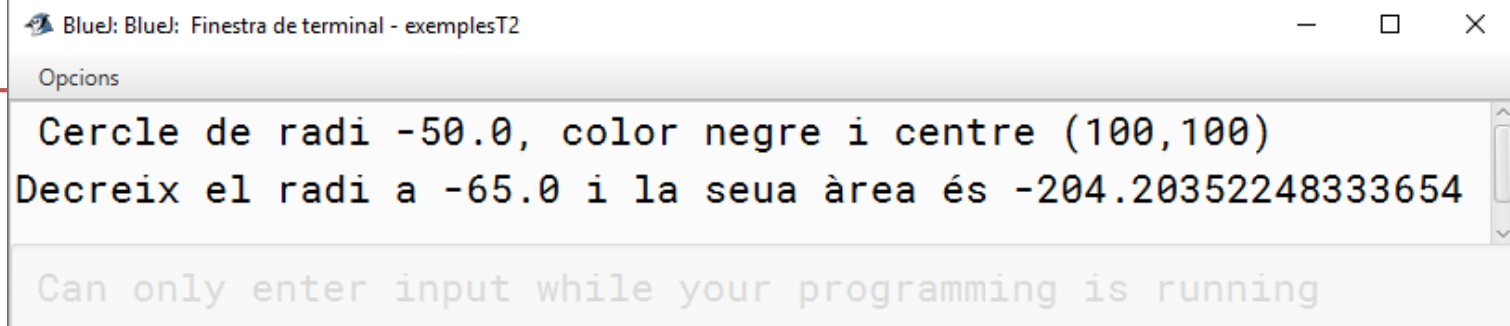
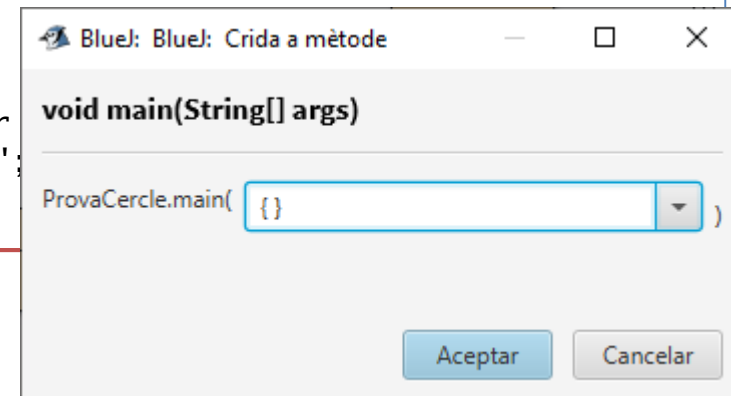
Can only enter input while your programming is running

java.lang.NullPointerException
    at PrimerPrograma.main(PrimerPrograma.java:18)
```

# Errors de compilació i execució en Java (lògics)

```
public class Cercle {
    private double radi; private String color;
    private int centreX, centreY;
    /** crea un cercle de radi 50, negre i centre en (100,100) */
    public Cercle() { radi = -50; color = "negre"; centreX = 100; centreY = 100; }
    /** consulta el radi del cercle. */
    public double getRadi() { return radi; }
    /** actualitza el radi del cercle a nouRadi. */
    public void setRadi(double nouRadi) { radi = nouRadi; }
    /** redueix el radi del cercle en un factor de 1.3. */
    public void decreix() { radi = radi * 1.3; }
    /** torna l'àrea del cercle. */
    public double area() { return Math.PI * radi; }
    /** obté un String amb les components del cercle. */
    public String toString() {
        return "Cercle de radi " + radi + ", color " + color
            + " i centre (" + centreX + ", " + centreY + ")";
    }
    // més mètodes
}
```

```
public class ProvaCercle {
    private ProvaCercle() { }
    public static void main(String[] args) {
        Cercle c1 = new Cercle(); System.out.println(c1.toString());
        c1.decreix(); System.out.print("Decreix el radi a " + c1.getRadi());
        System.out.println(" i la seva àrea és " + c1.area());
    }
}
```



# Organització de classes Java en llibreries (packages).

- Un **paquet** (**package**) de Java consisteix en un grup de classes que poden ser importades i, en conseqüència, utilitzades en altres classes.
- En Java, les classes s'estructuren sempre en paquets. Quan no s'indica explícitament, estan en un especial (**anonymous**).
- Un **package** facilita l'organització i ús de les classes ja definides i la definició i ús de noves.

```
package libUtil;  
import javax.swing.*;  
import java.awt.*;  
public class Pissarra extends JFrame { ... }
```

- El paquet **java.lang** s'importa per defecte. Formen part d'aquest paquet les classes **Object**, **String**, **System** i **Math**.

# Herència. Jerarquia de classes Java

- En els LOO el mecanisme bàsic per al reús del codi és l'**herència**: permet definir noves classes extenent o restringint les funcionalitats d'altres classes ja existents.
- L'herència permet modelar relacions jeràrquiques entre classes: l'hereua té les mateixes característiques, tal volta refinades per a definir-la com un cas especial.
- La llibreria de classes del llenguatge es troba organitzada de forma jeràrquica, éssent la base d'aquesta jerarquia **Object**.

