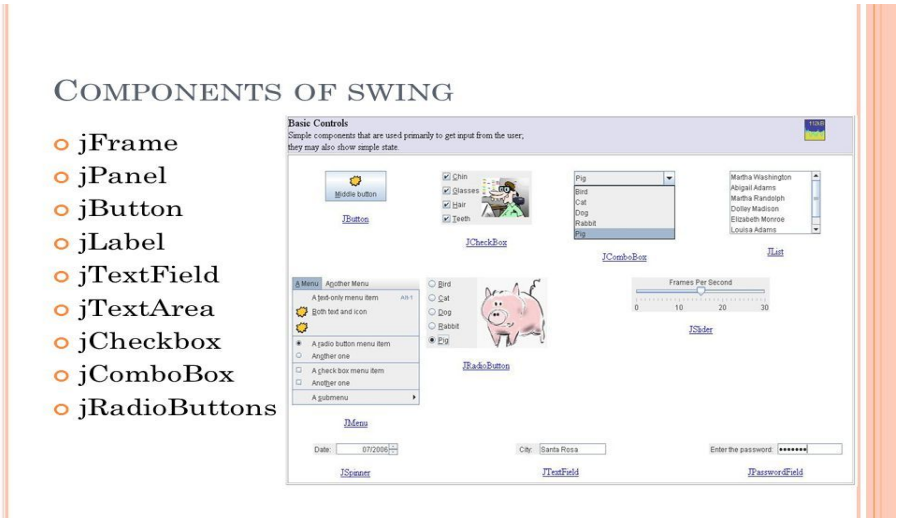
U.T. 6 - PROGRAMACIÓ VISUAL



1 - INTERFÍCIES D'USUARI

Java Swing és una llibreria de classes lleugera per a ús d'interfícies gràfiques d'usuari (GUI, Graphic User Interface) que ens inclou un ampli conjunt de components (*widgets*) independents de la plataforma. Està construïda sobre la llibreria de components gràfics originària de Java (AWT), un conjunt d'eines GUI més antic que depèn de la plataforma. Algun dels components més habituals són el botó, la caixa de text, l'etiqueta, etc.

Vegem el nostre primer exemple amb una finestra que únicament incloga una etiqueta:

```
import javax.swing.*;
import java.util.*;

public class HMG
{
    public static void main (String [] args)
    {
        // CREEM LA FINESTRA PRINCIPAL DE L'APLICACIÓ
        JFrame frame = new JFrame("Hola Món");
        // AIXÒ FARÀ QUE EL PROGRAMA ACABE AL TANCAR LA FINESTRA
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // ES CREA L'ETIQUETA, INICIALMENT INDEPENDENT DE LA FINESTRA
        JLabel label = new JLabel("Hola DAM1");
        // S'AFEGEIX L'ETIQUETA A LA FINESTRA
        frame.getContentPane().add(label);
        // ESTABLEIX LA MIDA DE LA FINESTRA PER DEFECTE AL VALOR MÍNIM DELS COMPONENTS
        frame.pack();
```

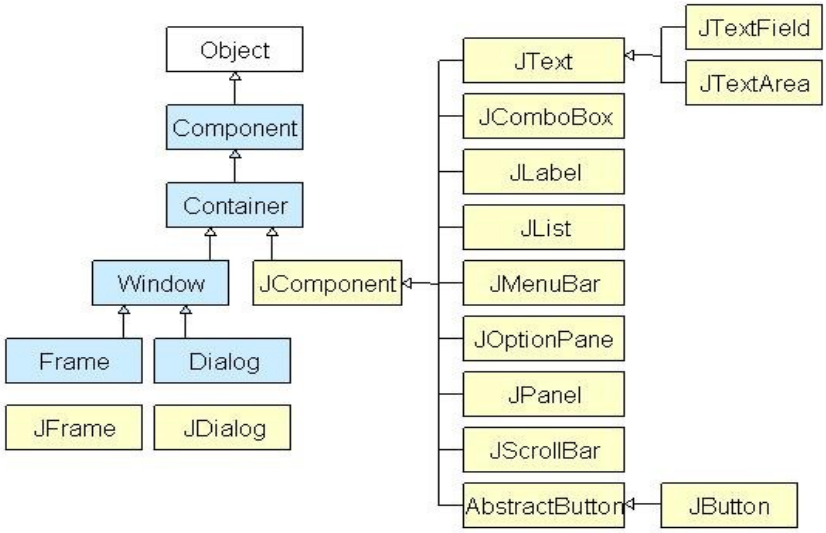
```
        // CENTREM LA FINESTRA AMB EL VALOR NULL
        frame.setLocationRelativeTo(null);
        // A MÉS, CAL FER LA FINESTRA VISIBLE
        frame.setVisible(true);
    }
}
```

2 - COMPONENTS SWING

La figura anterior ens mostra els components (*widgets*) principals, més utilitzats, de la llibreria Swing. Per fer ús d'ells utilitzarem la documentació de les classes corresponents. Així, pots consultar, com a exemple d'algunes de les operacions més freqüents (lectura i escriptura del text associat a algun d'aquests components), els mètodes *gettext()* i *setText(String s)*. També la documentació d'Oracle, que al seu portal web ens ofereix un complet tutorial de l'ús de Swing: https://docs.oracle.com/javase/tutorial/uiswing/. I, en especial, l'apartat referit a l'ús de cada un dels seus components: https://docs.oracle.com/javase/tutorial/uiswing/components/index.html.

Podràs executar els exemples de la pàgina amb alguna aplicació com *javaws*, inclosa al paquet *IcedTea-netx*, al temps que pots veure el seu codi.

La següent figura mostra les relacions entre aquestes diferents classes.

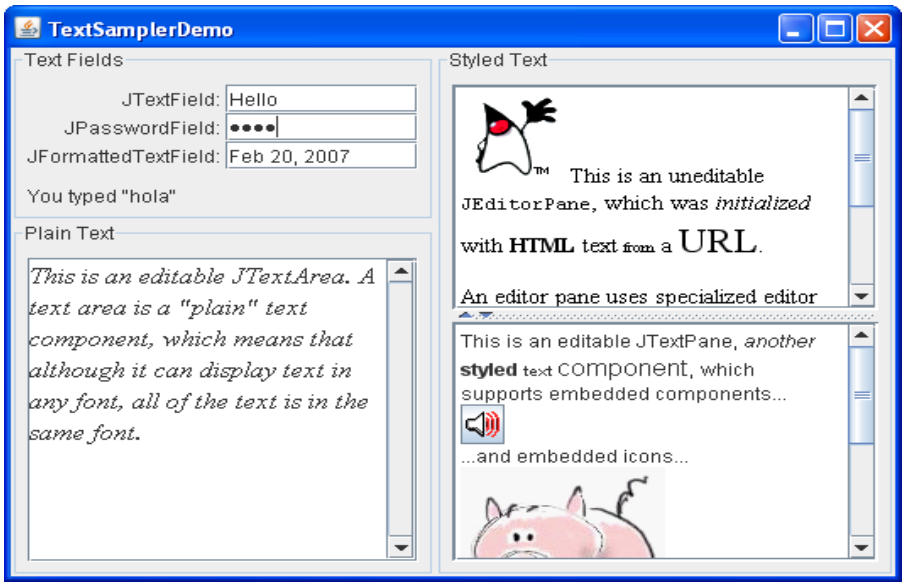


Vegem, a mode d'exemple, com treballar amb components de text. Aquests components (*JTextField* o *JTextArea*, per exemple) mostren text i, opcionalment, permeten l'usuari modificar aquest text. Swing inclou sis components de text instanciables, a part d'altres classes i interfícies de suport per a operacions

de text més complexes. Malgrat els seus diferents usos i possibilitats, totes elles hereten de les mateixes superclasses *JText* y *JComponent*.

Centrant-nos en les més senzilles d'aquestes classes, *JTextField* i les seves subclasses *JPasswordField* i *JFormattedTextField*, són classes que poden mostrar només una línia de text editable. Com altres controls no textuals, els botons per exemple, poden generar esdeveniments (per exemple, la pulsació de INTRO). El seu ús habitual és permetre llegir un text curt teclejat per l'usuari i realitzar alguna acció, indicada en el nostre codi, com a resposta.

En aquelles ocasions que necessitem treballar amb textos més llargs que una frase curta, la classe a utilitzar serà *JTextArea*. El seu ús és molt similar al de *JTextField*.



Un exemple d'ús:

```
JTextField TextField = new JTextField("text inicial");
textField.setColumns(20);
// afegim el TextField al panell que corresponga
panel.add(TextField);
// passats 10 segons (10000 ms) canvie el text
Thread.sleep(10000);
textField.setText("Un altre text diferent");
String content = textField.getText();
```

3 - CONTENIDORS SWING

Les classes de contenidor són aquelles que poden tenir altres components i permeten distribuir-los convenientment. Per a crear una GUI, necessitem sempre com a mínim un objecte contenidor exterior (generalment *JFrame*) i, molt freqüentment, altres interiors. Hi han 3 tipus de contenidors:

Panell: és un contenidor pur i no és una finestra en si mateixa. L'únic propòsit d'un panell és organitzar els components en una finestra.

Marc: és una finestra en ple funcionament amb el seu títol i icones.

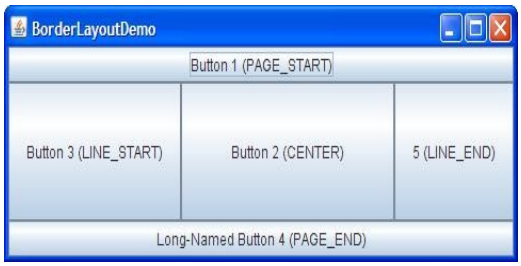
Diàleg: es pot considerar com una finestra emergent, o secundària, que apareix quan s'ha de mostrar un missatge. No és una finestra completament funcional com un marc.

A l'hora de disposar components, cal pensar que l'usuari generalment podrà canviar la mida de la nostra finestra i que l'aplicació, en aquests casos, ha de mantenir una aparença coherent, amb dimensions reduïdes (en aquest cas fins i tot podrien quedar ocults components fonamentals) o maximitzades. Per a això és important decidir com gestionar els components al seu interior. La decisió pot ser disposar-los un

a continuació d'un altre, d'esquerra a dreta o al revés, en files i columnes, etc. Per això comptem amb els "layouts managers" o gestors de disposició.

Java Layout Manager: el gestor de disseny s'usa per dissenyar (o organitzar) els components Java de la GUI dins d'un contenidor. Hi han molts administradors de disseny, però els més utilitzats són:

- *BorderLayout*: col·loca els components en fins a cinc àrees: amunt, avall, esquerra, dreta i centre.



- *FlowLayout*: és l'administrador de disseny predeterminat per a cada *JPanel*. Simplement estableix els components en una sola fila, un després de l'altre. En principi, ho fa d'esquerra a dreta, però podem fer-ho al revés. Observa la documentació per a aquesta classe.

- *GridLayout*: permet la disposició de components en forma de matriu, en files i columnes.

- *GridBagLayout*: és el més sofisticat de tots els dissenys. Alinea els components col·locant-los dins d'una graella de cel·les, permetent que els components ocupen més d'una cel·la.

Per assignar a una finestra el gestor corresponent, per exemple *GridLayout*, faríem:

```
frame.setLayout(new GridLayout(0,1));
```

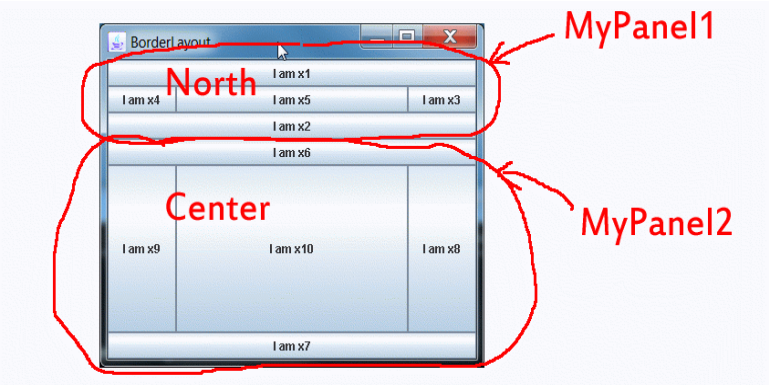
Això establiria una disposició de components tots en una mateixa columna (1, segon paràmetre) amb qualsevol nombre de files (0 al primer paràmetre). Els components s'afegirien posteriorment fent ús del mètode *add()*:

```
frame.getContentPane().add(label);
frame.getContentPane().add(btnlimpia);
frame.getContentPane().add(btnescribe);
```

El mètode *getContentPane()* retorna el contenidor associat, en aquest cas, a l'objecte *JFrame*. En la documentació de *JFrame* se'ns explica que cada objecte d'aquesta classe conté un *JRootPane*. *JRootPane* és un contenidor lleuger utilitzat per *JFrame* o també per *JDialog*. Aquest panell de contingut

proporcionat pel panell arrel hauria de contenir tots els components que no són de menú i que mostra el *JFrame*.

De vegades, ens pot interessar afegir diferents panells a la nostra finestra principal (diferents objectes *JPanel* dins d'un mateix *JFrame*). En aquest cas, podem, per exemple afegir un gestor de tipus *GridLayout* a la finestra per afegir diferents panells disposats en forma de matriu, i després cada panell interior podria tenir el seu propi gestor diferenciat (per exemple *FlowLayout*).



4 - ESDEVENIMENTS I CONTROLADORS D'ESDEVENIMENTS

En Java, cada esdeveniment està representat per un objecte d'una subclasse d'*EventObject*, en el paquet *java.util*. Cada subclasse d'*EventObject* representa un tipus d'esdeveniment particular. Per exemple:

-*MouseEvent*, per a representar accions del ratolí: moure, arrossegar, fer clic a un botó del ratolí, etc.

-*KeyEvent*, per a representar accions del teclat, és a dir, prémer tecles.

-*ActionEvent*, per representar un acció de l'usuari a la interfície, per exemple, prémer un botó a la pantalla.

No obstant això, en el model d'esdeveniments de Java, els *EventObject* no realitzen accions per si mateixos, sinó que aquests esdeveniments són enviats a un altre objecte encarregat de respondre a un tipus d'esdeveniment en particular. Aquests objectes són els que coneixem com *listeners* o "escoltadors", i hi ha diferents *listeners* que "escolten" als diferents esdeveniments, com ara els *mouse listeners*, *key listeners* o *action listeners* (més generals).

Els *listeners* no s'implementen com classes en Java, sinó com a interfícies. Una interfície seria aleshores una col·lecció de mètodes que defineixen un comportament en particular. D'aquesta manera, qualsevol classe que subministre informació per a aquests mètodes pot declarar que implementa aquesta interfície.

El gran avantatge d'implementar interfícies, front a la possibilitat d'heretar d'altres classes, és que una mateixa classe pot implementar diverses interfícies simultàniament però només pot heretar d'una.

Els listeners que farem servir seran al paquet *java.awt.event* (que haurem d'importar) i ens podem trobar els següents:

Nom: **ActionListener**

Descripció: es produeix al fer clic en un component, també si es prem *Intro* tenint el focus en el component.

Mètode: public void actionPerformed(ActionEvent e)

esdeveniments:

JButton: clic o prémer Enter amb el focus activat en ell.

JList: doble clic en un element de la llista.

JMenuItem: selecciona una opció de menú.

JTextField: al prémer Enter amb el focus activat.

Nom: **KeyListener**

Descripció: es produeix al prémer una tecla.

mètodes:

public void keyTyped(KeyEvent i) // en acabar de teclejar

public void keyPressed(KeyEvent i) // en pressionar la tecla

public void keyReleased(KeyEvent i) // en soltar la tecla

Esdeveniments: Quan premem una tecla, segons el listener:

keyTyped: al prémer i deixar anar la tecla.

keyPressed: al prémer la tecla.

keyReleased: al deixar anar la tecla.

Nom: **FocusListener**

Descripció: es produeix quan un component guanya o perd el focus, és a dir, canviar el seu estat de seleccionat o no.

Nom: **MouseListener**

Descripció: es produeix quan realitzem una acció amb el ratolí.

mètodes:

public void mouseClicked(MouseEvent e)

public void mouseEntered(MouseEvent e)

public void mouseExited(MouseEvent e)

public void mousePressed(MouseEvent e)

public void mouseReleased(MouseEvent e)

Esdeveniments (segons el listener):

mouseClicked: punxar i deixar anar.

mouseEntered: entrar en un component amb el punter.

mouseExited: eixir d'un component amb el punter.

mousePressed: pressionar el botó.

mouseReleased: deixar anar el botó.

Nom: **MouseMotionListener**

Descripció: es produeix amb el moviment del ratolí.

mètodes:

public void mouseDragged(MouseEvent e)

public void mouseMoved(MouseEvent e)

Esdeveniments: segons el listener:

mouseDragged: clic i arrossegar un component.

mouseMoved: al moure el punter sobre un element.

En general, podem consultar els *listeners* suportats per cada component en l'enllaç https://docs.oracle.com/javase/tutorial/uiswing/events/eventsandcomponents.html.

D'aquesta manera, els listeners es poden invocar amb l'ús d'una classe anònima:

```
import java.awt.event.*;

boton1.addActionListener (new ActionListener() {
        public void actionPerformed() {
    // Accions
    }
});
```

o d'aquesta altra forma que prèviament crea l'objecte *listener*:

```
import java.awt.event.*;

ActionListener al = new ActionListener() {
        public void actionPerformed() {
    // Accions
    }
};
boton1.addActionListener(al);
```

Centrant-nos per exemple en *JButton*: la mecànica per a atrapar el clic es pot fer mitjançant la implementació d'un *ActionListener*, com acabem de veure, incloent la definició del mètode associat a aquest esdeveniment, en aquest cas *actionPerformed* (observa el mètode i la interfície en la documentació).

Exemple 1: Finestra que mostre un botó. Quan es pressione finalitzarà l'execució del programa Java:

```
import javax.swing.*;
import java.awt.event.*;

public class Formulari extends JFrame implements ActionListener {
        JButton boto;
        public Formulari() {
                setLayout(null);
```

```
                boto = new JButton("Finalitzar");
                boto.setBounds(300,250,100,30);
                add(boto);
                boto.addActionListener(this);
    }

    public void actionPerformed (ActionEvent e) {
        if (e.getSource() == boto)
                System.exit(0);
    }

    public static void main (String [] args) {
                Formulari formulari1 = new Formulari();
                formulari1.setBounds(0,0,450,350);
                formulari1.setVisible(true);
    }
}
```

Exemple 2: Finestra que continga tres objectes de la classe *JButton* amb les etiquetes "1", "2" i "3". Al pressionar han de canviar el títol del *JFrame* indicant quin botó es va pressionar:

```
import javax.swing.*;
import java.awt.event.*;

public class Formulari extends JFrame implements ActionListener {

    private JButton boto1, boto2, boto3;

    public Formulari() {
                setLayout(null);

                boto1 = new JButton("1");
                boto1.setBounds(10,100,90,30);
                add(boto1);
                boto1.addActionListener(this);

                boto2 = new JButton("2");
                boto2.setBounds(110,100,90,30);
                add(boto2);
                boto2.addActionListener(this);

                boto3 = new JButton("3");
                boto3.setBounds(210,100,90,30);
                add(boto3);
                boto3.addActionListener(this);
    }

    public void actionPerformed(ActionEvent e) {
            if (e.getSource() == boto1)
                    setTitle("botó 1");
            if (e.getSource() == boto2)
                    setTitle ("botó 2");
            if (e.getSource() == boto3)
                    setTitle ( "botó 3");
```

```
    public static void main (String args[]) {
            Formulari formulari1 = new Formulari();
            formulari1.setBounds(0,0,350,200);
            formulari1.setVisible(true);
    }
}
```

5 - ESDEVENIMENTS AMB EXPRESSIONS LAMBDA

Als exemples anteriors hem vist com reaccionar als esdevinements dels objectes, com el clic a un botó. Ho podem fer de tres formes diferents:

1. instanciant un objecte d'una classe anònima que implemente la interfície (en el cas d'un botó, pot ser *ActionListener*). Es la opció utilitzada a l'exemple 1 de l'apartat anterior.
2. fent que la classe principal implemente la interfície. És la opció utilitzada a l'exemple 2 de l'apartat anterior. El codi resulta més simple però obliga a fer ús del mètode *getSource()*.
3. fent ús d'una expressió lambda. Vegem el mateix exemple 1 resolt d'esta forma.

```
import javax.swing.*;
import java.awt.event.*;

public class Formulari extends JFrame {

    JButton boto;
    public Formulari() {
            setLayout(null);
            boto = new JButton("Finalitzar");
            boto.setBounds(300,250,100,30);
            add(boto);
            ActionListener lambda = e -> System.exit(0);
            boto.addActionListener(lambda);
    }

    public static void main (String [] args) {
            Formulari formulari1 = new Formulari();
            formulari1.setBounds(0,0,450,350);
            formulari1.setVisible(true);
    }
}
```

Per a entendre l'exemple hem d'exposar prèviament alguns conceptes. Una vegada implementada la expressió lambda, per al seu ús es defineix una referència a la interfície funcional assignant-li la expressió lambda. En el cas de l'esdeveniment del botó seria:

ActionListener al = e -> System.exit(0);

En aquest cas evitem generar una nova classe que implemente *ActionListener* amb la definició del mètode *actionPerformed*. Així queda el codi molt més compacte. Inclús no es requereix especificar el paràmetre de tipus *ActionEvent*, perquè el compilador de Java el pot deduïr.

6 - JAVAFX

JavaFX es una llibreria de Java dissenyada per a oferir als desenvolupadors de Java una nova plataforma gràfica d'alt rendiment i lleugeresa, i amb major flexibilitat. La intenció és que les noves aplicacions utilitzen JavaFX, en comptes de Swing, per generar la interfície gràfica d'usuari (GUI) de l'aplicació. Això no vol dir que Swing siga obsolet. La gran quantitat d'aplicacions en ús que s'han construït amb Swing significa que encara formaran part de l'API de Java durant molt de temps.

Originalment, l'enfocament de la plataforma JavaFX incloia un llenguatge de scripts JavaFX pensat per facilitar la creació d'una interfície basada en web. Les versions de JavaFX que reflecteixen aquesta arquitectura van ser anteriors a la versió 2. Durant eixa primera vida de JavaFX, mai va quedar molt clar si JavaFX substituiria Swing. Després que Oracle es va fer càrrec de l'administració de Java, el focus es va desplaçar per a convertir JavaFX en la plataforma gràfica d'elecció en tot tipus d'aplicacions Java.

A l'Octubre de 2011, es va llançar JavaFX 2.0. Això va marcar el final del llenguatge de scripting JavaFX. Això significava que els desenvolupadors de Java ja no necessitaven aprendre un nou llenguatge gràfic i, en lloc d'això, poden crear les aplicacions JavaFX amb la sintaxi normal de Java. L'API de JavaFX conté tot el que s'espera d'una plataforma gràfica: controls de la UI, animacions, efectes, etc.

La principal diferència per als desenvolupadors que canvien de Swing a JavaFX és com es defineixen els components gràfics i la nova terminologia. La interfície d'usuari es construeix amb una sèrie de capes que es troben dins d'un gràfic d'**escena (Scene)** . El gràfic d'escena es mostra en un contenidor de nivell superior anomenat **escenari (Stage)**.

Altra característica destacables de JavaFX 2.0 és la de incloure un nou llenguatge de marcatge declaratari anomenat FXML. Està basat en XML i permet als desenvolupadors definir una interfície d'usuari per a una aplicació JavaFX.

```
import javafx.application.Application;
import javafx.scene.canvas.Canvas;
import javafx.scene.Scene;
import javafx.scene.Group;
import javafx.stage.Stage;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.shape.ArcType;
public class HappyFace extends Application
{
        public static void main(String[] args)
        {
                launch(args);
        }
        @Override
        public void start(Stage primaryStage) throws Exception
        {
                Group root = new Group();
                Scene scene = new Scene(root);
                Canvas canvas = new Canvas(400, 300);
                GraphicsContext gc = canvas.getGraphicsContext2D();
                gc.strokeOval(100, 50, 200, 200);
                gc.fillOval(155, 100, 10, 20);
                gc.fillOval(230, 100, 10, 20);
                gc.strokeArc(150, 160, 100, 50, 180, 180, ArcType.OPEN);
                root.getChildren().add(canvas);
                primaryStage.setTitle("HappyFace en JavaFX");
                primaryStage.setScene(scene);
                primaryStage.show();
        }
}
```

Observa a l'exemple com la classe fa un *extends* de la classe *Application* indicant que estem definint una aplicació JavaFX. L'aplicació conté dos mètodes, *main* i *start*. Un programa JavaFX comença la seua execució en el mètode *start*. El mètode *main* és ignorat normalmente. No obstant, és pràctica comuna incloure-lo amb la crida a *launch*, que acabarà per invocar a *start*.

```
@Override
public void start(Stage primaryStage) throws Exception
```

JavaFX utilitza la metàfora de *stage* i *scene*, com l'escenari i les escenes d'un teatre. Les següent línies generen un quadre o tela (*canvas*) en una escena per a dibuixar gràfics.

```
Group root = new Group();
Scene scene = new Scene(root);
Canvas canvas = new Canvas(400, 300);
GraphicsContext gc = canvas.getGraphicsContext2D();
```

Arribats a aquest punt, ja podem dibuixar:

```
gc.strokeOval(100, 50, 200, 200);
```

que dibuixa el cercle gran que forma la línia exterior del rostre, amb paràmetres mesurats en pixels. Els dos primers paràmetres indiquen el punt en el que el cercle és dibuixat. El mètode *strokeOval* dibuixa ovals. Els dos últims paràmetres indiquen l'ample i l'altura de l'oval. Per a un cercle, tots dos han de coincidir. El mètode *fillOval* és equivalent però omplint l'interior.