**TP2(partie 2)**

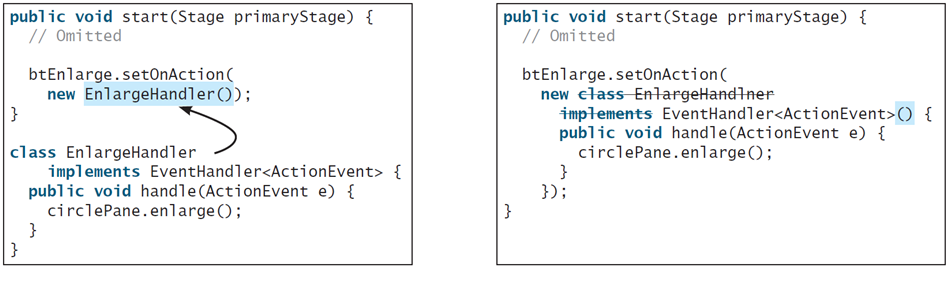
Activation des composants d’une interface - partie 2

1. **Ecoute et traitement des évènements par classe anonyme**

Pour répondre aux évènements, on utilise plusieurs alternatives :

* Par une classe interne à la classe dessinant l’interface (TP2 - partie 1)
* Par une classe interne anonyme

**Modèle**

****

1. Classe interne *EnlargeHandler* b) Classe anonyme equivalente

**Exercice** **1**:

Cet exemple permet d’afficher des messages sur la console en fonction du bouton cliqué.

**import** javafx.application.Application;

**import** javafx.event.ActionEvent;

**import** javafx.event.EventHandler;

**import** javafx.geometry.Pos;

**import** javafx.scene.Scene;

**import** javafx.scene.control.Button;

**import** javafx.scene.layout.HBox;

**import** javafx.stage.Stage;

**public** **class** AnonymousHandlerDemo **extends** Application {

**public** **void** start(Stage primaryStage) {

// Hold two buttons in an HBox

HBox hBox = **new** HBox();

hBox.setSpacing(10);

hBox.setAlignment(Pos.***CENTER***);

Button btNew = **new** Button("New");

Button btOpen = **new** Button("Open");

Button btSave = **new** Button("Save");

Button btPrint = **new** Button("Print");

hBox.getChildren().addAll(btNew, btOpen, btSave, btPrint);

// Create and register the handler

btNew.setOnAction(**new** EventHandler<ActionEvent>() {

@Override // Override the handle method

**public** **void** handle(ActionEvent e) {

System.***out***.println("Process New");

}

});

btOpen.setOnAction(**new** EventHandler<ActionEvent>() {

@Override // Override the handle method

**public** **void** handle(ActionEvent e) {

System.***out***.println("Process Open");

}

});

btSave.setOnAction(**new** EventHandler<ActionEvent>() {

@Override // Override the handle method

**public** **void** handle(ActionEvent e) {

System.***out***.println("Process Save");

}

});

btPrint.setOnAction(**new** EventHandler<ActionEvent>() {

@Override // Override the handle method

**public** **void** handle(ActionEvent e) {

System.***out***.println("Process Print");

}

});

// Create a scene and place it in the stage

Scene scene = **new** Scene(hBox, 300, 50);

primaryStage.setTitle("AnonymousHandlerDemo"); // Set title

primaryStage.setScene(scene); // Place the scene in the stage

primaryStage.show(); // Display the stage

}

**public** **static** **void** main(String[] args) {

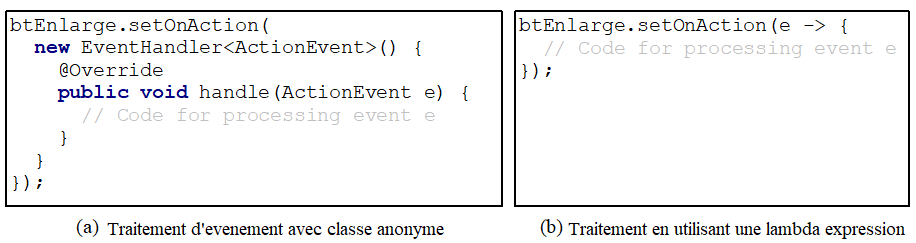
*launch*(args);

}

}

1. **Simplification des traitements d’évènements par les Lambda Expressions**

Les *Lambda expressions* sont de nouveaux mécanismes en Java 8. Elles peuvent être interprétées comme des méthodes anonymes avec une syntaxe plus concise (plus simple et plus courte).



**Syntaxe de base d’une lambda expression :**

Il y-a 2 types de lambda expression :

(type1 param1, type2 param2, ...) -> expression

ou

(type1 param1, type2 param2, ...) -> { instructions; }

Le type d’un paramètre peut être explicitement déclaré ou implicitement déduit par le compilateur. On peut omettre les parenthèses en cas de paramètre unique sans type explicite.

Les lamdba expressions peuvent être utilisées à condition que l’interface de l’écouteur ne contienne qu’une seule méthode à implémenter (la méthode *handle* dans ce cas). Les instructions à utiliser seront celles de cette méthode. Cette interface est connue par une interface SAM (*Single Abstract Method* SAM interface).

**Exercice 2** : en reprenant l’exemple 1 avec les lambda expression, on obtient :

import javafx.application.Application;

import javafx.event.ActionEvent;

import javafx.geometry.Pos;

import javafx.scene.Scene;

import javafx.scene.control.Button;

import javafx.scene.layout.HBox;

import javafx.stage.Stage;

public class LambdaHandlerDemo extends Application {

public void start(Stage primaryStage) {

// Hold two buttons in an HBox

HBox hBox = new HBox();

hBox.setSpacing(10);

hBox.setAlignment(Pos.CENTER);

Button btNew = new Button("New");

Button btOpen = new Button("Open");

Button btSave = new Button("Save");

Button btPrint = new Button("Print");

hBox.getChildren().addAll(btNew, btOpen, btSave, btPrint);

// Create and register the handler

btNew.setOnAction((ActionEvent e) -> {

System.out.println("Process New");

});

btOpen.setOnAction((e) -> {

System.out.println("Process Open");

});

btSave.setOnAction(e -> {

System.out.println("Process Save");

});

btPrint.setOnAction(e -> System.out.println("Process Print"));

// Create a scene and place it in the stage

Scene scene = new Scene(hBox, 300, 50);

primaryStage.setTitle("LambdaHandlerDemo"); // Set title

primaryStage.setScene(scene); // Place the scene in the stage

primaryStage.show(); // Display the stage

}

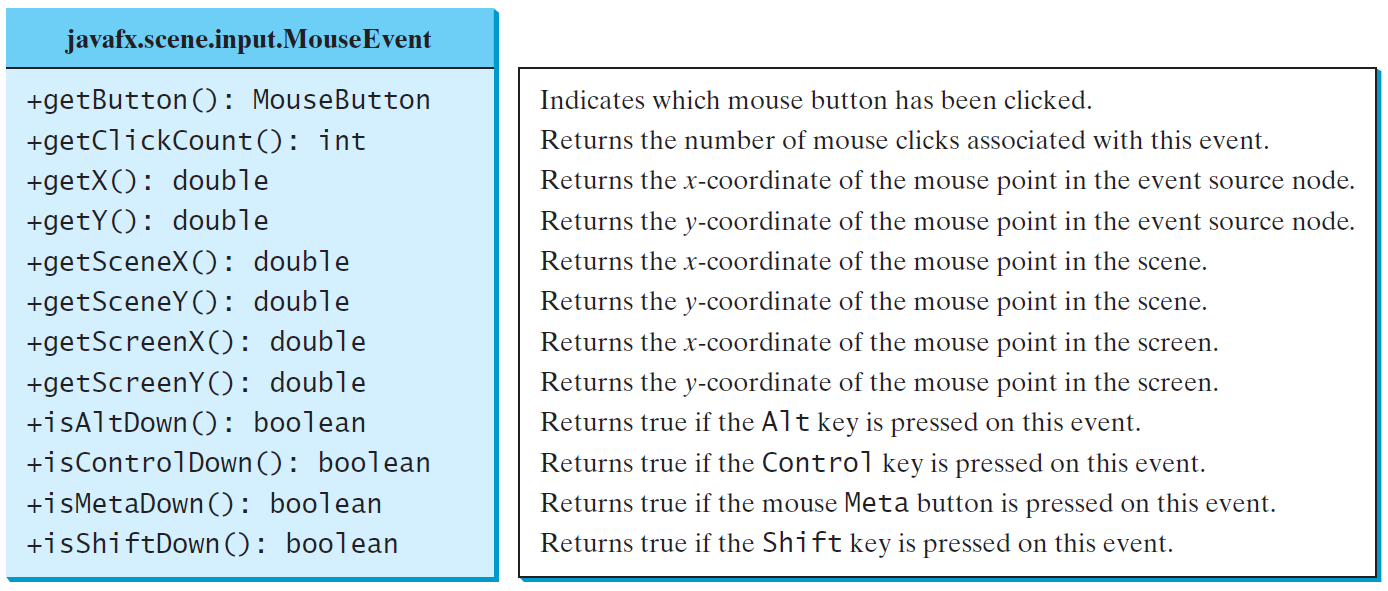
public static void main(String[] args) {

launch(args);

}

1. **La classe MouseEvent**

Nous disposons d’un ensemble de méthodes pour manipuler les évènements de cette classe :



**Exercice 3** :

package TP3;

import javafx.application.Application;

import javafx.scene.Scene;

import javafx.scene.layout.Pane;

import javafx.scene.text.Text;

import javafx.stage.Stage;

public class MouseEventDemo extends Application {

public void start(Stage primaryStage) {

// Create a pane and set its properties

Pane pane = new Pane();

Text text = new Text(20, 20, "Programming is fun");

pane.getChildren().addAll(text);

text.setOnMouseDragged(e -> {

text.setX(e.getX());

text.setY(e.getY());

});

// Create a scene and place it in the stage

Scene scene = new Scene(pane, 300, 100);

primaryStage.setTitle("MouseEventDemo"); // Set the stage title

primaryStage.setScene(scene); // Place the scene in the stage

primaryStage.show(); // Display the stage

}

public static void main(String[] args) {

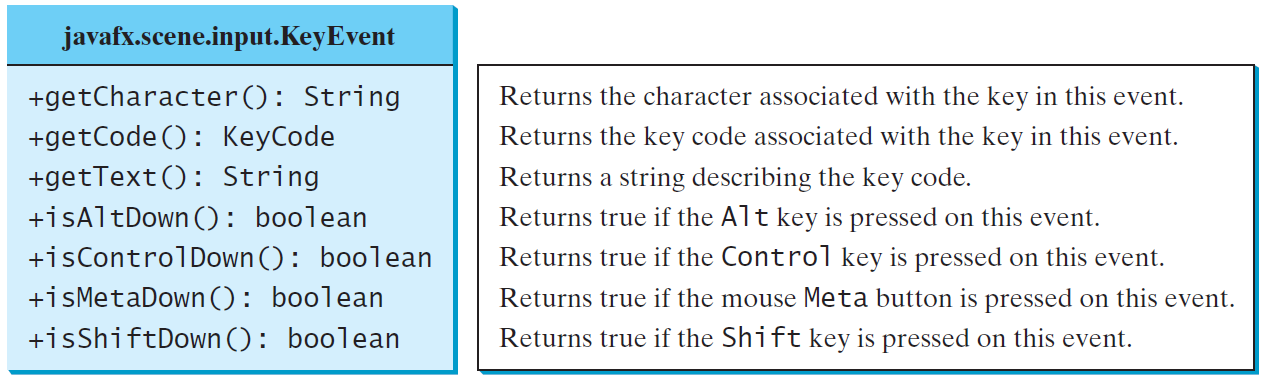
launch(args);

}

}

1. **La classe KeyEvent**

Nous disposons d’un ensemble de méthodes pour manipuler les évènements de cette classe :



**Exercice 4** :

import javafx.application.Application;

import javafx.scene.Scene;

import javafx.scene.layout.Pane;

import javafx.scene.text.Text;

import javafx.stage.Stage;

public class KeyEventDemo extends Application {

public void start(Stage primaryStage) {

// Create a pane and set its properties

Pane pane = new Pane();

Text text = new Text(20, 20, "A");

pane.getChildren().add(text);

// Create a scene and place the pane in the stage

Scene scene = new Scene(pane);

primaryStage.setTitle("KeyEventDemo"); // Set the stage title

primaryStage.setScene(scene); // Place the scene in the stage

primaryStage.show(); // Display the stage

text.setOnKeyPressed(e -> {

switch (e.getCode()) {

case DOWN: text.setY(text.getY() + 10); break;

case UP: text.setY(text.getY() - 10); break;

case LEFT: text.setX(text.getX() - 10); break;

case RIGHT: text.setX(text.getX() + 10); break;

default:

if (e.getText().length() > 0)

text.setText(e.getText());

}

});

text.requestFocus(); // text is focused to receive key input

}

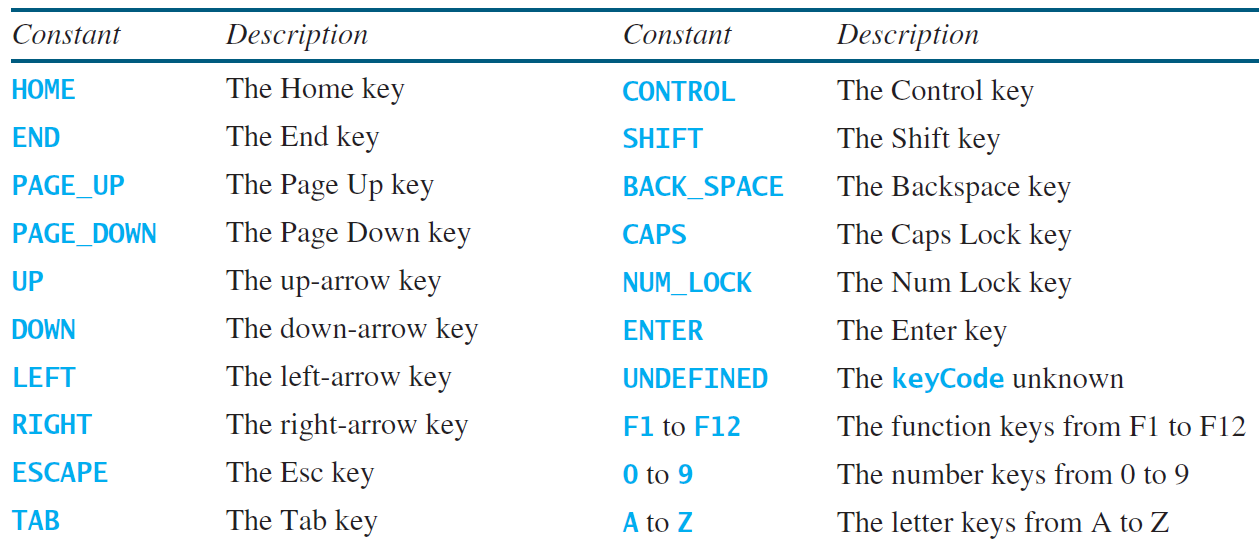
public static void main(String[] args) {

launch(args);

}

}

Constantes JavaFX représentant les codes des différentes touches du clavier



**Exercice 5** : utilisation des classes : ActionEvent, MouseEvent et KeyEvent :

package TP3;

import javafx.application.Application;

import javafx.geometry.Pos;

import javafx.scene.Scene;

import javafx.scene.control.Button;

import javafx.scene.input.KeyCode;

import javafx.scene.input.MouseButton;

import javafx.scene.layout.HBox;

import javafx.scene.layout.StackPane;

import javafx.scene.paint.Color;

import javafx.scene.shape.Circle;

import javafx.scene.layout.BorderPane;

import javafx.stage.Stage;

public class ControlCircleWithMouseAndKey extends Application {

private CirclePane circlePane = new CirclePane();

public void start(Stage primaryStage) {

HBox hBox = new HBox();

hBox.setSpacing(10);

hBox.setAlignment(Pos.CENTER);

Button btEnlarge = new Button("Enlarge");

Button btShrink = new Button("Shrink");

hBox.getChildren().add(btEnlarge);

hBox.getChildren().add(btShrink);

// Create and register the handler

btEnlarge.setOnAction(e -> circlePane.enlarge());

btShrink.setOnAction(e -> circlePane.shrink());

BorderPane borderPane = new BorderPane();

borderPane.setCenter(circlePane);

borderPane.setBottom(hBox);

BorderPane.setAlignment(hBox, Pos.CENTER);

// Create a scene and place it in the stage

Scene scene = new Scene(borderPane, 200, 150);

primaryStage.setTitle("ControlCircle"); // Set the stage title

primaryStage.setScene(scene); // Place the scene in the stage

primaryStage.show(); // Display the stage

circlePane.setOnMouseClicked(e -> {

if (e.getButton() == MouseButton.PRIMARY) {

circlePane.enlarge();

}

else if (e.getButton() == MouseButton.SECONDARY) {

circlePane.shrink();

}

});

scene.setOnKeyPressed(e -> {

if (e.getCode() == KeyCode.UP) {

circlePane.enlarge();

}

else if (e.getCode() == KeyCode.DOWN) {

circlePane.shrink();

}

});

}

class CirclePane extends StackPane {

private Circle circle = new Circle(50);

public CirclePane() {

getChildren().add(circle);

circle.setStroke(Color.BLACK);

circle.setFill(Color.WHITE);

}

public void enlarge() {

circle.setRadius(circle.getRadius() + 2);

}

public void shrink() {

circle.setRadius(circle.getRadius() > 2 ?

circle.getRadius() - 2 : circle.getRadius());

}

}

public static void main(String[] args) {

launch(args);

}

}