

Cours R2.02

Introduction à l'Interaction Humain-Machine

Cours 3 : Widgets et événements (1/2)

Plan du cours en 9 semaines

2

1. Introduction à l'interaction, placement
2. Programmation événementielle
- 3. Widgets et événements (1/2)**
4. Widgets et événements (2/2)
5. Conception et prototypage (1/2)
6. Conception et prototypage (2/2)
7. Heuristiques et recommandations
8. Modèles et théories
9. Méthodes d'évaluation des IHM

Compteur

3



Interface EventHandler

4

```
package javafx.event;

import java.util.EventListener;

// PENDING_DOC_REVIEW
/**
 * Handler for events of a specific class / type.
 *
 * @param <T> the event class this handler can handle
 * @since JavaFX 2.0
 */
@FunctionalInterface
public interface EventHandler<T extends Event> extends EventListener {
    /**
     * Invoked when a specific event of the type for which this handler is
     * registered happens.
     *
     * @param event the event which occurred
     */
    void handle(T event);
}
```

1ère solution : une classe externe par bouton

5

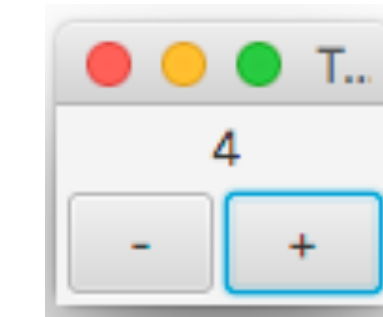
```
public class TestButtonIncDec extends Application {
    public void start(Stage stage) {
        Label label = new Label("0");
        Button bmoins = new Button(" - ");
        Button bplus = new Button(" + ");

        bplus.addEventHandler(ActionEvent.ACTION, new ClicListenerInc(label));
        bmoins.addEventHandler(ActionEvent.ACTION, new ClicListenerDec(label));

        VBox vbox = new VBox(3);
        vbox.setPadding(new Insets(3, 3, 3, 3));
        vbox.setAlignment(Pos.CENTER);
        HBox hbox = new HBox(3);
        hbox.getChildren().addAll(bmoins, bplus);
        vbox.getChildren().addAll(label, hbox);

        Scene scene = new Scene(vbox);
        stage.setTitle("TestButtonIncDec");
        stage.setScene(scene);
        stage.show();
    }

    public static void main(String[] args) {
        Application.launch(args);
    }
}
```



```
public class ClicListenerInc implements EventHandler<ActionEvent> {
    Label label;

    ClicListenerInc(Label lbl) {
        label = lbl;
    }

    public void handle(ActionEvent event) {
        int newValue = Integer.parseInt(label.getText()) + 1;
        label.setText("" + newValue);
    }
}
```

2e solution : une classe interne

6

```
public class TestButtonIncDec2 extends Application {
    Label label;
    Button bmoins, bplus;

    class ClicListenerIncDec implements EventHandler<ActionEvent> {
        public void handle(ActionEvent event) {
            int currentValue = Integer.parseInt(label.getText());
            if (event.getTarget() == bplus) {
                label.setText("" + (currentValue + 1));
            } else {
                label.setText("" + (currentValue - 1));
            }
        }
    }

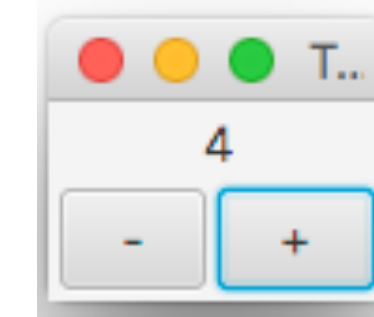
    public void start(Stage stage) {
        label = new Label("0");
        bmoins = new Button(" - ");
        bplus = new Button(" + ");

        bplus.addEventHandler(ActionEvent.ACTION, new ClicListenerIncDec());
        bmoins.addEventHandler(ActionEvent.ACTION, new ClicListenerIncDec());

        VBox vbox = new VBox(3);
        vbox.setPadding(new Insets(3, 3, 3, 3));
        vbox.setAlignment(Pos.CENTER);
        HBox hbox = new HBox(3);
        hbox.getChildren().addAll(bmoins, bplus);
        vbox.getChildren().addAll(label, hbox);

        Scene scene = new Scene(vbox);
        stage.setTitle("TestButtonIncDec2");
        stage.setScene(scene);
        stage.show();
    }

    public static void main(String[] args) {
        Application.launch(args);
    }
}
```



Classes internes en Java (inner class)

7

Une classe définie à l'intérieur d'une autre classe

Une classe interne peut accéder aux membres de sa classe englobante, même s'ils sont privés

3e solution : deux classes internes anonymes

8

```
public class TestButtonIncDec3 extends Application {

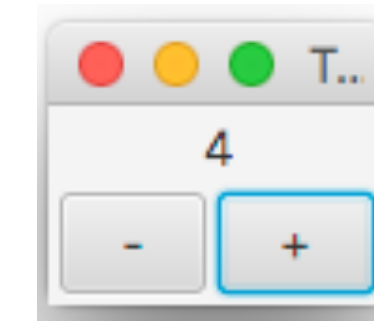
    public void start(Stage stage) {
        Label label = new Label("0");
        Button bmoins = new Button(" - ");
        Button bplus = new Button(" + ");

        bplus.addEventHandler(ActionEvent.ACTION, new EventHandler<ActionEvent>(){
            public void handle(ActionEvent event) {
                int newValue = Integer.parseInt(label.getText()) + 1;
                label.setText("" + newValue);
            }
        });
        bmoins.addEventHandler(ActionEvent.ACTION, new EventHandler<ActionEvent>(){
            public void handle(ActionEvent event) {
                int newValue = Integer.parseInt(label.getText()) - 1;
                label.setText("" + newValue);
            }
        });

        VBox vbox = new VBox(3);
        vbox.setPadding(new Insets(3, 3, 3, 3));
        vbox.setAlignment(Pos.CENTER);
        HBox hbox = new HBox(3);
        hbox.getChildren().addAll(bmoins, bplus);
        vbox.getChildren().addAll(label, hbox);

        Scene scene = new Scene(vbox);
        stage.setTitle("TestButtonIncDec2");
        stage.setScene(scene);
        stage.show();
    }

    public static void main(String[] args) {
        Application.launch(args);
    }
}
```



Classes internes anonymes

9

Une classe interne sans nom est une classe interne anonyme

Elles sont déclarées et instanciées en même temps

4e solution : implémenter directement EventHandler<ActionEvent>

10

```
public class TestButtonIncDec4 extends Application implements EventHandler<ActionEvent> {
    Label label;
    Button bmoins, bplus;

    public void start(Stage stage) {
        label = new Label("0");
        bmoins = new Button(" - ");
        bplus = new Button(" + ");

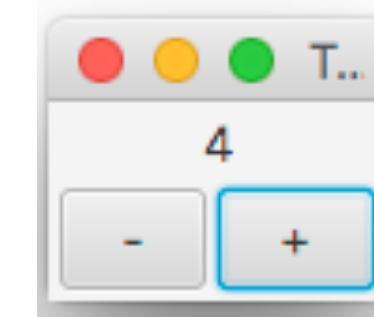
        bplus.addEventHandler(ActionEvent.ACTION, this);
        bmoins.addEventHandler(ActionEvent.ACTION, this);

        VBox vbox = new VBox(3);
        vbox.setPadding(new Insets(3, 3, 3, 3));
        vbox.setAlignment(Pos.CENTER);
        HBox hbox = new HBox(3);
        hbox.getChildren().addAll(bmoins, bplus);
        vbox.getChildren().addAll(label, hbox);

        Scene scene = new Scene(vbox);
        stage.setTitle("TestButtonIncDec4");
        stage.setScene(scene);
        stage.show();
    }

    public void handle(ActionEvent event) {
        int currentValue = Integer.parseInt(label.getText());
        if (event.getTarget() == bplus) {
            label.setText("" + (currentValue + 1));
        } else {
            label.setText("" + (currentValue - 1));
        }
    }

    public static void main(String[] args) {
        Application.launch(args);
    }
}
```



5e solution : expressions lambda

11

```
public class TestButtonIncDec5 extends Application {

    public void start(Stage stage) {
        Label label = new Label("0");
        Button bmoins = new Button(" - ");
        Button bplus = new Button(" + ");

        bplus.addEventHandler(ActionEvent.ACTION, e -> {
            int currentValue = Integer.parseInt(label.getText());
            label.setText("" + (currentValue + 1));
        });
        bmoins.addEventHandler(ActionEvent.ACTION, e -> {
            int currentValue = Integer.parseInt(label.getText());
            label.setText("" + (currentValue - 1));
        });

        VBox vbox = new VBox(3);
        vbox.setPadding(new Insets(3, 3, 3, 3));
        vbox.setAlignment(Pos.CENTER);
        HBox hbox = new HBox(3);
        hbox.getChildren().addAll(bmoins, bplus);
        vbox.getChildren().addAll(label, hbox);

        Scene scene = new Scene(vbox);
        stage.setTitle("TestButtonIncDec5");
        stage.setScene(scene);
        stage.show();
    }

    public static void main(String[] args) {
        Application.launch(args);
    }
}
```

Syntaxe des expressions lambda

12

```
e -> {  
    int currentValue = Integer.parseInt(label.getText());  
    label.setText("" + (currentValue - 1));  
}
```

```
e -> System.out.println(e.getEventType())
```

Expressions lambda

13

Raccourci de syntaxe aux classes internes anonymes

Ne peut être utilisé que pour les interfaces qui possèdent une seule méthode

```
public interface Operation {  
    public int runOperation(int a, int b);  
}
```

```
public class Addition implements Operation {  
    public int runOperation(int a, int b) {  
        return a + b;  
    }  
}
```

```
public class LambdaDemo {  
    public int doOperation(Operation op, int a, int b) {  
        return op.runOperation(a, b);  
    }  
  
    public static void main(String[] args) {  
        LambdaDemo demo = new LambdaDemo();  
  
        // Utilisation de l'interface implémentée  
        Addition myOp = new Addition();  
        int res1 = demo.doOperation(myOp, 3, 4);  
        System.out.println(res1);  
  
        // Utilisation d'une expression lambda  
        int res2 = demo.doOperation((a, b) -> a + b, 3, 4);  
        System.out.println(res2);  
    }  
}
```

Portée des variables pour les lambdas

14

Une expression lambda peut avoir accès aux variables définies dans son contexte englobant mais uniquement celles dont la valeur ne change pas

```
public class LambdaDemo2 {
    public int doOperation(Operation op, int a, int b) {
        return op.runOperation(a, b);
    }

    public static void main(String[] args) {
        LambdaDemo2 demo = new LambdaDemo2();

        int c = 1;
        int res = demo.doOperation((a, b) -> a + b + c, 3, 4);
        System.out.println(res);
    }
}
```

```
public class LambdaDemo3 {
    public int doOperation(Operation op, int a, int b) {
        return op.runOperation(a, b);
    }

    public static void main(String[] args) {
        LambdaDemo3 demo = new LambdaDemo3();

        for (int c = 1; c < 5; c++) {
            int res = demo.doOperation((a, b) -> a + b + c, 3, 4);
            System.out.println(res);
        }
    }
}
```

Local variable c defined in an enclosing scope must be final or effectively final

Abonneurs / abonnés

15

Deux catégories d'entités :

Un abonné, la source d'information, le lieu où sont gérés les événements.

Un **ou des** abonnés, utilisant l'information produite.

Résumé des différentes étapes :

Abonnement de l'abonné auprès de l'abonné.

Un événement se produit au niveau de l'abonné.

L'abonné notifie l'ensemble des abonnés en leur transmettant un événement.

Chaque abonné peut traiter l'événement.

Illustration

16

```
public class PlusieursHandlers extends Application {
    public void start(Stage stage) {
        VBox root = new VBox();
        Button b = new Button("Button");
        b.addEventHandler(ActionEvent.ACTION, e -> System.out.println("Event handler 1"));
        b.addEventHandler(ActionEvent.ACTION, e -> System.out.println("Event handler 2"));
        b.addEventHandler(ActionEvent.ACTION, e -> System.out.println("Event handler 3"));
        root.getChildren().add(b);

        Scene scene = new Scene(root);
        stage.setTitle("Plusieurs handlers");
        stage.setScene(scene);
        stage.show();
    }

    public static void main(String[] args) {
        Application.launch(args);
    }
}
```

Event handler 1
Event handler 2
Event handler 3



Utilisation de setOnXXX

17

Méthode setOnXXX pour enregistrer **un** event handler sur un noeud

```
public class TestButtonIncDec6 extends Application {  
  
    public void start(Stage stage) {  
        Label label = new Label("0");  
        Button bmoins = new Button(" - ");  
        Button bplus = new Button(" + ");  
  
        bplus.setOnAction(e -> {  
            int currentValue = Integer.parseInt(label.getText());  
            label.setText("" + (currentValue + 1));  
        });  
  
        bmoins.setOnAction(e -> {  
            int currentValue = Integer.parseInt(label.getText());  
            label.setText("" + (currentValue - 1));  
        });  
  
        VBox vbox = new VBox(3);  
        vbox.setPadding(new Insets(3, 3, 3, 3));  
        vbox.setAlignment(Pos.CENTER);  
        HBox hbox = new HBox(3);  
        hbox.getChildren().addAll(bmoins, bplus);  
        vbox.getChildren().addAll(label, hbox);  
  
        Scene scene = new Scene(vbox);  
        stage.setTitle("TestButtonIncDec5");  
        stage.setScene(scene);  
        stage.show();  
    }  
  
    public static void main(String[] args) {  
        Application.launch(args);  
    }  
}
```

Utilisation de setOnXXX

18

De la même manière:

`setOnMouseClicked`

`setOnMouseDragged`

`setOnKeyPressed`

`setOnTouchPressed`

...

Recommandations

19

Privilégier l'utilisation des méthodes setOnXXX

Peu de code : utilisation des lambda expressions

Ne pas utiliser les classes internes anonymes

Plus de code : classe interne ou implémentation EventHandler

Encore plus de code : classe externe

Toujours privilégier la lisibilité du code