**Laboratoire 2**

**Les événements et les gestionnaires/filtres d'événements**

## Objectifs d'apprentissage

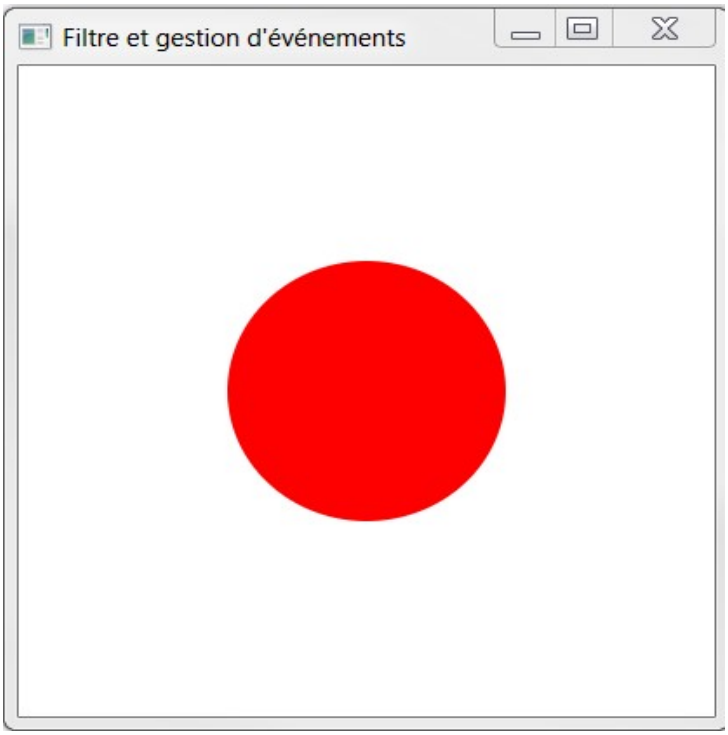
- Créer, enregistrer et retirer des filtres d'événements
- Créer, enregistrer et retirer des gestionnaires d'événements
- Comprendre le processus de traitement d'un événement
- Comprendre la différence entre un filtre et un gestionnaire d'événements
- Comprendre la notion de consommation d'un événement
- Associer des actions à des contrôles

## Évaluation du laboratoire

Ce laboratoire est **formatif**, mais il vous permettra d'obtenir des points lorsque le professeur jugera que vous avez atteint les objectifs d'apprentissage. Répondre le mieux possible aux questions posées, puis valider vos solutions avec le professeur. Votre code doit respecter les normes de programmation du cours.

## Exercice 1 : Création et enregistrement des filtres/gestionnaires d'événements, ordre d'exécution

- Créez un nouveau projet JavaFX (laboratoire 2)
- Renommez la classe *Main* en *FiltreEtGestionEvenement*
- Le panneau *root* doit être de type HBox.
- La taille de la fenêtre doit être de 400 x 400
- Le titre de la fenêtre est : *«Filtre et gestion d'événements»*
- Ajoutez un cercle (Circle) de rayon 80 et couleur rouge, au centre du panneau HBox.
- Votre application doit ressembler à ceci :




- Définissez un gestionnaire/filtre d'événements comme suit :

```
EventHandler<MouseEvent> gestionSouris = new EventHandler <MouseEvent>(){

    @Override
    public void handle(MouseEvent e) {

        System.out.println("gestionnaire ou filtre d'événements de souris a été appelé");
    }
};
```

- Enregistrez le gestionnaire d'événements *gestionSouris* pour l'événement clic de souris (MouseEvent.MOUSE_CLICKED) sur le nœud *Circle*. Utilisez la méthode *addEventHandler().*

- Enregistrez le filtre d'événement *gestionSouris,* pour l'événement clic de souris (MouseEvent.MOUSE_CLICKED), sur le nœud *Circle*. Utilisez la méthode *addEventFilter().*

- Exécutez votre application. L'affichage est :

```
gestionnaire ou filtre d'événements de souris a été appelé
gestionnaire ou filtre d'événements de souris a été appelé
```

- Notez que dans ce cas-ci, il n'y aucune distinction entre le filtre et le gestionnaire d'événements.

- Pour distinguer les filtres des gestionnaires d'événements, nous allons créer deux objets distincts comme suit :

```
EventHandler<MouseEvent> gestionEvenementSouris = new EventHandler <MouseEvent>(){

    @Override
    public void handle(MouseEvent e) {

    gererEvenement("gestionnaire d'événements de souris a été appelé", e);

    }
};

EventHandler<MouseEvent> filtreEvenementSouris = new EventHandler <MouseEvent>(){

    @Override
    public void handle(MouseEvent e) {

        gererEvenement("filtre d'événements de souris a été appelé",e);
    }


};
```

- Programmez la méthode *gererEvenement(String filtreOuGestionnaire, MouseEvent e)*
  En plus du type (filtre ou gestionnaire d'événements), cette méthode affiche aussi le type de l'événement (*e.getEventType().getName()),* la source de l'événement (*e.getSource().getClass().getSimpleName()*) et la destination de l'événement (*e.getTarget().getClass().getSimpleName()).* Voir l'exemple d'exécution, ci-dessous.

- Avant d'enregistrer les nouveaux gestionnaires/filtres d'événements, retirez *gestionSouris* du nœud. Uillisez les méthodes *removeEventFilter* et *removeEventHandler.*

- Enregistrez *filtreEvenementSouris* et *GestionEvenementSourisI* sur le nœud *Circle*

- Exécutez. Voici un exemple d'affichage après un clic sur le cercle.

```
filtre d'événements de souris a été appelé
type: MOUSE_CLICKED, Source: Circle , Destination: Circle

gestionnaire d'événements de souris a été appelé
type: MOUSE_CLICKED, Source: Circle , Destination: Circle
```

- Notez l'ordre d'exécution. Le filtre est exécuté avant le gestionnaire d'événements.

## Exercice 2 : Ordre d'exécution des gestionnaires d'événements pour un même nœud

- Ajoutez à votre projet une classe *OrdreExecutionGestionEvenement*, sous classe de *Application*
- Le panneau *root* doit être de type *HBox*.
- La taille de la fenêtre doit être de 400 x 400
- Le titre de la fenêtre est : *«Ordre d'exécution des gestionnaires d'événements pour un nœud»*
- Ajoutez un cercle (Circle) de rayon 80 et couleur rouge, au centre du panneau *HBox*.
- Utilisez une classe anonyme pour enregistrer à l'aide de la méthode *addEventHandler* un gestionnaire d'événements (clic de souris) sur le nœud *Circle*. La méthode *handle()* de cette classe affiche le message : « gestionnaire d'événements enregistré avec *addEventHandler »*
- Utilisez une classe anonyme pour enregistrer à l'aide de la méthode *setOnMouseClicked()* un gestionnaire d'événements sur le nœud *Circle*. La méthode *handle()* de cette classe affiche le message : « gestionnaire d'événements enregistré avec *setOnMouseClicked »*
- Utilisez une classe anonyme pour enregistrer à l'aide de la méthode *addEventHandler* un gestionnaire d'événements pour n'importe quel événement de souris (*MouseEvent.ANY*) sur le nœud *Circle*. La méthode *handle()* de cette classe affiche le message : « le gestionnaire *MouseEvent.ANY* a détecté l'événement : *»* suivi du type de l'événement (*e.getEventType*) seulement s'il s'agit d'un clic de souris. Ceci nous évitera des affichages inutiles.

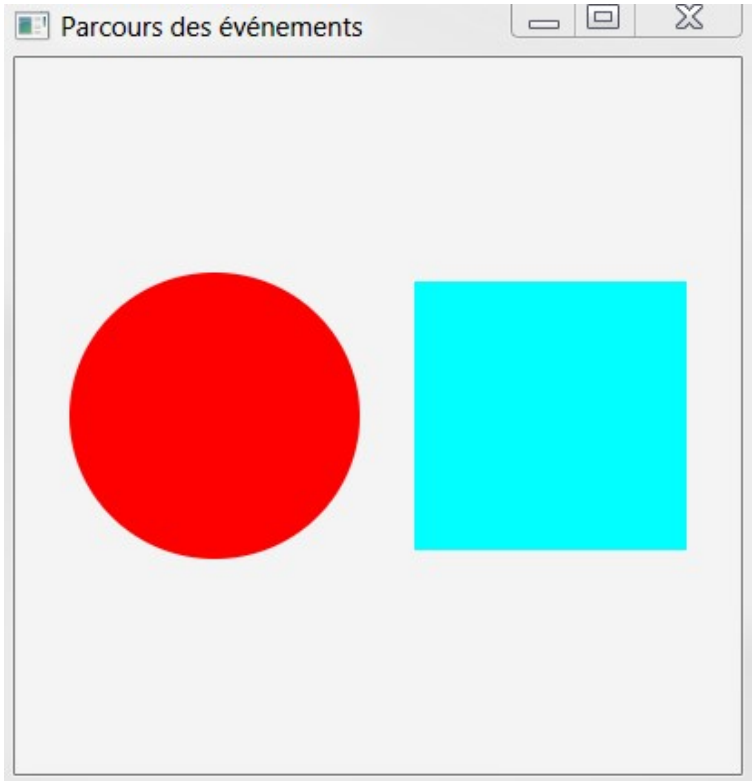
- Exécutez. L'affichage est :

```
gestionnaire d'événement enregistré avec addEventHandler
gestionnaire d'événement enregistré avec setOnMouseClicked
le gestionnaire MouseEvent.ANY a détecté l'événement: MOUSE_CLICKED
```

- Notez l'ordre d'exécution des gestionnaires d'événements.

## Exercice 3 : Parcours d'un événement, consommation d'un événement

- Ajoutez à votre projet une classe *ParcoursEvenement*, sous classe de *Application*
- Le panneau *root* doit être de type *HBox*
- La taille de la fenêtre doit être de 400 x 400
- Le titre de la fenêtre est : *«Parcours des événements»*
- Ajoutez un cercle *(Circle)* de rayon 80 et couleur rouge, au centre du panneau *HBox*.
- Ajoutez un rectangle (Rectangle) de dimension 150X150 et de couleur *CYAN*.
- Les composants sont espacés de 30 pixels.
- Exécutez. Votre affichage doit ressembler à ceci :




- Créez un gestionnaire d'événements *gestionEvenementSouris*, *filtreEvenementSouris* et la méthode *gererEvenement* de la même manière que dans l'exercice 1.
- Enregistrez *gestionEvenementSouris* et *filtreEvenementSouris* pour l'événement *MouseEvent.MOUSE_CLICKED* sur les nœuds Circle, *Stage*, *Scene* et *HBox*
- Exécutez. Cliquez sur le cercle, l'affichage est :

```
filtre d'événement de souris a été appelé
type: MOUSE_CLICKED, Source: Stage , Destination: Circle

filtre d'événement de souris a été appelé
type: MOUSE_CLICKED, Source: Scene , Destination: Circle

filtre d'événement de souris a été appelé
type: MOUSE_CLICKED, Source: HBox , Destination: Circle

filtre d'événement de souris a été appelé
type: MOUSE_CLICKED, Source: Circle , Destination: Circle

gestionnaire d'événement de souris a été appelé
type: MOUSE_CLICKED, Source: Circle , Destination: Circle

gestionnaire d'événement de souris a été appelé
type: MOUSE_CLICKED, Source: HBox , Destination: Circle

gestionnaire d'événement de souris a été appelé
type: MOUSE_CLICKED, Source: Scene , Destination: Circle

gestionnaire d'événement de souris a été appelé
type: MOUSE_CLICKED, Source: Stage , Destination: Circle
```

- Notez les différentes phases du traitement de l'événement (interception de l'événement, remontée de l'événement, exécution des filtres et des gestionnaires d'événements. Notez également les différentes sources de l'évènement.
- Exécutez. Cliquez sur le rectangle. L'affichage est :

```
filtre d'événement de souris a été appelé
type: MOUSE_CLICKED, Source: Stage , Destination: Rectangle

filtre d'événement de souris a été appelé
type: MOUSE_CLICKED, Source: Scene , Destination: Rectangle

filtre d'événement de souris a été appelé
type: MOUSE_CLICKED, Source: HBox , Destination: Rectangle

gestionnaire d'événement de souris a été appelé
type: MOUSE_CLICKED, Source: HBox , Destination: Rectangle

gestionnaire d'événement de souris a été appelé
type: MOUSE_CLICKED, Source: Scene , Destination: Rectangle

gestionnaire d'événement de souris a été appelé
type: MOUSE_CLICKED, Source: Stage , Destination: Rectangle
```

- Le rectangle n'est jamais la source de l'événement *MouseEvent.MOUSE_CLICKED.* Pourquoi?
- Pour comprendre l'effet de la consommation d'un événement par un filtre ou gestionnaire d'événements, compléter la méthode *gererEvenement*, pour consommer l'événement (e.*consume()),* si le cercle est la source d'un gestionnaire d'événements.
- Exécutez. Cliquez sur le cercle. L'affichage est :

```
filtre d'événement de souris a été appelé
type: MOUSE_CLICKED, Source: Stage , Destination: Circle

filtre d'événement de souris a été appelé
type: MOUSE_CLICKED, Source: Scene , Destination: Circle

filtre d'événement de souris a été appelé
type: MOUSE_CLICKED, Source: HBox , Destination: Circle

filtre d'événement de souris a été appelé
type: MOUSE_CLICKED, Source: Circle , Destination: Circle

gestionnaire d'événement de souris a été appelé
type: MOUSE_CLICKED, Source: Circle , Destination: Circle
```

- Notez que la remontée de l'événement n'a pas eu lieu.
- Consommez l'événement sur le filtre d'événements de la racine (HBox). Exécutez. L'affichage est :

```
filtre d'événement de souris a été appelé
type: MOUSE_CLICKED, Source: Stage , Destination: Circle

filtre d'événement de souris a été appelé
type: MOUSE_CLICKED, Source: Scene , Destination: Circle

filtre d'événement de souris a été appelé
type: MOUSE_CLICKED, Source: HBox , Destination: Circle
```

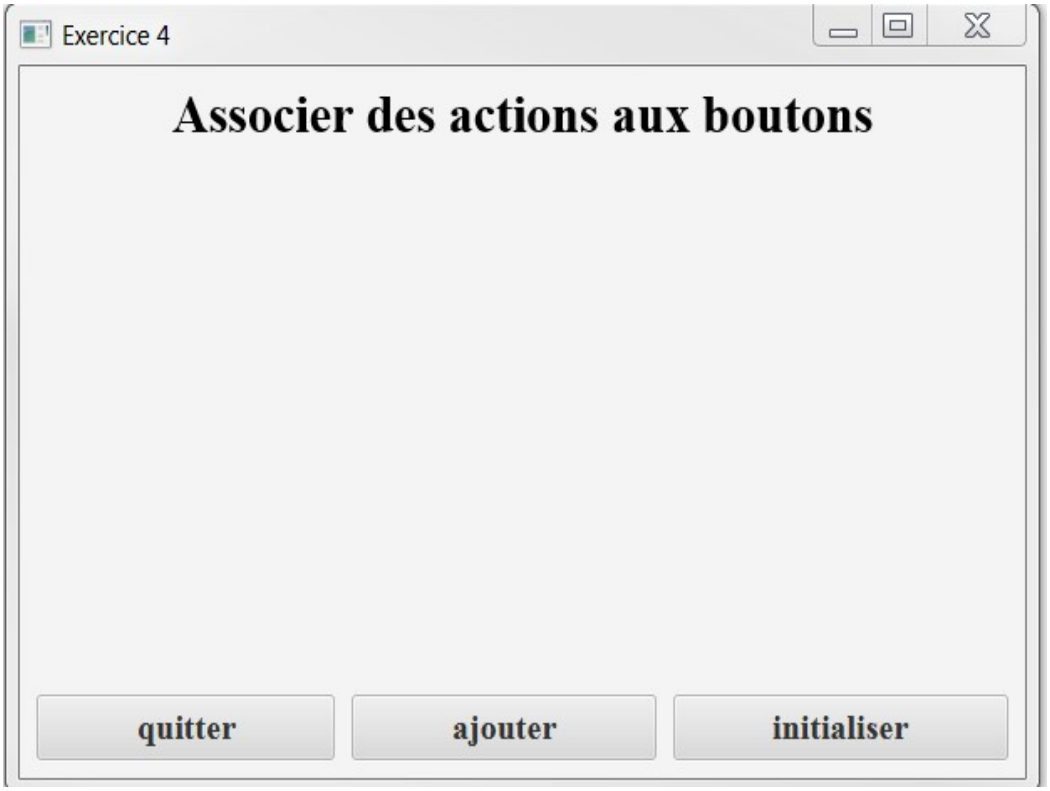
- Notez que le clic n'est pas traité sur le cercle. L'événement n'est pas arrivé à ce nœud.
- Faites d'autres tests en consommant l'événement dans d'autres filtres ou gestionnaire d'événements. Observez les résultats.
- Ajoutez une case à cocher *(CheckBox)* au panneau racine *(HBox).*
- Enregistrez le GestionEvementSouris et *filtreEvenementSouris* pour l'événement *MouseEvent.MOUSE_CLICKED*, sur le CheckBox
- Mettez en commentaire toutes les consommations d'événements.
- Exécutez. Cochez la case *CheckBox*. L'affichage est :

```
filtre d'événement de souris a été appelé
type: MOUSE_CLICKED, Source: Stage , Destination: StackPane

filtre d'événement de souris a été appelé
type: MOUSE_CLICKED, Source: Scene , Destination: StackPane

filtre d'événement de souris a été appelé
type: MOUSE_CLICKED, Source: HBox , Destination: StackPane

filtre d'événement de souris a été appelé
type: MOUSE_CLICKED, Source: CheckBox , Destination: StackPane

gestionnaire d'événement de souris a été appelé
type: MOUSE_CLICKED, Source: CheckBox , Destination: StackPane
```

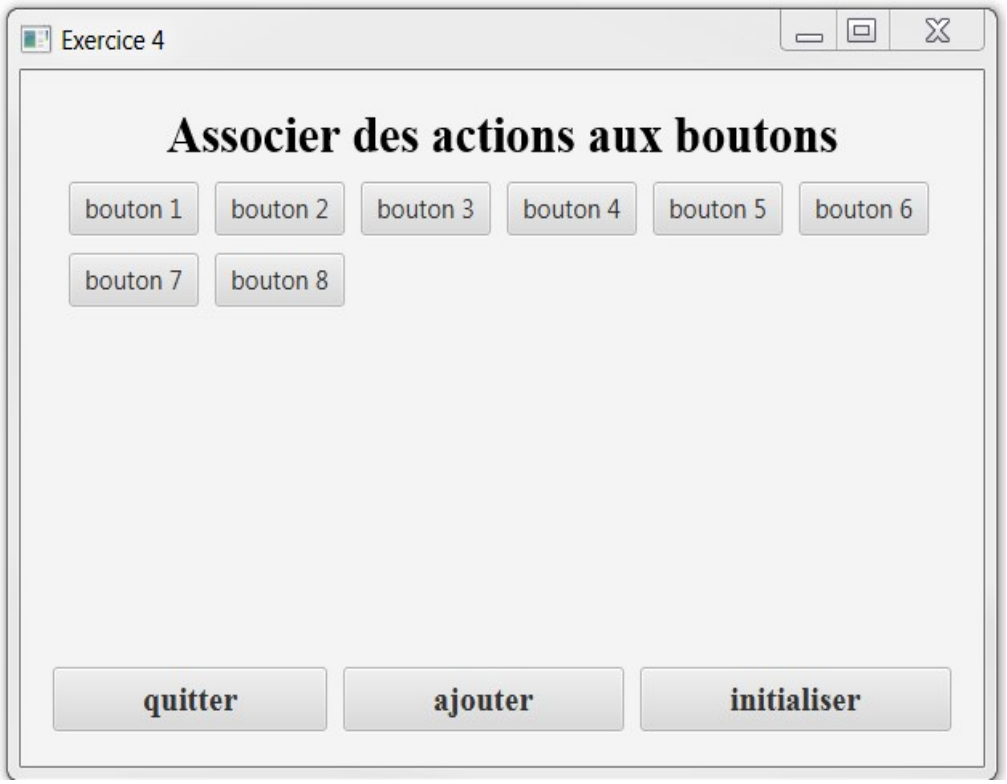
- Observez ce résultat. Noter que l'événement ne remonte pas vers la racine alors que nous n'avons pas consommé l'événement au niveau du destinataire *(CheckBox*).
  Le *CheckBox* a un gestionnaire d'événements par défaut qui consomme l'événement.

## Exercice 4 : Associer des actions aux boutons

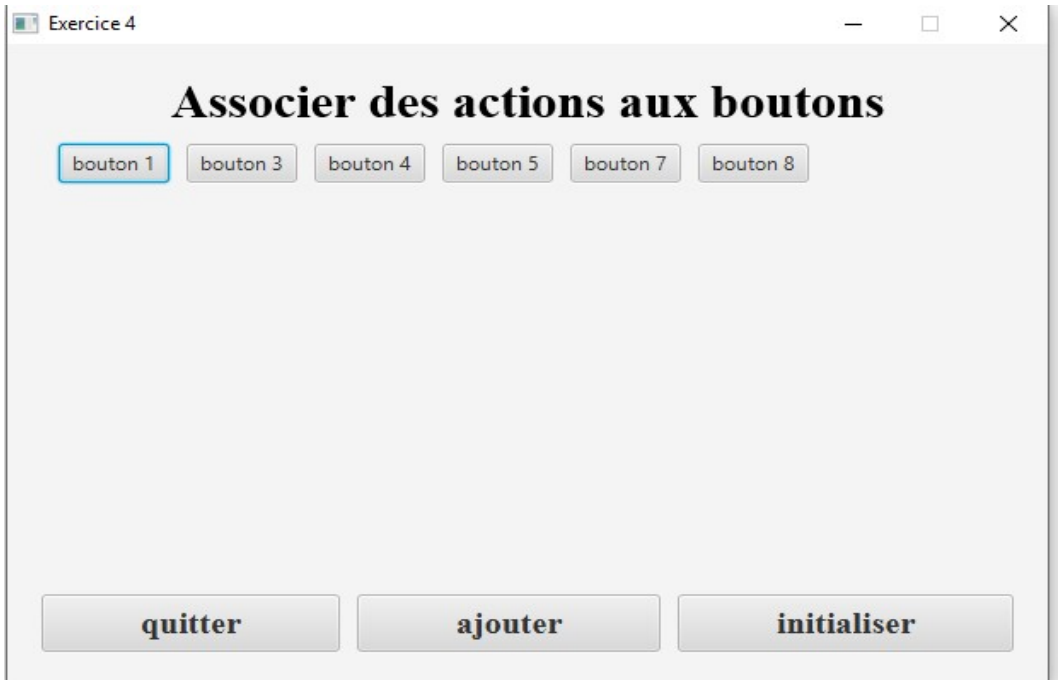
- Ajoutez à votre projet, une classe Exercice4, sous-classe de *Application*
- Reproduisez l'interface graphique suivante :




- Le titre de la fenêtre est : *Exercice 4*
- La taille de la fenêtre est 600x400 pixels
- Interdisez le redimensionnement de la fenêtre
- Le titre du formulaire est : associer des actions aux boutons le texte est en gras, de taille 30, de type *Serif* et est centré.
- En bas du formulaire, on trouve trois boutons de commandes : quitter, ajouter et initialiser. Le texte des boutons a la même police que le titre, mais avec une taille de 20.
- Le bouton *quitter* termine l'application
- Le bouton *ajouter* : à chaque clic sur ce bouton, un nouveau bouton est ajouté au centre du formulaire. Les boutons ajoutés sont numérotés à leur création. N'oubliez pas d'enregistrer le gestionnaire d'événements sur chaque nouveau bouton. Quand on clique sur l'un d'entre eux, il disparaît.
- Le bouton *initialiser*, supprime tous les boutons du panneau du centre.
- Utiliser une classe privée ***GestionBouton*** qui implémente ***EventHandler<ActionEvent >*** pour gérer les clics des boutons.
- Exemple après 8 clics sur le bouton *ajouter*




- **Attention** : bien respecter le format d'affichage. Notez l'espace entre les boutons et le titre ainsi que les espaces (horizontal et vertical) entre les boutons.
- Exemple après un clic sur bouton2 et bouton6



## Exercice 5 : Associer des actions à des boutons radio et des cases à cocher

- **Programmez** les gestionnaires d'événements de l'exercice 6 du laboratoire 1. Pour ce faire vous devez programmer trois classes privées : ***GestionBouton***, ***GestionFont*** et ***GestionCouleur*.** Ces trois classes implémentent ***EventHandler<ActionEvent>***

- **GestionBouton :** change la casse du texte **«** *interface graphique java »*, selon le bouton sur lequel on clique (*majuscule* ou *minuscule*).

- **GestionCouleur** : change la couleur du texte **«** *interface graphique java»*, selon le bouton radio choisi (*rouge*, *vert* ou *bleu*).

- **GestionFont** : met le texte **«** *interface graphique java »*, en gras ou en italique, selon la case qui est cochée (*gras* ou *italique*). Pour programmer ce gestionnaire, vous devez bien comprendre :

    - les énumérations (Enum), ***FontWeight,*** pour le gras et ***FontPosture***, pour l'italique.
    - La méthode statique ***Font.font(…)*** pour la mise à jour de la police (nom, taille, style …).

      **Remarque** : vous devez convertir votre source (*e.getSource())* en un objet *CheckBox* pour pouvoir utiliser la méthode *isSelected(),* qui permet de savoir si on a coché ou décoché la case.

- Associez les composants à leurs gestionnaires respectifs. Utilisez les méthodes ***setOnXXX*** des composants. Dans notre cas, il s'agit de ***setOnAction().***

  Exemple d'exécution :