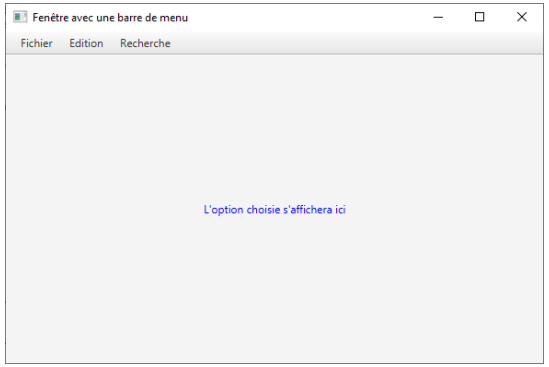
Annexes
Menu – MVC - Plusieurs scènes ou vues

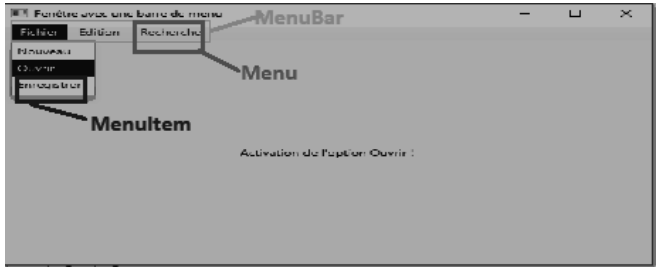
1) Menu

L'objectif est d'afficher une barre de menu en haut d'une fenêtre comme indiqué ci-dessous. Le menu « Fichier » contient lui-même 3 options de menu : « Nouveau », « Ouvrir » et « Enregistrer ». Lorsque l'utilisateur choisira l'une de ces 3 options, le nom de celle-ci devra apparaître dans le texte situé au centre de la fenêtre.



Il faut utiliser les composants JavaFx suivants :

- ***MenuBar*** pour afficher la barre de menu
- ***Menu*** pour afficher chacun des menus « Fichier », « Edition » et « Recherche »
- ***MenuItem*** pour afficher chacune des 3 options du menu « Fichier » : « Nouveau », « Ouvrir » et « Enregistrer »



Le code Java est le suivant :

```java
/*  Exemple d'affichage d'une barre de menu
 *  Projet ExempleMenu                      04/23
 */
package application;

import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.Menu;
import javafx.scene.control.MenuBar;
import javafx.scene.control.MenuItem;
import javafx.scene.layout.BorderPane;
import javafx.scene.paint.Color;

/**
 * Affiche une fenêtre dotée d'une barre de menu.
 * Les 3 menus de la barre de menu sont : Fichier, Edition et Rercherche
 * L'option Fichier permet d'afficher un sous-menu sous la forme d'une liste déroulante
 * avec les 3 options : Ouvrir, Editer et Enregistrer.
 *
 * Lorsque que l'utilisateur choisit l'une de ces 3 options, un message d'information
 * s'affiche au centre de la fenêtre.
 *
 * Dans cette version les options des menus Edition et Recherche ne sont pas codées
 *
 * @author C. Servières
 * @version 1.0
 *
 */
public class Main extends Application {


    @Override
    public void start(Stage primaryStage) {

        /* Le conteneur BorderPane est bien adapté pour contenir une barre de menu
         * Dans la zone haute, on insère la barre de menu et dans la zone centrale
         * on peut insérer un autre conteneur qui correspondra à la partie principale
         * de la fenêtre (celle située en dessous de la barre de menu.
         * Dans cet exemple, on a directement placé un label dans la partie centrale
         * du BorderPane
         * Remarque : on peut utiliser d'autres conteneurs à la place du BorderPane
         *            un VBox par exemple
         */
        BorderPane racine = new BorderPane();

        /*  Création de la barre de menu, elle est vide à sa création */
        MenuBar barreDeMenu = new MenuBar();

        /* Création des 3 options de la barre de menu */
        Menu menuFichier = new Menu("Fichier");
        Menu menuEdition = new Menu("Edition");
        Menu menuRecherche = new Menu("Recherche");

        // ajout des 3 options à la barre de menu
        barreDeMenu.getMenus().setAll(menuFichier, menuEdition, menuRecherche);

        // création des 3 options du menu "Fichier"
        MenuItem menuItemNouveau = new MenuItem("Nouveau");
        MenuItem menuItemOuvrir = new MenuItem("Ouvrir");
        MenuItem menuItemEnregistrer = new MenuItem("Enregistrer");
```

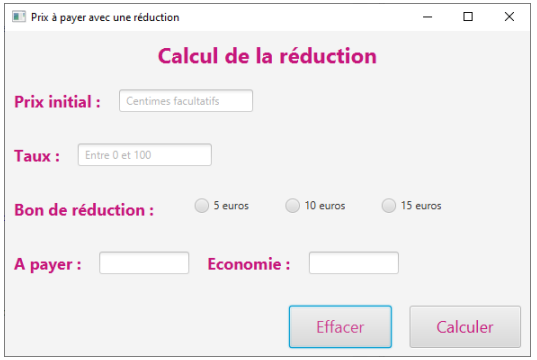
```
        // ajout des 3 options au menu fichier
        menuFichier.getItems().setAll(menuItemNouveau, menuItemOuvrir, menuItemEnregistrer);

        // création du label affichant l'action de l'utilisateur
        Label resultat = new Label("L'option choisie s'affichera ici");
        resultat.setTextFill(Color.BLUE);

        // on associe un écouteur aux options du menu Fichier
        menuItemNouveau.setOnAction(
                actionEvent ->  resultat.setText("Activation de l'option Nouveau !"));
        menuItemOuvrir.setOnAction(
                actionEvent ->  resultat.setText("Activation de l'option Ouvrir !"));
        menuItemEnregistrer.setOnAction(
                actionEvent ->  resultat.setText("Activation de l'option Enregistrer !"));

        // on insère la barre de menu et le label dans le BorderPane
        racine.setTop(barreDeMenu);
        racine.setCenter(resultat);

        // on crée la scène, on fait le lien entre le primaryStage, la scène et le BorderPane
        Scene scene = new Scene(racine);
        primaryStage.setTitle("Fenêtre avec une barre de menu");
        primaryStage.setWidth(600);
        primaryStage.setHeight(400);
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    /**
     * Progamme principal, lance l'affichage de la fenêtre
     * @param args  argument non utilisé
     */
    public static void main(String[] args) {
        Launch(args);
    }
}
```

2) Modèle – Vue - Controleur

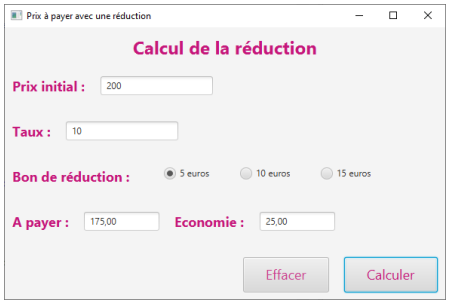
Le modèle de conception MVC est mis en œuvre dans l'exemple de l'application suivante. L'utilisateur est invité à renseigner :

- un prix initial qui doit être positif ou nul
- un taux de réduction compris entre 0 et 100. Eventuellement, cette zone de saisie peut ne pas être renseignée. Dans ce cas, le taux de réduction sera considéré comme égal à 0.
- Un bon de réduction éventuel pour lequel 3 montants sont possibles : 5, 10 ou 15 euros.

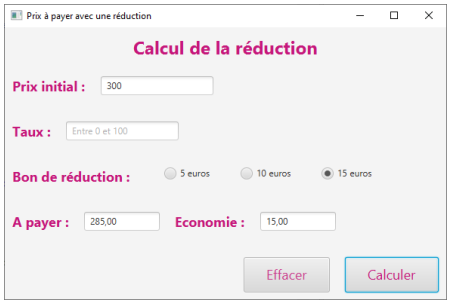
A partir de ces informations, l'application affiche le prix à payer et le montant de l'économie. Si une erreur de saisie est commise, une boîte d'alerte apparaît.



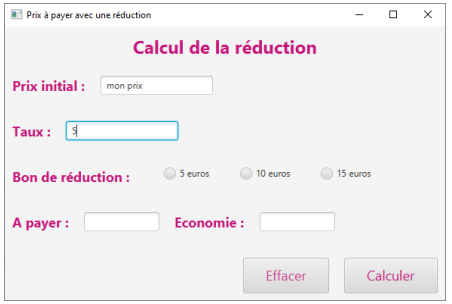
Exemples d'utilisation :



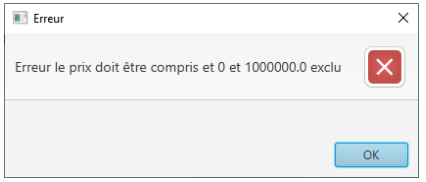
L'utilisateur a renseigné toutes les informations



L'utilisateur n'a pas renseigné le taux



L'utilisateur a commis une erreur pour le prix



Une boîte d'erreur apparaît

Fichiers nécessaires :

Le code de l'application est réparti sur 4 fichiers :

- un fichier fxml permettant d'afficher la vue. Ce fichier est produit avec *SceneBuilder*
- une classe ***Main*** qui hérite de la classe ***Application*** et qui contient une méthode ***start*** permettant de charger la vue et de placer la Scène sur la fenêtre principale
- une classe nommée ici ***CalculReduction*** qui joue le rôle du modèle
- la classe contrôleur associée à la vue et qui s'appuie sur le modèle pour effectuer les calculs

Classe *CalculReduction* – Le modèle

On a codé une classe qui joue le rôle du modèle. C'est elle qui gère les calculs pour obtenir l'économie réalisée et le prix à payer. Elle assure également la vérification des valeurs qui lui sont transmises (prix initial, taux et bon de réduction). Si l'une des valeurs n'est pas valide, une exception *IllegalArgumentException* est propagée.

```java
/*
 * Classe qui joue le rôle du modèle
 * Gère le calcul d'un prix à payer                                  05/23
 */
package application;


/**
 * Gère le calcul d'un prix à payer selon un taux de réduction et selon
 * la présence éventuelle d'un bon d'achat dont le montant peut être égal
 * à 5, 10 ou 15 euros
 * Cette classe joue le rôle du modèle dans l'application JavaFX qui permet
 * de calculer le prix à payer (avec réduction et bon d'achat)
 * @author C. Servières
 * @version 1.0
 *
 */
public class CalculReduction {


    /** prix maximum géré, doit être inférieur à la constante */
    private static final double MAX_PRIX = 1000000.0;

    /** Message d'erreur si l'utilisateur commet une erreur sur la saisie du prix */
    private static final String ERREUR_PRIX =
            "Erreur le prix doit être compris et 0 et " + MAX_PRIX + " exclu";

    /** Message d'erreur si l'utilisateur commet une erreur sur la saisie du taux */
    private static final String ERREUR_TAUX =
            "Erreur le taux doit être compris et 0 et 100";

    /** Message d'erreur si l'utilisateur commet une erreur sur la saisie
     * du montant du bon d'achat.
     * Remarque : avec la version actuelle de l'interface, cette erreur ne peut pas
     * se produire
     */
    private static final String ERREUR_BON =
            "Erreur le montant du bon de réduction doit être positif ou nul";



    /** Prix à payer avant la réduction */
    private double prixInitial;

    /** Taux de la réduction accordée, peut être égal à 0
     *  Remarque : il s'agit du taux divisé par 100
     */
    private double tauxReduction;

    /** Montant du bon d'achat à déduire, peut être égal à 0 */
    private double bonReduction;



    /* ======================   CONSTRUCTEURS   ========================== */

    /**
     * Constructeur qui initialise par défaut
     */
    public CalculReduction() {
        this.prixInitial = 0.0;
```



```java
        this.tauxReduction = 0.0;
        this.bonReduction = 0.0;
    }

    /**
     * Constructeur avec en argument les valeurs initiales
     * @param prixInitial       prix avant réduction (entre 0 et MAX_PRIX exclu)
     * @param tauxReduction     taux de la réduction (compris entre 0 et 100)
     * @param bonReduction      bon de réduction (supérieur ou égal à 0)
     * @throws IllegalArgumentException  levée si l'un des arguments est invalide
     */
    public CalculReduction(double prixInitial, double tauxReduction,
                           double bonReduction) {
       verifierEtAffecter(prixInitial, tauxReduction, bonReduction);
    }


    /**
     * Constructeur avec en argument les valeurs initiales en tant que chaîne de
     * caractères
     * @param prixInitial       prix avant réduction (entre 0 et MAX_PRIX exclu)
     * @param tauxReduction     taux de la réduction, si la chaîne est vide on
     *                          considère que le taux est 0  (compris entre 0 et 100)
     * @param bonReduction      bon de réduction (supérieur ou égal à 0)
     * @throws IllegalArgumentException  levée si l'un des arguments est invalide
     */
    public CalculReduction(String prixEnChaine, String tauxEnChaine, String bonEnChaine) {
        double prix = 0.0;
        double taux = 0.0;
        double bon = 0.0;

        try {  // conversion du prix
            prix = Double.parseDouble(prixEnChaine);
        } catch(NumberFormatException erreur) {
            throw new IllegalArgumentException(ERREUR_PRIX);
        }

        try {  // conversion du taux
            taux = Double.parseDouble(tauxEnChaine);
        } catch(NumberFormatException erreur) {
            throw new IllegalArgumentException(ERREUR_TAUX);
        }

        try {  // conversion du bon de réduction
            taux = Double.parseDouble(bonEnChaine);
        } catch(NumberFormatException erreur) {
            throw new IllegalArgumentException(ERREUR_BON);
        }
        verifierEtAffecter(prix, taux, bon);
     }



    /* ======================   ACCESSEURS   ========================== */

    /**
     * Accesseur sur le prix à payer
     * Si celui-ci est négatif, il est ramené à 0
     * @return  un double égal au prix à payer une fois les réductions déduites
     */
    public double getPrixAPayer() {
        double aPayer = prixInitial - getEconomie();
        return aPayer >= 0.0 ? aPayer : 0.0;
    }

    /**
     * Accesseur sur l'économie réalisée (application du taux de réduction
     * et déduction du bon de réduction)
     * @return un double égal à l'économie réalisée
     */
    public double getEconomie() {
        return (prixInitial * tauxReduction) + bonReduction;
```

```
    }

    /**
     * Accesseur sur le prix à payer
     * Si celui-ci est négatif, il est ramené à 0
     * @return  une chaîne contenant le prix à payer
     *          une fois les réductions déduites
     */
    public String getPrixAPayerEnChaine() {
        return String.format("%.2f", getPrixAPayer());

    }

    /**
     * Accesseur sur l'économie réalisée (application du taux de réduction
     * et déduction du bon de réduction)
     * @return une chaîne contenant l'économie réalisée
     */
    public String getEconomieEnChaine() {
        return String.format("%.2f", getEconomie());
    }



    /* ====================   MODIFICATEURS   ======================== */

    /**
     * Setteur sur le prix initial
     * @param nouveauPrix  double égal au nouveau prix
     * @throws IllegalArgumentException levée si le prix n'est pas valide
     */
    public void setPrixInitial(double nouveauPrix) {
        if (! prixValide(nouveauPrix)) {
            throw new IllegalArgumentException(ERREUR_PRIX);
        }
        prixInitial = nouveauPrix;
    }


    /**
     * Setteur sur le taux de réduction
     * @param nouveauTaux  double égal au taux de réduction
     * @throws IllegalArgumentException levée si le taux n'est pas valide
     */
    public void setTauxReduction(double nouveauTaux) {
        if (! tauxValide(nouveauTaux)) {
            throw new IllegalArgumentException(ERREUR_TAUX);
        }
        tauxReduction = nouveauTaux / 100.0;
    }

    /**
     * Setteur sur le bon de réduction
     * @param nouveauBon  double égal au bon de réduction
     * @throws IllegalArgumentException levée si le bon n'est pas valide
     */
    public void setBonReduction(double nouveauBon) {
        if (! bonReductionValide(nouveauBon)) {
            throw new IllegalArgumentException(ERREUR_BON);
        }
        bonReduction = nouveauBon;
    }


    /**
     * Setteur sur le prix initial
     * @param nouveauPrix  chaîne contenant le nouveau prix
     *                     Ne doit pas être vide
     * @throws IllegalArgumentException levée si le prix n'est pas valide
     */
    public void setPrixInitial(String prixEnChaine) {
        double prix = 0.0;
```



```
        try {
            prix = Double.parseDouble(prixEnChaine);
        } catch(NumberFormatException erreur) {
            throw new IllegalArgumentException(ERREUR_PRIX);
        }
        setPrixInitial(prix);
    }


    /**
     * Setteur sur le taux de réduction
     * @param nouveauTaux  chaîne contenant le taux de réduction
     *                     peut être vide, dans ce cas le taux est 0
     * @throws IllegalArgumentException levée si le taux n'est pas valide
     */
    public void setTauxReduction(String tauxEnChaine) {
        double taux = 0.0;

        // si la chaîne est vide, on considère que le taux est 0
        if (tauxEnChaine.trim().length() > 0) {
            try {
                taux = Double.parseDouble(tauxEnChaine);
            } catch(NumberFormatException erreur) {
                throw new IllegalArgumentException(ERREUR_TAUX);
            }
        }
        setTauxReduction(taux);
    }

    /**
     * Remise à 0 des attributs
     */
    public void remiseAZero() {
        this.prixInitial = 0.0;
        this.tauxReduction = 0.0;
        this.bonReduction = 0.0;
    }



    /* ================  VERIFICATION DE VALIDITE ==================== */

    /**
     * Vérifie si le prix initial est valide
     * (compris entre 0 et MAX_PRIX exclu)
     * @param aTester double égal au prix initial
     * @return un booléen égal à vrai ssi aTester est un prix valide
     */
    private static boolean prixValide(double aTester) {
        return aTester >= 0.0 && aTester < MAX_PRIX;
    }


    /**
     * Vérifie si le taux de réduction est valide
     * (compris entre 0 et 100)
     * @param aTester double égal au taux initial
     * @return un booléen égal à vrai ssi aTester est un taux valide
     */
    private static boolean tauxValide(double aTester) {
        return aTester >= 0.0 && aTester <= 100.0;
    }


    /**
     * Vérifie si le bon de réduction est valide
     * (>= 0)
     * @param aTester double égal au bon de réduction
     * @return un booléen égal à vrai ssi aTester est un bon valide
     */
    private static boolean bonReductionValide(double aTester) {
        return aTester >= 0.0;
```

```
    }


    /**
     * Vérifie si le prixInitial, le taux de réduction et le bon de réduction sont
     * valides.
     * @param prixInitial       double égal au prix initial à vérifier
     * @param tauxReduction     double égal au taux de réduction à vérifier
     * @param bonReduction      double égal au bon de réduction à vérifier
     * @throws IllegalArgumentException levée si l'un des arguments est invalide
     */
    private void verifierEtAffecter(double prixInitial, double tauxReduction, double bonReduction) {
       if (! prixValide(prixInitial)) {
           throw new IllegalArgumentException(ERREUR_PRIX);
       }
       if (! tauxValide(tauxReduction)) {
           throw new IllegalArgumentException(ERREUR_TAUX);
       }
       if (! bonReductionValide(bonReduction)) {
           throw new IllegalArgumentException(ERREUR_TAUX);
       }

       // tous les arguments sont valides : on les affecte aux attributs
       this.prixInitial = prixInitial;
       this.tauxReduction = tauxReduction / 100.0;
       this.bonReduction = bonReduction;
    }
}
```

Classe *ControleurCalculReduction* – Le contrôleur

Cette classe est codée de la manière habituelle pour gérer les interactions avec l'utilisateur :

- 2 méthodes gèrent les clics sur les 2 boutons
- 3 méthodes sont appelées automatiquement chaque fois que l'utilisateur sélectionne l'un des boutons radio pour choisir le montant d'un bon de réduction

De plus, on remarque dans cette classe la présence d'un attribut de type ***CalculReduction*** qui le modèle associé au contrôleur. Il faut noter les points essentiels suivants :

- l'instance ***CalculReduction*** est créée au chargement de la vue, donc lors de l'appel automatique à la méthode ***initialize***
- chaque fois que l'utilisateur sélectionne un bouton radio, le modèle est mis à jour avec le nouveau montant du bon de réduction, via les méthodes ***gererBtn…euros***
- par contre, chaque fois que l'utilisateur modifie un *TextField*, que ce soit celui du prix initial ou celui du taux de réduction, le modèle n'est pas mis à jour. En effet, il est plus simple de récupérer les saisies lors du clic sur le bouton ***Calculer***
- lorsque l'utilisateur clique sur le bouton ***Calculer***, le modèle est mis à jour avec le nouveau prix et le nouveau taux, donc dans la méthode ***gererClicCalculer***
- si l'utilisateur clique sur ***Effacer***, la méthode ***gererClicEffacer*** est appelée et elle met à jour le modèle pour le réinitialiser (voir l'appel à ***remiseAZero*** sur l'instance de type ***CalculReduction***)

```
/*  Classe contrôleur de l'application calcul de la réduction     05/23 */
package application;

import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.control.Alert;
import javafx.scene.control.RadioButton;
import javafx.scene.control.TextField;
import javafx.scene.control.ToggleGroup;
```



```
/**
 * Classe qui joue le rôle du contrôleur dans l'application calcul du prix à payer en
 * fonction d'un taux de réduction et d'un bon de réduction
 * REMARQUE : on trouve dans cette classe un attribut de type CalculReduction. Cette
 * instance est le modèle géré par le contrôleur. C'est grâce à elle que les vérifications
 * des saisies sont faites, ainsi que les calculs des résultats (prix à payer et économie)
 * @author C. Servières
 * @version 1.0
 */
public class ControleurCalculReduction {

    /** Titre de la boite affichée si erreur de saisie */
    private static final String TITRE_BOITE_ERREUR = "Erreur";

    /**
     * Modèle géré par le contrôleur
     * Effectue les vérifications et les calculs
     */
    private CalculReduction calcul;

    @FXML
    private TextField fieldPrix;

    @FXML
    private TextField fieldTaux;

    @FXML
    private TextField fieldAPayer;

    @FXML
    private TextField fieldEconomie;

    @FXML
    private RadioButton btnRadio5euros;

    @FXML
    private RadioButton btnRadio10euros;

    @FXML
    private RadioButton btnRadio15euros;


    /**
     * Méthode appelée automatiquement lorsque la vue est chargée
     * On crée le modèle calcul et on finit de préparer la vue
     */
    @FXML
    private void initialize() {
        // préparation du groupe de boutons radio
        ToggleGroup groupe = new ToggleGroup();
        groupe.getToggles().addAll(btnRadio5euros, btnRadio10euros, btnRadio15euros);

        /*
         * Création de l'instance modèle qui sera gérée par ce contrôleur
         * Ici il s'agit d'un objet de type CalculReduction
         */
        calcul = new CalculReduction();

    }

    /*
     * Méthode appelée automatiquement si l'utiliseur clique sur le bouton Effacer
     */
    @FXML
    void gererClicEffacer(ActionEvent event) {
        // les contrôles sont remis dans leur état initial
        calcul.remiseAZero();
        fieldPrix.setText("");
        fieldTaux.setText("");
        fieldAPayer.setText("");
        fieldEconomie.setText("");
        btnRadio5euros.setSelected(false);
```

```
        btnRadio10euros.setSelected(false);
        btnRadio15euros.setSelected(false);
    }


    /*
     * Méthode appelée automatiquement si l'utiliseur clique sur le bouton Calculer
     */
    @FXML
    void gererClicCalculer(ActionEvent event) {
        try {
            // on renseigne le modèle avec les valeurs saisies (prix et taux)
            calcul.setPrixInitial(fieldPrix.getText());
            calcul.setTauxReduction(fieldTaux.getText());

            // on affiche les résultats
            fieldAPayer.setText(calcul.getPrixAPayerEnChaine());
            fieldEconomie.setText(calcul.getEconomieEnChaine());
        } catch(IllegalArgumentException erreur) {

            /*
             * Une erreur de saisie a été commise : une boîte
             * d'alerte est affichée avec le message propagée avec l'exception
             */
            Alert boiteAlerte = new Alert(Alert.AlertType.ERROR);
            boiteAlerte.setTitle(TITRE_BOITE_ERREUR);
            boiteAlerte.setHeaderText(erreur.getMessage());
            boiteAlerte.showAndWait();
        }
    }

    /*
     * Méthode appelée automatiquement si l'utilisateur sélectionne le bouton
     * "5 euros"
     */
    @FXML
    void gererBtn5euros(ActionEvent event) {
         calcul.setBonReduction(5);
    }

    /*
     * Méthode appelée automatiquement si l'utilisateur sélectionne le bouton
     * "10 euros"
     */
    @FXML
    void gererBtn10euros(ActionEvent event) {
        calcul.setBonReduction(10);
    }

    /*
     * Méthode appelée automatiquement si l'utilisateur sélectionne le bouton
     * "15 euros"
     */
    @FXML
    void gererBtn15euros(ActionEvent event) {
        calcul.setBonReduction(15);
    }

}
```

La vue

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.geometry.*?>
<?import javafx.scene.text.*?>
<?import javafx.scene.control.*?>
<?import java.lang.*?>
<?import javafx.scene.layout.*?>
<?import javafx.scene.layout.AnchorPane?>
```



```
<VBox alignment="TOP_CENTER" maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity"
      prefHeight="400.0" prefWidth="600.0"
      spacing="20.0" xmlns="http://javafx.com/javafx/8" xmlns:fx="http://javafx.com/fxml/1"
      fx:controller="application.ControleurCalculReduction">
   <children>
      <Label text="Calcul de la réduction" textAlignment="CENTER" textFill="#c6187b">
         <font>
            <Font name="System Bold" size="24.0" />
         </font>
         <opaqueInsets>
            <Insets />
         </opaqueInsets>
      </Label>
      <HBox prefHeight="50.0" prefWidth="200.0" spacing="20.0">
         <children>
            <Label text="Prix initial :" textFill="#c6187b">
               <font>
                  <Font name="System Bold" size="18.0" />
               </font>
            </Label>
            <TextField fx:id="fieldPrix" focusTraversable="false" promptText="Centimes facultatifs">
               <opaqueInsets>
                  <Insets />
               </opaqueInsets>
               <HBox.margin>
                  <Insets />
               </HBox.margin>
            </TextField>
         </children>
      </HBox>
      <HBox prefHeight="50.0" prefWidth="200.0" spacing="20.0">
         <children>
            <Label text="Taux :" textFill="#c6187b">
               <font>
                  <Font name="System Bold" size="18.0" />
               </font>
            </Label>
            <TextField fx:id="fieldTaux" promptText="Entre 0 et 100" />
         </children>
      </HBox>
      <HBox prefHeight="50.0" prefWidth="200.0" spacing="40.0">
         <children>
            <Label text="Bon de réduction : " textFill="#c6187b">
               <font>
                  <Font name="System Bold" size="18.0" />
               </font>
            </Label>
            <RadioButton fx:id="btnRadio5euros" mnemonicParsing="false" onAction="#gererBtn5euros"
                         text="5 euros" />
            <RadioButton fx:id="btnRadio10euros" mnemonicParsing="false" onAction="#gererBtn10euros"
                         text="10 euros" />
            <RadioButton fx:id="btnRadio15euros" mnemonicParsing="false" onAction="#gererBtn15euros"
                         text="15 euros" />
         </children>
      </HBox>
      <HBox prefHeight="50.0" prefWidth="200.0" spacing="20.0">
         <children>
            <Label text="A payer :" textFill="#c6187b">
               <font>
                  <Font name="System Bold" size="18.0" />
               </font>
            </Label>
            <TextField fx:id="fieldAPayer" editable="false" maxWidth="100.0" />
            <Label text="Economie :" textFill="#c6187b">
               <font>
                  <Font name="System Bold" size="18.0" />
               </font>
            </Label>
            <TextField fx:id="fieldEconomie" editable="false" maxWidth="100.0" />
         </children>
      </HBox>
      <HBox alignment="BOTTOM_RIGHT" prefHeight="50.0" prefWidth="200.0" spacing="20.0">
         <children>
            <Button mnemonicParsing="false" onAction="#gererClicEffacer" text="Effacer" textFill="#c6187bde">
               <font>
                  <Font size="18.0" />
               </font>
               <padding>
                  <Insets bottom="10.0" left="30.0" right="30.0" top="10.0" />
               </padding>
```

```
            </Button>
            <Button mnemonicParsing="false" onAction="#gererClicCalculer" text="Calculer" textFill="#c6187b">
               <font>
                  <Font size="18.0" />
               </font>
               <padding>
                  <Insets bottom="10.0" left="30.0" right="30.0" top="10.0" />
               </padding>
            </Button>
         </children>
      </HBox>
   </children>
   <padding>
      <Insets bottom="10.0" left="10.0" right="10.0" top="10.0" />
   </padding>
</VBox>
```

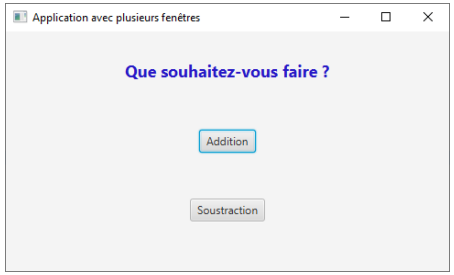
La classe *Main* qui hérite de *Application*

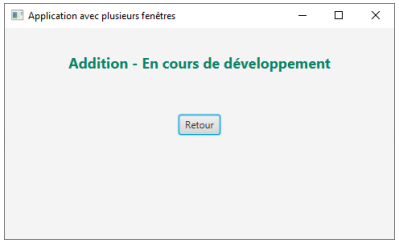
```
/* Programme principal de l'application calcul de la réduction  05/23 */
package application;

import java.io.IOException;
import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.stage.Stage;
import javafx.scene.Parent;
import javafx.scene.Scene;

/**
 * Classe principale de l'application calcul de la réduction
 * La fenêtre principale est affichée via une vue décrite dans un fichier fxml
 * @author C. Servières
 * @version 1.0
 */
public class Main extends Application {

	@Override
	public void start(Stage primaryStage) {

	    /* création d'un chargeur de code FXML
	     * et chargement de la vue de l'application
	     */
	    FXMLLoader chargeurFXML = new FXMLLoader();
	    chargeurFXML.setLocation(getClass().getResource("VueCalculReduction.fxml"));
	    Parent racine;
	    try {
	       racine = chargeurFXML.load();
	       Scene scene = new Scene(racine);
	       scene.getRoot().requestFocus();

	       // on définit les caractéristiques de la fenêtre et lui associe la scène
	       primaryStage.setTitle("Prix à payer avec une réduction");
	       primaryStage.setHeight(400);
	       primaryStage.setWidth(600);
	       primaryStage.setScene(scene);
	       primaryStage.show();
	    } catch (IOException e) {
	       e.printStackTrace();
	    }
	}

	/**
	 * Programme principal
	 * @param args  non utilisé
	 */
	public static void main(String[] args) {
		launch(args);     // appelera la méthode start
	}
}
```

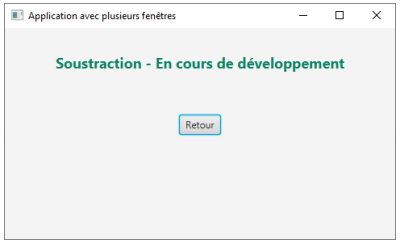

3) Plusieurs scènes différentes dans une application

L'objectif est de coder une application comportant plusieurs fonctionnalités, chacune ayant une vue différente. L'exemple d'illustration est simplifié dans le sens où les fonctionnalités ne sont pas entièrement codées.

A son lancement, l'application affichera une scène contenant 2 boutons « Addition » et « Soustraction ». Selon le clic de l'utilisateur, une scène différente sera affichée, celle de l'addition ou celle de la soustraction. Sur chacune de ces 2 vues, un bouton « retour » permet de revenir à la scène principale.







C'est dans la méthode ***start*** de la classe principale que le développeur doit coder les principaux traitements. Il s'agit de créer 3 scènes différentes, une pour chacune des vues : ***scenePrincipale***, ***sceneAddition***, ***sceneSoustraction***.

Par exemple, le code ci-dessous permet de créer la scène de l'addition :

```
/*
 * Chargement de la vue de l'addition et
 * création de la scène associée à cette vue
 */
FXMLLoader chargeurFXMLAddition = new FXMLLoader();
chargeurFXMLAddition.setLocation(getClass().getResource("VueAddition.fxml"));
conteneur = chargeurFXMLAddition.load();
sceneAddition = new Scene(conteneur, 500, 300);
```

Ces 3 scènes sont stockées en tant qu'attribut statique de la classe principale, le but est d'éviter de les recréer par la suite, à chaque fois que l'utilisateur demandera à changer de vue (cliquera sur un bouton). Remarquer la déclaration :

```
/** Scène permettant de gérer l'addition */
private static Scene sceneAddition;
```

En complément dans la classe principale, il faut coder 3 méthodes outils pour accéder respectivement à chacune des scènes. Par exemple, pour rendre active la scène de l'addition :

```
    /**
     * Permet de modifier la scène de la fenêtre principale
     * pour qu'elle devienne celle de l'addition
     */
    public static void activerAddition() {
        fenetrePrincipale.setScene(sceneAddition);
    }
```

Classe principale – *PlusieursCalculs*

```
/*
 * Classe principale de l'application permettant d'afficher des fenêtres au contenu
 * différent : addition, soustraction ... (ce sont les vues qui changent)
 */
package application;

import java.io.IOException;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.stage.Stage;
import javafx.scene.Parent;
import javafx.scene.Scene;

/**
 * Cette classe est la classe principale de l'application permettant à l'utilisateur
 * d'effectuer plusieurs calculs via des vues différentes.
 * Le but est d'illustrer comment changer la scène associée associée à la fenêtre
 * principale.
 *
 * Au lancement de l'application, la vue principale contient 2 boutons : addition
 * et soustraction.
 * Selon le bouton cliqué, une scène différente prend la place de la vue principale :
 * soit celle de l'addition, soit celle de la soustraction.
 * Sur chacune de ces 2 vues, il y a un bouton retour permettant d'afficher à
 * nouveau la vue principale
 * @author C. Servières
 * @version 1.0
 *
 */
public class PlusieursCalculs extends Application {


    /** Scène principale de l'application, celle qui contient les 2 boutons */
    private static Scene scenePrincipale;

    /** Scène permettant de gérer l'addition */
    private static Scene sceneAddition;

    /** Scène permettant de gérer la soustraction */
    private static Scene sceneSoustraction;



    /** Fenêtre principale de l'application
     * La scène qui lui est associée sera modifiée en fonction
     * des clics de l'utilisateur
     */
    private static Stage fenetrePrincipale;



    /**
     * Permet de modifier la scène de la fenêtre principale
```



```
     * pour qu'elle devienne celle de l'addition
     */
    public static void activerAddition() {
        fenetrePrincipale.setScene(sceneAddition);
    }

    /**
     * Permet de modifier la scène de la fenêtre principale
     * pour qu'elle devienne celle de la soustraction
     */
    public static void activerSoustraction() {
        fenetrePrincipale.setScene(sceneSoustraction);
    }


    /**
     * Permet de modifier la scène de la fenêtre principale
     * pour qu'elle devienne la scène principale, celle qui
     * affiche les 2 boutons addition et soustraction
     */
    public static void activerPrincipale() {
        fenetrePrincipale.setScene(scenePrincipale);
    }



    @Override
    public void start(Stage primaryStage) {
        try {

            /*
             *  chargement de la vue de la scène principale dans le conteneur
             *  de type Parent
             */
            FXMLLoader chargeurFXML = new FXMLLoader();
            chargeurFXML.setLocation(getClass().getResource("VueFenetrePrincipale.fxml"));
            Parent conteneur = chargeurFXML.load();

            /*
             * Création de la scène principale
             */
            scenePrincipale = new Scene(conteneur, 500, 300);

            /*
             * Chargement de la vue de l'addition et
             * création de la scène associée à cette vue
             */
            FXMLLoader chargeurFXMLAddition = new FXMLLoader();
            chargeurFXMLAddition.setLocation(getClass().getResource("VueAddition.fxml"));
            conteneur = chargeurFXMLAddition.load();
            sceneAddition = new Scene(conteneur, 500, 300);

            /*
             * Chargement de la vue de la soustraction et
             * création de la scène associée à cette vue
             */
            FXMLLoader chargeurFXMLSoustraction = new FXMLLoader();
            chargeurFXMLSoustraction.setLocation(getClass().getResource("VueSoustraction.fxml"));
            conteneur = chargeurFXMLSoustraction.load();
            sceneSoustraction = new Scene(conteneur, 500, 300);

            // on définit le titre, la hauteur et la largeur de la fenêtre principale
            primaryStage.setTitle("Application avec plusieurs fenêtres");
            primaryStage.setHeight(300);
            primaryStage.setWidth(500);

            /*
             *  on associe la scène principale à la fenêtre principale
             *  Cette dernière est stockée en tant qu'attribut afin d'être accessible
             *  dans les méthodes activer...  Celles qui permettent de rendre active
             *  l'une des 3 scènes
             */
```

```
            primaryStage.setScene(scenePrincipale);
            fenetrePrincipale = primaryStage;
            primaryStage.show();

        } catch (IOException e) {

            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }


    /**
     * Programme principal
     * @param args  non utilisé
     */
    public static void main(String[] args) {
        launch(args);
    }

}
```

Contrôleur de la vue principale – *ControleurFenetrePrincipale.java*

```
/* Controleur de la vue principale             05/23
 *
 */
package application;

import javafx.fxml.FXML;

/**
 * Classe contrôleur associée à la vue de la fenêtre principale.
 * La vue est dotée de 2 boutons :
 *     - addition => permet de modifier la vue,
 *                   c'est celle de l'addition qui s'affiche
 *     - soustraction => permet de modifier la vue,
 *                       c'est celle de la soustraction qui s'affiche
 * @author C. Servières
 * @version 1.0
 *
 */
public class ControleurFenetrePrincipale {

    @FXML
    private void gererClicAddition() {

        // échanger la vue courante avec celle de l'addition
        PlusieursCalculs.activerAddition();
    }

    @FXML
    private void gererClicSoustraction() {

        // échanger la vue courante avec celle de la soustraction
        PlusieursCalculs.activerSoustraction();
    }

}
```

Contrôleur de la vue de l'addition – *ControleurAddition.java*

```
/* Controleur de la vue permettant de faire une addition            05/23
 *
```



```
 */
package application;

import javafx.fxml.FXML;

/**
 * Classe contrôleur associée à la vue permettant de faire une addition.
 * Pour l'instant, la vue est dotée d'un seul bouton :
 *     - retour => permet de modifier la vue, afin de revenir à celle de
 *                 la fenêtre principale
 * @author C. Servières
 * @version 1.0
 *
 */
public class ControleurAddition {

    @FXML
    private void gererClicRetour() {

        // retour à la vue principale
        PlusieursCalculs.activerPrincipale();
    }
}
```

4) Plusieurs vues différentes dans une application

Le code ci-dessous est une variante permettant de gérer l'affichage de vues différentes dans une même application : cette fois ce sont les vues qui changent (et pas les scènes), de plus le code est mieux structuré grâce à des classes intermédiaires qui gèrent le changement des vues. Il s'agit de coder la même application que celle de la section précédente : addition et soustraction.

Classe *EnsembleDesVues*

```
/*
 * Gére la correspondance entre le code d'une vue (un entier) et le nom
 * du fichier fxml décrivant cette vue                      05/23
 */
package application;

/**
 * Classe outil qui établit la correspondance entre un code de vue (sous la
 * forme d'un entier) et le nom du fichier fxml contenant la vue associée
 * à ce code.
 * @author C. Servières
 * @version 1.0
 *
 */
public class EnsembleDesVues {

    /** Code de la vue principale */
    public static final int VUE_PRINCIPALE = 0;

    /** Code de la vue de l'addition */
    public static final int VUE_ADDITION = 1;

    /** Code de la vue de la soustraction */
    public static final int VUE_SOUSTRACTION = 2;

    /** Tableau contenant les noms des fichiers fxml des différentes vues
```

```
     * de l'application. Il y a une correspondance entre l'indice de la case
     * du tableau et le code de la vue défini en tant que constante
     */
    private static final String[] NOM_DES_VUES =
        { "VueFenetrePrincipale.fxml", "VueAddition.fxml", "VueSoustraction.fxml"};



    /**
     * Renvoie le nom du fichier fxml contenant la vue dont le code est donné
     * en paramètre
     * @param codeVue   code de la vue dont le fichier fxml doit être renvoyé
     * @return une chaîne contenant le nom du fichier fxml
     * @throw IllegalArgumentException levée si le code argument n'est pas valide
     */
    public static String getNomVue(int codeVue) {
        if (codeVue < 0 || codeVue >= NOM_DES_VUES.length) {
            throw new IllegalArgumentException("Code vue " + codeVue + " invalide");
        }
        return NOM_DES_VUES[codeVue];
    }

}
```

Classe *EchangeurDeVue*

```
/*
 * Gère l'échange entre les vues affichées par la scène de l'application     05/23
 */
package application;

import java.io.IOException;
import java.util.HashMap;
import java.util.Map;

import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.fxml.FXMLLoader;


/**
 * Classe outil permettant de gérer le changement de la vue affichée par la
 * scene de l'application.
 *
 * Afin d'optimiser, un cache est mis en place : l'objectif est que le code
 * fxml soit chargé une seule fois, donc lors du 1er chargement de cette vue.
 * Pour ce faire, une HashMap (table de hachage) fait la correspondance entre
 * le code de la vue (une constante de type entier définie dans la classe
 * EnsembleDesVues) et l'élément racine de la vue (de type Parent qui est la classe
 * Parente de toutes les classes conteneurs comme HBox, VBox, BorderPane ...)
 * @author C. Servières
 * @version 1.0
 *
 */
public class EchangeurDeVue {

    /**
     * HashMap ou table de hachage qui contient des paires formées de :
     *     - un code de vue sous la forme d'une constante de type entier définie
     *       dans la classe EnsembleDesVues
     *       Ce code est la clé de la paire
     *     - un conteneur correspondant à la vue définie dans le fichier fxml
     *        celui qui a pour code la valeur de la clé
     * Cette table est renseignée au 1er chargement d'une vue.
     * Au 2ème chargement, la vue recherchée n'est pas obtenue à partir du fichier
     * fxml, mais à partir de l'élément de type Parent associée à la clé (la clé
     * étant le code de la vue)
     */
```



```
    private static Map<Integer, Parent> cache;

    // création de la table cache
    static {
        cache = new HashMap<>();
    }

    /** Scene courante, ou scène qui est associée à la fenêtre principale */
    private static Scene sceneCourante;


    /**
     * Affecte à la sceneCourante la scène créée dans la méthode start, donc
     * celle associée à la fenêtre principale
     * @param nouvelleScene  Scene à affecter
     */
    public static void setSceneCourante(Scene nouvelleScene) {
        sceneCourante = nouvelleScene;
    }


    /**
     * Modifie la vue associée à la scène courante, pour qu'elle devienne celle dont
     * le code est donné en argument
     * La scène courante doit avoir été initialisée
     * @param codeVue  code de la vue à placer sur la scène courante
     */
    public static void echangerAvec(int codeVue) {
        if (sceneCourante == null) {

            // pas de scène courante : impossible de modifier sa vue
            throw new IllegalStateException("Echange de vue impossible. Pas de scène courante.");
        }

        try {
            Parent racine;      // recevra le conteneur racine de la vue à afficher

            if (cache.containsKey(codeVue)) {

                /*
                 *  la vue associée au codeVue est présente dans la table cache,
                 *  ce qui signifie qu'elle a déjà été chargée.
                 *  Pour optimiser, on ne la recharge pas. Au contraire, on la récupère
                 *  dans la table cache, à partir de sa clé (codeVue)
                 */
                racine = cache.get(codeVue);
            } else {

                /*
                 * La vue associée au codeVue n'est pas présente dans la table cache.
                 * Elle n'a donc pas encore été chargée. Il faut donc la charger et
                 * ensuite on la stocke dans la table (pour éviter de la recharger
                 * à nouveau si l'utilisateur souhaite revenir sur cette vue)
                 */
                racine = FXMLLoader.load(
                        EchangeurDeVue.class.getResource(EnsembleDesVues.getNomVue(codeVue)));

                // ajout de la vue à la table cache
                cache.put(codeVue, racine);
            }
            sceneCourante.setRoot(racine);
        } catch(IOException erreur) {

            // problème lors de l'accès au fichier décrivant la vue
            System.out.println("Echec du chargement de la vue de code " + codeVue);
        }
    }

}
```

Classe *ControleurFenetrePrincipale*

```
/* Controleur de la vue principale              05/23
 *
 */
package application;

import javafx.fxml.FXML;

/**
 * Classe contrôleur associée à la vue de la fenêtre principale.
 * La vue est dotée de 2 boutons :
 *     - addition => permet de modifier la vue,
 *                   c'est celle de l'addition qui s'affiche
 *     - soustraction => permet de modifier la vue,
 *                       c'est celle de la soustraction qui s'affiche
 * @author C. Servières
 * @version 1.0
 *
 */
public class ControleurFenetrePrincipale {

    @FXML
    private void gererClicAddition() {

        // échanger la vue courante avec celle de l'addition
        EchangeurDeVue.echangerAvec(EnsembleDesVues.VUE_ADDITION);
    }

    @FXML
    private void gererClicSoustraction() {

        // échanger la vue courante avec celle de la soustraction
        EchangeurDeVue.echangerAvec(EnsembleDesVues.VUE_SOUSTRACTION);
    }

}
```

Classe *ControleurAddition*

```
/* Controleur de la vue permettant de faire une addition            05/23
 *
 */
package application;

import javafx.fxml.FXML;

/**
 * Classe contrôleur associée à la vue permettant de faire une addition.
 * Pour l'instant, la vue est dotée d'un seul bouton :
 *     - retour => permet de modifier la vue, afin de revenir à celle de
 *                 la fenêtre principale
 * @author C. Servières
 * @version 1.0
 *
 */
public class ControleurAddition {

    @FXML
    private void gererClicRetour() {

        // retour à la vue principale
        EchangeurDeVue.echangerAvec(EnsembleDesVues.VUE_PRINCIPALE);
    }
}
```