

TP MVC

Model - Vue - Controller avec Java (Swing)

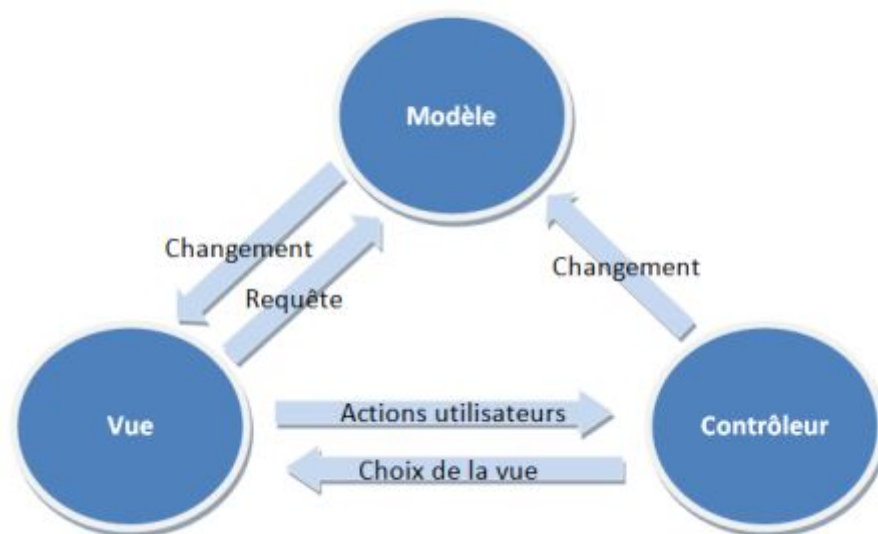
Ce TP a pour but de manipuler le design d'architecture Model-View-Controller. Il s'agit d'un TP noté avec pour rendu le code source applicatif réalisé.

Introduction

Modèle-vue-contrôleur est un motif d'architecture logicielle destiné aux interfaces graphiques lancé en 1978. L'objectif de ce design est de séparer les responsabilités au sein du code applicatif lorsqu'il est question de gérer un affichage et une interaction utilisateur au sein de votre logiciel.

Le motif est composé de trois types de modules ayant trois responsabilités différentes : les modèles, les vues et les contrôleurs.

- Un modèle (Model) contient les données à afficher.
- Une vue (View) contient la présentation de l'interface graphique.
- Un contrôleur (Controller) contient la logique concernant les actions effectuées par l'utilisateur.

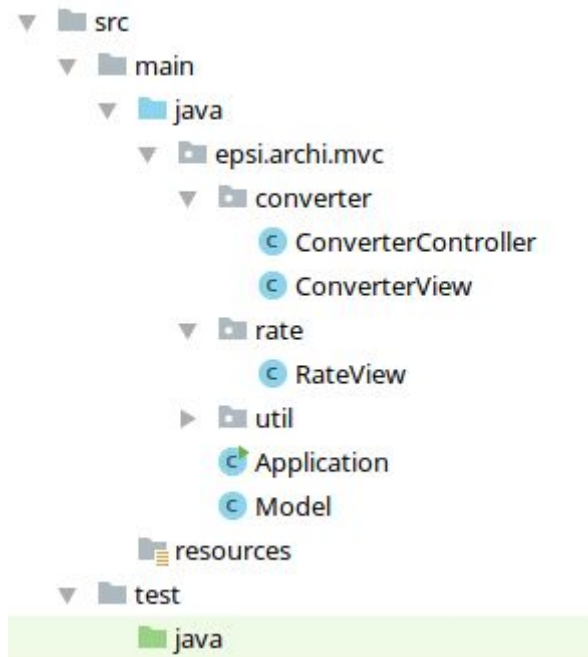


Cette architecture permet de découpler l'interface des données qu'elle manipule et permet par extension de brancher plusieurs vues (interfaces) sur les mêmes données métiers.

Nous allons voir comment réaliser cette architecture dans le cadre d'une application Java Swing (le code fourni gère tout l'affichage, aucune demande ne concerne la réalisation de l'interface graphique). L'application va permettre de convertir un montant en plusieurs autres monnaies en utilisant des taux de changes qui pourront être modifiés.

Une version 0 du code source est disponible sur le repository git <https://github.com/fteychene/epsi-i3-tpmvc> en tant que projet maven.

Il s'agit d'un projet maven qui fournit la structure suivante :



La classe `epsi.archi.mvc.Application` est le point d'entrée du programme.

Conversion

Dans cette première partie nous allons implémenter une interface en respectant le design MVC pour réaliser la conversion d'un montant en plusieurs devises.

The screenshot shows the MVC Converter EPSI application interface. At the top, there is an 'Amount' input field with the value '100.0', a 'Convert' button, and a 'Custom' checkbox that is checked. Below this is a 'Conversion' section displaying the results of the conversion for four currencies: USD, GBP, JPY, and CHF. Each currency is followed by its rate in curly braces and the converted amount. Below the 'Conversion' section is a 'Custom' section with a 'Change rate' input field containing '0.7543' and the resulting converted amount '75.42999999999999'. At the bottom, the text 'MVC Converter EPSI' is displayed.

Currency	Rate	Converted Amount
USD	{1.23072}	123.072
GBP	{0.88855}	88.85499999999999
JPY	{131.431}	13143.1
CHF	{1.17071}	117.071

Custom

Change rate: 0.7543 → 75.42999999999999

MVC Converter EPSI

Model

L'application va utiliser le modèle fourni par la classe `epsi.archi.mvc.Model`.

Le modèle correspond aux données métiers et au fonctionnement de l'application. Dans le cadre de cette application le modèle métier peut être ramené aux informations suivantes :

- Montant à convertir
- Taux de change EUR-USD
- Taux de change EUR-GBP
- Taux de change EUR-JPY
- Taux de change EUR-CHF

Ce modèle est pré implémenté, il s'agit d'un simple POJO Java.

Vue

L'interface graphique de cette fonctionnalité est la fenêtre déjà fournis par la classe `epsi.archi.mvc.converter.ConvertableView`.

```
public class ConvertableView {

    private JTextField amount;
    private JButton convert;
    private JCheckBox customToggle;

    // Conversion display
    private JLabel usd;
    private JLabel gbp;
    private JLabel jpy;
    private JLabel chf;

    // Custom panel
    private JPanel customPanel;
    private JTextField customRate;
    private JLabel customAmount;

    private ConverterController controller;

    public ConvertableView(Model model) {
        buildIHM();
    }

    ...
}
```

Les attributs de cette classe sont les représentations objets des champs et zones qui seront modifiables lors de l'interaction utilisateur avec l'application.

Ces champs seront instancié par la méthode `buildIHM()` qui gère l'affichage de l'interface utilisateur (positionnement et définition de l'interface).

Etape 1 : Enrichir la vue à partir du modèle

Ce modèle permet déjà de voir la séparation des responsabilités appliqué par le modèle MVC. La classe `epsi.archi.mvc.Model` est responsable du domain métier de l'application (dans notre cas uniquement stocker les valeurs de taux de change et du montant) alors que `epsi.archi.mvc.converter.ConvertView` est en charge de la vue utilisateur.

Néanmoins actuellement si vous lancez l'application ces deux éléments ne sont pas reliés et votre interface ne fait rien.

Lors de l'instanciation de la vue `epsi.archi.mvc.converter.ConvertView` le modèle utilisé est disponible et passé en paramètre du constructeur.

Nous pouvons donc nous en servir pour afficher les informations du modèle dans notre interface.

Créez une méthode `public void update(Model model)` qui va permettre de mettre à jour les champs de l'interface par rapport au `epsi.archi.mvc.Model` passé en paramètre.

```
public void update(Model model) {  
    double modelAmount = model.getAmount();  
    amount.setText(String.valueOf(modelAmount));  
    usd.setText(String.valueOf(modelAmount * model.getUsdChangeRate()));  
    gbp.setText(String.valueOf(modelAmount * model.getGbpChangeRate()));  
    jpy.setText(String.valueOf(modelAmount * model.getJpyChangeRate()));  
    chf.setText(String.valueOf(modelAmount * model.getChfChangeRate()));  
}
```

Appelez maintenant votre méthode `update` dans le constructeur de la vue pour mettre à jour ces champs après la création de l'interface graphique.

Si vous lancez l'application vous devriez voir que votre interface est bien alimentée par les valeur du modèle.

De même si vous changez la valeur du montant contenu dans le model avant l'affichage ces valeurs (en changeant la valeur à 20 par défaut par exemple).

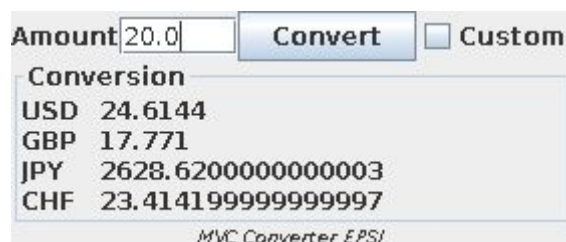


Amount: 0.0 Convert ☐ Custom

Conversion

USD	0.0
GBP	0.0
JPY	0.0
CHF	0.0

MVC Converter EPSI



Amount: 20.0 Convert ☐ Custom

Conversion

USD	24.6144
GBP	17.771
JPY	2628.6200000000003
CHF	23.414199999999997

MVC Converter EPSI

Etape 2 : Gérer les actions utilisateurs

Nous avons réussi à alimenter notre vue à partir de notre modèle. Mais lorsque nous modifions le champs et cliquons sur le bouton *Convert* l'application ne réagit toujours pas.

Dans le design MVC c'est le Controller qui est en charge de la gestion des actions utilisateurs pour mettre à jour notre modèle. La classe `epsi.archi.mvc.converter.ConvertController` va nous permettre de gérer les actions sur notre interface de conversion.

```
public class ConvertController {  
  
    private Model model;  
    private ConverterView view;  
  
    public ConvertController(Model aModel, ConverterView aConverterView) {  
        model = aModel;  
        view = aConverterView;  
        bind();  
    }  
  
    public void bind() {  
        // Add code to manage view inputs  
    }  
}
```

Dans la méthode `bind` nous allons réagir à l'événement de clic sur le bouton de conversion. Pour ajouter un comportement à un bouton nous pouvons utiliser la méthode `addActionListener` du composant `JButton` de la vue.

```
view.getConvert().addActionListener(new ActionListener() {  
    @Override  
    public void actionPerformed(ActionEvent e) {  
        // Action to be executed on button click  
    }  
});
```

Nous pouvons maintenant mettre à jour la source de donnée de notre vue lors du clic sur le bouton de conversion.

```
model.setAmount(Double.valueOf(view.getAmount().getText()));
```

Dernière étape, créer le controller lié à la vue. Pour cela nous allons instancier le contrôleur lors de l'initialisation de la vue.

```
public ConverterView(Model model) {  
    buildIHM();  
    update(model);  
    controller = new ConvertController(model, aConverterView: this);  
}
```

Nous avons lié notre vue à notre modèle via un contrôleur. Mais il reste une partie qui n'est pas encore fonctionnelle : la mise à jour de la vue lors d'une modification du modèle.

Si vous lancez l'application, lors du clic sur le bouton le modèle a beau être mis à jour avec la valeur présente dans l'interface celle-ci ne met pas à jour les informations de conversion lié

...



A partir de ce point les implémentations à réaliser ne seront plus fournis par le sujet mais vous serez toujours guidé par des instructions.

Etape 3 : Mise à jour de la vue lors d'un changement de modèle

Pour réaliser cette partie, nous allons utiliser le design pattern [Observer](#). Le pattern Observer définit une relation entre objets, de façon que, si un objet change d'état, tous ceux qui en dépendent en soient informés et mis à jour automatiquement.

Ce design pattern répond parfaitement au problème que nous rencontrons, nous voulons que les vues liées à un modèle soient mises à jour lors de sa modification.

Pour réaliser ce pattern les interfaces `epsi.archi.mvc.util.Observable` et `epsi.archi.mvc.util.Observer` ont été faites et sont disponibles dans le projet.

```
public interface Observable {
    void addObserver(Observer observer);
    void removeObserver(Observer observer);
    void notifyObservers();
}

public interface Observer {
    void update(Observable observable);
}
```



Implémentez l'interface `epsi.archi.mvc.util.Observer` pour la classe `epsi.archi.mvc.converter.ConvertView`.

La méthode `update` sera appelée par une notification du modèle lors d'une modification de celui-ci. L'objectif est donc de mettre à jour les champs à partir du modèle reçu.

Dans l'étape 1 nous avons réalisé une méthode qui nous permet de mettre à jour notre interface à partir d'un modèle. Nous pouvons donc utiliser cette méthode lors de l'appel à la méthode `update` lorsque l'observable est une instance de `epsi.archi.mvc.Model`.

Il vous suffit alors d'enregistrer la vue comme un observateur du modèle lors de son initialisation.



Implémentez l'interface `epsi.archi.mvc.util.Observable` pour la classe `epsi.archi.mvc.Model`.

L'objectif est de notifier des mises à jours de l'attribut amount.

Etape 4 : Et si c'est dynamique mais pas lié au modèle

Dans cette section nous allons rajouter une nouvelle fonctionnalité qui va nous faire voir un cas particulier par rapport au design MVC.

L'IHM du convertir cible possède une section "Custom" qui permet à l'utilisateur de définir un taux de change manuellement pour convertir son montant.

The screenshot shows a Java Swing window titled "MVC Converter EPSI". At the top, there is a text field labeled "Amount" containing "0.0", a "Convert" button, and a checked checkbox labeled "Custom". Below these is a section titled "Conversion" containing a list of currency pairs with their respective rates: USD 0.0, GBP 0.0, JPY 0.0, and CHF 0.0. Below the "Conversion" section is another section titled "Custom" which contains a text field labeled "Change rate" containing "1.0".

Cette section n'est pas affichée par défaut et doit l'être lors du click sur la checkbox.

The screenshot shows the same "MVC Converter EPSI" window, but the "Custom" section is now hidden. The "Amount" field still contains "0.0", the "Convert" button is present, and the "Custom" checkbox is now unchecked. The "Conversion" section with the currency list remains visible.

Si l'on suit le schéma MVC le clic sur la checkbox entraîne une action gérée par le contrôleur qui va modifier une valeur du modèle qui permettra de mettre à jour la vue pour afficher le panneau spécifique.

Or l'information de l'affichage d'un panneau dans l'interface graphique n'est pas une donnée métier et n'a donc pas de sens dans le modèle de l'application.

Pour gérer ce cas de figure nous allons toujours gérer l'action utilisateur du clic dans le contrôleur mais nous allons faire un appel directement à la vue dans celui-ci pour définir l'affichage du panneau.



Implémentez la gestion de l’affichage du panneau lors du click sur la checkbox.



Le panneau à afficher est représenté par l'attribut `customPanel` de la classe `epsi.archi.mvc.converter.ConvertableView`.



La classe `JPanel` possède une méthode `setVisible(boolean)` qui permet de définir si la zone doit être affichée.



Pour réagir à un click sur une checkbox il faut utiliser le code suivant

```
checkBox.addItemListener(new ItemListener() {
    @Override
    public void itemStateChanged(ItemEvent e) {
        if (e.getStateChange() == ItemEvent.SELECTED) {
            // Code if checkbox is checked
        } else {
            // Code if checkbox isn't checked
        }
    }
});
```



A partir de ce point les implémentations à réaliser ne seront plus fournies par le sujet et vous devez définir par vous même comment les réaliser.

Etape 5 : Conversion custom lors du clic

Maintenant que nous pouvons afficher notre zone de conversion custom il faut que celle-ci réagisse lorsque l’on convertit avec le bouton “Convert”.



Implémentez la conversion custom lors du clic sur le bouton “Convert”



Le champ qui gère l’input du taux de change custom est représenté par l'attribut `customRate` de la classe `epsi.archi.mvc.converter.ConvertableView`.



Le champ qui gère l’affichage de la valeur du taux de change custom est représenté par l'attribut `customAmount` de la classe `epsi.archi.mvc.converter.ConvertableView`.

Gestion des taux de changes

L'application possède une deuxième fenêtre qui permet de changer les taux de changes configurés dans le modèle applicatif (`epsi.archi.mvc.rate.RateView`).

Elle est actuellement commenté dans le code de la classe `epsi.archi.mvc.Application`

USD	1.23072
GBP	0.88855
JPY	131.431
CHF	1.17071

Cette vue est liée au même modèle que la vue `epsi.archi.mvc.converter.ConvertterView` existante.



Implémentez le modèle MVC pour la vue `epsi.archi.mvc.rate.RateView` lié au modèle `epsi.archi.mvc.Model`.



Pour activer la vue décommentez la ligne 17 `new RateView(model);` dans la classe `epsi.archi.mvc.Application`



Les deux vues étant liée au même modèle lors de la modification des taux de changes cette modification doit impacter la vue de conversion.



Les taux de changes sont représentés par les attributs `usdChange`, `gbpChange`, `jpyChange`, `chfChange` dans la vue .