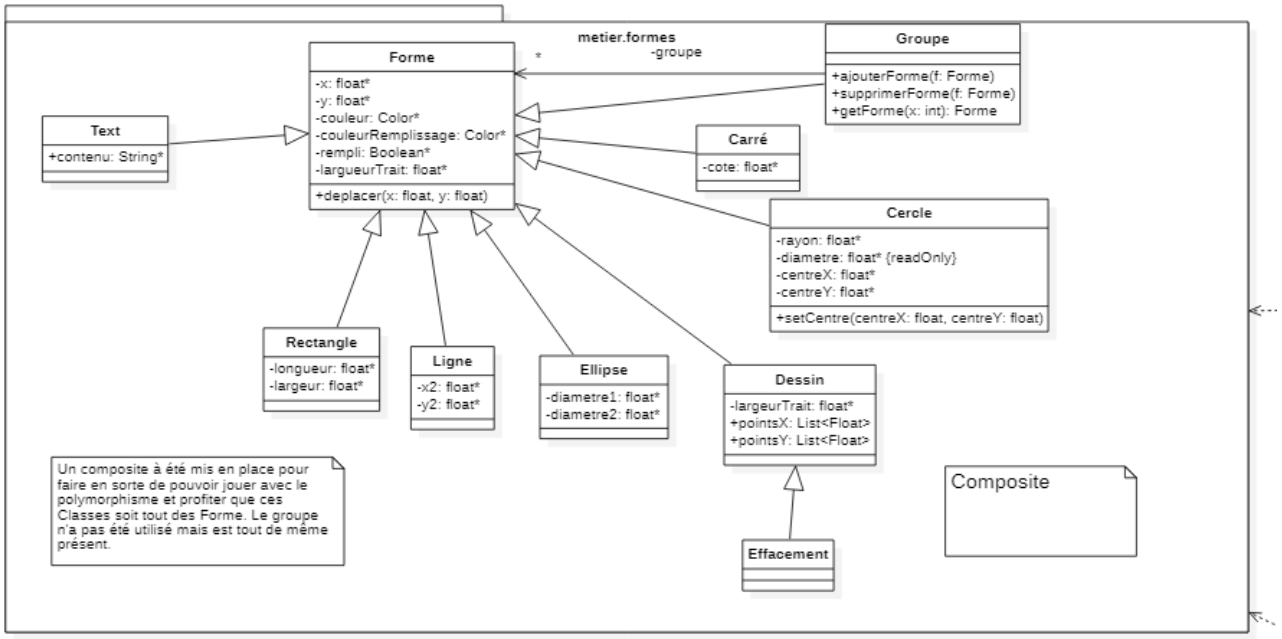
Preuves pour le projet JavaFX n°10 : ColorMaster (Paint revisité) réalisé par Yoann PERIQUOI

Documentation

Je sais concevoir un diagramme UML intégrant des notions de qualité et correspondant exactement à l'application que j'ai à développer :

Vous pouvez trouver donc le fichier documentation un fichier comportant le diagramme de classe de l'application avec quelques commentaires annoté dessus. Un aperçu :



Je sais décrire un diagramme UML en mettant en valeur et en justifier les éléments essentiels.

Comme vu sur l'exemple juste au-dessus des éléments essentiels comme le Composite mis en place sont mis en valeur par des notes qui décrivent l'utilité du patron ici.

Je sais documenter mon code et en générer la documentation :

La totalité du code est documenté en Java doc ainsi qu'avec des commentaires lors de méthodes complexe. La Java doc généré peut-être retrouvé dans le fichier documentation/JavaDoc. Exemple de code :

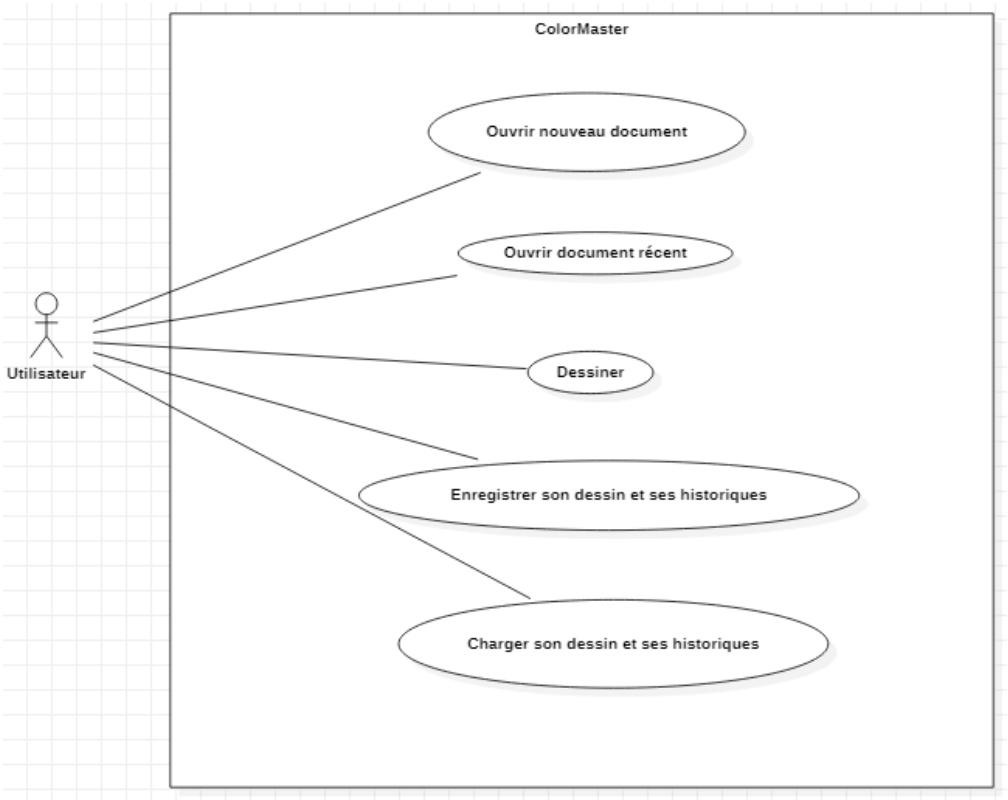
```java
/**
 * Méthode permettant de refaire une action qui a été annulé
 * @param gc contexte graphique du canvas de l'application sur laquelle on dessine
 */
public void redo(GraphicsContext gc) {
    //Pour faire annuler la dernière action on la récupére simplement puis on l'exécute
    if (!redoHistorique.empty()) {
        ICommande dernièreCommande = redoHistorique.pop();
        dernièreCommande.execute(gc);
        undoHistorique.push(dernièreCommande);
    }else{
        Alert alert = new Alert(Alert.AlertType.ERROR);
        alert.setTitle("Action impossible !");
        alert.setHeaderText("L'historique de redo est vide !");
        alert.setContentText("Il n'est plus possible de refaire une action.");
        alert.show();
    }
}
```

Je sais décrire le contexte de mon application, pour que n'importe qui soit capable de comprendre à quoi elle sert :

Cette description du contexte peut être retrouvé dans le fichier README qui est disponible dans le dossier documentation.

Je sais faire un diagramme de cas d'utilisation pour mettre en avant les différentes fonctionnalités de mon application :

Celui-ci est retrouvable dans le fichier contenant le diagramme UML le voici :



Code :

Je maîtrise les règles de nommage Java :

Les noms des classes sont en Pascal Case et les attributs sont en Camel Case… Ces règles sont retrouvées dans le code :

```
public void redo(GraphicsContext gc) {
    //Pour faire annuler la dernière action on la récupére simplement puis on l'exécute
    if (!redoHistorique.empty()) {
        ICommande dernièreCommande = redoHistorique.pop();
        dernièreCommande.execute(gc);
        undoHistorique.push(dernièreCommande);
    }else{
        Alert alert = new Alert(Alert.AlertType.ERROR);
        alert.setTitle("Action impossible !");
        alert.setHeaderText("L'historique de redo est vide !");
        alert.setContentText("Il n'est plus possible de refaire une action.");
        alert.show();
    }
}
```

Je sais binder bidirectionnellement deux propriétés JavaFX :

J'ai utilisé le binding bidirectionnel à plusieurs endroits dans le projet :

```
fileName.textProperty().bindBidirectional(nomFichier.textProperty());
dessinateurManager.fileNameProperty().bindBidirectional(nomFichier.textProperty());
```

FenetrePrincipal.java lignes 324 et 325

Je sais binder unidirectionnellement deux propriétés JavaFX :

Encore une fois une j'ai binder plusieurs fois une propriété unidirectionnellement :

```
//On bind les items de la list view à ceux de la liste observable du manager
laListView.itemsProperty().bind(recentManager.lesFichiersProperty());
```

FenetreAcceuil.java ligne 79

Je sais coder une classe Java en respectant des contraintes de qualité de lecture de code :

Dans les classes : les propriétés sont en premières puis on voit les constructeurs et enfin les méthodes :

```
public class DessinerCarre  implements ICommande {

    Carre carre;


    private final static String type = "Carre";


    public DessinerCarre(Carre carre) { this.carre=carre; }


    @Override
    public void execute(GraphicsContext gc) {
        //On vérifie si le carré est rempli et si c'est le cas on dessine le remplissage du carré
        if (carre.getRempli()){
            gc.setFill(carre.getCouleurRemplissage());
            gc.fillRect(carre.getX(),carre.getY(),carre.getCote(),carre.getCote());
        }
        //Et enfin on dessine le contour du carré
        gc.strokeRect(carre.getX(),carre.getY(),carre.getCote(),carre.getCote());
    }
}
```

Je sais contraindre les éléments de ma vue, avec du binding FXML :

Le Text n'apparait pas tant qu'aucun texte n'est renseigné à l'intérieur :

```
<Text layoutX="796.0" layoutY="349.0" strokeType="OUTSIDE" strokeWidth="0.0" fx:id="fichierSelected"
      disable="${fichierSelected.text.empty}"/>
```

fenetreAcceuil.fxml ligne 39.

Je sais définir une CellFactory fabriquant des cellules qui se mettent à jour au changement du modèle :

```
public class CelluleRecent extends javafx.scene.control.ListCell<Recent> {

    /**
     * Méthode définissant la structure de la cellule pour l'affichage d'un fichier récent
     * @param item item a afficher (ici des Recent)
     * @param empty si la case est vide ou non
     */
    @Override
    protected void updateItem(Recent item, boolean empty){
        super.updateItem(item,empty);
        //On oublie pas de faire le test histoire de ne pas avoir de "duplication"
        if(!empty){
            HBox container = new HBox();
            Label label1 = new Label();
            label1.textProperty().bind(item.fileNameProperty());
            Label label2 = new Label();
            label2.textProperty().setValue("Nom fichier : ".concat(item.getNom()).concat(" | Chemin : "));
            container.getChildren().add(label2);
            container.getChildren().add(label1);
            setGraphic(container);
        }else{
            //Si il y a rien à afficher alors on affiche rien
            setGraphic(null);
        }
    }
}
```

```
//On défini la cell Factory qu'elle devra utiliser pour afficher les fichiers récents
laListView.setCellFactory(__ -> new CelluleRecent());
```

Classe CelluleRecent.java et FenetreAcceuil.java ligne 92

Je sais développer une application graphique en JavaFX en utilisant FXML :

Mon application est composée de deux pages différentes que l'on peut retrouver dans le fichier Code/ressources/fxml. Un aperçu :

```
<GridPane xmlns="http://javafx.com/javafx"
          xmlns:fx="http://javafx.com/fxml"
          fx:controller="fenetre.FenetrePrincipal"
          stylesheets="/css/FenetrePrincipal.css"
          prefHeight="750" prefWidth="1100" fx:id="grid">
    <GridPane.columnSpan/>
    <GridPane.columnSpan/>

    <VBox GridPane.columnIndex="1" fx:id="vbox">
        <ToggleButton  text="Dessin" fx:id="dessinBtn" styleClass="custom-button"/>
        <ToggleButton text="Effacer" fx:id="effacerBtn" styleClass="custom-button"/>
        <ToggleButton text="Ligne" fx:id="ligneBtn" styleClass="custom-button"/>
        <ToggleButton text="Carré" fx:id="carreBtn" styleClass="custom-button"/>
        <ToggleButton text="Rectangle" fx:id="rectangleBtn" styleClass="custom-button"/>
        <ToggleButton text="Cercle" fx:id="cercleBtn" styleClass="custom-button"/>
        <ToggleButton text="Ellipse" fx:id="ellipseBtn" styleClass="custom-button"/>
        <ToggleButton text="Text" fx:id="textBtn" styleClass="custom-button"/>
```

Je sais éviter la duplication de code :

Utilisations de petites fonctions utilisées dans plusieurs autres méthodes pour ne pas dupliquer le code : exemple méthode charger historique présent dans le DessinateurManager.

```
private void chargerHistoriques(File file) {
    String path1 = file.getAbsolutePath() + "Undo.json";
    String path2 = file.getAbsolutePath() + "Redo.json";

    if (!Files.exists(Path.of(path1)) | !Files.exists(Path.of(path2))){
        Alert alert = new Alert(Alert.AlertType.ERROR);
        alert.setTitle("Erreur ouverture document");
        alert.setHeaderText("Les historique de undo et de redo n'ont pas pu être chargé");
        alert.setContentText("Un fichier de sauvegarde d'historique de undo et de redo n'a pas été trouvé.");
        undoHistorique= new Stack<ICommande>();
        redoHistorique= new Stack<ICommande>();
        alert.show();
    }
    undoHistorique = Chargement.charger(path1);
    redoHistorique = Chargement.charger(path2);
    setFileName(file.getName());
}
```

Je sais hiérarchiser mes classes pour spécialiser leur comportement :

J'utilise par exemple un manager qui vas ensuite utiliser une autre classe de sauvegarde afin de sauvegarder.

Je sais intercepter des évènements en provenance de la fenêtre JavaFX :

J'utilise énormément les évènements en provenance de la souris pour permettre le traçage des formes :

```
canvas.setOnMouseReleased(e->{
    if(outils.getSelectedToggle() != null) {
        dessinateurManager.definirFinFigure(e, gc, dessinBtn, effacerBtn,textBtn);
    }
});
```

Je sais maintenir, dans un projet, une responsabilité unique pour chacune de mes classes :

J'ai essayé de maintenir les responsabilités uniques en utilisant par exemple une classe pour la sauvegarde une autre pour le chargement… En utilisant ces classes dans un manager.

Je sais gérer la persistance de mon modèle :

J'ai utilisé deux persistances différentes pour persister mon modèle, les historiques de undo et de redo sont sauvegarder en Json grâce à la librairie Gson qui me permet alors de sauvegarder des éléments non sérializable comme la couleur de celle. J'ai aussi réalisé la persistance des fichiers ouverts récemment avec les méthodes write et read Object qui sont directement implémenté dans Java :

```java
public class ChargementRecent {
    /**
     * Méthode permettant de charger les fichiers ouverts récemment
     * @param lesFichiersObs Liste des fichiers
     */
    public static void charger(ObservableList<Recent> lesFichiersObs){
        File file = new File(System.getProperty("user.dir").concat("/recent"));
        //Si l'utilisateur à supprimer touts les fichiers récent il ne reste plus que 4 caractères dans le fichier
        //Si c'est le cas il n'y a pas besoin de le lire
        if ((file.length()>4)) {
            try {
                ObjectInputStream ois = new ObjectInputStream(new FileInputStream(file));
                ArrayList<Recent> lesFichiers = new ArrayList<>();
                int size = ois.readInt();
                for (int i = 0; i < size; i++) {
                    lesFichiers.add(new Recent((String) ois.readObject(), (String) ois.readObject(),(Boolean) ois.readObject()));
                }
                lesFichiersObs.addAll(lesFichiers);
            } catch (IOException | ClassNotFoundException e) {
                e.printStackTrace();
            }
        }
    }
}
```

```java
public class SauvegardeRecent {
    /**
     * Méthode permettant de sauvegarder des fichier récemment ouvert
     * @param lesFichiersObs liste des fichiers récemment ouvert
     */
    public static void sauver(ObservableList<Recent> lesFichiersObs){
        File file = new File(System.getProperty("user.dir").concat("/recent"));
        try {
            ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(file));
            oos.writeInt(lesFichiersObs.size());
            for (Recent recent: lesFichiersObs) {
                oos.writeObject(recent.getFileName());
                oos.writeObject(recent.getNom());
                oos.writeObject(recent.getEnregistre());
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Fichiers SauvegardeRecent.java et ChargementRecent.java permettant la persistance en local des fichiers récents ouverts.

Je sais surveiller l'élément sélectionné dans un composant affichant un ensemble de données :

Je surveille l'élément sélectionné dans la ListView et j'affiche le chemin d'accès au fichier dans un label disposé au-dessus :

```
//On défini un binding pour afficher le fichier sélectionné dans un label
laListView.getSelectionModel().selectedItemProperty().addListener(((observableValue, oldValue, newValue) -> {
    if(oldValue!=null){
        fichierSelected.textProperty().unbind();
    }
    if(newValue != null){
        fichierSelected.textProperty().bind(newValue.fileNameProperty());
    }
}));
```

FenetreAcceuil.java ligne 82 à 89. Ce mécanisme peut être retrouvé en exécutant l'application et sélectionnant un fichier récent à charger.

Je sais utiliser à mon avantage le polymorphisme.

J'ai utilisé le polymorphisme à la plusieurs reprise afin de simplifier les algorithmes des méthodes et de ne pas avoir à créer un switch énorme pour gérer les différents cas :

```
public void undo(GraphicsContext gc){
    //Pour faire un retour en arrière on supprimer juste le dernier élement de l'historique de undo
    //puis on retrace tout les figures restante dans l'historique
    if(!undoHistorique.empty()){
        gc.setFill(Color.WHITE);
        gc.fillRect( v: 0, v1: 0,  v2: 1080, v3: 720);
        ICommande derniereCommande = undoHistorique.pop();
        redoHistorique.push(dernieresCommande);
        for (ICommande c: undoHistorique) {
            c.execute(gc);
        }
    }else{
        Alert alert = new Alert(Alert.AlertType.ERROR);
        alert.setTitle("Action impossible !");
        alert.setHeaderText("L'historique de undo est vide !");
        alert.setContentText("Il n'est plus possible de revenir en arrière.");
        alert.show();
    }

}
```

Fichier DessinateurManager.java ligne 167 à 184. Ici par exemple j'ai seulement à exécuter des ICommande plutôt qu'a vérifier qu'elle dessinateur je vais devoir exécuter.

Je sais utiliser certains composants simples que me propose JavaFX.

Je sais utiliser certains layout que me propose JavaFX.

Dans mes pages en FXML j'utilise différent layout comme des VBox ou StackPane.

A l'intérieur sont disposé des éléments comme des ToggleButton qui me permettent de définir des groupes par la suite.

```
<VBox GridPane.columnIndex="1" fx:id="vbox">
    <ToggleButton  text="Dessin" fx:id="dessinBtn" styleClass="custom-button"/>
    <ToggleButton text="Effacer" fx:id="effacerBtn" styleClass="custom-button"/>
    <ToggleButton text="Ligne" fx:id="ligneBtn" styleClass="custom-button"/>
    <ToggleButton text="Carré" fx:id="carreBtn" styleClass="custom-button"/>
    <ToggleButton text="Rectangle" fx:id="rectangleBtn" styleClass="custom-button"/>
    <ToggleButton text="Cercle" fx:id="cercleBtn" styleClass="custom-button"/>
    <ToggleButton text="Ellipse" fx:id="ellipseBtn" styleClass="custom-button"/>
    <ToggleButton text="Text" fx:id="textBtn" styleClass="custom-button"/>
    <TextArea fx:id="textArea"/>
    <Label text="Couleur ligne" fx:id="couleurLigne"/>
    <ColorPicker minWidth="90" fx:id="selectionCouleur"/>
    <Label text="Couleur remplissage" fx:id="couleurRempl" wrapText="true"/>
    <ColorPicker minWidth="90" fx:id="selectionRempl"/>
    <Label text="1.0" fx:id="labelSlider"/>
    <Slider showTickLabels="true" fx:id="slider" showTickMarks="true"/>
    <Button text="Undo" fx:id="undoBtn" styleClass="custom-button"/>
    <Button text="Redo" fx:id="redoBtn" styleClass="custom-button"/>
    <Button text="Save" fx:id="saveBtn" styleClass="custom-button"/>
    <Button text="Open" fx:id="openBtn" styleClass="custom-button"/>
    <Label text="Nom fichier" fx:id="nomFichier"/>
    <TextArea fx:id="fileName" maxHeight="45" prefWidth="100"/>
    <Label text="Nom fichier" fx:id="details" wrapText="true"/>
</VBox>
<StackPane GridPane.columnIndex="2">
    <Canvas  fx:id="canvas"/>
</StackPane>
```

Fichier FenetrePrincipal.fxml.

Je sais utiliser GIT pour travailler avec mon binôme sur le projet.

J'ai travaillé seul sur le projet j'ai donc continué à faire des commit avec des messages mais je n'ai pas eu à créer de branche puisque le développement suivait mon activité.

Je sais utiliser le type statique adéquat pour mes attributs ou variables.

Les méthodes pour les classes du type Sauvegarder et Chargement sont en statiques car ceux-ci ne changent pas lors de l'instanciation de la classe :

```java
public static void sauvegarder(String file, Stack<ICommande> historique) {
    if (file != null) {
        GsonBuilder builder = new GsonBuilder();
        Gson gson = builder.setPrettyPrinting().create();
        String s = gson.toJson(historique);
        try (FileWriter writer = new FileWriter(file);
             BufferedWriter bw = new BufferedWriter(writer)) {
             bw.write(s);
        } catch (IOException ex) {
            System.out.println("Error!");
        }
    }
```

Fichier Sauvegarde.java.

Je sais utiliser les collections.

J'utilise plusieurs types de collection comme ici une pile pour la gestion des historiques de undo et redo mais aussi une ObservableList pour l'affichage des fichiers récents ;

```
Stack<ICommande> undoHistorique = new Stack<>();
```

```
private final ObservableList<Recent> lesFichiersObs = FXCollections.observableArrayList();
```

Fichier DessinateurManger.java ligne 62 et fichier RecentManger.java ligne 18.

Je sais utiliser les différents composants complexes (listes, combo…) que me propose JavaFX.

```
private final ObservableList<Recent> lesFichiersObs = FXCollections.observableArrayList();
```

Encore une fois j'utilise les listes d'observable afin d'afficher la liste des fichiers récents.

Fichier RecentManger.java ligne 18.

Je sais utiliser les lambda-expression.

```
openBtn.setOnAction(e->{
    dessinateurManager.charger(gc,e);
});
```

Utilisation des lambda-expression avec le symbole -> afin définir le code directement après la déclaration d'événement.

Je sais utiliser les listes observables de JavaFX.

```
private final ObservableList<Recent> lesFichiersObs = FXCollections.observableArrayList();
```

J'utilise les listes observables afin d'afficher les fichiers ouverts récemment dans la page fenetreAcceuil.

Fichier RecentManger.java ligne 18.

Je sais utiliser un convertisseur lors d'un bind entre deux propriétés JavaFX.

Je sais utiliser un formateur lors d'un bind entre deux propriétés JavaFX.

Utilisation à la fois d'un formateur et d'un convertisseur afin d'afficher le nombre de caractères du nom d'un document :

```
details.textProperty().bind(Bindings.format( s: "Nom fichier : %s caractères",
        Bindings.createStringBinding(
                () -> Integer.toString(dessinateurManager.getFileName().length()), nomFichier.textProperty())));
```

Fichier PagePrincipal.java ligne 335.

Je sais utiliser un fichier CSS pour styler mon application JavaFX.

J'utilise un peu de CSS afin d'ajouter du style à mon application comme des couleurs ou des styles sur les boutons.

```
.Vbox{
    -fx-background-color: grey;
    -fx-padding: 5px;
    -fx-max-width: 90px ;
}

.custom-button{

    -fx-background-color:
        linear-gradient(#f2f2f2, #d6d6d6),
        linear-gradient(#fcfcfc 0%, #d9d9d9 20%, #d6d6d6 100%),
        linear-gradient(#dddddd 0%, #f6f6f6 50%);
    -fx-background-radius: 8,7,6;
    -fx-background-insets: 0,1,2;
    -fx-text-fill: black;
    -fx-effect: dropshadow( three-pass-box , rgba(0,0,0,0.6) , 5, 0.0 , 0 , 1 );
}

.custom-button:selected {
    -fx-background-color: #ff0000;
    -fx-text-fill: black;
}
```

Fichier FenetrePrincipal.css présent dans le dossier ressources/css.