Contexte:

Ce projet réside en la réalisation d'un logiciel de dessin type "Paint" qui permet de dessiner avec son curseur ou bien de sélectionner des figures afin de les tracer. J'ai donc réalisé un logiciel nommé ColorMaster permettant d'effectuer cela. On peut par exemple tracer un rectangle en appuyant sur le bouton correspondant et en cliquant sur le canvas puis en déplacent la souris pour tracer sa forme. De plus, j'ai fait en sorte que l'on puisse faire un mécanisme de retour en arrière et en avant afin de simplifier l'utilisation de celui-ci. Ces undo et redo forment un historique d'action qui vont ensuite pouvoir être sauvegardé lors de la sauvegarde du dessin. Lorsque l'on finit de traiter son fichier et qu'on l'ouvre à nouveau on a donc encore accès au retour en arrière et avant. De plus j'ai voulu implémenter une mécanique permettant de suivre les fichiers qui ont été ouvert récemment et ainsi les ouvrir depuis une liste.

La difficulté de ce projet a résidé dans l'équipe puisque le camarade qui était censé fournir un travail équivalent au miens n'a même pas écrit une seule ligne de code.

Cela m'a donc pénalisé dans la réalisation du projet et je n'ai pas totalement pu faire ce que je voulais.

En effet il manque énormément de refactorisation de code qui me permettrait par exemple de rendre la responsabilité des classes unique à certains endroits ou bien rendre l'interface plus responsive, ect...

instancier de manière vide. metier fenetre.commande La plupart des méthodes hérités d'un héritage ne sont pas à metier.formes Groupe nouveau mentionné pour gagner de la place et de la lisibilité. -groupe **Forme** On remarque par ailleur la présence de 2 patrons de conceptions différents que sont la Commande et le «interface» +ajouterForme(f: Forme) **ICommande** -x: float* +supprimerForme(f: Forme) DessinerText -y: float* **DessinerCarre** +execute(gc: GraphicsContext) Composite. +getForme(x: int): Forme -couleur: Color* **Text** -carre: Carre → -couleurRemplissage: Color* Carré -type: String {readOnly} -type: String {readOnly} -rempli: Boolean* +contenu: String* +DessinerText(text: Text) -cote: float* -largueurTrait: float* +DessinerCarre(carre: Carre) +@execute(gc: GraphicsContext) +@execute(gc: GraphicsContext) +deplacer(x: float, y: float) Cercle -diametre: float* {readOnly} DessinerCercle -centreX: float* DessinerDessin DessinerEffacement -cercle: Cercle -centreY: float* -type: String {readOnly} -dessin: Dessin -effacement: Effacement +setCentre(centreX: float, centreY: float) +Attribute1 -type: String {readOnly} -type: String {readOnly} Rectangle +DessinerCercle(cercle: Cercle) +DessinerDessin(dessin: Dessin) +DessinerEffacement(effacement: Effacement) Ligne +@execute(gc: GraphicsContext +@execute(gc: GraphicsContext) +@execute(gc: GraphicsContext) -longueur: float* Ellipse Dessin -x2: float* -y2: float* -largeur: float* DessinerEllipse -diametre1: float* -largeurTrait: float* DessinerLigne -diametre2: float* -ellipse: Ellipse +pointsX: List<Float> -type: String {readOnly} -ligne: Ligne +pointsY: List<Float> -type: String {readOnly} Ces classes ont permises de +DessinerEllipse(ellipse: Ellipse) mettre en place une partie du Un composite à été mis en place pour +@execute(gc: GraphicsContext) +DessinerLigne(ligne: Ligne) faire en sorte de pouvoir jouer avec le polymorphisme et profiter que ces Classes soit tout des Forme. Le groupe n'a pas été utilisé mais est tout de même patron de conception Commande permettant d'exécuter le dessin Composite +@execute(gc: GraphicsContext) d'une forme n'importe laquelle soit-elle. Cela permet donc la mise en place d'historique comme le undo et le redo. Ce paquet Commande présent. Effacement représente dans les commandes concrète à exécuter. fenetre.dessinateur «interface» **IDessineOnMousePressed** «interface» metier.persistance **IDessine** +definirFormeOnMousePressed(event: MouseEvent, gc: GraphicsContext, couleur: Color, couleurRemplissage: Color) +dessiner(GraphicsContext gc) ChargementRecent «interface» **IDessineOnMouseReleased** Chargement +charger(lesFichierObs: ObservableList<Recent> +definirFormeOnMouseReleased(event: MouseEvent) +charger(file: String Dessinateur +commande: ICommande **DessinateurCercle** Sauvegarde +dessiner(GraphicsContext gc) -cercle: Cercle* +definirFormeOnMousePressed(event: MouseEvent, gc: GraphicsContext, couleur: Color, couleurRemplissage: Color) +sauvegarder(file: String, historique: Stack<ICommande> +definirFormeOnMouseReleased(event: MouseEvent) Recent -fileName: StringProperty* {readOnly} -nom: String* DessinateurRectangle | -enregistre: Boolean* SauvegardeRecent DessinateurText DessinateurLigne DessinateurEllipse **DessinateurCarre** -rectangle: Rectangle* +fileNameProperty(): StringProperty -text: Text* -ligne: Ligne* -ellipse: Ellipse* +sauver(lesFichiersObs: ObservableList<Recent> +Recent(fileName: String, nom: String, enregistre: Boolean) -carre: Carre* La classe Recent est DessinerDeserializer **DessinateurDessin** disposé ici car elle sers -formeTypeElementName: String {readOnly} uniquement à servir de -dessin: Dessin* -gson: Gson {readOnly} modèle pour stocker les fichiers récents -formeTypeRegistery: Map {readOnly} +definirPendant(e: MouseEvent, gc: GraphicsContext) Les dessinateur permettent simplement de définir la forme à dessiner lors de +DessinerDeserializer(formeTypeElementName: String)
+registerShapeType(formeTypeName: String, formeType: Class) DessinateurEffacement l'intéraction de l'utilisateur avec l'application. +deserialize(json: JsonElement, type: Type, jsonDeserializationContext; JsonDeserializationContext): ICommande -effacement: Effacement* +definirPendant(e: MouseEvent, gc: GraphicsContext) Le chargement et la sauvegarder ont été implémenter de manière à être utilisé de manière static car ces méthodes ne changents pas à l'exécution «interface» **IDessinePendant** +definirPendantFigure(e: MouseEvent, gc: GraphicsContext) DessinateurManager +dessinateur: Dessinateur +dessinateurText: DessinateurText +dessinateurDessin: DessinateurDessin +dessinateurEffacement: DessinateurEffacement +undoHistorique: Stack<ICommande> +redoHistorique: Stack<ICommande> +fileName: StringProperty* +fileNameProperty(): StringProperty
+definirDebutFigure(slider: Slider, e: MouseEvent, gc: GraphicsContext, couleur: Color, couleurRempl: Color, outils: ToggleGroup, dessinBtn: ToggleButton, effacerBtn: ToggleButton, textBtn: ToggleButton, textArea: TextArea)
+definirPendantFigure(dessinBtn: ToggleButton, effacerBtn: ToggleButton, e: MouseEvent, gc: GraphicsContext)
+definirFinFigure(e: MouseEvent, gc: GraphicsContext, couleur: Color, couleurRemplissage: Color, outils: ToggleButton, dessinBtn: ToggleButton, effacerBtn: ToggleButton, textBtn: ToggleButton) +undo(gc: GraphicsContext) +redo(gc: GraphicsContext) launch +sauvegarder(canvas: Canvas, event: Event) +chargerHistorique(file: File) +chargerFichier(gc: GraphicsContext, recentManager: RecentManager) +chàrger(gc: GraphicsContext gc, event: Event) Launch +start(stage: Stage) Classe nécessaire à la Classe permettant de définir RecentManager **FenetrePrincipal** la fenêtre s'ouvrant lors du mise en place d'une -lesFichierObs: ObservableList<Recent> CelluleRecent CellFactory et démarrage de l'application +fileName: TextArea -lesFichiers: ListProperty<Recent>* permettant de définir +grid: GridPane #updateItem(item: Recent, empty: boolean) comment un fichier +lesFichiersProperty(): ListProperty<Recent> +details: Label récent s'affiche +RecentManager() +vbox: VBox +dessinBtn: ToggleButton +charger() +sauvegarder() +effacerBtn: ToggleButton +ligneBtn: ToggleButton +add(recent: Recent) +carreBtn: ToggleButton +del(recent: Recent) +rectangleBtn: ToggleButton +cercleBtn: ToggleButton Code-behind des fenetres comportant le même nom +ellipseBtn: ToggleButton mais en extension FXML. C'est la méthode initialize +textBtn: ToggleButton qu'il est indiqué tout les intéractions avec les +tableauOutils: ToggleButton éléments de la vue. Ces deux utilise le manager +selectionCouleur: ColorPicker afin d'utiliser les différentes méthode du modèle +selectionRempl: ColorPicker +textArea: TextArea +slider: Slider **FenetreAcceuil** +labelSlider: Label +nomFichier: Label +btnOuvrir: Button +undoBtn: Button +fileName: TextField +redoBtn: Button +btnNouveauDoc: Button +saveBtn: Button +laListView: ListView<Recent> +openBtn: Button +fichierSelected: Text +tableauBtn: Button +btnAjouter: Button

fenetre

+canvas: Canvas

+initialize()

+btnSupprimer: Button

+initialize()

Les propriétés qui terminent pas des * possèdent des

Les constructeurs ne sont pas représenter quand l'objet est

getters et setters dans la classe correspondante.

