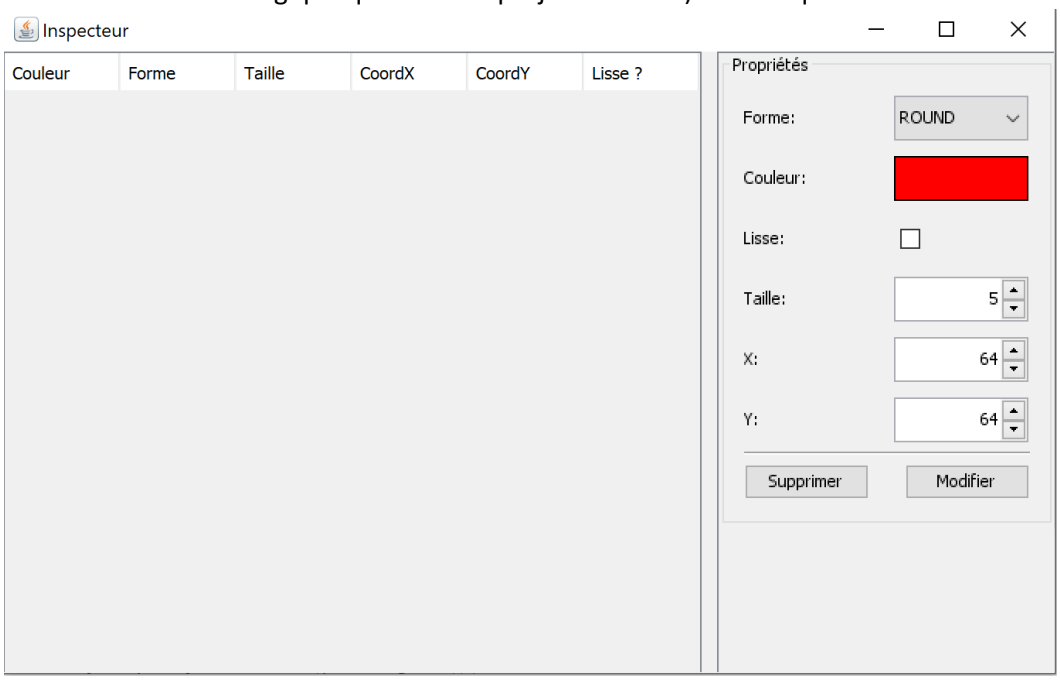
TP4

Recommandation : Veuillez lire le sujet en entier avant de le commencer !

Le TP3 était basé sur le fait de dessiner sur une ardoise avec une couleur choisie par l'utilisateur avec un pinceau d'une forme carrée ou ronde. Ce TP reprend le code du TP3 et lui apporte des modifications. L'idée étant de pouvoir prochainement supprimer ou modifier la forme, la position, la couleur d'un point du dessin. Pour rappel, un dessin sur l'ardoise est une suite de points placés à des coordonnées précises. Mais dans ce TP4, on va seulement se concentrer sur l'affichage des objets « Paint » dans une JTable.

1) Activez le menu Edition et ajoutez-y une JCheckBoxMenuItem « Inspecteur ». Ce menu est désactivé par défaut car aucun objet « Slate » n'a été créé avec le menu « Fichier>Nouveau ». Mais dès lors que cela sera fait, ce menu s'activera pour permettre à l'utilisateur de l'utiliser. Mettez en œuvre ce mécanisme.

2) Créez une nouvelle **JFrame** « Inspector » dans le package « iut.tp.ardoisemagique ». Par défaut, cette fenêtre se referme juste après un clic sur la croix en haut à droite (contrairement à la JFrame ArdoiseMagique qui ferme le projet en entier). Voici à quoi elle doit ressembler :



Son ContentPane possède un BorderLayout avec un hgap de 10px.

- Sur la partie de droite, il y a un JPanel *(nommé : rightPanel)*. Son layout est de type BoxLayout et est orienté à la verticale. Ce qui permet d'ajouter des panneaux les uns par-dessus les autres. Dans ce JPanel, il y a un JPanel *(nommé : propertiesPanel)*. Celui-ci contient 2 boutons « Supprimer » et « Modifier », un JSeparator au-dessus et un JPanel *(disposé en GridLayout) encore au dessus*. Celui-ci contient les 6 propriétés d'un point du dessin. Ces 6 propriétés correspondent aux 6 attributs de la classe Paint. Les composants utilisés sont : JLabel pour les titres des propriétés, JComboBox pour la forme d'un point, JPanel pour sa couleur, JCheckBox pour déterminer si le point est lisse ou pas et JSpinner pour la taille et les coordonnées x, y du point. Ensuite rendez le JPanel « propertiesPanel » invisible dans le constructeur *(juste après l'appel à la méthode « initComponents(); »)*. En effet ce panneau ne devra apparaitre que si une ligne de la JTable est sélectionnée *(voir dans le prochain TP)*.

- Sur la partie centrale, il y a une JScrollPane avec une JTable à l'intérieur. Mais si vous utilisez la palette NetBeans, il vous suffit simplement d'ajouter la JTable. La JScrollPane sera ajoutée automatiquement. Pour le moment, n'essayez pas de mettre des noms de colonnes. Cela viendra plus tard.

La JComboBox qui gère la forme du pinceau possède un modèle qui ne peut prendre que les objets Tool.ROUND et Tool.SQUARE.

3) Dans la question 1, on a ajouté une JCheckBoxMenuItem dans le menu « Edition ». Lorsque l'on clique sur ce menu, la fenêtre « Inspector » (cf question 2) apparait. Le coche montre que la fenêtre est bien visible. Si l'on reclique sur ce même menu, la JCheckBoxMenuItem doit se décocher ce qui fait disparaitre la fenêtre « Inspector ». De même que si on la ferme par sa croix en haut à droite, celle-ci se ferme et décoche la JCheckBoxMenuItem. On constate donc que cela marche dans les deux sens. Mettez en œuvre ce mécanisme.

4) Maintenant, il est temps de créer un « Model » pour notre JTable. Créez une nouvelle classe dans le package « iut.tp.ardoisemagique » appelée « TableModelInspector ». Cette classe doit étendre « AbstractTableModel ». À ce stade l'IDE vous demande de redéfinir « getRowCount() », « getColumnCount() » et « getValueAt() ». Pour réaliser cela, reprenez le cours sur les AbstractTableModel. L'exemple du cours est complet pour vous permettre de définir le nombre de colonnes, de lignes… Gardez à l'esprit que ce modèle va gérer des objets « Paint ».

5) Dans le cours, c'est le modèle de notre JTable qui contient la liste des objets à afficher dans la JTable. Dans le TP3, l'équivalent de cette liste se trouve dans la classe « Slate ». Donc le mieux à faire est de déplacer cette liste dans le modèle. Ne serait-ce pour éviter un doublon de liste de « Paint » dans le projet.

   Bien sûr cela va provoquer certains problèmes. Car comment va se débrouiller l'objet Slate pour redessiner son dessin s'il n'accède plus à la liste des points ? Eh bien parce que vous devrez créer un objet TableModelInspector *(c'est l'objet qui contient maintenant notre liste)* avant de créer un objet Slate. Cet objet Slate devra donc récupérer ce model en paramètre du constructeur. Et là, c'est presque gagné. L'objet « Slate » peut posséder la liste de tous les points. Mais il faudra créer la bonne méthode dans le model pour que celui-ci puisse y accéder réellement. Une autre classe a besoin impérativement de ce modèle. C'est la nouvelle fenêtre « Inspector ». Il sera donc important de lui transmettre ce modèle lorsque la fenêtre s'affichera. Pour cela vous pouvez créer une méthode setVisible qui prend en paramètre le modèle. Cette méthode aura également pour fonction de faire un setVisible(true) sur elle-même.

6) Lorsque la fenêtre « Inspector » s'affiche, attribuez le modèle à la JTable.

Maintenant si vous dessinez sur l'objet Slate, la JTable de la fenêtre « Inspector », se remplie automatiquement.