TP-Projet 6 Visualiser le modèle d'usine de production de biens grâce aux interfaces et au MVC

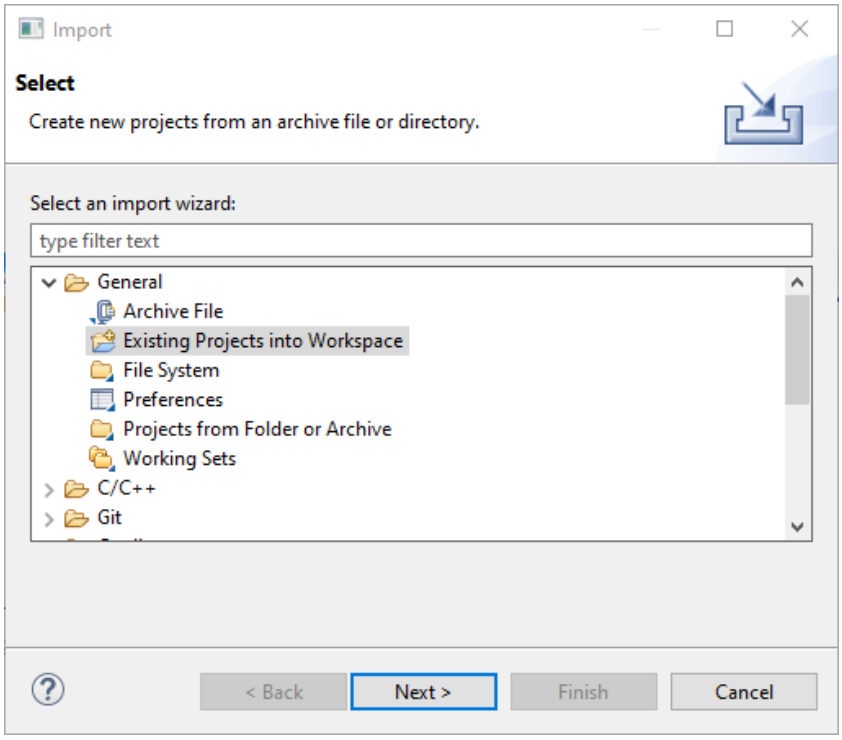
Au cours du TP précédent, nous avons modéliser la structure du système de production robotisé pour lequel nous voulons développer un simulateur. Nous avons donc créé un **modèle objet** de l'usine de production. Au cours de ce TP, nous allons modifier ce modèle afin de permettre de le visualiser grâce à une interface graphique qui vous est fournie. Pour cela, il faudra que votre modèle implémente les interfaces Java fournies par cette interface graphique.

Tel que vu en classe, une interface fournie un **point de vue** sur les classes qui l'implémentent. Ces interfaces spécifient donc un simple point de vue **figures 2D** du modèle requis par l'interface graphique fournie. En effet, tout ce que sait faire cette interface graphique consiste à afficher des figures géométriques (de forme rectangulaire, ovale ou polygonale) dans un canevas de dessin.

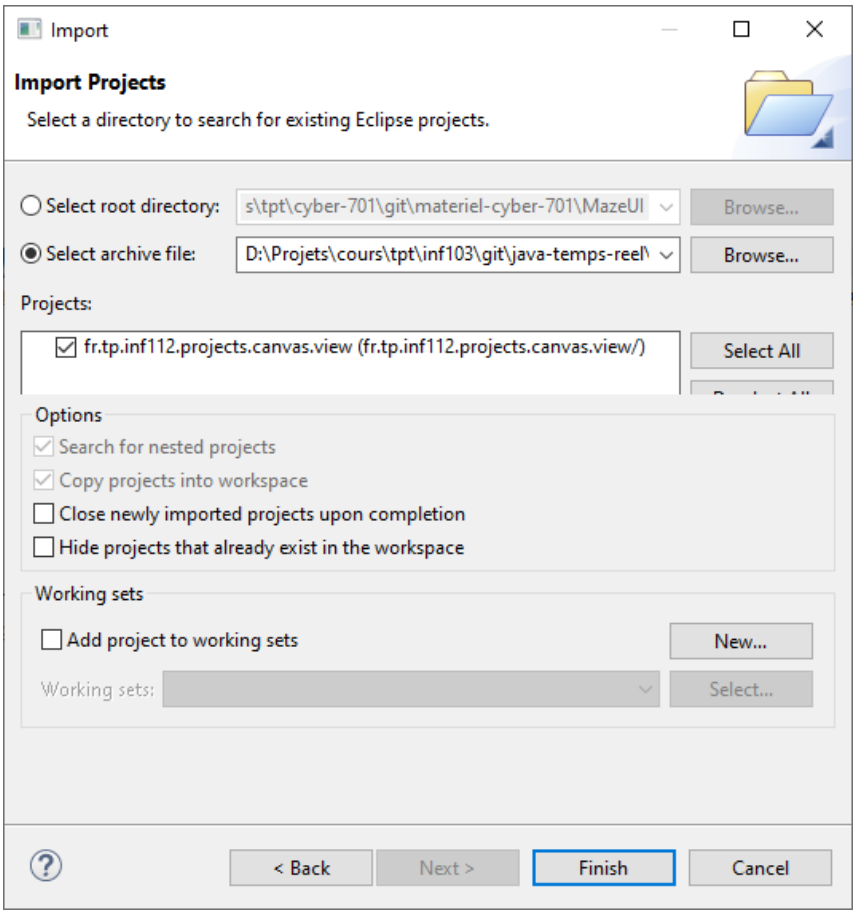
L'interface graphique s'appuie sur le patron de conception Modèle-Vue-Contrôleur (MVC) tel que vu en classe. Ainsi, lorsque le modèle est modifié, il notifiera alors la vue (l'interface graphique) qu'il a été modifié et celle-ci alors réaffichera tout le canevas afin de s'assurer que les figures affichées correspondent bien aux données du modèle.

L'interface graphique de visualisation de figures

Télécharger le code de l'interface graphique de visualisation de figures qui se trouve ici. Dans Eclipse cliquez sur le menu *File>>Import*. Dans la boîte de dialogues qui s'affiche, sélectionner *Existing Projects into Workspace.*



Cliquez sur le bouton *Next*. Dans la boîte de dialogue qui s'affiche sélectionnez le radio bouton *Select Archive File* et naviguez vers l'archive téléchargée précédemment.



Un projet nommé *fr.tp.inf112.projects.canvas.view* devrait apparaître dans le workspace d'Eclipse. Examinez le contenu de ce projet. Il devrai contenir les interfaces **`Canvas`**, **`Figure`**, **`Observable`**, **`Observer`**, **`OvalFigure`**, **`PolygonFigure`** et **`RectangleFigure`** dans le package **`fr.tp.inf112.projects.canvas.model`**. Ces interfaces seront a faire implementer par différentes classes de votre modèle en fonction de vos choix de visualisation des composants de votre modèle d'usine de rpduction de biens.

Dans le package **`fr.tp.inf112.projects.canvas.controller`** se trouve une dernière interface nommée **`Controller`** que devra implémenter le contrôleur de votre application.

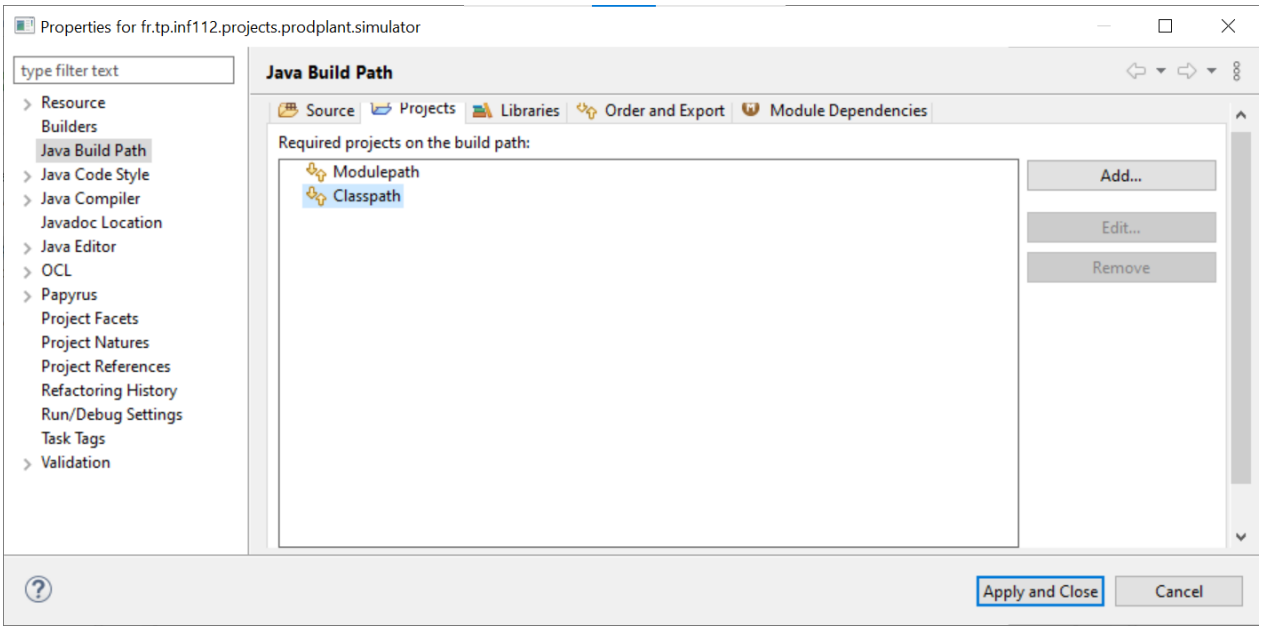
Dans le package **`fr.tp.inf112.projects.canvas.view`** se trouvent les classes qui réalisent l'interface graphique qui servira à visualiser votre modèle dusine de prodiction de biens lorsque celui-ci aura mplémenter les interfaces sus-mentionnées.

Faire implémenter les interfaces de figures de l'interface graphique par votre modèle d'usine de production de biens

Tout composant de notre usine de production peut être représenté par une figure géométrique 2D. Ainsi, votre classe abstraite **`Component`** devra implémenter l'interface **`Figure`** :

`public abstract class Component implements Figure`

Cependant, afin que votre projet puisse utiliser les interfaces et classes du projet de l'interface graphique, il faut configurer votre projet pour lui dire d'utiliser le projet de l'interface graphique. Pour ce faire, sélectionner votre projet dans l'explorateur de packages (*Package Explorer*), puis faire un clic droit et sélectionner le menu *Properties*. La boite de dialogue suivante sera affichée.



Dans cette fenêtre, sélectionnez la branche *Java Build Path* et l'onglet *Projects*. Dans cet onglet, sélectionner *Classpath* et cliquez sur le bouton *Add* afin d'ajouter le projet de l'interface graphique fournit.

L'interface graphique ne sait dessiner que des figures de formes de rectangulaire (**`RectangleFigure`**), ovale (**`OvalFigure`**) ou polygonale (**`PolygonFigure`**). Pour chaque sous-classe **concrète** de **`Component`**, faire implémenter l'une de ces classes en fonction de la forme désirée. Ainsi, votre classe Robot pourrait implémenter **`OvalFigure`** tandis que les classes **`Room`** ou **`Area`** pourraient implémenter la classe **`RectangleFigure`**.

Créer les méthodes requises par ces interfaces par vos classes. Ainsi, la classe Robot si elle étend la classe **`OvalFigure`** devra implémenter les méthodes **`getWidth()`** et **`getHeight()`** qui devront retourner la taille du robot en pixels. Ces données devront être récupérées de l'objet de type **`Dimension`** associé au composant.

Faites de même pour chaque classe composant de votre modèle devant être visualisée par l'interface graphique à l'exception de la classe **`Factory`**. En effet, cette dernière devra plutôt implémenter l'interface **`Canvas`**, qui représente le conteneur des figures. En effet, la classe **`Factory`** contient les composants à représenter sous forme de figures dans l'interface graphique.

Notez que l'interface **Canvas** spécifie une méthode **Collection<Figure> getFigures()**. Puisque toutes vos composants implémentent l'interface **Figure**, il sera possible de directement retourner l'attribut de votre classe **Factory** qui contient les composants. Cependant, il se peut que vous ayez à transtyper cette référence comme suit :

```
public Collection<Figure> getFigures() {
    return (Collection) components;
}
```

Faire implémenter l'interface Observable par votre modèle

Afin de notifier la vue lorsque ses données changent, votre modèle devra implémenter l'interface **Observable**.

```
public class Factory extends Component implements Canvas, Observable {
```

Vous devrez ensuite implémenter les méthodes demandées par l'interface tel que vu en cours.

Implémenter le contrôleur

Il ne reste plus qu'à fournir une implémentation de l'interface **Controller** fournie par l'interface graphique, qui elle utilisera cet objet afin d'obtenir les données de votre modèle.

Créez une classe nommée **SimulatorController** qui implémentera l'interface **Controller**. En lisant la Javadoc des méthodes de l'interface **Controller**, fournissez des implémentations des méthodes demandées. Notez que votre classe devra définir un attribut de type **Factory** afin d'en récupérer les données à fournir par les méthodes devant fournir les données du Canvas de l'interface graphique (**getCanvasName()**, **getCanvasWidth()**, etc.).

Cette classe de contrôleur servira également à démarrer et arrêter la simulation via les méthodes **startAnimation()** et **stopAnimation()** de l'interface **Controller** que nous n'implémenteront qu'au prochain TP. Pour l'instant, il vous suffit de coder des méthodes au contenu vide.

Exécuter l'interface graphique

Créez une classe nommée **SimulatorApplication** afin de représenter l'application complète incluant le modèle, le contrôleur et la vue (interface graphique).

Dans cette classe, créez une méthode **main** afin de lancer l'application. Dans cette méthode, instanciez un modèle d'usine comme vous l'avez fait au TP précédent. Puis instanciez votre contrôleur en lui fournissant l'instance de **Factory** créée précédemment. Puis, instanciez la classe **CanvasViewer** fournie par l'interface graphique en lui fournissant cotre contrôleur. Finalement, ajoutez la vue **CanvasViewer** comme observateur de votre modèle via la méthode **addObserver** de votre contrôleur. Votre code devrait ressembler au code suivant :

```
final Controller controller = new SimulatorController(factory);
final CanvasViewer viewer = new CanvasViewer(controller);
```

```
controller.addObserver(viewer);
```

Lancez la méthode main et vérifiez que votre modèle s’affiche correctement dans l’interface graphique.