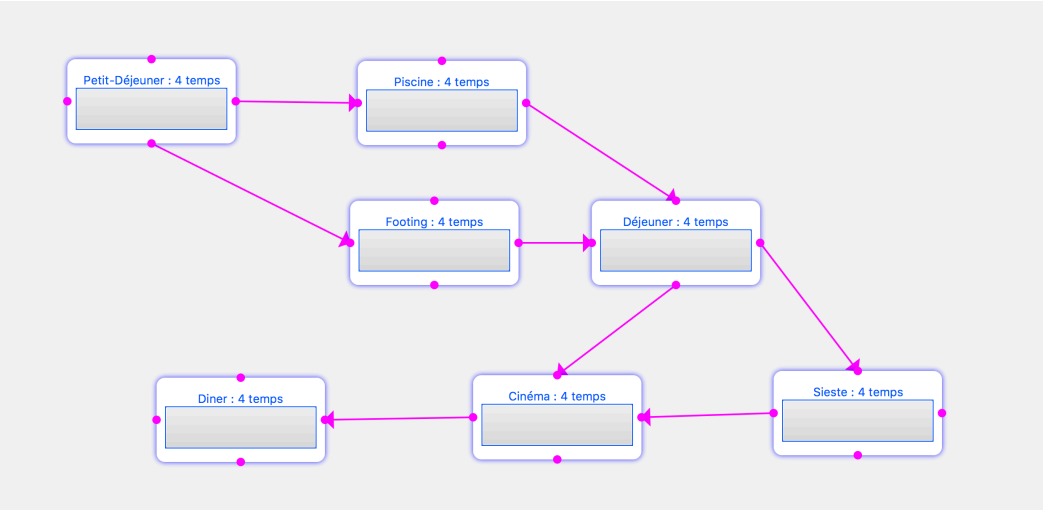
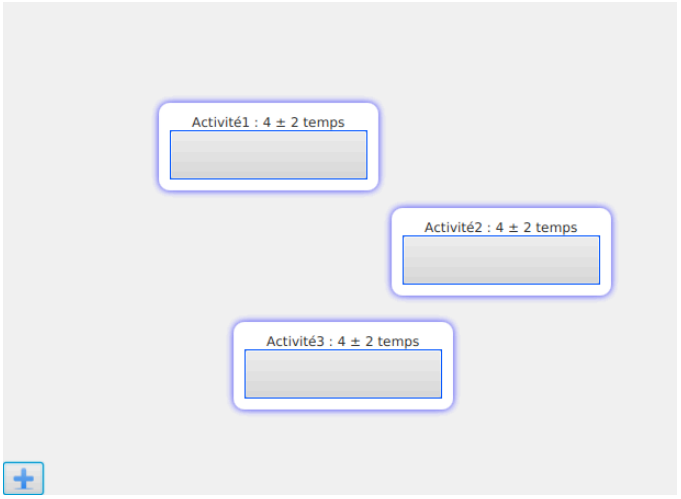
Interfaces Graphiques
TPs 4, 5, 6 et 7 - Architecture MVC - Twisk

L'objectif des quatre prochaines séances de TPs est la réalisation d'un éditeur de monde pour **twisk**. Il y aura sur Arche un sujet de TP à chaque séance.
Cet éditeur sera intégré à **twisk** ultérieurement. Voici un exemple de monde que l'on veut pouvoir créer : des activités (en bleu) et des arcs entre ces activités (en rose). Les arcs ne peuvent relier que deux points de contrôle (en rose).



La conception de cet éditeur va être réalisée pas à pas, selon un mode de développement itératif, qui consiste à compléter les fonctionnalités au fur et à mesure. Pour commencer, on va se limiter à un éditeur capable d'afficher des activités, comme le montre la capture d'écran ci-dessous.



Créez un nouveau projet **Java** de nom **twiskIG**. Ce nouveau projet sera versionné sur la forge, un nouveau repository personnel a été créé.

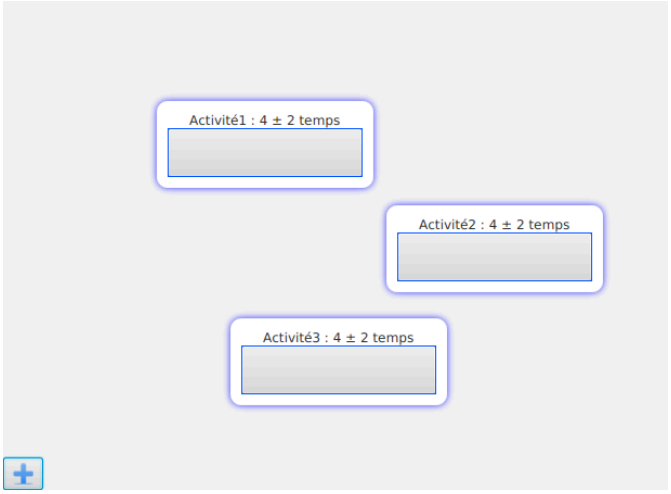
Travaillez avec rigueur et méthode, toutes les classes feront ultérieurement partie intégrante du projet **twisk**. On rappelle ici que ce projet est **individuel** contrairement au projet **twisk** qui peut être fait en binôme.

TP4

L'objectif de ce TP se limite à la création d'un éditeur permettant l'affichage d'un monde composé uniquement d'activités, avec un bouton permettant d'ajouter une activité, placée de façon aléatoire. L'interface graphique se résume à deux parties : un dessin du monde et un bouton.

L'architecture MVC doit être mise en place, conformément au diagramme de classes consultable à cette adresse.

À toutes ces classes, s'ajoute la classe principale **twisk.MainTwisk**.



1- Définir les classes gérant le modèle

La classe **EtapeIG** est une classe du modèle qui décrit une étape graphique : elle est définie par un nom (fixé par défaut et ultérieurement modifiable par l'utilisateur), un identifiant unique attribué une fois pour toutes par **twisk**, une position sur le dessin (fixé aléatoirement à la construction et ultérieurement modifiable par l'utilisateur), une largeur et une hauteur (en pixels). Pour mémoire, l'origine du repère est en haut à gauche de l'écran.

On se contente ici d'une sous-classe **ActiviteIG** qui décrit une activité, avec les mêmes caractéristiques.

La classe **FabriqueIdentifiant** est responsable de la fabrication d'identifiants uniques (peu importe comment ils sont construits) ; la classe est gérée en singleton.

La classe principale du modèle est la classe **MondeIG**. Elle gère une collection d'**EtapeIG**, rangées dans une **HashMap** dont la clé est l'identifiant de l'étape. À sa construction, le monde contient une activité. La fonction **ajouter** permet d'ajouter une étape définie par le type passé en paramètre (ici "*Activité*") ; un nom par défaut est attribué à cette activité. Le nom par défaut est laissé à votre convenance, sous réserve que chaque nom soit différent des précédents.

La classe **MondeIG** est itérable : son itérateur permet de consulter toutes les étapes mémorisées.

Écrivez et testez avec JUnit les classes **EtapeIG, ActiviteIG, MondeIG, FabriqueIdentifiant.**

2- Définir VueOutils et MainTwisk

La classe **VueOutils** se résume pour l'instant au bouton permettant d'ajouter une activité. L'écouteur du bouton fait simplement appel à la fonction **ajouter** de **MondeIG**. Si vous mettez une image sur ce bouton, pensez à l'inclure dans le répertoire **twisk/ressources/images.**

La classe principale **MainTwisk** crée le modèle, instance de **MondeIG**, place la vue **VueOutils** au sud de la fenêtre.

Écrivez les classes **MainTwisk** et **VueOutils**. Pour vérifier l'effet d'un clic de bouton, ajouter (momentanément) une trace sur la sortie standard dans la fonction **ajouter** de **MondeIG**.

Ajoutez un tooltip (instance de **Tooltip**) au bouton qui permet d'ajouter une activité.

3- Définir VueMondeIG, VueEtapeIG et VueActiviteIG

La classe **VueMondeIG** affiche toutes les activités répertoriées dans **MondeIG**. À chaque mise à jour dans la fonction **reagir**, le dessin est complètement refait, après suppression des anciens composants (**getChildren().clear()**).

La classe **VueEtapeIG** n'a pour l'instant qu'une sous-classe **VueActiviteIG** qui décrit le composant graphique d'une activité. Le label du titre est factorisé dans **VueEtapeIG**.

Une activité est représentée par une instance de **VueActiviteIG** ; c'est une boîte avec un titre (**Label**) et une zone destinée à contenir les clients (**HBox**). Le composant est positionné par l'intermédiaire de la fonction **relocate**.
Écrivez les classes **VueMondeIG, VueEtapeIG** et **VueActivite**.

Idée : vous pouvez utiliser le code ci-dessous pour ajouter un style de présentation à la zone destinée à contenir les clients :

setStyle("-fx-border-color: #0059FF; -fx-background-insets: 0 0 -1 0, 0, 1, 2; -fx-background-radius: 3px, 3px, 2px, 1px;")
;

Vérifiez que l'application est opérationnelle, en ajoutant plusieurs activités.

4- Centraliser la définition des tailles

En l'état, la taille d'un composant de type Activité est codée en dur dans la classe **ActiviteIG**. En terme d'évolutivité, ce n'est guère pratique, surtout si on continue sur cette lancée pour les composants à venir. Il est préférable de centraliser des constantes dans une seule et même classe de sorte qu'une modification soit plus facilement envisageable.

La classe **TailleComposants** est la seule responsable de la taille des différents composants ; la classe est gérée en singleton.

Ajoutez cette nouvelle classe et modifiez le code existant en conséquence.

5- Ajouter du style

Pour finir, il est temps de s'intéresser au look de l'application. Rien n'est imposé, vous pouvez laisser libre court à votre imagination dans le choix des formes, des polices et des couleurs.
Vous pouvez continuer à utiliser la fonction **setStyle**, mais il est plus pratique d'associer du ***css*** aux composants.

Vous pouvez vous aider de ce tutoriel, qui explique comment ajouter du ***css*** à un bouton. Ces mêmes explications s'appliquent à tous les autres composants.

http://tutorials.jenkov.com/javafx/button.html#button-css-styles