Shapes - Un tutoriel d'introduction à MPS

Par Aleksey Makarkin - Oscar Rodriguez

Date de publication : 21 mars 2020

Si vous découvrez MPS et souhaitez l'essayer rapidement, ce tutoriel est fait pour vous.

En moins de deux heures, vous obtiendrez un nouveau langage et du code opérationnel basé sur ce langage. Dans ce tutoriel, vous partirez de zéro et suivrez un parcours à la fois sûr, pratique et rapide, afin de parvenir à concevoir les éléments essentiels d'un nouveau langage.

Par souci d'efficacité et de rapidité, nous allons éviter les concepts avancés et constructions compliquées. Finalement, vous saurez à quoi sert MPS et connaîtrez ses principes de base.

Pour réagir au contenu de ce tutoriel, un espace de dialogue vous est proposé sur le forum.
Commentez

I - Conditions préalables....3
II - Objectif....3
III - Créer un projet....4
IV - Langages et programmes sous le capot....6
V - Forme graphique....7
VI - Circle....10
VII - Carré....13
VIII - Toile....13
IX - Une phase de test initiale....15
X - Éditeurs....19
X-A - Shape....19
X-B - Circle....19
X-C - Carré....21
X-D - Toile....21
XI - Deuxième exécution....23
XII - Application de couleurs....24
XII-A - Concept pour les couleurs....24
XII-B - Couleurs prédéfinies....25
XII-C - Première touche sur les dépendances....27
XII-D - Concept de référence de couleur....28
XII-E - Mise à jour d'une forme....30
XIII - Intégration du composant d'éditeur....31
XIV - Une troisième exécution....32
XV - Générateur....33
XV-A - Paramétrage du modèle....41
XV-B - Dessiner des formes....43
XV-C - Génération de cercles....46
XV-D - Paramétrage....50
XV-E - Réduction des carrés....55
XVI - Une génération plus robuste des carrés....56
XVII - Génération de code....58
XVIII - Exécution du code....59
XIX - Que faire ensuite....62

Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/

I - Conditions préalables

Dans ce tutoriel, nous partons du principe que vous avez déjà pris connaissance du : **tutoriel Fast Track to MPS** et que vous vous êtes familiarisé avec l'environnement MPS, que vous comprenez le concept de langage et de solution, et que vous êtes en mesure de commander l'éditeur projectionnel MPS. Si cela n'est pas le cas, nous vous invitons à consacrer environ 30 minutes à cette phase préliminaire. Plus précisément, *les raccourcis clavier* seront essentiels à votre survie :

- *Ctrl* + *Espace* : pour compléter un mot ou transformer un identificateur non valide (rouge) en identificateur correct (noir) ;
- *Alt* + *Entrée* : pour afficher un menu contextuel regroupant des options pratiques et pouvant être appliquées à la position actuelle de l'éditeur ;
- *Ctrl/Cmde* + *Flèche haut/Flèche bas* : développer/réduire la zone de texte sélectionnée ;
- *Tabulation* : parcourir les éléments modifiables de l'éditeur ;
- *Ctrl/Cmde* + *Z* : annuler l'action précédente.

Nous supposons que vous avez installé MPS et qu'il est en cours d'exécution devant vous. Bon, maintenant nous sommes prêts.

Vous pouvez également regarder ce tutoriel en ligne sous forme de screen-cast. Essayez.

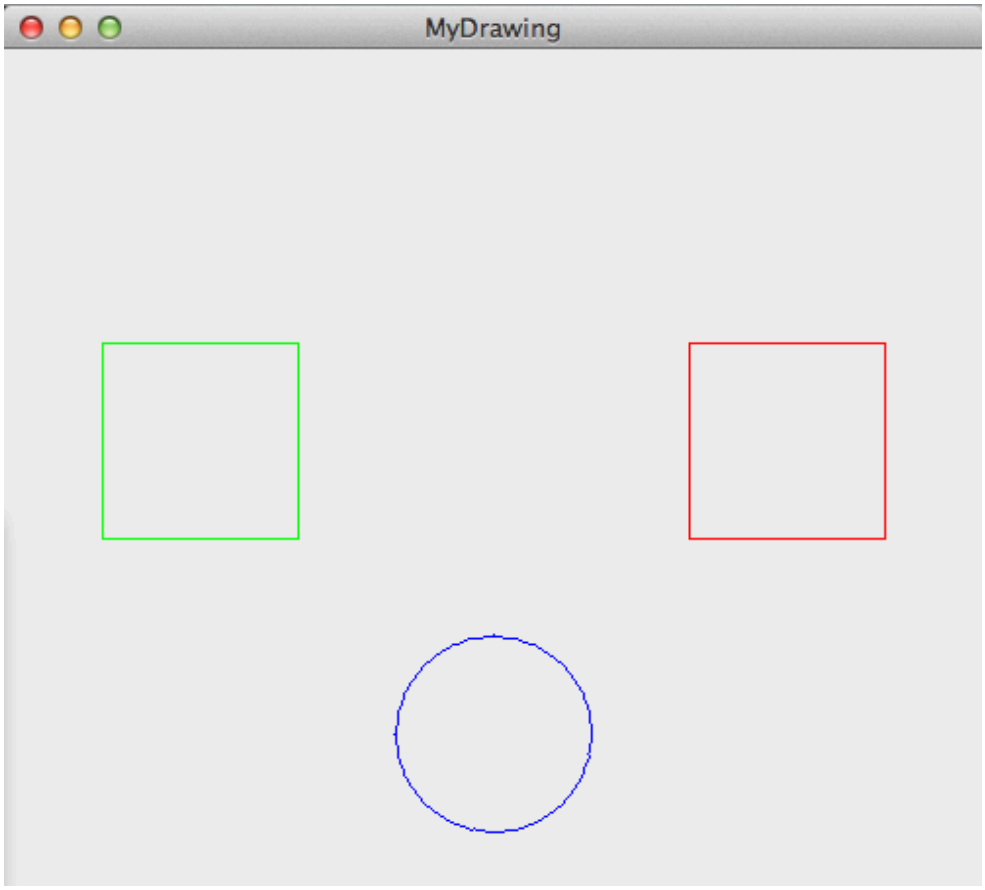
Cliquer sur ce lien pour lancer l'animation

II - Objectif

Vous allez implémenter un exemple de langage dédié à la spécification de formes graphiques. Ce langage permettra à ses utilisateurs de créer visuellement des formes bidimensionnelles sur un fond plat. Leur définition sera ensuite convertie en application Java Swing, qui permettra de visualiser la mise en forme à l'écran.

Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/

```
Painting MyDrawing
circle x: 200 y: 300 radius: 100 color: BLUE
square x: 50 y: 150 size: 100 color: GREEN
square x: 350 y: 150 size: 100 color: RED
```

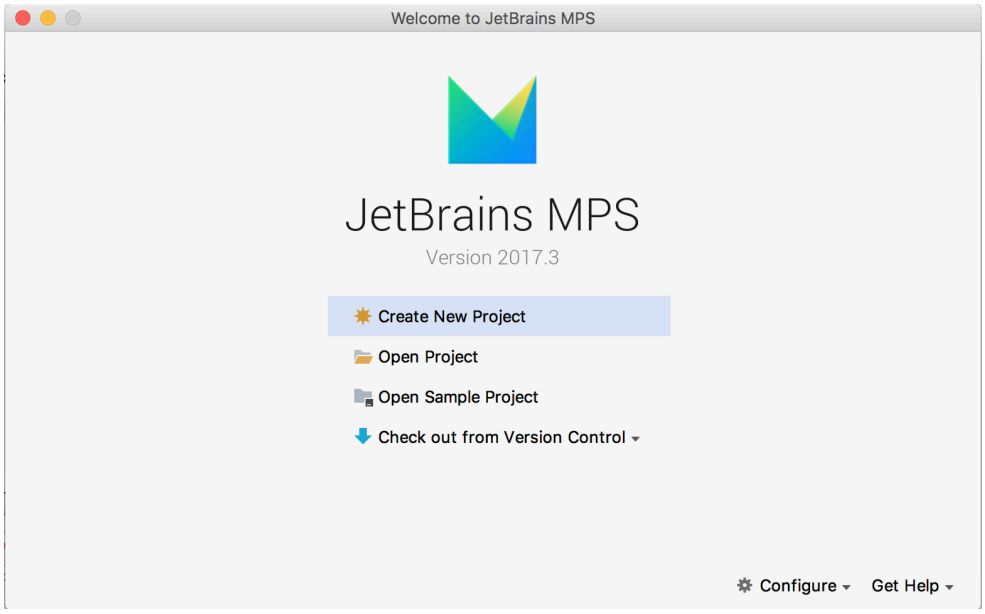


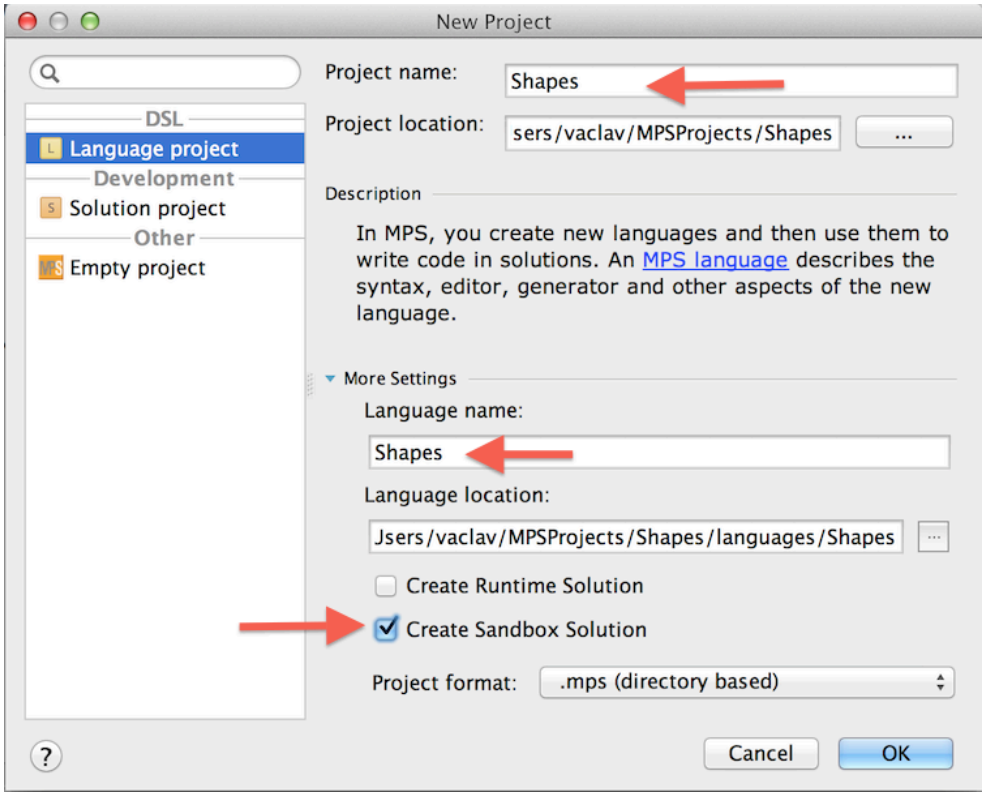
Ce langage pourrait permettre aux néophytes de créer des applications Java sans connaissance préalable de Java, Swing ou de l'API 2D Graphics. Vous allez jouer le rôle de concepteur de langage en vous appuyant sur vos connaissances de Java pour le rendre aussi accessible et simple à utiliser que possible. Vous allez automatiser le travail d'un programmeur d'interfaces en fournissant un langage et un générateur couvrant la plupart des cas qu'un programmeur d'interfaces doit résoudre manuellement.
Si vous ne maîtrisez pas Java ou Swing, ne vous inquiétez pas trop ni trop vite. En effet, nous en avons tenu compte et conçu ce tutoriel pour vous guider de façon précise. Vous pourrez ainsi aller jusqu'au bout facilement, comme vous allez le voir.

III - Créer un projet

Nous allons commencer par un tout nouveau projet. Sur l'écran d'accueil, cliquez sur *Create New Project*, puis laissez-vous guider par l'assistant.

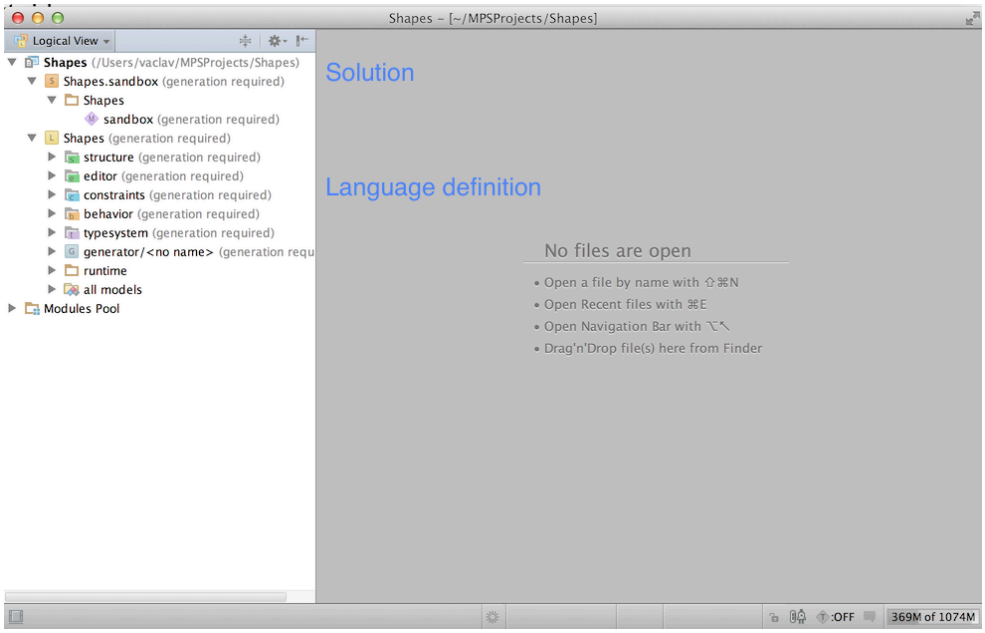
Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/





Vous allez ainsi créer un projet vide dans lequel les sections *Language Definition* et *Solution* sont également vides.

Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/



La section *Solutions* sert à enregistrer vos programmes. Pour les écrire, vous utilisez des langages, qu'ils soient définis au sein du même projet ou importés. Dans notre tutoriel nous allons commencer par définir un langage et l'utiliser pour écrire du code exécutable.

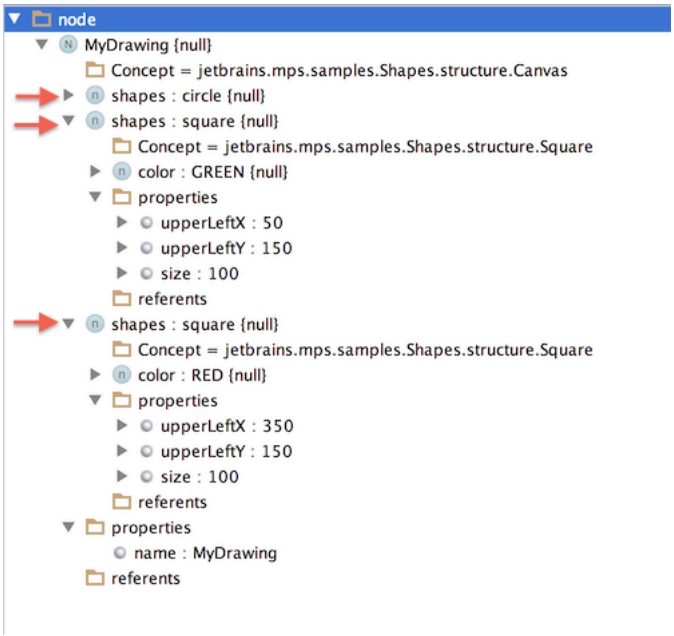
IV - Langages et programmes sous le capot

Tout d'abord, voici quelques connaissances de base à prendre en compte avant de continuer.

```
Painting MyDrawing
circle x: 200 y: 300 radius: 100 color: BLUE
square x: 50 y: 150 size: 100 color: GREEN
square x: 350 y: 150 size: 100 color: RED
```

Le langage que nous allons créer doit rendre possibles les définitions de dessin, sous forme de commandes individuelles, chacune sur une ligne distincte et définissant une forme spécifique à dessiner. Notre langage doit couvrir chacune de ces commandes avec un *Concept*. Les *concepts* définissent la syntaxe abstraite d'un langage, à savoir l'ensemble des constructions logiques admissibles du langage. Le programme consiste ensuite en une série d'*arbres de syntaxe abstraits* (Abstract Syntax Trees - Ast) qui regroupent les instances de ces *concepts*.

Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/



L'AST du programme ci-dessus présente la syntaxe abstraite qui regroupe plusieurs *nœuds* et chacun de ces *nœuds* est une instance d'un *concept* défini par le langage. Les *concepts* définissent des propriétés, des enfants et des références, tandis que les *nœuds* apportent les valeurs concrètes.

Première petite synthèse :

- les *langages* reposent sur des *concepts* ;
- les *concepts* définissent les éléments logiques (abstraits) avec leurs *propriétés*, *enfants* et *références* ;
- les *programmes (solutions)* reposent sur des AST, qui consistent en des *nœuds* ;
- les *nœuds* sont des instances de *concepts* conférant des valeurs concrètes aux *propriétés*, aux *enfants* et aux *références* de leurs *concepts.*

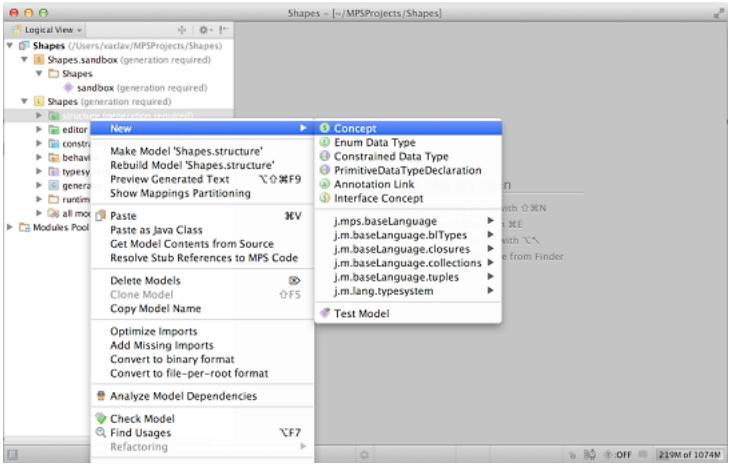
V - Forme graphique

Notre langage va être assez simple. Nous n'avons besoin que de quelques concepts :

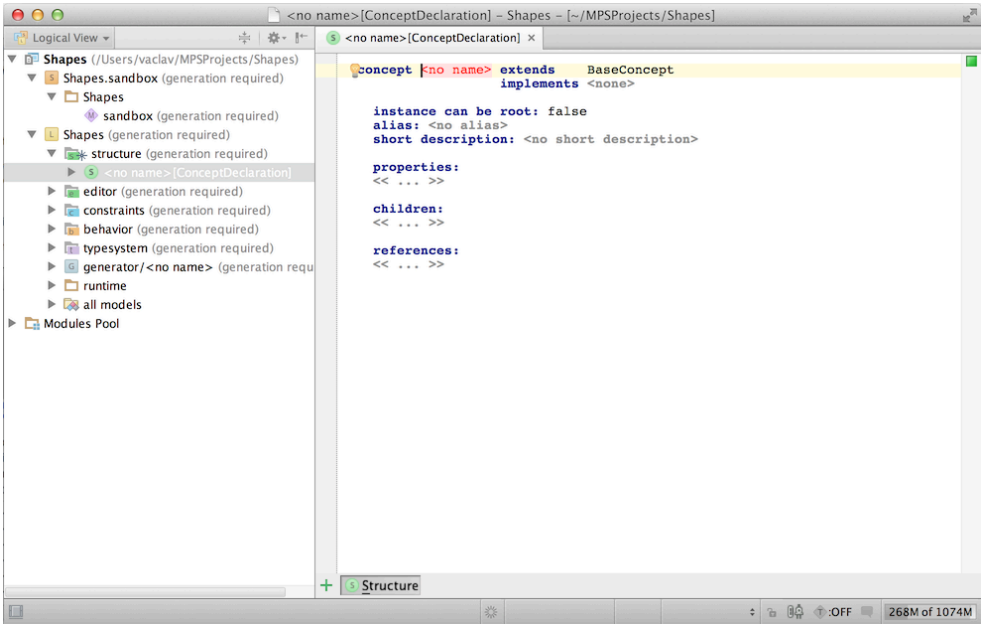
- *Canvas* : pour définir le nœud de plus haut niveau représentant l'intégralité de la **définition de dessin** et regroupant toutes les formes (*Shapes*);
- *Shape* : représente une commande permettant de dessiner une forme sur le fond (*Canvas*), qui servira de *super-concept* commun à l'ensemble des formes concrètes, telles qu'un cercle ou un carré;
- *Circle* : représente une commande permettant de dessiner un cercle;
- *Square* : représente une commande permettant de dessiner un carré;
- *ColorReference* : représente l'une des couleurs prédéfinies dans *java.awt.Color.*

Nous allons commencer notre exercice pratique par le concept *Shape*. Tout comme dans la programmation orientée objet, les *concepts* peuvent s'étendre les uns les autres et par conséquent hériter des fonctionnalités de leur *super-concept*. Le concept *Shape* sert précisément de *super-concept*, contenant la propriété *color*, car toutes les formes de notre langage auront besoin de la propriété *color* et il est donc intéressant de pouvoir en hériter.

Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/

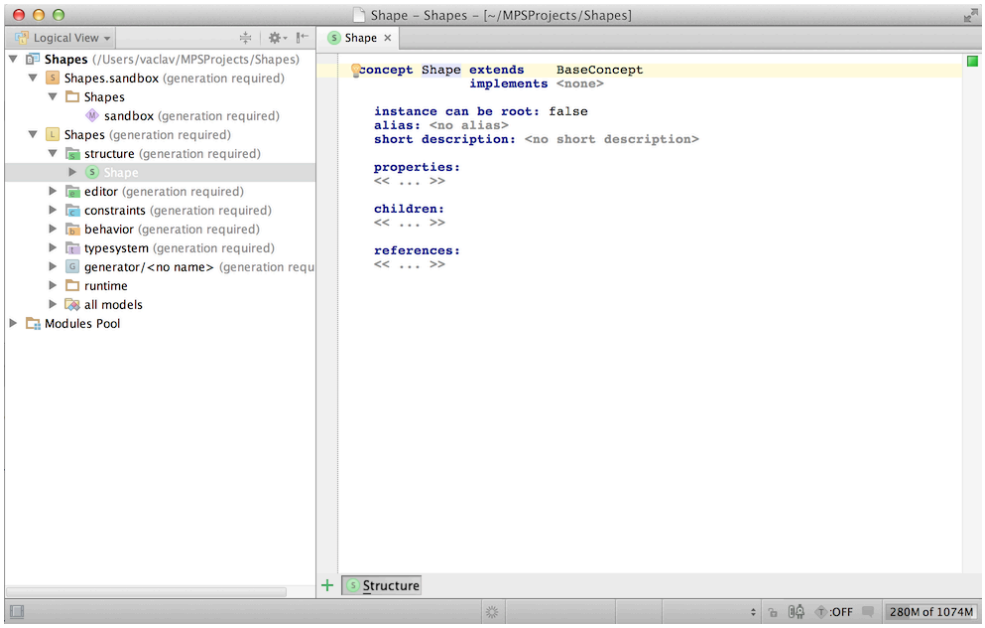


Faites un clic droit sur l'aspect *Structure* du langage et créez un *concept*. Une nouvelle définition de *concept* s'ouvre dans l'éditeur.



Nous devons donner au *concept* un nom descriptif, *Shape* dans le cadre de ce tutoriel, fera l'affaire.

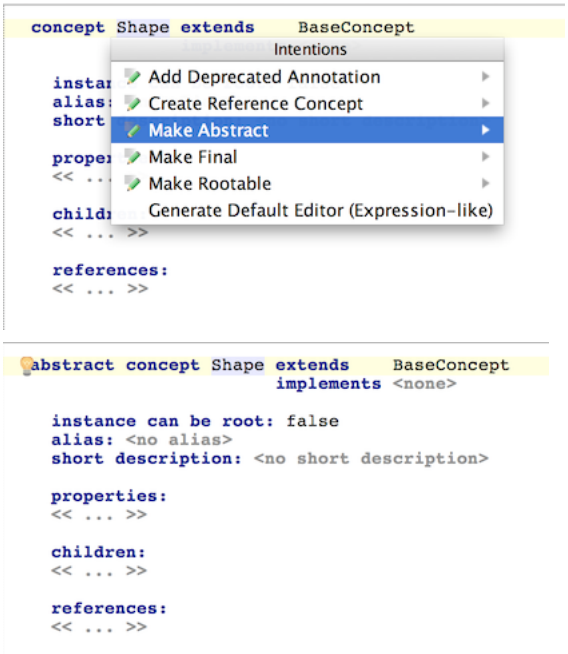
Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/



Nous avons créé le concept *Shape* en tant que *super-concept* commun à l'ensemble des formes de notre langage et en tant que tel, le concept *Shape* ne sera pas utilisé directement dans les AST. Nous allons marquer *Shape* comme **abstrait** pour indiquer de façon explicite qu'aucune instance (*nœud*) de *Shape* ne peut être créée.

Si vous faites une faute de frappe sur le nom du concept et décidez de le changer, MPS vous guidera, qu'il s'agisse d'un changement de nom ou d'une refactorisation, afin de mettre à jour toutes les références et utilisations possibles du concept dans d'autres parties de votre code. Dans la mesure où nous venons de commencer et où nous n'avons pas encore de références à *Shape*, appuyez simplement sur le bouton « No ».

Il est maintenant temps de s'entraîner à utiliser le raccourci clavier *Alt* + *Entrée*.



Une fois positionné sur le nom du *concept*, le raccourci clavier *Alt* + *Entrée* affiche un menu contextuel permettant d'appliquer l'intention « Make Abstract » au *concept*. Si vous choisissez cette option, le concept est marqué comme **abstrait**. Nous pouvons désormais ajouter des propriétés, des enfants et des références qui doivent être partagés

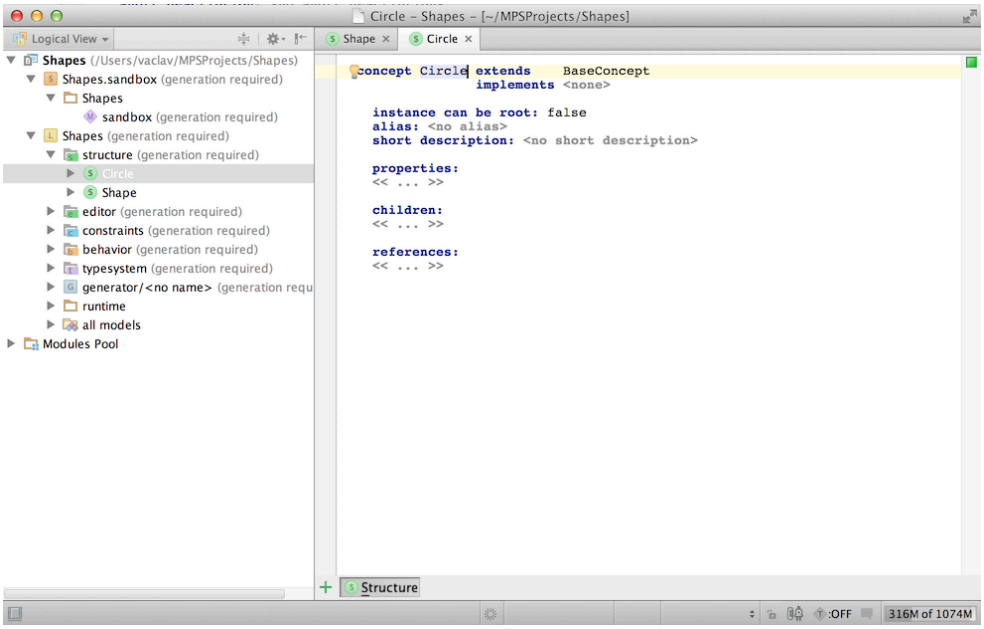
Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/

avec tous les sous-concepts de *Shape*, mais nous allons garder cela pour plus tard, afin de ne pas altérer notre logique d'apprentissage.

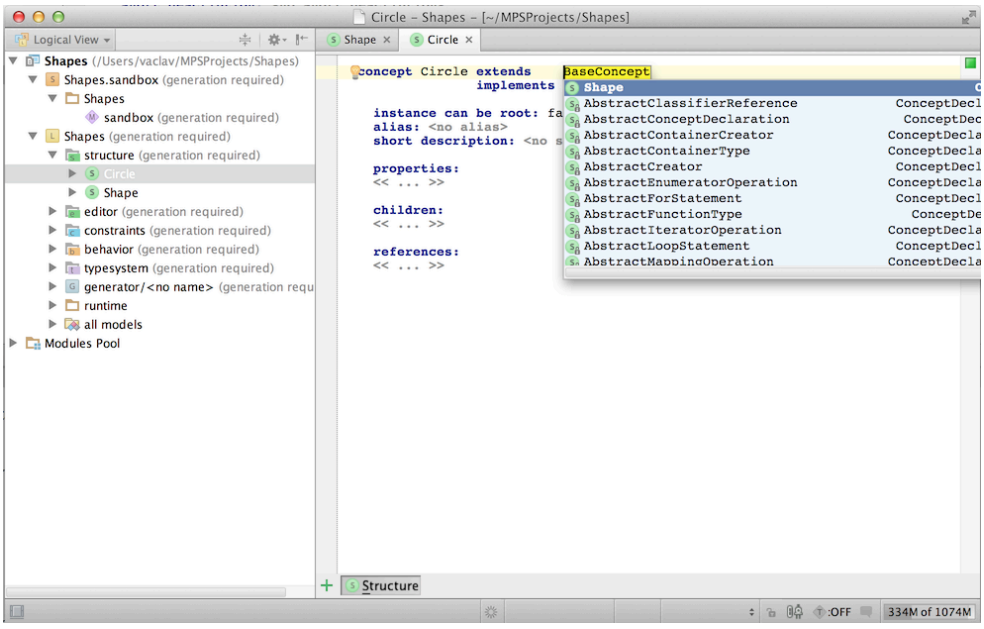
Notre premier concept est fait ! Hourra ! Il est temps d'en ajouter un autre. Pourquoi pas *Circle* ? En faisant un clic droit sur l'aspect *Structure,* nous créons un autre*Concept* et pouvons l'appeler *Circle*.

VI - Circle

En suivant les mêmes étapes, vous allez faire un clic droit sur l'aspect *Structure* du langage et ajouter un nouveau concept. Nommez-le *Circle*.

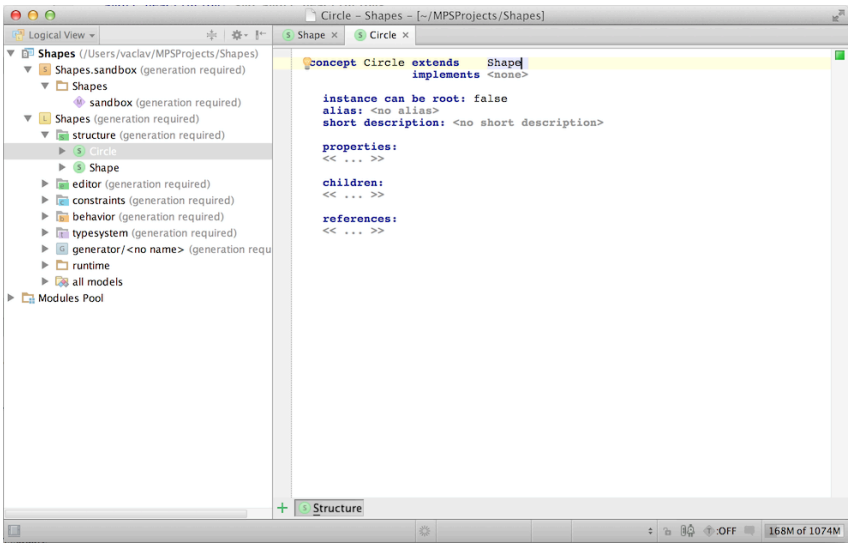


Les cercles doivent hériter des fonctionnalités de *Shape* et nous devons par conséquent le préciser dans la clause **extends**. Positionnez le curseur au début de la cellule portant le texte « BaseConcept » et appuyez sur le raccourci clavier le plus utile de MPS - *Ctrl* + *Espace*, afin de lancer la saisie automatique.

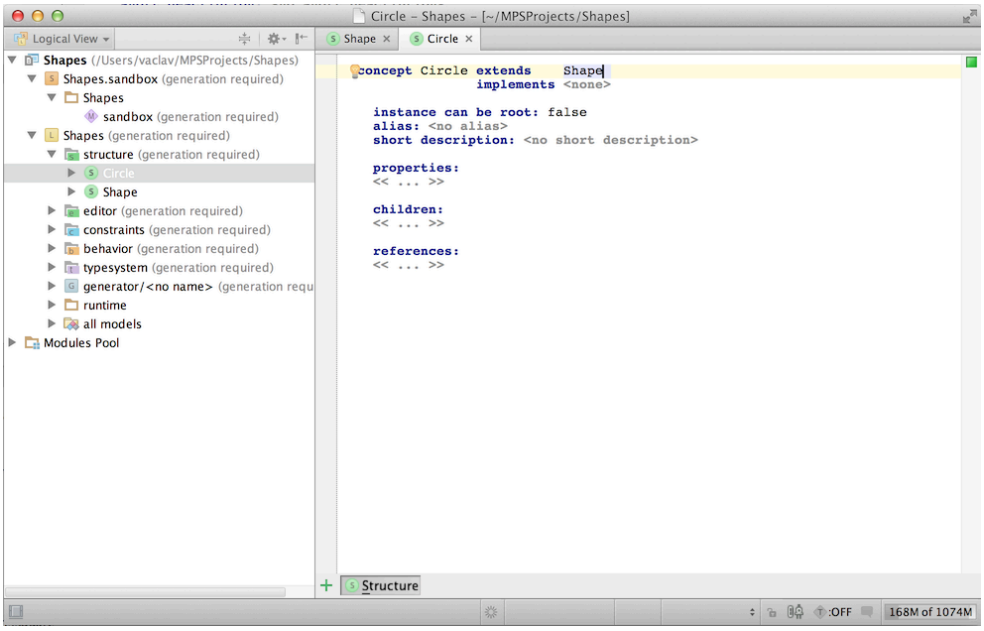


Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/

MPS présente la liste des options pouvant servir de remplacement au texte « BaseConcept ». Nous devons sélectionner *Shape* ici pour que *Circle* étende *Sh*



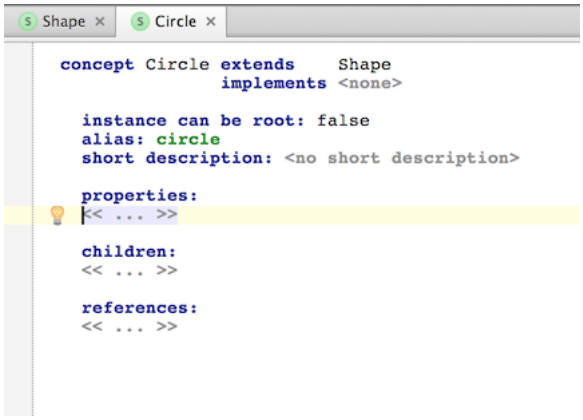
Il s'agit ici de notre premier concept concret qui sera utile aux utilisateurs de notre langage. Pour associer au concept un texte élégant dans les boîtes de dialogue à saisie automatique et permettre à MPS de créer intelligemment une instance de *Circle* dès que l'utilisateur tape « circle », vous devez donner à ce concept l'alias « circle ».

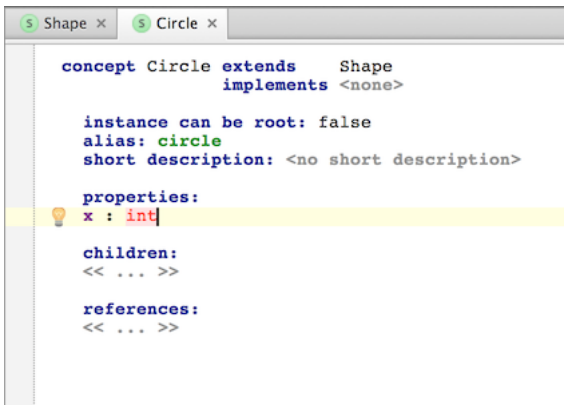


*Remarque : la touche de **tabulation** permet ici de parcourir les éléments modifiables de l'éditeur. Utilisez la touche de **tabulation** fréquemment.*

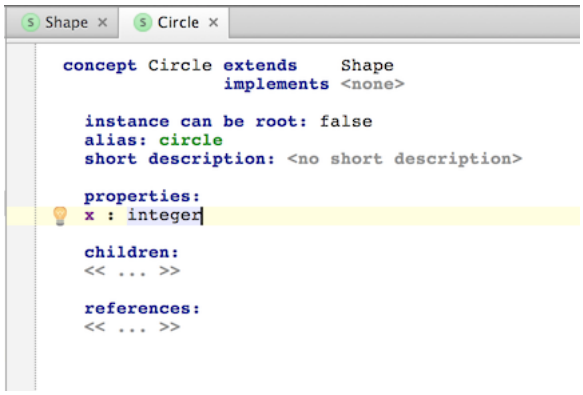
Chaque cercle doit spécifier ses coordonnées à l'écran et son rayon. Nous allons créer des propriétés contenant les valeurs de ces entiers. Accédez à la section **properties**, placez le curseur sur le symbole « << … >> » qui représente une collection vide de valeurs et appuyez sur *Entrée*. Ce processus crée une propriété vide.

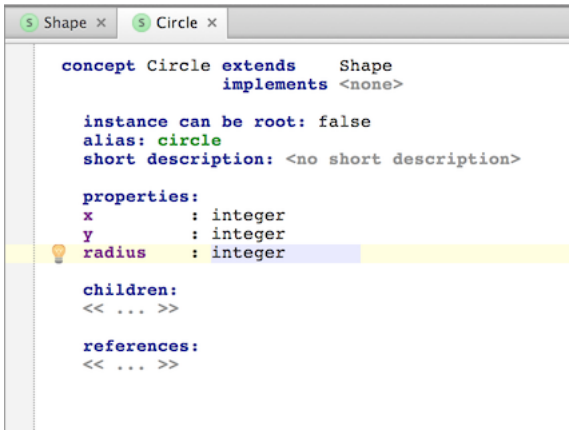
Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/





Appelez cette propriété « x », puis appuyez sur la touche de **tabulation** et indiquez le type de la propriété, à savoir « integer ». Pendant que vous saisissez « integer », vous pouvez utiliser le raccourci clavier *Ctrl* + *Espace* pour nommer le type complété par MPS.



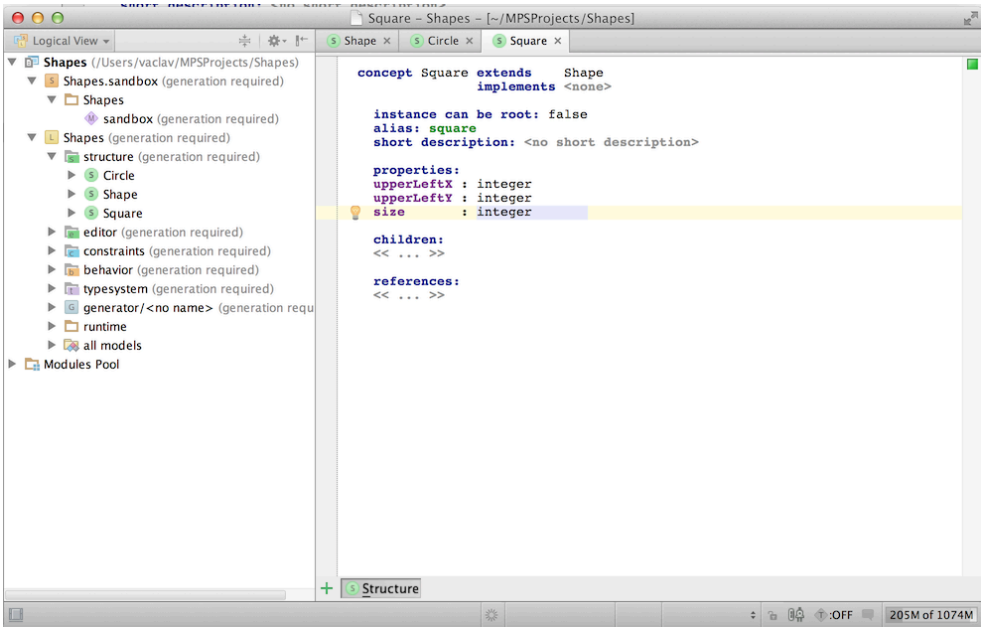


Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/

Créez ensuite les propriétés pour « Y » et « radius » et vous aurez terminé votre premier concept concret.

VII - Carré

Maintenant vous devriez vous lancer et créer un concept pour le carré. Répétez simplement les étapes réalisées pour *Circle* en créant des propriétés différentes : *upperLeftX*et *upperLeftY* pour les coordonnées du coin supérieur gauche et *size* pour spécifier la longueur des côtés du carré. À la fin de ce processus, vous devez obtenir ce qui suit :

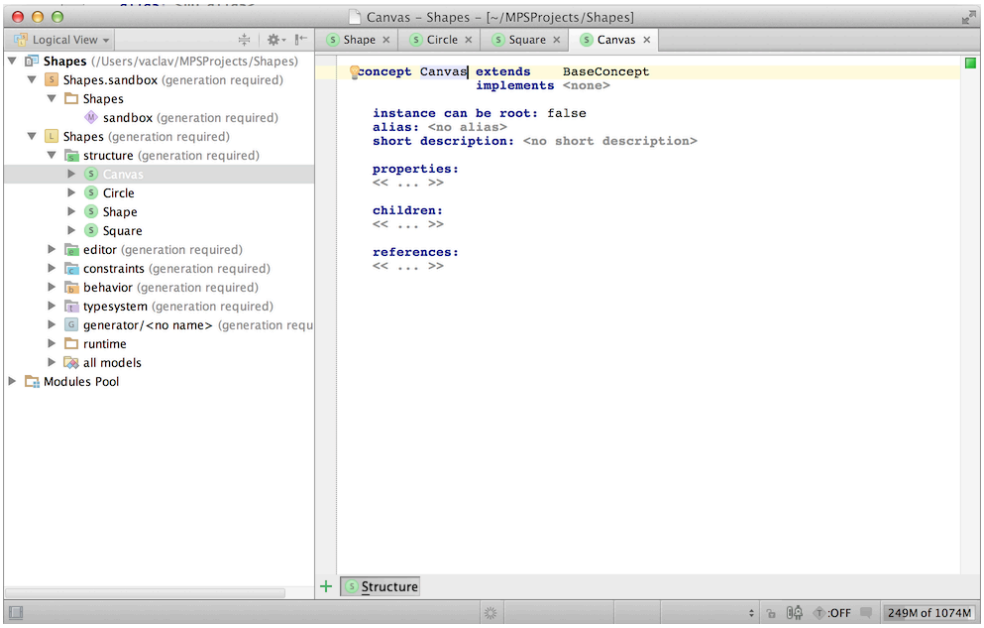


VIII - Toile

Après avoir créé deux formes, nous pouvons désormais définir un concept pour réaliser un dessin. Nous allons créer un autre concept, appelé *Canvas*, pour représenter une scène composée de formes. L'utilisateur peut créer plusieurs scènes (concepts *Canvas*) qui sont indépendantes les unes des autres et ne partagent pas de formes. Chaque *Canvas* contient un nom et la liste des formes qu'il inclut.

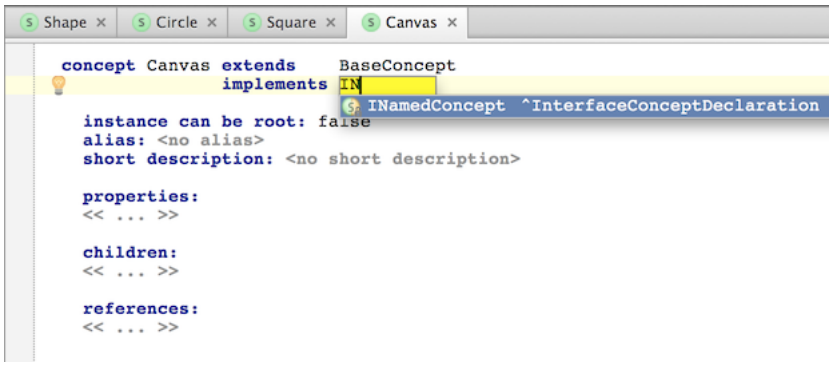
Encore une fois, cliquez sur l'aspect *Structure* du langage pour créer un concept et appelez-le *Canvas*.

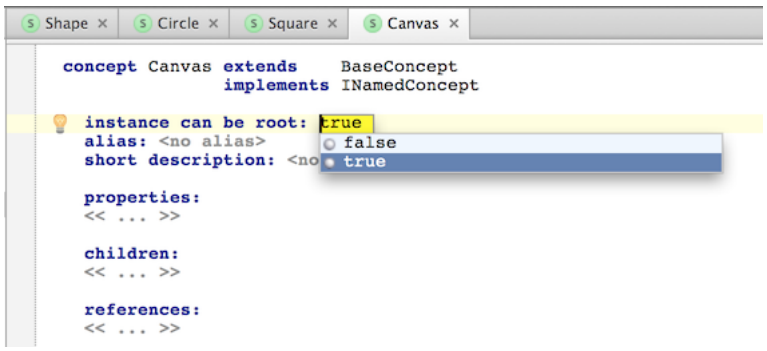
Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/



Remarque : contrairement à Circle ou Square, Canvas ***n'étend pas*** *Shape, mais BaseConcept. Canvas n'est pas supposé être dessiné ou placé sur d'autres éléments Canvas, par conséquent il ne doit pas étendre Shape.*

Accédez (avec la touche de **tabulation**) au texte « <none> » dans la section **implements** et spécifiez *INamedConcept*, ainsi que le nom de l'*interface de concept* que *Canvas* va implémenter. Les *interfaces de concept* permettent également d'ajouter de nouvelles fonctionnalités à *Concepts* qui les implémente. *INamedConcept*, dans notre cas, enrichit *Canvas* avec la propriété *name*, de sorte que les instances de *Canvas* (appelées *Nodes*) auront une propriété *name* permettant à l'utilisateur de les reconnaître facilement.

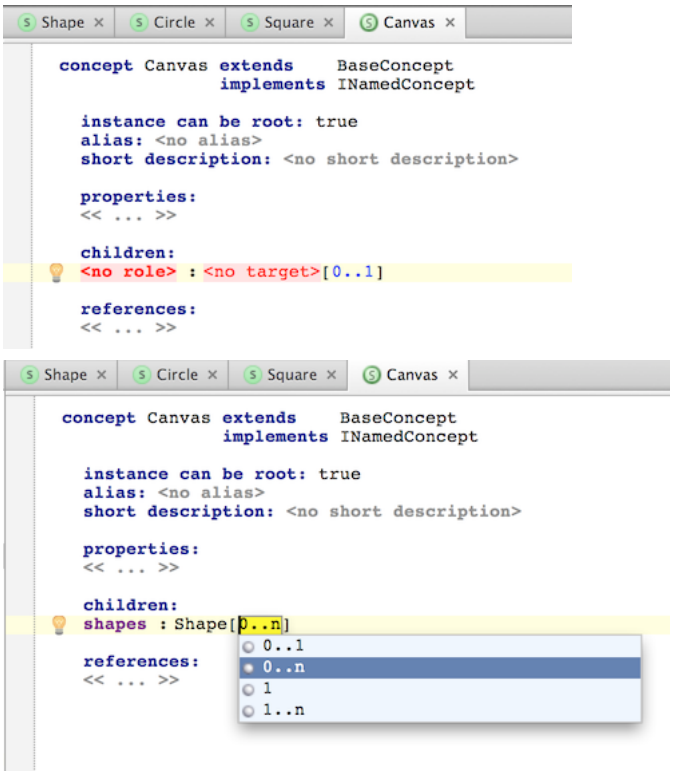




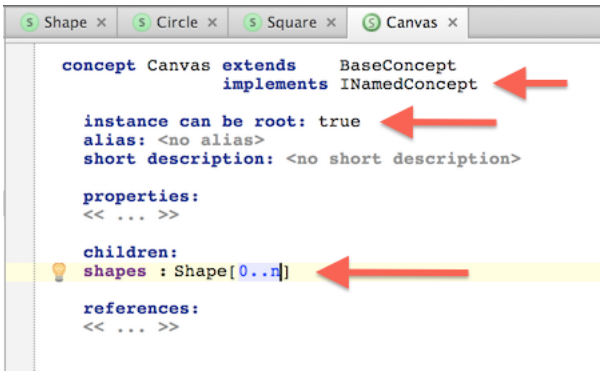
Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/

Comme *Canvas* représente le lieu du dessin et ne fait partie en lui-même d'aucun autre concept, nous lui donnons l'attribut **instances can be root**. Cela permet aux instances de *Canvas* de devenir des racines des AST.

Pour indiquer que les éléments *Canvas* peuvent contenir des éléments *Shape*, nous allons créer une collection enfant de **formes**. Encore une fois, appuyez sur la touche *Entrée* de la cellule « << … >> » de la section **children**, tapez le nom de l'enfant, ainsi que le type de nœuds qu'il peut contenir. N'oubliez pas le raccourci clavier * **Ctrl* + *Espace* pour afficher la boîte de dialogue de saisie automatique.



Vous devez obtenir une définition de concepts du type suivant :



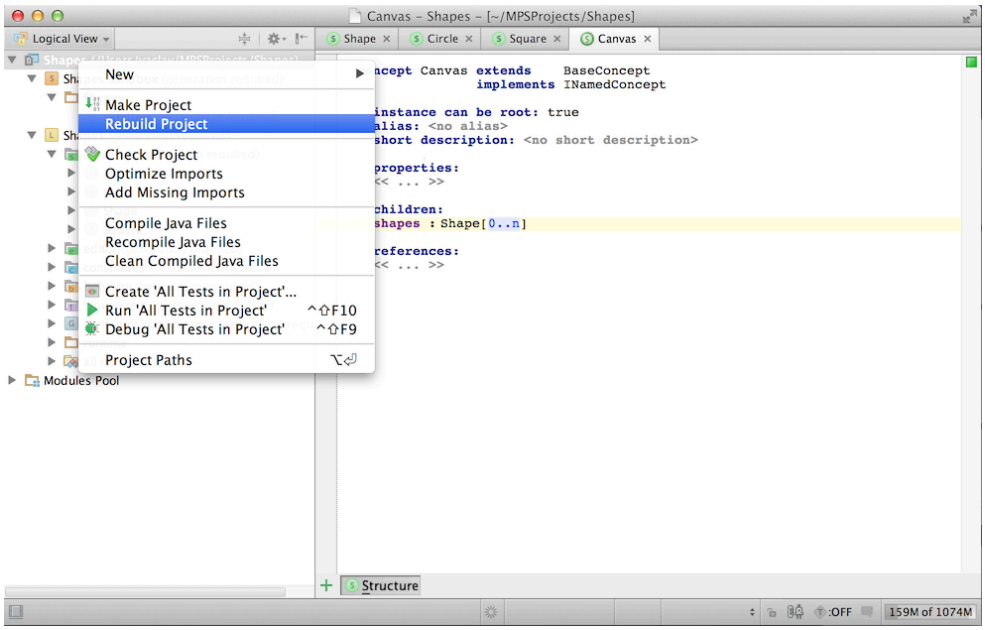
Désormais vous avez créé suffisamment de concepts pour obtenir un embryon de langage, prêt à l'emploi. Essayons-les.

IX - Une phase de test initiale

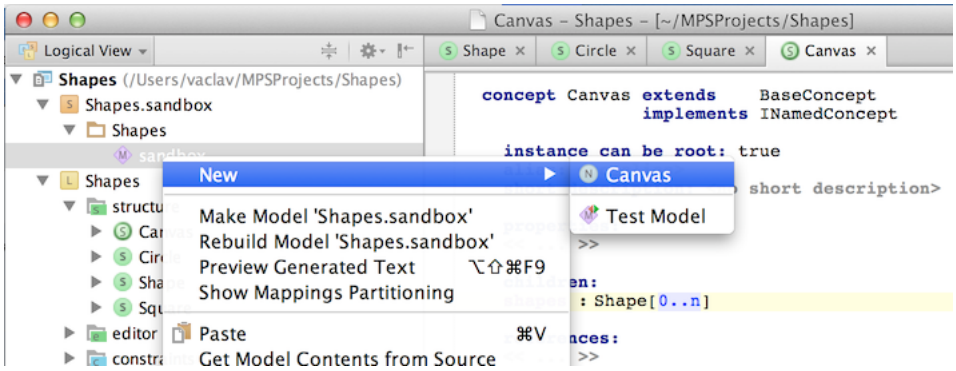
Nous devons construire le langage avant de pouvoir l'utiliser. À l'avenir, lorsque vous modifierez l'une des définitions du langage, n'oubliez pas de répéter le processus « Rebuild », de sorte que la modification puisse prendre effet.

Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/

Faites un clic droit sur le nœud de plus haut niveau (la racine ultime et représentant l'ensemble du projet, **ce qui inclut le langage et la solution**) dans la *vue de projet* et choisissez **Rebuild Project**.

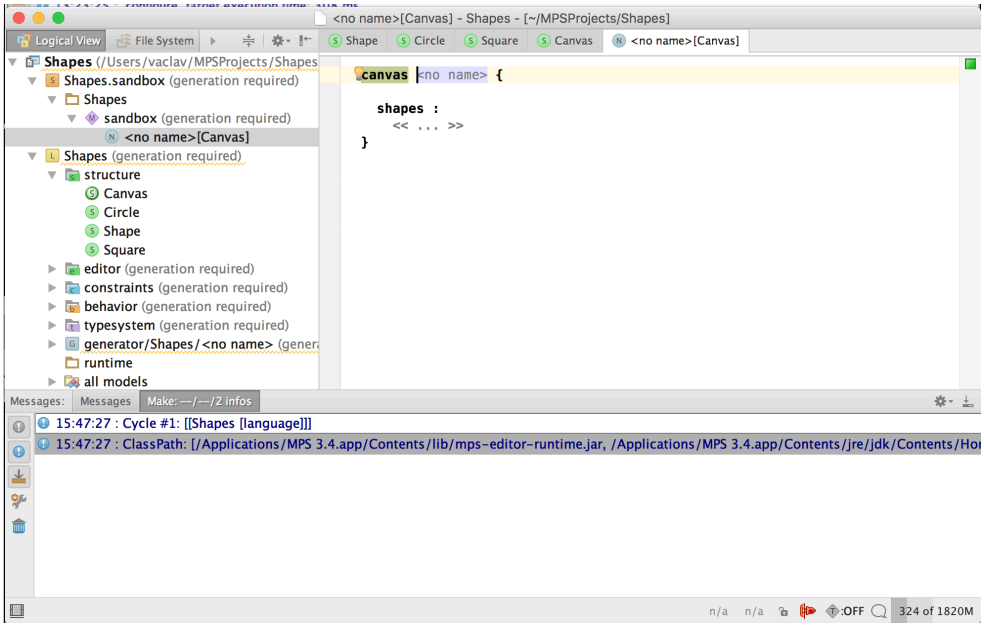


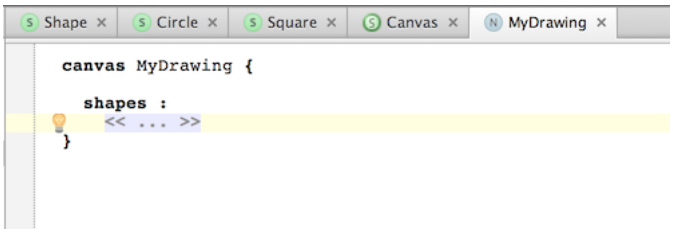
Une fois la build de langage créée, le langage sera exploitable dans le modèle, dans les limites du sandbox *Solution*. Créez simplement une nouvelle instance de *Canvas* en faisant un clic droit sur le modèle et en choisissant le concept racine à instancier. Remarque : les concepts racine s'affichent dans ce menu et peuvent être instanciés au niveau supérieur d'un modèle.



À présent, nommez l'instance *Canvas* pour obtenir votre premier dessin. Vous pouvez commencer à utiliser les concepts définis dans le langage pour placer des formes visuelles sur le fond.

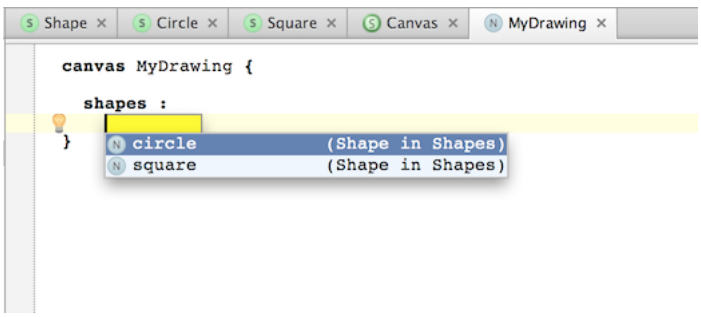
Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/



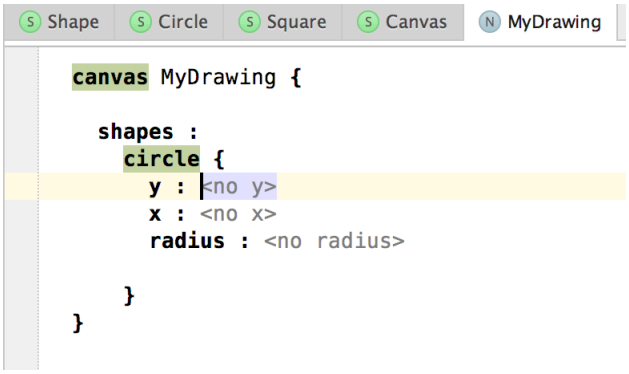


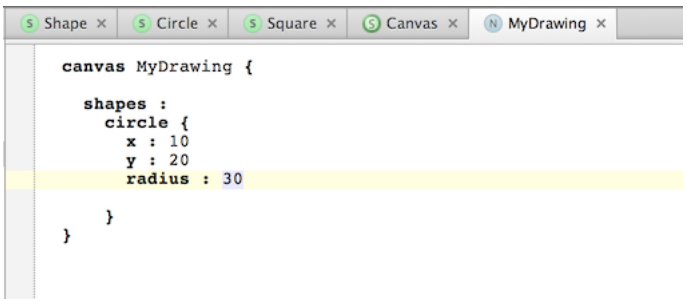
L'expérience d'édition de notre langage ainsi que sa mise en forme appliquent encore à ce stade des valeurs par défaut que nous allons changer bientôt pour personnaliser un peu plus la présentation du code.

Encore une fois, le raccourci clavier *Ctrl + Espace* va être très utile ici. Utilisez-le à chaque fois que vous avez une hésitation. Remarque : La boîte de dialogue de saisie automatique propose les deux types de formes que nous avons créées dans le langage.

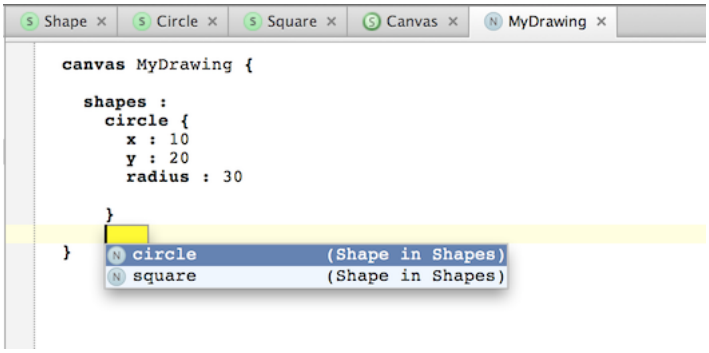


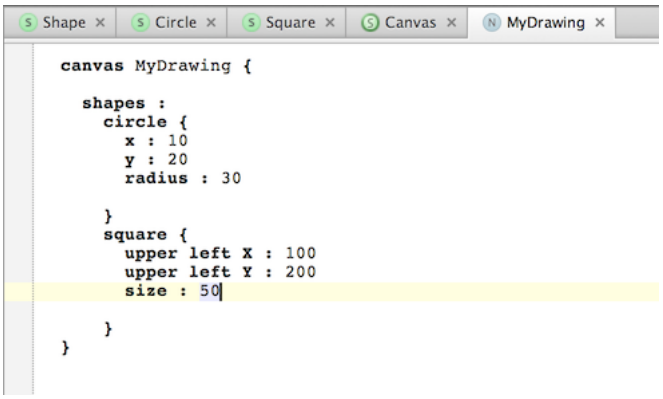
Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/





Vous pouvez insérer des formes supplémentaires en appuyant sur *Entrée* à la fin de la dernière forme de la liste de formes, ce qui dans notre cas est le symbole « } » de « circle ».





J'espère que vous fêtez votre succès comme il se doit ! Maintenant, nous vous proposons de voir comment optimiser ces éditeurs pour améliorer le rendu du code à l'écran.

Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/

X - Éditeurs

MPS utilise un éditeur projectionnel. Vous aviez peut-être déjà remarqué que l'éditeur se comportait légèrement différemment par rapport à ce que vous attendiez. Contrairement aux langages basés sur le texte, MPS ne représente jamais le code sous forme de texte brut. Dans MPS, le code est associé aux arbres de syntaxe abstraite (AST). Toujours.

Cela présente des avantages énormes pour la conception de langage, sa composabilité et les notations non analysables qui sont définies plus en détail dans la documentation de MPS.

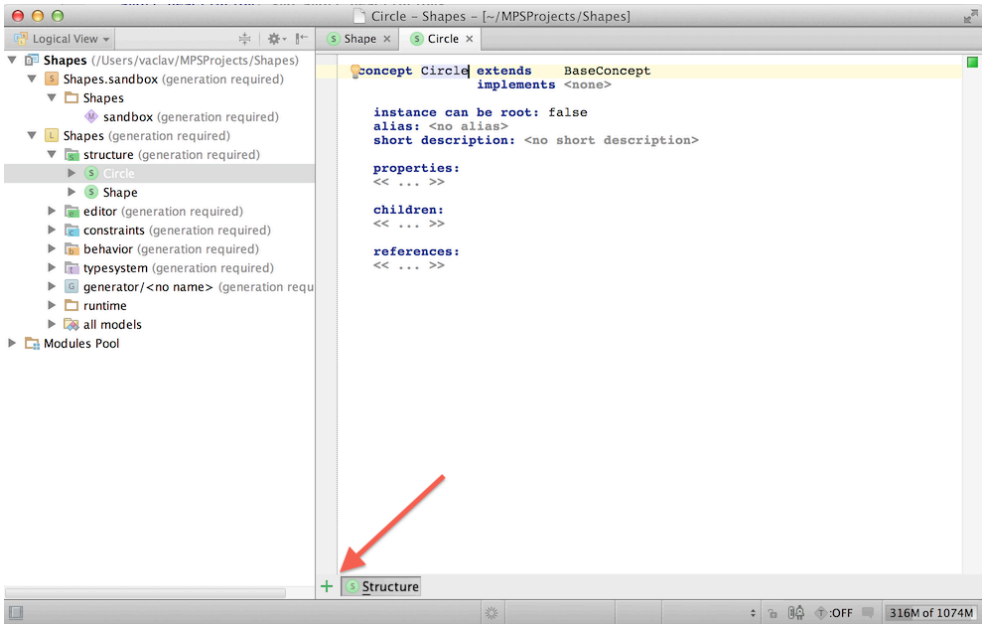
Dans le cadre de ce tutoriel, nous allons nous limiter à l'aspect édition des langages projectionnels. Puisque les éditeurs en texte brut ne permettent pas de représenter les AST de façon fiable et que l'édition directe des AST serait très compliquée, les langages projectionnels doivent fournir des éditeurs pour tous les *concepts* qui font partie de ce langage. Parfois, il faut plusieurs éditeurs pour un *concept* unique, mais cela va au-delà des objectifs que nous nous sommes fixés dans ce tutoriel. Dans la plupart des cas, MPS fournit un éditeur par défaut pour les *concepts* qui n'en ont pas encore. Ceci est très utile pour le prototypage du langage. Cependant, pour faciliter la tâche des utilisateurs, il est important de consacrer du temps à la préparation d'un éditeur explicite.

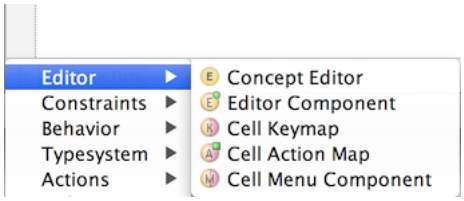
X-A - Shape

Le concept *Shape* ne nécessite pas d'éditeur, car il s'agit d'un concept **abstrait**. Par conséquent, nous allons laisser cela tel quel.

X-B - Circle

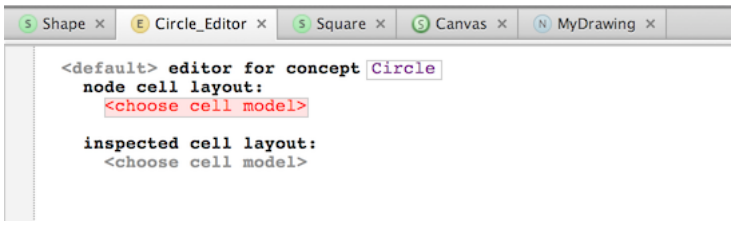
Pour *Circle* nous pourrions créer un éditeur qui place parfaitement toutes les propriétés nécessaires sur une seule ligne. Nous allons ouvrir le concept *Circle* dans l'éditeur, cliquer sur le bouton vert « + » dans le coin inférieur gauche et sélectionner *Editor -> Concept Editor*.





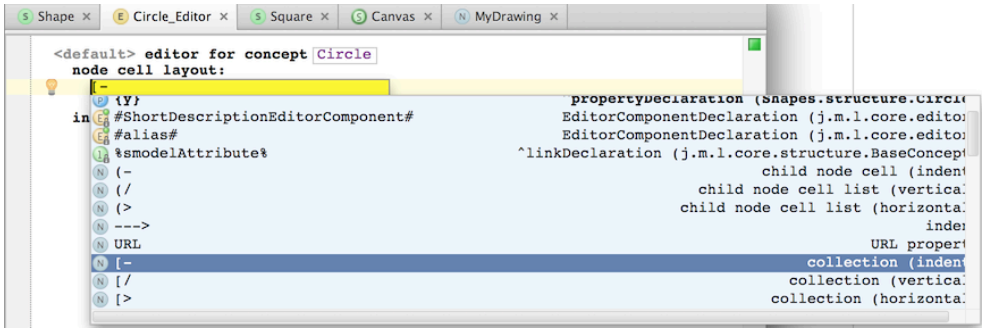
Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/

Vous allez obtenir une définition d'éditeur vide pour le concept *Circle*. Rappelez-vous qu'ici *Ctrl + Espace* sera très utile pour éditer les valeurs nécessaires.

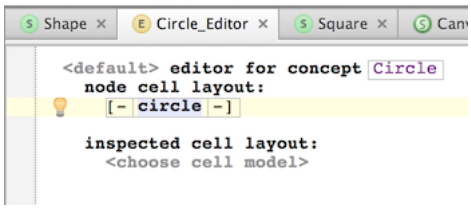


L'éditeur de concept de MPS consiste en des cellules visuelles, chacune représentant une information en relation avec le concept sous-jacent. Dans la mesure où les nœuds sont composés de façon hiérarchique dans un AST, leurs éditeurs sont composés à l'écran avec des éditeurs de sous-nœuds incorporés dans ceux de leurs ancêtres.

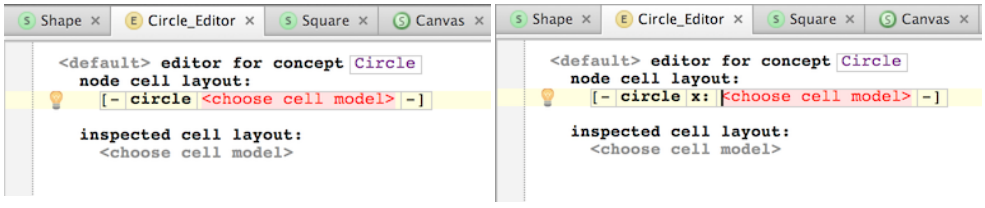
Pour *Circle*, nous aimerions afficher la valeur de toutes les propriétés (*x*, *y*, *radius*) ainsi que le texte arbitraire qui les entoure, le tout sur une seule ligne.
Par conséquent, nous allons commencer par choisir une disposition pour ces cellules et *indent layout* est parfaitement adapté ici. Appuyez sur *Ctrl + Espace* dans la cellule rouge et sélectionnez cette option dans la liste. Vous pouvez accélérer la recherche en tapant un crochet suivi par un symbole moins.



Maintenant, tapez « circle » pour entrer du texte constant qui sera placé au début de la ligne.

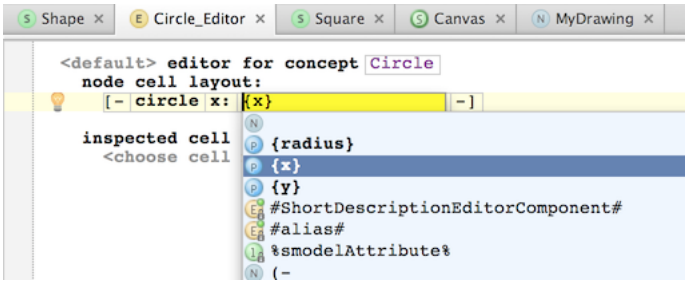


Appuyez sur *Entrée* pour créer une nouvelle cellule. Tapez « x: » pour indiquer que la cellule suivante contient la valeur de la propriété *x*.

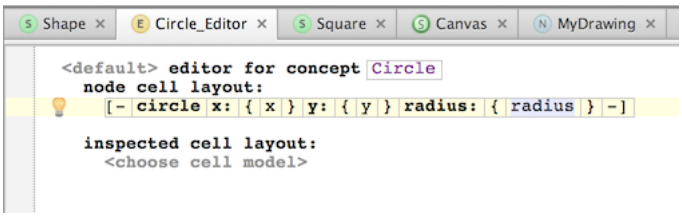


Maintenant, vous devez sélectionner la propriété *x* dans le menu de saisie automatique de code pour lier la cellule à la bonne propriété.

Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/

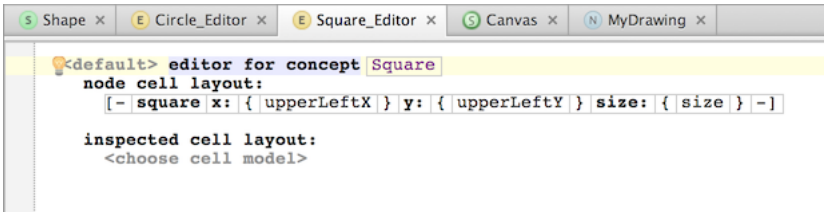


Vous pouvez à présent continuer seul pour insérer des cellules contenant du texte constant, ainsi que les valeurs des propriétés *y* et *radius*. Pour rappel, *Entrée* insère de nouvelles cellules, *Ctrl* + *Espace* ouvre le menu de saisie automatique. Vous devez obtenir un éditeur comme celui-ci.



X-C - Carré

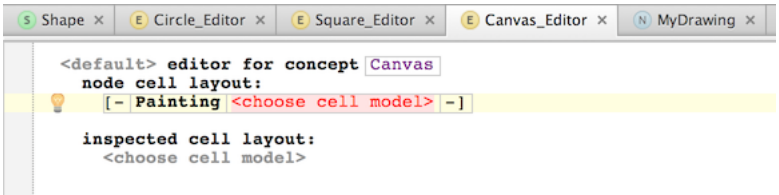
Square nécessite également un éditeur. Ouvrez le concept *Square* dans l'éditeur, appuyez sur le symbole « + » dans le coin inférieur gauche et créez un nouvel *éditeur de concept*. En suivant les instructions de la section précédente, entrez la définition de l'éditeur de la façon suivante :



Vous disposez désormais de tous les éléments nécessaires pour y parvenir.

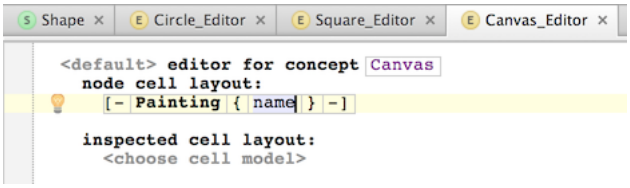
X-D - Toile

L'éditeur du concept *Canvas* sera légèrement différent, car il s'étale sur plusieurs lignes et contient une collection de nœuds enfants. Commencez toutefois de la même façon que précédemment, ouvrez le concept *Canvas*, appuyez sur le symbole « + » pour créer un nouvel *éditeur de concept*, définissez les retraits avec *indent layout* et entrez du texte pour obtenir ce qui suit :

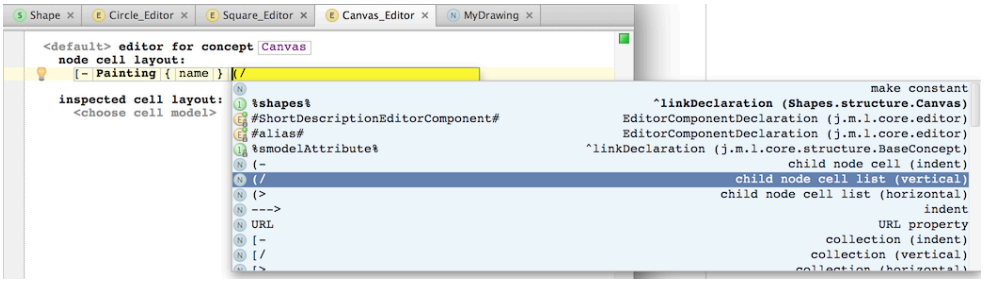


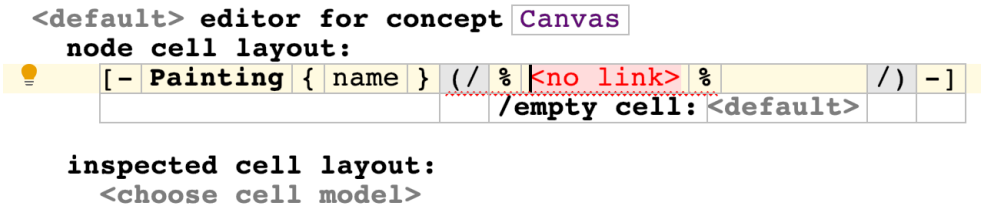
Liez maintenant la cellule rouge à la propriété *name*. Dans la mesure où la propriété est héritée de l'interface _InamedConcept _concept, elle apparaît plus bas dans la liste, mais est bien présente.

Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/

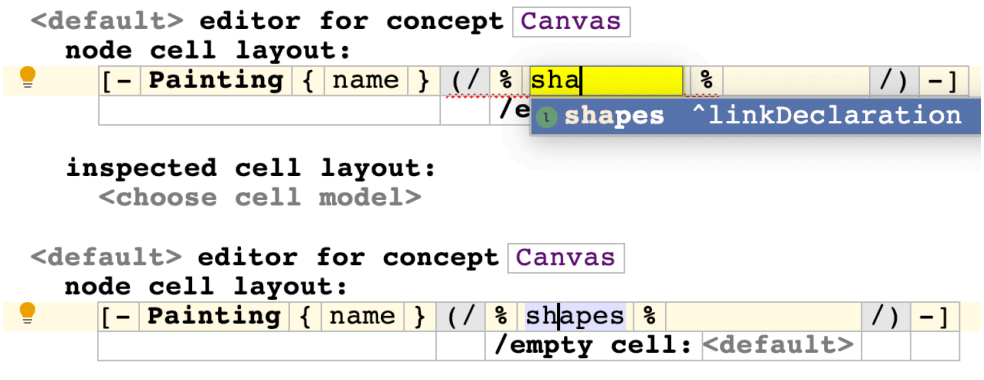


La cellule suivante contient une collection de formes qui ont été ajoutées à la section *Canvas*. Remarquez la composition des éditeurs ici : *Canvas* désigne uniquement la zone dans laquelle des formes doivent être modifiées et ne spécifie pas comment la zone doit être utilisée.
Dans la mesure où les formes doivent être organisées verticalement, chacune sur leur ligne, vous devez sélectionner *vertical collection layout* dans le menu de saisie automatique.



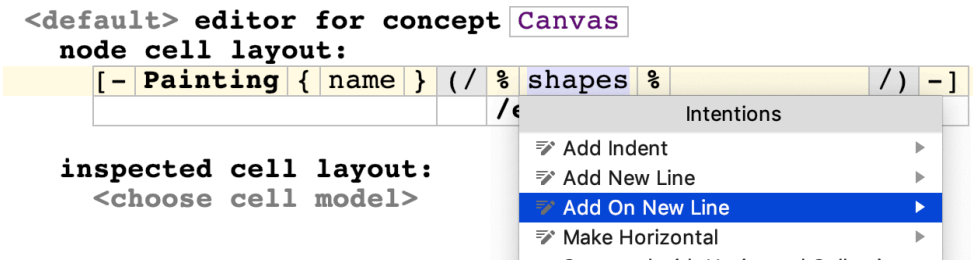


Vous devez lier la cellule rouge à la collection enfant *shapes* de *Canvas*.

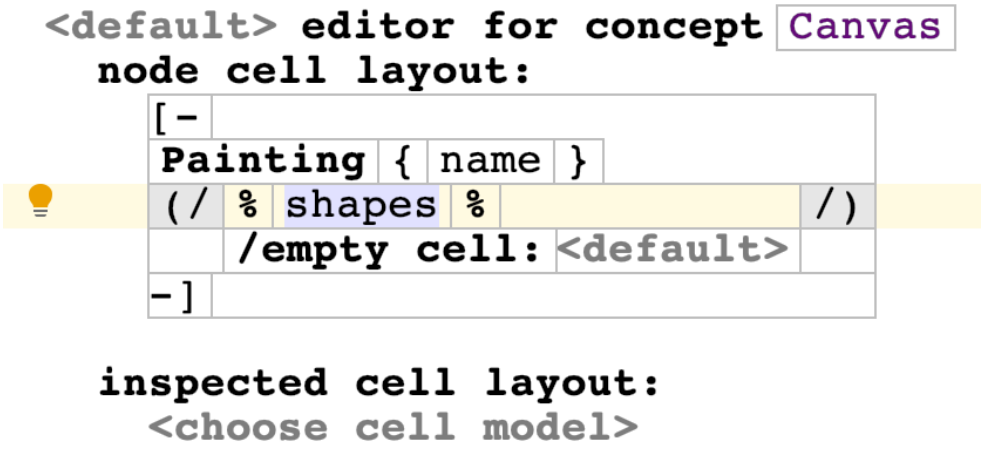


Désormais, pour placer la collection sous le nom utilisé pour *Canvas*, vous devez utiliser le raccourci *Alt* + *Entrée* pour afficher la fenêtre contextuelle d'intention et sélectionner « Add On New Line ».

Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/

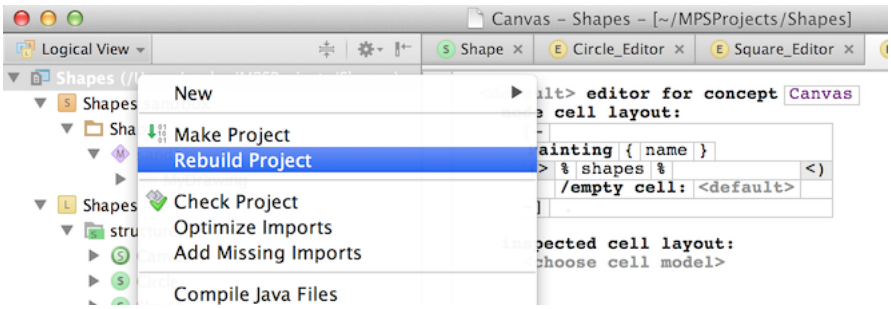


Vous obtenez la définition finale de l'éditeur :

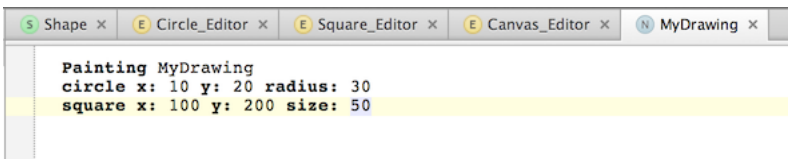


XI - Deuxième exécution

Désormais, vous pouvez recréer la build (faites un clic droit sur le nœud de plus haut niveau dans la *vue de projet*) et consultez le code de la sandbox dans *MyDrawing*.



Regardez comment la disposition du code a changé :



Il s'agit du même code (AST), mais il est organisé différemment à l'écran.

Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/

XII - Application de couleurs

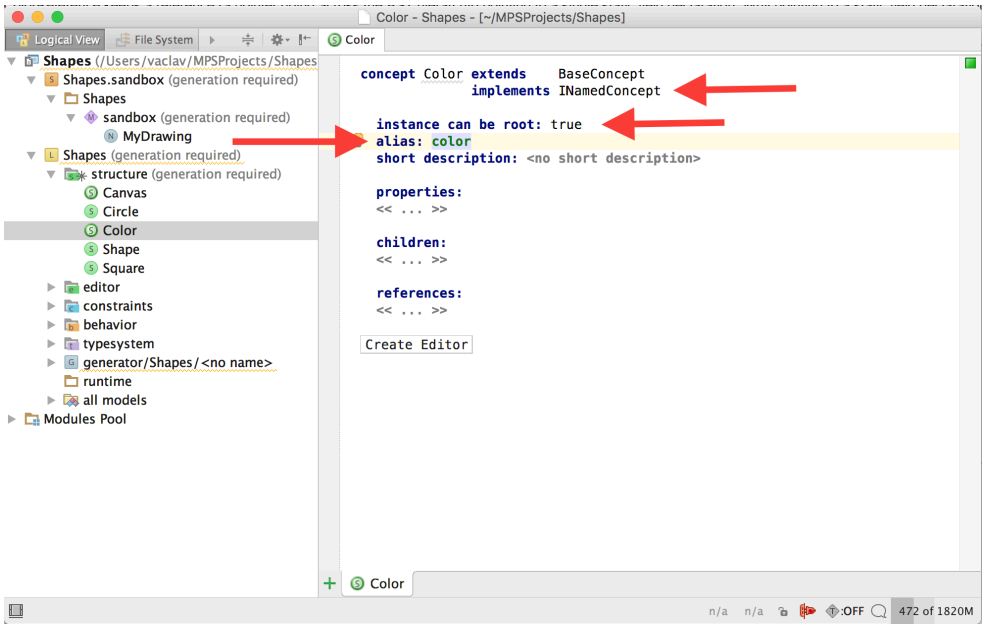
Nous devons à présent revenir à la section *Shape* et assurer la prise en charge des couleurs. Puisque *Circle* et *Square* étendent *Shape*, ils héritent tous les deux de la couleur.

Si pour une raison ou une autre, vous devez passer directement au générateur, c'est le moment idéal pour le faire. Vous devez simplement ignorer tous les emplacements où nous faisons référence aux couleurs dans le chapitre consacré au générateur.

Comme nous souhaitons permettre aux utilisateurs de notre langage de sélectionner la couleur de la forme dans une liste de couleurs prédéfinies, nous ne pouvons pas nous contenter d'utiliser une propriété de type texte pour contenir la valeur de couleur. À la place, nous allons ajouter une référence à *Shape* et obliger cette référence à pointer vers l'une des constantes de couleur prédéfinies. Nous allons commencer par créer ce type de constante pour représenter les couleurs prédéfinies.

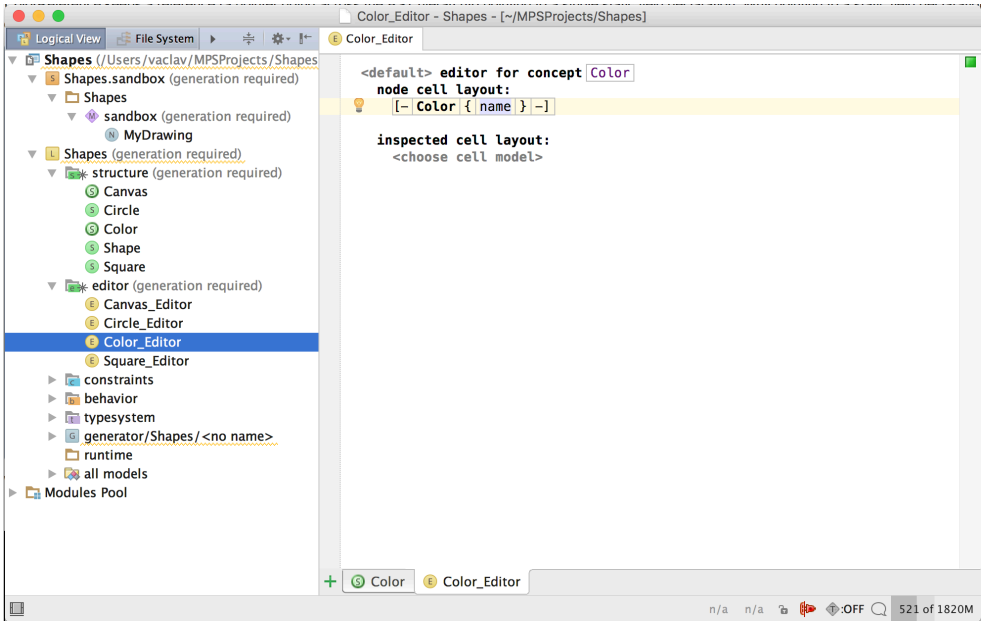
XII-A - Concept pour les couleurs

Tout d'abord, nous allons créer un concept représentant une constante de couleur. Nous allons l'appeler *Color* et lui donner un attribut racine, afin de la placer dans les modèles :



Un éditeur est également nécessaire :

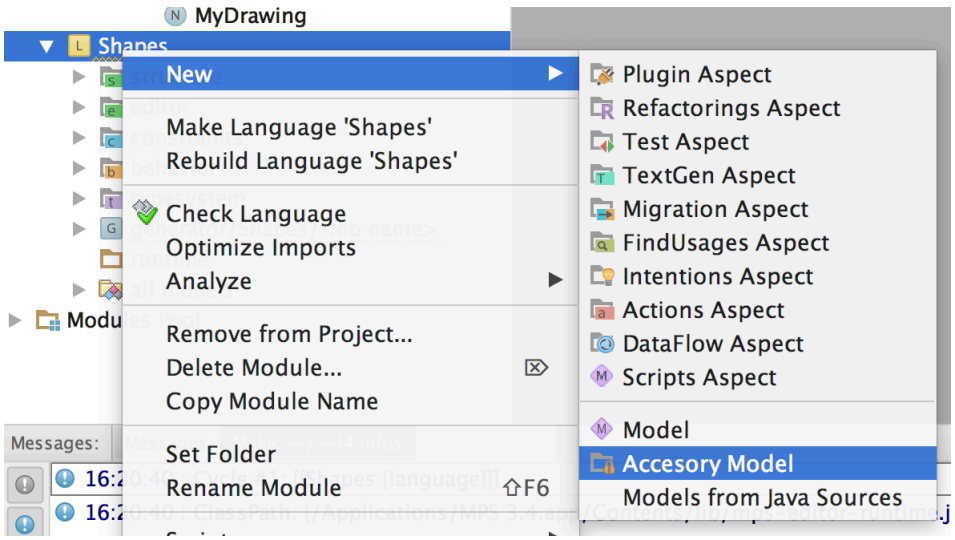
Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/



XII-B - Couleurs prédéfinies

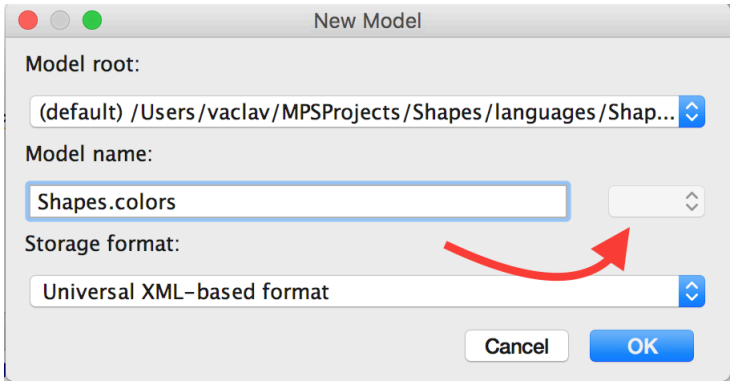
Nous devons maintenant fournir des nœuds concrets pour le concept *Color*, afin de représenter les constantes de couleur individuelles auxquelles l'utilisateur de notre langage pourra faire référence depuis ses éléments *Canvas*. Nous allons utiliser les **modèles d'accessoires** pour cela. Les **modèles d'accessoires** sont des modèles appartenant à une définition de langage et qui contiennent des nœuds arbitraires qui deviennent une partie intégrante du langage et sont visibles par les utilisateurs de ce langage.

Tout d'abord, nous devons créer un modèle d'accessoire **Accessory model** dans le langage :

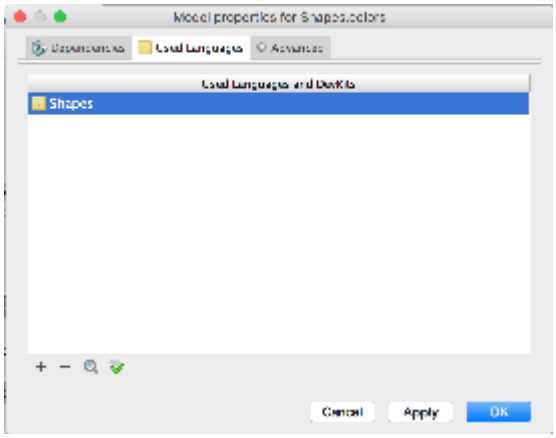


Ce modèle doit avoir un nom. Assurez-vous que la zone **stereotype** reste vide :

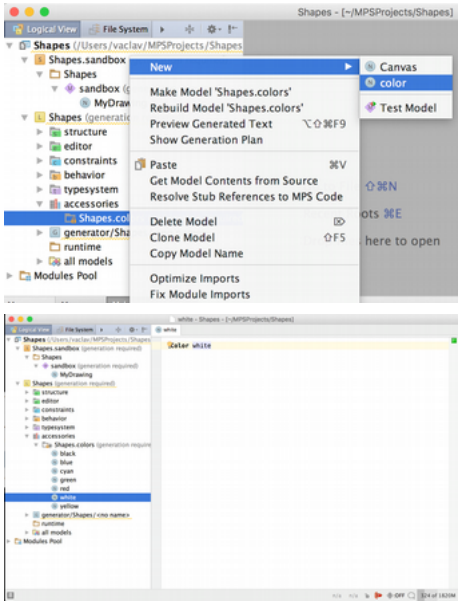
Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/



Le langage **Shapes**, qui déclare le concept *Color*, est le seul langage qui doit être importé dans le modèle d'accessoires :



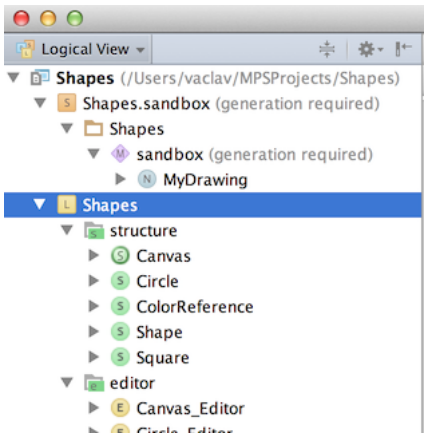
Vous devez désormais être capable de créer des constantes de couleur pour le modèle d'accessoires que vous venez de créer :



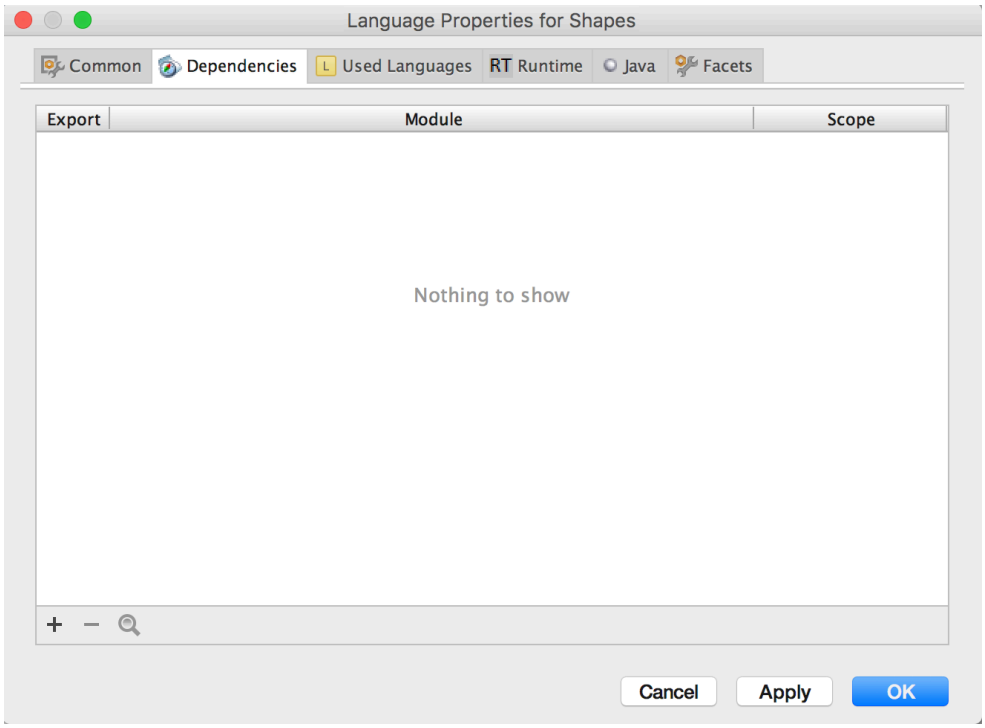
Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/

XII-C - Première touche sur les dépendances

Dans MPS, il est crucial de savoir comment gérer les dépendances et les langages importés. Pour afficher les dépendances d'un module ou d'un modèle, vous devez y accéder depuis la **vue de projet** située dans la partie gauche (*Alt + 1* pour ouvrir ce panneau) :

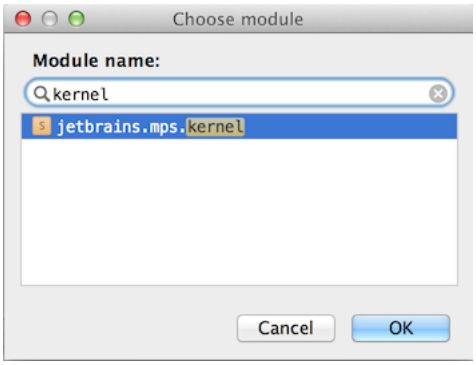


Appuyez sur *Alt + Entrée* (ou faites un clic droit dessus et choisissez **Module/Model Properties**) et vous obtenez une boîte de dialogue regroupant les propriétés du module/modèle. L'onglet *Dependencies* affiche les modules/modèles dont le module/modèle actuel dépend (par exemple l'importation dans Java nécessite Ruby, etc.).

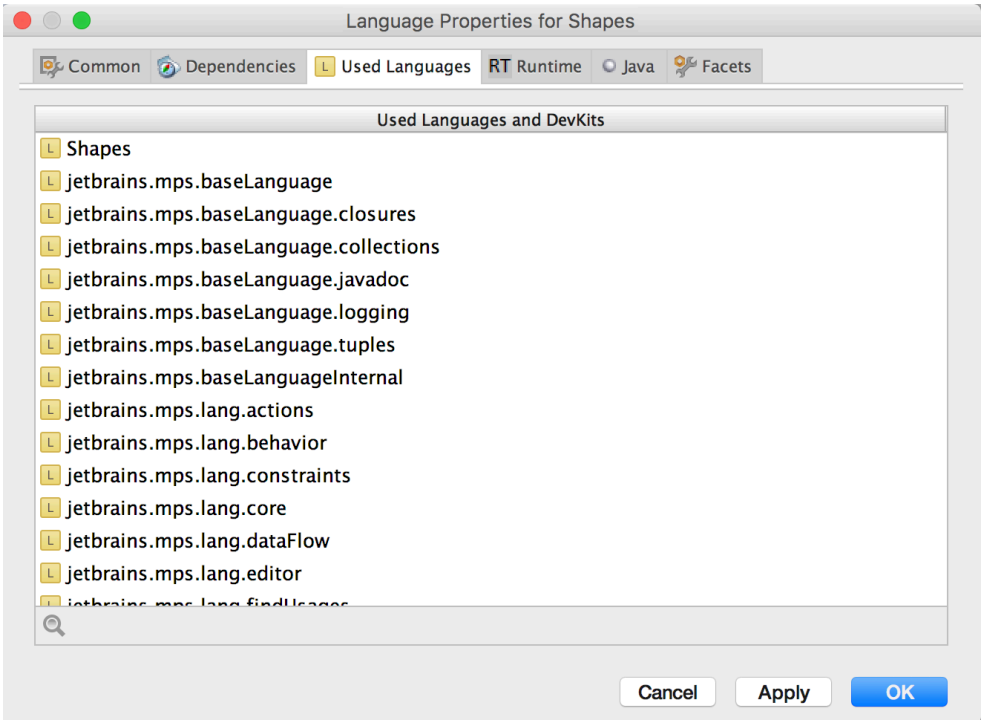


Le vôtre est probablement vide pour l'instant. Utilisez le bouton « + » pour ajouter des éléments. Dans la petite boîte de dialogue, tapez les premiers caractères du nom de la dépendance voulue pour préciser la recherche et appuyez sur *Entrée* lorsque vous la trouvez.

Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/



Dans la section *Used Languages*, indiquez les langages (syntaxes) à associer à votre module/modèle.



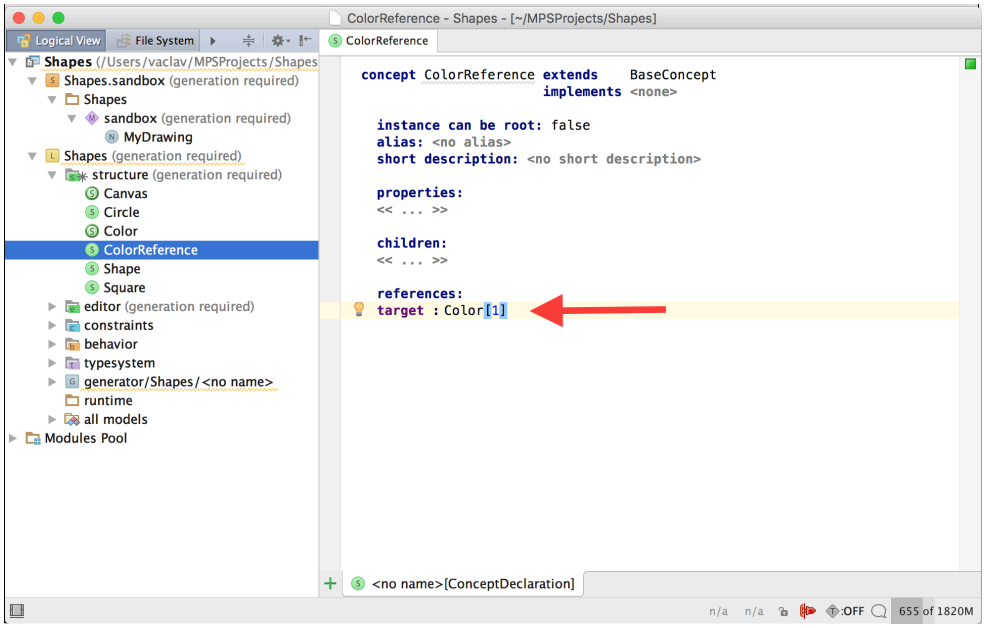
Nous n'avons pas à changer quoi que ce soit ici, alors poursuivons.

Si vous ne pouvez pas taper le code que nous suggérons dans ce tutoriel, vous avez probablement fait une erreur de définition des dépendances. Aussi, portez une attention particulière aux sections où le tutoriel mentionne des dépendances et suggère l'ajout d'une dépendance ou l'importation d'un langage.

XII-D - Concept de référence de couleur

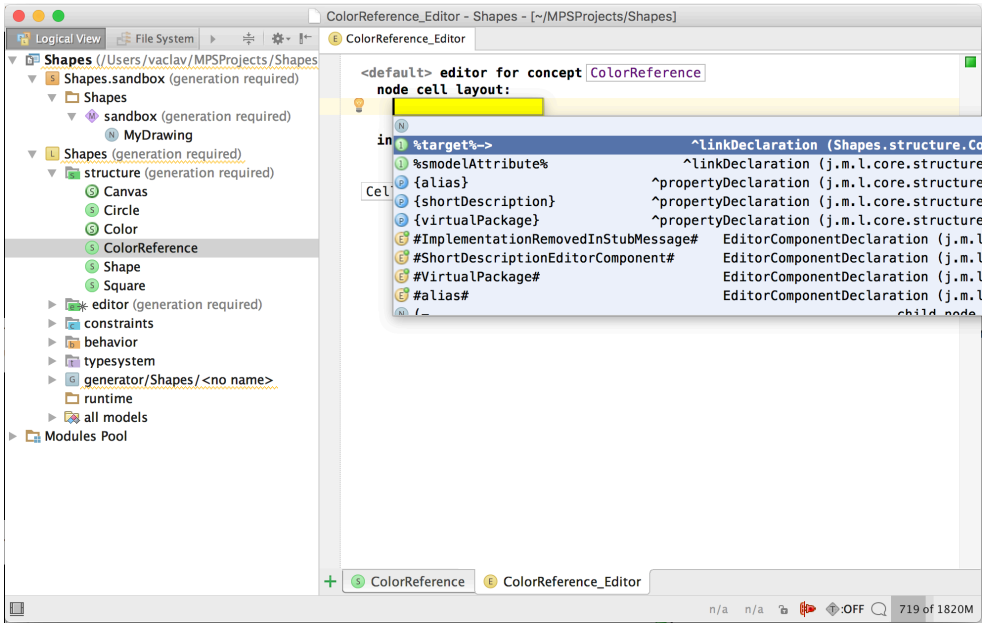
Il est temps de donner à notre langage la possibilité d'indiquer la couleur voulue d'une forme dans notre code. *Shape* doit faire référence à l'une des constantes de couleur dans le **modèle d'accessoires** du langage **Shapes**. MPS ajoute automatiquement l'ensemble des constantes de couleur disponibles dans le menu de saisie automatique à chaque fois que l'utilisateur est sur le point de spécifier une couleur pour un élément *Shape*.

Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/



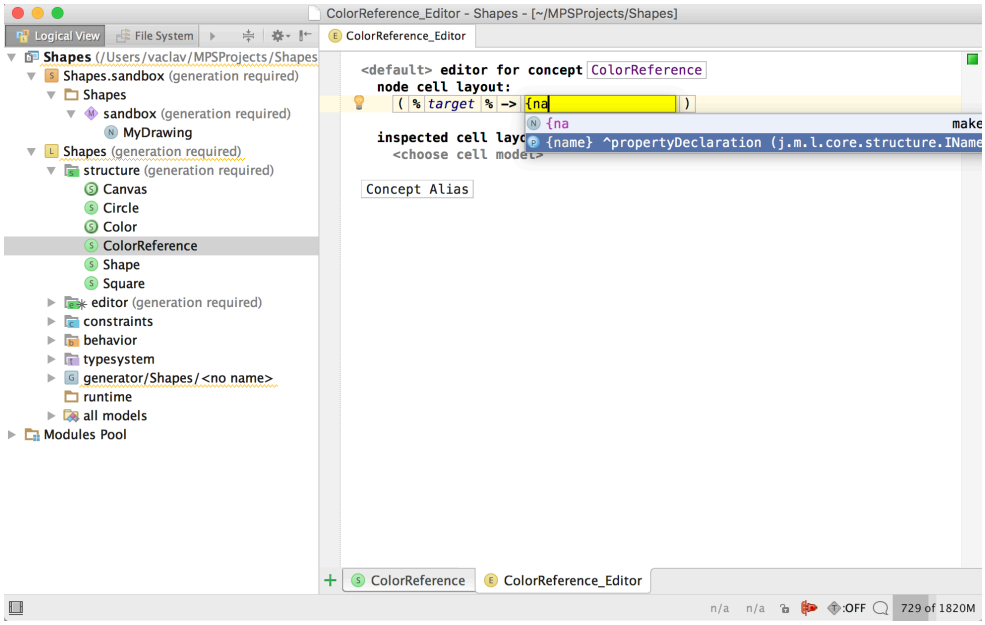
ColorReference intègre une référence (un pointeur couvrant toute la hiérarchie AST) pointant vers une constante de couleur unique (nœud du concept *Color*).

Pour afficher et modifier les couleurs, nous avons également besoin d'un éditeur pour *ColorReference*.

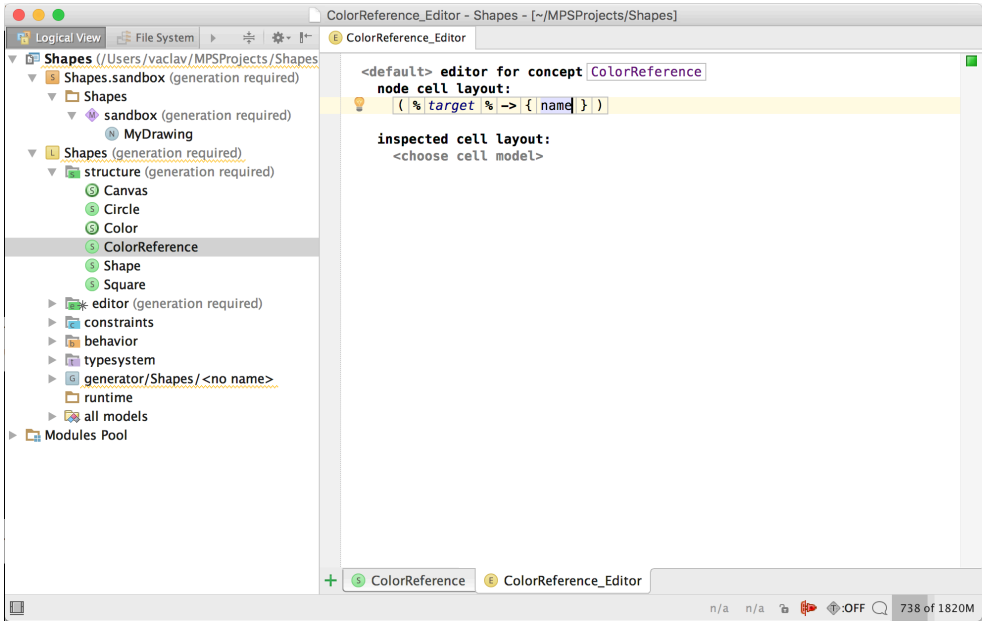


La référence doit simplement afficher le nom de la constante de couleur à laquelle elle fait référence, vous devez donc sélectionner *target* dans le menu de saisie automatique du code et spécifier que la propriété *name* de la constante de couleur doit être présentée à l'utilisateur.

Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/



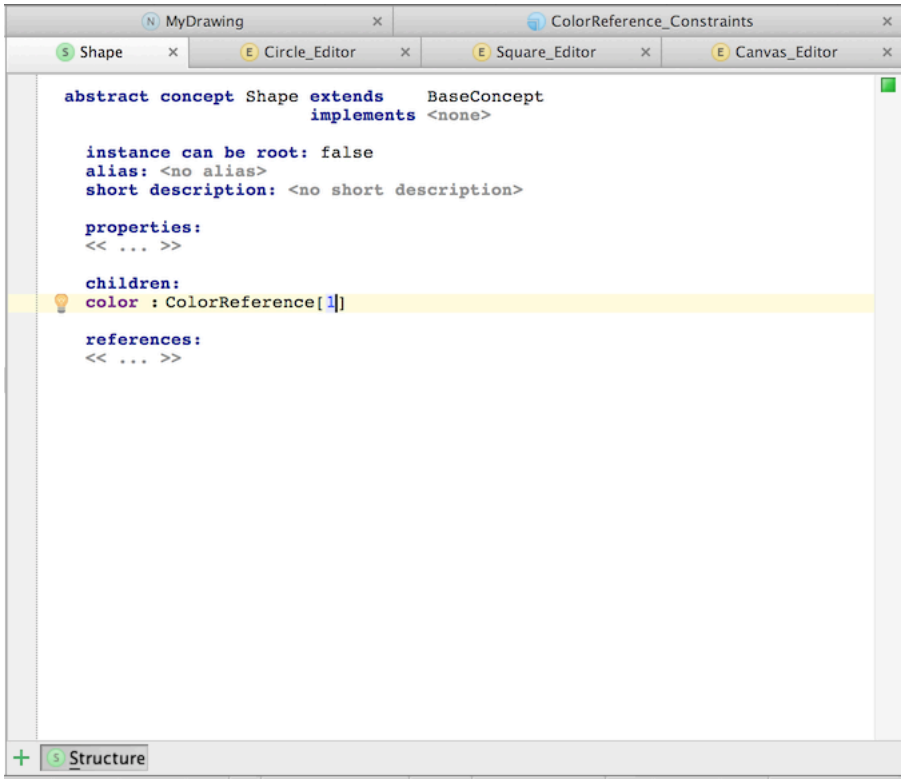
Vous devez alors obtenir la définition d'éditeur suivante :



XII-E - Mise à jour d'une forme

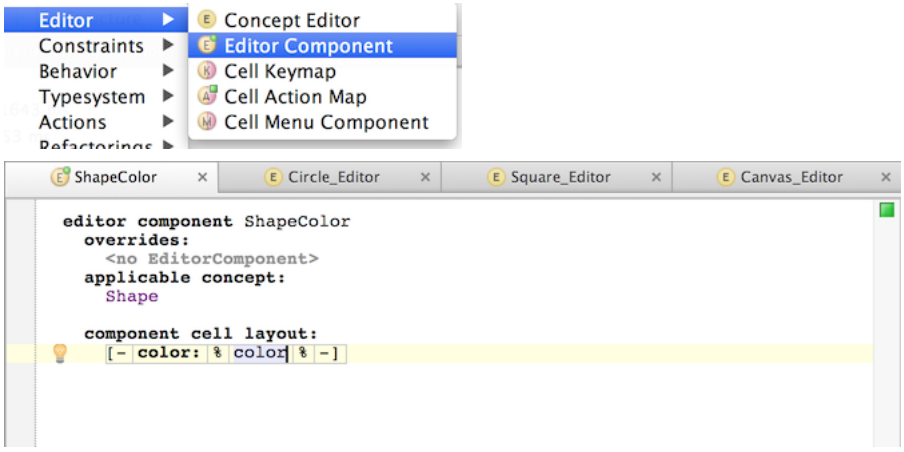
Le concept *Shape* est un bon endroit pour placer notre nouvel élément *ColorReference*, car *Circle* et *Square* vont en hériter.

Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/



Shape peut également définir un **composant d'éditeur** pour définir l'éditeur de la couleur. D'autre part, *Circle* et *Square* pourront réutiliser ce **composant d'éditeur** dans leurs éditeurs, afin d'éviter toute duplication.

Appuyez sur le symbole « + » et créez un nouveau *composant d'éditeur* (Editor Component).

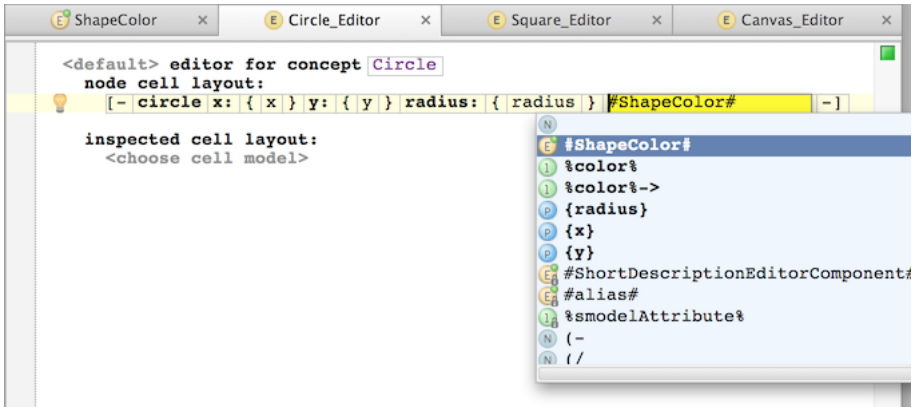


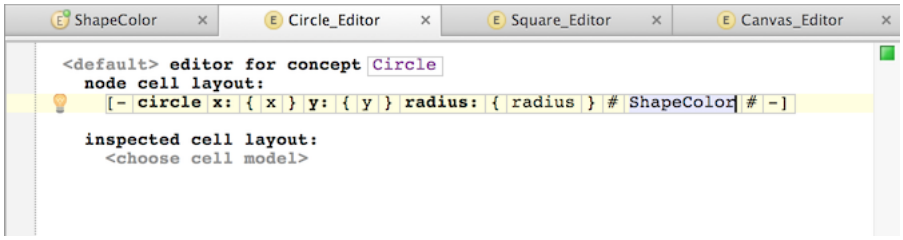
Vous connaissez déjà le langage de la définition du composant d'éditeur et par conséquent la disposition des retraits et la définition d'une cellule de texte de constante suivie d'une cellule liée à la propriété *color* ne devrait vous poser aucun problème.

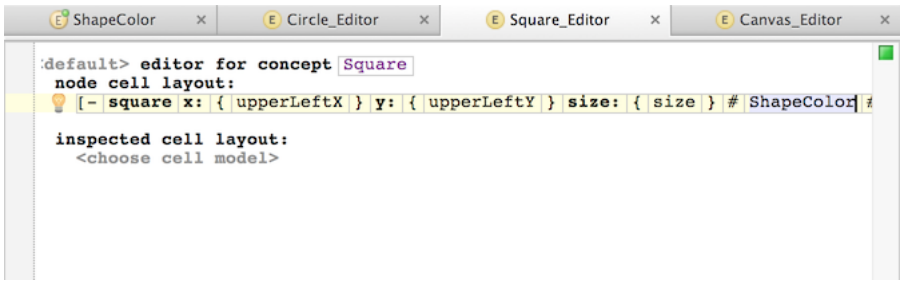
XIII - Intégration du composant d'éditeur

Le composant d'éditeur défini pour *Shape* doit maintenant être ajouté dans les éditeurs de *Circle* et *Square*.

Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/







Le **composant d'éditeur** forme simplement une nouvelle cellule dans la mise en page de l'éditeur.

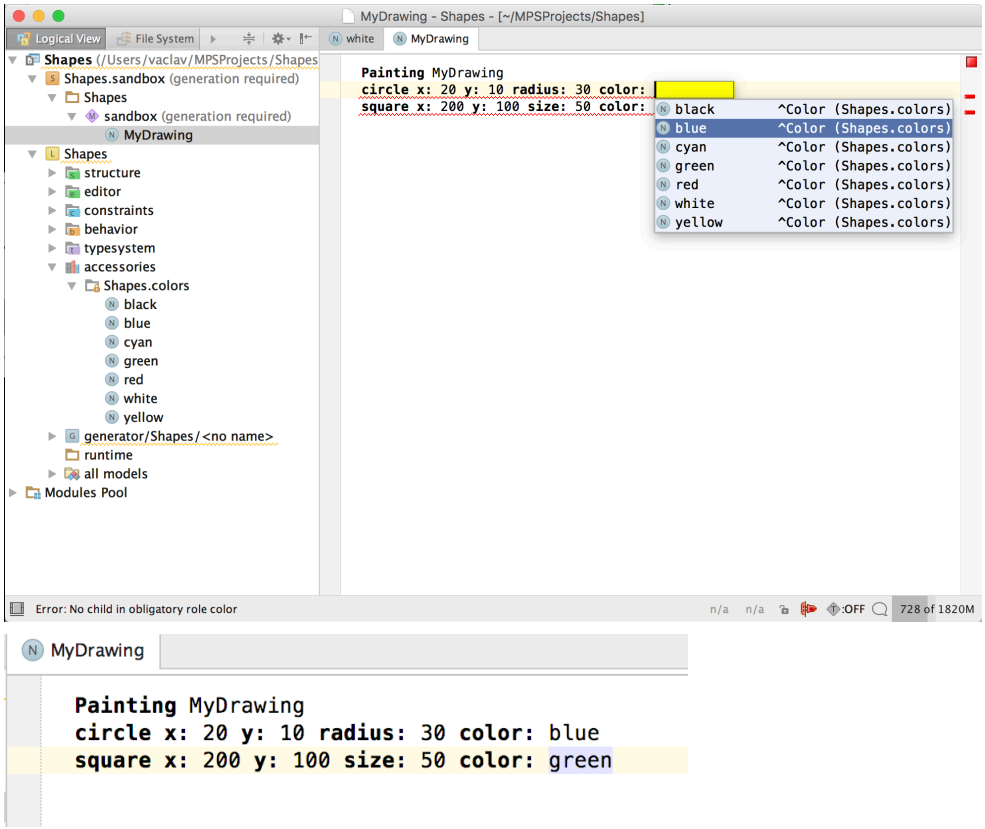
XIV - Une troisième exécution

Nous sommes arrivés au point le plus propice pour la génération d'une nouvelle build du programme. Faites un clic droit sur le nœud Language de la *vue de projet* et sélectionnez « Rebuild Language ».

Ici si vous recréez l'intégralité du projet et non pas simplement le langage, vous générez une ou deux erreurs, car la Solution perd ainsi les propriétés de couleurs qui viennent d'être définies. Nous allons les ajouter immédiatement.

Ouvrez le programme *MyDrawing* pour afficher les cellules de couleur vides en rouge. Essayez d'utiliser le raccourci *Ctrl + Espace* dans ces cellules pour obtenir la liste des couleurs sélectionnables.

Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/



Notre langage est maintenant totalement défini. Nous pouvons créer un élément *Canvas* et y ajouter des éléments *Circle* et *Square*, en spécifiant leurs positions, tailles et couleurs. C'est une réalisation considérable en un temps aussi court.

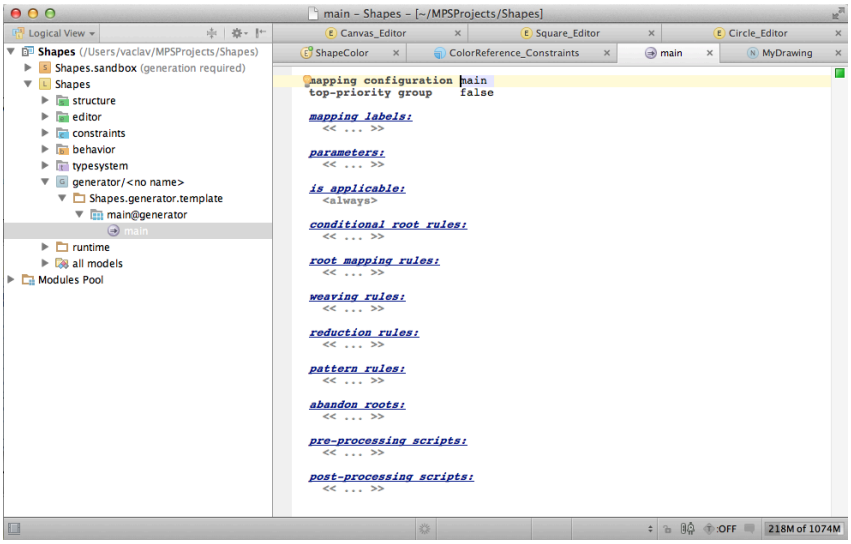
Il ne nous reste plus qu'à convertir ces programmes en Java, de façon à pouvoir les exécuter et voir les formes dessinées correctement à l'écran. Si vous poursuivez, vous verrez que nous y sommes presque.

XV - Générateur

Notre langage nécessite désormais un générateur pour nous permettre de générer le code à compiler et exécuter. Nous allons choisir *BaseLanguage* en tant que cible pour notre DSL. *BaseLanguage* est une copie de Java distribuée avec MPS et qui, par conséquent, peut être facilement transformée en sources texte Java que le compilateur Java peut ensuite compiler en code binaire. Nous pourrions également choisir toute autre plateforme et tout langage cible, à condition d'intégrer une définition de ce langage cible dans votre projet.

Le générateur sera très simple et nécessitera uniquement quelques règles et une seule configuration de mapping.

Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/

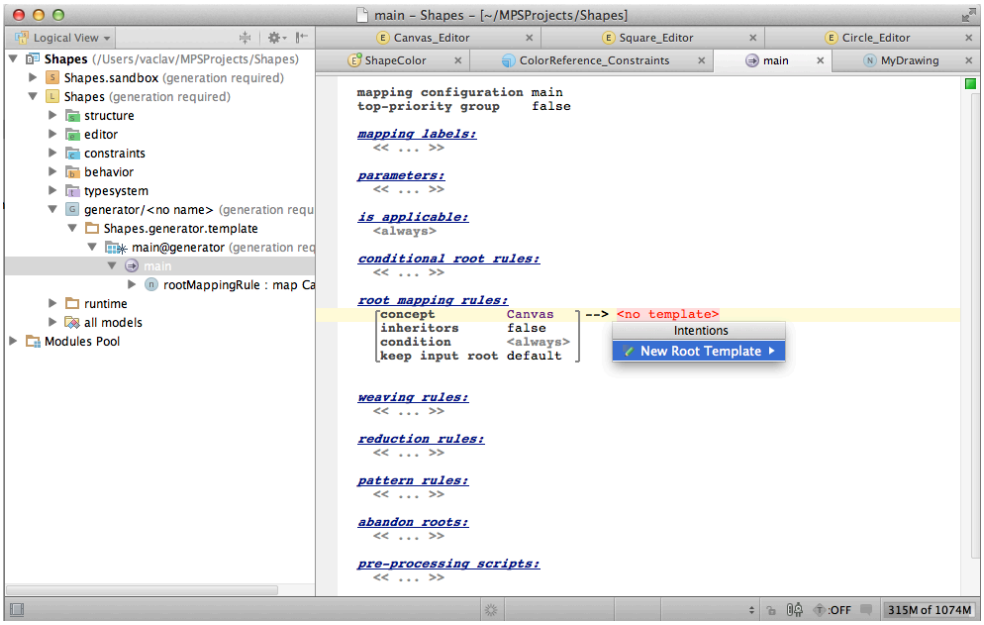


Votre langage contient déjà le squelette d'un générateur vide. Vous pouvez ouvrir la configuration de mapping qui va servir à spécifier quelle règle appliquer et quand. Nous allons ajouter des entrées de configuration ici graduellement.

Voici l'idée derrière le générateur que nous allons implémenter :

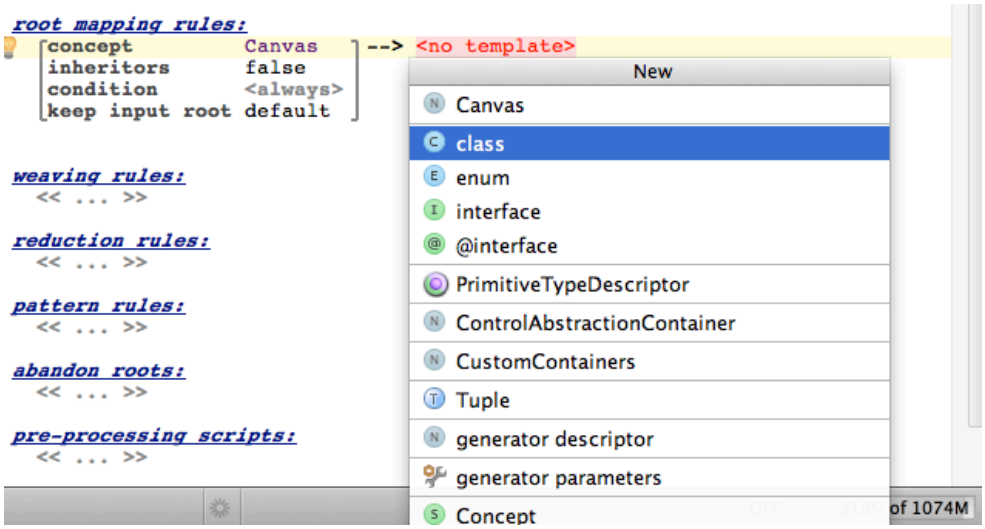
- un élément *Canvas* est converti en une classe Java qui vient étendre la classe *Jframe* de Java et contient un *Jpanel* sur lequel toutes les formes seront dessinées ;
- chaque élément *Shape* est converti en un appel de méthode sur l'objet *Graphics* pour dessiner la forme sur le *Jpanel* ;
- *ColorReference* est converti en une référence à la constante de couleur pertinente dans la classe *java.awt.Color*.

Commençons par la classe destinée à *Canvas*. Vous devez ajouter une nouvelle entrée dans les *rôles de mapping racine*, car *Canvas* est un concept racine.

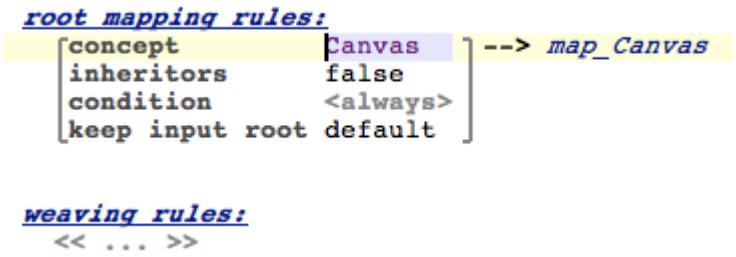


Cela indique que les nœuds *Canvas* doivent être remplacés par une classe Java. Vous devez utiliser *Alt* + *Entrée* pour sélectionner « New Root Template » dans le menu contextuel.

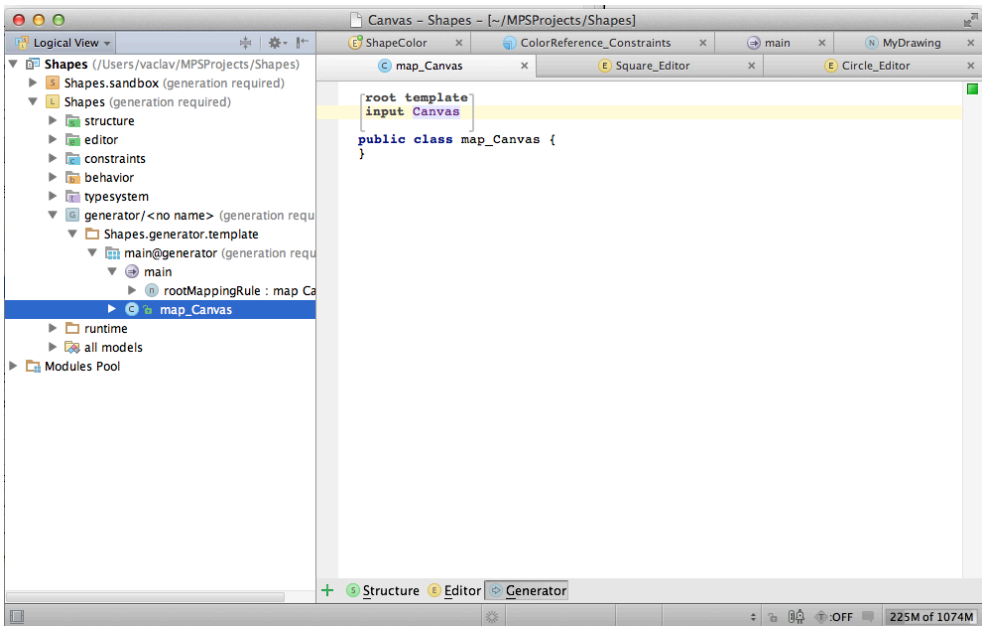
Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/



Le modèle racine doit générer une classe Java et par conséquent, vous devez sélectionner « class ».

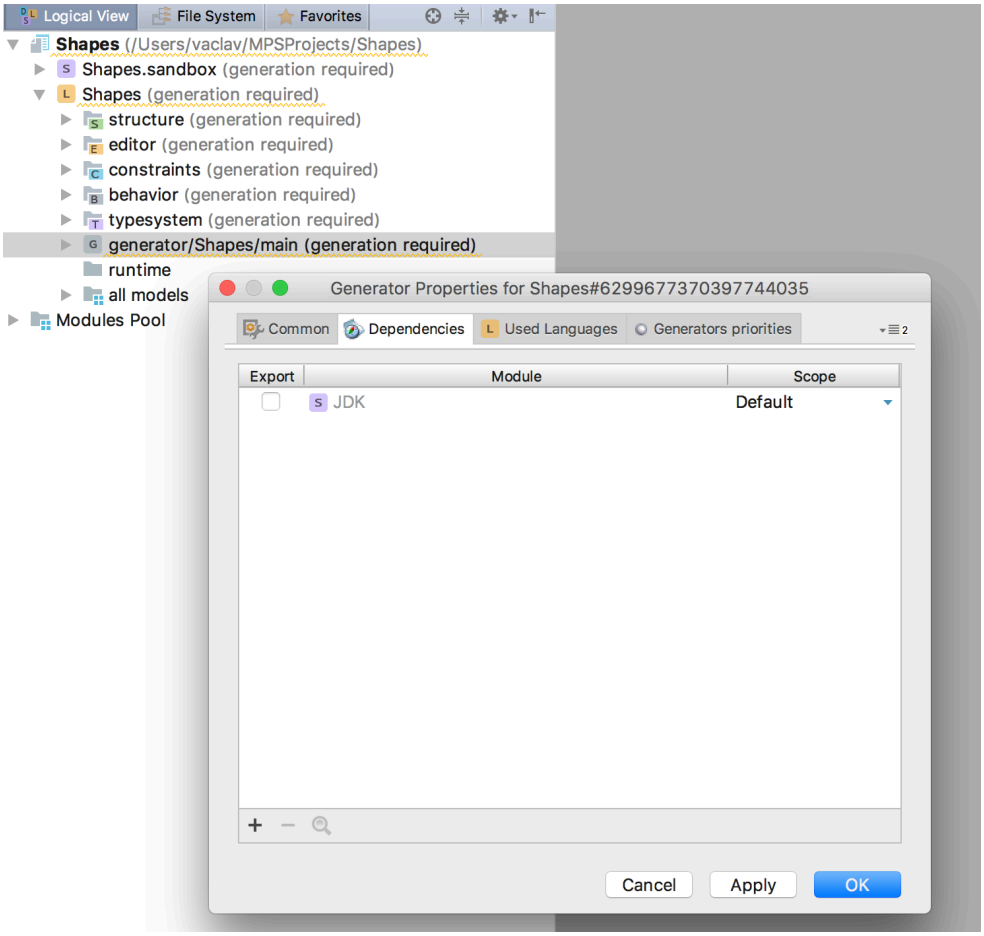


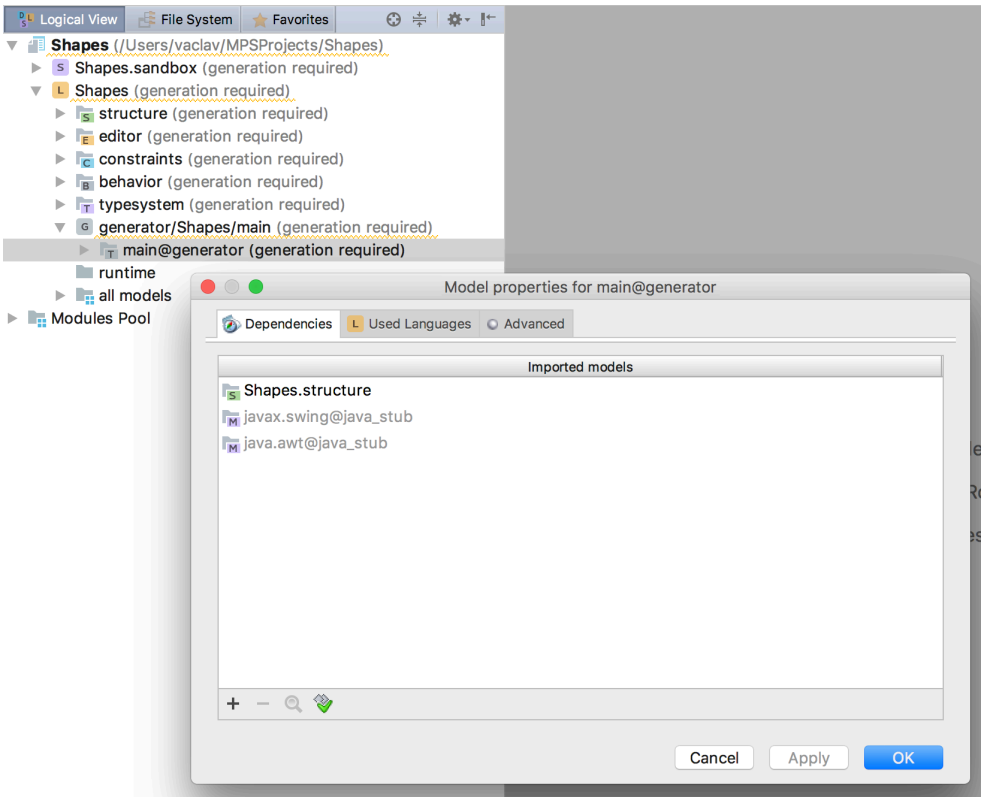
La règle de mapping source est maintenant terminée. Le nom *map_Canvas* désigne le modèle racine qui a été créé dans le générateur. Vous devez l'ouvrir pour pouvoir y apporter des modifications.



Mais avant cela, nous devons définir les dépendances du **module de générateur** et du **modèle de générateur** comme spécifié ci-dessous :

Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/



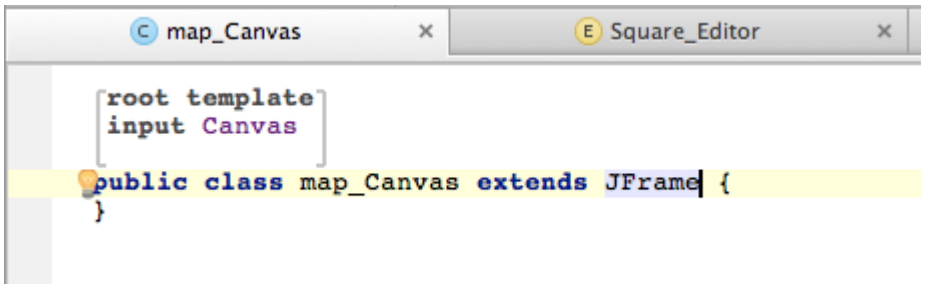


Souvenez-vous, *Alt + Entrée* affiche les propriétés du nœud sélectionné dans la *vue de projet* qui est située à gauche.

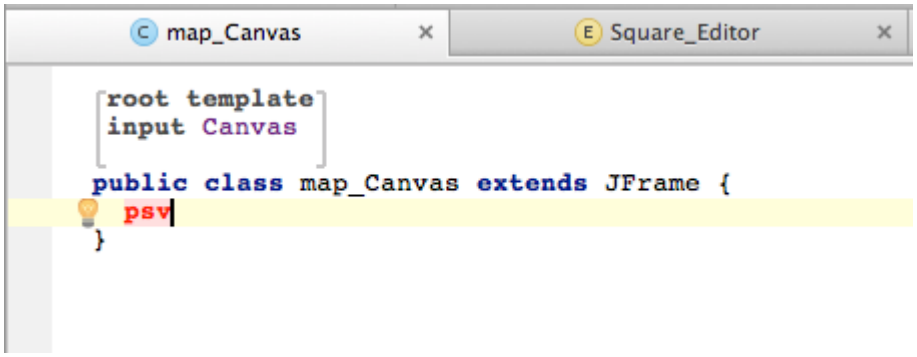
Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/

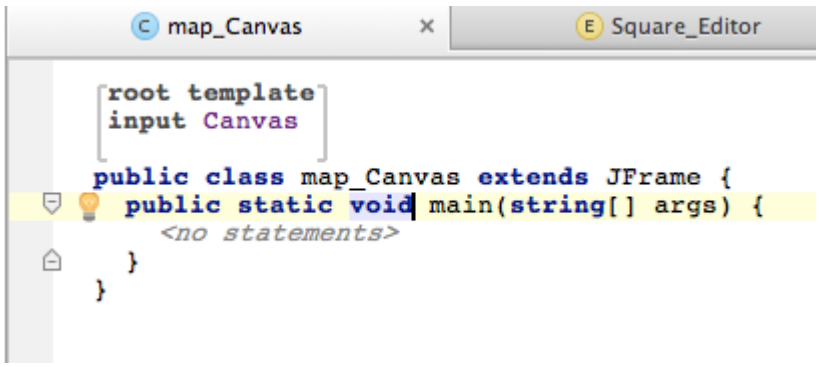
Les dépendances permettent de commencer à taper le code Java qui fera partie de la classe Java générée. Nous allons ensuite paramétrer le code avec des valeurs provenant de *Canvas*, afin qu'il reflète l'intention de l'utilisateur.

Cette classe doit étendre *JFrame*.



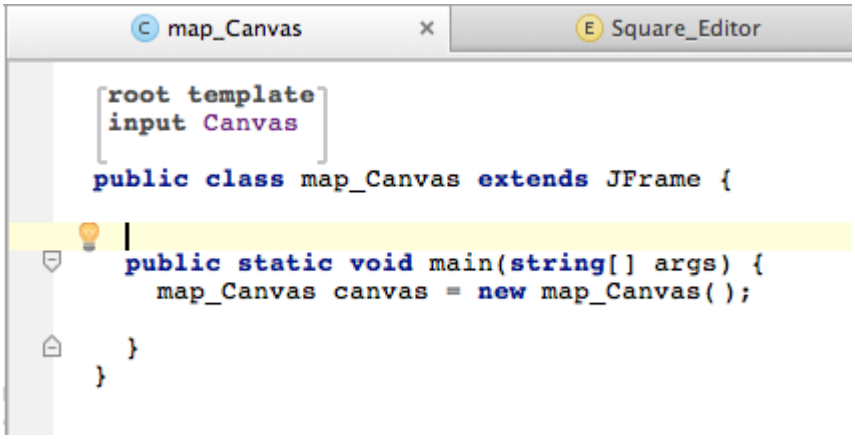
Vous allez maintenant ajouter la méthode *main* pour obtenir une classe Java exécutable. Vous pouvez utiliser le modèle actif « psvm » pour entrer la méthode rapidement.



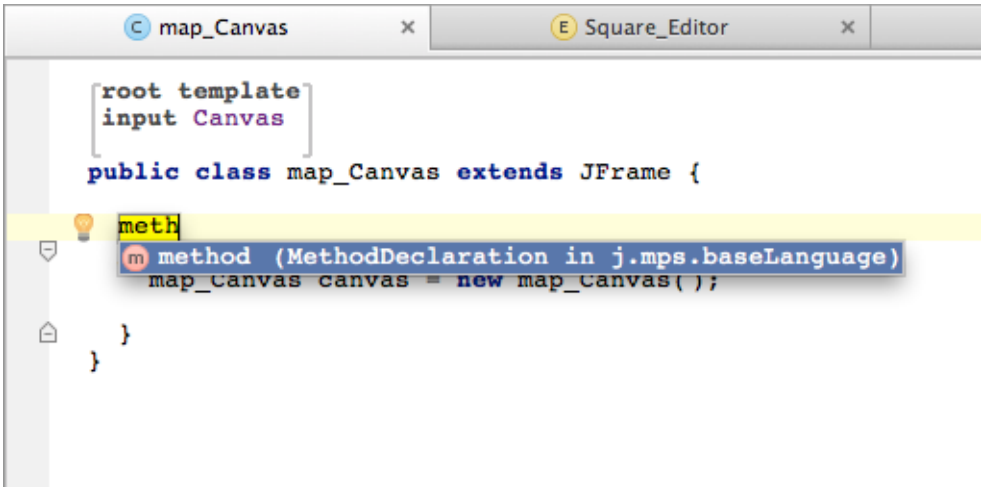


Dans la méthode, nous devons instancier *map_Canvas*.

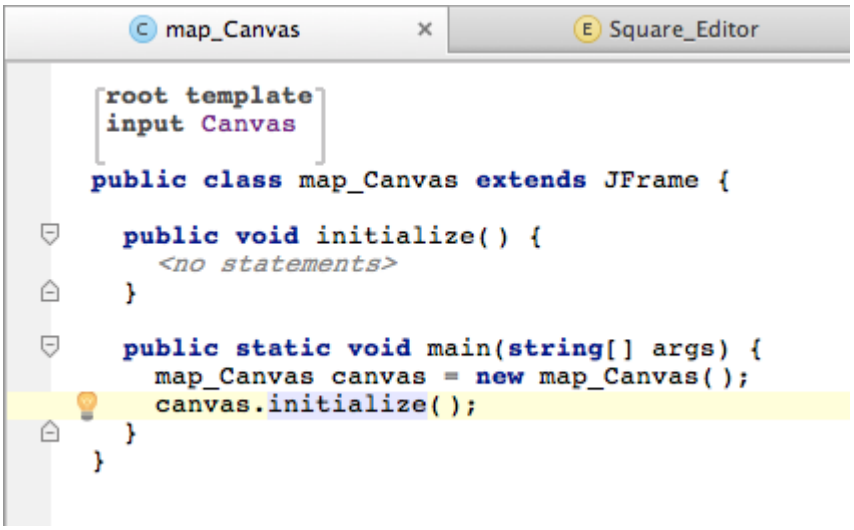
Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/



Il nous faut également une méthode pour initialiser l'image. Tapez « method » et utilisez *Ctrl + Espace* pour compléter la définition de la méthode :



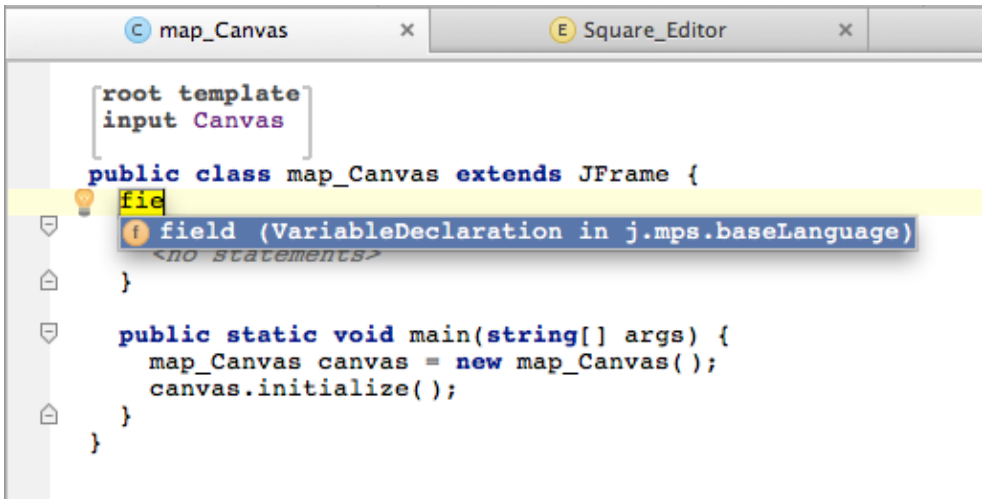
Appelez la méthode *Initialize()* et assurez-vous que **cette méthode est appelée** depuis *main*.

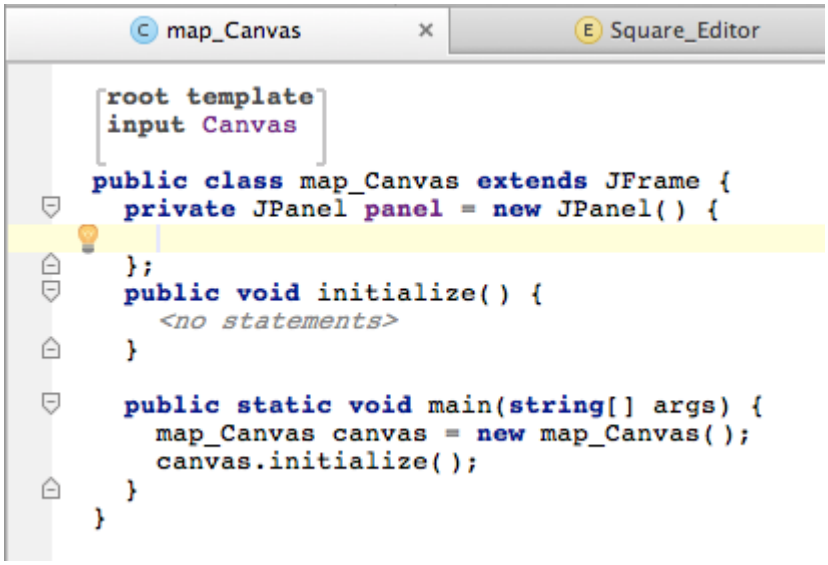


Remarquez l'appel canvas.initialize() dans la méthode main(). L'avez-vous ajouté ?

Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/

Toutes les formes seront ajoutées sur un *JPanel*, ce qui signifie que nous devons désormais en ajouter un en tant que champ.



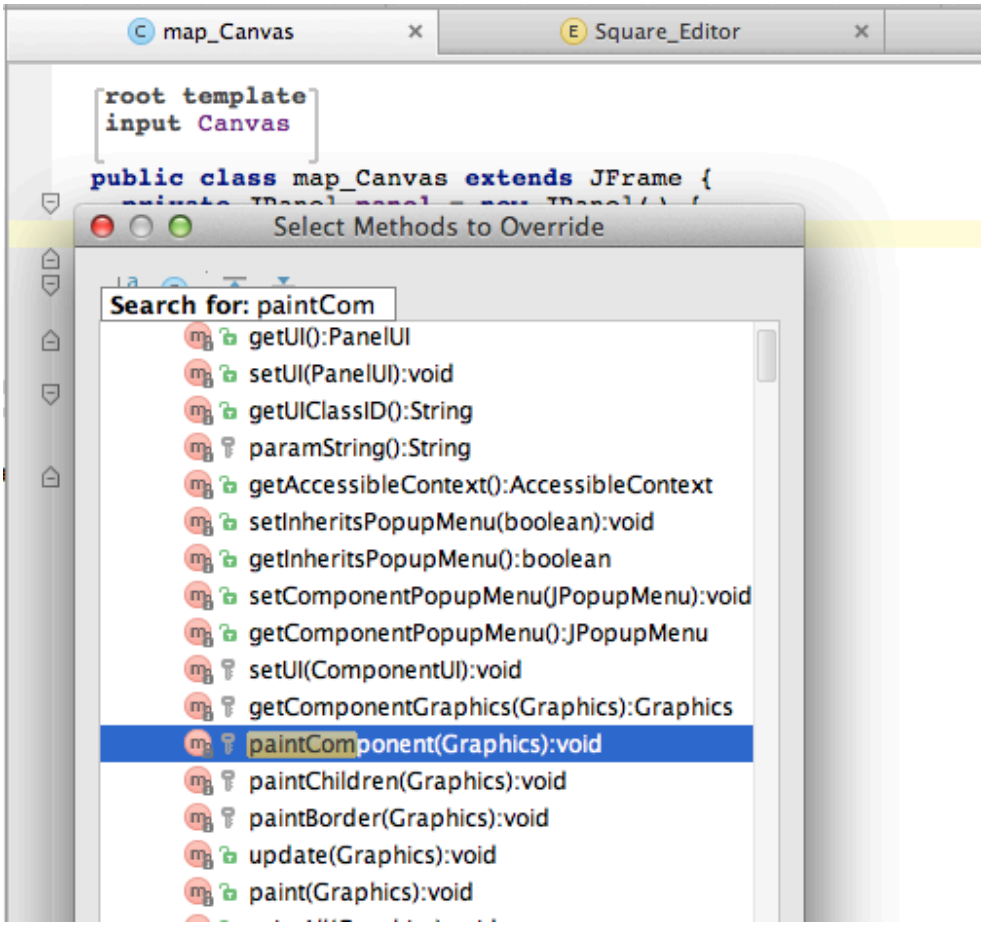


Remarque : nous utilisons une **classe intérieure anonyme** pour personnaliser en partie le *JPanel*.

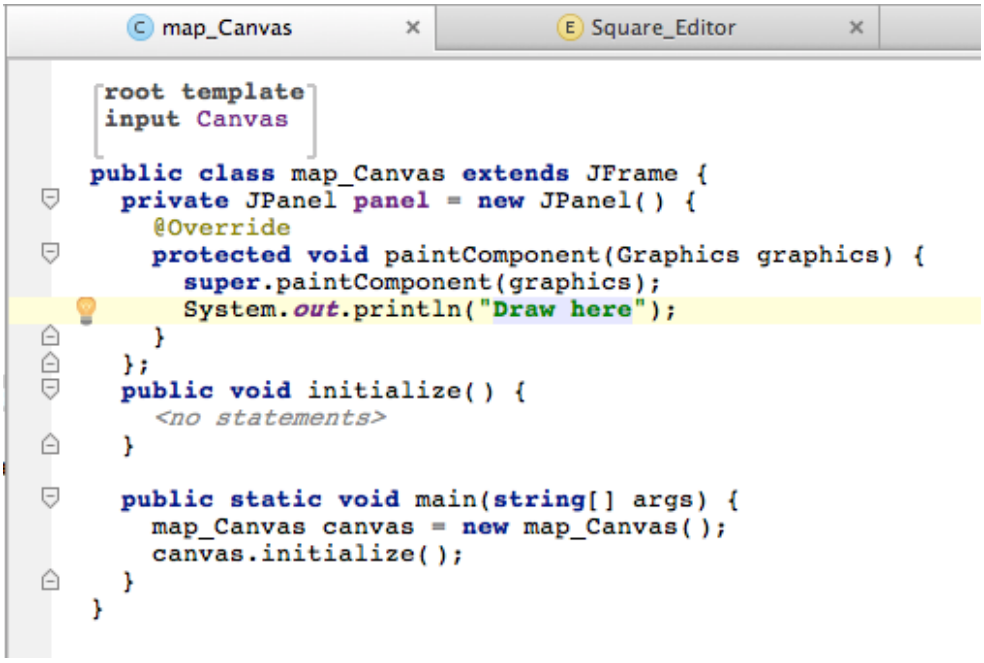
Important : pour créer une classe anonyme intérieure dans BaseLanguage, placez votre curseur juste après *new Jpanel()* et avant le point-virgule de fin. Appuyez ensuite sur la touche « { » (accolade gauche) et MPS se charge d'ajouter le symbole « } » final. Maintenez le curseur entre les symboles « { » et « } » pour ajouter des méthodes à la classe intérieure anonyme du panneau.

Nous allons ignorer la méthode *paintComponent* du *JPanel*, car il s'agit de la méthode Java permettant de dessiner facilement des formes sur le *JPanel*. Appuyez sur *Ctrl/Cmde + O* pendant que le curseur est dans le corps de classe anonyme du *JPanel*, entre les symboles « { » et « } », pour appeler la **boîte de dialogue de la méthode Override** pour le *Jpanel* et sélectionner la méthode **paintComponent**.

Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/

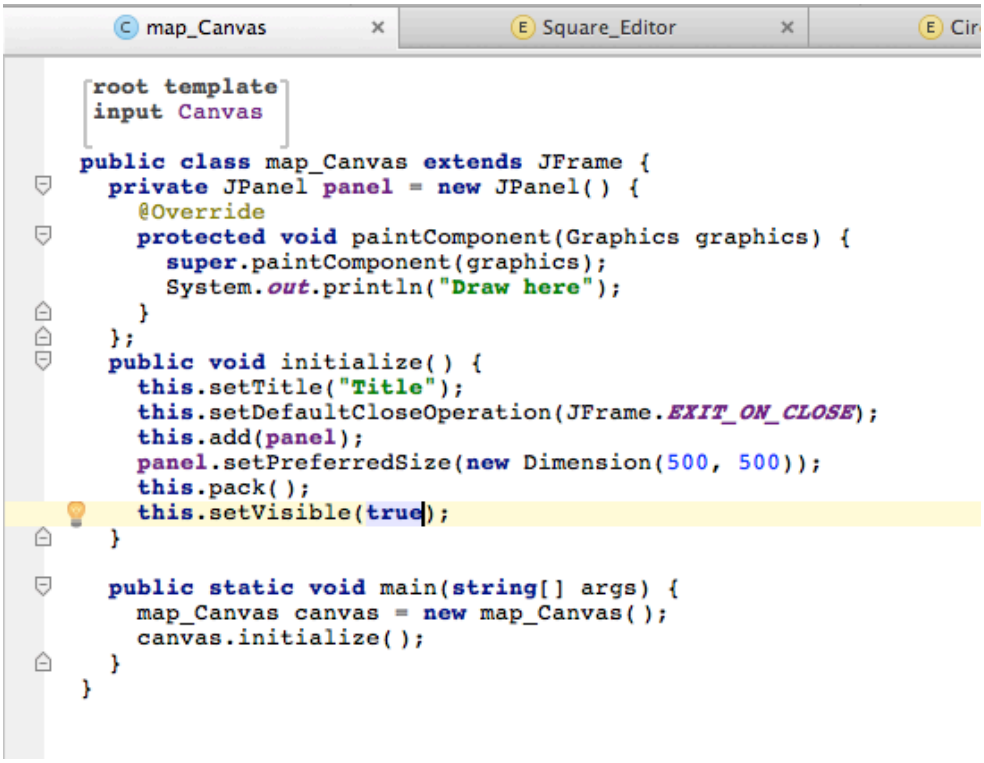


Assurez-vous que la méthode *paintComponent()* est bien imbriquée dans la classe intérieure anonyme du *JPanel*, comme illustré dans les copies d'écran. Assurez-vous également qu'elle est bien appelée **paintComponent** et non pas **paintComponents**.



Ensuite, il ne vous reste plus qu'à compléter la méthode *initialize* et le modèle est prêt :

Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/



XV-A - Paramétrage du modèle

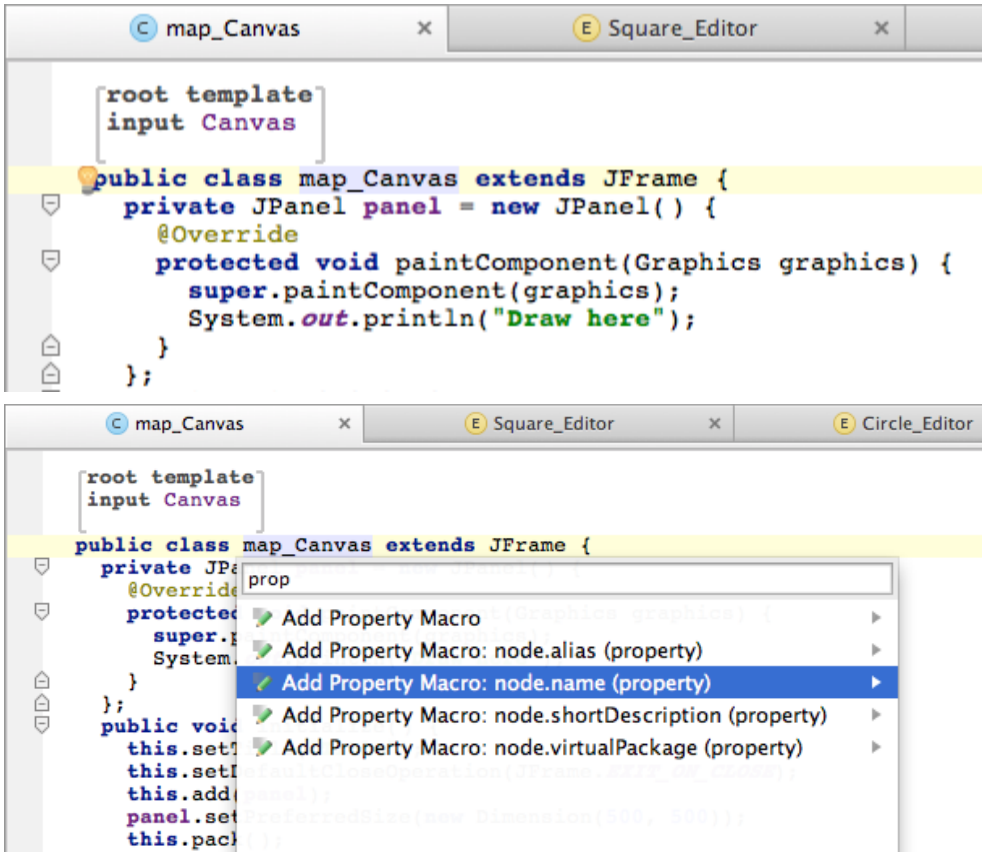
Le modèle ne contient actuellement aucune valeur spécifiée par l'utilisateur. Les propriétés et les enfants de *Canvas* doivent être insérés dans le modèle au moyen de **macros**. MPS dispose de trois types de macros *:*

- **property macros** : pour insérer des propriétés à partir du modèle d'entrée ;
- **node macros** : pour remplacer des nœuds dans le modèle avec des nœuds du modèle d'entrée ;
- **reference macros** : pour ajuster des références dans les modèles afin de pointer vers des nœuds dans le modèle d'entrée.

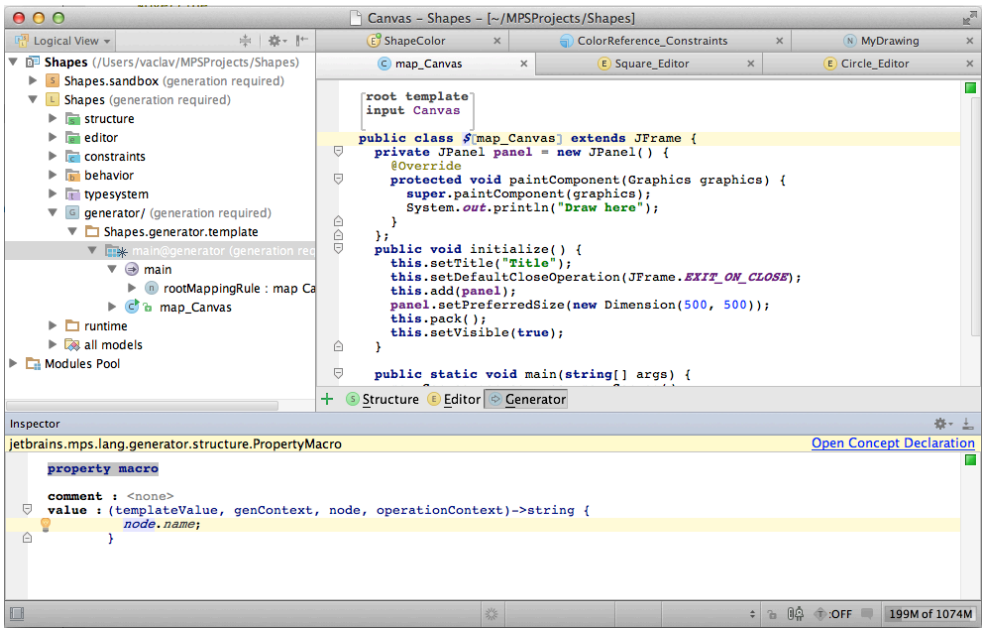
Nous allons utiliser toutes ces macros graduellement.

Pour commencer, nous allons personnaliser le nom de la classe générée et le **titre** de la fenêtre en reprenant le **nom** de l'élément *Canvas*. Placez le curseur sur le nom de la classe : *map_Canvas* et appuyez sur *Alt + Entrée*.

Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/



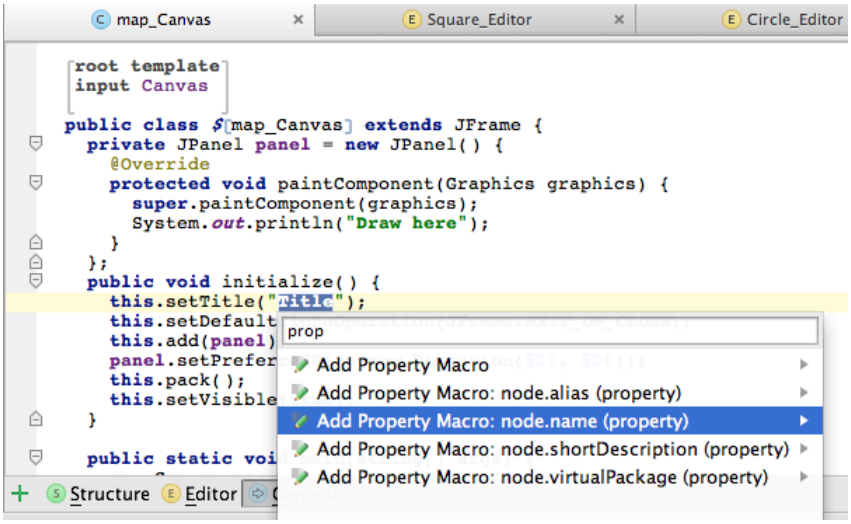
Ensuite, sélectionnez la macro de propriétés **node.name** dans le menu contextuel.



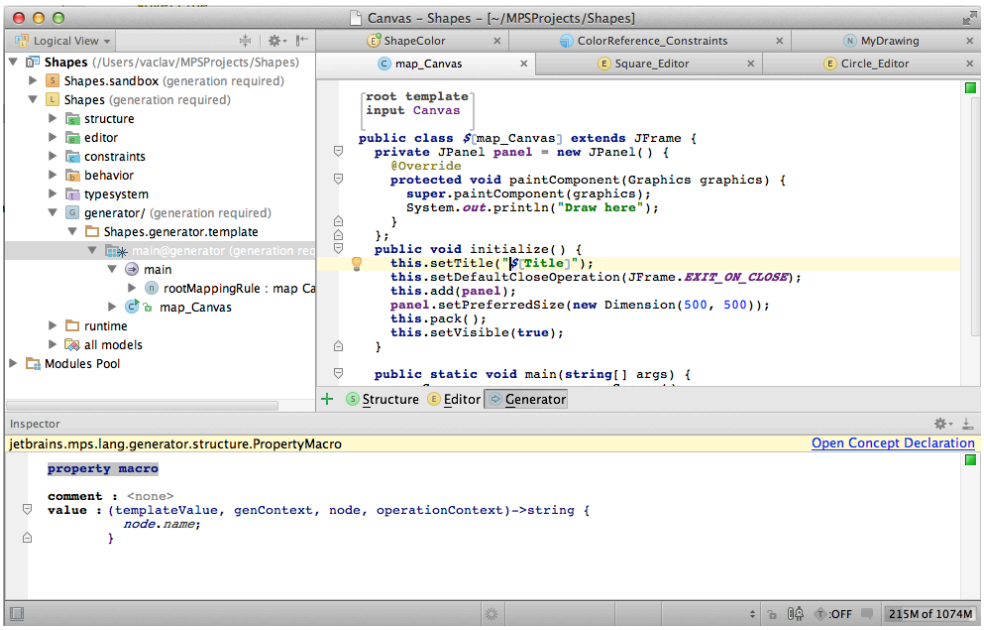
Le texte « map_Canvas » est désormais entouré (annoté) par une **macro de propriétés**, qui remplace la valeur de la propriété **name** par le **nom** du concept *Canvas*. Le panneau **Inspector** (*Alt + 2*) peut également être utilisé pour entrer ou modifier la macro de propriétés. Actuellement, la valeur de *node.name* est renvoyée, ce qui correspond au nom actuel du concept *Canvas*.

Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/

Vous pouvez maintenant entourer le texte « Title » pour personnaliser le titre de la fenêtre. En utilisant le raccourci clavier *Ctrl/Cmde + Flèche haut*, sélectionnez le texte « Title » **sans** ses guillemets, puis utilisez *Alt + Entrée* pour insérer la macro de propriété correcte :



Le code doit prendre la forme suivante :

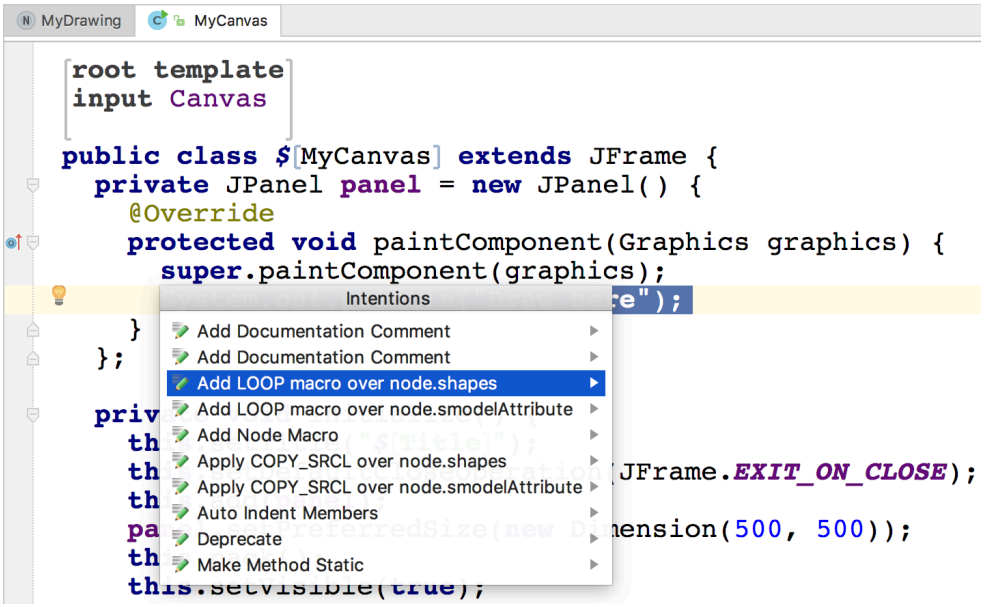


XV-B - Dessiner des formes

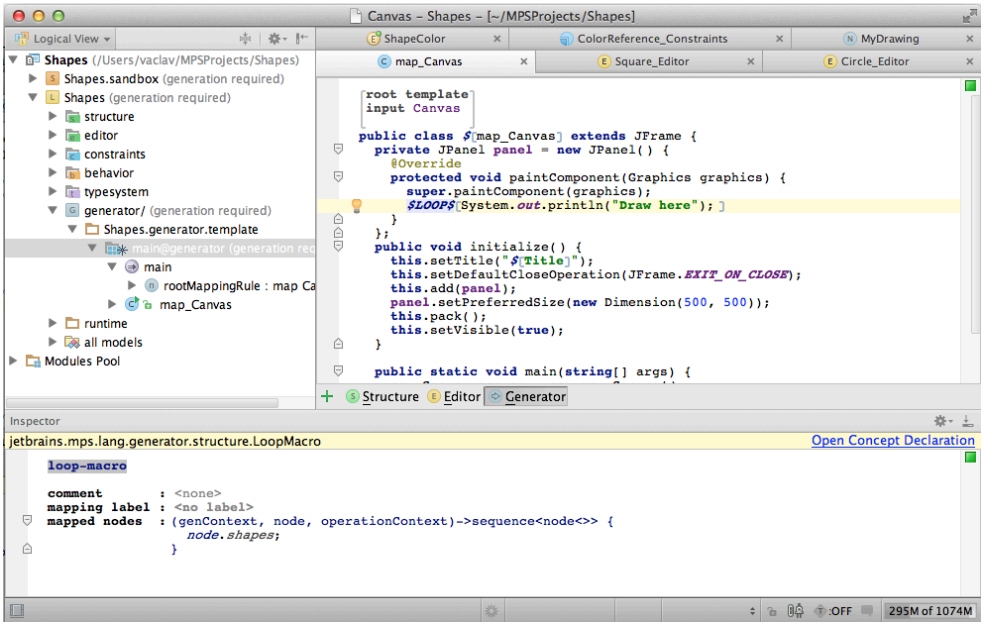
Notre modèle suppose que le code qui dessine les formes doit être placé dans la méthode *paintComponent* du champ *JPanel*. La déclaration « System.out.println("Draw here"); » sert d'emplacement pour le code qui va servir à dessiner les formes. Nous allons utiliser la macro **COPY_SRC** pour remplacer la déclaration d'emplacement par une déclaration dessinant une forme unique. Nous allons utiliser la macro **LOOP** pour répéter cela pour toutes les formes définies dans le concept *Canvas* actuel.

Maintenant, sélectionnez la déclaration d'emplacement en incluant le point-virgule final avec le raccourci *Ctrl/Cmde + Flèche haut*, appuyez sur *Alt + Entrée* et choisissez l'option de **macro de nœuds** la mieux adaptée afin d'insérer une **macro LOOP** pour traiter tous les éléments *Shapes* du concept *Canvas* actuel.

Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/

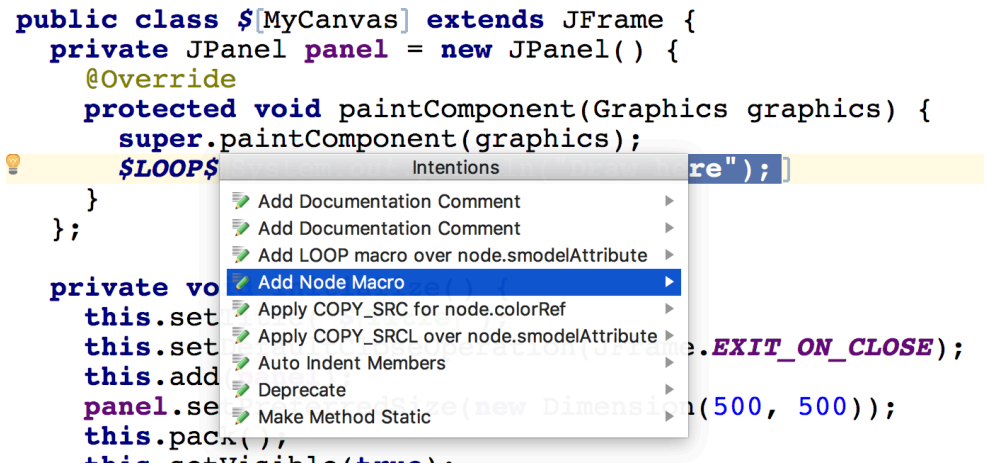


Encore une fois, l'**inspecteur** (*Alt* + *2*) affiche le code de liaison.

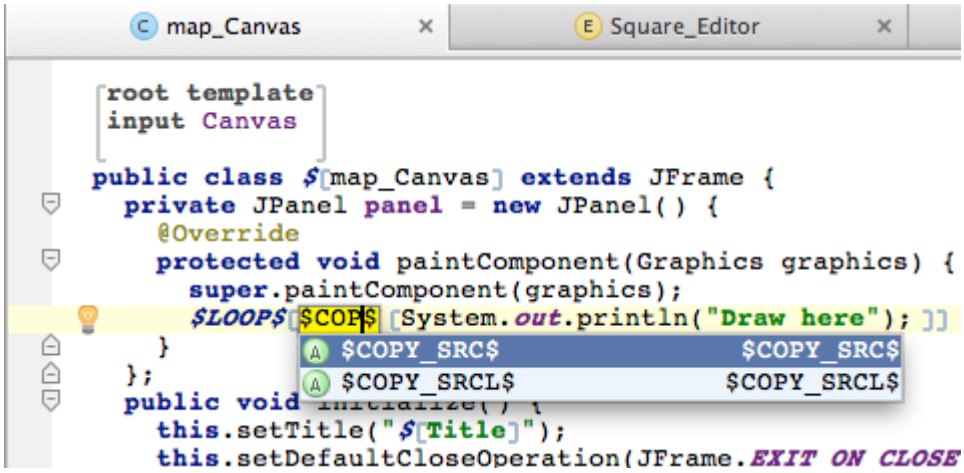


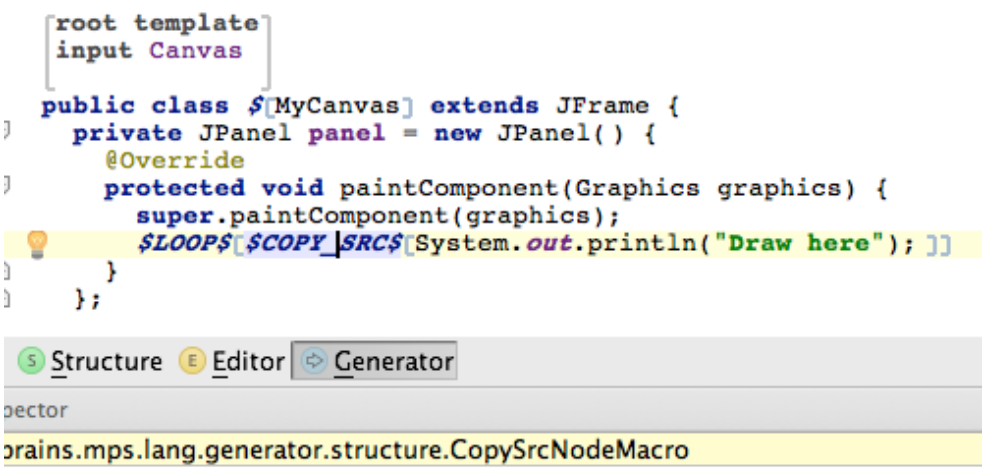
La **macro LOOP** assure l'itération de la déclaration « System.out.println("Draw here"); » pour chacune des formes énumérées dans *node.shapes*. Toutefois, dans notre cas, nous devons remplacer les déclarations « System.out.println("Draw here"); » par le code dessinant chacune de ces formes. La **macro COPY_SRC** fait précisément cela. Sélectionnez à nouveau la déclaration, appuyez sur *Alt* + *Entrée* et choisissez la **macro de nœud**.

Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/



Tapez **COPY_SRC** (*Ctrl* + *Espace*) pour obtenir une macro remplaçant « System.out.println("Draw here"); » par la forme actuelle pour toutes les formes fournies par la **macro LOOP**.

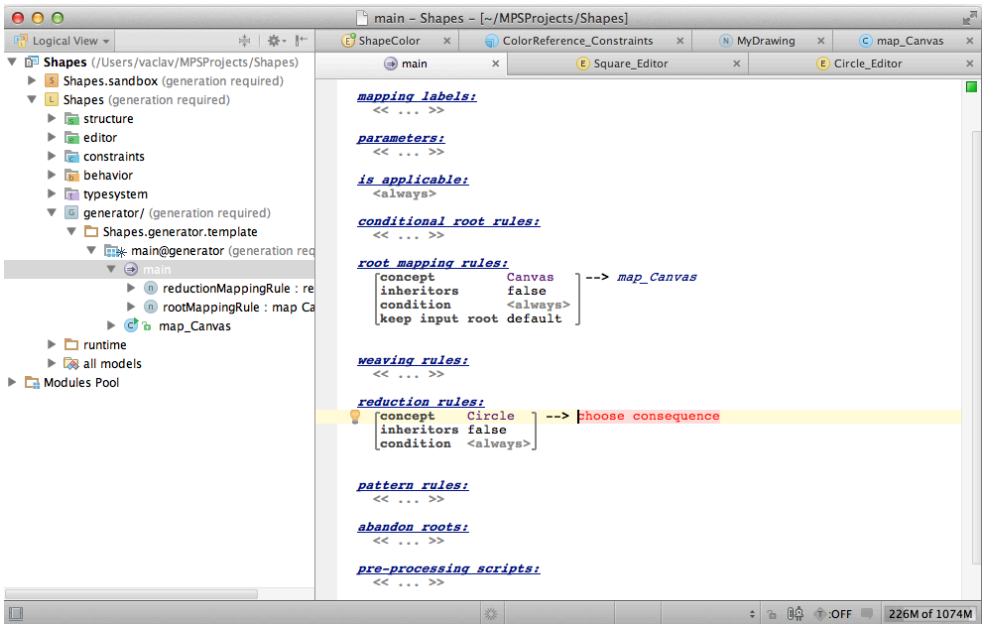




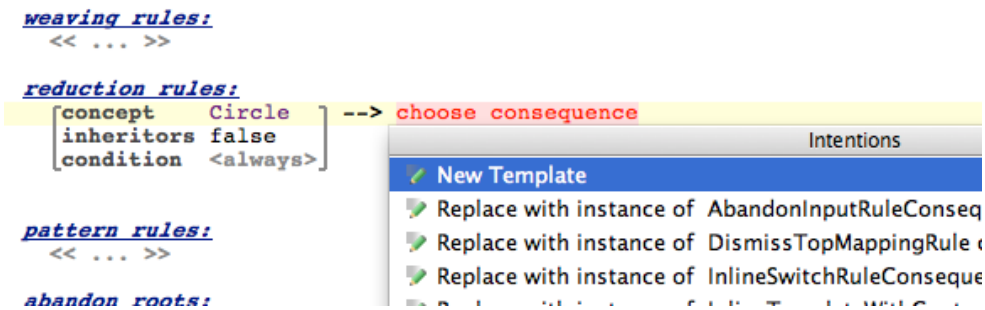
Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/

XV-C - Génération de cercles

Désormais, le concept *Canvas* peut être converti en classe Java et nous avons également prévu un espace pour que les éléments *Shape* puissent ajouter le code indispensable à la génération des formes. Il est donc temps pour nous de définir les règles de conversion à appliquer directement aux *formes*, de sorte qu'une méthode « graphics.drawCircle() » soit insérée dans le code généré en remplacement de la forme *Circle*. Vous devez ouvrir la configuration de mapping *main* et ajouter une entrée dans la section « reduction rules » :

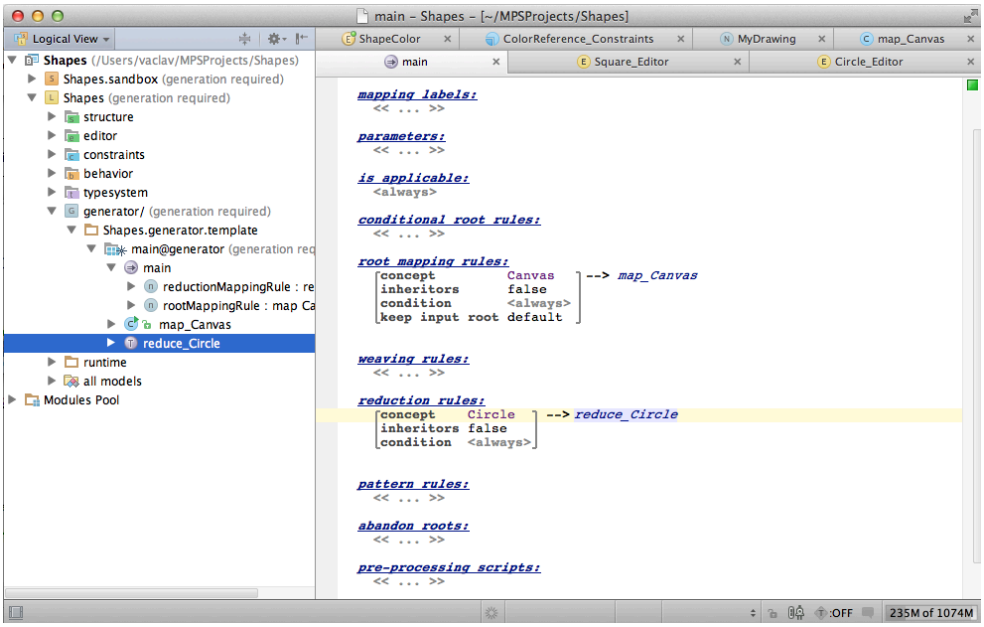


Alt + *Entrée* pour créer un nouveau modèle :

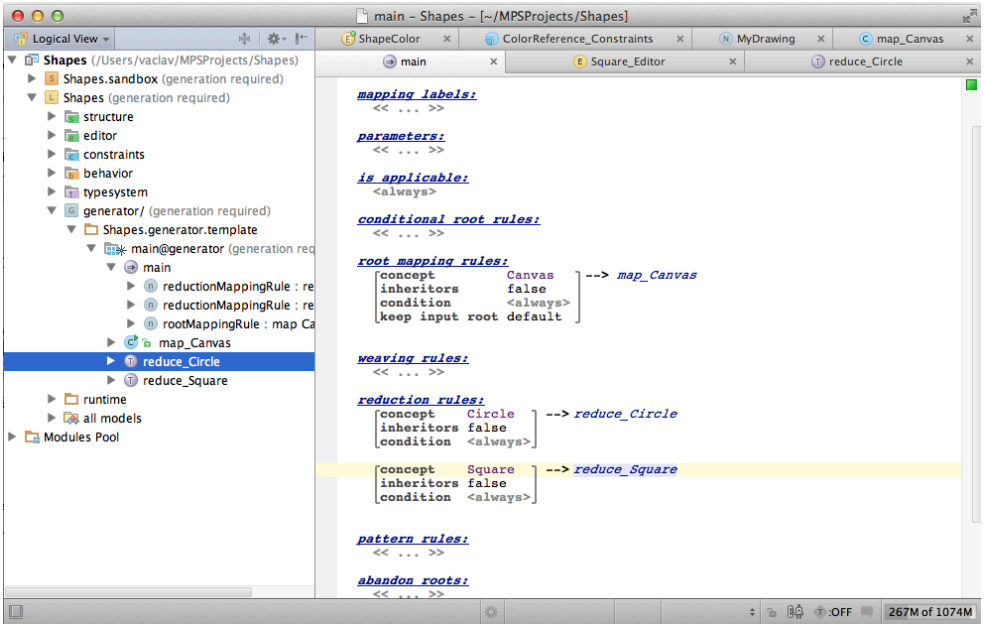


Le nouveau modèle *reduct_Circle* s'affiche dans la **vue de projet** à gauche.

Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/

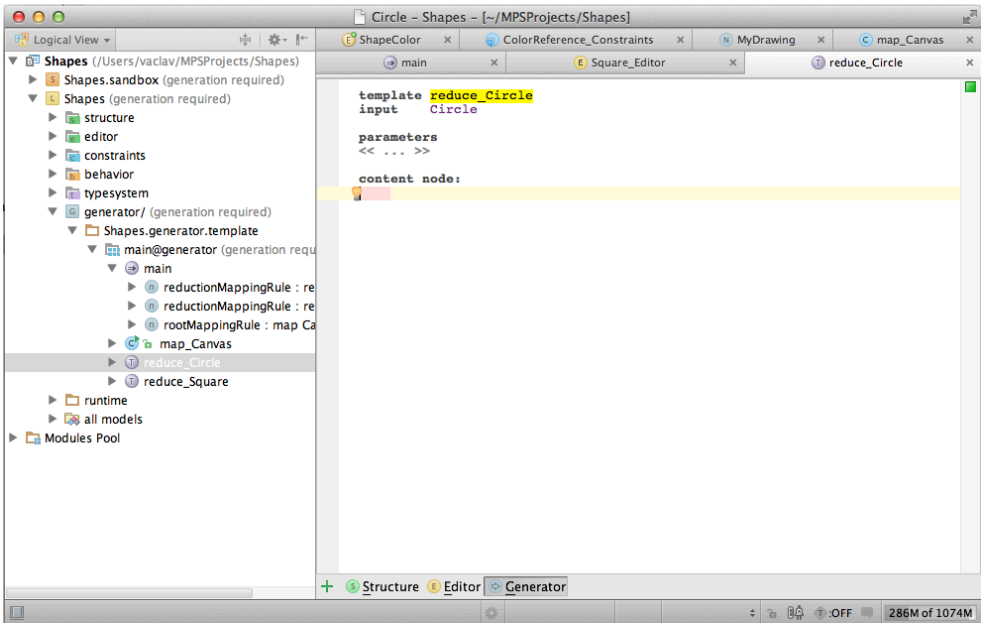


Vous pouvez également créer une règle de réduction pour *Square* :

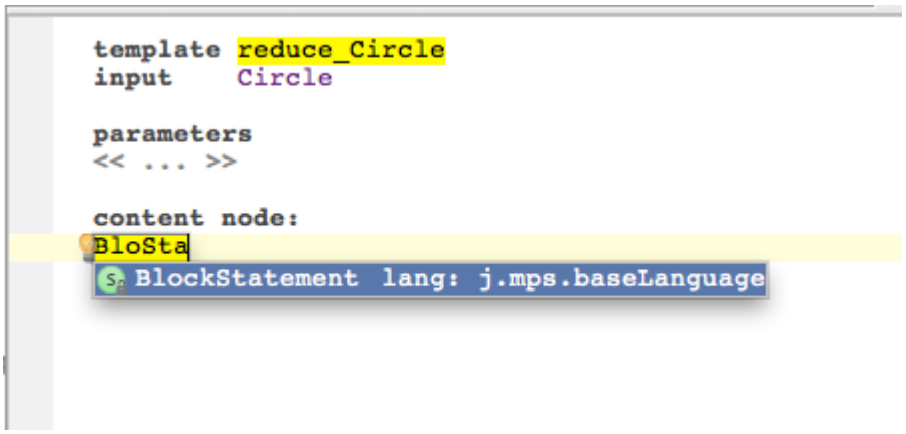


Ouvrez le modèle *reduce_Circle*. Nous devons désormais spécifier le code Java qui doit remplacer *Circles*. Tenez compte du fait que le code Java sera placé dans *map_Canvas*, dans la méthode *paintComponent*.

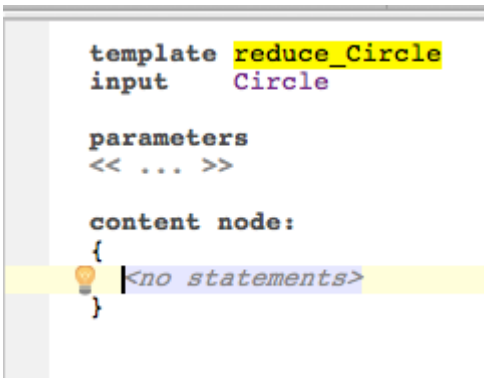
Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/



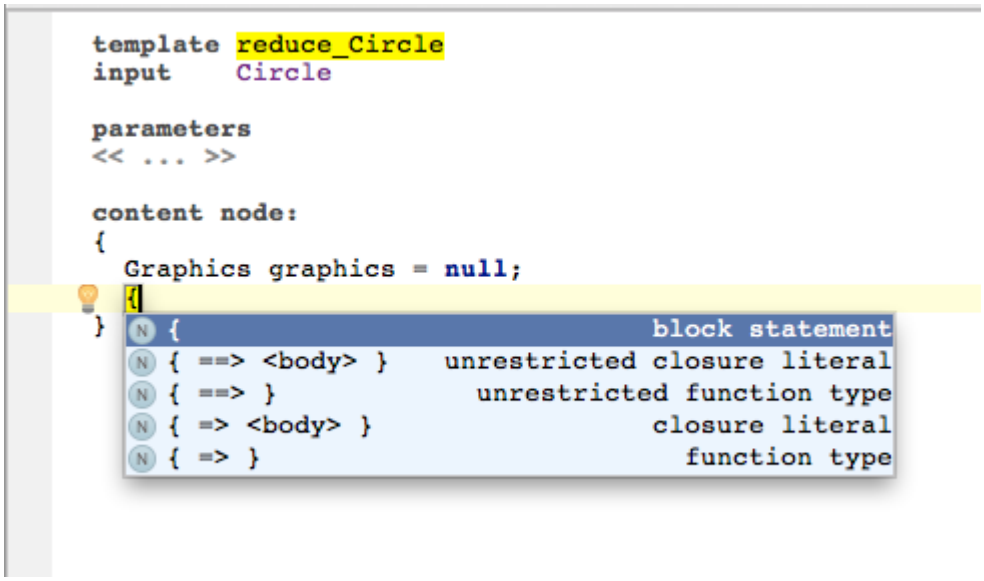
Nous allons commencer par entrer une instruction *BlockStatement* qui va entourer notre modèle :

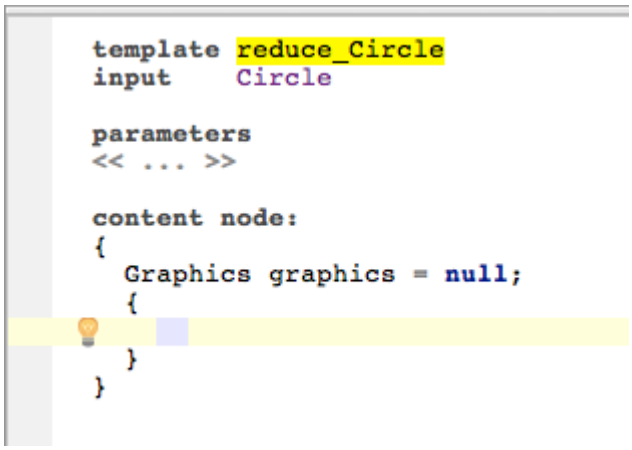


Nous allons avoir besoin d'une variable locale de type *Graphics* pour servir d'emplacement pour le paramètre *paintComponent* du même nom. Et encore une autre entrée *BlockStatement*.



Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/

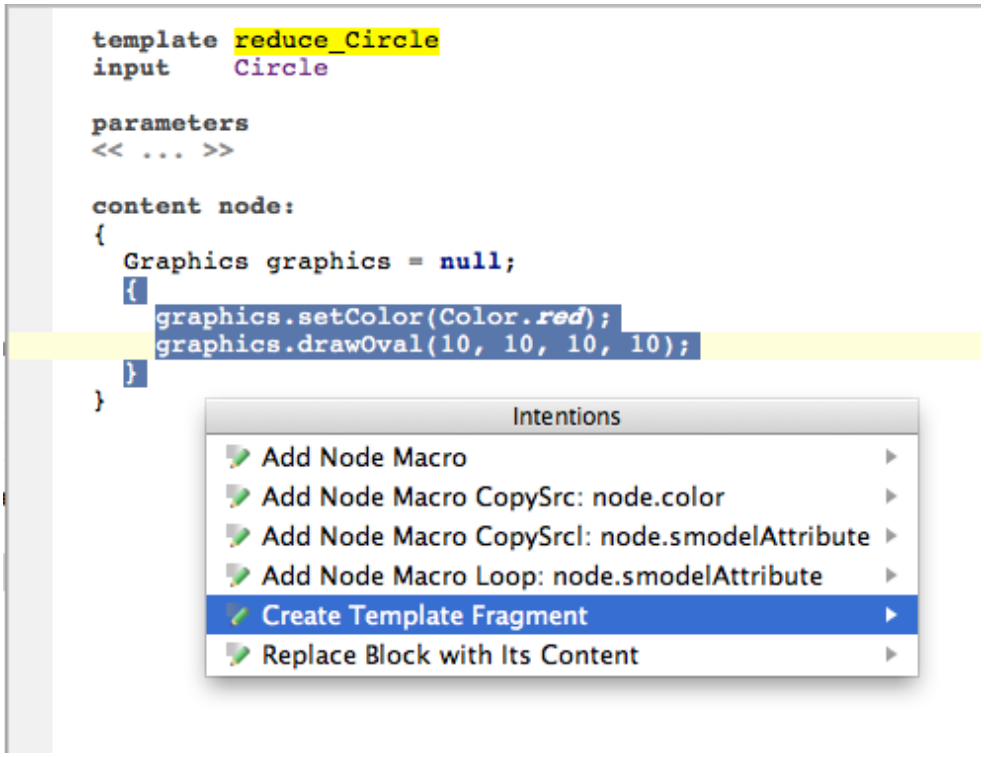




Pour dessiner un cercle dans Java, nous allons utiliser l'objet *Graphics* pour définir la couleur en premier, puis appeler la méthode *drawOval*. Entrez le code ci-dessous :

```
content node:
{
  Graphics graphics = null;
  {
    graphics.setColor(Color.red);
    graphics.drawOval(10, 10, 10, 10);
  }
}
```

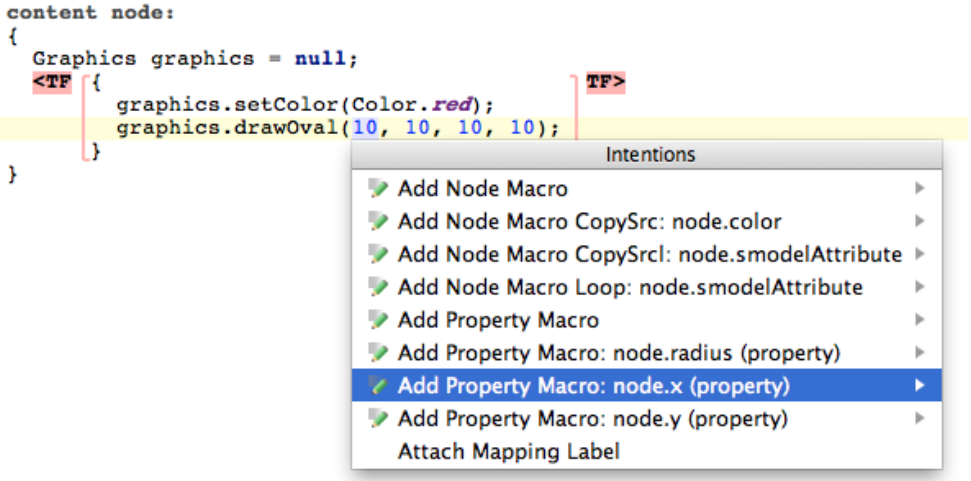

Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/



*La variable locale graphics nécessite une attention particulière. Tel que nous l'avons construit, le modèle actuel repose sur une **correspondance de noms** entre le modèle reduce_Circle, qui fait référence à la variable locale graphics et le modèle map_Canvas qui a défini le paramètre de méthode graphics dans la méthode paintComponent. Cette solution est fonctionnelle, mais assez fragile. Le simple fait de renommer l'une des deux variables graphics entraînera la génération d'un code incomplet. Nous allons mettre en œuvre une solution plus robuste pour **Squares**, juste sous la solution pour **Circle**.*

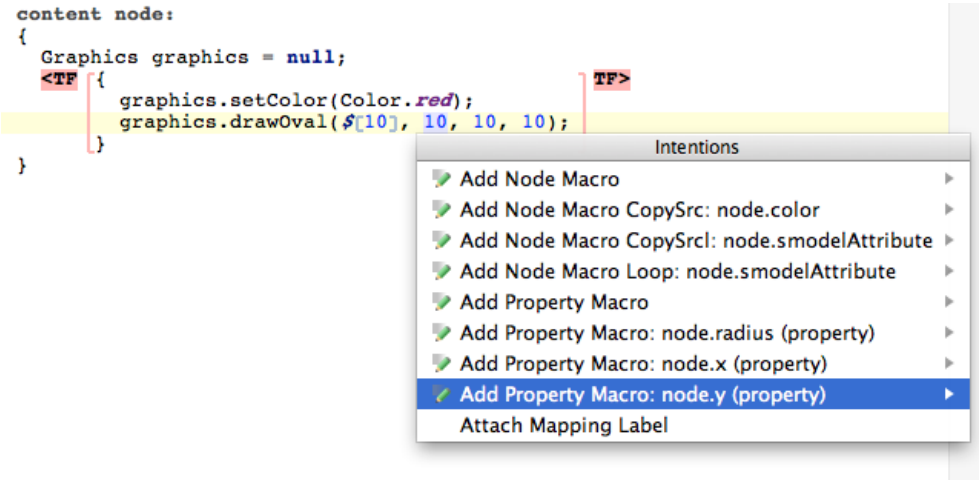
XV-D - Paramétrage

Le code doit maintenant être paramétré avec des valeurs réelles provenant du nœud *Circle*.

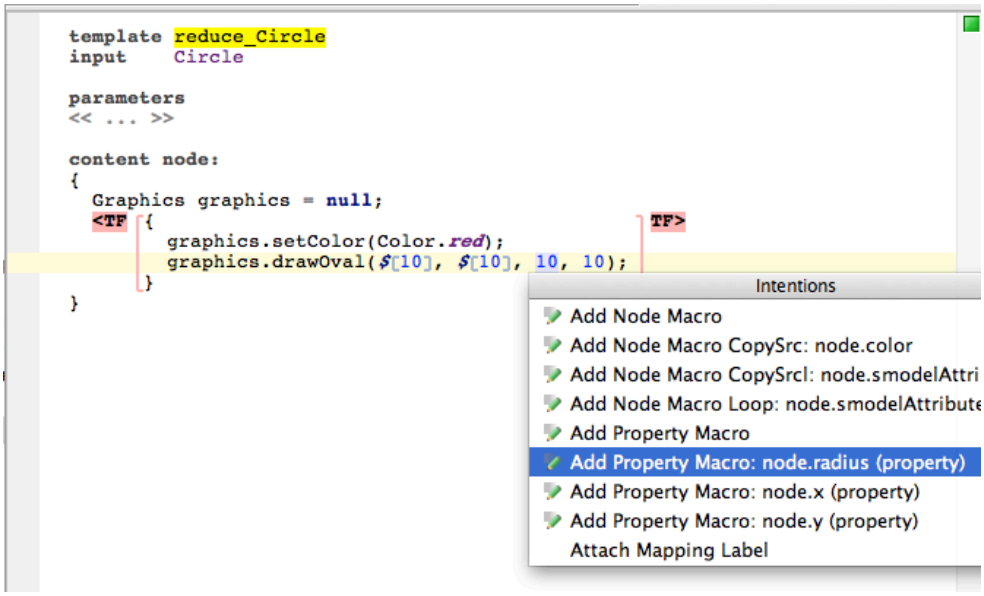


Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/

La première valeur (10) doit être remplacée par les *x* coordonnées du nœud *Circle*. La **macro de propriétés** réalisera cette action. De même, la deuxième valeur « 10 » doit être remplacée par la coordonnée *y*.



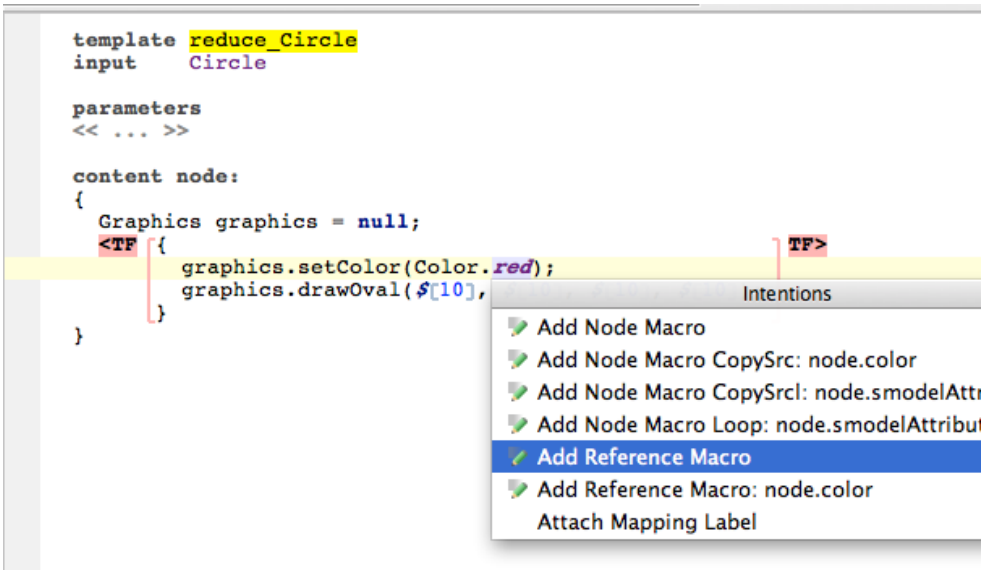
La **troisième** valeur « 10 » et la **quatrième** doivent être toutes les deux remplacées par la valeur *radius* du nœud *circle*.



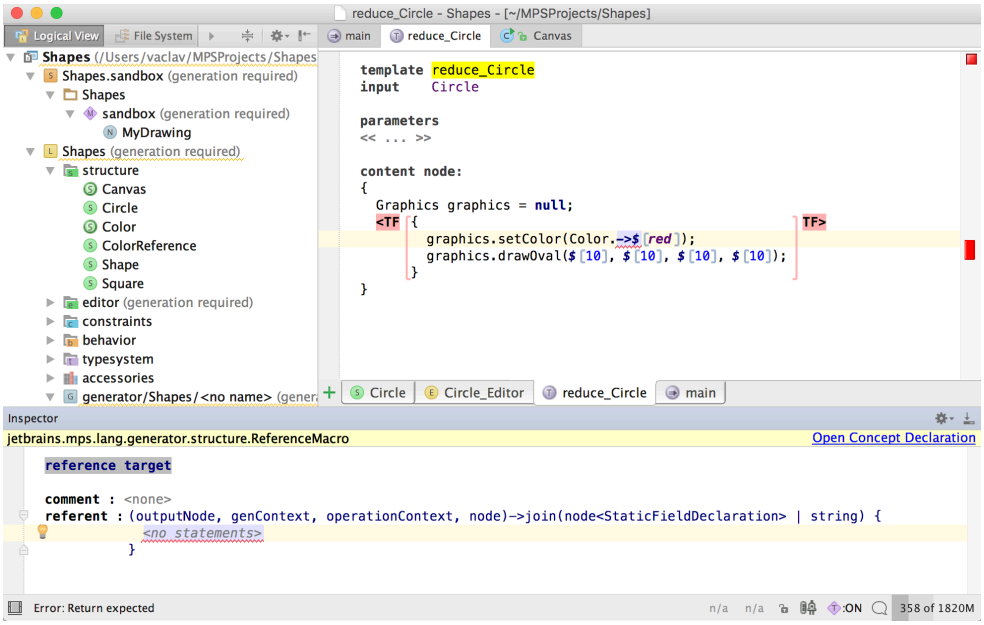
*Le troisième et le quatrième paramètre drawOval() de la méthode correspondent respectivement à la **largeur** et la **hauteur** de la forme ovale. Dans la mesure où nous souhaitons dessiner un **cercle**, nous devons fournir les mêmes valeurs pour les deux arguments. C'est pourquoi les macros de propriétés du **troisième** et du **quatrième** paramètre spécifient toutes les deux node.radius.*

Enfin, la référence de l'emplacement de couleur « Color.red » doit être remplacée par la cible réelle de la référence *color* du nœud *Circle*. Nous allons utiliser une **macro de référence** pour remplacer les références. Placez le curseur sur le mot « red » et appuyez sur *Alt* + *Entrée*.

Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/



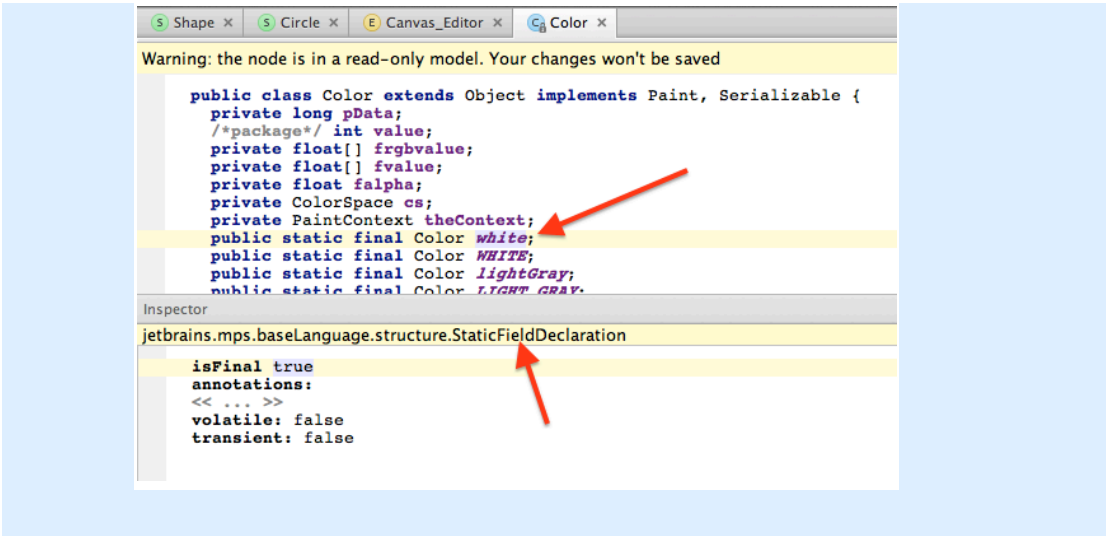
La macro de référence remplace la référence à **red** par une référence au nœud spécifié dans la fenêtre **Inspector** :



La fonction **referent** peut renvoyer une *chaîne* (le nom de la déclaration voulue comme référence) ou un *node<StaticFieldDeclaration>* (un nœud représentant *StaticFieldDeclaration*), car **red** est une référence à *node<StaticFieldDeclaration>*. En fait, toutes les constantes de couleur de la classe *java.awt.Color* sont déclarées en tant que StaticFieldDeclarations.

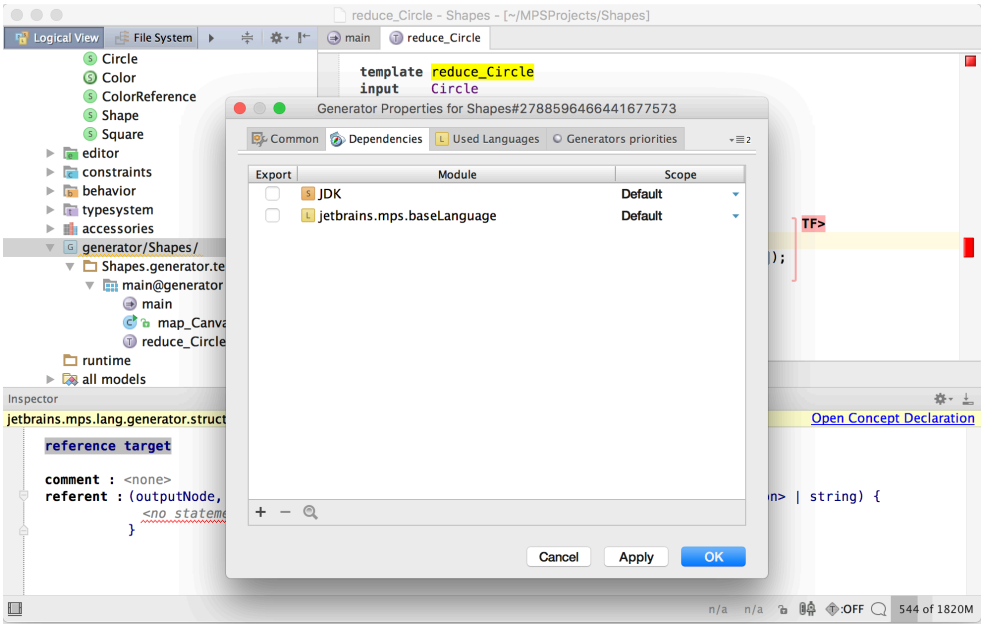
Vous pouvez essayer d'ouvrir la classe java.awt.Color (Ctrl/Cmde + N, tapez « Color » ou cochez la case « Include stubs and non-project models ») pour vérifier, par exemple, si la constante « white » est de type StaticFieldDeclaration.

Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/



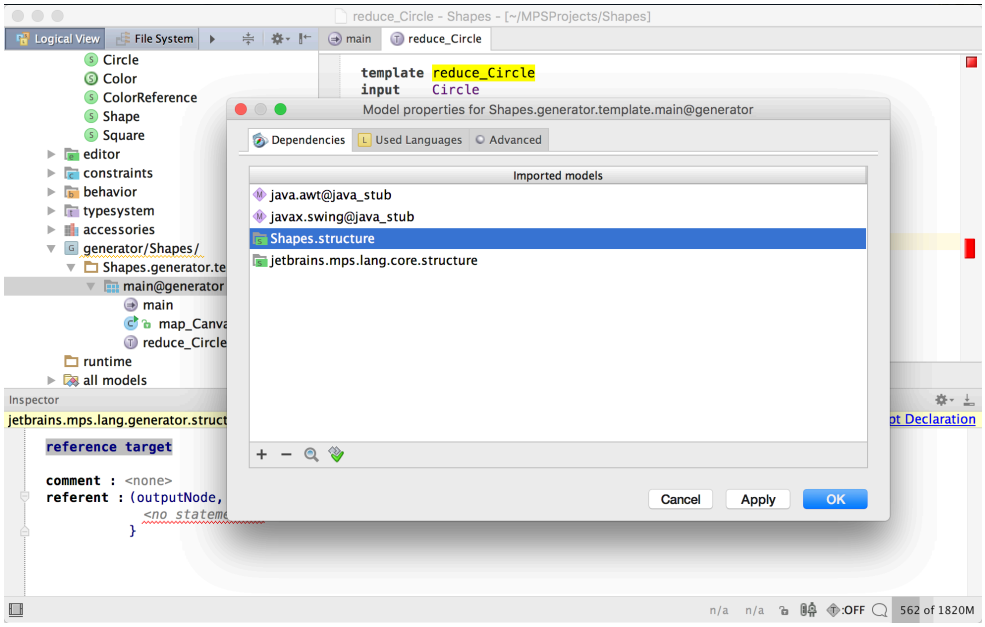
Par conséquent, la macro de référence va nous servir à extraire la section *StaticFieldDeclaration* de la classe *Color* qui correspond à la couleur que le concept *Circle* a définie comme étant sa *couleur* enfant. Nous allons utiliser un programme pour ce faire, nous devons donc définir des dépendances sur les bons modules et modèles, ce qui nous permettra finalement d'écrire le code requis.

Tout d'abord, le module générateur doit dépendre de *BaseLanguage* de façon à pouvoir faire référence au concept *StatifFieldDeclaration* qui est déclaré dans ce langage :

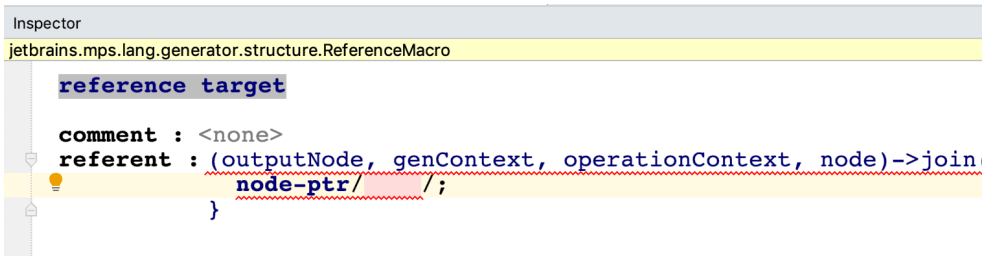


En second lieu, le modèle de générateur :

Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/

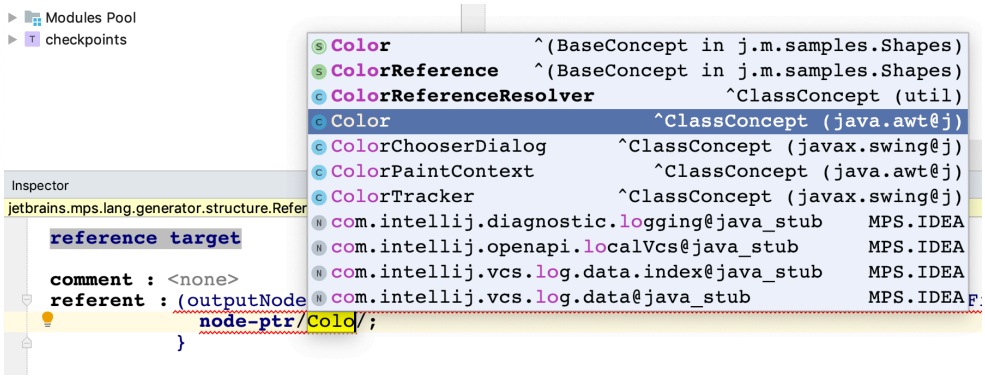


Une fois ces langages importés, nous devrions pouvoir entrer le code qui découvre la déclaration de champ statique correcte dans la classe *Color* :



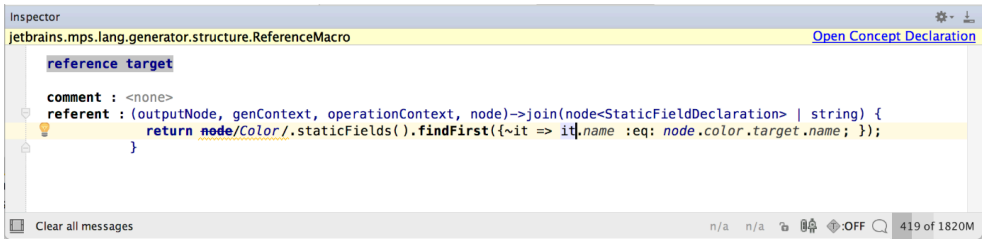
La construction *node/…/* vous permet d'obtenir un nœud dans les modèles importés du concept spécifié, représenté par le nom donné. Comme il n'existe qu'une seule classe *Color* dans le JDK, la référence *node/Color/* est unique et pointe vers la classe *Color* du modèle.

Assurez-vous que vous sélectionnez le bon concept *Color* dans le menu de saisie automatique :



Le langage de *collections* permet de créer une requête concise :

Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/



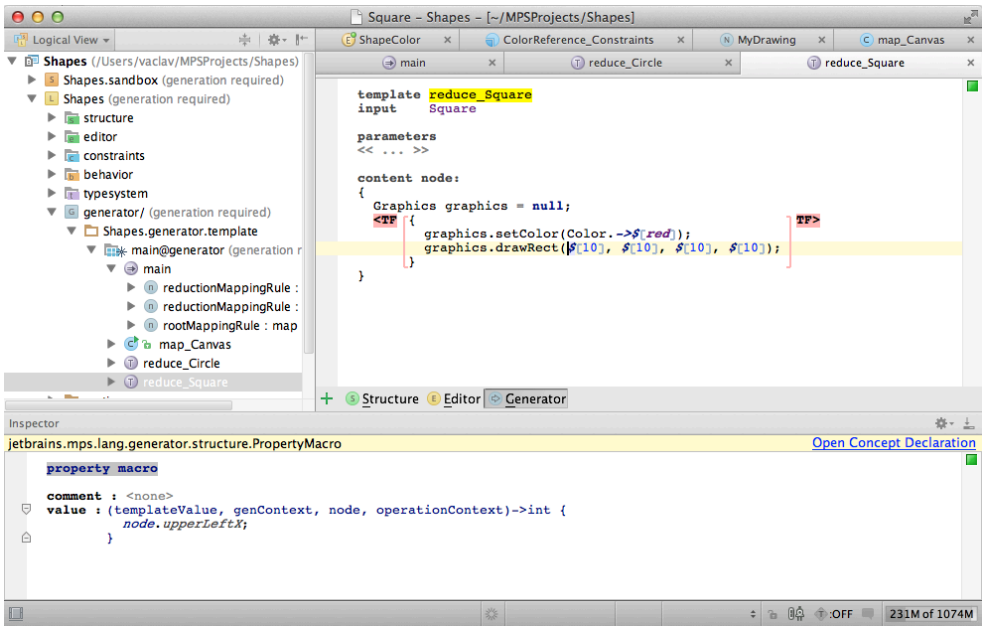
Comme *node* est une instance du concept *Circle*, *node.color* désigne la référence de couleur du cercle (une instance de *ColorReference*) et *node.color.target* est un élément *Color* (une instance du concept Shapes.Color) provenant du **modèle d'accessoires**.

En résumé, la requête recherche la première déclaration de champ statique dans les déclarations de champ statique de la classe *java.awt.Color* ayant le même nom que la couleur spécifiée pour le concept *Circle*.

L'opérateur :eq: effectue une comparaison d'égalité null-safe, équivalente à (it.name !=null && it.name.equals(node.color.target.name)). MPS propose également un opérateur :ne:, qui est une négation de :eq:.

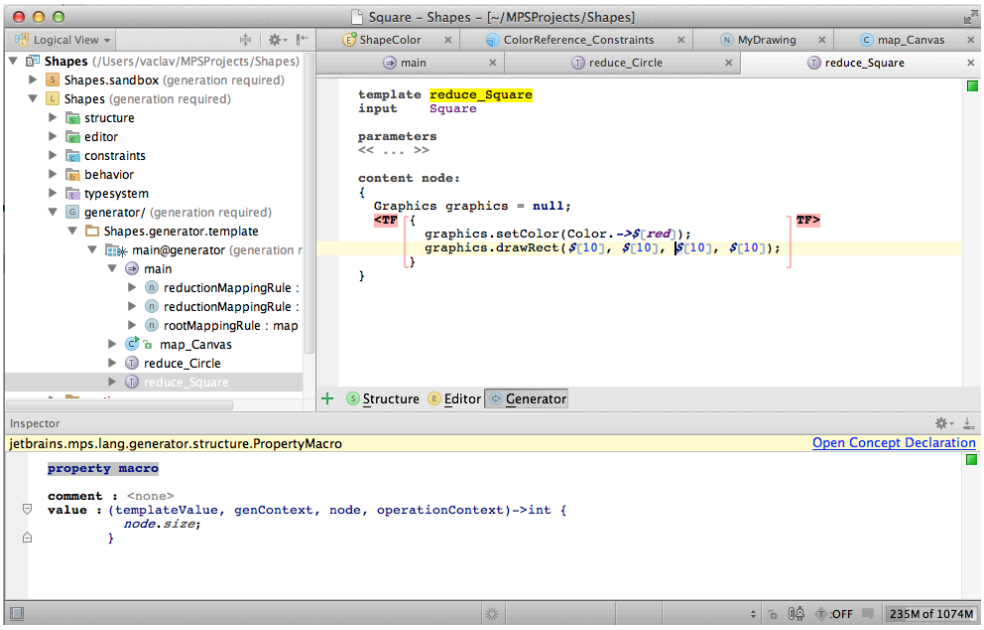
XV-E - Réduction des carrés

Nous allons commencer par reproduire le processus de génération de *Circles*. Utilisez le code suivant à l'identique pour le modèle *reduce_Square :*



Les valeurs passées dans « drawRect » doivent être remplacées par les propriétés *upperLeftX*, *upperLeftY* et *size* du concept *Square* au moyen de **macros de propriétés**.

Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/



*Le troisième et le quatrième paramètre drawRect() de la méthode correspondent respectivement à la **largeur** et la **hauteur** de la forme rectangulaire. Dans la mesure où nous souhaitons dessiner un **carré**, nous devons fournir les mêmes valeurs pour les mêmes arguments. C'est pourquoi les macros de propriété du **troisième** et du **quatrième** paramètre spécifient toutes les deux size.*

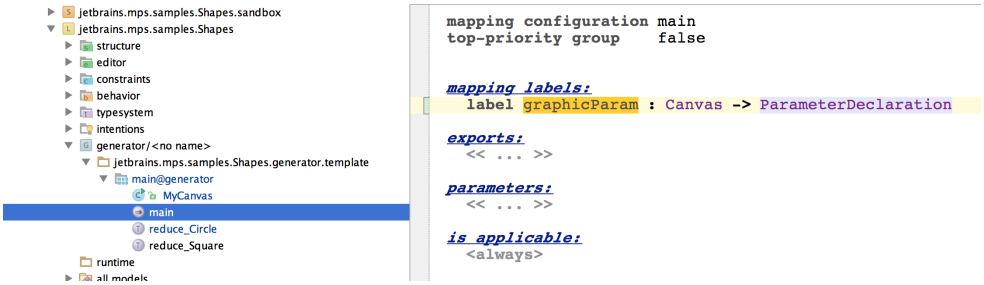
XVI - Une génération plus robuste des carrés

Comme mentionné plus haut, dans la section sur le générateur *Circle*, nous allons utiliser le modèle *reduce_Square* pour lier correctement la variable locale *graphics* avec le paramètre *graphics* généré par le modèle *map_Canvas*. Il serait imprudent de se reposer uniquement sur une correspondance de nom.

Nous devons procéder en trois étapes :

1 Définir le stockage des paramètres *graphics* créés ;
2 Stocker le paramètre *graphics* dans le modèle *map_Canvas* ;
3 Récupérer le bon paramètre *graphics* dans le modèle *reduce_Square.*

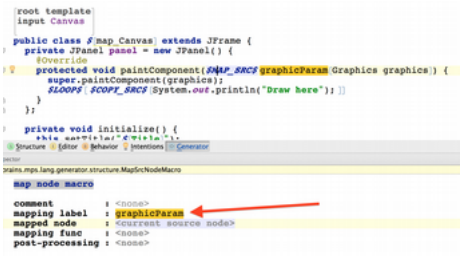
Nous allons commencer par créer une étiquette de mapping dans la configuration de mapping. Cela servira de stockage pour *ParameterDeclarations*, chacun étant identifié par le concept *Canvas* ayant servi à sa génération. Cette étiquette de mapping peut également être considérée comme un dictionnaire servant à mapper les concepts *Canvas* aux paramètres *ParameterDeclarations*.



Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/

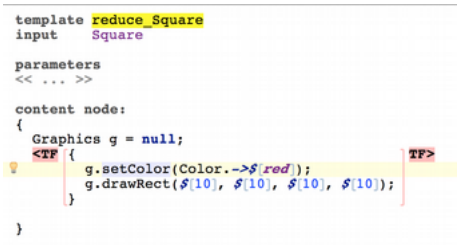
L'étiquette de mapping *graphicsParam* stocke les entités *ParameterDeclarations* mappées en fonction des concepts *Canvas* auxquels elles appartiennent.

Le modèle *map_Canvas* doit désormais stocker le paramètre *graphics* dans l'étiquette de mapping. La macro *MAP_SRC* peut être utilisée à cette fin avec beaucoup d'efficacité :

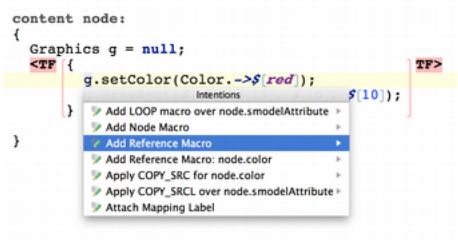


Entourez la déclaration de paramètre (en incluant le type) avec la macro *MAP_SRC* (*Alt + Entrée*, sélectionnez une *macro de nœud* et tapez MAP_SRC_). *Dans la fenêtre _Inspector* en bas de l'écran, sélectionnez l'**étiquette de mapping** à utiliser pour stocker la déclaration de paramètre générée. Le **nœud source actuel**, qui est une instance du concept *Canvas*, servira de clé pour identifier la déclaration de paramètre *graphics* dans l'étiquette de mapping.

Enfin, nous devons extraire la déclaration de paramètres de l'étiquette de mapping dans le modèle *reduce_Square*. Afin d'indiquer clairement que nous ne dépendons plus de la correspondance de nommage, nous pouvons utiliser un nom de variable autre que *graphics*. Nous allons employer *g* dans l'exemple :



Si vous utilisez le raccourci *Alt + Entrée* pour la référence *g* et choisissez une *macro de référence*, vous pouvez extraire le bon paramètre *graphics* de l'**étiquette de mapping** :



Répétez ensuite le processus de création de macros de référence pour la deuxième référence *g*. **Les deux** macros de référence attachées à la référence de variable *g* que vous venez de créer doivent spécifier dans la fenêtre **Inspector** le détail du processus d'obtention de la cible voulue de la référence. L'objet *genContext* donne accès à des méthodes et des propriétés utiles de la session de génération en cours. Utilisez l'opération « **get output for label and input** » sur *genContext* et fournissez l'étiquette de mapping *graphicsParam*, ainsi que le concept *Canvas* contenant le concept *Square* en cours de génération.

Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/

```
template reduce_Square
input    Square

parameters
<< ... >>

content node:
{
  Graphics g = null;
  <TF {
      ->$[g].setColor(Color.->$[red]);
      ->$[g].drawRect($[10], $[10], $[10], $[10]);
    }
}
```

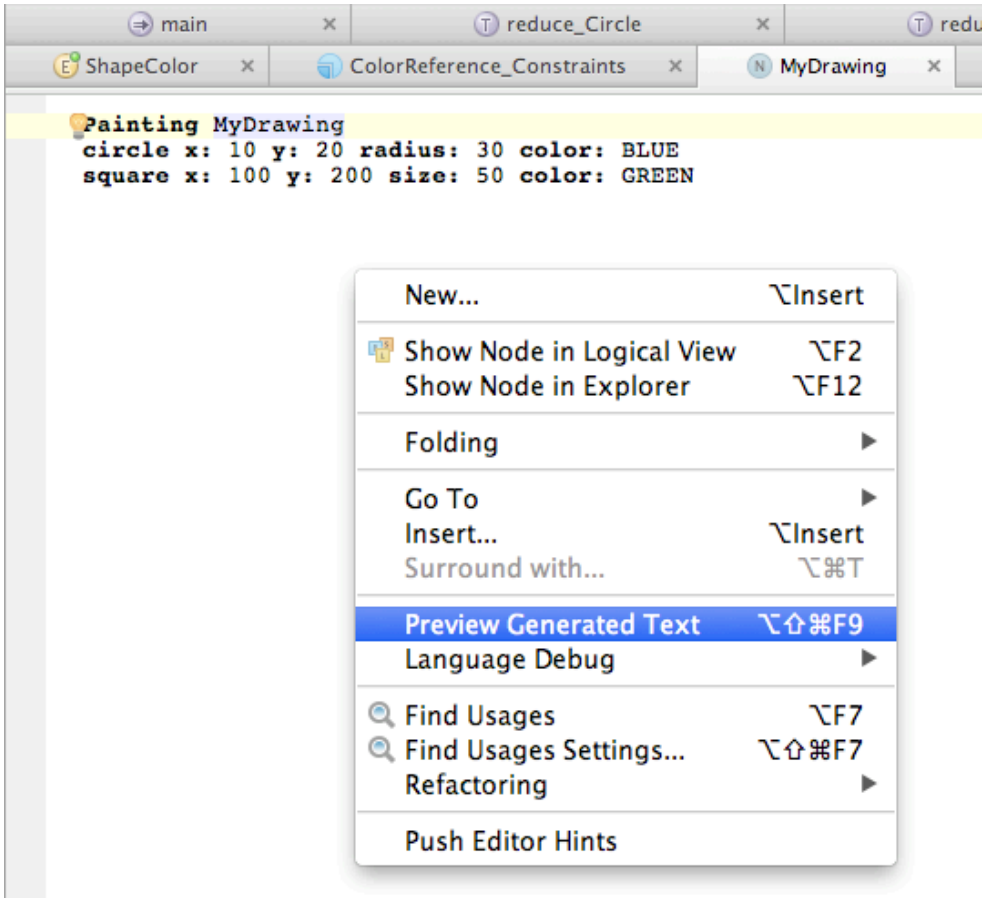
Structure | Editor | Constraints | Behavior | Generator

```
jrains.mps.lang.generator.structure.ReferenceMacro

reference target

comment : <none>
referent : (outputNode, genContext, operationContext, node)->join(node<VariableDeclaration> | string) {
             genContext.get output graphicParam for (node.parent : Canvas);
           }
```

Nous sommes désormais en mesure de répliquer également la récupération du paramètre *graphics* pour le modèle *reduce_Circle*.

XVII - Génération de code

Nous venons de terminer la définition du générateur. Si vous recréez la build de langage, ouvrez *MyDrawing*, faites un clic droit dessus et choisissez « Preview Generated Text ».



Vous obtenez du code Java bien structuré qui initialise correctement une *Jframe* et dessine toutes les formes :

Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/

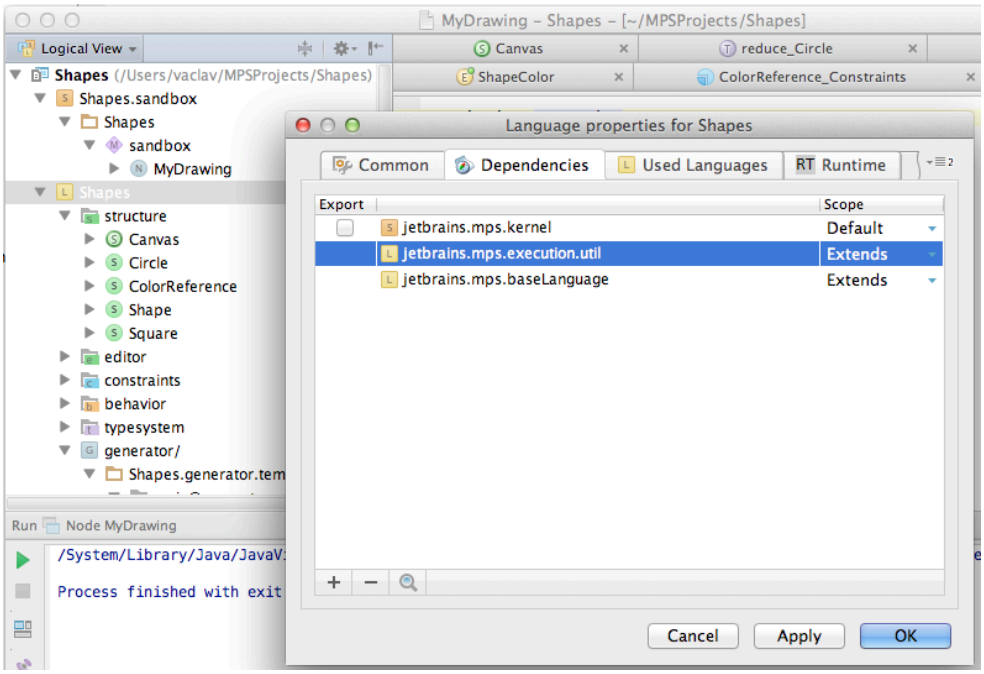
ShapeColor | ColorReference_Constraints | MyDrawing | MyDrawing.java

```
/*Generated by MPS */

import javax.swing.JFrame;
import javax.swing.JPanel;
import java.awt.Graphics;
import java.awt.Color;
import java.awt.Dimension;

public class MyDrawing extends JFrame {
  private JPanel panel = new JPanel() {
    @Override
    protected void paintComponent(Graphics graphics) {
      super.paintComponent(graphics);
      graphics.setColor(Color.BLUE);
      graphics.drawOval(10, 20, 30, 30);
      graphics.setColor(Color.GREEN);
      graphics.drawRect(100, 200, 50, 50);
    }
  };

  public void initialize() {
    this.setTitle("MyDrawing");
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    this.add(panel);
    panel.setPreferredSize(new Dimension(500, 500));
    this.pack();
    this.setVisible(true);
  }


  public static void main(String[] args) {
    MyDrawing canvas = new MyDrawing();
    canvas.initialize();
  }
```

XVIII - Exécution du code

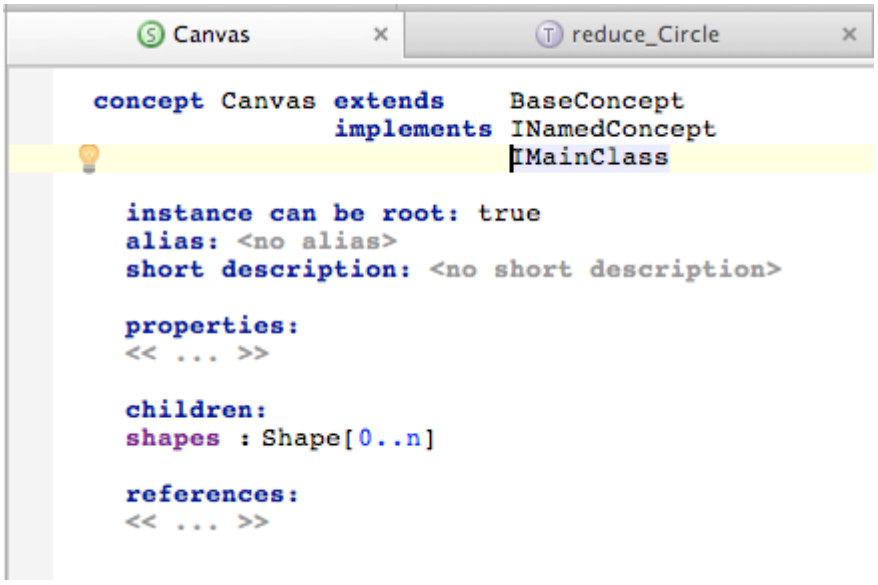
Il est certes intéressant de voir le code généré, mais il est préférable de le voir s'exécuter. MPS peut compiler et exécuter facilement le code Java généré. Nous devons seulement indiquer que *Canvas* est généré dans une classe Java exécutable et que *Canvas* doit donc être traité en tant que classe exécutable ou principale (« main »). Il nous reste simplement à demander à *Canvas* d'implémenter l'interface *ImainClass* et MPS s'occupe du reste. L'interface *ImainClass* provient du langage *jet brains.mps.execution.util* et nous devons maintenant l'ajouter à la liste de dépendances de notre langage :

Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/



Utilisez le raccourci *Alt + Entrée* pour accéder à la boîte de dialogue des propriétés. Vous pouvez constater que le langage doit être marqué comme **Extends**.

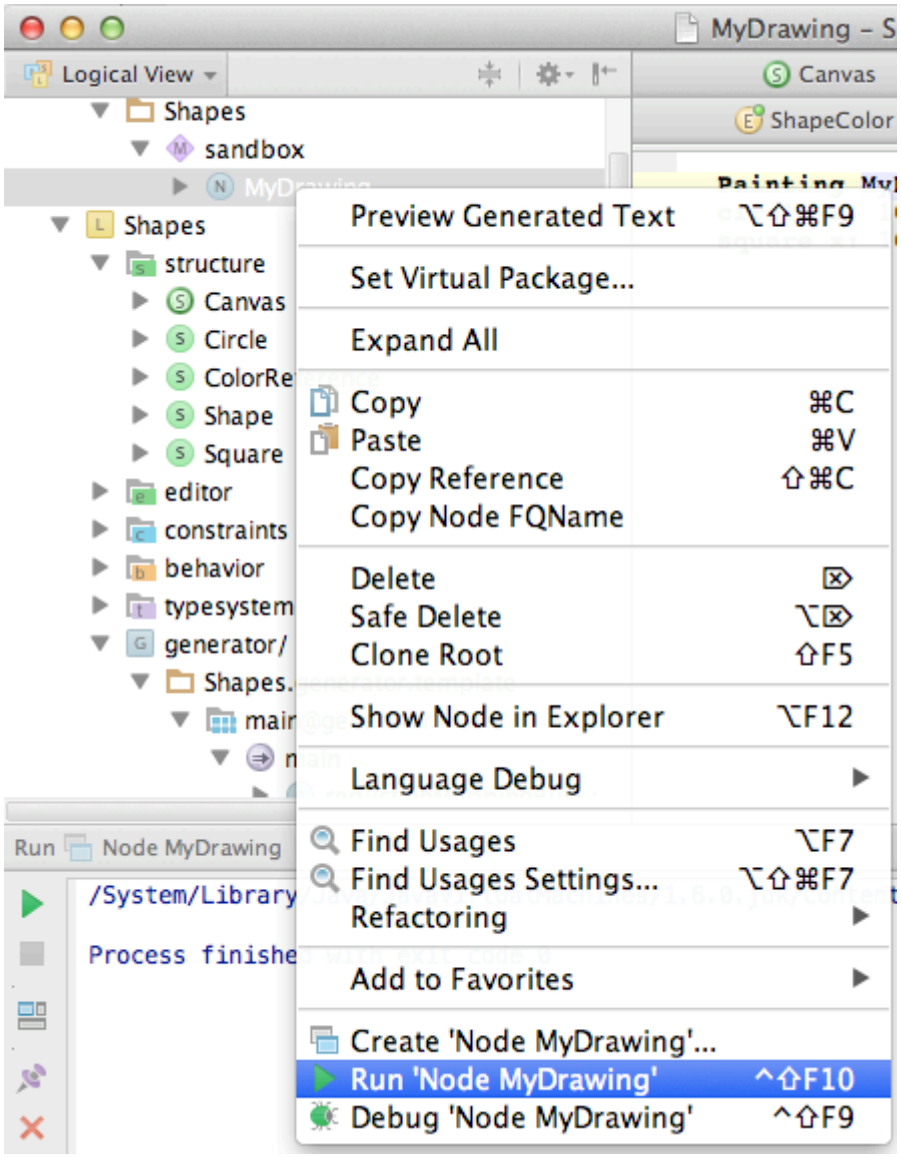
Il est maintenant possible d'ajouter l'interface *ImainClass* dans la section *implements* du concept *Canvas*.



Générez une nouvelle build de langage, puis faites un clic droit sur *MyDrawing* dans la **vue de projet** et cliquez sur « Run ».

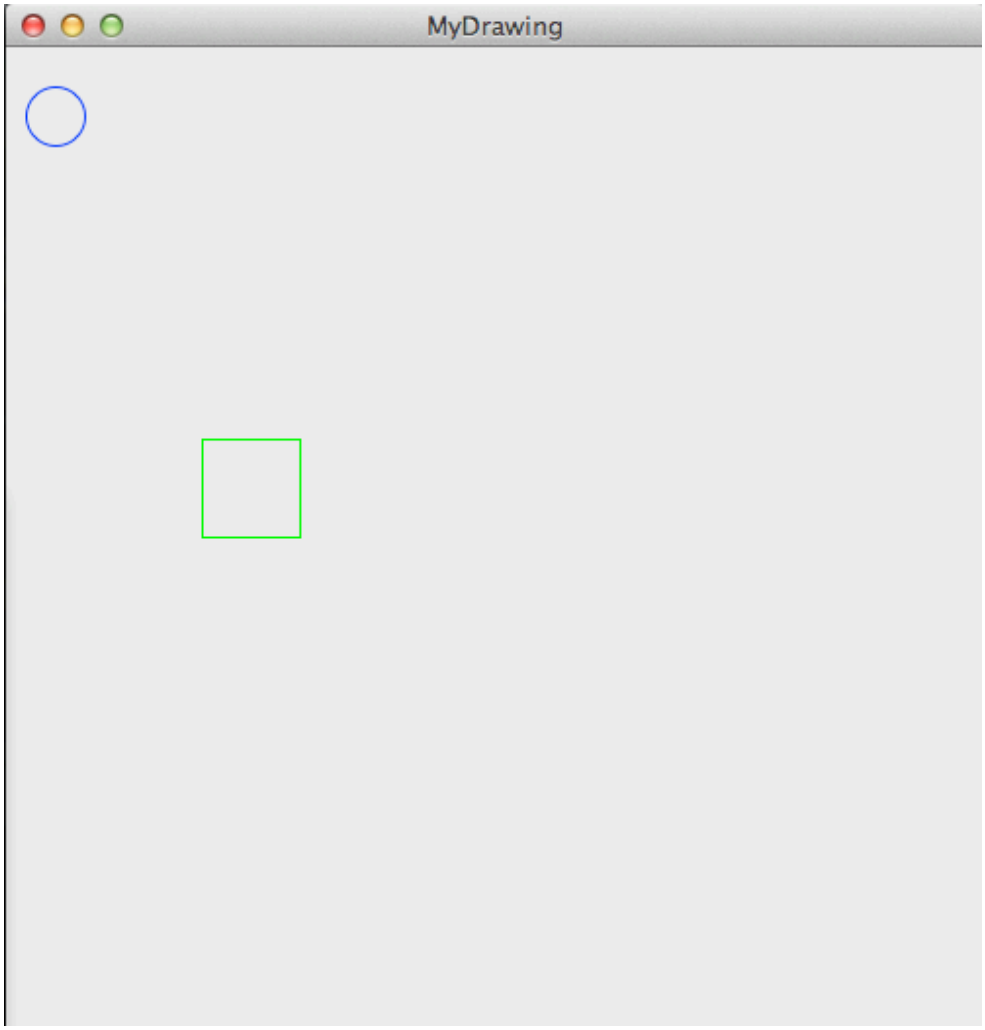
Une fois le nouveau dessin créé, vous pourriez avoir besoin de le compiler. Si l'option Run 'Node MyDrawing' ne s'affiche pas dans le menu contextuel, cela indique que MyDrawing n'a pas encore été compilé. Faites un clic droit sur la solution sandbox ou sur le modèle contenant MyDrawing et choisissez Make. Dès que cette opération est terminée, vous devez être en mesure de réaliser votre dessin.

Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/



En récompense de vos efforts, vous obtenez une application Java avec votre dessin.

Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/



XIX - Que faire ensuite

Félicitations ! Vous venez de réaliser les différentes étapes de notre tutoriel d'introduction à MPS.

Ce tutoriel vous a fourni tous les éléments nécessaires pour ajouter des formes dans votre langage. Il peut aussi être intéressant d'ajouter des points, des lignes, des triangles, des rectangles ou des formes avec des trames de couleur à notre langage.

Si vous souhaitez approfondir votre connaissance de MPS, nous vous invitons à suivre notre **cours en ligne**, qui explore la plupart des concepts avancés et couvre la structure, l'éditeur, le générateur, la génération de texte et les contraintes de la définition de langage, ainsi que le langage de build et les plug-ins de langage.

La version anglaise de ce tutoriel est disponible sur le site de JetBrains : **https://www.jetbrains.com/help/mps/shapes-an-introductory-mps-tutorial.html**

Copyright ® 2019 Aleksey Makarkin et Oscar Rodriguez. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://jetbrains.developpez.com/tutoriel/shapes-introduction-mps/