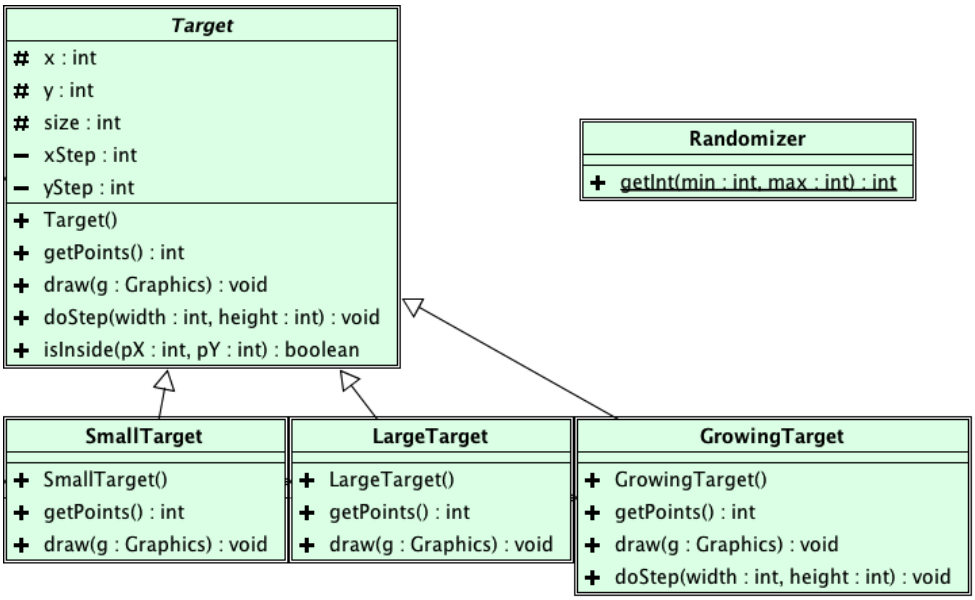
Après votre login, renommez le répertoire **NomPr123** par votre code IAM. Créez le projet de l'exercice (projet **QuickClick**) dans ce répertoire. Une version exécutable du programme qui vous servira de référence se trouve dans le dossier **dist**.



Dans la suite, vous allez développer le petit jeu **QuickClick** qui consiste à tirer (cliquer) sur des cibles carrées en mouvement. Les cibles sont définies selon le diagramme UML suivant :

`x`, **`y`** les coordonnées du **centre** de la cible, initialisées à des valeurs choisies au hasard entre 30 et 300

`size` la largeur d'un côté de la cible. Les cibles ont différentes largeurs selon leur type :

`SmallTarget`	largeur : 20 pixels
`LargeTarget`	largeur : 40 pixels
`GrowingTarget`	la largeur change à chaque pas : elle varie de 5 à 50 pixels, puis recommence à 5 pixels et ainsi de suite, en ajoutant chaque fois 5 pixels

`xStep`, **`yStep`** définissent les pas du déplacement en horizontale et en verticale.

`xStep`	positif ↔ déplacement vers la droite;	négatif ↔ vers la gauche
`yStep`	positif ↔ déplacement vers le bas;	négatif ↔ vers le haut
`xStep` et **`yStep`**	sont initialisés à des valeurs aléatoires entre -5 et 5	

`getPoints`() retourne le nombre de points reçus (→ score) si on a touché l'objet :

`SmallTarget`	:	20 points
`LargeTarget`	:	10 points
`GrowingTarget`	:	calculée en fonction de la largeur : **`(50 - size)`** points

`draw`(...) dessine l'objet sur le canevas (**`SmallTarget`** en vert, **`LargeTarget`** en jaune, **`GrowingTarget`** en orange). Le nombre actuel de points est affiché au milieu de l'objet.

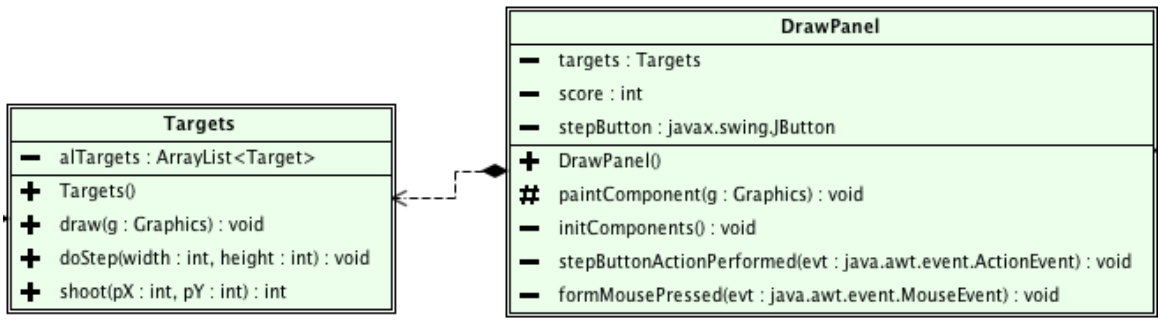
`doStep`(...) déplace l'objet d'un pas (→ **`xStep`**, **`yStep`**). La méthode obtient comme paramètres la largeur et la hauteur du canevas sur lequel se trouve la cible. Ainsi **`doStep`** sait faire rebondir la cible sur les bords **sans qu'elle ne sorte ou déborde du canevas**. Si au départ ou après avoir redimensionné le canevas, une cible se trouve à l'extérieur du canevas, elle va y retourner automatiquement.

`isInside`(...) retourne **true** ssi les coordonnées données se trouvent à l'intérieur de l'objet

Définir une classe **`Randomizer`** avec une méthode statique **`getInt(...)`** pour générer les nombres aléatoires.

IMPORTANT : Ne recopiez pas le même code dans votre programme ! Profitez au mieux de l'héritage et du polymorphisme !

Les objets cibles sont gérés dans la classe **`Targets`** à l'aide d'une liste appelée **`alTargets`** :



Au démarrage, 20 objets cibles de **différents types choisis au hasard** sont ajoutés à la liste.

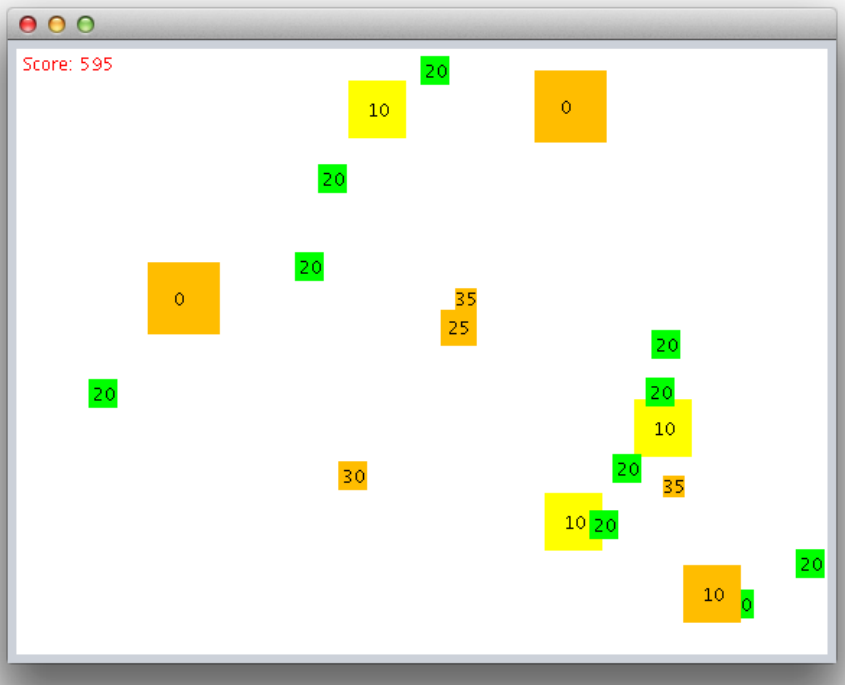
`doStep`(...) déplace toutes les cibles

`draw`(...) dessine toutes les cibles sur le canevas **`g`**

`shoot`(...) détecte si les coordonnées données comme paramètres se trouvent à l'intérieur de l'une des cibles. Si c'est le cas, le nombre de points de la cible touchée est retourné comme résultat et une nouvelle cible est créée à la place de la cible touchée. Le type de la nouvelle cible est choisi au hasard.

La classe **`DrawPanel`** possède une instance du type **`Targets`** et un **Timer** pour effectuer les mouvements (20 fois par seconde). Le score est affiché en rouge sur le canevas. Un tir est effectué lors d'un clic de la souris sur **`DrawPanel`**. Le score est affiché en conséquence.

Une instance de **`DrawPanel`** est placée sur une fiche **`MainFrame.`**



Barême : Target :30 points ; Targets&DrawPanel:30 points