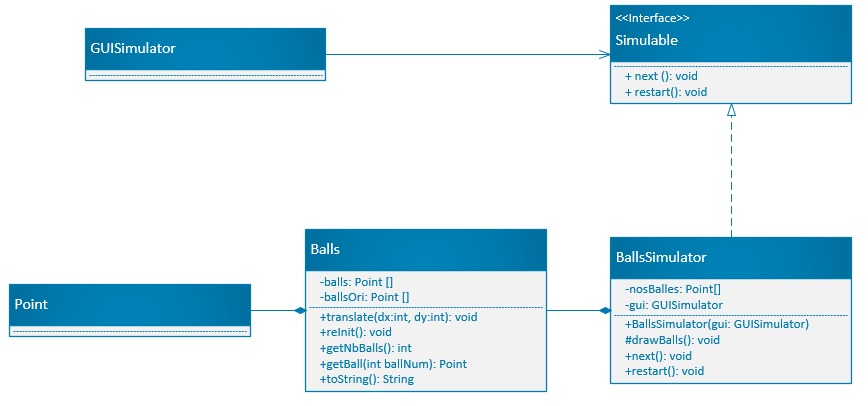
TPL POO Livrable

## Attendu : un document au format PDF de 4 pages maximum expliquant et justiﬁant vos choix de conception, l’utilisation à bon escient des classes et des méthodes les plus adaptées. Vous décrirez également les principaux tests eﬀectués et résultats obtenus.

# Les Balles



## Choix de conception

### Classe Balls

* Le constructeur initialise le premier tableau balls[] qui contient les positions des 4 A améliorer ? balles courantes a des coordonnées que nous avons fixées arbitrairement puis les copie dans ballsOri[].
* Translate utilise la méthode translate de la classe Point pour déplacer toutes les balles de dx, dy.
* reInit() réinitialise balls[] aux valeurs conservées dans ballsOri[].
* getNbBalls renvoie simplement la longueur du tableau balls ce qui évite d’ajouter une variable nbBalls dans la classe.
* getBall() permet d’accéder a une balle dans le tableau des balles courantes balls[].
* toString() renvoie un StringBuffer avec le nombre de points et leurs coordonnées.

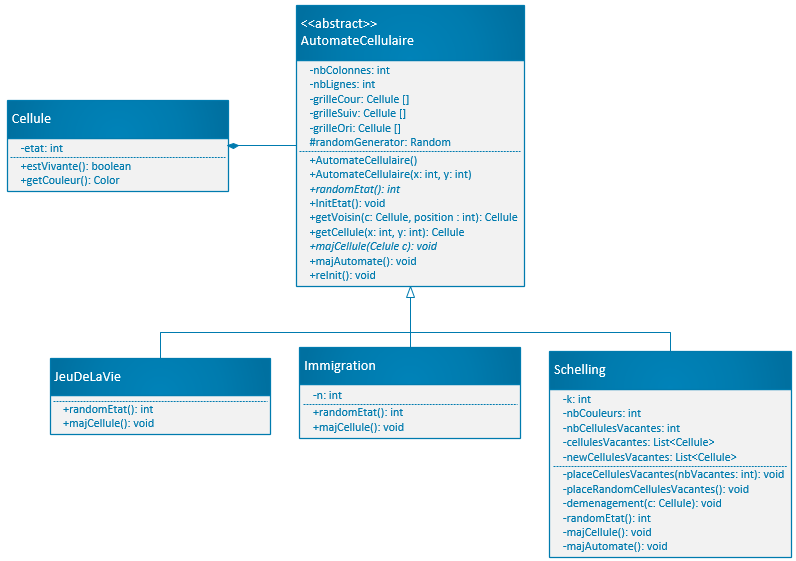
### BallsSimulator

* Le constructeur initialise le GUISimulator, les balles et les dessine grâce a la méthode drawBalls().
* Comme BallsSimulator réalise l’interface simulables elle doit redéfinir les méthodes next et restart. A compléter.

## Tests et Résultats

La classe TestBalls vérifie me bon fonctionnement de Balls sur la console. La classe TestBallsSimulator test BallsSimulator. A compléter quand fini.

# Automates Cellulaires



## Choix de conception

### AutomateCellulaire

### Cellule

La classe Cellule est utilisée indifféremment pour les trois automate. Celle-ci hérite de la classe Point à laquelle on ajoute un attribut entier représentant l’état de la cellule et les méthodes suivantes :

* getCouleur() qui renvoie selon l’état de la cellule une couleur parmi les 10 choisies
* estVivante qui renvoie une boolean true si l’état est supérieur à 1.

### abstraite AutomateCellulaire

Les automates héritent tous de la classe abstraite ‘AutomateCellulaire’. Chaque automate possède trois grilles de cellules :

* grilleCour les cellules a l’instant t ;
* grilleSuiv les cellules a l’instant t+1, qui permet de faire les calculs ;
* grilleOri qui permet de sauvegarder les cellules dans leur état d’origine.
* Elle possède deux constructeurs : celui par défaut et un constructeur avec choix du nombre de lignes et de colonnes.
* Une méthode initEtat qui initialise les états lors de la création du jeu.
* Un abstract int randomEtat qui génère un état aléatoire selon l’automate.
* getVoisin(Cellule c,int i) qui permet de récupère la cellule voisine de c en position i.
* majAutomate() qui met à jour la grille grâce à la méthode abstraite majCellule qui elle-même met à jour une cellule selon les règles de chaque jeu.

Ainsi dans chaque automate il n’y a plus qu’à définir randomEtat, majCellule, leurs constructeurs, ainsi que leurs variables propres :

#### Jeux de la vie

#### Immigration

On ajoute une variable int n = le nombre d’états

#### Schelling

on ajoute les variables int k = seuil de la simulation, int nbCouleurs, nbCellulesVacantes et les listes de cellules celluleVacantes et newCellulesVacantes ainsi que les méthodes suivantes : placeCellulesVacantes pour placer un nombre fixe de cellules vacantes sur la grille courante, placerandomCelluleVacantes pour en placer un nombre aléatoire, déménagement pour déplacer les familles.  
Remarque : il faut aussi redéfinir la méthode MajAutomate pour cet Automate car la liste des cellules vacantes doit se mettre à jour.

## Tests & Résultats

A COMPLETER

# Boids

## Choix de conception