

Rapport :

Diagramme des classes



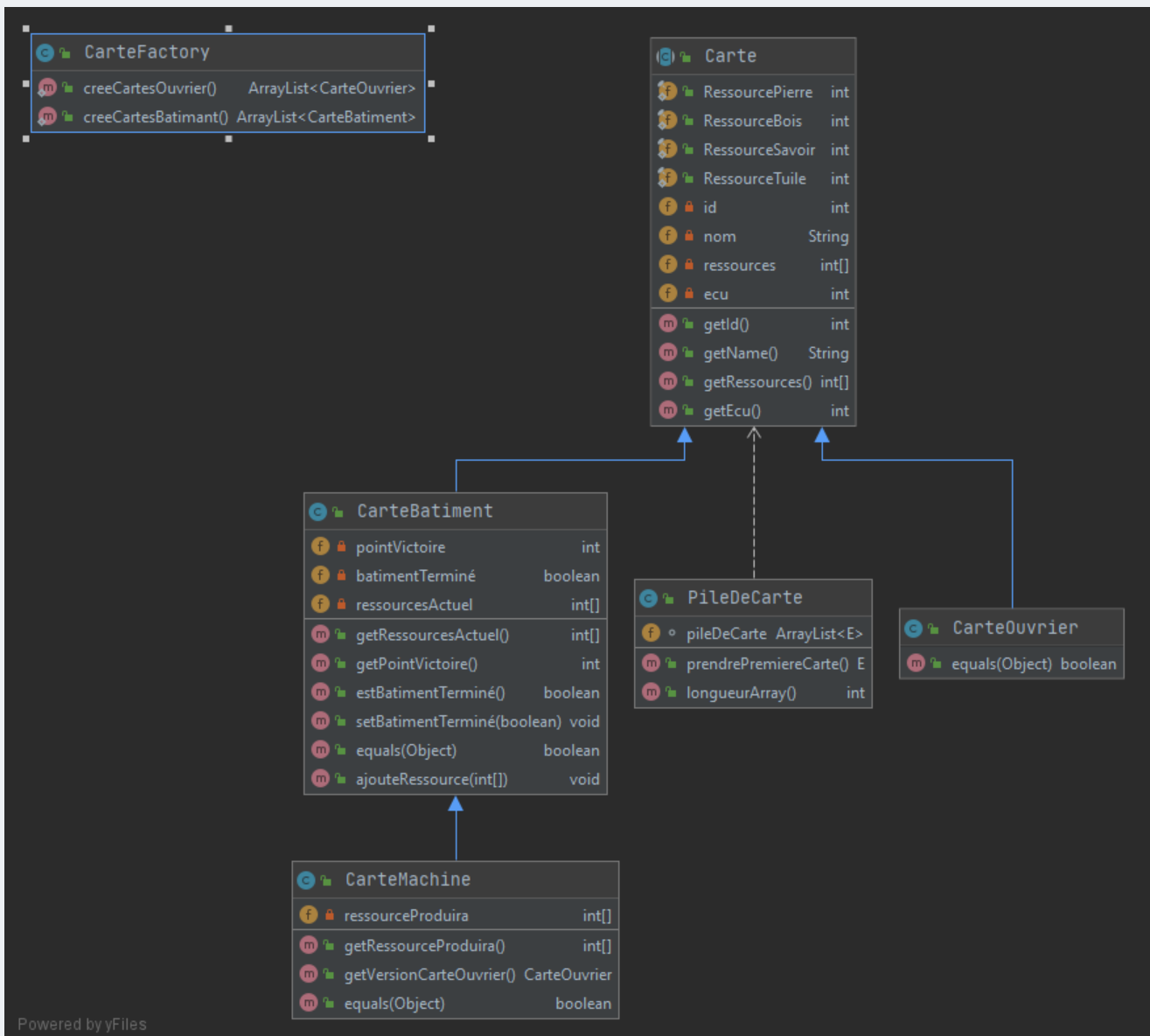
Cartes :

Chaque carte appartient à une catégorie de carte (Bâtiment, Machine, Ouvrier) et c'est catégorie se différencie par leur présentation mais surtout par leurs données et leurs fonctionnalités.

Il a donc été décidé de créer une class mère abstraite « Carte », qui ne sera pas instanciable car une carte doit obligatoirement dans une catégorie, elle sera donc instanciable via les classes filles « CarteOuvrier », « CarteBatiment » et « CarteMachine » (classe fille de Bâtiment et donc de Carte).

Afin de gérer une pile de carte (qu'elles soient une Carte Batiment, Machine ou Ouvrier), une classe générique « PileDeCarte » a été créé. Cette classe générique n'accepte seulement que les Objets de la classe « Carte » ou des classes filles.

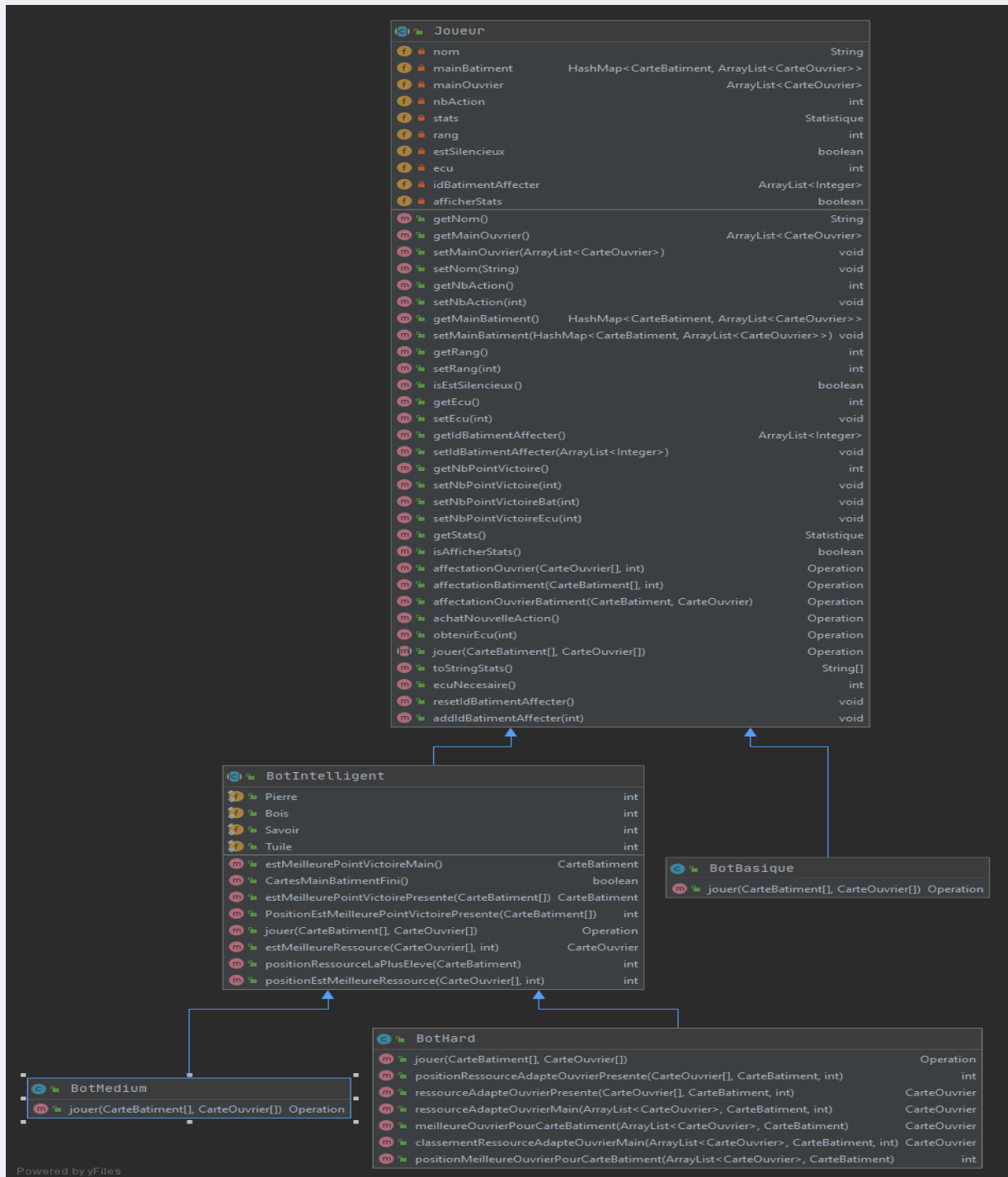
La class CarteFactory possède seulement 2 méthodes statiques qui permet de retourner une collection de cartes ouvriers et de bâtiments (qui comprennent les cartes machines) prédéfinie. Cela évite une réécriture constante de l'instanciation des cartes.



Joueur :

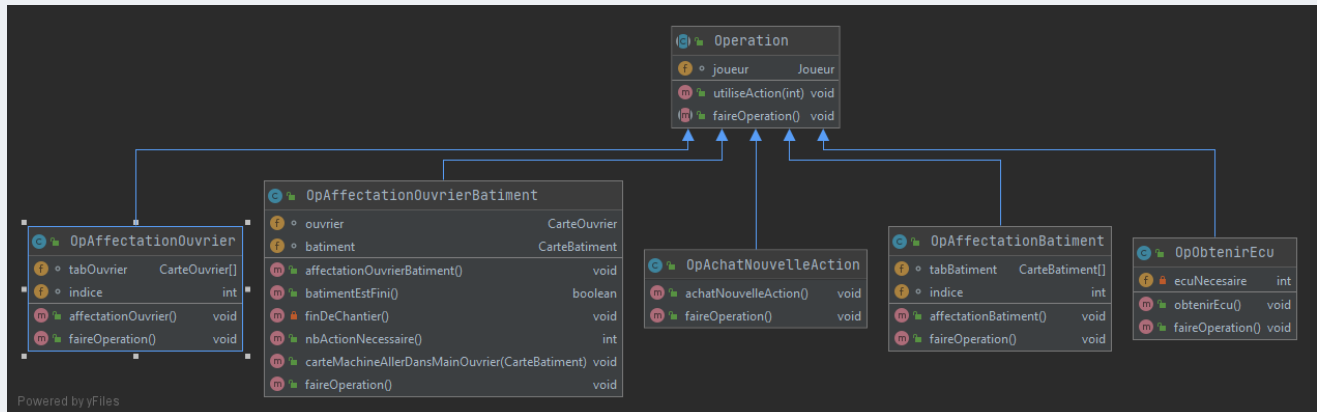
Chaque joueur aura différent niveau de stratégie, soit les même tâches mais pas les même résultats. De ce fait, il a été décidé de créer une classe abstraite « Joueur » afin que les différentes tâches soit les même mais avec des résultats. Ces tâches sont définies dans des classes filles « BotBasique », « BotIntelligent », « BotMedium » et « BotHard ».

La classe abstraite « BotIntelligent » permet de rajouter des méthodes abstraites à ces classes filles « BotMedium » et « BotHard ».



Opération :

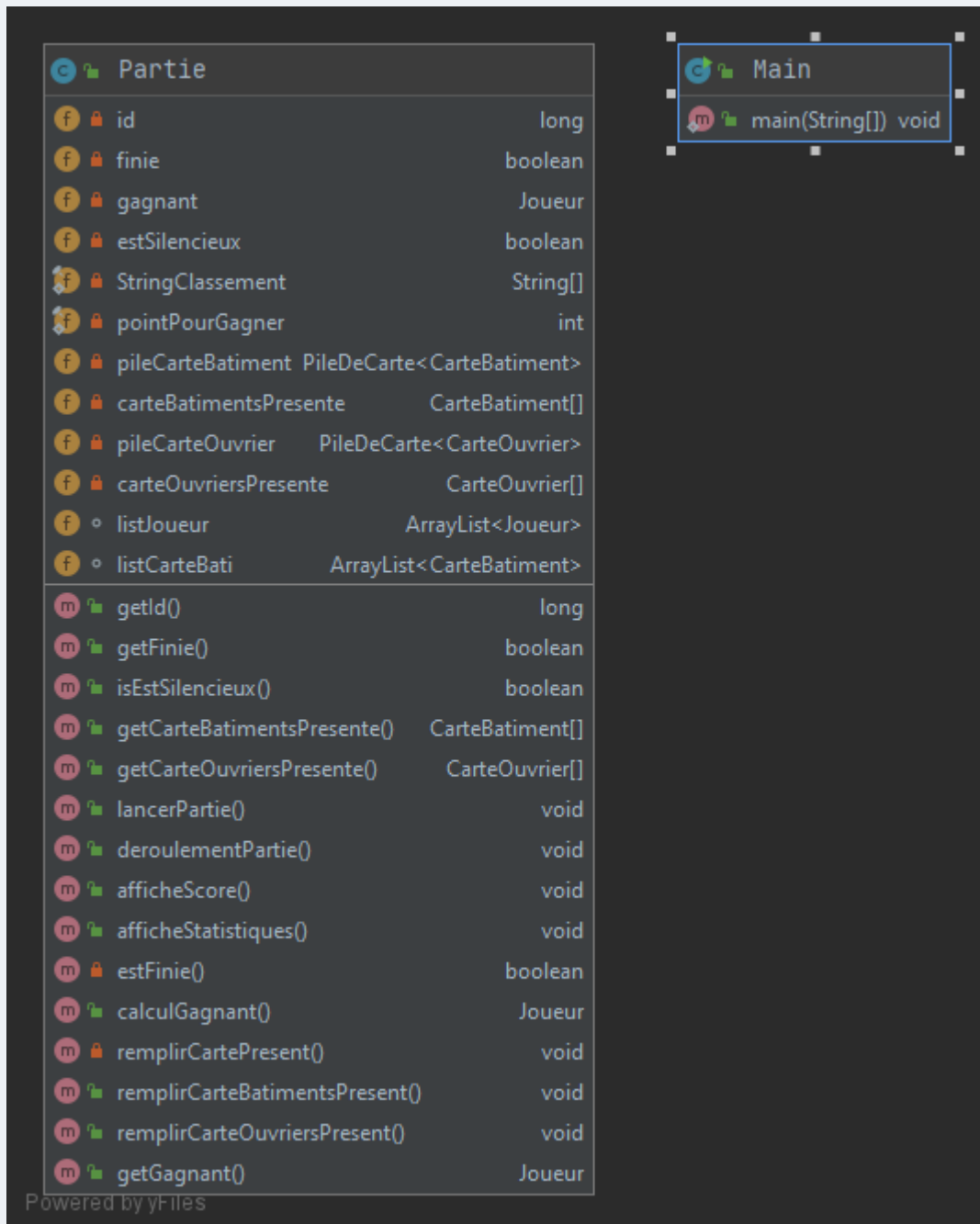
Afin de différencier les différentes opérations, une classe mère abstraite à été créée « Opération ». Cela permettra d'avoir dans chaque classes filles d'avoir la methode « faireOperation() » qui permettra d'exécuter l'opération. Ces classes filles sont « OpAffectationOuvrier », « OpAffectationBatiment », « OpObtenirEcu », « OpAffectationOuvrierBatiment » et « OpNouvelleAchat ».



Partie :

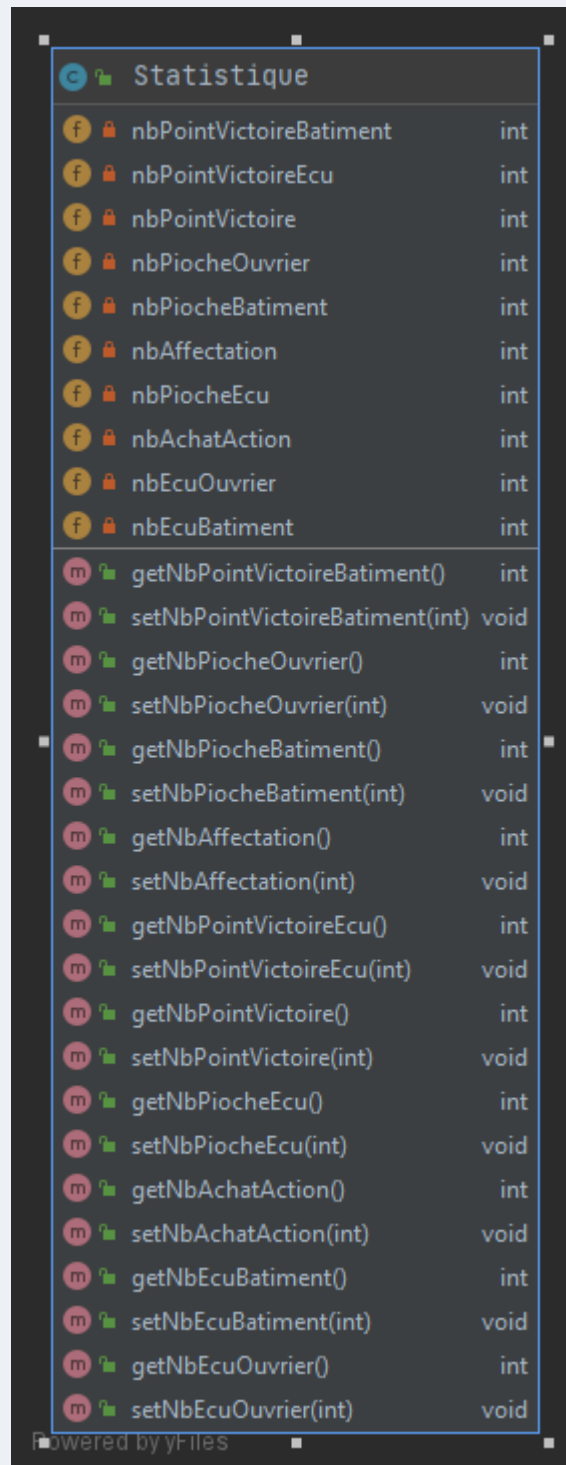
Afin de différencier les parties, une classe « Partie » a été créée. Chaque partie ne se finira pas de la même manière.

Afin d'exécuter et de tester les parties la classe « Main » possède la méthode « main ». Cela exécutera 1000 parties.



Statistique :

Afin que le client connaisse, les détails de la partie à la fin d'une partie, la classe « Statistique » permettra d'enregistrer certaine donnée. Ces données évolueront pendant la partie et se fixeront à la fin de partie.



Statistique		
f	nbPointVictoireBatiment	int
f	nbPointVictoireEcu	int
f	nbPointVictoire	int
f	nbPiocheOuvrier	int
f	nbPiocheBatiment	int
f	nbAffectation	int
f	nbPiocheEcu	int
f	nbAchatAction	int
f	nbEcuOuvrier	int
f	nbEcuBatiment	int
<hr/>		
m	getNbPointVictoireBatiment()	int
m	setNbPointVictoireBatiment(int)	void
m	getNbPiocheOuvrier()	int
m	setNbPiocheOuvrier(int)	void
m	getNbPiocheBatiment()	int
m	setNbPiocheBatiment(int)	void
m	getNbAffectation()	int
m	setNbAffectation(int)	void
m	getNbPointVictoireEcu()	int
m	setNbPointVictoireEcu(int)	void
m	getNbPointVictoire()	int
m	setNbPointVictoire(int)	void
m	getNbPiocheEcu()	int
m	setNbPiocheEcu(int)	void
m	getNbAchatAction()	int
m	setNbAchatAction(int)	void
m	getNbEcuBatiment()	int
m	setNbEcuBatiment(int)	void
m	getNbEcuOuvrier()	int
m	setNbEcuOuvrier(int)	void

Powered by yFiles