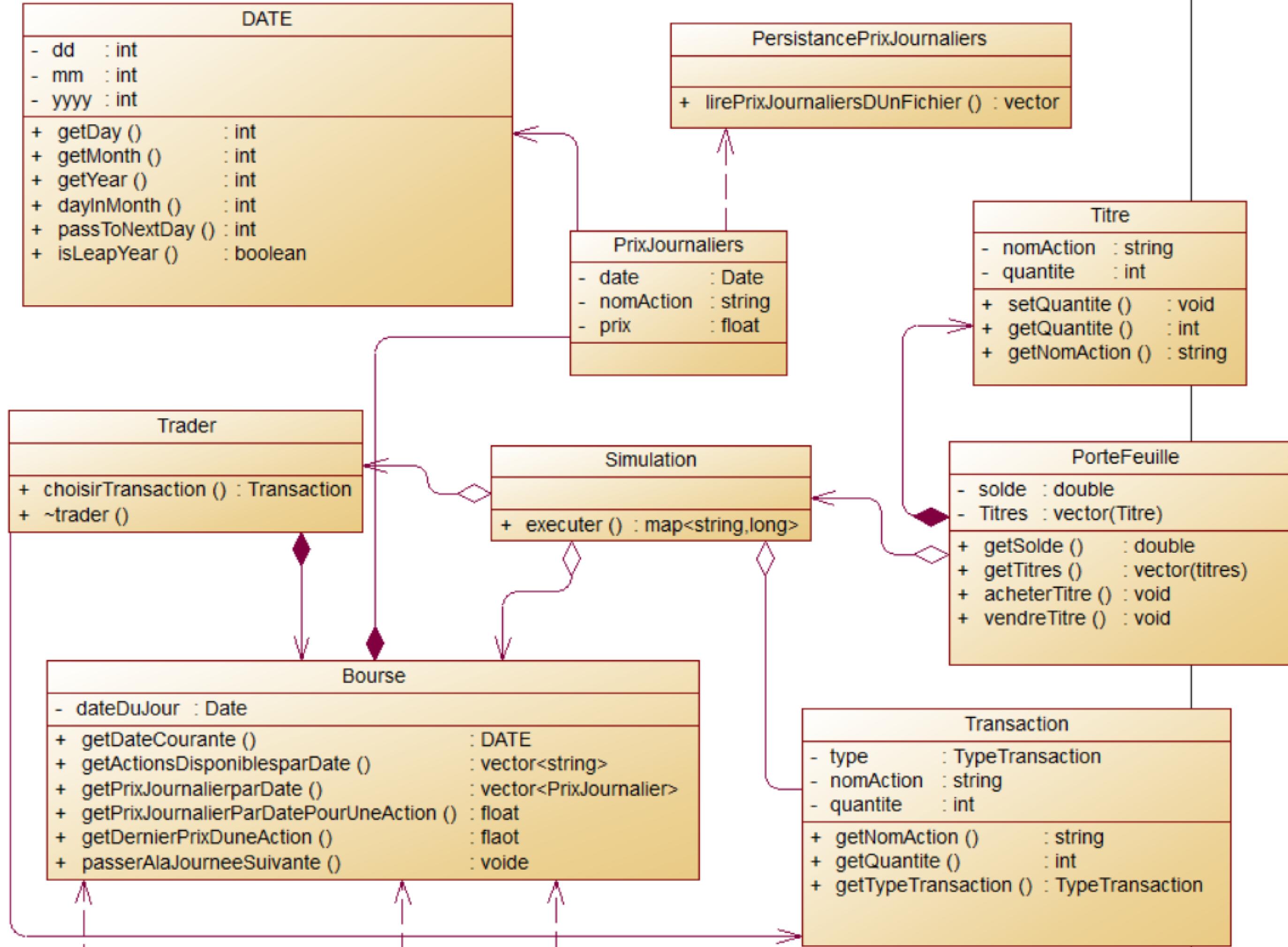
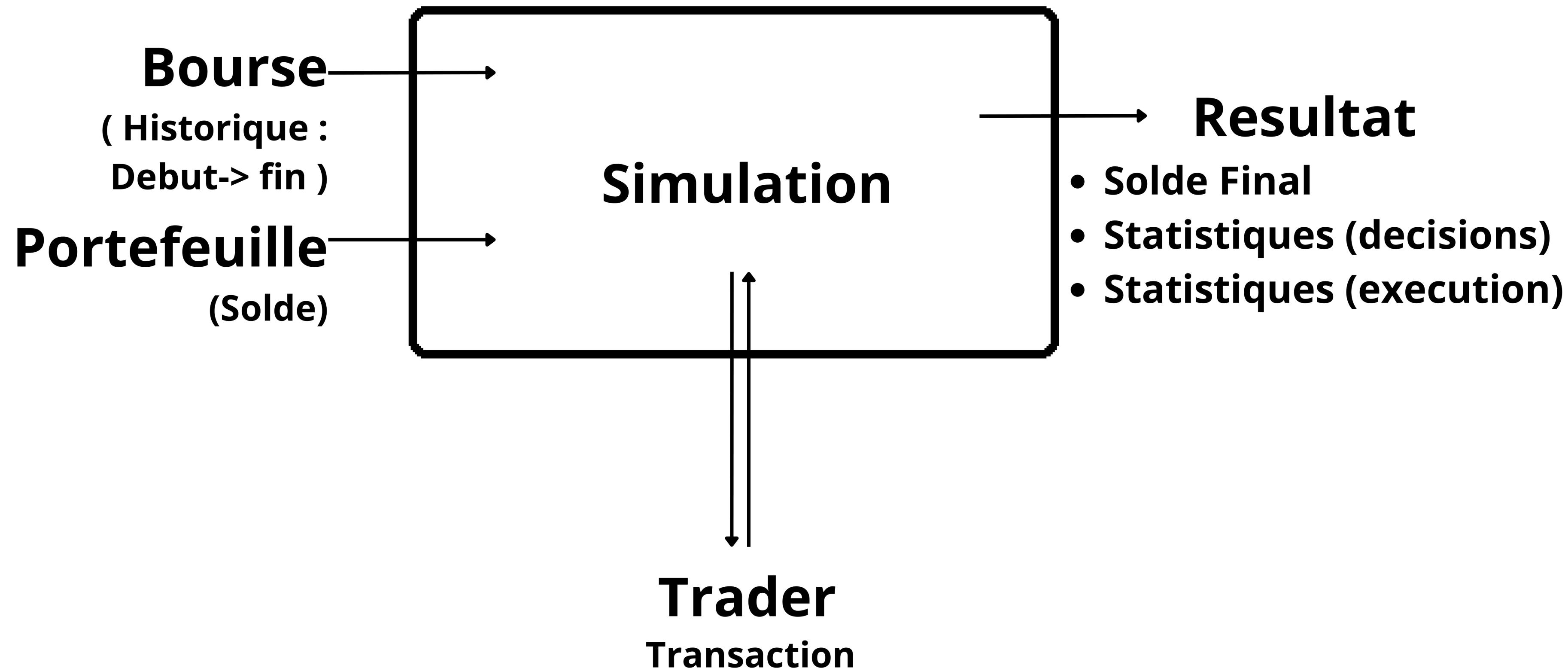


Classes :

- Date
- PrixJournalier
- PersistancePrixJournalier
- Titre
- PorteFeuille
- Transaction
- Trader → **TraderAleatoire et TraderAlgorithmique**
- Simulation
- Bourse → **BourseVector , BourseSet, BourseMultiMap**



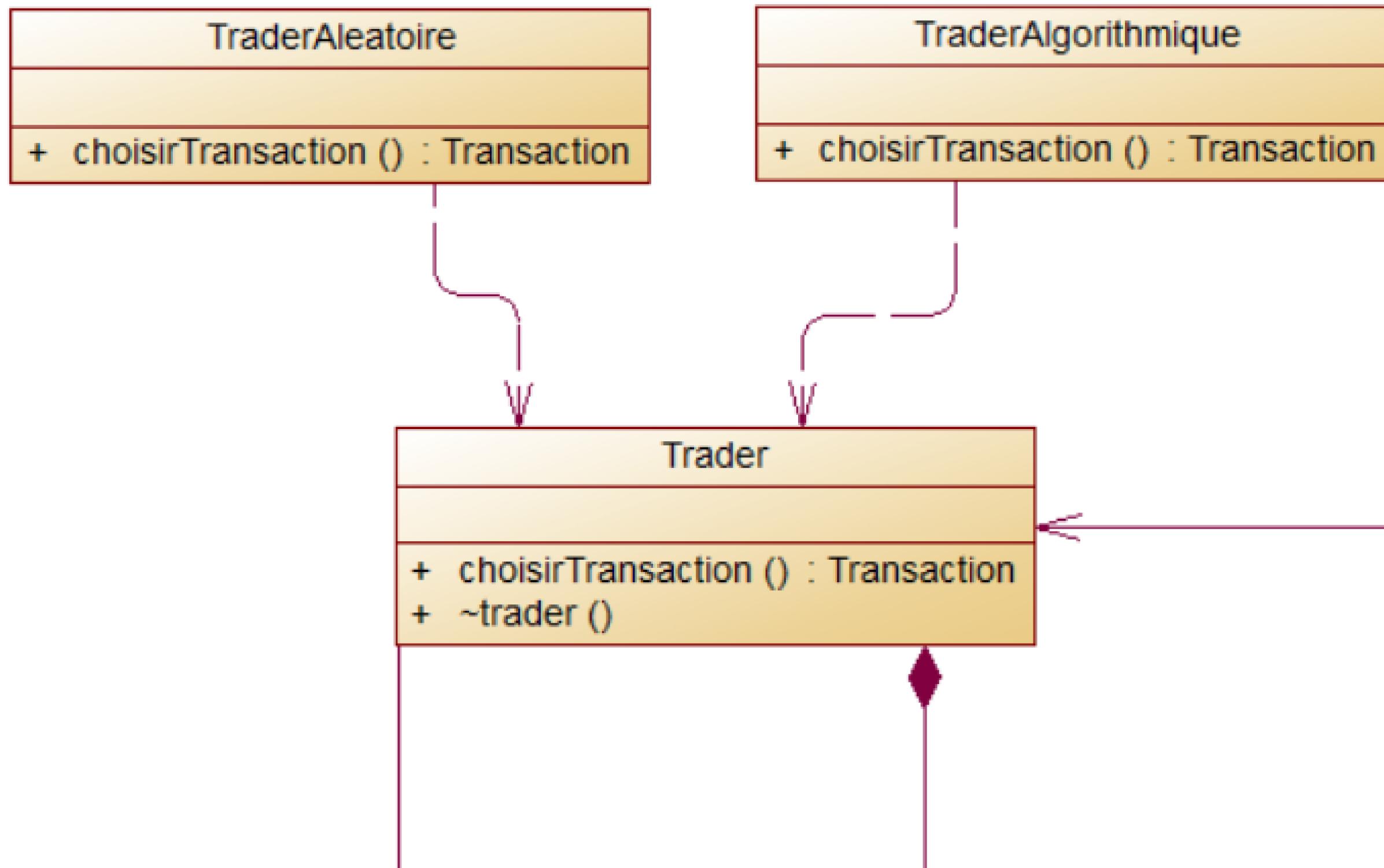


MENU

Notre Simulateur présente un menu qui permet à l'utilisateur de définir les paramètres de la simulation :

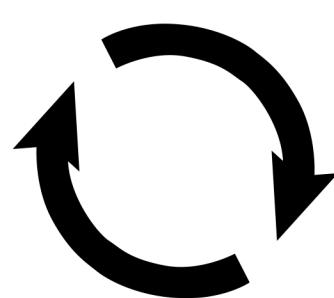
- Date Debut
- Date Fin
- Type Bourse : SET - VECTOR - MULTIMAP
- Choisir le tradeur : ALEATOIRE - ALGORITHMIQUE

Trader



Trader Aleatoire

```
TraderAleatoire
+
+ choisirTransaction () : Transaction
```



- ACHETER
- VENDRE
- RIEN

Choix Aleatoire



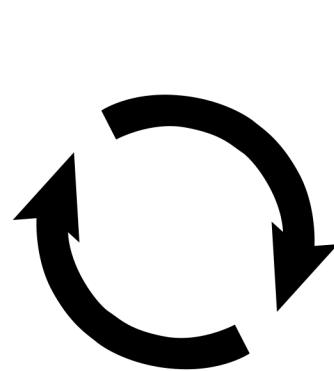
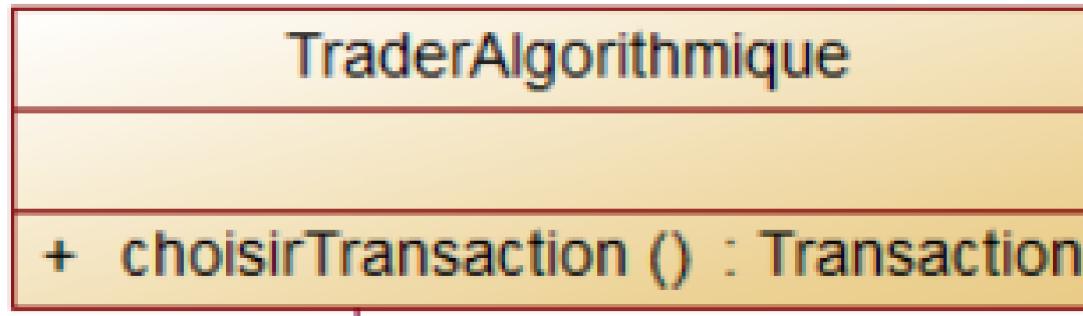
- Quantité
- Nom Action

Choix Aleatoire (Max 10)



Transaction

Trader Algorithmique



- ACHETER
- VENDRE
- RIEN

Choix Cyclique



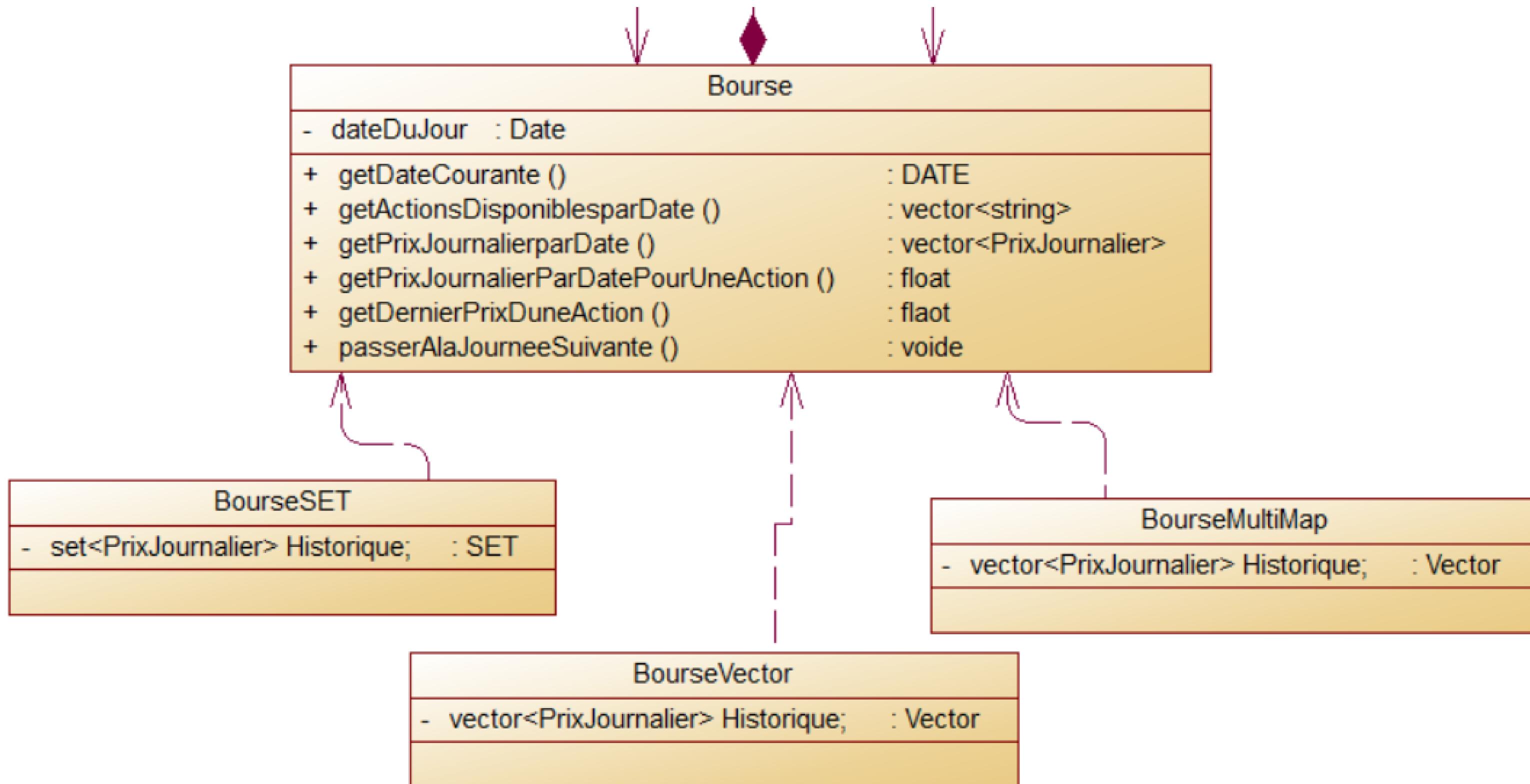
- Quantité
- Nom Action

Choix Aleatoire (Max 10)



Transaction

• Bourse



3 Implémentations : **BourseSet, BourseMultiMap et BourseVector,**

Toutes dérivées de la classe **Bourse**, Chaque implémentation utilise une structure de données différente (set, multimap et vector) pour stocker les données **historiques** de prix (PrixJournalier).

BourseVector :

- Permet un **accès direct** aux éléments en fonction de leurs indices, ce qui permet de récupérer les prix historiques en parcourant tout le vecteur.
- La **recherche** d'une action/prix pour une date donnée nécessiterait de **parcourir le vecteur**, ce qui entraînerait une complexité temporelle de **O(n)**
- La modification des prix historiques peut avoir une complexité temporelle de **O(1)** pour l'ajout à la fin, **O(N)** pour l'insertion ou la suppression au milieu et **O(N)** pour la mise à jour.

BourseSet :

- La structure de données set garantit que les prix historiques sont automatiquement triés par valeur, ce qui permet de récupérer efficacement les actions et les prix dans un ordre trié.
- La recherche d'une action ou d'un prix spécifique pour une date donnée peut être effectuée avec une complexité temporelle de $O(\log N+M)$, où N est le nombre de prix historiques et M est le nombre de prix pour la date donnée.
- Cependant, la modification des prix historiques (ajout, suppression ou mise à jour) peut avoir une complexité temporelle de $O(\log N)$, ce qui peut être plus lent par rapport aux autres implémentations.

BourseMultiMap:

- Stocker plusieurs prix pour la même date, ce qui la rend adaptée aux scénarios où plusieurs prix peuvent être enregistrés par jour.
- Les prix sont stockés dans **l'ordre d'insertion**, ils ne sont donc pas automatiquement triés par valeur.
- La recherche d'une action ou d'un prix spécifique pour une date donnée peut avoir une complexité temporelle de $O(\log N + M)$, où N est le nombre de prix historiques et M est le nombre de prix pour la date donnée.
- La modification des prix historiques a une complexité temporelle de $O(\log N)$, similaire à **BourseSet**.

Statistiques

Après l'execution de la simulation, le programme affiche les statistiques :

NBRE_GET_ACTIONS_DISPO_AUJ : Nombre d'appels de la fonction

NBRE_GET_PRIX_ACTION_DATE : Nombre d'appels de la fonction

NBRE_TX : Nombre de Transactions Total

NBRE_TX_ACHAT : Nombre de Transactions Achat

NBRE_TX_VENTE : Nombre de Transactions VENTE

SOLDE_DEPART : Solde Initial

SOLDE_FIN : Solde Final

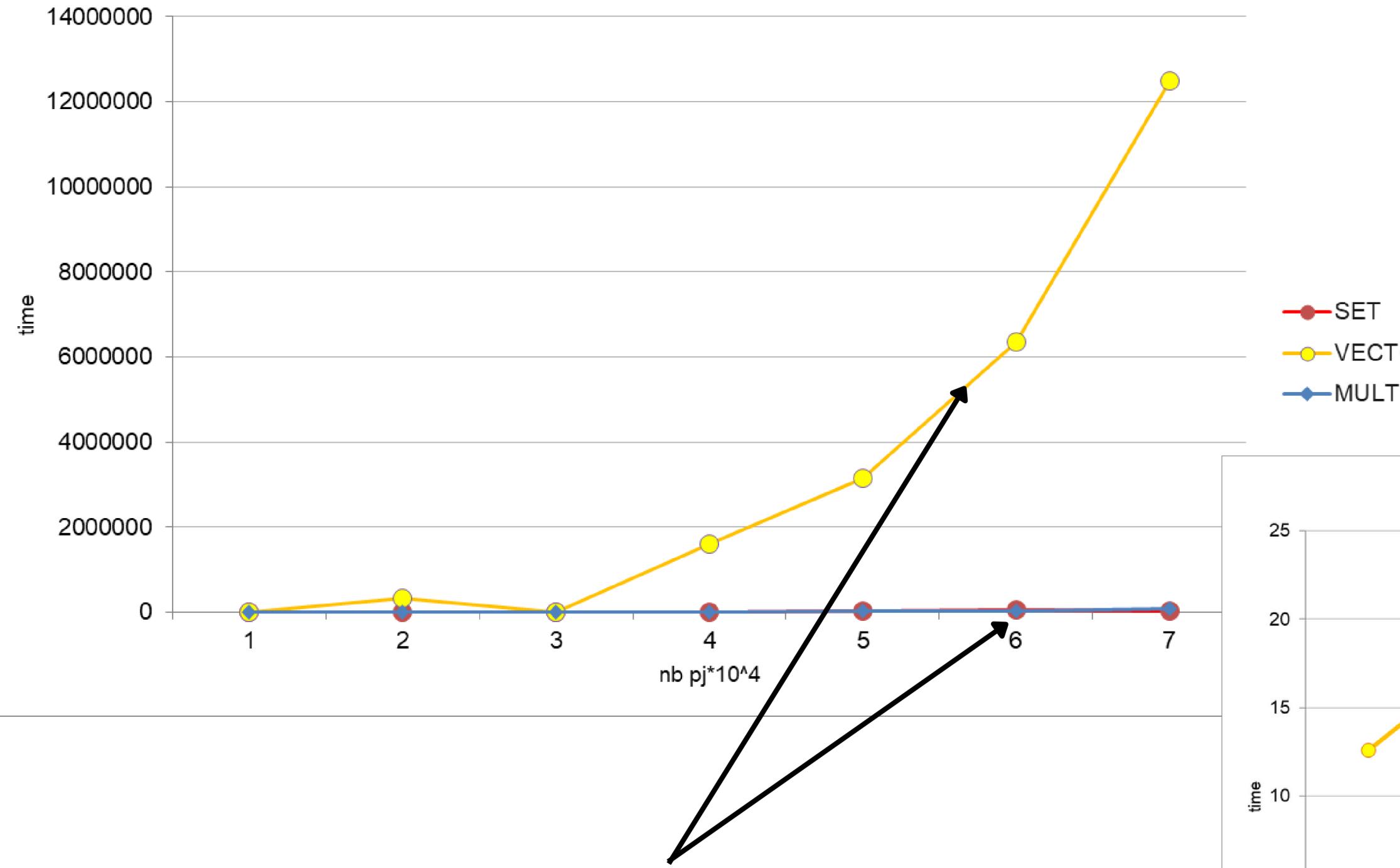
GAIN : Montant Gagné (ou perdu)

TEMPS_GET_ACTIONS_DISPO_AUJ_µs : Temps de réponse

TEMPS_GET_PRIX_ACTION_DATE_µs : Temps de réponse

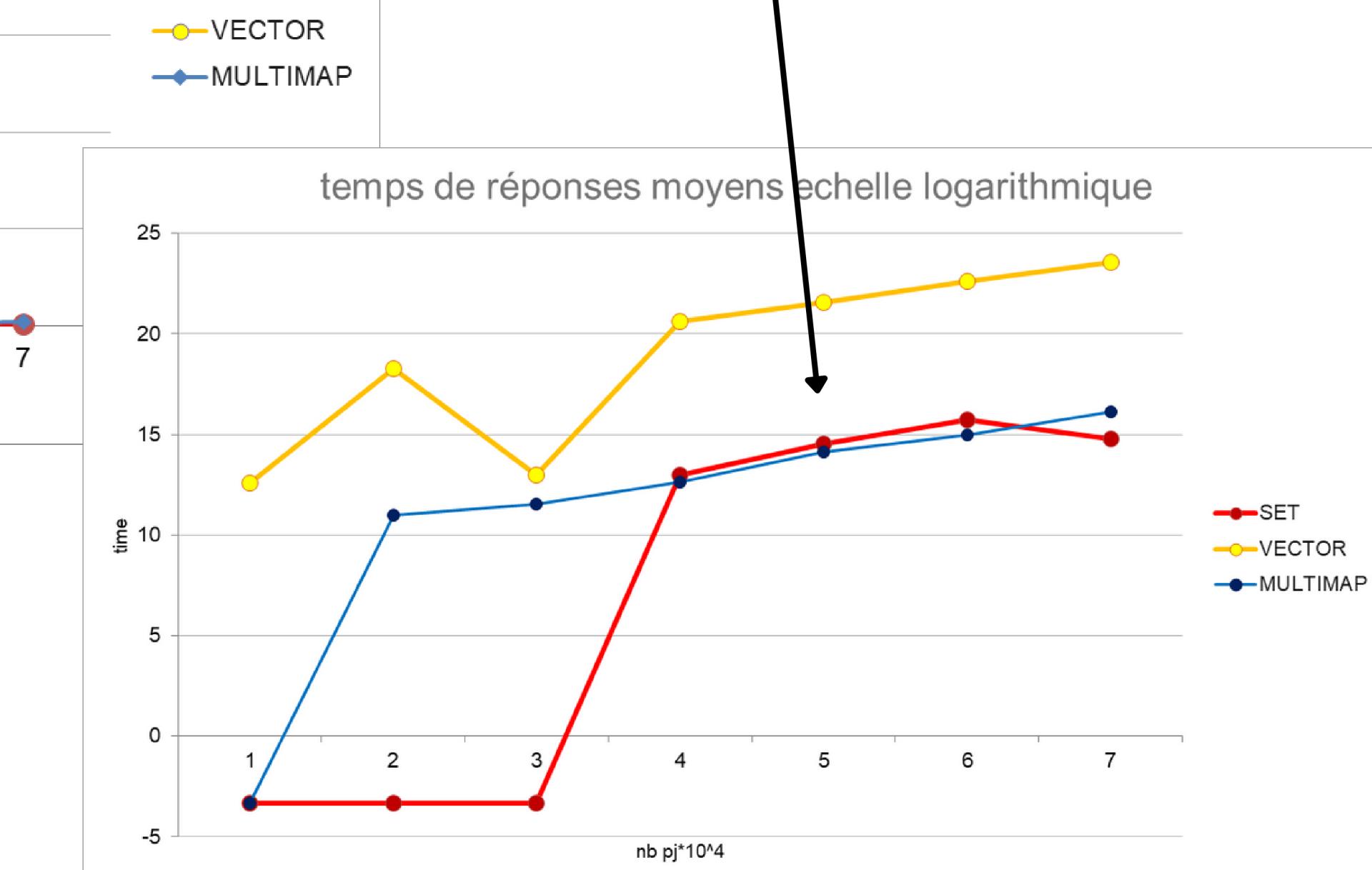
TEMPS_TX_µs : Temps de réponse

temps de réponses moyens (μ s) echelle normale
GET ACTION

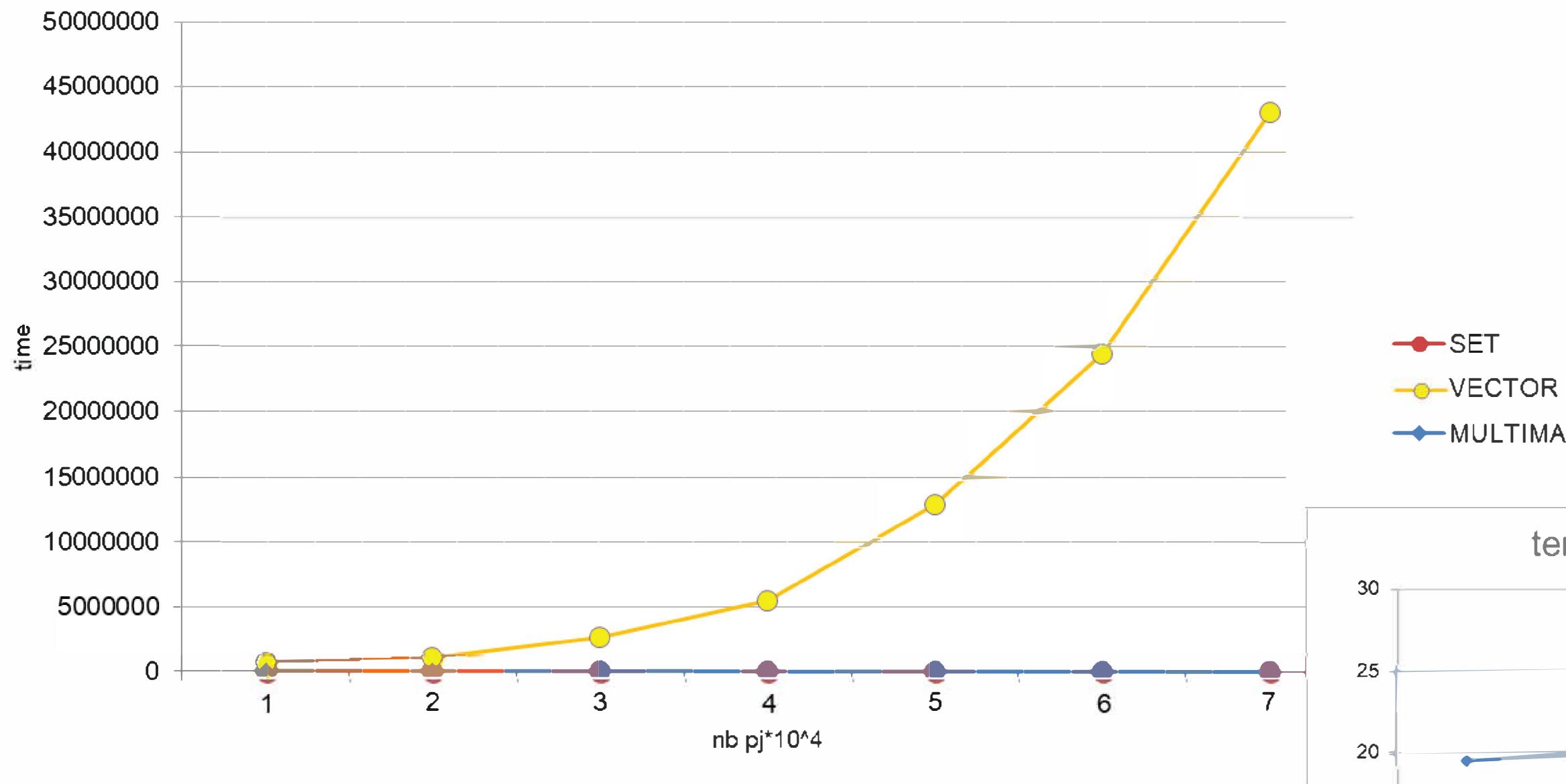


Claire amélioration en utilisant Set et Multimap

Resultats :
Methode getAction
Set et Multimap sont Comparables ($pj > 4 \times 10^4$)



temps de réponses moyens (μ s) echelle normale
GET PRIX ACTION



Results :
Methode getPrixAction
Grande difference

